AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING, AUTOMATICS, COMPUTER SCIENCE AND BIOMEDICAL ENGINEERING

Department of Automatics and Bioengineering

Queue systems and networks

Optimization of customers scheduling in queue network using cockraoch swarm algorithm.

Authors: Andrzej Bobola

Jakub Kubisiowski

Jakub Łopatka

Katarzyna Pawlik

Michał Stęchły

Michał Tomański

Master studies: Control science and Robotics

Kraków, 2015

Table of Content

1 Queue problem

A queue problem is widely described in many scientific articles. This chapter is mainly based on article [1]. A queue is a waiting line formed whenever the demand for service exceeds the service availability. A queuing system consists of customers waiting in a queue for service if necessary and leaving the system after they are served. Customer may not be a human customer necessarily – it is generic description of any unit which needs a form of service. The main features of queuing system are:

- arrival pattern of customers
- service pattern of customers
- queue discipline
- system capacity
- number of service channels

In most queuing systems, the arrival pattern is based on probability distribution describing the interval times. If an arrival pattern does not depend on time, it is called a stationary arrival pattern. Otherwise it is called nonstationary pattern. The most important factor when it comes to service patterns is the probability distribution that describes the sequence of customer service times. Service may be single (for example shop service) or batch (people using elevator). The service efficiency may depend on many factors such as queue length or service difficulty. Such service is called state-dependent service. Queue discipline describes the manner in which customers receive service. The most popular queue disciplines are FIFO (First In First Out), LIFO (Last In Firs Out) or disciplines based on priorities. The capacity of queuing system may be infinite or it may be limited amount of customers. A system may have a single server or multiple parallel servers. A customer arriving to find more than one free server may choose at random between them to receive service.

The standard notation to describe a queueing process is A/B/X/Y /Z, where:

- A indicates the interarrival time distribution,
- B indicates the probability distribution describing the service time,
- X is the number of parallel service channels,

- Y is the restriction on system capacity
- Z is the queue discipline.

X and Y can be any integer in [1, ∞), but A and B are described by symbols that represent the probability distributions. For example M is an exponential distribution, D is deterministic, and GD is a general distribution. In many cases, only the first three symbols are used. If there is no restriction on system capacity, that is, if Y = ∞, the standard is to omit the symbol. Another convention is to omit the queue discipline if it is FIFO.

2 Cockroach Swarm Optimization (CSO)

Cockroach Swarm Optimization algorithm is one of Swarm Intelligence (SI) methods of optimization. There is a significant number of articles [2, 3, 4, 5, 6] dealing with CSO.

Cockroaches are social insects. They go in swarms and develop some techniques, that are useful while looking for food. This was an inspiration for researchers who developed CSO algorithm.

Cockroaches behaviors such as going in swarms and scattering from the light are the key concepts of CSO. Escaping from light is of utmost importance because it protects the swarm of cockroaches from being blocked. In other words it guarantees their movement. In CSO every cockroach moves to local optimum of its visibility. The best local optimum is a global optimum at the end of the cycle. In the next cycle all the strongest cockroaches form small swarms and follow the global optimum. Looking at small swarms, the follower might be better, because it follows in similar but no the same way as local optima. Dispersion of cockroaches means that at a time, a cockroach will have to change direction in order to maintain the current individual diversity. This is related to appearance of the light. Another important concept of CSO is phenomenon of replacing a randomly chosen individual by current best individual. It depicts natural cockroaches behavior facing lack of food – stronger cockroach eats the weaker one.

The cockroach swarm optimization algorithm can be presented as follows [3, 5, 6]:

STEP 1: Initialize algorithm's parameters and randomly generate population (step, the visual distance of cockroach visual, D – space dimension, n – number of cockroach individuals, MaxIt – maximum number of iterations, w – the inertia factor from range < 0, 1 >), the i-th individual represents a vector $Xi = (x_{i1}, x_{i2}, \ldots, x_{iD}),\ i = 1, 2, \ldots, n.$

STEP 2: Search P_i (the optimal individual within the visual scope of X_i) and P_g (the global optimal individual).

