J. Automation & Systems Engineering 1-1 (2007): 32-39

Regular paper

Journal of Automation & Systems Engineering

APPLYING GENETIC ALGORITHMS TO QUEUING NETWORK DESIGN

M. Hourani, M. Ozden, P. Maynard-Zhang and F. Moore

This study evaluates the efficacy of using the continuous genetic algorithm (GA) to solve complex queuing network problems. Queuing network design involves random processes and decisions that are subject to a cost constraint. Therefore, we run a simulation program to determine the fitness of each solution in the population. The GA produces good results compared to alternative methods, without the need for restrictive assumptions.

Keywords:.

1. INTRODUCTION

A queuing network [3] is a set of interconnected queuing systems. They are commonly used for the performance and reliability evaluation of computer communication systems, manufacturing systems, airport check-in systems, fast food restaurants, and many other service-based applications. The difficulty in designing such systems lies in the fact that there may be several decision variables (such as service rates, routing probabilities, and buffer sizes) that must be determined optimally in terms of a system's performance (e.g., throughput). There may also be constraints on these variables (e.g., a cost function to minimize). These define a challenging optimization problem. Existing analytical optimization models can only handle very simple versions of the problem in terms of network size and forms of the distributions.

GAs [4] are a family of optimization algorithms inspired by the biological process of natural selection, and have been successfully applied to a wide variety of optimization problems. GAs can be used to construct numerical optimization techniques that perform effectively on problems characterized by huge search spaces. Continuous GAs [5] represent the subclass of these algorithms where the variables to be optimized are real-valued.

In this research, we adapt and apply the continuous variable genetic algorithm (GA) strategy to the important and challenging problem of designing queuing networks. This research utilizes the continuous GA to identify state-of-the-art solutions to the queuing network design problems in the presence of complex constraints. In so doing, we demonstrate that the GA can be made to handle random (noisy) performance evaluations well. Accommodating this randomness necessitates a sophisticated simulation model run for each chromosome evaluation. We show experimentally that our approach is competitive with analytical approaches that return near-optimal results, yet without the need for the restrictive assumptions they require.

BITS Technolead Co. Damascus SYRIA. mhourani@technolead.com
Department of Computer Science and Systems Analysis, Miami University Oxford, Ohio 45056 USA. ozdenm@muohio.edu, pedmayn@gmail.com.
Mathematical Sciences Department, University of Alaska Anchorage, Anchorage, Alaska 9508 USA. affwm@uaa.alaska.edu.

2. QUEUING NETWORK DESIGN

A queuing network problem can be specified in terms of functional, cost, performance, or dependability constraints. [2] A *functional* constraint specifies the behavior of the system. A *cost* constraint is either a fixed constraint that restricts the optimization search space to solutions satisfying the constraint, or calls for minimizing the total cost during optimization. A *performance* constraint states how well the system should perform in terms of throughput, response time, etc. A *dependability* constraint states how the system tolerates component faults, and may include tolerances on minimum acceptable reliability, availability, etc.

Queuing network design takes these specifications and converts them into a system configuration that fulfills the specifications. This configuration consists of a set of hardware parameters (such as the number of processors, memory sizes, and the speed and capacity of network resources), together with a system topology.

The large number of candidate solutions and the complexity of the requirements make the task of manually choosing optimal configurations infeasible. Optimization techniques offer a natural approach to solving such problems.

2.1 Problem Specification

We have borrowed a challenging network problem described by Bolch *et al.* ([2], problem 11.1) for this study. The aim of this problem is to maximize throughput given a cost constraint in a closed queuing network. This closed network, shown in Fig. 1, is a multiprogramming computer system with a CPU and three disk devices. The CPU is labeled node 1 and the three disks are labeled nodes 2, 3, and 4, respectively. In our study, the degree of multiprogramming, K (the number of jobs in the system), is fixed at a value of four. Since the network is closed, no new jobs may enter the network and none of the current jobs may leave the network. We refer to this problem as *QN4*.

