OPTIMIZATION OF WATER DISTRIBUTION NETWORKS USING MIXED-INTEGER LINEAR PROGRAMMING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EREN UZUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CIVIL ENGINEERING

AUGUST 2016

Approval of the thesis:

OPTIMIZATION OF WATER DISTRIBUTION NETWORKS USING MIXED-INTEGER LINEAR PROGRAMMING

submitted by **EREN UZUN** in partial fulfillment of the requirements for the degree of **Master of Science in Civil Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Director, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmail Özgür Yaman
Head of Department, **Civil Engineering** _____

Prof. Dr. Ayşe Burcu Altan Sakarya
Supervisor, **Civil Engineering Dept., METU** _____

Examining Committee Members

Prof. Dr. Mustafa Göğüş
Civil Engineering Dept., METU _____

Prof. Dr. Ayşe Burcu Altan Sakarya
Civil Engineering Dept., METU _____

Prof. Dr. Nuray Tokyay
Civil Engineering Dept., METU _____

Assoc. Prof. Dr. Mete Köken
Civil Engineering Dept., METU _____

Asst. Prof. Dr. Önder Koçyiğit
Civil Engineering Dept., Gazi University _____

Date: 3 August 2016

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Eren UZUN

Signature :

ABSTRACT

OPTIMIZATION OF WATER DISTRIBUTION NETWORKS USING MIXED-INTEGER LINEAR PROGRAMMING

UZUN, Eren

M.Sc., Department of Civil Engineering

Supervisor: Prof. Dr. Ayşe Burcu ALTAN SAKARYA

August 2016, 82 pages

The present study aims to discuss the advantages and disadvantages of the design of water distribution networks by making use of mixed integer linear programming. The developed optimization algorithm considers the minimization of the total cost as the objective function. The total cost of water distribution network is defined as cost of pipes, reservoirs and pumps. Nodal demands, nodal pressure limits and pipe velocity limits are satisfied while optimizing the network. Energy equation is the equality constraint that is satisfied for all the links of the network. In this study, the method proposed by Samani and Zanganeh (2010) is coded as Java based computer program. The consistency of the proposed method is tested on three networks and further improvement is achieved by making changes on proposed method. The developed computer program finds the optimal values of the decision variables which are the pipe diameters, reservoir heights and pump characteristics. The solution of the optimization problem is iteratively obtained by running both hydraulic solver (EPANET) and linear programming solver (lp_solve) in succession. Results are compared with previous studies.

Keywords: Water Distribution Network, Linear Programming, Optimization, Computer Programming

ÖZ

SU DAĞITIM ŞEBEKELERİNİN KARIŞIK-TAMSAYILI LİNEER PROGRAMLAMA KULLANILARAK OPTİMİZASYONU

UZUN, Eren
Yüksek Lisans, İnşaat Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Ayşe Burcu ALTAN SAKARYA

Ağustos 2016, 82 sayfa

Bu çalışmanın amacı karışık tamsaylı lineer programlama kullanarak su dağıtım şebekesi tasarlamak ve bunun avantaj ve dezavantajlarını tartışmaktır. Geliştirilmiş optimizasyon algoritmasının amacı toplam maliyeti minimize etmektir. Su dağıtım şebekesinin toplam maliyeti boruların, rezervuarların ve pompaların toplam maliyeti olarak tanımlanmıştır. Şebeke optimize edilirken basınç limitleri ve su akış hızı limitleri göz önünde bulundurulmuştur. Enerji denklemi şebekedeki bütün borular için sağlanmıştır. Bu çalışmada Samani ve Zanganeh (2010) tarafından önerilen çözüm metodu Java tabanlı bilgisayar programı olarak kodlanmıştır. Bu metodun güvenilirliği ve doğruluğu üç su dağıtım şebekesi üstünde denenmiştir ve metot üzerinde değişiklikler yapılarak sonuçlarda gelişme sağlanmıştır. Geliştirilen bilgisayar programı üç ayrı kriteri optimize etmektedir. Bu kriterler boru çapları, reservuar yükseklikleri ve pompa karakteristikleri olarak tanımlanmıştır. Optimizasyon probleminin çözümü su şebekesi çözüm yazılımı (EPANET) ve lineer programlama çözücüsünün (lp_solve) bir arada ve tekrarlı olarak kullanımıyla sağlanmıştır. Sonuçları önceki çalışmalar ile kıyaslanmıştır.

