The `qnetworks` Toolbox: a Software Package for Queueing Networks Analysis

Moreno Marzolla

Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura Anteo Zamboni 7, I-40127 Bologna, Italy
`marzolla@cs.unibo.it`

Abstract. Queueing Networks (QNs) are a useful performance modelling notation. They can be used to describe many kinds of systems, and efficient solution techniques have been developed for some classes of QN models. Despite the fact that QNs have been extensively studied, very few software packages for QN analysis are available today. In this paper we describe the `qnetworks` toolbox, a free software package for QN analysis for GNU Octave. `qnetworks` provides implementations of solution algorithms for single station queueing systems as well as for product and some non product form QN models. Exact, approximate and bound analysis can be performed. Additional utility functions and algorithms for Markov Chains analysis are also included. The `qnetworks` package is available as free and open source software, allowing users to study, modify and extend the code. This makes `qnetworks` a viable teaching tool.

1 Introduction

QNs are a very powerful modelling notation; they can be applied to many different domains, including computer networks, supply chain analysis, software systems, street traffic and others [1]. QNs have been extensively studied and a vast literature of solution algorithms exists. QN models can be evaluated either by simulation, or using analytical and numerical techniques. Simulation has the advantage of being able to evaluate any kind of system, including extended QN models for which other solution techniques are either not available, or only produce approximate results. However, simulation can require significant time to accurately evaluate complex models, and the computed results are only given as confidence intervals. Furthermore, evaluation of the same model with different parameters (the so-called "what-if" analysis) is computationally costly as it involves a large number of simulation runs.

Many numerical solution techniques for QN models exist (see [2] and references therein); despite this, there is a shortage of software tools implementing these algorithms. This is particularly unfortunate for many reasons: people keep reimplementing the same old algorithms over and over again, which is error prone and time consuming. This is especially true since some QN algorithms can be tricky to implement correctly due to their complexity. Effort put on implementing old algorithms could be better spent solving interesting modelling problems, or developing new solution techniques for QN models.

In this paper we present `qnetworks`, a QN analysis package written in GNU Octave. GNU Octave [3] is an interpreted language for numerical computations very similar to MATLAB[1][4], to which it is mostly compatible. `qnetworks` provides a set of functions for analyzing Product form (PF) as well as some non PF Queueing Network models; computation of performance bounds, evaluation of single-station queueing systems and Markov Chain analysis are also possible. `qnetworks` is free and open source software: users can inspect, modify and redistribute the code, which makes `qnetworks` a viable teaching tool.

`qnetworks` is not an integrated modelling tool, like JMT [5] or RESQ [6]. Rather, it is a library of functions which can be used as building blocks for analyzing QN models. The Octave interactive environment provides the "glue" which allows complex models to be quickly analyzed, enabling a greater degree of flexibility which is usually not provided by rigid integrated modelling environments. Models can be defined and solved programmatically, so that fully automated batch analysis can be easily implemented. However, a significant understanding of QN modelling is necessary in order to use `qnetworks`. For this reason, casual users may prefer a less flexible but more user-friendly tool such as JMT.

Different usage scenarios for `qnetworks` can be identified:

- **Incremental model development**: `qnetworks` and GNU Octave are an ideal platform for rapid prototyping and iterative refinement of QN models. Models can be defined and analyzed quickly using the function provided by `qnetworks`. The Octave language provides very convenient features for vector manipulation which allow models to be defined concisely.
- **Modelling environment**: large and complex performance studies can be performed, as models involving repetitive or embedded structures can be easily defined. Ad-hoc solution techniques can be realized on top of the available functions. As a specific example, we show later in this paper how hierarchical modelling with flow-equivalent service centers can be done with `qnetworks`, even if no facility to perform such kind of analysis is provided by the package.
- **Queueing Network research**: new QN analysis algorithms can be implemented inside `qnetworks` and tested against existing ones. Contributions to the `qnetworks` package are highly welcome. For many QN algorithms described in the literature, no implementation is readily available. We hope that `qnetworks` will encourage researchers and practitioners to provide implementations of their own algorithms, so that others can use and improve them.
- **Teaching**: `qnetworks` is suitable for introducing QN modelling concepts and solution techniques. Students can immediately get a visual feedback from the

[1] MATLAB is a trademark of The MathWorks Inc.

solution of QN models by using the graphing capabilities provided by GNU Octave. All plots in this paper has been produced by GNU Octave after solving the appropriate model with `qnetworks`.

