Queueing Networks and Markov Chains Analysis with the Octave queueing package

Moreno Marzolla
Dipartimento di Informatica–Scienza e Ingegneria (DISI), Università di Bologna
Mura Anteo Zamboni 7, I-40127 Bologna, Italy
moreno.marzolla@unibo.it

ABSTRACT

Queueing networks and Markov chains are a widely used modeling notation that has been successfully applied to many kind of systems. In this paper we describe the `queueing` package, a free software package for product-form queueing networks and Markov chains analysis written in GNU Octave. The `queueing` package allows users to compute performance measures of Markov chains, single-station queueing systems and product- and some non product-form Queueing Network (QN) models. We present some practical examples showing how the `queueing` package can be used for reliability analysis, capacity planning and general systems modeling.

Keywords

Queueing Networks, Markov Chains, Mean Value Analysis

1. INTRODUCTION

QNs and Markov chains are widely used for capacity planning, bottleneck analysis and performance evaluation of many kinds of systems. QN analysis usually involves the computation of steady-state performance measures, such as throughput X_k, mean queue length Q_k, mean response time R_k and utilization U_k of each server $k = 1, \ldots, K$. QN models can be evaluated either by simulation, or using analytical and numerical techniques. Some classes of QN models enjoy product-form solution [3], meaning that steady-state performance measures can be computed efficiently. Therefore, product-form QNs are attractive when evaluating the same model with different parameters ("what-if" analysis), or in all situations where models must be evaluated at run-time.

Despite the vast literature on numerical solution techniques for QN models (see [5] and references therein), only a few tools are currently available and actively maintained [4, 7,9,14,23]. To further contribute to this area, we present the `queueing` package, a software package for QN and Markov chains analysis written in GNU Octave. GNU Octave is an interpreted language for numerical computations that is freely available for all major Operating Systems [13]. The `queueing` package provides implementations of numerical algorithms for transient and stationary analysis of discrete and continuous Markov chains. single-station queueing systems, and stationary analysis of some classes of product-form QNs.

This paper is structured as follows. In Section 2 we illustrate the main design principles behind `queueing`. Section 3 gives an overview of the most important functions provided by the package. Section 4 describes some practical usage examples in the area of reliability analysis (Section 4.1), bound analysis of QNs (Section 4.2) and multiclass QN analysis (Section 4.3). Concluding remarks are given in Section 5.

2. DESIGN PRINCIPLES

The `queueing` package is a collection of functions running within the GNU Octave interpreter. The Octave environment provides the glue which allows complex models to be built and evaluated programmatically. Therefore, the user needs to "program" the model using the GNU Octave language (there is no graphical user interface). This can be useful, e.g., to do parametric evaluation or to perform ad-hoc analysis. However, this flexibility imposes a steep learning curve. Plans are underway to make the `queueing` package more accessible through a GUI.

The `queueing` package has been used in the following areas: (*i*) Incremental model development: `queueing` and GNU Octave can be used for rapid prototyping and iterative refinement of QN models. (*ii*) Modeling environment: large and complex performance studies can be done quickly, since models involving repetitive or embedded structure can be easily defined. (*iii*) Queueing Network research: new algorithms can be programmed and tested against existing ones. The Octave language is well suited for implementing numerical algorithms which operate on arrays or matrices, of which many QN algorithms are an instance of. (*iv*) Reference implementations: as observed in [10], some large research communities (e.g., linear algebra and parallel computing) have a long history of sharing implementations of standard algorithms. The `queueing` package aims at providing reference implementations of core QN algorithms. (*v*) Teaching: the package is being used in some Universities to teach performance modeling courses. Since the package implements many textbook QN algorithms, students can immediately put those algorithms to work to solve practical problems, encouraging "learning by doing".

Special care has been put to make `queueing` suitable for general use. Functions are thoroughly documented and usage examples are provided for most of the functions. Documentation and demo blocks can be accessed at the Octave prompt using the `help()` and `demo()` built-in commands, e.g, `help(ctmc)` prints the documentation and `demo(ctmc)` displays and executes all demo blocks for the `ctmc()` function. A complete, printable user manual is included with

Name	Description
`dtmc()`	Stationary/Transient state occupancy probabilities
`dtmcbd()`	Birth-death process
`dtmcexps()`	Mean number of visits
`dtmcfpt()`	First passage times
`dtmcmtta()`	Mean time to absorption

Table 1: Some functions for discrete time Markov chains analysis

the package.