Anahtar Kelimeler: Su Dağıtım Şebekesi, Lineer Programlama, Optimizasyon, Bilgisayar Programlama

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

TABLE OF CONTENTS vii

LIST OF TABLES ix

LIST OF FIGURES xi

CHAPTERS

1 INTRODUCTION 1

1.1 Problem Definition 1

1.2 Literature Review 2

1.2.1 Deterministic Approaches 3

1.2.2 Simulation based Meta-heuristic Approaches 5

1.3 Research Objective 6

1.4 Thesis Outline 6

2 PROBLEM FORMULATION 9

3 DEVELOPED COMPUTER PROGRAM 15

4 DISCUSSION OF RESULTS 21

4.1 Example 1 21

4.1.1 Problem definition of Example 1 21

4.1.2 Case 1 23

4.1.3 Case 2 27
4.1.4 Case 3 27
4.1.5 Case 4 28
4.2 Hanoi Network 29
4.2.1 Problem Definition of Hanoi Network 29
4.2.2 Solution with Proposed Model by Samani and Zanganeh (2010) 32
4.2.3 Solution with Improved Model 38
4.3 Two-loop Network 44
4.3.1 Problem Definition of Two-loop Network 44
4.3.2 Solution of Two-loop Network 46
5 CONCLUSION 49
5.1 Summary of the Research 49
5.2 Conclusion and Comments 49
REFERENCES 53
APPENDIX: DEVELOPED PROGRAM 57

LIST OF TABLES

TABLES

Table 4.1 Pipe data for Example 1 22

Table 4.2 Nodal data for Example 1 22

Table 4.3 Velocity and pressure constraints for Example 1..... 22

Table 4.4 Available pipes and their costs for Example 1..... 22

Table 4.5 Available reservoir heights and their costs for Example 1 22

Table 4.6 Discharge values with initial parameters for Example 1 24

Table 4.7 Lengths of pipes for Hanoi Network..... 31

Table 4.8 Demand of nodes for Hanoi network..... 32

Table 4.9 Commercially available pipe diameters and their costs for Hanoi 32

Table 4.10 Value of objective function for each iteration 33

Table 4.11 Comparison of pressure head values (m) with proposed model for Hanoi network 34

Table 4.12 Comparison of diameters (in.) and total costs with proposed model for Hanoi network..... 36

Table 4.13 Comparison of pressure values (m) with improved model for different *Pmin* values for Hanoi network..... 39

Table 4.14 Comparison of diameters (in.) and total costs with improved model for different *Pmin* values for Hanoi network..... 40

Table 4.15 Pipe related results of improved model for *Pmin*= 30 m 42

Table 4.16 Elevations and demands for nodes for Two-loop Network..... 45

Table 4.17 Available pipe diameters and their unit costs for Two-loop Network 45

Table 4.18 Pipe related results for Two-loop Network 46

Table 4.19 Node related results for Two-loop Network 47

Table 4.20 Result comparison with previous studies for Two-loop Network..... 47

LIST OF FIGURES

FIGURES

Figure 3.1 Simplified diagram of the optimization process 16

Figure 3.2 Example configuration input file 18

Figure 4.1 Network of example 1 21

Figure 4.2 Hanoi water distribution network 30

Figure 4.3 Two loop network 44

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Water is the vital source of life for mankind and it is an essential element for sustaining life. Sustainable water distribution has been a problem for decades.

According to researches, first public water supply systems are established almost 3000 years ago. Qanat systems, which can be described as slightly sloped hillside tunnels, are considered one of the first water distribution systems built by mankind. According to Jesperson (2001) these systems were built around 700 BC to supply water to Persia. Romans started constructing aqueducts from 300 BC for urban distribution and drainage. Machu Picchu spring water collection system, which is considered as a remarkable achievement of civil engineering, is built around 1450 AD in Peru. In the earlier periods water systems were made from stone, brick clay and wood. As years passed water supply systems also changed. In the early 20^{th} century pipelines made of steel, iron and reinforced concrete are used for this purpose with reservoirs, tanks, pumps and valves. With the significant increase in world population, the water demand also increased for cities. This made water distribution systems very big, complex and expensive infra-structures.

For today, a large amount of money and time are invested around the world to establish sufficient water distribution systems. Nearly 85% of the cost of a total water supply system is contributed toward water transmission and the water distribution network. This is the main reason why a lot of researchers have been working on optimization of water distribution systems in last 50 years.

In 2016, a water distribution network can be described as an infra-structure that delivers water to fulfill consumer demands (residential, industrial, commercial etc.) by considering pressure limits, velocity limits and quality limits. Water distribution networks usually contain pipes, reservoirs, tanks, pumps and valves. So the optimization of a water distribution network should satisfy all criteria stated above with the minimum cost. Even though other criteria such as reliability, environmental impact and water quality are of great importance, most of the studies have been done regarding only capital cost.

1.2 Literature Review

Before the computer era, the design of water distribution networks was based only on the designer's experience. The designer's experience and budget allocated for specific project were very important and critical for design process. Although Hardy Cross developed a method to solve for pressures or flows in closed loop water distribution systems in 1936, the method could not be implemented on big and complex networks because of the slow convergence and the amount of computing required. The method known as "Hardy Cross method" would have to wait for the invention of the computing hardware and software necessary for its full implementation. With the start of computer age, in 1957, Hoag and Weinberg adjusted the Hardy Cross method as hydraulic solver to computer system and applied the method to the water distribution network of the city of Palo Alto, California. The results were promising. With the help of computerized hydraulic solver, researchers started to study on developing methods for optimization of water distribution network.