STEP 3: Implement behavior of a ‘’going in swarm” as follows:

- If a cockroach X_i is locally most powerful (local optimum), then the cockroach goes to the global optimum P_g, which is the most powerful cockroach:

$$X_i = w * X_i + step * rand < 0,1 > * \left(P_g - X_i\right) for\ X_i = P_i$$

- Otherwise, the cockroach goes to a local optimum of P_i, which is the locally most powerful cockroach:

$$X_i = w * X_i + step * rand < 0,1 > * (P_i - X_i)\ for\ X_i \neq P_i$$

Update P_g.

STEP 4: Implement dispersing behavior by the equation:

$$X_i = w * X_i + rand < 1, D >, update\ Pg.$$

STEP 5: Implement ruthless behavior by $X_k=P_g$, or $X_k=0$; k is a random integer within <1,n>.

STEP 6: If a predefined stopping criterion is met then output the results, otherwise go back to STEP 2.

3 Defining a problem

The first objective of this project is to create a queuing network consisting of servers and queues. Queueing networks assume a Poisson arrival process.

A Poisson process can be characterized in different ways [7]:

- Process of independent increments
- Pure birth process – the arrival intensity λ (mean arrival rate; probability of arrival per time unit
- The "most random" process with a given intensity λ

Definition:

The number of arrivals N(t) in a finite interval of length t obeys the Poisson(λt) distribution,

$$P\{N(t) = n\} = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

Moreover, the number of arrivals $N(t_1, t_2)$ and $N(t_3, t_4)$ in non-overlapping intervals ($t_1 \leq t_2 \leq t_3 \leq t_4$) are independent.

When it comes to the program, there is a file called networkStucture.csv that defines all the servers and queues. Its format has been described in table 3.1

Table 3.1 networkStructure.csv description

	arg1	**arg2**	**arg3**	**arg4**
Label	Server/Queue	id	lambda(Poisson)	Next Server ID (if arg1 = S), Queue to Server ID (if arg1 = Q)
Values	S - Server Q - Queue	1, 2, 3 ...	1, 2, 3 ...	1,2,3 … -1 means that it's the final server and Client has been served

Each row of this file defines a new Server or Queue. The number of defined Services as well as Queues is unlimited.

After creating the network consisting of servers and Queues, the next step is to create a list of customers. This goal is achieved by creating a file called routes.csv. Every row in this file defines customer and IDs of Servers of destination. It is very important to choose servers that are connected together, because every single mistake may result in runtime error of program. The number of customers is unlimited.

Having created a queuing network as well as customers list, the next objective is to optimize queuing sequence of customers. This is achieved by implementing Cockroach Swarm Optimization (CSO) algorithm. The implementation of algorithm for this specific problem has been widely described in further part of this article.

4 Simulation of a queuing network

4.1 Structure

The program for Simulation queuing network has been written in python programming language. It consists of following source files:

- CockroachLibrary.py
- QueueLibrary.py
- SimLibrary.py
- runQueue.py

CockroachLibrary.py contains definitions of Cockroach and CSO classes as well as some auxiliary functions.

QueueLibrary.py is crucial when it comes to queuing network and customers initialization. It contains definitions of Customer, Queue and Server classes as well as methods that make queuing network run properly.

Simlibrary.py is responsible for simulation process. It contains definition of Sim class which methods care for information from servers and queues gathering. It also calculates statistical information, such as the mean time spent in the queue by the customers or the mean queues' length.

RunQueue.py is a launch file of entire project.

4.2 Launching simulation

RunQueue.py can be launched with several flags, which modify the behavior of simulation.

- -f: in order to run the script we need to specify input file ID. Default is f=1
- -T: positive values are total time of the simulation. However value "T=0" means, that simulation will run as long as it is necessary to serve all the customers.
- -v: verbose. If set to 0 prints out only the statistics. If set to 1 prints out all the logs, as shown below. If set to 2 prints out state of all the customers in the specific format used by GUI.

- -s: if set to 0 executes one time only. If set to 1 runs the CSO algorithm
- -p: let us specify the permutation of clients at the input.