One important aspect of any software, especially numerical programs, is its correctness. The `queueing` package includes unit tests embedded as specially-formatted comments within the source code. Tests are used to check the results against reference values for models described in the literature, or against values computed by other tools. When reference results are not available, cross-validation has been employed by evaluating the same model with different functions. For example, a closed QN might be analyzed using Mean Value Analysis (MVA) or the convolution algorithm; both should produce compatible results, up to numerical inaccuracies.

Given the large number of algorithms for analyzing queueing networks that have been proposed in the literature, it is infeasible – and probably not even desirable – to include them all in the `queueing` package. Indeed, many numerical algorithms are of limited practical use, being concerned with very specific types of networks. Therefore, a key design decision has been to focus on implementations of the MVA and convolution algorithms for (a subset of) BCMP networks [3]; of course, contributions of additional algorithms from users are always welcome, some of which have already been incorporated.

3. PACKAGE CONTENT

In this section we illustrate some of the functions contained in `queueing` grouped by area: Markov chain analysis, single station queueing systems and queueing networks.

3.1 Markov chains

A finite, discrete-time Markov chain is a set of N states with an $N \times N$ transition probability matrix $\mathbf{P}$ such that $P_{i,j}$ is the transition probability from state i to state j. A continuous-time Markov chain can be represented by a stochastic matrix $\mathbf{Q}$, where $Q_{i,j}$ is the transition rate from state i to state $j \neq i$.

Table 1 lists some of the functions provided by `queueing` to compute some performance measures on discrete Markov chains. Corresponding functions for continuous chains are available as well, substituting the prefix `dtmc` with `ctmc` (e.g., `ctmcexps()` computes the mean sojourn time on a continuous-time Markov chain).

Let $\pi_i(t)$ be the probability that the system is in state $i \in \{1, \ldots, N\}$ at time t. Vector $\boldsymbol{\pi}(t) = (\pi_1(t), \ldots, \pi_N(t))$ denotes the state occupancy probabilities at time t.

Under certain conditions [5] a Markov chain has a stationary distribution $\boldsymbol{\pi}$ which is independent from the initial occupancy probability $\boldsymbol{\pi}(0)$. The stationary distribution can be computed with `p = dtmc(P)` (for discrete chains) or `q = ctmc(Q)` (for continuous chains). The same function can be invoked with three parameters, e.g., `pn = dtmc(P,n,p0)` to compute the state occupancy probability vector `pn` at step `n` given the initial probabilities `p0` at step 0. Function `ctmc()` can similarly be used for continuous chains.

The *mean time to absorption* is the average number of steps (time, in the continuous case) it takes to reach an absorbing state, given the initial occupancy probability vector $\boldsymbol{\pi}(0)$. A state i is *absorbing* if it has no outgoing transitions. The *first passage time* $M_{i,j}$ is defined as the average number of transitions (time, in case of continuous chains) before state j is visited for the first time, starting from state i. Finally, the *mean number of visits* $L_{n,j}$ is the average number of visits to state j during the first n transitions; the equivalent metric for continuous Markov chains is called *expected sojourn time*. These parameters are useful for reliability analysis, as will be demonstrated in Section 4.1.

3.2 Single station queueing systems

The `queueing` package provides functions for analyzing several types of single station queueing systems [5, 18]: $M/M/m$, $M/M/m/k$, $M/M/\infty$, asymmetric $M/M/m$ (this system contains m service centers with possibly different service rates), $M/G/1$ (general service time distribution) and $M/H_m/1$ (Hyperexponential service time distribution). For each kind of system, the following steady-state performance measures can be computed: utilization U, mean response time R, average number of requests in the system Q and throughput X.

3.3 Queueing Networks

Table 2 lists some of the functions for QN analysis provided by the `queueing` package, which can be roughly grouped in three classes: algorithms for bound analysis, product-form and non product-form QNs.

Model Specification. A general multiclass network is specified in terms of the following parameters: $\lambda_{c,i}$ (open networks only) external arrival rate of class c requests to service center i; N_c (closed networks only) total number of class c requests in the system: Z_c (closed networks only) external delay ("think time") experienced by class c requests; $S_{c.i}$ mean service time of class c requests at center i; $P_{r,i,s,j}$ probability that a class r request that completes service at center i is routed to class j as a class s request; $V_{c,i}$ mean number of visits of class c requests to center i. The same parameters are used for single-class models, by dropping dimensions associated with classes.