The word "optimal" according to Merriam-Webster online dictionary means "most desirable or satisfactory". In engineering it can be stated as "maximum benefit with minimum cost". However, optimization process with different methods yields different result. Two different results that are achieved as optimum may be significantly different cost wise. Different methods of optimization almost always result in different results for water distribution networks. For example New York City water distribution network have been studied by many researchers. The total

cost of optimum water distribution network found by Schaake and Lai (1969) was 73.3 million dollars, where Quindry et al. (1981) found another solution with the cost of 63.6 million dollars. It should be noted that Cunha and Sousa (2001) found an optimal solution with the cost of only 37.1 million dollars for the same network. All researchers have used same limitations and requirements in their studies, but the methods they have used resulted in different solutions.

Many mathematical formulations and many problem solving methods are used in different optimization techniques such as linear programming, non-linear programming, dynamic programming, gradient-based techniques, enumeration methods, genetic algorithms etc. “The term optimization methods often refers to mathematical techniques used to automatically adjust the details of the system in such a way as to achieve the best possible system performance or, alternatively, the least-cost design that achieves a specified performance level.” (Walski, et al., 2003). In the following sub-chapters some of the widely used optimization methods are explained.

1.2.1 Deterministic Approaches

Deterministic approaches include linear programming and non-linear programming which are very commonly used by researchers. Alperovits and Shamir (1977); Quindry et al. (1981); Bhave and Sonak (1992) and many other researchers have used these methods in their studies.

If pipe diameters are selected as the main decision variables three approaches can be found in the literature. These are the split pipe, the continuous, and the discrete diameter approaches.

Alperovits and Shamir (1977) published one of the earliest attempts with split pipe approach. They proposed a linear programming gradient method which simplifies original problem by solving generated linear sub-problems in succession. The proposed model was about two steps. First step was about solving a flow status for a given water distribution network. Second step was about solving the linear programing sub-problem for network diameters for the present discharge

distribution. This was an iterative procedure and at each iteration program itself calculated a hydraulic solution at the same time. So there was no need for a network solver in their study. It optimizes the design for the given flow distribution. The method proposed by Alperovits and Shamir (1977) was modified and developed by many researchers such as Quindry et al. (1981), Fujiwara et al. (1987), Kessler and Shamir (1989).

The solutions provided by split pipe approach have some significant disadvantages. Firstly, convergence was not always guaranteed and secondly, optimization may result in finding two pipe segments with different available pipe diameters for a link. This means impracticable solution. According to Savic and Walters (1997) split-pipe design creates problems when they are implemented in real life situations. The solutions should be changed and modified in order to have one pipe diameter for each pipe segment. However, this process may result in going off the limits of current constraints which makes the solution infeasible. Moreover, converting split-pipe design into one commercially available diameter may make total cost higher.

In the continuous diameter approach, the diameter is taken as a continuous variable. It is used by researchers such as Fujiwara and Khang (1990) and Varma et al. (1997). The solutions provided by this method are also impracticable since only a discrete number of pipe diameters exist in practice. Converting continuous diameters to discrete ones causes lots of problems: cost of objective function may rise or even solution may become infeasible. The discrete diameter approach should be executed in order to avoid above mentioned problems.

Even if the discrete diameter approach is used, there are still big disadvantages of using deterministic methods. It is proven that water distribution network optimization problem is mathematically nonconvex and multimodal by Templeman (1982). This is actually an important problem since the optimization of unknown pipe diameters strictly depends on the unknown pipe discharges. Furthermore; the optimization process should satisfy pressure constraints, nodal demands and etc. The general idea of optimization of water distribution networks with linear programming method is to relax this non-linearity by assuming or selecting an initial discharge

pattern. Then created linear programming sub-problem is solved to obtain pipe diameters and predict change in discharge values. Then an iterative process goes on until convergence on the values is established. According to Sonak and Bhave (1993) the solutions achieved by deterministic methods always rely on assumed initial discharges and complexity of the network problem. There is always a risk of getting trapped at the nearest local minimum before finding the global minimum. In order to prevent this situation different runs should be made with different starting points (initial discharge distributions). By doing that, local minimum achieved by this method may coincide with the global one.

1.2.2 Simulation based Meta-heuristic Approaches

In order to overcome the weaknesses of deterministic approaches, researchers began to work on meta-heuristic algorithms such as genetic algorithm (Goldberg and Kuo, 1987; Murphy et al., 1993; Simpson et al., 1994; Dandy et al., 1996; Savic and Walters, 1997; Walters et al., 1999), simulated annealing (Cunha and Sousa, 1999) and tabu search (Cunha and Ribeiro, 2004) and harmony search (Geem et al., 2001) to optimize water distribution network.

Meta-heuristic methods are able to handle nonlinear mixed integer models while searching for global optimality. These methods do not need to simplify the problem to relax nonlinearity. Moreover, they can deal with discrete pipe diameters quite easily.