In order to partially support the above claims, most of this paper is structured in a tutorial style, showing how `qnetworks` can actually be used in simple modelling studies. In Sect. 2 we briefly review related works in the area of QN software. In Sect. 3 we introduce some basic concepts and definitions about QNs. In Sect. 4 we illustrate the features of `qnetworks` and the algorithms which have been implemented. In Sect. 5 we give some usage examples to demonstrate how `qnetworks` can be used in practice. Section 6 contains some performance considerations. Finally, Sect. 7 describes conclusions and future works.

2 Related Works

Over the years, many software packages for the solution of QN models have been developed. As an example, see the list available at `http://web2.uwindsor.ca/math/hlynka/qsoft.html`; however, that most of the tools listed there are of very limited scope, obsolete or no longer available (many hyperlinks are actually broken).

The Research Queueing Package (RESQ) [6] developed at IBM Research was one of the first very successful QN analysis packages. It provided a modelling language for describing extended QN models, which then could be solved by either analytical of simulation techniques. A graphical user interface (called RESQME [7]) was developed in order to facilitate the model definition process. A similar tool was QNAP2 [8], which provided different solution methods (analytical or simulation-based) for analyzing product and non-product form QNs. Networks are described using a textual notation; the QNAP2 tool was written in FORTRAN 77.

Unfortunately, both QNAP2 and RESQ are no longer available. Among the tools which are still available and in use are SHARPE, PDQ and JMT. The Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) [9] is an integrated package for describing and analyzing hierarchical stochastic models, including QN, fault trees, reliability models and so on. Pretty Damn Quick (PDQ)[2] is a QN package with bindings for multiple languages (including Java, PHP, Perl, Python and C). PDQ implements the exact and approximate Mean Value Analysis (MVA) algorithm for closed QNs.

Java Modelling Tools (JMT)[3] [5] is a recent free and open source tool for the construction and evaluation of QN models. JMT is developed by the Performance Evaluation Lab of the Politecnico di Milano, Italy. This tool deserves special consideration, because it is actively developed, highly portable (it is written in Java) and is capable of handling a large class of QN models. JMT supports fixed

[2] `http://www.perfdynamics.com/Tools/PDQcode.html`

[3] `http://jmt.sourceforge.net/`

capacity regions, blocking, non exponential service times, general routing strategies, priorities and other advanced features. Its graphical interface makes JMT particularly suited for inexperienced users with little or no background on QN analysis. While JMT uses simulation to analyze QN models (it also implements the MVA algorithm), `qnetworks` provides analytical solution techniques which, for some classes of models, are much faster and more accurate. Moreover, tools like `qnetworks` are more appropriate in performance studies involving automated construction and analysis of the model.

3 Queueing Networks

QNs are used to describe systems consisting of a collection of resources and a population of requests (or jobs) which circulate demanding service from the resources. Each resource consists of a *service center*, which is represented by a queue connected to a number of identical servers. A QN model contains a finite number K of service centers. In an *open* network there are infinite streams of requests originating outside the system, which arrive to center k with rate λ_k; requests can leave the system from any node. In a *closed* network there is a fixed population of N requests which continuously circulate through the system. *Mixed* models are also possible, in which there are multiple chains of requests, some of which are open and other closed.

QN analysis for single-class networks usually involves computing the steady-state probabilities $\pi_k(i)$ that there are i requests at center k. A class of QN models is said to have *product-form solution* if the steady state of the network can be expressed as the product of factors describing the state of each individual node. The first class of Product form Queueing Networks (PFQNs) was identified by Jackson [10] who discovered that single-class, open networks with the following properties have PF solution:

- Each node of the network can have Poisson arrivals from outside; a job can leave the network from any node. λ_k denotes the external arrival rate to node k. Arrival rates may depend on the total population of the network.
- All service times are exponentially distributed, and service discipline at all nodes is First-Come First-Served (FCFS).
- The k-th node consists of $m_k \geq 1$ identical servers with average service time S_k. The service time S_k may depend on the number of requests at node k.

The result of Jackson has been later extended to closed networks by Gordon and Newell [11], and to open, closed and mixed networks with multiple request classes by Baskett, Chandy, Muntz and Palacios (BCMP) [12]. Specifically, BCMP networks satisfy the following properties:

- Service discipline at each node can be FCFS, Processor Sharing (PS), Infinite Server (IS) or Last-Came First-Served, Preemptive Resume (LCFS-PR).