Bound Analysis. Bound analysis allows efficient computation of upper and/or lower limits on the system throughput X and response time R, from which bounds on throughput and response times of individual servers can be derived. Performance bounds are useful where accuracy can be sacrificed in favor of speed, e.g., when doing on-line performance tuning of running systems. The `queueing` package provides algorithms for computing several classes of bounds: Asymptotic Bounds (AB) [12] for open and closed networks, Balanced System Bounds (BSB) [24] for open and closed networks, and Geometric Bounds (GB) [11] for closed networks; single- and multiclass networks are supported. The Composite Bounds (CB) [17] technique for

Function Name	Description
`qnosaba()`	Asymptotic Bounds for open networks [12]
`qncsaba()`	Asymptotic Bounds for closed Networks [12]
`qnosbsb()`	Balanced System Bounds for open networks [24]
`qncsbsb()`	Balanced System Bounds for closed networks [24]
`qncsgb()`	Geometric Bounds for closed networks [11]
`qnopensingle()`	Analysis of open Jackson networks [16]
`qnopenmulti()`	Analysis of open, multiclass product form networks
`qncsconv()`	Convolution algorithm for closed, single class QNs with fixed-rate servers [6]
`qncsconvld()`	Convolution algorithm for closed, single class QNs with general load dependent servers
`qncsmva()`	MVA for closed, single class networks with fixed-rate and multiple server nodes [20]
`qncsmvald()`	MVA for closed, single class networks with general load dependent servers
`qncscmva()`	Conditional MVA for closed, single class networks with a load dependent server [8]
`qncmmva()`	MVA for closed, multiclass networks with fixed-rate and multiple server nodes [20, 22]
`qncmmvabs()`	Approximate MVA for closed, multiclass networks with fixed-rate servers using Schweitzer's approximation [21]
`qnmix()`	MVA for mixed networks with fixed-rate servers [22]
`qncsmvablo()`	Approximate MVA for closed, single class networks with blocking [1]
`qnmarkov()`	Exact analysis of closed, single class networks with blocking by direct solution of the underlying Markov chain

Table 2: Some functions for QN analysis

closed multiclass networks is available as well.

Product-Form QNs. The `queueing` package allows the computation of exact and approximate steady-state performance measures of open and closed product-form networks. Networks can have a single class of requests, or multiple, independent request classes. Open networks are handled by the `qnos()` (single customer class) and `qnom()` (multiple customer classes) functions. For single-class closed networks, the MVA [20] and convolution [6] algorithms are implemented by the `qncsmva()` and `qncsconv()` functions, respectively. Both support First-Come First-Served (FCFS), Last-Came First-Served, Preemptive Resume (LCFS-PR), Processor Sharing (PS) and Infinite Server (IS) nodes; single and multiple server FCFS nodes are handled. For efficiency reasons, and to make the code more readable, the convolution and MVA algorithms for single class networks with general load-dependent service times are implemented in separate function `qncsconvld()` and `qncsmvald()`, respectively.

Product-form closed networks with multiple classes of requests can be analyzed using `qncmmva()`. Class switching is supported: requests are allowed to change class after completing service. Due to its computational cost, multiclass MVA is appropriate for networks with small population and limited number of chains. For larger networks, approximations based on the MVA have been proposed in the literature; function `qncmmvabs()` implements the approximation scheme by Bard and Schweitzer [2, 19, 21]. Finally, mixed multiclass Product-form Queueing Network (PFQN) [3] are handled by the `qnmix()` function. In mixed networks, both open and closed classes can be present at the same time.

Non product-form QNs. The `queueing` package includes a few algorithms for evaluating closed single class networks with blocking. In blocking networks, queues have a maximum capacity: a request joining a full queue will block until a slot in the destination node becomes available. Function `qncsmvablo()` implements the MVABLO algorithm [1] which is based on an extension of MVA. MVABLO computes approximate stationary performance measures for closed, single class networks with Blocking After Service (BAS) blocking. According to the BAS discipline, a request joining a full queue blocks the source until a slot is available at the destination.