Another advantage of meta-heuristic approach is the randomness process during the search of optimum. This randomness causes the optimization process to be set free from any trapped local minimum value. The ability to get away from local minimum makes meta-heuristic algorithms more powerful when they are compared with deterministic methods.

On the other hand, if the optimization problem is not calibrated adequately, the optimization process may take a lot of time as a result of high number of iterations. As computer processor technology develops, meta-heuristic methods converge results faster than before.

1.3 Research Objective

In this research, optimization of water distribution networks is done by mixed integer linear programming method. The optimized parameters are pipe diameters, reservoir heights and pump characteristics. The main purpose is to minimize objective function which is the total cost of optimized parameters above. Nodal demands, nodal pressure limits and pipe velocity limits are satisfied while optimizing the network.

As previously mentioned in Chapter 1.2, linear programming method become outdated for optimization of water distribution networks. With the help of developing computer processor technology, most of recent studies use meta-heuristic methods such as genetic algorithm, harmony search, tabu search. They get improved result compared to other studies before. However, Samani and Mottaghi (2006) came up with a new optimization method which uses linear programming. According to their research, the results were promising. Later on Samani and Zanganeh (2010) improved this method and got satisfying results. However, the comments about their method were mostly negative. Martinez (2008) stated that proposed method is not actually a new method, just a repetition of previous studies. Savic and Cunha (2008) added "The proposed approach may be viewed as a step backward, both in terms of research and practical application of optimization to water distribution network design".

In this study, the method proposed by Samani and Zanganeh (2010) is coded as Java based computer program. The reliability of the proposed method is tested. The comments of other researchers are taken into consideration. Advantages and disadvantages of this method are discussed widely. Moreover, further improvement is achieved by making changes in the proposed method.

1.4 Thesis Outline

The research thesis is divided into four chapters excluding Introduction chapter. These chapters are problem formulation, computer program, discussion of results and conclusion.

Chapter 2 gives information about problem formulation. The equations used for optimization of water distribution network are explained in this part. Linearization of equations is also mentioned.

In Chapter 3, coded optimization software is described briefly. The relation between hydraulic solver and linear programming algorithm solver is explained. Moreover, information about iterative solution algorithm is clarified in this chapter.

In Chapter 4, the proposed optimization procedure is performed on three different water distribution networks. Detailed network descriptions for each network and broad analysis of results can be found in this chapter. Main disadvantages of proposed model also explained in examples.

Finally in Chapter 5, conclusion of this research is summed up. Personal comments on the future of linear programming based water distribution network optimization are stated.

CHAPTER 2

PROBLEM FORMULATION

Since the main aim of this study is to minimize the total cost, the total cost of water distribution network should be defined first. The total cost of water distribution network which is also the objective function can be formulated as;

$$F = f(D_N) + g(H_K) + h(PC_j) \tag{2.1}$$

where

F : total cost of water distribution network

$f(D_N)$: total expenses of pipes

$g(H_K)$: total expenses of reservoirs

$h(PC_j)$: total expenses of pumps

D_N : diameter of pipe N

H_K : level of reservoir K

PC_j : characteristic of pump J

These cost functions can be formulated as;

$$f(D_N) = \sum_{N=1}^{NP} L_N CP_N(D_N) \tag{2.2}$$

$$g(H_K) = \sum_{K=1}^{NR} CR_K(H_K) \tag{2.3}$$

$$h(PC_j) = \sum_{J=1}^{NPU} CPU_J(PC_J) \tag{2.4}$$

where

N : pipe number in the network

L_N : length of pipe number N

CP_N : unit cost of pipe N

NP : number of pipes in the network

K : reservoir number in the network

CR_K : cost of reservoir number K

NR : number of reservoirs in the network

J : pump number in the network

CPU_J : cost of pump number J

NPU : number of pumps in the network

Substituting Equations 2.2, 2.3 and 2.4 into Equation 2.1 the objective function is obtained.

$$F(D_N, H_K, P_J) = \sum_{N=1}^{NP} L_N CP_N(D_N) + \sum_{K=1}^{NR} CR_K(H_K) + \sum_{J=1}^{NPU} CPU_J(PC_J) \tag{2.5}$$

However, it should be noted that Equation 2.5 is still a non-linear equation and it should be converted into a linear equation so that the linear optimization process can work properly.

To formulate the optimization problem, pressure constraints should be defined. For that, a reference node should be selected first. Location of any reservoir on the system can be selected as reference node. Applying energy equation between reference node R and the successive node $R+1$, by neglecting velocity heads, results in

$$\frac{\boldsymbol{P}_{R+1}}{\gamma} = H_R - \Delta Z_{R-R+1} - h_{f\ R-R+1} \tag{2.6}$$

where

$\boldsymbol{P}_{R+1}/\gamma$: pressure head at node *R+1*

$\Delta Z_{R\text{-}R+1}$: elevation difference between nodes *R* and *R+1*

H_R : total head at the reference node, *R*

$h_{f\,R\text{-}R+1}$: head loss between nodes *R* and *R+1* resulted from pipe

R : reference node

R+1 : node immediately after the reference node, *R*

If energy equation is applied between all successive nodes *i* and *j* including node *R+1*, all nodal pressure data can be expressed as