- Service times for FCFS nodes must be exponentially distributed and class-independent. Service times for the other kind of nodes must have rational Laplace transform, and can in general be class-dependent. The service time S_{ck} of class c requests at service center k might depend on the number of requests at that center.
- There are L disjoint routing chains; each chain may be either open or closed.
- External arrivals to node k (if any) must be a Poisson process. λ_{ck} denotes the class c arrival rate at service center k.
- A class r customer completing service at queue i will either move to queue j as a class s request with probability P_{risj}, or leave the system with probability $1 - \sum_{j,s} P_{risj}$ which can be nonzero for some subset of queues serving open chains.

Additional network types have been shown to have PF solution as well. PFQNs are of particular interest because they have efficient solution algorithms; furthermore, despite their limitations (as stated above) PFQNs are general enough to be useful for modelling large classes of actual systems. Unfortunately, there are many situations which can be encountered in modern systems which can only be represented with extended QN models which do not have PF solution. For example, fork-join parallelism, simultaneous resource possession, non-exponential service times and blocking due to finite capacity queues lead to networks which in general do not have PF solution. In some cases, approximate analysis is possible (the approach of flow-equivalent centers illustrated in Sec. 5.3 is widely used); in other cases, the network can be analyzed through simulation.

4 Overview of `qnetworks`

`qnetworks` is a collection of numerical algorithms written in GNU Octave for exact or approximate solution of single and multiclass QN models; open, closed or mixed networks are supported. GNU Octave has been chosen for different reasons. It is free software, available on multiple operating systems, including Windows, MacOSX and most Unix variants. Furthermore, GNU Octave is mostly compatible with MATLAB, a language for numerical computations which is widely used in the research and industrial community. Thus, many students, researchers or practitioners interested in the numerical analysis of QN models will likely be already familiar with GNU Octave or MATLAB.

Technically, the `qnetworks` package is a set of *m-scripts*; an m-script is a program specified in the GNU Octave interpreted language. While m-scripts are slower than compiled code, they allow maximum portability as they can be executed on any platform where the Octave interpreter has been ported. It should be observed that in most practical cases execution times of the algorithms in `qnetworks` are acceptable, so there is currently no need to rewrite the functions as compilable C/C++ code (see Sect. 6 for actual execution times of `qnetworks`).

Table 1. Some functions provided by the `qnetworks` package to analyze QN models

	Supported network type			
Function Name	**Open**	**Closed**	**Single**	**Multi**
qnopensingle()	$\surd$	–	$\surd$	–
qnopenmulti()	$\surd$	–	–	$\surd$
qnconvolution()	–	$\surd$	$\surd$	–
qnconvolutionld()	–	$\surd$	$\surd$	–
qnclosedsinglemva()	–	$\surd$	$\surd$	–
qnclosedsinglemvald()	–	$\surd$	$\surd$	–
qnclosedmultimva()	–	$\surd$	–	$\surd$
qnclosedmultimvaapprox()	–	$\surd$	–	$\surd$
qnmix()	$\surd$	$\surd$	–	$\surd$
qnsolve()	$\surd$	$\surd$	$\surd$	$\surd$
qnmvablo()	–	$\surd$	$\surd$	–
qnmarkov()	–	$\surd$	$\surd$	–
qnopenab()	$\surd$	–	$\surd$	–
qnclosedab()	–	$\surd$	$\surd$	–
qnopenbsb()	$\surd$	–	$\surd$	–
qnclosedbsb()	–	$\surd$	$\surd$	–
qnclosedgb()	–	$\surd$	$\surd$	–

4.1 Single Station Queueing Systems

`qnetworks` provides functions for analyzing several types of single-station queueing systems [13,2]: $M/M/1$[4] and $M/M/m$ [`qnetworks` functions qnmm1() and qnmmm(), respectively], $M/M/1/k$ and $M/M/m/k$ [qnmm1k() and qnmmmk()], $M/M/\infty$ [qnmminf()], asymmetric $M/M/m$ which contains m servers with possibly different service rates [qnammm()], $M/G/1$ with general service time distribution [qnmg1()], $M/H_m/1$ with hyperexponential service time distribution [qnmh1()]. For each kind of system, the following performance measures are computed: utilization U, mean response time R, average number of requests in the system Q and throughput X.

4.2 Queueing Networks

`qnetworks` provides a set of functions for analyzing product-form and non product-form QNs; Table 1 lists some of these functions, specifying for each one whether it can be applied to open or closed networks, and whether it supports single or multiple request classes.