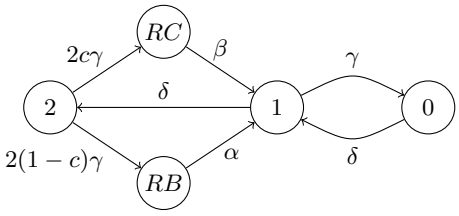

Figure 1: Reliability Model

4. EXAMPLES

In this section we give three practical usage examples of `queueing`, for reliability analysis using Markov chains, capacity planning using bound analysis, and multiclass QNs analysis.

4.1 Reliability Analysis with Markov chains

Figure 1 shows a reliability model of a multiprocessor system described in [15]. There are $N = 2$ processors, each subject to failures with Mean Time To Failure (MTTF) $1/\gamma$. States $n \in \{0, 1, 2\}$ denote that there are n working processors. If one processor fails, it can be recovered (state RC) with probability c; recovery takes time $1/\beta$. When recovery is not possible, a reboot is required (state RB), which brings down the whole system for time $1/\alpha > 1/\beta$. The mean time to repair a failed processor is $1/\delta$.

If we enumerate the states as $\{2, RC, RB, 1, 0\}$, we can define the model and compute the steady state occupancy probability vector using the following Octave code (we use

`a, b, g, d` instead of $\alpha, \beta, \gamma, \delta$; values are from [15]):

```
mm = 60; hh = 60*mm; dd = 24*hh; yy = 365*dd;
a = 1/(10*mm);      # duration of reboot (10 min)
b = 1/30;           # reconfiguration time (30 sec)
g = 1/(5000*hh);    # processor MTTF (5000 h)
d = 1/(4*hh);       # processor MTTR (4 h)
c = 0.9;            # recovery probability
[TWO,RC,RB,ONE,ZERO] = deal(1,2,3,4,5); # states
#        2     RC       RB          1       0
Q = [ -2*g   2*c*g 2*(1-c)*g     0       0;    # 2
        0     -b        0        b       0;    # RC
        0      0       -a        a       0;    # RB
        d      0        0     -(g+d)     g;    # 1
        0      0        0        d     -d];   # 0
p = ctmc(Q);
```

which gives `p` $= (9.9839 \times 10^{-1}, 2.9952 \times 10^{-6}, 6.6559 \times 10^{-6}, 1.5974 \times 10^{-3}, 1.2779 \times 10^{-6})$. From these values we can compute other availability metrics, e.g., the average time spent over a year in states RC, RB and 0:

```
p(RC)*yy/mm    # minutes/year spent in RC
# => 1.5743
p(RB)*yy/mm    # minutes/year spent in RB
# => 3.4984
p(ZERO)*yy/mm # minutes/year spent in 0
# => 0.67169
```

that is, over a year the system is unavailable for about 1.57 minutes due to reconfigurations, 3.50 minutes due to reboots and 0.67 minutes due to failure of both processors.

We can compute the Mean Time Between Failures (MTBF) of the whole system, defined as the average duration of continuous system operation. We assume that the system starts in state 2, and we consider the system operational also when in the reconfiguration state. Therefore, the set of operational states is $\{2, 1, RC\}$. If we make states 0 and RB absorbing by removing all their outgoing transitions, the MTBF is the mean time to absorption of the (modified) Markov chain. The `ctmcmtta()` function can be used to compute the mean time to absorption:

```
Q(ZERO,:) = Q(RB,:) = 0;  # make {0, RB} absorbing
p0 = [1 0 0 0 0];         # initial occupancy prob.
MTBF = ctmcmtta(Q, p0)/yy # MTBF (years)
# => 2.8376
```

from which we get a MTBF of 2.84 years.

4.2 Bound Analysis

Let us consider a simple model of a scientific computing cluster, where a closed population of N jobs process data stored in a tape library. A disk-based cache is used to reduce the access of the (slow) tape library. A job reads the data it needs from disk; in case of a cache miss, which happens with probability $1 - p$, the data is copied from tape to disk before the job is allowed to proceed.

The system can be modeled by the closed network shown in Figure 2. There are two FCFS servers representing the tape library and disk cache, respectively. A delay center (IS node) represents the pool of CPUs; we assume that there are enough CPUs so that jobs are always executed as soon as they are ready. We denote with Z the mean duration of each job, with S_1 the mean tape transfer time and with S_2 the mean disk transfer time.