The routing probabilities are $p_{11} = 0.05$, $p_{12} = 0.5$, $p_{13} = 0.3$, $p_{14} = 0.15$, $p_{21} = p_{31} = p_{41} = 1.0$ where p_{ij} is the probability that a job will go to node j after leaving node i. For example, after a job gets out of the CPU (node 1), there is 5% probability for it to return to the CPU, 50% probability to go to node 2 (the first disk), and so on. After a job leaves a disk, it always goes to the CPU.

The *service rate* μ_i is the average number of jobs that can be completed in a node i in a single unit of time. It is defined as $\mu_i = v_i/d_i$ where v_i and d_i are the speed and the mean number of work units per visit, respectively, for device i. For a CPU, d_i is the mean number of instructions per CPU burst and v_i is the CPU speed (the number of instructions per time unit). For a storage device, d_i is the mean number of words in an I/O operation and v_i is the speed of the storage device in words per time unit.

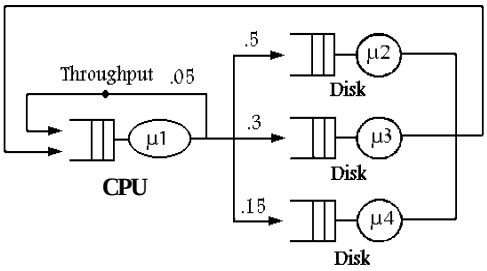

Fig. 1. Queuing network problem QN4.

In this problem, service rates are the values we need to determine. We use an exponential distribution with an average $1/\mu_i$ to calculate the service time for a job to be completed in node *i*.

Each node is either idle or busy serving a job. If a job reaches an idle node, that node will immediately start serving this job. Once the service is completed, the job will move on to the next node. If a job reaches a busy node, the job joins the waiting queue for that node. When a node has finished serving all the items in its queue, it returns to the idle state.

Throughput is the average number of jobs passed through a point in a single unit of time. It serves as a useful measure of the network's performance: the higher the throughput, the better the network. In this closed queuing network problem, throughput is measured on the path from the output of the CPU to the input of the CPU.

A solution must not only maximize throughput, but its cost must also be below some threshold. This cost has linear and nonlinear components. The non linear component is obtained by using the following formula:

$$\sum_{i=1}^{N} c_i \mu_i^{\alpha_i} \quad \text{where } \alpha_i > 1, \tag{1}$$

In the case of a multiprogramming computer system, there is an additional linear cost $C_m K$ for the main memory where K is the degree of multiprogramming (i.e., the number of jobs in the system) and C_m is a constant. Thus, the total cost function is given by:

$$COST = C_m K + \sum_{i=1}^{N} C_i \mu_i^{\alpha_i} , \tag{2}$$

2.2 Analytical Solution

Bolch *et al.* [2] describe an analytical approach to solving the generalized version of QN4 where the closed queuing network has N stations and K jobs. This approach, called *BFS*, is an approximation method whose solution is obtained by solving the following equations:

$$\lambda^* = \frac{\cos T \cdot K}{\left(\sum_{i=1}^{N} \sqrt{c_i e_i}\right)^2 + K \sum_{i=1}^{N} c_i e_i} \tag{3}$$

$$\mu_i^* = \lambda^* \cdot e_i \cdot \left(\frac{\sum_{j=1}^{N} c_j e_j}{K \sqrt{c_i e_i}} + 1 \right), \tag{4}$$

where λ^* is the optimal throughput and e_i is the visit ratio for node *i* (i.e., the fraction of jobs that visit node *i*). All the nodes service one job at a time.

For QN4, we assume the total budget maximum COST for this problem is 500 with C_m = 50 and cost factors c_i and exponents α_i as given in Table 1.

Table 2 gives the optimal solution for this problem as computed by the BFS method.

The BFS approach is known to produce solutions close to the optimal. Unfortunately, its usage is restricted to networks having a specific configuration and using exponential service time distributions. GAs hold the promise of flexibility with respect to these restrictions without sacrificing optimality.