[4] We use the standard Kendall's notation $A/B/C/K$, where A denotes the arrival process (M=Poisson), B denotes the service time distribution (M=exponential), C is the number of servers, K is the capacity of the system

Algorithms for Product-form Networks. For open networks, the qnopensingle() and qnopenmulti() functions can be used on networks with single or multiple customer classes, respectively. These functions implement the well known equations for Jackson networks, and the extensions for BCMP open multiclass networks [14,2].

For PF closed networks, exact as well as approximate algorithms are provided. For single-class closed networks, the MVA [15] and convolution [16] algorithms are provided by the qnclosedsinglemva() and qnconvolution() functions respectively. Both support FCFS, LCFS-PR, PS and IS nodes; single and multiple server FCFS nodes are supported as well. Note that the BCMP theorem allows a general form of state-dependent service rates: for instance IS, PS and LCFS-PR nodes may exhibit service rates depending on the population of a sub-network. This is useful for modeling some kind of systems, but is currently not supported by `qnetworks`.

qnclosedsinglemvald() and qnconvolutionld() implement the MVA and convolution algorithms, respectively, for networks with general load-dependent service centers. We provide separate functions for networks with and without general load-dependent service centers because the former have a higher computational cost and require more memory. Thus, we provide efficient implementations for the common case of networks without general load-dependent centers, while still allowing users to handle the general case using different functions.

It is important to remember that the MVA and convolution algorithms have very different numerical properties [17]. In particular, MVA is numerically stable for models with only fixed rate and infinite server queues; this is true also when extreme parameter values are considered. Unfortunately, MVA does not retain its numerical stability for variable rate queues (general load-dependent service centers). On the other hand, the convolution algorithm behaves badly for fixed rate and infinite server queues, but has better numerical stability for variable rate queues when the probability of small queue lengths at those queue is small.

For PF multiclass closed networks we implemented the multiclass MVA algorithm in the qnclosedmultimva() function. For networks with K service centers, C customer chains and population vector $(N_1, N_2, \dots N_C)$, the multiclass MVA algorithm requires time $O\left(CK \prod_{i=1}^{C}(N_i+1)\right)$ and space $O\left(K \prod_{i=1}^{C}(N_i+1)\right)$. Due to its computational complexity, the multiclass MVA algorithm is appropriate only for networks with small population and a limited number of customer chains. For larger networks, approximations based on the MVA have been proposed in the literature. `qnetworks` provides the Bard and Schweitzer approximation [18,14] in function qnclosedmultimvaapprox().

Mixed multiclass PFQNs [12] are handled by the qnmix() function. In mixed networks, customer classes can be partitioned into disjoint chains; some chains are open, and the others are closed and have fixed populations. The qnmix() function does does not currently supports general load-dependent queueing centers.

Finally, the higher-level function qnsolve() can be used as a single front-end to all the algorithms described above. This function uses a less efficient, but more flexible representation of the network to be evaluated, and delegates the actual analysis to the appropriate solution algorithm (if available), depending on the network type.

Algorithms for non Product-form Networks. In blocking QNs, queues have a fixed capacity: a request joining a full queue will block until a slot in the destination node becomes available. Different blocking strategies have been investigated in the literature (see [19] for a detailed review). The qnmvablo() function implements the MVABLO algorithm [20]. MVABLO is based on an extension of MVA, and computes approximate solutions 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 server until a slot is available at the destination.

Networks with blocking can also be analyzed with the qnmarkov() function. This function supports either open or closed, single-class networks where all queues have fixed capacity. Exact performance measures are derived by explicit construction of the underlying Markov Chain. The qnmarkov() function is appropriate for small networks only, due to the exponential growth of the Markov Chain size as the network increases.

Bound Analysis. It is often useful to compute bounds for the system throughput X or response time R. Performance bounds can be obtained very quickly, and can be useful for many performance studies, such as those involving on-line performance tuning of systems. `qnetworks` implements three different algorithms for computing performance bounds: Asymptotic Boundss (ABs) [21] for open and closed networks (functions qnopenab() and qnclosedab() respectively), Balanced System Boundss (BSBs) [22] for open and closed networks (functions qnopenbsb() and qnclosedbsb() respectively) networks, and Geometric Boundss (GBs) [23] for closed networks (function qnclosedgb()).

4.3 Validation

Almost all the functions provided by the `qnetworks` package include unit tests embedded inside the m-files. The tests can be invoked using Octave test function; it is also possible to run all tests with a single command, which is particularly useful for checking the whole source distribution before releasing a new version.