Suppose that budget constraints allows us to choose between two different system configurations. Configuration (a)

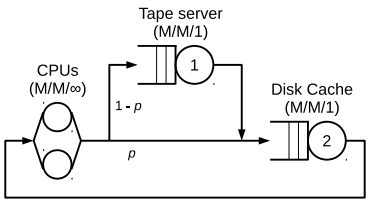

Figure 2: Queueing model of Tape Farm

	p	S_1	S_2	Z
Conf. (a)	0.9	$300s$	$40s$	$1800s$
Conf. (b)	0.75	$300s$	$30s$	$1800s$

Table 3: Parameters for the model in Figure 2

uses a large cache of inexpensive disks; this means that the cache hit ratio p is higher, but disk transfer times are larger because disks are slower. Configuration (b) uses a smaller cache of fast disks; in this scenario, the cache hit ratio is smaller, but disk transfer times are lower. The parameters for both scenarios are reported in Table 3.

We can perform a bound analysis to understand which configuration provides the better throughput when a large number of jobs is present. This can be done with the following Octave code:

```
# Configuration (a)
S = [300 40];
p = .9;
P = [  0   1; \
     1-p   p ];
V = qncsvisits(P);
DA = S.*V
XA = 1/max(DA);
```

```
# Configuration (b)
S = [300 30];
p = .75;
P = [  0   1; \
     1-p   p ];
V = qncsvisits(P);
DB = S.*V
XB = 1/max(DB);
```

The instruction `V=qncsvisits(P)` computes the visit ratios `V` from the routing matrix `P`; the visit ratio V_k at center k satisfies the equation $V_k = \sum_{k=1}^{N} P_{j,k} V_j$. The visit ratios are used to compute the service demands `DA` and `DB` for configurations (a) and (b), respectively. The service demand D_k at center k is defined as $D_k = S_k V_k$, and represents the total time spent by a request on each server. Finally, the throughputs `XA` and `XB` are the inverse of the maximum service demand. We get `XA` $= 2.5 \times 10^{-3}$ jobs/s, `XB` $= 3.3 \times 10^{-3}$ jobs/s.

Figure 3 shows the upper bounds (continuous line) on the system throughput X computed by `qncsbsb()`. The dashed lines are the exact values computed using the MVA as implemented in `qncsmva()`. We conclude that scenario (b) – large pool of slow disk – allows higher throughput with a sufficiently large number of concurrent jobs.

4.3 Multiclass QN Analysis

We now consider the more complex model of scientific compute farm shown in Figure 4. The system has three classes of jobs that access data stored on four disk servers and three tape libraries. Each job spends some time doing CPU-intensive computations and then accesses data on external storage. Tape libraries and disk servers are modeled as $M/M/1$ service centers.

Let N be the total number of jobs. Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3)$ be the population mix of the network, where β_c is the fraction of class c jobs, $0 \leq \beta_c \leq 1$ and $\beta_1 + \beta_2 + \beta_3 = 1$. Thus, the

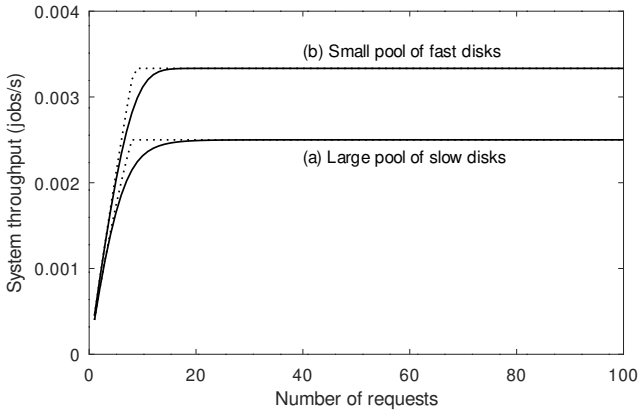

Figure 3: System throughput X for the two configurations. The continuous line is the upper bound, dashed line is the exact value computed using MVA