$$\frac{\boldsymbol{P}_j}{\gamma} + \Delta Z_{j-i} - h_{f\ j-i} - \frac{\boldsymbol{P}_i}{\gamma} + HP_J = 0 \tag{2.7}$$

where

i, j : two successive node connected with only one link

HP_J : total head of pump between nodes *j* and *i*

$\boldsymbol{P}i/\gamma$: pressure head at node *i*

$\boldsymbol{P}_j/\gamma$: pressure head at node *j*

$\Delta Z_{j\text{-}i}$: elevation difference between node *j* and *i*

$h_{f\,j\text{-}i}$: total head loss between node *j* and *i*

Since every node on the system now has pressure equation, pressure constraints for all nodes can be written as

$$\frac{\boldsymbol{P}_i}{\gamma} \leq \frac{\boldsymbol{P}_{max}}{\gamma} \tag{2.8}$$

$$\frac{\boldsymbol{P}_i}{\gamma} \geq \frac{\boldsymbol{P}_{min}}{\gamma} \tag{2.9}$$

where

$\boldsymbol{P}_{max}/\gamma$: maximum allowable pressure head

$\boldsymbol{P}_{min}/\gamma$: minimum allowable pressure head

Also it should be noted that pressure variables that are mentioned in Equations 2.6, 2.7, 2.8 and 2.9 are dependent variables. Their value strictly depends on pipe diameters, reservoir heights and pump characteristics. Similarly, velocity constraints for all pipes can be expressed as

$$V_N \leq V_{max} \tag{2.10}$$

$$V_N \geq V_{min} \tag{2.11}$$

where

V_{max} : maximum allowable velocity

V_{min} : minimum allowable velocity

V_N : velocity in pipe N

As stated earlier; Equation 2.5, the objective function, is a nonlinear equation since it contains functions that are nonlinearly related to variables as pipe diameters, reservoir heights and pump characteristics. However, these variables are discrete in nature. For example, at most 10 different pipe diameters can be selected for a specific water network design and these pipe diameters must be commercially available to use in real life. So that Equation 2.5 can be converted to a linear equation.

$$F(D_N, H_K, P_J) = \sum_{B=1}^{NPA} \sum_{N=1}^{NP} L_{NB} CP_{NB}(D_N) X_{NB} + \sum_{M=1}^{NRA} \sum_{K=1}^{NR} CR_{KM}(H_K) Y_{KM} + \sum_{I=1}^{NPUA} \sum_{J=1}^{NPU} CPU_{JI}(PC_J) P_{JI} \quad (2.12)$$

where

NPA : the number of available pipe diameters

NRA : number of available reservoirs

NPUA : number of available pumps

B : subscript indicating an available pipe diameter number

M : subscript indicating an available reservoir height number

I : subscript indicating an available pump characteristic number

X : 0-1 variable indicating selected pipe diameter

Y : 0-1 variable indicating selected reservoir height

P : 0-1 variable indicating selected pump characteristic

In order to ensure only one pipe diameter is selected for each link, total sum of X_{NJ} for each link should be equal to 1.

$$\sum_{B=1}^{NPA} X_{NB} = 1 \quad (2.13)$$

Also for the same reason stated above, sum of Y_{KM} and P_{JI} should be equal to 1 in order to select only one reservoir height for each reservoir and one pump characteristic for each pump.

$$\sum_{M=1}^{NRA} Y_{KM} = 1 \quad (2.14)$$

$$\sum_{I=1}^{NPUA} P_{JI} = 1 \tag{2.15}$$

Similarly pressure constraints, Equation 2.6 and 2.7, can be converted as

$$\frac{\boldsymbol{P}_{R+1}}{\gamma} = \sum_{M=1}^{NRA} H_{RM} Y_{RM} - \Delta Z_{R-R+1} - \sum_{B=1}^{NPA} h_{f\ R-R+1} X_{NB} \tag{2.16}$$

$$\frac{\boldsymbol{P}_j}{\gamma} + Z_j - \sum_{B=1}^{NPA} h_{f\ j-i} X_{NB} - \frac{\boldsymbol{P}_i}{\gamma} - Z_i + \sum_{I=1}^{NPUA} HP_J P_{JI} = 0 \tag{2.17}$$

Velocity constraints, Equation 2.10 and 2.11, can be converted to linear equation as

$$\sum_{B=1}^{NPA} \frac{Q_N}{(\pi/4) D^2{}_{NB}} X_{NB} \leq V_{max} \tag{2.18}$$

$$\sum_{B=1}^{NPA} \frac{Q_N}{(\pi/4) D^2{}_{NB}} X_{NB} \geq V_{min} \tag{2.19}$$

After all these linearization procedures, the decision variables for linear programming problem has become *X*, *Y*, *P* and ***P***. *X*, *Y* and *P* are 0-1 variables for selecting pipe diameters, reservoir heights and pump characteristics. However; ***P***, nodal pressure, is a real number variable. In order to solve the problem, mixed integer linear programming must be used.

