JOURNAL OF ADVANCED TRANSPORTATION
J. Adv. Transp. 2016; **50**:96–119
Published online 11 September 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/atr.1330

Analyzing and optimizing pedestrian flow through a topological network based on *M*/*G*/*C*/*C* and network flow approaches

Ruzelan Khalid*, Md. Azizul Baten, Mohd. Kamal Mohd. Nawawi and Nurhanis Ishak†

Department of Decision Sciences, School of Quantitative Sciences, Universiti Utara Malaysia, 06010 UUM, Sintok, Kedah, Malaysia

SUMMARY

An *M*/*G*/*C*/*C* state dependent queuing network measures the performance of a system whose service rate decreases with the increasing number of residing entities. However, the performance in terms of throughputs, levels of congestions, the expected number of entities, and the expected service time is typically analyzed based on a series of arrival rates without any further discussion on the optimal arrival rate. This paper derives the optimal arrival rates of corridors in a topological network using calculus and numerical analysis approaches. These optimal rates are then used as capacity parameters in the network's flow model to obtain the optimal arrival rates that maximize its total throughput. To ease the construction and performance evaluation of the network, we design and construct an *M*/*G*/*C*/*C* framework based on the Object-Oriented Programming approach that integrates the LINGO software as an optimization tool. The framework is then tested on virtual and real networks. This framework can be used to develop a more advanced traffic management tool for studying and managing traffic flow through a complex network. Copyright © 2015 John Wiley & Sons, Ltd.

KEY WORDS: *M*/*G*/*C*/*C* state dependent; network flow model; queuing system; performance evaluation; topological network

1. INTRODUCTION

Queuing networks have long been used to explore the effects of capacity constrained resources on common performance measures such as throughputs and response time. The resources' service times in most queuing systems strictly fluctuate according to a statistical distribution regardless of the number of residing entities. Examples of systems that follow this behavior include service, production, and manufacturing processes. Other systems meanwhile physically adjust their service times based on the current number of entities. This behavior can be best described using an *M*/*G*/*C*/*C* state dependent queuing network.

The *M*/*G*/*C*/*C* network imitates how residing entities affect a resource's service time and influence its system's performance. Common performance measures collected are the throughput, blocking probability, expected number of entities, and expected service time. The service time becomes longer when the number of requesting entities increases. However, any decrement of the number will speed up the processing time to offer better service. Examples of systems that follow this behavior are entities moving through a constrained network; for example, a corridor and road.

The maximum number of residing entities in an *M*/*G*/*C*/*C* system is limited by the capacity of its resource. Because the capacity (i.e., the available space to accommodate entities) is fixed, the only parameter that can be controlled to improve its performance is the current number of residing entities. The number is implicitly influenced by the arrival rates of entities into the system. Slow arrival rates

*Correspondence to: Ruzelan Khalid, School of Quantitative Sciences, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah, Malaysia. E-mail: ruzelan@uum.edu.my
† Current address: Faculty of Management and Information Technology, Kolej Universiti Islam Sultan Azlan Shah, Kuala Kangsar, Perak, Malaysia.

make the system process the residing entities faster, but this causes a small throughput at the end. Higher arrival rates make the system process the entities slower, but this causes a higher throughput at the end. Any arrival rates higher than its optimal arrival rate cause congestion instead of improving the throughput. The importance of controlling and analyzing entities' arrival rates to effectively flow them through a network has been highlighted in much literature (e.g., [1–6]).

Our source of innovation is driven by previous research arguing that an *M*/*G*/*C*/C network is an appropriate tool for modeling congestion in a state-dependent system. The approach has extensively been used to evaluate the performance of evacuating occupants from a facility (e.g., multi-story building [1, 7, 8] and hall [9, 10]), flowing vehicles through a road [11], and handling materials in an accumulating conveyer system [12]. The analytical results have also been validated using a discrete event simulation model [13–18] constructed either using a programming language [1] or simulation software [19]. The previous research also suggests that appropriately controlling entities during their travel is crucial to achieve the best possible performance. However, the performance is only derived based on a series of experimental analyses, that is, by inputting a set of arrival rates and observing its impact to the total throughput without any discussions on scientific methods to optimize it. Additionally, the discussions are based on a model specifically designed and constructed for a considered network. These deficiencies motivated us to provide a tool that can generally be used to easily handle and analyze any *M*/*G*/*C*/*C* networks and automatically find arrival rates optimizing their performance based on operations research techniques.

This paper has two objectives. The first objective is to derive the optimal arrival rate of pedestrians at the origin of a corridor that maximizes its throughput. For this, we use calculus and numerical analysis methods. We then show how the optimal arrival rate of each of available corridors in a topological network can be utilized to find the optimal arrival rates of its source corridors, which maximize its total throughput using a *network flow programming* approach. The second objective is to design an *M*/*G*/*C*/*C* software application as a platform for analyzing the performance of a network. The application employs an *Object-oriented Programming* (OOP) approach to ease the network's construction (in terms of its topology and network flow programming model) and performance evaluation (based on the initial and optimal arrival rates) and uses the LINGO optimization commercial software through its Dynamic Link Library (DLL) approach to solve an automated flow programming script that maximizes the network's total flow subject to relevant flow constraints. LINGO is a comprehensive modeling language and solver developed by LINDO Systems Incorporation (Chicago, IL, USA) (www.lindo.com) for efficiently building and solving linear, nonlinear, integer and stochastic optimization models. Thus, our main purpose of using the LINGO software is to solve linear network optimization models created by our application. This proposed approach helps policymakers implement *M*/*G*/*C*/*C* state queuing networks to best flow people through a network especially in emergency cases.

The main contribution of this paper is to present an algorithm for optimizing the arrival rates of pedestrian traffic flows through a topological network based on the combination of *M*/*G*/*C*/*C* state dependent queuing network and *network flow programming* approaches. The demonstration of how our methodology and algorithm can be applied to find the optimal arrival rates of any imaginary and real complex topological networks through a software application is also a novel contribution of this paper. The derived optimal arrival rates will effectively flow entities throughout the network and relatively minimize the time needed to vacant them from any considered environment, for example, a building and hall.

This paper is structured as follows. Section 2 reviews the pedestrian traffic and congestion model and illustrates the effect of arrival rates to the throughputs of some considered corridors. Note that we have to present the *M*/*G*/*C*/*C*'s mathematical background in detail because our intention is to derive its optimal throughput. Section 3 derives the equation maximizing a corridor's throughput using calculus and numerical analysis methods. Combining both methods, we can then find the optimal arrival rate of a corridor optimizing its throughput. In Section 4, we design an *M*/*G*/*C*/*C* framework for developing a tool that easily structures a topological network, analyzes its performance based on inputted arrival rates, and then automatically calculates the optimal arrival rates of each available source corridors optimizing the whole network based on OOP and DLL approaches. In Section 5, we show how the tool structures an imaginary and real topological network and analyzes its performance. This section also discusses the performance of the application in analyzing various numbers of corridors in terms of computational time. Section 6 discusses the practical scope of our methodology especially in dealing

DOI: 10.1002/atr

with the optimal arrival rates. Finally, Section 7 summarizes the findings and presents some conclusions and future work.

2. PEDESTRIAN TRAFFIC AND CONGESTION MODEL

Evacuation planning that deals with evacuees' behavior in a network can be modeled using microscopic or macroscopic approaches. Microscopic approaches consider each individual evacuee as a separate flow object and model the effect of their attributes (e.g., gender and age) and other relevant factors (e.g., blockage and congestion at a particular point) to their movement. Macroscopic approaches meanwhile aggregate the evacuees and model their movements through an evacuation path, for example, the effect of their density on the current flow. Examples of microscopic and macroscopic models based on analytical and simulation techniques and their classification are discussed in detail in [5]. Wang *et al.* [20] meanwhile provide a comprehensive review on various macroscopic models and their performance compared with empirical data observation. There are two approaches of utilizing the macroscopic models. The first approach is to use a relevant model to evaluate the performance of a set of potential routes, while the second approach is to use an optimization technique to optimize the model. We focus on the second approach and use the *M*/*G*/*C*/*C* as our evacuation model.

An *M*/*G*/*C*/*C* queuing network was developed based on Tregenza's empirical study [21], who asserted that a pedestrian's walking speed through a corridor was influenced by the current number of its residing pedestrians. By plotting the graph of crowd density versus mean walking speed, he discovered two significant findings. First, the lone pedestrian's walking speed (V_1) was around 1.5 m/second. Second, the current walking speed (V_n) decreased when the current number of residing pedestrians (n) approaches to the capacity of the corridor (c), with small movement may exist when $n = c$. Whenever $n \geq c + 1$, all movement should then stop; that is, $V_n = 0$. He also discovered that the capacity of a corridor is equal to the highest integer of five times its area in square meters.

The effect of the number of pedestrians on the walking speed was then formalized by Yuhaski and Smith [6]. They presented linear and exponential models of walking speed as follows:

$$\text{Linear}: V_n = \frac{V_1}{c}(c + 1 - n) \tag{1}$$

$$\text{Exponential}: V_n = A \exp\left[-\left(\frac{n-1}{\beta}\right)^{\gamma}\right] \tag{2}$$

where

$$\gamma = \frac{\ln\left[\frac{\ln(V_a/V_1)}{\ln(V_b/V_1)}\right]}{\ln\left(\frac{a-1}{b-1}\right)}$$

$$\beta = \frac{a-1}{\left[\ln\left(\frac{V_1}{V_a}\right)\right]^{1/\gamma}} = \frac{b-1}{\left[\ln\left(\frac{V_1}{V_b}\right)\right]^{1/\gamma}}$$

γ, β = shape and scale parameters for the exponential model
V_n = average walking speed for n pedestrians in a corridor
V_a = average walking speed when crowd density is 2 peds/m^2 = 0.64 m/second
V_b = average walking speed when crowd density is 4 peds/m^2 = 0.25 m/second
V_1 = average walking speed for a single pedestrian = 1.5 m/second
n = number of pedestrians in a corridor
a = $2 \times l \times w$
b = $4 \times l \times w$

DOI: 10.1002/atr

c $= 5 \times l \times w$
l = corridor length in meters, and
w = corridor width in meters.