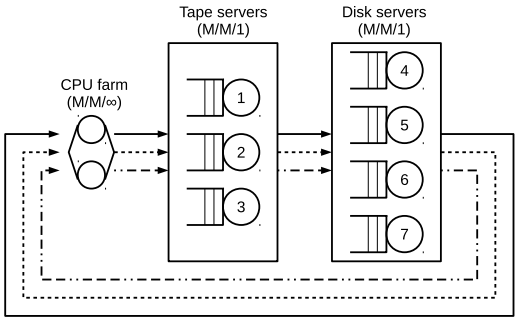

Figure 4: Multiclass closed network model of a scientific compute farm.

number of class c jobs is $N_c = \beta_c N$ rounded to the nearest integer. Let $D_{c,i}$ be the service demand of class c requests at center i (recall that the service demand is the product of the mean service time and the number of visits, $D_{c,i} = S_{c,i}V_{c,i}$). Let Z_c be the average duration of a CPU burst of a class c job. The parameter values are shown in Table 4.

We consider $N = 300$ jobs, and we want to study how different population mixes $\boldsymbol{\beta}$ affect the system throughput X. The following GNU Octave code computes the per-class utilizations $U_{c,i}$, response times $R_{c,i}$, mean queue lengths $Q_{c,i}$ and throughput $X_{c,i}$ when $\boldsymbol{\beta} = (0.2, 0.3, 0.5)$:

```
N = 300; # n. of jobs
S = [100 140 200  30  50  20  10; # demands
     180  10  70  10  90 130  30;
     280 160 150  90  20  50  18];
Z = [2400 1800 2100];          # CPU bursts
V = ones(size(S));             # n. of visits
m = ones(1,columns(S));        # n. of servers
beta = [0.2, 0.3, 0.5];        # population mix

pop = round(N*beta); pop(3) = N-pop(1)-pop(2);
[U R Q X] = qncmmva(pop, S, V, m, Z);
X_sys = sum(X(:,1) ./ V(:,1)); # System tput
```

The system throughput of a multiclass network is $X_{\text{sys}} = \sum_c X_c$, where X_c is the class c throughput. The values of X_c can be computed from the individual servers throughput $X_{c,i}$ as $X_c = \sum_i X_{c,i}/V_{c,i}$. In the example above we get $X_{\text{sys}} = 0.0053793$.

Param	Description	Class 1	Class 2	Class 3
$D_{c,1}$	Tape Server	100	180	280
$D_{c,2}$	Tape Server	140	10	160
$D_{c,3}$	Tape Server	200	70	150
$D_{c,4}$	Disk Server	30	10	90
$D_{c,5}$	Disk Server	50	90	20
$D_{c,6}$	Disk Server	20	130	50
$D_{c,7}$	Disk Server	10	30	18
Z_c	Cpu farm	2400	1800	2100

Table 4: Parameters for the model in Figure 4

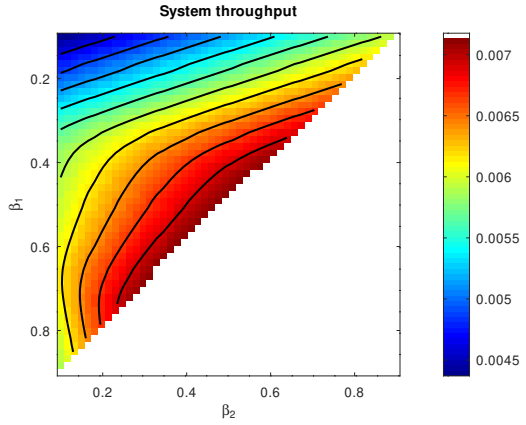

Figure 5: System throughput X as a function of the population mix $\boldsymbol{\beta} = (\beta_1, \beta_2, 1-\beta_1-\beta_2)$. (Best viewed in color)

Even on such a small network, `qncmmva()` requires about $170s$ of CPU time on an Intel i7-4790 CPU at 3.60GHz running Ubuntu Linux 18.04 with GNU Octave 5.1. This makes the multiclass MVA algorithm impractical for this type of study, since analyzing many population mixes would require a large amount of time. We therefore resort to the much faster Bard-Scweitzer approximation, realized by function `qncmmvabs()`.

Figure 5 shows the system throughput for different population mixes. Each square corresponds to a combinations of $\beta_1, \beta_2, \beta_3 = N - \beta_1 - \beta_2$. Contour lines show the regions of the parameter space of equal throughput; the population mixes producing the maximum throughput are those towards the center of the image.