TABLE 1: The Cost Factors c_i And Exponents α_i For Queuing Network Components

	c_i	α_i
CPU	131.9	0.55
Disk1	11.5	1.00
Disk2	54.2	0.67
Disk3	54.2	0.67

TABLE 2: The BFS Optimal Result

K	λ	μ_1	μ_2	μ_3	μ_4
4	0.076	1.967	1.979	0.847	0.571

3. CONTINUOUS GENETIC ALGORITHMS

A GA is an iterative process that operates on a *population* of candidate solutions. GAs are heuristic-driven and, as such, do not ensure an optimal solution. Furthermore, GAs are stochastic, so the same algorithm may evolve different solutions in different runs. Consequently, it is common to perform several independent runs when solving a particular problem, and retain the best solution.

A *continuous GA* [5] is a GA where the decision variables that define each candidate solution take on real values. We consider each component of the continuous GA strategy in a little more detail.

3.1 Encoding

The choice of chromosome encoding depends mainly on the problem to be solved. A variety of encoding methods are common in the literature. For this study, we use *value encoding* [4] to represent each chromosome with a string of digits. For example, consider the problem of encoding ten variables in a 50-digit long chromosome, where each variable can take on any value in [0, 0.99999]. We would represent the five most significant digits of each value as characters (Fig. 2). Each sequence of digits encoding a variable value is called a *gene*. Note that we can easily scale the approach to accommodate larger and different variable domains.

Variable values	0.13463	0.69785	…
Chromosome	1 3 4 6 3	6 9 7 8 5	…

Fig. 2: A Chromosome Encoding.

3.2 Selection

Selection is the process by which chromosomes are chosen from the current generation so that those with higher fitness values have a higher probability of contributing offspring to the next generation. In the *steady-state selection* approach used in this study, the population is sorted according to their fitness values. Good chromosomes (e.g., the top 50%) are selected as parents, with the same probability for creating new offspring, while other chromosomes with lower fitness values do not qualify to be parents. Table 3 gives an example.

3.3 Crossover, Mutation, and Elitism

Once selection is complete, pairs of chromosomes are chosen to parent a pair of chromosomes in the next generation. *Crossover* – the exchange of genetic material – is applied to each pair according to a *crossover probability* parameter to create two new offspring. Crossover is performed in the hope that new chromosomes will exhibit better fitness than their parents. We use two types of crossover in this study: *single-point* and *micro*. Single-point crossover (Fig. 3) operates by picking a random point within the two parent chromosomes, then exchanging the genes of the two chromosomes at and below this point to produce two new offspring. Micro crossover, on the other hand, picks one random digit to swap.

After crossover is performed, mutation randomly changes a selected gene in selected offspring. Mutation occurs according to the (typically very small) *mutation probability*, and helps to prevent the GA from converging to a locally optimal but globally suboptimal solution [4]. For continuous GAs, *value mutation* involves setting a whole gene from the selected offspring chromosome to a new random value. On the other hand, *add-value mutation* adds or subtracts a small random number to or from the value of the gene.

TABLE 3: Fitness Values And Selection Probabilities For Steady-State Selection With a 50% Selection Rate.

Chromosome #	1	2	3	4	5	6	7	8
Fitness value	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6
Selection probability	0.25	0.25	0.25	0.25	0.0	0.0	0.0	0.0

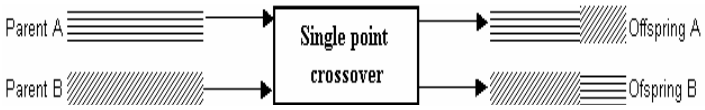

Fig. 3: Single-Point Crossover.

Elitism consists of copying the best chromosome (or a few of the best chromosomes) to the new population without crossover or mutation [4]. Elitism can rapidly increase the performance of GA, because it prevents the loss of the best chromosomes from the previous generation.