As for many numerical softwares, testing QN packages can be nontrivial [24]. When possible, testing is done by computing results on reference networks for which correct values are known (e.g., from the literature). When exact solutions are not known, results can still be validated by computing them with different algorithms. For example, the MVA and convolution algorithms can be applied to the same network, and they must provide the same results (apart for deviations due to numerical inaccuracies [17]). As another example, a $M/M/1/K$ queue is a special case of an $M/M/m/K$ queue with $m = 1$ servers. Thus, in this case the performance results provided by the qnmmmk() and qnmm1k() must be the same. Finally, even when results cannot be directly compared, consistency checks

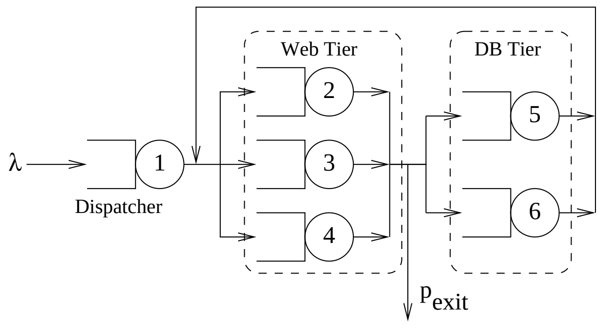


Fig. 1. Open model of a two-tier E-commerce site; arrows denote nonzero flows

can nevertheless be done. For example, the bounds on the system throughput computed by the AB or BSB equations must include the exact result provided by the MVA algorithm. In this way it is possible to cross-check the `qnclosedab()`, `qnclosedbsb()` and `qnclosedmva()` functions.

5 Examples

In this section we present some usage examples of the `qnetworks` package.

5.1 Open Network

Let us consider a simple model of a two-tier E-commerce site. The model is shown in Fig. 1 and consists of six FCFS service centers. Center 1 is the *dispatcher*, and is responsible for routing incoming requests to one of the Web servers (centers 2–4) with uniform probability. After being processed by one of the Web servers, each request may leave the system with probability p_{exit}, or be forwarded to one of the Database servers (centers 5 and 6).

We assume average service times $S_1 = 0.5$ at the dispatcher, $S_2 = S_3 = S_4 = 0.8$ at the Web servers and $S_5 = S_6 = 1.8$ at the Database servers; we set the arrival rate at center 1 as $\lambda_1 = 0.1$ requests/s and exit probability $p_{\text{exit}} = 0.5$. The transition probability matrix P is:

$$P = \begin{pmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 1/4 & 1/4 \\ 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \end{pmatrix}$$

This model can be defined with the following GNU Octave code:

```
p_exit = 0.5; # exit probability
```

```
i = 2:4; # indexes of Web servers
j = 5:6; # indexes of DB servers
P = zeros(6,6);
P(1,i) = 1/3;
P(i,j) = (1-p_exit)/2;
P(j,i) = 1/3;
S = [0.5 0.8 0.8 0.8 1.8 1.8];
lambda = [0.1 0 0 0 0 0];
V = qnvisits(P,lambda);
```

Note the use of *array slicing* to define the matrix P: variables i and j are ranges, and the single instruction P(j,i)=1/3 sets $P_{ji} = 1/3$ for all $j \in \{5, 6\}$ and $i \in \{2, 3, 4\}$.

In the code above we compute the visit counts V_k to center k using the qnvisits() function. The visit counts V_k satisfy the equality $V_k = \lambda_k + \sum_{j=1}^{K} V_j P_{jk}$. In the example above, we get $V_1 = 1$, $V_2 = V_3 = V_4 = 0.6\bar{6}$ and $V_5 = V_6 = 0.5$.

The network is a PFQN and can be evaluated using the qnopensingle() function, as follows:

```
[U R Q X] = qnopensingle(sum(lambda),S,V);
```

where **sum**(lambda) is the global arrival rate $\sum_k \lambda_k$. The resulting utilizations are $U_1 = 0.05$, $U_2 = U_3 = U_4 = 0.05\bar{3}$ and $U_5 = U_6 = 0.09$. It is also easy to compute the maximum arrival rate λ_{sat} which the system can sustain: it is known that $\lambda_{\text{sat}} = 1/\max_k\{S_k V_k\}$, and can be computed by the GNU Octave expression lambda_sat=1/**max**(S.*V), which gives $\lambda_{\text{sat}} = 1.1\bar{1}$; S.*V is the vector of element-by-element products of S and V.