According to Cheah [22] and Cheah and Smith [23], the exponential model also presents the walking speed for bi-directional and multi-directional flows except the values for parameters V_a and V_b, which are slightly smaller. For bi-directional flows, $V_a = 0.60$ peds/second and $V_b = 0.21$ peds/second while for multi-directional flows, $V_a = 0.56$ peds/second and $V_b = 0.17$ peds/second. This corresponds with what Fruin [24] found that there was a relatively small range in average walking speed between unidirectional, bi-directional, and multi-directional traffic flows and proposed that the capacity a corridor with bi-directional flow is nearly equal to that of unidirectional flow corridor. Based on the models, Yuhaski and Smith [6] developed the limiting probabilities for the number of pedestrians in an *M*/*G*/*C*/*C* model as follows:

$$P_n = \frac{[\lambda E(S)]^n}{n!f(n)f(n-1)\ldots f(2)f(1)} P_0 \quad \text{n} = 1, 2, 3, \ldots, \text{c} \tag{3}$$

where

$$P_0^{-1} = 1 + \sum_{n=1}^{C} \left[\frac{[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)} \right].$$

In this model, λ is the arrival rate to a corridor, $E(S)$ is the expected service time of a single pedestrian in the corridor, P_n is the probability when there are n pedestrians in the corridor, P_0 is the probability when there is no pedestrian in the corridor, and $f(n)$ is the service rate and is given by $f(n) = \frac{V_n}{V_1}$. c meanwhile refers to the capacity of the corridor. Any pedestrians attempting to enter the full capacity corridor will be blocked. The probability of such blocking (P_{balk}) is equal to P_n where n equals to c. Because Cheah and Smith [23] showed that *M*/*G*/*C*/*C* networks are equal to *M*/*M*/*C*/*C* networks, various performance measures of the corridor can then be computed as follows:

$$\theta = \lambda(1 - P_{balk}), E(N) = \sum_{n=1}^{C} nP_n \; and \; E(T) = \frac{E(N)}{\theta} \tag{4}$$

where θ is the throughput of the corridor (in pedestrians per second, that is, peds/second), $E(N)$ is the expected number of pedestrians in the corridor (peds/second), and $E(T)$ is the expected service time in seconds.

Figure 1 charts the effects of arrival rates to the throughputs of considered corridors. Four corridors with relevant lengths and widths are considered, that is, 8×2.5 m, 8×4 m, 10×3 m, and 9.45×1.5 m. For the first three corridors, the traveling distances that pedestrians must walk are the same as their lengths. This situation represents that arrival sources are located at the origin of the corridors. For the fourth corridor, the traveling distance is only 2.7 m. This situation represents that the corridor has multiple arrival sources located along the corridor and the average distance of the arrival sources to the end of the corridor is 2.7 m. More details on this can be found in other articles (e.g., [6,9]). Our objective is to find the arrival rates that maximize the throughput of the corridors.

3. CORRIDOR OPTIMAL FLOW

This section derives the equation optimizing a corridor's throughput. From the equation, we then use available numerical analysis methods to find its optimal arrival rate. As mentioned earlier, the optimal arrival rate is the best arrival rate that effectively and safely flows entities through the corridor with little congestion. Any lower values than the optimal arrival rate will cause fewer throughputs, while

DOI: 10.1002/atr

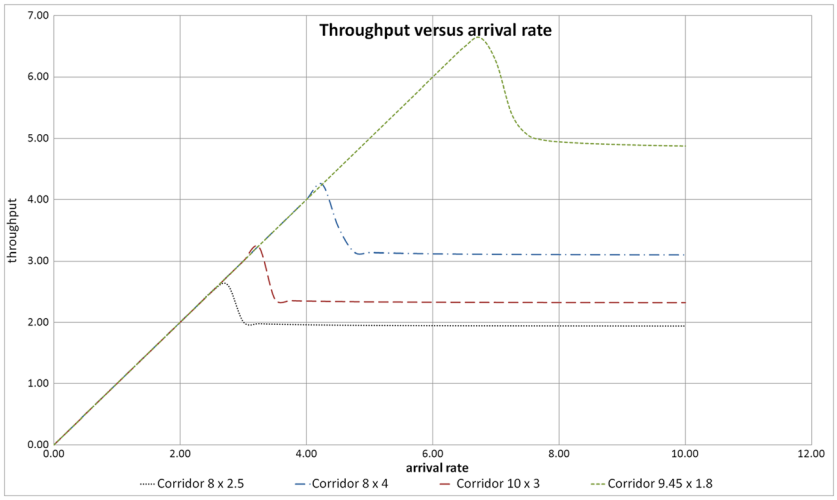


Figure 1. Arrival rates versus throughputs of corridors.

any higher values than the optimal rate will cause congestion without improving the optimal throughput.

3.1. Analytical optimal flow equation maximizing corridor throughput

Figure 1 shows that at a certain level of arrival rates, the throughput of a corridor is maximized. For example, an arrival rate between 2 and 3 peds/second will maximize the throughput of corridor 8 × 2.5 m. For corridor 8 × 4 m, the maximum throughput (i.e., more than 4 peds/second) can be obtained if its arrival rate is controlled between 4 and 5 peds/second. Thus, our objective is to derive the optimal arrival rate using calculus and numerical analysis methods.

From Equation (4), the throughput of an *M*/*G*/*C*/*C* network is defined as

$$\theta = \lambda(1 - P_c)$$

By substituting $P_c = \frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)} P_0$ to the equation, we obtain

$$\theta = \lambda\text{-}\lambda\left\{\frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)}\right\} \Big/ \left\{1 + \sum_{i=1}^{c}\left[\frac{[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right]\right\} \tag{5}$$

Taking

$$u = \lambda\left\{\frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)}\right\}, \text{ we obtain } \frac{du}{d\lambda} = (c+1)\left\{\frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)}\right\}$$

while taking

$$v = 1 + \sum_{i=1}^{c}\left[\frac{[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right], \text{ we obtain } \frac{dv}{d\lambda} = \frac{1}{\lambda}\sum_{i=1}^{c}\left[\frac{i[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right]$$

DOI: 10.1002/atr

Using the quotient rule [25], we obtain

$$\frac{d\theta}{d\lambda} = 1 - \frac{1}{\left(P_0^{-1}\right)^2}\left\{1 + \sum_{i=1}^{c}\left[\frac{[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right]\right\}\left\{\frac{(c+1)[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)}\right\} -$$

$$\frac{\lambda}{\left(P_0^{-1}\right)^2}\left\{\frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)}\right\}\left\{\sum_{i=1}^{c}\left[\frac{[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right]\right\} \tag{6}$$

This equation can be simplified as

$$\frac{d\theta}{d\lambda} = 1 - \frac{\frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)}\left\{(c+1)\left(P_0^{-1}\right) - \sum_{i=1}^{c}\left[\frac{i[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right]\right\}}{\left(P_0^{-1}\right)^2} \tag{7}$$

A critical point of the throughput function θ happens when $\frac{d\theta}{d\lambda} = 0$. Thus, setting Equation (7) equals to zero and solving it, we obtain $\lambda = \lambda_{opt}$, which maximizes the throughput function. To confirm that the (λ_{opt}, θ_{opt}) point is a maximum point, we have to substitute $\lambda = \lambda_{opt}$ to $\frac{d^2\theta}{d\lambda^2}$, which is given by

$$\frac{d^2\theta}{d\lambda^2} = \frac{2\psi P_0^{-1}[c+1-\Phi] + \psi\left(P_0^{-1}\right)^2\left[P_0^{-1}\Omega - 2\left(P_0^{-1}-1\right)\Phi - c(c+1)\left(P_0^{-1}\right)^2\right]}{\left(P_0^{-1}\right)^3} \tag{8}$$

where

$$\psi = \frac{P_c}{\lambda P_0},\ \Phi = \sum_{i=1}^{c}\left[\frac{i[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right],\ \Omega = \sum_{i=1}^{c}\left[\frac{i^2[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)}\right],\ \forall\lambda > 0$$

If $\frac{d^2\theta}{d\lambda^2} < 0$ then the (λ_{opt}, θ_{opt}) point is a maximum point. Otherwise, it is a minimum point.

3.2. Computing the optimal arrival rate using numerical analysis methods

We can observe that the throughput graphs for all considered corridors follow the same pattern. For $\lambda \in [0, \lambda_{opt})$, the slope of the throughput function, $\frac{d\theta}{d\lambda} = 1$. At the optimal arrival rate, that is, $\lambda = \lambda_{\text{opt}}$, the slope of the throughput function, $\frac{dq}{d\lambda} = 0$. After this optimal point, $\frac{d\theta}{d\lambda}$ has big negative values and finally decreases to -1 when λ reaches a certain arrival rate. Figure 2 shows the graphs of $\frac{d\theta}{d\lambda}$ versus arrival rates for the considered corridors while Figure 3 shows the graphs of the second derivative of the throughputs versus arrival rates for the corridors.

We have to solve $P(\lambda) = \frac{d\theta}{d\lambda} = 0$ to get $\lambda = \lambda_{opt}$. Because its analytical solution is impossible, we used numerical analysis methods [26, 27]. Examples of numerical methods include *Bisection*, *Regula-Falsi*, *Secant*, *Newton–Raphson* (whose computational methods in finding the optimal arrival rate of a corridor are discussed in Appendix A), and *Iterative* methods. We found that the *Iterative* method is the best numerical method in terms of the number of iterations for searching λ_{opt} of all of the considered corridors.

The *Iterative* method requires us to change $P(\lambda) = \frac{d\theta}{d\lambda}$ to $\lambda = \varphi(\lambda)$. $\lambda_{\text{new}} = \varphi(\lambda_{\text{old}})$ is repeated until $|\lambda_{\text{new}} - \lambda_{\text{old}}| < \varepsilon$. Thus, the new equation for $P(\lambda) = \frac{d\theta}{d\lambda}$ is

DOI: 10.1002/atr

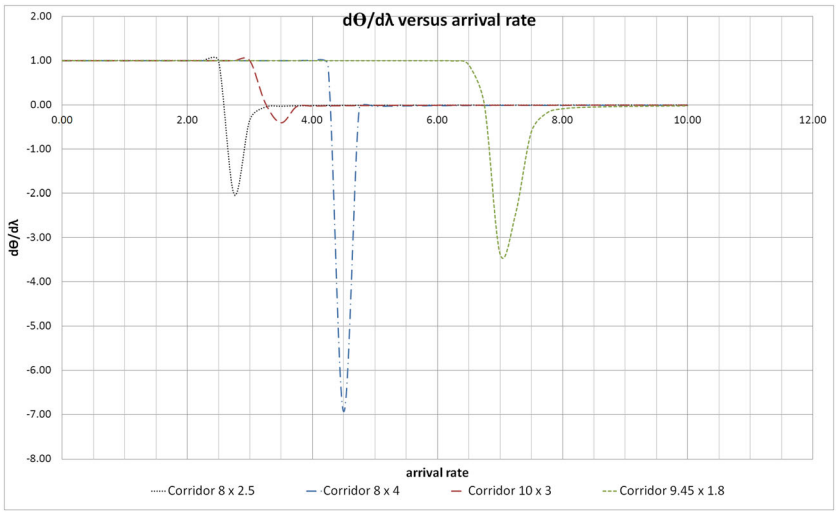


Figure 2. First derivative of throughput versus arrival rate.