Note that Figure 5 can be computed in about five seconds using `qncmmvabs()` on the same system above, i.e., orders of magnitude faster than the time required by the multiclass MVA implementation from function `qncmmva()`.

5. CONCLUSIONS

In this paper we presented the `queueing` package for GNU Octave. The package provides functions for analyzing Markov Chains, single-station queueing systems and product- and some non product-form QN models; exact, approximate and bound analysis of are supported.

The `queueing` package is available at `https://octave.sourceforge.io/queueing/index.html` and can be used, modified and distributed under the terms of the GNU General Public License (GPL) version 3 or later.

6. REFERENCES

[1] I. F. Akyildiz. Mean Value Analysis for Blocking Queueing Networks. *IEEE Trans. Softw. Eng.*, 1(2):418–428, Apr. 1988.

[2] Y. Bard. Some Extensions to Multiclass Queueing Network Analysis. In *Proc. 4th Int. Symp. on Modelling and Performance Evaluation of Computer Systems*, volume 1, pages 51–62, Feb. 1979.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *JACM*, 22(2):248–260, 1975.

[4] M. Bertoli, G. Casale, and G. Serazzi. JMT: performance engineering tools for system modeling. *SIGMETRICS Performance Evaluation Review*, 36(4):10–15, 2009.

[5] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley, 1998.

[6] J. P. Buzen. Computational Algorithms for Closed Queueing Networks with Exponential Servers. *Comm. ACM*, 16(9):527–531, Sept. 1973.

[7] P. Canadilla. *queueing: Analysis of Queueing Networks and Models*, 2019. R package version 0.2.12.

[8] G. Casale. A note on stable flow-equivalent aggregation in closed networks. *Queueing Syst. Theory Appl.*, 60:193–202, December 2008.

[9] G. Casale. Automated Multi-paradigm Analysis of Extended and Layered Queueing Models with LINE. In *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, pages 37–38, New York, NY, USA, 2019. ACM.

[10] G. Casale, M. Gribaudo, and G. Serazzi. Tools for Performance Evaluation of Computer Systems: Historical Evolution and Perspectives. In *Performance Evaluation of Computer and Communication Systems. Milestones and Future Challenges. IFIP WG 8.3/7.3 Int. Workshop, PERFORM 2010*, volume 6821 of *LNCS*, pages 24–37. 2011.

[11] G. Casale, R. R. Muntz, and G. Serazzi. Geometric Bounds: a Non-Iterative Analysis Technique for Closed Queueing Networks. *IEEE Trans. on Computers*, 57(6):780–794, June 2008.

[12] P. J. Denning and J. P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Comput. Surv.*, 10(3):225–261, Sept. 1978.

[13] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring. *GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations*, 2020. Accessed on 2020-12-23.

[14] N. J. Gunther. *The Practical Performance Analyst: Performance-by-Design Techniques for Distributed Systems*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[15] D. I. Heiman, N. Mittal, and K. S. Trivedi. Dependability modeling for computer systems. In *Proc. Ann. Reliability and Maintainability Symposium*, pages 120–128, 1991.

[16] J. R. Jackson. Jobshop-Like Queueing Systems. *Manage. Sci.*, 50(12 Supplement):1796–1802, 2004.

[17] T. Kerola. The composite bound method for computing throughput bounds in multiple class environments. *Perf. Eval.*, 6(1):1–9, Mar. 1986.

[18] L. Kleinrock. *Queueing Systems: Volume I–Theory*. Wiley Interscience, New York, 1975.

[19] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall, 1984.

[20] M. Reiser and S. S. Lavenberg. Mean-Value Analysis of Closed Multichain Queuing Networks. *J. of the ACM*, 27(2):313–322, Apr. 1980.

[21] P. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proc. Int. Conf. on Stochastic Control and Optimization*, June 1979.

[22] H. Schwetman. Implementing the Mean Value Algorithm for the Solution of Queueing Network Models. Technical Report CSD-TR-355, Purdue University, Feb. 5 1982.

[23] K. S. Trivedi and R. Sahner. SHARPE at the Age of Twenty Two. *SIGMETRICS Performance Evaluation Review*, 36(4):52–57, Mar. 2009.

[24] J. Zahorjan, K. C. Sevcick, D. L. Eager, and B. I. Galler. Balanced Job Bound Analysis of Queueing Networks. *Comm. ACM*, 25(2):134–141, Feb. 1982.