5.2 Closed Network

We show in Fig. 2 a closed model which is based on the open model of Fig. 1. In the closed model we have a fixed population of $N = 20$ requests. Each request spends an average delay $Z = 5$ outside the system between service cycles. Z is also known as *think time* and is represented by the IS node in Fig. 2.

Again, we can define and solve the model with the following GNU Octave code:

```
p_back = 0.5; # back probability
i = 2:4; # range of Web servers
j = 5:6; # range of DB servers
P = zeros(6,6);
P(1,i) = 1/3;
P(i,j) = (1-p_back)/2;
P(i,1) = p_back;
P(j,i) = 1/3;
S = [0.5 0.8 0.8 0.8 1.8 1.8];
V = qnvisits(P); # Compute visit counts
Z = 5; # Think Time
```

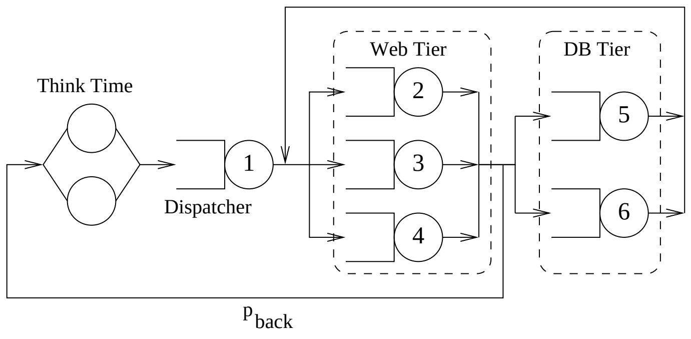


Fig. 2. Closed model of a two-tier E-commerce site

```
N = 20; # Population
m = ones(1,6); # m(k)=number of servers at center k
[U R Q X] = qnclosedsinglemva(N,S,V,m,Z);
```

The qnclosedsinglemva() function solves the given network using the MVA algorithm. The computed utilizations are $U_1 = 0.50112$, $U_2 = U_3 = U_4 = 0.53453$ and $U_5 = U_6 = 0.90202$.

5.3 Flow-equivalent Centers

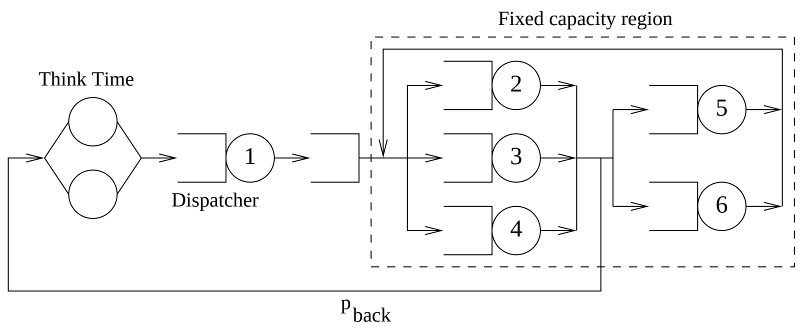
We now show how a more complex analysis can be performed with `qnetworks`. Let us consider the closed model of Fig. 3(a). which is similar to the one of Fig. 2 with the additional introduction of a capacity constraint: no more than M requests can be in the dashed region at the same time. Any request entering the fixed capacity region when it is full, must wait in a queue until a request leaves the region.

Models with capacity constraints have in general no PF solution. However, in this case we can replace the fixed capacity region with a load-dependent service center [14], and solve the resulting model (which does have PF solution). More specifically, we proceed as follows:

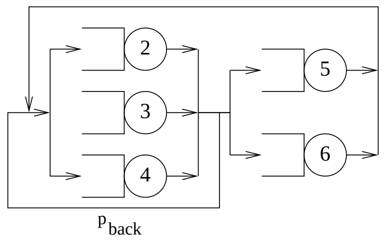
1. Define the complete model as in Sect. 5.2. Then, "short circuit" center 1 by setting its service time to zero (S(1)=0); we get the submodel in Fig. 3(b).
2. Solve the short-circuited submodel by computing the throughput $X_{\text{sub}}(n)$ along the removed node(s) as a function of the population size $n = 1, 2, \ldots M$. The computed value for $X_{\text{sub}}(n)$ can be used to derive the average service time $S_{\text{sub}}(n)$ of the flow-equivalent center which will replace the capacity constrained region. $S_{\text{sub}}(n)$ is defined as:

$$S_{\text{sub}}(n) = \begin{cases} 1/X_{\text{sub}}(n) & \text{if } 1 \leq n \leq M \\ 1/X_{\text{sub}}(M) & \text{if } M < n \leq N \end{cases}$$