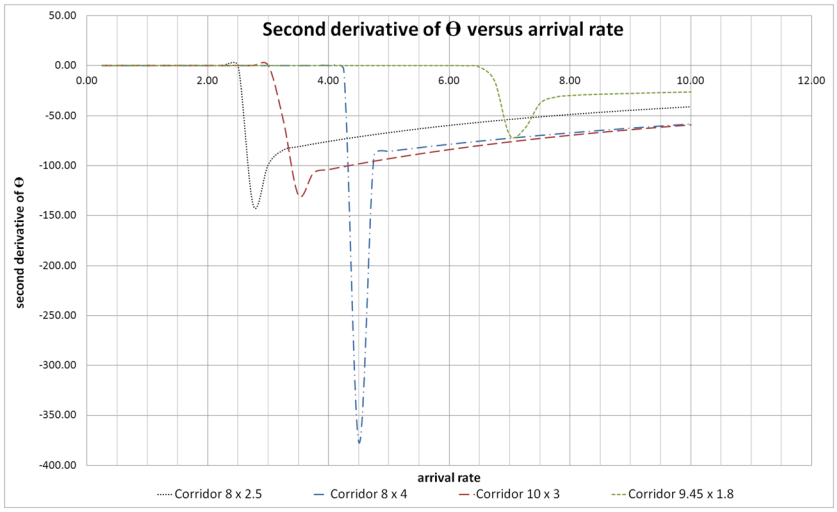


Figure 3. Second derivative of throughput versus arrival rate.

$$\lambda = \left(\frac{\left(P_0^{-1}\right)^2}{\frac{[\lambda E(S)]^c}{c!f(c)f(c-1)\ldots f(2)f(1)} \left\{ (c+1)\left(P_0^{-1}\right) - \sum_{i=1}^{c} \left[\frac{i[\lambda E(S)]^i}{i!f(i)f(i-1)\ldots f(2)f(1)} \right] \right\}} \right)^{1/c}$$

Table I shows the effect of arrival rates to the corridors' performance measures. For each corridor, three arrival rates are considered, that is, an arrival rate lower than its optimal arrival rate, its optimal arrival rate, and an arrival rate higher than its optimal arrival rate. The table also shows the number of iterations required to search λ_{opt} of the corridors based on the *Iterative*, *Bisection*, and *Regula-Falsi* methods.

From Table I, we can deduct four significant patterns of the effect of arrival rates to the corridors' throughputs. First, any arrival rates less than the optimal arrival rates flow pedestrians smoothly

DOI: 10.1002/atr

Table I. Samples of arrivals rates that optimize corridors.

Corridor	λ	⊖	P(c)	E(N)	E(T)
5.0 × 4.0	2.0000	2.0000	0.0000	7.8197	3.9098
	[1]4.3173	4.2573	0.0139	28.9942	6.8104
	8.0000	3.1198	0.6100	99.3507	31.8448
8.0 × 2.5	2.0000	2.0000	0.0000	14.4875	7.2438
	[2]2.6983	2.6608	0.0139	28.9942	10.8966
	4.0000	1.9593	0.5102	99.0114	50.5337
8.0 × 4.0	3.0000	3.0000	0.0000	20.9090	6.9697
	[3]4.3378	4.3012	0.0085	42.7223	9.9327
	8.0000	3.1045	0.6119	159.3598	51.3322
4.0 × 8.0	3.0000	3.0000	0.0000	8.9150	2.9717
	[4]8.6757	8.6023	0.0085	42.7243	4.9666
	16.0000	6.2090	0.6119	159.3598	25.6661
10.0 × 3.0	2.5000	2.5000	0.0000	22.8638	9.1455
	[5]3.2513	3.2219	0.0090	40.3966	12.5380
	6.0000	2.3296	0.6117	149.3588	64.1128
9.45 × 1.80 (ATD = 2.7000 m)	4.5000	4.5000	0.0000	10.5286	2.3397
	[6]6.7516	6.6423	0.0162	25.2361	3.7993
	10.0000	4.8753	0.5125	85.0150	17.4380
n of iterations 1 = 17*, 30†, 130‡; 2 = 16*, 30†, 506‡; 3 = 15*, 30†, 218‡; 4 = 16*, 30†, 23‡; 5 = 15*, 30†, 476‡; 6 = 18*, 30†, 20‡					

*_Interative_ method based on λ = 1.0 ped/second and error tolerance = 0.00000001.
†_Bisection_ method based on λ_1 = 1.0 ped/second and λ_2 = 10.0 peds/second and error tolerance = 0.00000001.
‡_Regula-Falsi_ method based on λ_1 = 1.0 ped/second and λ_2 = 10.0 peds/second and error tolerance = 0.00000001.
ATD, Average Travelling Distance.

without any blocking. However, the rates cause little throughputs at the end. Second, at the optimal arrival rates, the throughputs are maximized with some pedestrians that are blocked from entering the corridors. Third, any arrival rates greater than the optimal arrival rates cause high blocking, and this situation decreases the corridors' throughputs at the end. Fourth, for corridors that have the same capacities (e.g., corridors 8 × 4 m and 4 × 8 m), the lengths determine their throughputs; that is, the corridor with a shorter length has a higher throughput.

4. TRAFFIC MANAGEMENT TOOL

4.1. Object-oriented M/G/C/C framework

A complex topological network typically consists of series, merging, and splitting corridors. These corridors have their own lengths, widths, capacities, arrival rates, optimal arrival rates, and so on. To model the network's structure and evaluate its performance, we have designed and constructed an *M/G/C/C* framework, which is mainly based on an OOP approach. Figure 4 shows the structure of our *M/G/C/C* framework.

Our framework consists of two classes. The first class is a *Corridor* class that is used to create corridor objects to represent available corridors in a topological network. The second class is a *Network* class whose object is used to manage the execution of the network. This includes reporting its current performance (in terms of corridors' performances), creating its network flow programming script, sending the script to the LINGO optimization software, retrieving its source corridors' optimal arrival rates, re-measuring its optimal performance based on the optimal arrival rates, and finally reporting the maximum throughput that can be achieved. All of these functionalities are available through their class methods as shown in Figure 5.

There are three types of methods provided in the *Corridor* class. All these methods are named based on their functionality. The first type is to initialize a corridor's identification and physical dimension, for example, ID, length, width, average, and traveling distance. The second type is to set or report its performance, for example, arrival rate, throughput, blocking probability, expected service time, expected number of occupants, and optimal arrival rate. The optimal arrival rate of each available

DOI: 10.1002/atr

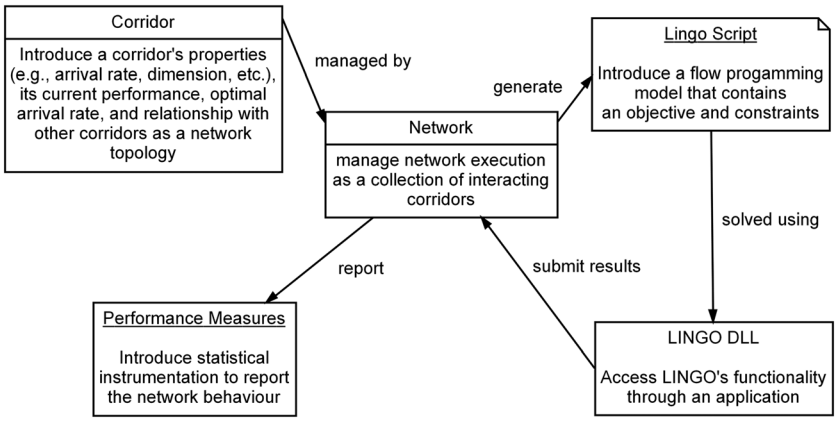


Figure 4. *M*/*G*/*C*/*C* framework. DLL, Dynamic Link Library.

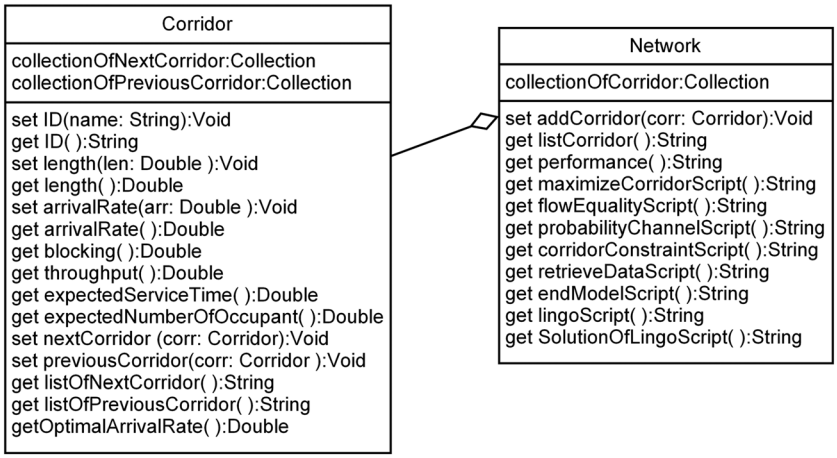


Figure 5. Class diagram for the *M*/*G*/*C*/*C* network.

corridor can be obtained through the *getOptimalArrivalRate( )* method that implements the *Iterative* numerical method. The ID and optimal arrival rate are used as a corridor's identification and capacity in a network flow programming model, respectively. The performance of the corridor based on this optimal arrival rate is generated using the *getOptimalPerformance( )* method. Note that all of these functions should be used after the instantiation and initialization of the corridor. The third type is to define how this corridor is linked to other corridors to form a topological network. The relevant methods are *nextCorridor*(*Optional prob*:*Double*, *corr*:*Corridor*) and *previousCorridor*(*corr*:*Corridor*). All corridors that have connection with the corridor can then be accessed through the *listOfNextCorridor( )* and *listOfPreviousCorridor( )* methods.

The *nextCorridor*(*Optional probability*:*Double*, *nextCorridor*: *Corridor*) is called by a corridor to set the *nextCorridor* object as its downstream corridor, which is then stored in its *collectionOfNextCorridor* variable. Upon calling this method, the *previousCorridor*(*prevCorridor*: *Corridor*) method is activated to set the corridor as the upstream corridor of the *previousCorridor* object, which is then stored in its *collectionOfPreviousCorridor* variable. For example, *Corridor1*. *nextCorridor* = *Corridor2* links *Corridor1* and *Corridor2* objects. This statement implies that *Corridor2* is set as a downstream corridor of *Corridor1*. At the same time, the program automatically sets *Corridor1* as an upstream corridor of *Corridor2*. The *probability* parameter in the method is an optional parameter. It specifies how the throughput of an upstream corridor is channeled to its downstream corridor. If unspecified, the probability is automatically calculated based on the number of its downstream corridors. For example, if a corridor has two downstream corridors, the probability is then set to 0.5. Otherwise, the statement of *Corridor1.nextCorridor*(*0.4*) = *Corridor2* means that 40% of the

DOI: 10.1002/atr

throughput of *Corridor1* is channeled to *Corridor2*. Using these strategies, we can construct a topological network for series, splitting, and merging networks as follows:

Series topology:

```
Corridor1.nextCorridor = Corridor2
Corridor2.nextCorridor = Corridor3
```

Splitting topology:

```
Corridor1.nextCorridor (0.4) = Corridor2
Corridor1.nextCorridor (0.6) = Corridor3
```

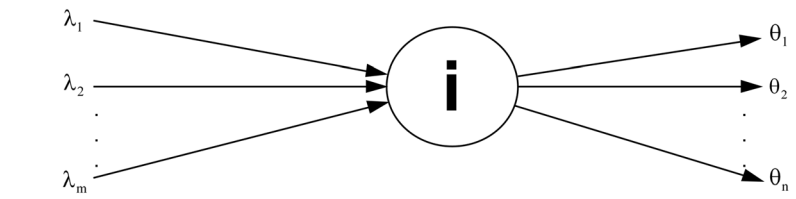
Merging topology:

```
Corridor1.nextCorridor = Corridor3
Corridor2.nextCorridor = Corridor3
```

Storing a corridor's upstream and downstream corridors also allows us to differentiate between source, intermediate, and exiting corridors. Source corridors are corridors that do not have any upstream corridors unless arrival rates are imposed to the corridors. Intermediate corridors are corridors that have both upstream and downstream corridors. Exiting corridors are corridors that only have upstream corridors. Using this strategy, we can calculate arrival rates to any intermediate or exiting corridors as illustrated in Figure 6.