4.3 User interface

The application provides User Interface in two different ways. There has been designed user-friendly GUI that depicts the flow of customers in the queuing network. For those who prefer quick access to results the best option is to read the logs from Python Console. More details on those launch options have been described below.

a) GUI

It is possible to launch a project using graphical user interface. In order to do this man should run file Queues.exe located in GraphSharp/bin/Debug directory. It is necessary to have .NET 4.5 installed on the local machine. A screen from GUI is presented in figure 4.1.

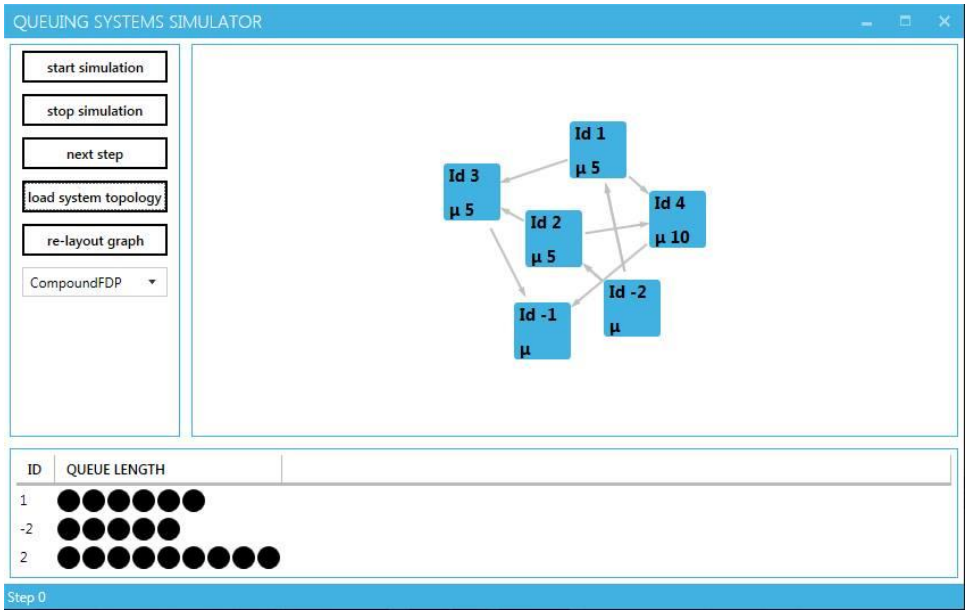

Figure 4.1 GUI

GUI consists of three panels:

- buttons panel

- visualization panel
- queue length panel

Buttons panel provides communication between user and program. It enables to load the queue network and start, stop or move towards the simulation. Visualization panel shows the layout of network. Queue length panel gives information about current states of queues.

b) Console Launch

To get quick access to results, using Python Console is much more convenient. In order to get results all the user need to do is to launch runQueue.py file, which contains main method.

Python Console provides following results of simulation:

- average time that customer waits in specific queues
- average length of specific queues
- average service time for specific servers
- average time in network

For debugging purposes verbose results logs may be desirable. To get verbose results run **runQueue.py** with **-v 1** script parameters. Small fragment of verbose console logs has been presented in fig. 4.2

```
------ t= 35 ---------
10: number of customers in queue Q1
9: number of customers in queue Q1
7: number of customers in queue Q2
6: number of customers in queue Q2
4: time left in server S1
3: time left in server S1
1: time left in server S2
0: time left in server S2
0: time left in server S3
0: time left in server S3
1: time left in server S4
0: time left in server S4
------ t= 36 ---------
9: number of customers in queue Q1
8: number of customers in queue Q1
7: number of customers in queue Q2
5: number of customers in queue Q2
3: time left in server S1
2: time left in server S1
3: time left in server S2
2: time left in server S2
1: time left in server S3
0: time left in server S3
3: time left in server S4
2: time left in server S4
------ t= 37 ---------
8: number of customers in queue Q1
8: number of customers in queue Q1
5: number of customers in queue Q2
5: number of customers in queue Q2
2: time left in server S1
1: time left in server S1
2: time left in server S2
1: time left in server S2
0: time left in server S3
0: time left in server S3
2: time left in server S4
1: time left in server S4
```