CHAPTER 3

DEVELOPED COMPUTER PROGRAM

In Chapter 2 formulation needed for linear programming problem is explained. The idea is to perform hydraulic solver with initial diameters and use the discharge values obtained from hydraulic solver in LP problem solver. If Equations 2.16 to 2.19 are examined, it can be seen that initial discharge values are needed. On the other hand, performing hydraulic solver and LP problem solver one time does not give any realistic result. LP problem solver ensures that all constraints stay in the limits with minimum cost. However, when newly selected diameters applied to the network and hydraulic solver is performed once more, it is seen that many constraints go off the limits. This situation is totally expectable because when new diameters applied to the network, discharge values may change in many links. If discharge values change, then velocities in pipes and pressure values at nodes also change.

In order to solve the problem stated above, an iterative process is applied. Simply, after running hydraulic solver and LP problem solver once, network is updated with new diameter values. Hydraulic solver is executed again with these new diameters and new discharge values are obtained. LP solver is fed with updated discharge values and new diameters are obtained through LP solving process. This iterative process goes on until LP solver selects same diameters 2 times in a row in optimization process. That means the optimization process cannot be improved further with the discharge values found on last hydraulic solver step. Also this procedure ensures that all constraints stay within the limits. Since diameters would not be changed in the last cycle, the discharge values would not be updated. That

means pressure values arranged by LP problem solver on the last step should have the same values with the pressure values calculated on the last step by hydraulic solver. Summarized version of this whole iterative optimization process can be seen on Figure 3.1.

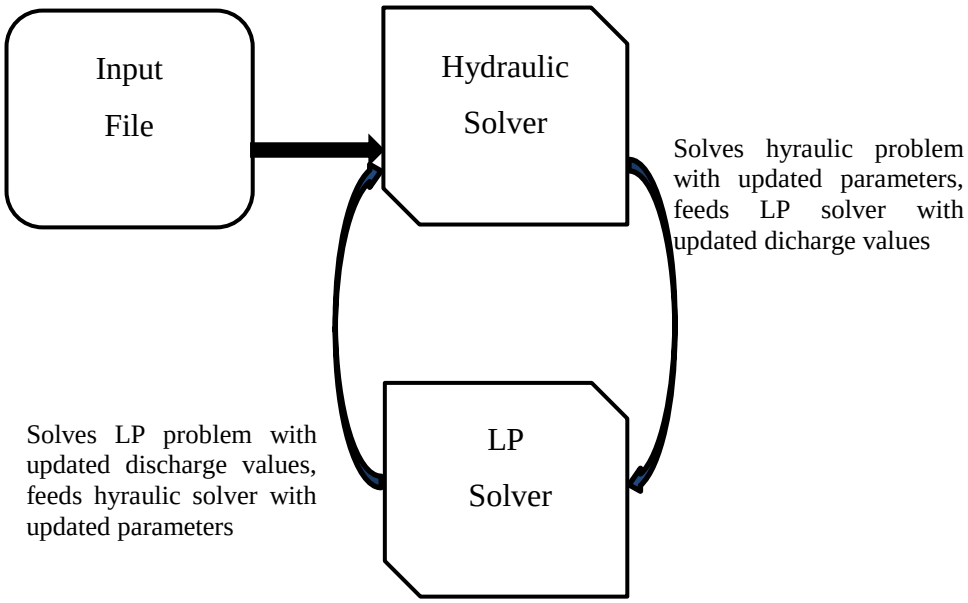

Figure 3.1 Simplified diagram of the optimization process

The optimization process, which can be summarized as running both hydraulic solver and LP problem solver successively, can take a lot of time and effort if it is done manually. Even though there are many software products that can do LP problem solving and hydraulic analysis individually, there is not any computer software that can do both successively. In order to speed up the optimization and make whole process automated, new computer program is developed.

To develop an optimization program one hydraulic solver software and one LP problem solver software are selected. Main criteria for making this selection are compatibility with Java platform and having an open source code.

EPANET is selected as hydraulic solver for the program. EPANET is a software product that models water distribution systems. It is also a public domain software which can be freely copied. EPANET has a dynamic link library (DLL) of functions that allow users to customize this software to their own needs which makes it very suitable for this study. Finally, it has a user interface that allows user to create water network systems and edit properties visually.

As LP problem solver, software called lp_solve is selected. It is a mixed integer linear programming solver which also has an open source code. The solver based on the revised "simplex method" and the "branch and bound method" for the integers. It has a dynamic link library (DLL) of functions, just like EPANET. This DLL file can be dynamically linked to the user's code. Lp_solve is fully compatible with Java platform.