3.4 Population size

Population size is the number of chromosomes in a population. If there are too few chromosomes, GAs will have fewer possibilities to perform crossover and only a small part of the search space will be explored. On the other hand, if there are too many chromosomes, then GAs will slow down. Note that the population size is not *necessarily* kept fixed, although this is common practice.

4. CONTINUOUS GAS FOR QUEUING NETWORK DESIGN

The BFS method [2] gives the (near) optimal solution to the QN4 problem assuming specific configurations and service time distributions, in particular, exponential distributions. We compare the results of applying a continuous GA against these known optimal settings.

4.1 Fitness function and the simulator

In the QN4 problem, the decision variables are the speeds of the processor and the disks. Fitness is a composite function of throughput and cost. The cost is introduced into the objective function with a penalty coefficient using the same concept of the penalty functions that leads to an analytical solution in the BFS method. According to the *Penalty Function method* [1], the fitness value is penalized proportionally to the violation amount; otherwise, the fitness value is equal to the throughput. That is,

$$F(\mu_1, \mu_2, \mu_3, \mu_4) = \lambda - \eta(c), \tag{4}$$

where λ is the throughput obtained by the simulation; *c* is

the cost of the design defined by μ_1, μ_2, μ_3, and μ_4; and

$$\eta(c) = \begin{cases} 0, & \text{if } c <= l \\ 0.02\ (c\text{-}l), & \text{if } c > l, \end{cases} \tag{5}$$

where l is the cost limit 500. The value for the cost penalty factor (0.02) was determined by a simple initial trial-and-error method, so that if a constraint violation occurred, the cost would remain close to the limit value (500).

Due to the random interaction of system components, fitness for each candidate queuing network design solution is determined by running a simulation program for the design. The input to this program specifies the speeds of the processor and disks, and the output is the estimated throughput. To allow direct comparison to the BFS method, the program used a random exponential distribution to generate service times to the network stations and/or arrival rates to the entities. Thus, each run of the simulation produces slightly different results, even for the same parameters. Note, however, that our GA-based model can be used to design any queuing network configuration with any time distribution: the flexibility of our approach should make it suitable for identifying optimized solutions to other analytically intractable versions of this and other problems.

TABLE 4: Average Throughput For QN4 With Optimal Parameter Settings Of The BFS In Our Simulator

i	1	2	3	4	5	6	7	8	9	10
λ_i	0.0742	0.0810	0.0747	0.0798	0.0752	0.0779	0.0763	0.0755	0.0804	0.0762

In order to verify its correctness, our simulation program was executed with the optimal service rates as input to see if the average throughput was close to the analytical result obtained by the BFS approach. The simulation program was run for 100,000 time units and the average throughput was collected at every 10,000 time units. We used the results (Table 4) to obtain the following statistics: overall average throughput = 0.0771 with variance $S^2 = 6.2\times10^{-6}$, $t_{0.05/2,9} = 2.262$, and half width (hw) = 0.00178 (95% confidence interval). So the throughput is 0.0771±0.00178 at a 95% confidence level. The BFS result 0.076 falls inside this range, indicating the correctness of our simulator.

4.2 GA-Based Algorithm

We use the following value encoding scheme: Each chromosome represents a candidate solution by storing the speeds of the processor and the three disk drivers. Since each variable's value consists of 5 digits, the total length of each chromosome is 5*4 = 20 digits.

The speed range for all four parameters is restricted to the range [0.1, 5). To convert the five digit number into a speed, we use the formula

Speed = 0.1 + (5 – 0.1)×(5-digit number)×10-6, (6)

Our GA attempts to evolve optimal values for the speed of the processor and three disk drivers. We used a trial-and-error approach to determine the following best-result parameters for the GA:

- Population size (fixed) = 500
- Steady-state selection rate = 25%
- Simple-point crossover rate = 37.5%
- Micro crossover rate = 37.5%
- Value mutation rate = 2.5%
- Add-value mutation rate = 2.5%
- Elitism number = 1
- Termination criterion: Best value has not changed for 50 generations.