Figure 4.2 verbose console logs example

Verbose console logs enables debugging queueing network step by step. It helps developer to find possible errors of network and it is a tool that can give the answer whether such network runs well. The logs for single simulation step contain information about network elements - servers and queues. The results are paired together - it means that there is information about changing states of elements. For instance "just before" time t = 35 there are 10 customers in queue Q1 and "just after" time t = 35 there are 9 customers in that queue (fig 4.2). Simulation updates the state of all queues first, then the states of all servers.

4.4 Simulation pseudocode

Here we present the pseudocode of our simulation:

Single simulation of queue network

1. Create Sim object
    a. Load files with network structure and routes
    b. Set permutation of customers at input
    c. Create all necessary objects - queues, servers and customers
2. Run simulation (Sim.run())
    a. add all customers to the first queue
    b. for each step of simulation:
        i. update state of all queues
        ii. update state of all servers
    c. print output

Simulation of CSO

1. Create CSO object
    a. Load file with routes
    b. Initialize cockroaches
    c. Calculate fitness of all cockroaches
    d. Find best cockroach
2. For each iteration
    a. Move cockroaches
    b. Scatter cockroaches
    c. Check if best cockroach is the same as in the last iteration
    d. If there was no progress in the last 5 iterations - break loop.
3. Print output

5 CSO implementation

Nature inspired algorithms are very hard ones - there are a lot of parameters and options we may, or may not, include into our implementation. However our aim was not to optimise scheduling in the network, but rather implement the CSO algorithm to learn how it is working and how to use it in the queuing network. Thus we have not performed extensive research trying to find the best parameters and one can argue with decisions we made.

Our optimization problem was to schedule customers in the input queue in order to minimize mean time all customers spent in the network.

Main class used in this algorithm is called Cockroach. Each Cockroach object represents a permutation of customers in the input queue. It has following fields:

- size - number of customers
- fileID - from which it should take routes and network structure
- sequence - permutation of clients
- fitness - fitness value
- visual - range of view

It also has following methods:

- calculateFitness - calculates fitness of a cockroach
- setSequence - sets given sequence for a cockroach.
- checkIfInVisual - checks if other cockroach is in range of view
- moveToward - moves one cockroach toward another

We calculate fitness of a cockroach by running it 1500 times and taking the mean time of a clients in the network.

To calculate the distance between two cockroaches we count how many swaps are needed to change one permutation into another. So if we have two sequences: s_A = [0,1,2,3,4,5] and s_B = [0,3,2,1,5,4], the distance between them is 2 (we have to swap 1 with 3 and 4 with 5 in s_B).

Moving cockroach A toward cockroach B works as follows:

1. we choose random integer i, which represents position in the sequence of A and B
2. if corresponding values in A's and B's sequences (v_A and v_B) are the same we go back to 1
3. if not, we search v_B in seq_A and swap it with v_A.
4. As a result we diminished distance between seq_A and seq_B by one.

Scattering cockroaches is achieved by creating a new cockroach with random sequence and moving the old one step into its direction. Every cockroach excluding the best one is scattered with the probability of 25%.

Our CSO algorithm implementation does not include the ruthless behavior.

6 Tests

6.1 Number of iterations

Serving time for each server is generated by the Poisson distribution. So each simulation can differ very much from the other started with the same parameters. Therefore the first thing we had to check was to the number of simulations required for the cockroach to have stable mean - it is not changing much with each new simulation.

The results were very surprising. As shown at the figure below, it took usually about 2000 iterations for the mean to stabilize, however at least 4000 iterations would be really satisfying. Unfortunately, we didn't have enough time to run simulations with such accuracy and we decided to run them with 1500 iterations.