The developed program needs 2 separate input files to work. First input file is network file. The network file should have all pipe data and nodal data. Pipe data includes length, ID, Hazen-Williams roughness coefficient, diameter, start node, end node of each pipe. Nodal data includes elevation, ID, demand of each node. Reservoirs also count as nodes. Such input file can be created quite easily by visual interface of EPANET. It should be noted that pipe diameters and reservoir heights in this network input file would be the initial parameters for hydraulic solver to find discharge values for the first iteration.

The second input file is the configuration file in Extensible Markup Language (XML) format which can be readable both human and machine. The configuration file should have maximum and minimum allowable pressure head values for each node and maximum and minimum allowable velocity values for each pipe. Moreover, different velocity limits can be enforced on specific pipes on the network. Pressure head values must be in meters and velocity values must be in meter/second. The configuration file should have commercially available pipe diameters, available reservoir heights, available pump characteristics and their corresponding costs. The diameters must be in millimeters and reservoir heights must be in meters. Example input file for configurations can be seen in Figure 3.2.

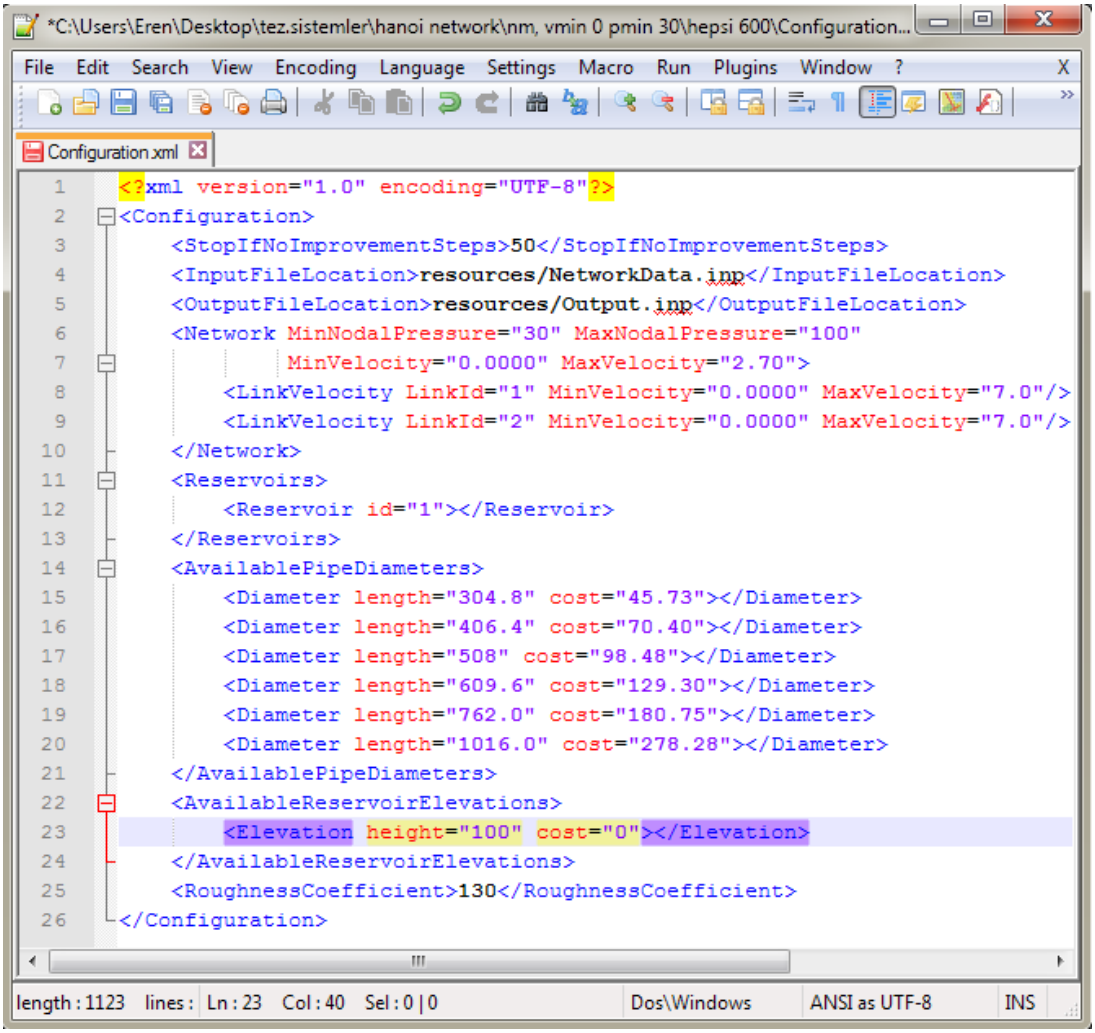

Figure 3.2 Example configuration input file

Firstly, the program uses EPANET Application Program Interface (API) to run hydraulic solver with initial diameters and reservoir heights. Then it uses data from both network file and configuration file and generates mixed integer linear programming problem in the format that can be read and solved by lp_solve API. To create the objective function and constraints, it uses equations from Equation 2.12 to 2.19 and discharge values from EPANET API. The created linear programming problem is solved by lp_solve API and all diameters and reservoir heads on existing network are updated with these new values. Then EPANET API runs hydraulic

solver with updated parameters, and find new discharge values. New linear programing problem is created once again with new discharge values. This iterative process goes on until lp_solve API gives same results two times in a row. LP problem created for each iteration can be found as output of this program in human readable format. Also network file after each iteration can be found as output as well.