The arrival rate of a corridor is how much the throughputs of its upstream corridors are channeled to this corridor. For example, consider a corridor with three upstream corridors. To calculate its arrival rate, we have to find the throughputs of the upstream corridors and multiply them with their routing probabilities to this corridor. The arrival rate of each of the upstream corridors is again based on how much the throughputs of its upstream corridors are channeled to the corridor. The process is repeated until a source corridor is found, that is, a corridor that has no upstream corridors. Thus, this process should be recursive. The recursive function guarantees that the arrival rate of a relevant corridor is calculated correctly. The algorithm for calculating the arrival rate of a corridor is shown in Table II. In the algorithm, V refers to a set of nodes (corridors), A is a set of arcs connecting corridors, and G(V, A) is a graph that defines the topological network.

To manage the execution of corridors, we have to insert them to a *Network* object. The network object must first be created and fed with corridors using the *addCorridor*(*CorridorName*:*Corridor*) method. For example, the statement *myNetwork.addCorridor* = *Corridor1* inserts *Corridor1* to the *myNetwork* object. Note that this method should only be used after we have created *Corridor1*. Other methods in the *Network* class will be discussed in the next section. We used Visual Basic [28] as an implementation language for the framework.



$\lambda_i = \sum_{j=1}^{m} P_{ji}\theta_j$ $\quad$ $\theta_i = \sum_{k=1}^{n} P_{ik}\theta_i$

m = # of upstream nodes of node i $\quad$ n = # of downstream nodes of node i

θ_j = throughput of node j $\quad$ θ_i = throughput of node i

P_{ji} = routing probability from node j to node i $\quad$ P_{ik} = routing probability from node i to node k

Figure 6. Relationship between arrival rate and throughput of a node.

DOI: 10.1002/atr

Table II. Throughput algorithm.

```
read G(V, A)
read routing probabilities P_ij, ∀ i, j ∈ A
read arrival rates, λ_i, ∀ i ∈ V
for ∀ i ∈ V do
  read its upstream nodes U_k where P_ki ≠ 0
  if U_k > 0 then
    /*traverse until its source nodes*/
    for ∀ j ∈ U_k do
      /* λ_i in the right hand side represents the node has its own arrival rate*/
      λ_i = θ_j × P_ji + λ_i
    end for
  end if
  /* arrive at a source node and calculate the node's throughput*/
  θ_i = λ_i × (1 - blocking_i)
end for
```

4.2. Optimizing a topological network total throughput

An *M*/*G*/*C*/*C* queuing network is typically used to evaluate the performance of evacuation strategies based on certain flow rates. The impacts of various flow rates to source corridors are documented, and the ideal evacuation rates are recommended to best flow pedestrians through a topological network. In order to maximize the network's throughput, we propose the combination of the *M*/*G*/*C*/*C* network and network flow programming approaches.

Implementing these approaches requires us to convert a network's structure to its equivalent network flow diagram. In this case, each corridor is represented using a node, while links between corridors are presented using edges. Because the network may have multiple sources (s_1, s_2, …, s_n) and sinks (t_1, t_2, …, t_n), we introduce a fictitious super-source S and a fictitious super-sink T. The flow capacities from the super-source S into each available source (S, s_i) and the flow capacities from each available sink into the super-sink T (t_j, T) are unlimited, that is, $c(S, s_i) = c(t_j, T) = \infty$.

The network flow programming has the following characteristics:

NOTATION

i = an index for origin node (vertex) i
j = an index for destination node j

DECISION VARIABLE

$x_{i \to j}$ = the flow from origin node i to destination node j

MATHEMATICAL FORMULATION

Maximize $x_{T \to S}$	$x_{T \to S}$ represents the flow from super-sink node T back to super-source node S
subject to:	
$\sum_j x_{i \to j} - (\sum_i x_{j \to i} - x_{T \to S}) = 0$	outflow of node i must equal to its inflow for node $i = S$
$\sum_j x_{i \to j} - (\sum_i x_{j \to i} - x_{S \to i}) = 0$	outflow of source node i must equal to its inflow for every node $i = s_1, s_2, \ldots, s_n$
$\sum_j x_{i \to j} - \sum_i x_{j \to i} = 0$	outflow of node i must equal to its inflow for every node $i \neq s, t$

DOI: 10.1002/atr

$(\sum_j x_{i\to j} + x_{i\to T}) - \sum_i x_{j\to i} = 0$	outflow of sink node i must equal to its inflow for every node $i = t_1, t_2, \ldots, t_n$
$(\sum_j x_{i\to j} + x_{T\to S}) - \sum_i x_{j\to i} = 0$	outflow of node i must equal to its inflow for node $i = T$
$x_{i\to j} \le u_{i\to j}$	flow capacity for every edge $i \to j$
$x_{i\to j} \ge l_{i\to j}$	minimum flow for every edge $i \to j$

Note that the summation of arrival rates to a corridor must be smaller or equal to its optimal arrival rate; that is, $\sum_i x_{i\to j} \le c_j$,where c_j is the optimal arrival rate to corridor j.

The performance of traffic flows through a single node (i.e., corridor) can be measured using the discussed analytical model. It is however impossible to construct an analytical model for measuring the performance of entity flows through a network of nodes because any arrivals or departures of entities at any particular nodes require the adjustment of service rates in the whole network. Thus, any saturated downstream nodes due to heavy traffic or low capacity will affect the performance of their upstream nodes. In order to comprehensively compute its performance, simulation or approximation methods such as the Generalized Expansion Method (GEM) [29, 30] can be used. The essence of the GEM is to add an artificial holding node prior to each node in the network to register entities blocked from joining the node when it is in full capacity. By doing this, the network is now converted to an equivalent Jackson network [31, 32] where each node can now be decomposed and analyzed separately. A detailed discussion on how the GEM is used to analyze traffic flows in an *M*/*G*/*C*/*C* network and how well it approximates the actual performance of the network can be found in the literature (e.g., [7, 8, 11, 23]).

Our network flow programming approach is also based on the stability condition as in a Jackson network. This condition requires us to ensure that there is no blockage between the links of nodes in a finite capacity topological network. To achieve this, our application software first analyzes the network by relaxing the finite capacity of each node and modeling them as *M*/*G*/∞ queues so that its service time is drawn independently of the service time in other nodes. This technique enables us to decompose each node and analyze them separately. For each node, our software then searches its optimal arrival rate maximizing its throughput while minimizing its blocking probability.

The minimal blocking ensures that entities do not overflow each of the state-dependent queues in the whole network. The optimal arrival rates are then used in the network flow programming model to search the optimal arrival rates of source nodes and other nodes (when they are considered as a network) subject to the total flow out of each node that is equal to its total flow in, and the total arrival rate to the node must be smaller or equal to its optimal arrival. This is the main reason the network flow programming model does not contain any terms related to congestion because the congestion has been accommodated during the searching process of the optimal arrival rate for each node. Solving the model will give the optimal arrival rates of source corridors that guarantee the flow to each downstream node that is smaller than its optimal flow and will ensure that the number of entities in all *M*/*G*/*C*/*C* queues is below the threshold value for their blocking probabilities so that no entities will be lost during their flows. These optimal arrival rates of source nodes are then fed to the network to assess the real throughput and blocking probability based on the algorithm in Table II.

Our purpose is to develop a software application that automatically creates and solves a *network flow programming* script for any *M*/*G*/*C*/*C* networks structured by users. For this, we equip the *Network* class with relevant LINGO commands (Appendix B) to establish the network's objective function and constraints. All of these commands are private (which are accessible within the *Network* class) and can only be accessed through our *lingoScript( )* method. Thus, the *lingoScript( )* generates a complete network flow programming script based on the LINGO syntax. To report the performance of each available corridor based on the initial and optimal arrival rates, we provide the *performance( )* method. This method forces all of the corridor objects stored in the *collectionOfCorridor* variable to calculate their throughputs, blocking probabilities, expected number of occupants and the expected service times.

Users' tasks are only to create a topological network. Upon completing this, our application dynamically generates its network flow script, runs relevant LINGO commands in the background, and retrieves the optimal arrival rates of source corridors. These values are then used to calculate the network's new total throughput. Our application is linked to the LINGO software using its DLL. The DLL provides access to all LINGO's major features using command scripts, for example, to create

DOI: 10.1002/atr

and delete a LINGO environment object, to create and open a LINGO's log file, to map LINGO's memory pointers to an application, and to execute a model and solves it. Some of the important commands used to link our application and LINGO are found in Appendix B.

5. RESULTS AND ANALYSIS

5.1. Imaginary topological network

This section shows how our application measures the performance of a selected topological network, generates its network flow programming script, and analyzes its optimal performance. We consider a network structure as in Figure 7. The dimension of each available corridor and its relationship with the throughputs of other corridors are displayed in Table III.

The network consists of series, merging, and splitting topologies. For example, a series topology is formed by corridors 2, 4, and 6. Splitting topologies are formed by corridor 1 (splits to corridors 3 and 5), corridor 2 (splits to corridors 3 and 4), corridor 5 (splits to corridors 7 and 8), and corridor 6 (splits to corridors 7 and 8). Merging topologies are formed by corridor 3 (merged by corridors 1 and 2), corridor 7 (merged by corridors 3, 5, and 6), and corridor 8 (merged by corridors 5, 6, and 7). To construct the network, we first define the network object via

```
Set myNetwork = New Network
```

We then instantiate all available corridors and specify their lengths and widths. For the source corridors, we have to initialize their arrival rates. For example, corridor 1 (which refers to Corridor1 in our application) is defined and initialized using this code:

```
Set Corridor1 = New Corridor
Corridor1.ID = "Corr1"
Corridor1.Length = 9
```

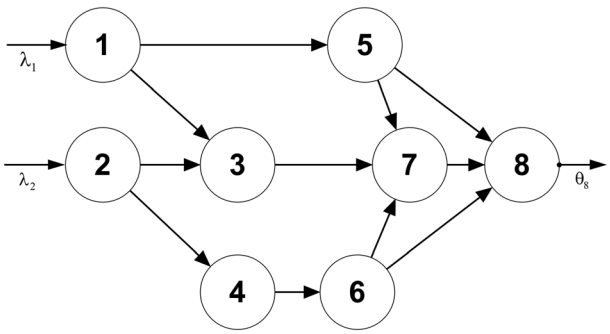


Figure 7. A topological network structure.