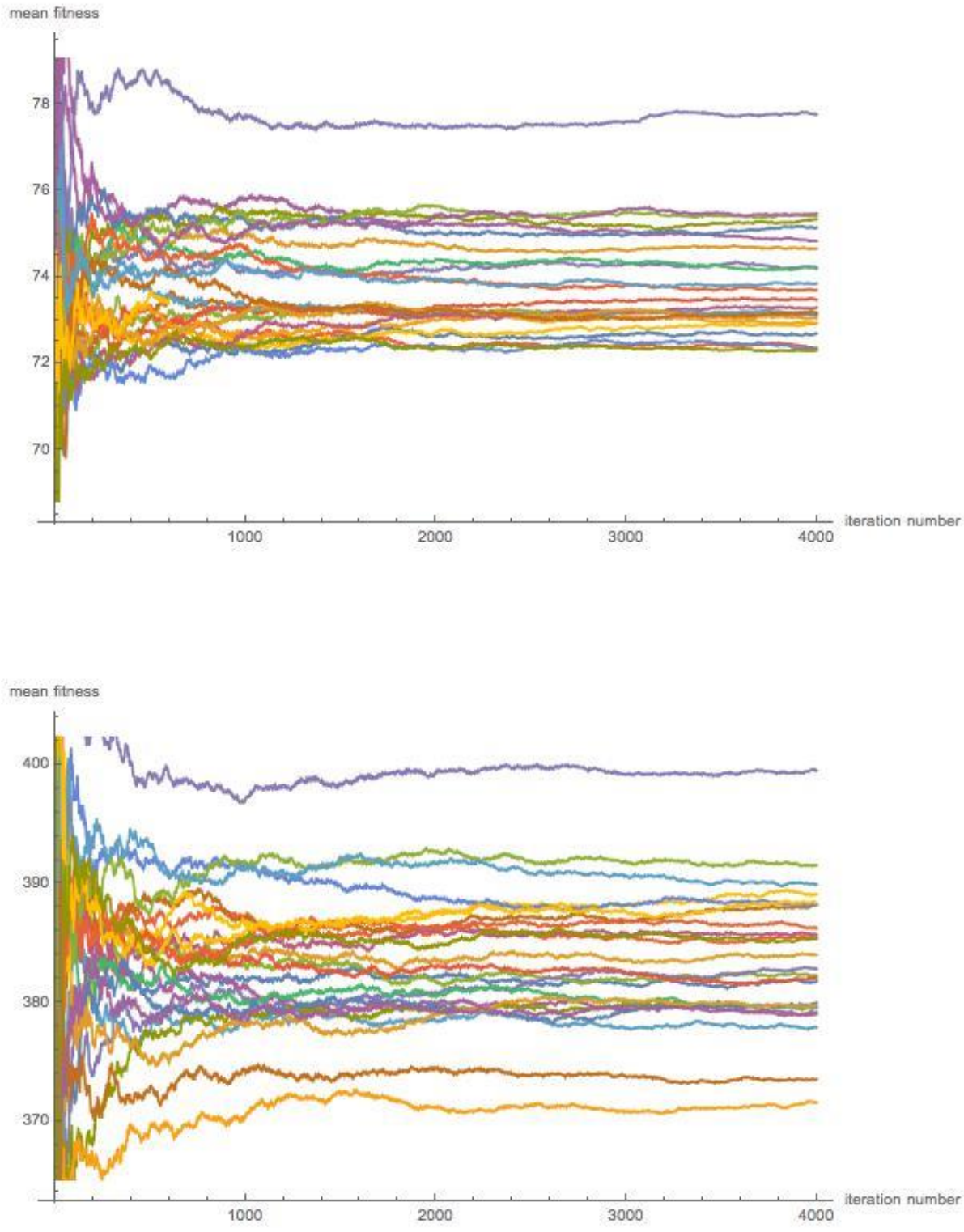

Figure 6.1 Mean fitness as a function of number of iterations, for two different network structures. For each network we tested 25 cases, shown in different colors.

6.2 Testing CSO performance

The next step was to check how the algorithm works on different networks. We created 4 different networks and ran simulation in order to check how the algorithm works with different sizes of the population.