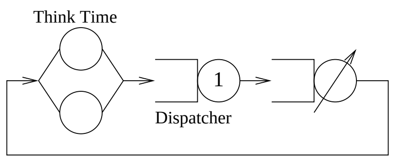
and can be computed with the following GNU Octave code:



(a) Original model



(b) Isolated submodel



(c) Equivalent model

Fig. 3. Closed model with capacity constraint

```
Ssub = zeros(1,N); # Initialize  to  zero
M = 10; # Capacity constraint
for n=1:M
  [U R Q X] = qnclosedsinglemva(n,S,V);
  Ssub(n) = V(1)/X(1);
endfor
Ssub(M+1:N) = Ssub(M);
```

3. Build an equivalent model (see Fig. 3(c)) starting from the full model with the capacity constrained region replaced by a Flow Equivalent Service Center (FESC). The service times for the FESC are those computed in the previous step. Let S_{kn} be the service time at center k of the equivalent model when there are n requests; we have that $S_{1n} = 0.5$ and $S_{2n} = S_{\mathrm{sub}}(n)$, for all n. The equivalent model is defined and solved as follows:

```
S = [ 0.5*ones(1,N); Ssub ];
V = [1 1];
Z = 5;
[U R Q X] = qnclosedsinglemvald(N,S,V,Z);
```

By repeating the above for different values of the population size N we can produce the plot shown in Fig. 4. We show the system throughput $X(N)$ as a

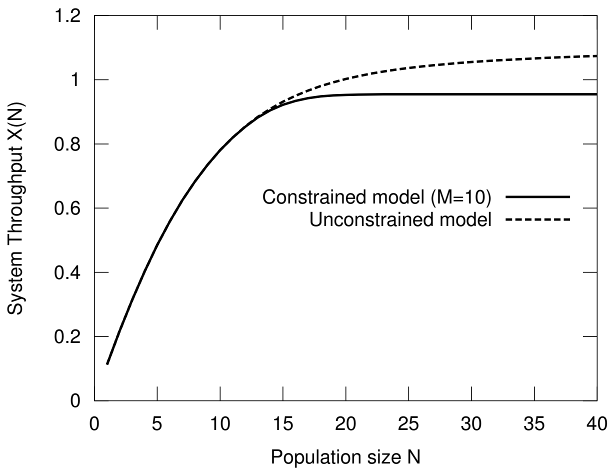


Fig. 4. System Throughput $X(N)$ for the models of Fig. 2 and 3 as a function of the number of requests N.

function of N. As expected, the system saturates shortly after the number of requests N exceeds the population constraint M.

6 Performance Considerations

As already described in Sect. 4, all functions provided by `qnetworks` are implemented as m-scripts running inside the Octave interpreter. Despite this, performance are in general quite good. To give an example, we consider the implementing the MVA algorithm for single-class closed networks as provided by the `qnclosedsinglemva()` function. Single-class, closed networks are widely used in practice, so it is important to analyze them efficiently.

Figure 5 illustrates the execution time of `qnclosedsinglemva()` for different values of the network size K and population N. The tests have been performed by creating a network with K servers with random service times S_k and visit counts V_k. Execution times have been measured on a Linux PC with an AMD Athlon 64 X2 Dual Core processor 3800+ with 2GB of RAM, using GNU Octave version 3.2.3. For each combination of K and N, we consider the average execution time of 5 runs.

We observe that the largest network ($K = 2000$ service centers and $N = 500$ requests) takes only a fraction of a second to be analyzed on the test machine. We also observe that for a fixed population size N, the execution time increases linearly with the number of service centers K. This is expected, as the computational complexity of MVA for single-class, load-independent service centers is $O(NK)$.

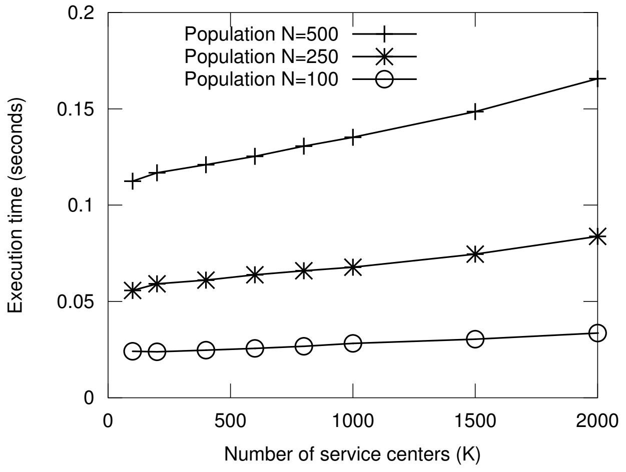


Fig. 5. Execution time of the qnclosedsinglemva() function (in seconds, average of five measurements).