Table III. Corridor dimension, arrival rate, and throughput.

Type	Corridor	Length	Width	Arrival rate	Throughput
Source	1	9.0	3.5	λ_1	θ_1
	2	8.0	2.0	λ_2	θ_2
Intermediate	3	10.0	3.0	$\lambda_3 = \frac{\theta_1}{2} + \frac{\theta_2}{2}$	θ_3
	4	7.0	4.0	$\lambda_4 = \frac{\theta_2}{2}$	θ_4
	5	6.0	4.5	$\lambda_5 = \frac{\theta_1}{2}$	θ_5
	6	5.0	4.0	$\lambda_6 = \theta_4$	θ_6
	7	8.0	4.0	$\lambda_7 = \theta_3 + \frac{\theta_5}{2} + \frac{\theta_6}{2}$	θ_7
Exiting	8	8.0	2.5	$\lambda_8 = \frac{\theta_5}{2} + \frac{\theta_6}{2} + \theta_7$	θ_8

DOI: 10.1002/atr

```
Corridor1.Width = 3.5
Corridor1.AverageLength = 9
Corridor1.Lambda = 2
myNetwork.addCorridor = Corridor1
```

Note that the ID property, that is, *Corr1* is used to represent the corridor name in the network's flow programming script. The same procedure must be followed to create other corridors. However, only source corridors or intermediate corridors that have additional arrival rates need to specify the *Lambda* property values. To create links between the available corridors, the following code must be written:

```
Corridor1.nextCorridor(0.5) = Corridor3
Corridor1.nextCorridor(0.5) = Corridor5
Corridor2.nextCorridor(0.5) = Corridor3
Corridor2.nextCorridor(0.5) = Corridor4
Corridor3.nextCorridor(1) = Corridor7
Corridor4.nextCorridor(1) = Corridor6
Corridor5.nextCorridor(0.5) = Corridor7
Corridor5.nextCorridor(0.5) = Corridor8
Corridor6.nextCorridor(0.5) = Corridor7
Corridor6.nextCorridor(0.5) = Corridor8
Corridor7.nextCorridor(1) = Corridor8
```

Note that the splitting probabilities for all corridors are assumed to be 0.5. These values can be changed to any values as long as the total splitting probabilities of each node is 1. Once the code has been written, users can now run the application. The application then evaluates the performance of each corridor, generates the LINGO script based on the established network structure, sends it to the LINGO software, retrieves the optimal arrival rates of source corridors, and reports the performance for each corridor based on the optimal arrival rates. Table IV shows the results reported by our application.

We can confirm that the throughputs of the upstream corridors (based on their splitting probabilities) are the arrival rates of their downstream corridors. For example, the arrival rate of intermediate Corridor 3 (i.e., 2.9364 peds/second; line 5) is the summation of half of the throughputs of Corridor 1 (i.e., 3.757 peds/second; line 3) and Corridor 2 (i.e., 2.1159 peds/second; line 4). The arrival rate of Corridor 8 (i.e., 5.6707 peds/second; line 10) is half of the throughput of Corridor 5 (i.e., 1.8785 peds/second; line 7), half of the throughput of Corridor 6 (i.e., 1.0579 peds/second; line 8), and the full throughput of Corridor 7 (i.e., 4.2025 peds/second; line 9). Note also that we purposely used the optimal arrival rates of the source corridors; that is, Corridor 1 and Corridor 2 as their initial arrival rates because we wish to test if the rates maximize the total throughput of the network. Using these initial arrival rates, the total throughput of the network is 1.9466 peds/second (line 11), that is, the throughput of exiting Corridor 8. To search the optimal rates of the source corridors, the application automatically generates the network flow programming script for the network structure (lines 14 to 58).

The application first writes the network's objective function, that is, to maximize the pedestrian flow from super-sink node *T* back to super-source node *S*. Because this network only has two source corridors, its *XT_S* is the summation of the arrival rates of Corridor 1 and Corridor 2 (line 18). Second, the application guarantees that all flow out of each node is equal to its flow in. For example, the flow from Corridors 1 and 2 to Corridor 3 must equal to its flow out, that is, the flow from Corridor 3 to Corridor 7 (line 23). Third, the flow out of the super-sink node *T* must equal to its flow in. Because the considered network only has one exiting corridor, that is, Corridor 8, the throughput of Corridor 8 must equal to *XT_S* (line 18). Fourth, the upper limit of arrival rate of a node must be smaller or equal to its optimal arrival rate to guarantee no blocking exists. For example, the arrival rate from Corridor 1 and Corridor 2 to Corridor 3 must be smaller or equal to its optimal arrival rate, that is, 3.2513 peds/second (line 36). Finally, the application ensures that the splitting probabilities of a node are satisfied. For example, the splitting probabilities of Corridor 2 to Corridors 3 and 4 are 0.5, respectively, (line 45). In order to capture the solutions generated by LINGO, the application maps the memory linkage with the

DOI: 10.1002/atr

Table IV. Application outputs

```
1                  Performance based on the current arrival rates:
2  Corridor    Lambda       Theta       Blocking        E(N)          E(T)
3  Corr1       3.7893    3.757       0.0085        41.9674       11.1706
4  Corr2       2.1541    2.1159      0.0177        24.5859       11.6196
5  Corr3       2.9364    2.9364      0             29.7747       10.1398
6  Corr4       1.0579    1.0579      0             5.3093        5.0185
7  Corr5       1.8785    1.8785      0             8.5107        4.5306
8  Corr6       1.0579    1.0579      0             3.8002        3.5921
9  Corr7       4.4046    4.2025      0.0459        59.5345       14.1666
10 Corr8       5.6707    1.9466      0.6567        99.4712       51.1009
11 Total throughput of the network: 1.9466
12
13 Network Flow Programming Model:
14 MODEL:
15           [R_OBJ] MAX = XT_S;
16
17           !Flow out of a super source node equals its flow in;
18           XT_S = XS_Corr1 + XS_Corr2;
19
20           !Flow out of each node equals its flow in;
21           XS_Corr1 = X_Corr1_Corr3 + X_Corr1_Corr5;
22           XS_Corr2 = X_Corr2_Corr3 + X_Corr2_Corr4;
23           X_Corr1_Corr3 + X_Corr2_Corr3 = X_Corr3_Corr7;
24           X_Corr2_Corr4 = X_Corr4_Corr6;
25           X_Corr1_Corr5 = X_Corr5_Corr7 + X_Corr5_Corr8;
26           X_Corr4_Corr6 = X_Corr6_Corr7 + X_Corr6_Corr8;
27           X_Corr3_Corr7 + X_Corr5_Corr7 + X_Corr6_Corr7 = X_Corr7_Corr8;
28           X_Corr5_Corr8 + X_Corr6_Corr8 + X_Corr7_Corr8 = X_Corr8_T;
29
30           !Flow out of a super sink node equals its flow in;
31           X_Corr8_T = XT_S;
32
33           !Maximum arrival rate for a node;
34           XS_Corr1 <= 3.7893;
35           XS_Corr2 <= 2.1541;
36           X_Corr1_Corr3 + X_Corr2_Corr3 <= 3.2513;
37           X_Corr2_Corr4 <= 4.3321;
38           X_Corr1_Corr5 <= 4.8719;
39           X_Corr4_Corr6 <= 4.3173;
40           X_Corr3_Corr7 + X_Corr5_Corr7 + X_Corr6_Corr7 <= 4.3378;
41           X_Corr5_Corr8 + X_Corr6_Corr8 + X_Corr7_Corr8 <= 2.6983;
42
43           !Channeling probabilities of a node;
44           0.5* X_Corr1_Corr3 = 0.5* X_Corr1_Corr5;
45           0.5* X_Corr2_Corr3 = 0.5* X_Corr2_Corr4;
46           0.5* X_Corr5_Corr7 = 0.5* X_Corr5_Corr8;
47           0.5* X_Corr6_Corr7 = 0.5* X_Corr6_Corr8;
48           XS_Corr1 = XS_Corr2;
49           DATA:
50                @POINTER(1) = @STATUS();
51                @POINTER(2) = R_OBJ;
52                @POINTER(3) = XS_Corr1;
53                @POINTER(4) = XS_Corr2;
54           END DATA
55 END
56 SET GLOBAL 1
57 GO
58 QUIT
59
60 SOLUTION FOR THE MODEL
61 Status: Globally Optimal
```

(Continues)

DOI: 10.1002/atr

Table IV. (Continued)

```
62      Objective value = 2.6983
63      XS_Corr1 = 1.34915
64      XS_Corr2 = 1.34915
65
66      Performance based on the optimal arrival rates:
67      Corridor      Lambda      Theta       Blocking      E(N)          E(T)
68      Corr1         1.3492      1.3492      0             9.0545        6.7113
69      Corr2         1.3492      1.3492      0             9.1763        6.8015
70      Corr3         1.3492      1.3492      0             10.3001       7.6345
71      Corr4         0.6746      0.6746      0             3.2894        4.8762
72      Corr5         0.6746      0.6746      0             2.8048        4.1579
73      Corr6         0.6746      0.6746      0             2.3528        3.4879
74      Corr7         2.0237      2.0237      0             12.6221       6.2371
75      Corr8         2.6983      2.6608      0.0139        28.9923       10.8959
76      Total throughput of the network: 2.6608
77
78      Elapsed run time: 401.58 seconds
```

LINGO solver (lines 50 to 53). The *@POINTER* commands capture the values of status, objective function, and arrival rates of source corridors.

The script is then transferred to the LINGO software through the DLL approach to find the optimal arrival rates of the source corridors (lines 61 to 64). The optimal arrival rates of the source Corridors 1 and 2 that maximize the overall throughput of the network are 1.34915 peds/second, respectively. Both of these values are then supplied back to the main application to measure the new performance of the available corridors (lines 67 to 75). The optimal throughput of the network is 2.6608 peds/second (line 76). Notice that the optimal arrival rates of the source corridors cause little or no blocking at intermediate and exiting corridors. This finding corresponds to the recommendation (e.g., [5, 30, 33]) that the population safety during an evacuation process can be achieved by minimizing congestion along route links. Notice that the arrival rates that only optimize the source corridors (i.e., 3.7893 and 2.1541 peds/second) do not guarantee the optimal throughput of the whole network.

Controlling pedestrian arrival rates to maximize a network's throughput is a desirable objective. However, the real issue is not necessarily how to control pedestrian flow rates, but how to efficiently and safely route the pedestrians through the network. For this case, we can request the application not to write the channeling probabilities of each node (e.g., lines 44 to 47). Removing these constraints will simply route the pedestrians accordingly. For this considered network, we should flow pedestrians with 1.34915 peds/second from Corridor 1 and Corridor 2 to Corridor 3, and then to Corridor 7, and lastly to Corridor 8 to achieve the same optimal throughput.