For each network we ran CSO multiple times, starting with population of 10 cockroaches and increasing it gradually. We defined following rules for the simulations:

- maximum number of iterations was 15 in case of the first network and 20 for the rest
- if fitness of the best cockroach was the same for 5 consecutive iterations optimization ended
- visual of each cockroach was 0.6 times the length of the sequence (rounded).

Execution of one CSO took 1 - 12 hours, so we have not completed full range of population sizes from 10 to 100 in each case.

Results of the tests are shown on the figure 6.2, 6.3 and 6.4.

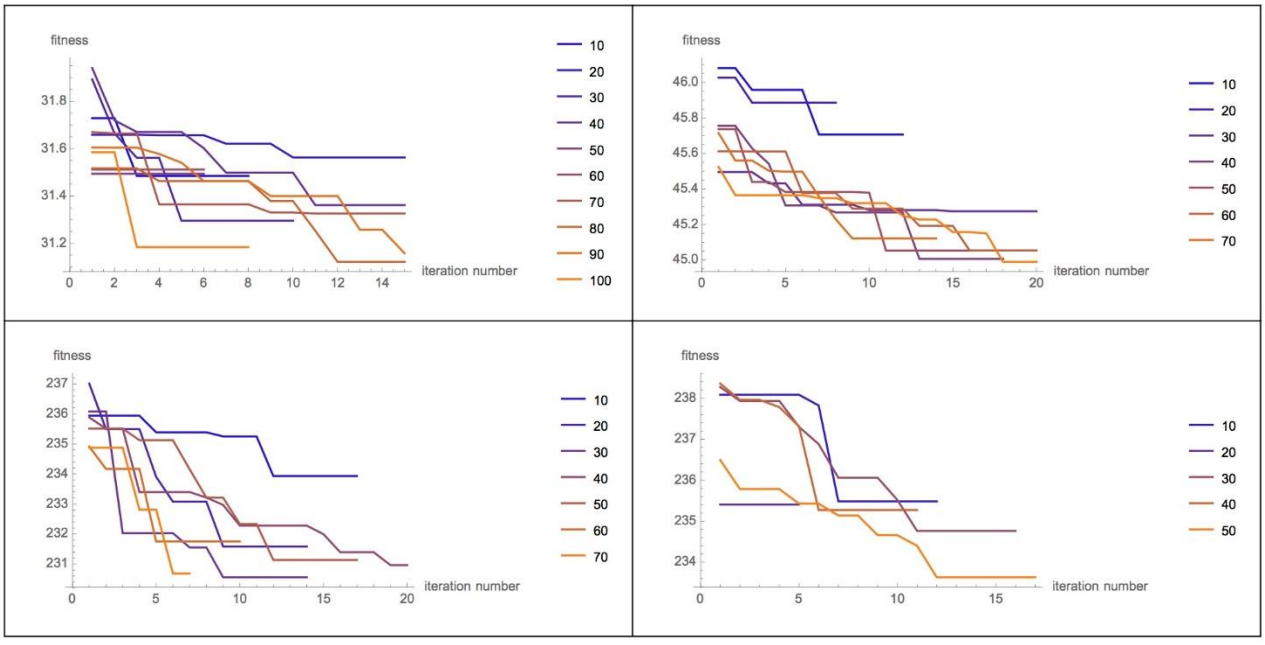

Figure 6.2 Fitness of the best cockroach in the population as a function of iteration number. Different lines correspond to different sizes of population.