7 Conclusions

In this paper we described `qnetworks`, a QN analysis package for GNU Octave. `qnetworks` supports single station queueing systems, as well as open, closed or mixed networks; implementations of the MVA and convolution algorithm for product-form QNs are provided. Moreover, computation of performance bound and approximate analysis of some classed of non product-form networks is also possible. We gave some practical usage example showing how the Octave environment coupled with `qnetworks` can be used to conduct performance modelling studies.

We are currently extending `qnetworks` by including support for non exponential single station queueing systems, as well as additional classes of QNs models, in particular QNs with blocking, general state-dependent routing or state-dependent service times.

`qnetworks` is available at `http://www.moreno.marzolla.name/software/qnetworks` and can be used, modified and distributed under the terms of the GNU General Public License (GPL) version 3.

References

1. Serazzi, G.: Performance Evaluation Modelling with JMT: learning by examples. Technical Report 2008.09, Politecnico di Milano (2008)
2. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. Wiley (1998)
3. Eaton, J.W.: GNU Octave Manual. Network Theory Limited (2002)

4. The MathWorks Inc. Natick, Massachussets: MATLAB. (2003)
5. Bertoli, M., Casale, G., Serazzi, G.: JMT: performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev. **36**(4) (2009) 10–15
6. Sauer, C.H., Reiser, M., MacNair, E.A.: RESQ: a package for solution of generalized queueing networks. In: AFIPS National Computer Conference. Volume 46 of AFIPS Conference Proceedings., AFIPS Press (1977) 977–986
7. Chang, K.C., Gordon, R.F., Loewner, P.G., MacNair, E.A.: The Research Queuing Package Modeling Environment (RESQME). In: Winter Simulation Conference. (1993) 294–302
8. Véran, M., Potier, D.: QNAP2: A portable environment for queueing systems modelling. Technical Report 314, Institut National de Recherche en Informatique et en Automatique (June 1984)
9. Sahner, R., Trivedi, K.S., Puliafito, A.: Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package. Kluwer Academic Publishers (1996)
10. Jackson, J.R.: Jobshop-like queueing systems. Man. Science **10**(1) (1963) 131–142
11. Gordon, W.J., Newell, G.F.: Closed Queuing Systems with Exponential Servers. Operations Research **15**(2) (1967) 254–265
12. Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, F.G.: Open, closed, and mixed networks of queues with different classes of customers. J. ACM **22**(2) (1975) 248–260
13. Kleinrock, L.: Queueing Systems: Volume I–Theory. Wiley Interscience, New York (1975)
14. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice Hall (1984)
15. Reiser, M., Lavenberg, S.S.: Mean-value analysis of closed multichain queuing networks. Journal of the ACM **27**(2) (April 1980) 313–322
16. Buzen, J.P.: Computational algorithms for closed queueing networks with exponential servers. Comm. ACM **16**(9) (September 1973) 527–531
17. Chandy, K.M., Sauer, C.H.: Computational algorithms for product form queueing networks. Comm. ACM **23**(10) (1980) 573–583
18. Schweitzer, P.: Approximate analysis of multiclass closed networks of queues. In: Proc. Int. Conf. on Stochastic Control and Optimization. (June 1979) 25–29
19. Balsamo, S. De Nitto Personé, V., Onvural, R.: Analysis of Queueing Networks with Blocking. Kluwer Academic Publishers (2001)
20. Akyildiz, I.F.: Mean value analysis for blocking queueing networks. IEEE Transactions on Software Engineering **1**(2) (April 1988) 418–428
21. Denning, P.J., Buzen, J.P.: The operational analysis of queueing network models. ACM Computing Surveys **10**(3) (September 1978) 225–261
22. Zahorjan, J., Sevcick, K.C., Eager, D.L., Galler, B.I.: Balanced job bound analysis of queueing networks. Comm. ACM **25**(2) (February 1982) 134–141
23. Casale, G., Muntz, R.R., Serazzi, G.: Geometric bounds: a non-iterative analysis technique for closed queueing networks. IEEE Transactions on Computers **57**(6) (June 2008) 780–794
24. Schwetman, H.: Testing network-of-queues software. Technical Report CSD-TR-330, Purdue University (January 1 1980)