To test the overall performance of our application regarding its computational time in solving various numbers of nodes and edges, we consider a network structure as illustrated in Figure 8. The structure consists of n node layers, n source nodes, $^1/_2 n(n+1)$ nodes, and $n^2 - n$ edges. The central processing unit time consumed by our application for this various numbers of layers is shown in Table V. It is clear that the performance of the application is much influenced by the number of layers and edges in the network because much time must be allocated to calculate the optimal arrival rate of each node and to measure the performance of the node, which depends on the throughputs of its previous nodes, and this must be repeated until its last upstream nodes.

5.2. A real college university hall

We consider a real college university hall in Malaysia as a test platform of our application software. The hall is used as a place for important events, for example, convocation ceremony, big seminar,

DOI: 10.1002/atr

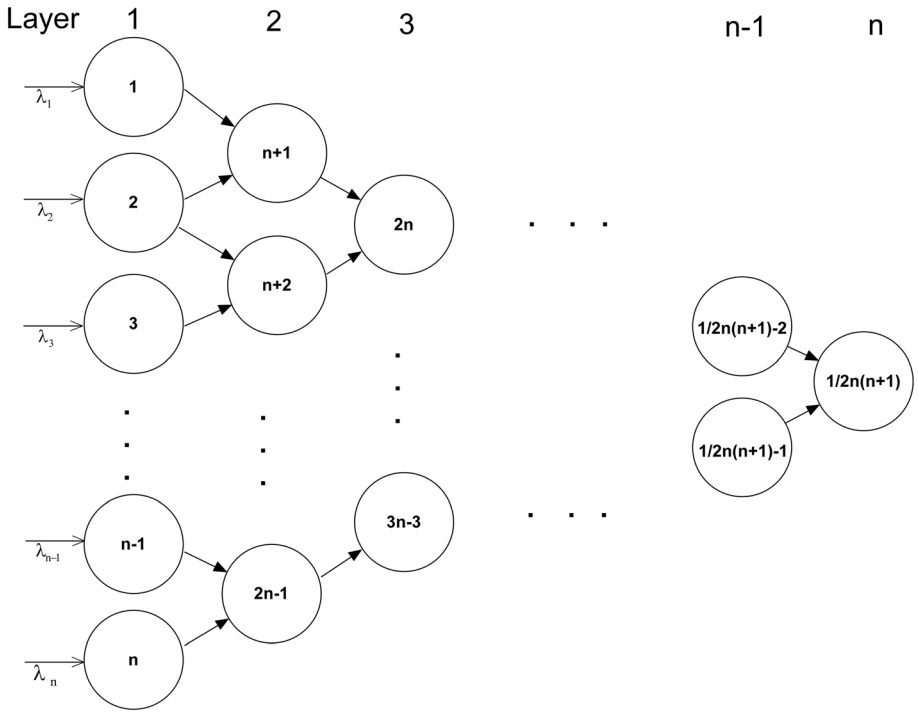


Figure 8. Performance measure structure.

Table V. Performance of the program

No. of layers n	No. of nodes $^1/_2 n(n+1)$	No. of edges $n^2 - n$	CPU time (second)
3	6	6	97.47
4	10	12	162.39
5	15	20	255.80
10	55	90	1424.88
12	78	156	3685.97

CPU, central processing unit.

and grand staff meeting. Its available corridors are presented in Figure 9. The numbers represent the corridors, the alphabets *A* to *I* represent the entrances to source corridors, *A*' is an exit door while *B*' and *C*' are exit corridors. Notice that the items 1 to 11 are not rooms, they are corridors. We intentionally partitioned each of them to measure their dimensions, show the sequence of networks, and ease measurement of their performance. The dimensions of the corridors in terms of lengths and widths and their capacities are shown in Table VI.

The hall consists of 13 corridors. Nine of them are source corridors, two are intermediate corridors, and the other two are exit corridors (Table VII). After entering the source corridors, pedestrians choose their nearest corridors to exit. For example, pedestrians near to entrance doors *D*, *E*, and *F* choose exit corridor *B*' because it is the nearest exit, while pedestrians near to entrance doors *G*, *H*, and *I* choose exit corridor *C*'. Other pedestrians near to entrance doors *A*, *B*, and *C* choose exit door *A*'. The relationships between the arrival rates and throughputs of the corridors when the facility is considered as a topological network are presented in Table VII.

The flow of occupants from corridor to corridor in the facility can be illustrated as in Figure 10. The facility only consists of a combination of series and merging networks. A series of networks is formed, for example, by corridors 1, 2, 3, 4, and 5. In this topology, the throughput of the corridor will be the arrival rate for its next corridor. Thus, the throughput of corridor 1 will be the arrival rate for corridor 2,

DOI: 10.1002/atr

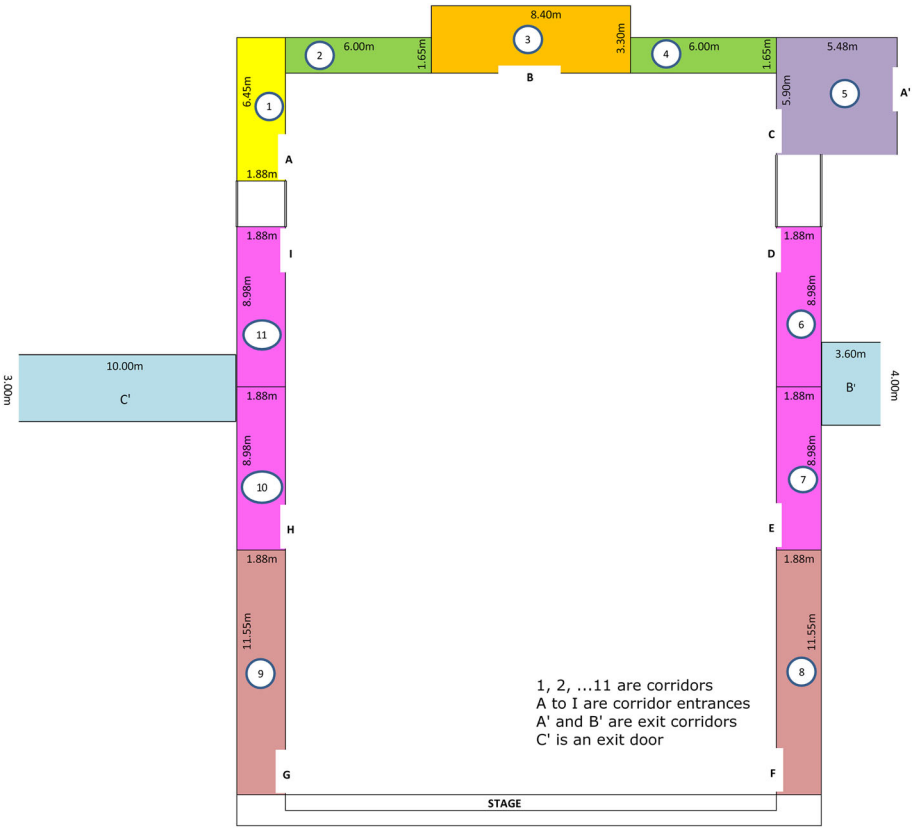


Figure 9. Hall structure.

Table VI. Dimensions (in meter) and capacities of the corridors.

Corridor	Length (l)	Width (w)	Capacity ($5lw$)
1	6.45	1.88	61
2	6.00	1.65	50
3	8.40	3.30	139
4	6.00	1.65	50
5	5.48	5.90	160
6	8.98	1.88	84
7	8.98	1.88	84
8	11.55	1.88	109
9	11.55	1.88	109
10	8.98	1.88	84
11	8.98	1.88	84
B′	3.60	4.00	72
C′	10.00	3.00	150

and its throughput will then be the arrival rate for corridor 3. A merging network is formed, for example, by corridors 6 and 7, which later merge to exit corridor *B*'. In this topology, the arrival rate for the merging corridor is the sum of the throughputs of its previous corridors. Thus, the arrival rate for corridor *B*' is the sum of the throughputs of corridors 6 and 7. The arrival rate of a relevant corridor is sometimes the sum of the throughput of its previous corridor and an arrival rate to the corridor. This case can be seen, for example, in corridor 3 where its arrival rate is the sum of the throughput of corridor 2 and the arrival rate of the entrance door *B*. All of the corridors must be instantiated and initialized its properties, and their links must then be set up in the application software. Notice that we assume that there are no occupants in any of the corridors during the initial state of the network. Any pre-existence of occupants will decrease the calculated throughput in our analysis.

DOI: 10.1002/atr

Table VII. Arrival rates and throughputs of corridors.

Corridor		Total λ	θ
Source corridor	1	λ_1	θ_1
	3	$\lambda_3 = \lambda_3 + \theta_2$	θ_3
	5	$\lambda_5 = \lambda_5 + \theta_4$	θ_5
	6	λ_6	θ_6
	7	$\lambda_7 = \lambda_7 + \theta_8$	θ_7
	8	λ_8	θ_8
	9	λ_9	θ_9
	10	$\lambda_{10} = \lambda_{10} + \theta_9$	θ_{10}
	11	λ_{11}	θ_{11}
Intermediate corridor	2	$\lambda_2 = \theta_1$	θ_2
	4	$\lambda_4 = \theta_3$	θ_4
Exit door	A'	$\lambda_{A'} = \theta_5$	$\theta_{A'}$
Exit corridor	B'	$\lambda_{B'} = \theta_6 + \theta_7$	$\theta_{B'}$
	C'	$\lambda_{C'} = \theta_{10} + \theta_{11}$	$\theta_{C'}$

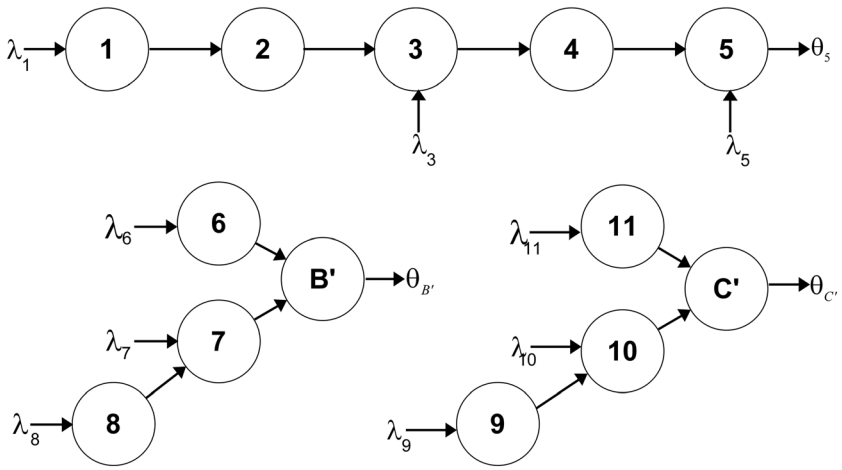


Figure 10. Hall network flow diagram.