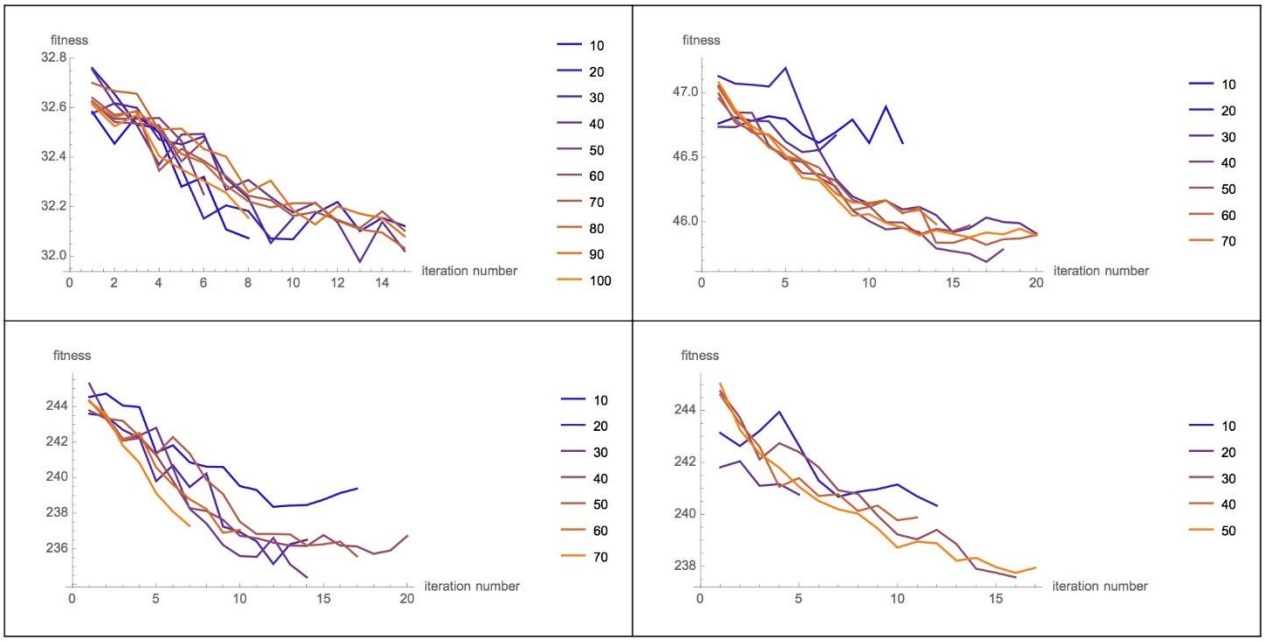

Figure 6.3 Mean fitness of the population as a function of iteration number. Different lines correspond to different sizes of population.