Our algorithm consists of three steps:

Step 1: Run GA

Run GA. Limit the simulation run time for each chromosome to a short period of 1000 time units to obtain a quick evaluation. Repeat this process 500 times for every candidate solution in each generation and return the average throughput.

Step 2: Re-evaluate Best Five Chromosomes of GA

Pick the best five chromosomes returned by step 1 and use the simulator to re-evaluate them for 10,000 time units each in order to obtain a better estimate of their performance.

Step 3: Simulate Best Chromosome for a Long Time

Use the parameters of the best chromosome from step 2 as input to the simulation program and run it for 100,000 time units. Estimate the throughput every 10,000 time units.

5. EXPERIMENTAL RESULTS

Table 5 shows the results of executing our GA for 65 generations. We conclude the following from these results: the overall average throughput = 0.0774, $S^2 = 1.2\times10^{-5}$, $t_{0.05/2,9} = 2.26$, and hw = 0.00248 (95% confidence interval). So the throughput is 0.0774±0.00248 at a 95% confidence level.

TABLE 5: The Average Throughput Estimated For QN4 With The GA

i	1	2	3	4	5	6	7	8	9	10
λ_i	0.0760	0.0725	0.0812	0.0818	0.0765	0.0792	0.0783	0.0781	0.0711	0.0793

We Compare the BFS and GA results in Table 6. We find that the GA reaches a solution very close to the optimal within the cost constraint. Thus, our GA achieves results

comparable to BFS, without requiring the complex mathematical analysis and the assumptions about specific configurations or service time distributions made by BFS.

6. CONCLUSION

We have shown that continuous GAs can be used to evolve effective solutions in the complex domain of queuing network design. The difficulty in such problems is created by the many decision variables, such as service rates, buffer sizes at the queuing stations, and routing decisions that must optimize system throughput in the presence of randomness and constraints such as cost functions. By utilizing a simulation program to evaluate fitness, our GA accounts for the complex interactions among the system components, as well as the randomness of services and job routing. The simulation program readily models complexity in system behavior.

Specifically, this study focused on a multiprogramming computer system application whose goal is to determine the speeds of a processor and three disk drivers in order to maximize the throughput in a closed queuing network that allows only a fixed number of jobs. Furthermore, the solution must be subject to a complex cost function that must be kept below a fixed budget value.

The analytical method BFS in [2] yields an approximate solution close to the optimal, but only under the assumption of exponentially distributed service times for a small network.

Our GA approach produced results that were very close to the solution found by the BFS method. Unlike BFS, however, our GA required neither complex mathematical analysis nor unnecessary assumptions about specific configurations or service time distributions. Thus, the GA proved to be a good heuristic approach to the queuing network design problem.

The trade off, of course, is that the GA approach takes much more time online to find the solution. This can be seen as a flexibility-time trade off.

In future research, our approach should be tested on queuing networks with a larger number of stations and with different kinds of service time distributions.

TABLE 6: Comparison of GA and BFS results

	μ_1	μ_2	μ_3	μ_4	Total cost	Throughput
BFS results	1.967	1.979	0.847	0.571	499.84	0.076
GA results	1.972	2.018	0.801	0.590	499.58	0.07740 ± 0.00248

References

[1] Bazaraa, M.-S. and C.-M. Sherry. *Nonlinear Programming: Theory and Algorithms*, John Wiley & Sons: 1979.

[2] Bolch, G., S. Greiner, H. de Meer and K. Trivedi, *Queuing Networks and Markov Chains*, John Wiley & Sons: 1998, p.p 565-568 , problem 11.1

[3] Gelenbe, E. and G. Pujolle, *Introduction to Queuing Networks*, John Wiley & Sons: 1999.

[4] OBITKO, M.. "INTRODUCTION TO GENETIC ALGORITHMS", PRAGUE UNIVERSITY OF APPLIED SCIENCES, 1998.

[5] Yang, J.-M., J.-.T. Horng and C.-Y. Kao, "A Continuous Genetic Algorithm for Global Optimization," in Proceedings of the 1997 International Conference on Genetic Algorithms, 230-237.