To run our application, we set the default arrival rates of source corridors (i.e., corridors 1, 3, 5, 6, 7, 8, 9, 10, and 11) to 2 peds/second. The total throughput for the setting is 9.1641 peds/second. The arrival rates that maximize the total throughput of the hall, that is, 11.2495 peds/second, are shown in Table VIII. The processes of calculating the hall's performance based on the default arrival rates, writing and solving its Lingo script, and recalculating its optimal performance consumed 227.64 seconds.

We also conducted two other cases of arrival rates to compare the performance of the hall. In the first case, we set the arrival rates that maximize the throughputs of all of the source corridors, that is, $\lambda_1 = 1.9972$ peds/second, $\lambda_3 = 3.5565$ peds/second, $\lambda_5 = 4.1237$ peds/second, $\lambda_6 = \lambda_7 = 2.0188$ peds/second, $\lambda_8 = \lambda_9 = 2.0179$ peds/second, and $\lambda_{10} = \lambda_{11} = 2.0188$ peds/second. The total throughput of the hall for these settings is only 8.8532 peds/second. In the second case, we set the arrival rates that maximize the throughput of exit corridors, that is, $\lambda_5 = 4.1237$ peds/second, $\lambda_6 = \lambda_7 = 2.0188$ peds/second, $\lambda_{10} = \lambda_{11} = 2.0188$ peds/second, and let $\lambda_1 = \lambda_3 = 0.0000$ peds/second. The total throughput of the hall for this setting is 10.3893 peds/second.

To obtain the optimal throughput, we should flow the occupants in the upper hall directly to entrance door *C* with the evacuation rate of 4.1237 peds/second. They then travel through corridor 5 and exit the hall using exit door *A*'. We do not recommend they use entrance doors *A* and *B* because they have to travel quite far through relevant corridors before exiting through exit door *A*'. For occupants located in the middle of the hall, they have two choices to exit. If they are in the right side of the hall, they can then use entrance door *D*, travel through corridor 6, and then exit through corridor *C*', or they can

DOI: 10.1002/atr

Table VIII. Performance measures of the source corridors for the arrival rates that maximize the total throughput.

Performance based on the optimal arrival rates:

Corridor	Lambda	Theta	Blocking	E(N)	E(T)
1	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	0.00000	0.00000
5	4.1237	4.07036	0.01294	29.74026	7.30654
6	2.0188	1.98558	0.01645	25.26272	12.72308
7	2.0188	1.98558	0.01645	25.26272	12.72308
8	0.00000	0.00000	0.00000	0.00000	0.00000
9	0.00000	0.00000	0.00000	0.00000	0.00000
10	2.0188	1.98558	0.01645	25.26272	12.72308
11	1.2325	1.2325	0.00000	9.30706	7.55137
B′	3.97116	3.97015	0.00025	15.40113	3.87923
C′	3.21808	3.20884	0.00287	37.02719	11.53914

Total throughput of the network: 11.2493
Elapsed run time: 227.64 seconds

choose entrance door *E*, travel through corridor 7, and then exit through corridor *B*'. The best evacuation rates for both corridors are 2.0188 peds/second. We should block the entrance to corridor 8 because the pedestrians using this corridor have to compete with other occupants from corridors 6 and 7. For those near to the left side of the hall, they should use entrance door *H* and travel through corridor 10 with the rate of 2.0188 peds/second or use entrance door *I* and travel through corridor 11 with the rate of 1.2325 peds/second, and then exit through corridor *C*'. We should block the entrance to corridor 9 because the pedestrians using this corridor have to compete with other pedestrians from corridors 10 and 11. The total throughput of the network based on the arrival rates is 11.2493 peds/second; that is, the total throughputs of 4.07036 peds/second (exit door *A*'), 3.97015 peds/second (corridor *B*'), and 3.20884 peds/second (corridor *C*'). Notice that this is the optimal throughput that can be achieved if we control the arrival rates to source corridors based on the suggested values. In this case, the optimal arrival rates will maximize pedestrian flows (i.e., impose the pedestrians to enter the source corridors) as long as no blockage along the corridors. Any lower or higher arrival rates than the optimal values will decrease the throughput of the hall as shown in our previous cases. Based on the optimal arrival rates, the hall can approximately be emptied within 2 minutes and 13 seconds if we consider the number of occupants is 1500, that is, the maximum number of occupants who can occupy the hall.

The analysis of the results shows that people should always use the routes near to them in emergency cases while the routes are still safe. However, in a real-life evacuation scenario, the optimal arrival rates of pedestrians estimated by our application are difficult to implement. For example, we cannot control the entrance to corridor 5 to match 4.1237 peds/second. However, it is clear that any real evacuation rates that exceed the estimated optimal values will cause congestion and decrease the total throughput. Thus, it is important to have and train traffic control officers to direct and assist the pedestrians to enter the corridor as long as they will not contribute to the congestion. How should we know this? We could extend the estimated value to a bigger scale; that is, we should convert from 4.1237 to 41.237 peds/10 seconds. In other words, we can have a good flow during emergency if we ensure that at the end of each 10 seconds, 41 people have been allowed to enter corridor 5.

6. THE PRACTICAL SCOPE OF OUR METHODOLOGY

Our methodology derives the optimal rates of arrival sources maximizing pedestrian flows through a topological network. The flows are modeled using the founded *M*/*G*/*C*/*C* state dependent network, which formulates the interaction between the current walking speed and crowd density under a normal situation. There also exists other variations of state-dependent queuing systems, for example, *PH*/*PH*(*n*)/*C*/*C* [34] and *G*/*G*(*n*)/*C*/*C* [35]. In case of emergency situations, the formula may not consider real pedestrians' behavior because such behavior is unpredictable, very complex, and difficult to model

DOI: 10.1002/atr

[36]. Thus, our methodology only simulates the emergency situation based on the main assumption that pedestrians walk at their best speed to achieve the overall optimal throughput without the possibility of panic and confuse during the movement because these factors tend to make havoc in the entire system, delay their movement, and significantly decrease the throughput.

As mentioned earlier, it is quite difficult to control the exact optimal arrival rates to arrival sources. We may however control the arrival rate if we extend the estimated value to a bigger scale. For example, if the optimal arrival rate of an arrival source is 3 peds/second, then it should be converted to 30 peds/10 seconds; that is, in each 10-second interval time, there should be 30 pedestrians enter the arrival source. Following this rule, the level of congestion along the downstream evacuation network links can be minimized without exceeding the network capacity. Other suggested methods of controlling traffic flows include the development of a sensor that can compute the arrival rates through light array (e.g., [37, 38]).

7. CONCLUSION AND FUTURE WORK

This paper finds the optimal arrival rates of available corridors in a topological network based on an *M*/*G*/*C*/*C* approach. We then show how these values can be utilized to find the optimal arrival rates of source corridors that maximize the network's throughput using the network flow programming approach. In order to ease the network's construction and its performance evaluation, we employ OOP as the design and development of an *M*/*G*/*C*/*C* software application. The application automatically analyzes the current performance of the network, generates its network flow programming script, links it to the LINGO solver, and then retrieves the LINGO solution to report the network's optimal performance. Our application can be used as a decision tool to best flow people through a complex topological network that may consist of any combination of series, merging, and splitting topologies.

The instantiation of corridor objects and initialization of their property values still require much programming efforts. Thus, a better approach to make it more user-friendly, for example, by decreasing the use of code is crucial and offers some advantages. This includes the use grids to allow users to create corridors, to define their dimensions, and to establish their links with other corridors and the provision of a canvas to allow them to drag, drop, and link corridors to structure the network graphically. Such an extension will greatly help users analyze and optimize pedestrian flow in any considered network.

8. LIST OF SYMBOLS AND ABBREVIATIONS

$\frac{d\theta}{d\lambda}$	the first derivate of throughput
$\frac{d^2\theta}{d\lambda^2}$	the second derivate of throughput
λ_{opt}	the optimal arrival rate maximizing throughput where $\frac{d\theta}{d\lambda}(\lambda_{opt}) = 0$
θ_{opt}	the optimal throughput
λ_{new}	the current arrival rate value after a certain iteration
λ_{old}	the previous arrival rate value before the current iteration
ε	error tolerance; i.e., the absolute error in calculating λ_{opt}
λ_a	the lower value of an arrival rate in an interval
λ_b	the upper value of an arrival rate in an interval

ACKNOWLEDGEMENTS

This study was supported by the Research Acculturation Grant Scheme (RAGS) (account number 12685), Universiti Utara Malaysia. We wish to thank Universiti Utara Malaysia for the financial support. The funder had no role in this study design, data collection and analysis, decision to publish, or preparation of the manuscript.