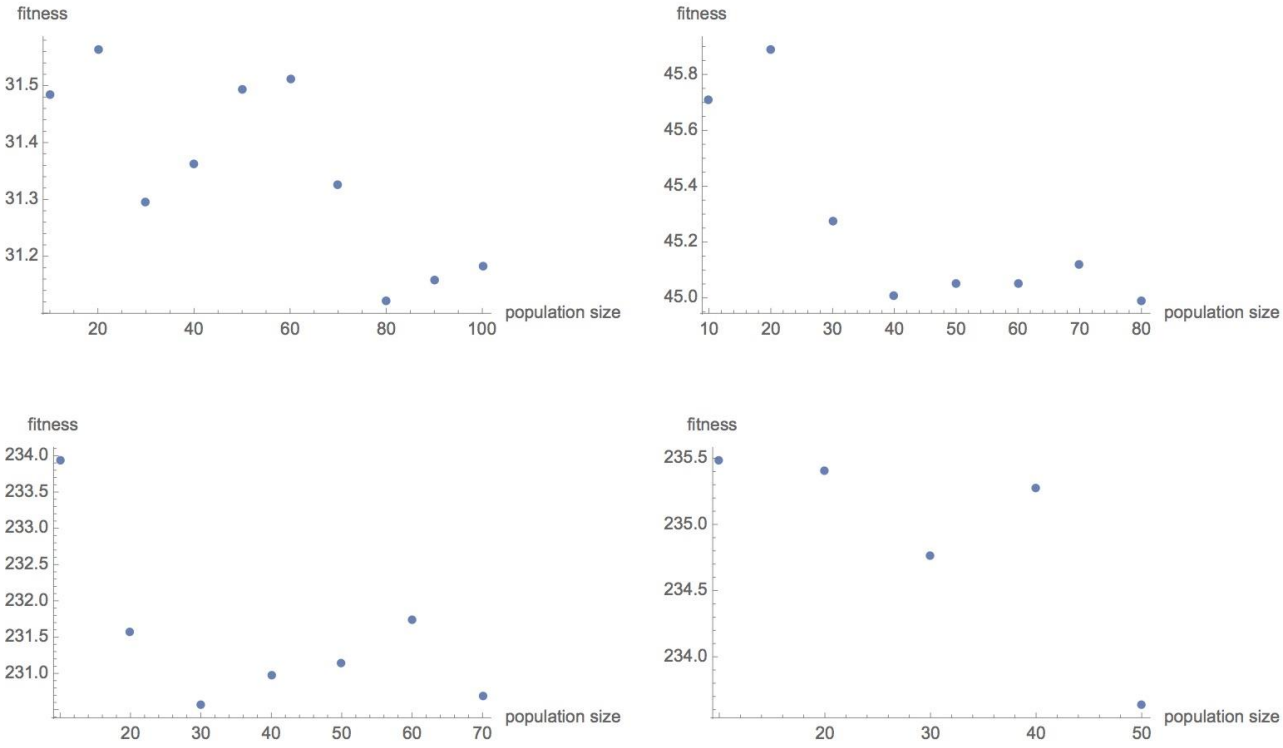

Figure 6.4 Fitness of the best cockroach as a function of population size

We expected that with the increase of the size of population we should get better results. Data shown on the figure 6.4 show such general trend, however the sample is too small to draw any definite conclusions.

From the results we obtained it is clear that CSO works as a optimization method in the case of scheduling customers in the queuing network. However, we have not compared it with other methods, so it is hard to assess its performance.

7 Conclusion

All defined goals of this project were achieved. First of all information about queuing networks were found. Looking up scientific articles helped us to understand such networks better. We created fully functional simulation of a queueing system, which provides interface for user-defined network structure and GUI. For the optimization problem we used CSO algorithm. The simulation part of the project is implemented in Python.

To test our simulation we created some sample network architectures and analyzed it. Simulations with test data ran perfectly so we were ready for the next step that was providing user-friendly GUI. Simplicity of GUI was of utmost importance. Therefore our team spent a lot of time vividly discussing on this topic.

The next step was CSO implementation. We adapted it for our purpose, which was optimizing scheduling of the customers. After some preliminary tests to adjust some of the parameters, we ran multiple tests to see how this algorithm works in our case. Results are satisfying, even though not very spectacular - we managed to obtain only few percent of improvement in comparison to the starting states. Further research requires additional simulations as well as much more reading about the subject.

In conclusion, this project was both interesting and educational. The results we obtained came up to expectations that we had made at the very beginning of this project.

8 References

[1] H. Constantin, "Markov Chains and Queuing Theory", The University of Chicago

http://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/Constantin.pdf

[2] C. Obagbuwa, A .O. Adewumi, "An improved Cockroach Swarm Optimization", The Scientific World Journal Volume 2014

http://www.hindawi.com/journals/tswj/2014/375358/

[3] J. Kwiecień, B. Filipowicz, "Comparison of firefly and cockroach algorithms in selected discrete and combinatorial problems", Biulletin of the Polish Academy of Sciences Vol. 62, No. 4, 2014

http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.baztech-fde9de18-abac-445e-867e-75f14ea94d66?q=2baf9c67-8952-42a7-a300-b5f57c9ae05d$1&qt=IN_PAGE

[4] L. Cheng, Z. Wang, S. Yanhong, A. Guo, "Cockroach Swarm Optimization Algorithm for TSP", Huaian College of Information Technology

http://www.scientific.net/AEF.1.226

[5] Z. Chen and H. Tang, “Cockroach swarm optimization”, II Int. Conf. on Computer Engineering and Technology 6, 652–655 (2010).

[6] Z. Chen, "A modified cockroach swarm optimization", Energy Procedia 11, 4–9 (2011).

https://www.infona.pl/resource/bwmeta1.element.elsevier-d7448ab9-9b13-39a9-b16f-14e01ef0bc16

[7] J. Virtamo, "Queueing Theory / Poisson process", Helsinki University of Technology https://www.netlab.tkk.fi/opetus/s383143/kalvot/E_poisson.pdf