DOI: 10.1002/atr

REFERENCES

1. Cruz FRB, Smith JM, Medeiros RO. An M/G/C/C state dependent network simulation model. *Computers and Operations Research* 2005; **32**(4): 919–941.
2. Kachroo P. *Pedestrian Dynamics: Mathematical Theory and Evacuation Control*. Taylor and Francis Group: Boca Raton, 2009.
3. Shende A, Kachroo P, Reddy CK, Singh M. Optimal control of pedestrian evacuation in a corridor. *Intelligent Transportation Systems Conference* 2007; **1**(2): 385–390.
4. Shende A, Singh MP, Kachroo P. Optimization-based feedback control for pedestrian evacuation from an exit corridor. *Transactions on Intelligent Transportation Systems* 2011; **12**(4): 1167–1176.
5. Stepanov A, Smith JM. Multi-objective evacuation routing in transportation networks. *European Journal of Operational Research* 2009; **198**(2): 435–446.
6. Yuhaski SJ Jr, Smith JM. Modeling circulation systems in buildings using state dependent queueing models. *Queueing Systems* 1989; **4**(4): 319–338.
7. Cruz FRB, Smith JM. Approximate analysis of M/G/c/c state-dependent queueing networks. *Computers & Operations Research* 2007; **34**(8): 2332–2344.
8. Mitchell DH, Smith JM. Topological network design of pedestrian networks. *Transportation Research Part B: Methodological* 2001; **35**(2): 107–135.
9. Kawsar LA, Ghani NA, Kamil AA, Mustafa A. Restricted pedestrian flow performance measures during egress from a complex facility. *World Academy of Science, Engineering and Technology* 2012; **67**: 399–404.
10. Kawsar LA, Ghani NA, Kamil AA, Mustafa A. Pedestrian performance measures of an M/G/C/C state dependent queueing network in emergency. *Journal of Applied Sciences* 2013; **13**(3): 437–443.
11. Jain R, Smith JM. Modeling vehicular traffic flow using M/G/C/C state dependent queueing models. *Transportation Science* 1997; **31**(4): 324–336.
12. Smith JM. Optimal workload allocation in closed queueing networks with state dependent queues. *Annals of Operations Research* 2013; 1–27.
13. Banks J, Carson JS, Nelson BL, Nicol DM. *Discrete-event System Simulation*, Pearson: New Jersey, 2010.
14. Garrido J. *Practical Process Simulation Using Object-oriented Techniques and C++*Artech House: Norwood, 1998.
15. Garrido JM. *Object-oriented Discrete-event Simulation with Java—A Practical Introduction*, Springer: New York, 2001.
16. Rossetti MD. *Simulation Modeling and Arena*, John Wiley & Sons: New Jersey, 2010.
17. Wainer GA, Mosterman PJ. *Discrete-event Modeling and Simulation: Theory and Applications (Computational Analysis, Synthesis, and Design of Dynamic Systems)*. CRC Press: Boca Raton, 2010.
18. Watkins K. *Discrete Event Simulation in C*. McGraw-Hill Companies: New York, 1994.
19. Khalid R, Nawawi MKM, Kawsar LA, Ghani NA, Kamil AA, Mustafa A. A discrete event simulation model for evaluating the performances of an M/G/C/C state dependent queuing system. *PLoS One* 2013; **8**(4): .
20. Wang H, Li J, Chen Q-Y, Ni D. Logistic modeling of the equilibrium speed-density relationship. *Transportation Research Part A: Policy and Practice* 2011; **45**(6): 554–566.
21. Tregenza P. *The Design of Interior Circulation*. Van Nostrand Reinhold: New York, 1976.
22. Cheah J, Master thesis, University of Massachusetts, 1990.
23. Cheah J, Smith JM. Generalized M/G/C/C state dependent queueing models and pedestrian traffic flows. *Queueing Systems* 1994; **15**(1-4): 365–386.
24. Fruin JJ. *Pedestrian Planning and Design*. Metropolitan Association of Urban Designers and Environmental Planners: New York, 1971.
25. Stewart J. *Multivariable Calculus*. Brooks/Cole Cengage Learning: Belmont, 2012.
26. Epperson JF. *An Introduction to Numerical Methods and Analysis*. John Wiley & Sons: Hoboken, 2007.
27. Faires JD, Burden R. *Numerical Methods*, Cengage Learning: Boston, 2013.
28. McGrath M. *Visual Basic in Easy Steps* (3rd edn). Easy Steps Limited: Southam, 2010.
29. Kerbache L, MacGregor Smith J. Asymptotic behavior of the expansion method for open finite queueing networks. *Computers & Operations Research* 1988; **15**(2): 157–169.
30. Kerbache L, Smith JM. The generalized expansion method for open finite queueing networks. *European Journal of Operational Research* 1987; **32**(3): 448–461.
31. Jackson JR. Networks of waiting lines. *Operations Research* 1957; **5**(4): 518–521.
32. Jackson JR. Jobshop-like queueing systems. *Management Science* 1963; **10**(1): 131–142.
33. Karbowicz CJ, Smith JM. K-shortest paths routing heuristic for stochastic network evacuation models. *Engineering Optimization* 1984; **7**(4): 253–279.
34. Hu L, Jiang Y, Zhu J. Chen Y. A PH/PH(n)/C/C state-dependent queuing model for metro station corridor width design. *European Journal of Operational Research* 2015; **240**(1): 109–126.
35. Jiang Y, Zhu J, Hu L, Lin X. Khattak A. A G/G(n)/C/C state-dependent simulation model for metro station corridor width design. *Journal of Advanced Transportation* 2015. DOI:10.1002/atr.1315.
36. Peacock RD, Kuligowski ED, Averill JD. *Pedestrian and Evacuation Dynamics*. Springer: New York, 2011.
37. Shende A, PhD Thesis, Virginia Polytechnic Institute and State University, 2008.
38. Shende A, Singh MP, Kachroo P. Optimal feedback flow rates for pedestrian evacuation in a network of corridors. *Transactions on Intelligent Transportation Systems* 2013; **14**(3): 1053–1066.

APPENDIX 1: COMPUTATIONAL METHODS FOR FINDING THE OPTIMAL ARRIVAL RATE.

The *Bisection* method searches the value of $\lambda_{opt} \in [\lambda_a, \lambda_b]$, which satisfies $P(\lambda_{opt}) = \frac{d\theta}{d\lambda} = 0$ if $P(\lambda_a) < 0$ and $P(\lambda_b) > 0$. A new value of $\lambda_{\text{new}} = \frac{\lambda_a + \lambda_b}{2}$ replaces λ_a if $P(\lambda_{\text{new}}) < 0$ or replaces λ_b if $P(\lambda_{\text{new}}) > 0$. This process is repeated until $|\lambda_b - \lambda_a| < \varepsilon$, where ε is an error tolerance. Thus, every iteration cuts the length of the range in half. After n number of iterations, the range is only cut to $|\lambda_b - \lambda_a|/2^n$. As a result, a lot of iterations are required for getting a good solution of $\lambda = \lambda_{opt}$ with a very small error. For all of the considered corridors, 30 iterations were required for getting their optimal arrival rates with $\varepsilon = 1 \times 10^{-8}$.

The *Regula-Falsi* method is an alternative to the *Bisection* method. Instead of cutting the range length to half for every iteration, this method expedites the convergence process to $\lambda = \lambda_{opt}$ by drawing a straight line that connects the points of $(\lambda_a, P(\lambda_a))$ and $(\lambda_b, P(\lambda_b))$ with $P(\lambda_a) < 0$ and $P(\lambda_b) > 0$ to find the point where the line crosses the x-axis, that is, $\lambda_{\text{new}} = \lambda_a - P(\lambda_a)\frac{\lambda_b - \lambda_a}{P(\lambda_b) - P(\lambda_a)}$. If $P(\lambda_{\text{new}}) < 0$, λ_{new} then replaces λ_a. Otherwise, it replaces λ_b. The process is repeated until $|\lambda_a - \lambda_b| < \varepsilon$. The iteration number to obtain the optimal arrival rate depends on the initial values of λ_a and λ_b. The closer both of the values to $\lambda = \lambda_{opt}$, the lesser number of iterations required to converge to $\lambda = \lambda_{opt}$. For example, consider corridor 8×2.5 m, where λ_{opt} is located between 2 and 3 peds/second (Figure 2). Setting these values for λ_a and λ_b, respectively, the number of iterations required for obtaining λ_{opt} is only 25. However, setting $\lambda_a = 1$ ped/second and $\lambda_a = 7$ peds/second required 178 iterations, while setting $\lambda_a = 1$ ped/second and $\lambda_a = 10$ peds/second required 506 iterations to obtain λ_{opt}. The large number of iterations is required in both cases because λ_{new} is almost calculated on one side of λ_{opt}, which makes the convergence process relatively slow.

The *Secant* method attempts to avoid the one-sided calculation as happened in the *Regula-Falsi* method. The default values of λ_a and λ_b are still required, but its range does not need to contain λ_{opt}. The new solution for every iteration is given by $\lambda_{n+1} = \lambda_n - P(\lambda_n)\frac{\lambda_n - \lambda_{n-1}}{P(\lambda_n) - P(\lambda_{n-1})}$. While both *Bisection* and *Regula-Falsi* methods confirm to find λ_{opt} (because it keeps bracketing λ_{opt}), the *Secant* method sometimes fail to find λ_{opt}, which does exist. For example, if we set $\lambda_a = a$ and $\lambda_b = b$ and λ_{opt} is greater than b, we will then get overflow because the slopes of the throughput graph at both of the values are the same, that is, 1. Thus, $P(\lambda_a) - P(\lambda_b) = 0$ (Figure 2). If the range of both of the initial values covers the λ_{opt} but the slopes of the throughput graph at the two points are quite perpendicular to x-axis, λ_{new} is then getting far and far away from λ_{opt}. This situation can be observed when setting $\lambda_a = 1$ and $\lambda_a = 8$ for corridor 8×2.5 m as in Table A.I. It shows that for every iteration, the new value of λ does not converge to λ_{opt}. We only manage to get λ_{opt} if we bracket λ_{opt} before it reaches the flat function. Thus, this method should not be implemented in the congestion model to find its λ_{opt}.

Table A.1 Iterations of the *Secant* method for corridor 8.0×2.5

Iteration	x_0	x_1	$P(x_0)$	$P(x_1)$	x_2	$P(x_2)$
1	1.00000	8.00000	1.00000	−0.00149	7.98960	−0.00149
2	8.00000	7.98960	−0.00149	−0.00149	10.96514	−0.00066
3	7.98960	10.96514	−0.00149	−0.00066	13.34260	−0.00041
4	10.96514	13.34260	−0.00066	−0.00041	17.28492	−0.00023
5	13.34260	17.28492	−0.00041	−0.00023	22.10499	−0.00013

(*Continues*)

DOI: 10.1002/atr

(Continued).

6	17.28492	22.10499	−0.00023	−0.00013	28.69806	−0.00007
7	22.10499	28.69806	−0.00013	−0.00007	37.32824	−0.00004
8	28.69806	37.32824	−0.00007	−0.00004	48.81541	−0.00002
9	37.32824	48.81541	−0.00004	−0.00002	64.00611	−0.00001

The *Newton–Raphson* method improves the value of λ_{new} based on the formula $\lambda_{\text{new}} = \lambda_{\text{old}} - \frac{P(\lambda_{\text{old}})}{P'(\lambda_{\text{old}})}$ where λ_{old} is any values that are near to λ_{opt}. This method is not supposed to be used because in our case, the value of $P'(\lambda) = \frac{d^2\theta}{d\lambda^2}$ is almost zero (unless for the range that is near to λ_{opt} (see Figure 3)).

APPENDIX 2: LINGO COMMANDS AND THEIR PURPOSES.

Command	Purpose
objectiveFunctionScript( )	Maximize the pedestrian flows through a network
flowEqualityScript( )	Provide the equal outflow and inflow for available nodes
probabilityChannelScript( )	Ensure the throughput of a node is correctly channeled to its downstream nodes based on their routing probabilities
corridorConstraintScript( )	Set the total arrival rate so that it is smaller or equal to its optimal arrival rate to avoid any blocking
LScreateEnvLng( )	Create a LINGO environment object
LSopenLogFileLng(*pLINGO*, *path*)	Open a LINGO log file for debugging the application • *pLINGO* is the variable where the value of the *LScreateEnvLng*() is assigned • *path* is the directory path where the log file (.log) is stored
LSsetPointerLng (*pLINGO*, *dObj*, *nPointersNow*)	Create the direct memory linkage between the application and a LINGO solver so that its data can be transferred and retrieved n and out of LINGO • *dObj* is the variable to be mapped • *nPointersNow* is the *n*th pointer in the stored memory list, and this requires a Lingo script to embed the @*POINTER*(*n*) command. For example, @*POINTER*(*3*) = *XS_Corridor1* requests LINGO to export the value of *XS_Corridor1* to its third memory location. The first memory location is to be reserved for retrieving the solution status; that is, @*POINTER*(*1*) = @*status* where the command tells if the solution is the global optimum, infeasible, unbounded, feasible, and so on.
LSexecuteScriptLng (*pLINGO*, *cScript*)	Request LINGO to solve the model where *cScript* is a generated script whose solution information is passed from LINGO back to the application' variables via @*POINTER* (*n*) reference
LSdeleteEnvLng (*pLINGO*)	Delete the created LINGO environment to free up the system memory allocated to the LINGO object

DOI: 10.1002/atr