The whole solution process can be summarized as follows. Hydraulic solver is used with estimated initial pipe diameters and reservoir heights. Initial discharge values are obtained in this step by using EPANET. Then with these discharge values LP problem is solved by lp_solve to determine pipe diameters and reservoir heights by considering pressure and velocity constraints. After that, hydraulic solver is used again with updated pipe diameters and reservoir heights. This iterative procedure continues until the solution cannot be improved any further.

CHAPTER 4

DISCUSSION OF RESULTS

4.1 Example 1

4.1.1 Problem definition of Example 1

The first example problem is selected as a simple one in order to see the downsides of the developed program clearly. Moreover, this network has been studied by Samani and Mottaghi (2006). As it can be seen on Figure 4.1 the system has 1 reservoir, 3 pipes and 2 nodes and also it is a looped system. The unknowns for this system are diameters of 3 pipes and height of reservoir. Hazen – Williams coefficient of each pipe and pipe lengths are given in Table 4.1, elevation of each node and their water consumption are given in Table 4.2, velocity and pressure constraints are given in Table 4.3, available pipe diameters and their unit costs are given in Table 4.4 , available reservoir heights and their costs are given in Table 4.5.

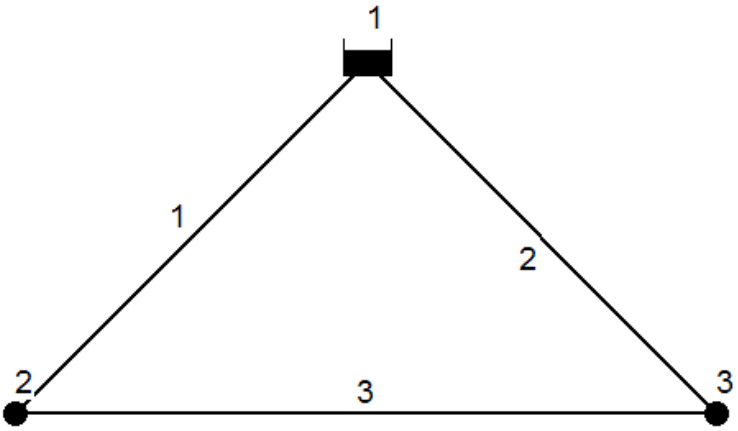

Figure 4.1 Network of example 1

Table 4.1 Pipe data for Example 1

Pipe		Hazen – Williams coefficient	Pipe length (m)
Start node, i	**End node, j**		
1	2	130	300
1	3	130	300
2	3	130	450

Table 4.2 Nodal data for Example 1

Node number	Elevation (m)	Consumption (L/s)
1	0	-180
2	0	100
3	0	80

Table 4.3 Velocity and pressure constraints for Example 1

	Allowable velocity (m/s)	Allowable pressure head (m)
Minimum	0.3	20.0
Maximum	2.0	60.0

Table 4.4 Available pipes and their costs for Example 1

Diameter (mm)	Cost (units/m)
80	32000
250	43000

Table 4.5 Available reservoir heights and their costs for Example 1

Reservoir height (m)	Cost (units)
25	21000000
30	30000000

As previously mentioned, this system has been studied by Samani and Mottaghi (2006). They used integer linear programming for the solution of the problem. In this study mixed integer real linear programming is used. Same results are expected since both methods should lead to the same solution theoretically. However, results obtained are quite different. In order to find out deficiencies of the proposed model, different cases are studied. These cases can be found in Sections 4.1.2 to 4.1.5.

4.1.2 Case 1

In the previous study done by Samani and Mottaghi (2006) the global optimum is found as D_1 = 250 mm, D_2 = 250 mm, D_3 = 80 mm with reservoir height of 25 m. The value of objective function is found as 42300000 units. This value is incorrect; the value of objective function is calculated as 6120000 units if the calculations are made manually with selected diameters and reservoir height. So, there is a calculation mistake in the published paper.

When the computer program is fed by current input data and same initial diameters for all pipes (80 mm or 250 mm for all pipes), the result is “infeasible”. The reason is that there is not any suitable pipe diameter to choose to satisfy both pressure and velocity constraints for the first step. This situation can be seen below in the problem which is generated by the developed program. This is the linear programming optimization problem of lp_solve which is the first step of solution procedure. The discharge values used for this step can be seen on Table 4.6. These values are obtained from EPANET with all diameters selected as 250 mm and reservoir height is selected as 30 m.

Table 4.6 Discharge values with initial parameters for Example 1

Pipe number	Flow direction		Discharge (m^3/s)
	Start node, i	End node, j	
1	1	2	90.56
2	1	3	89.44
3	3	2	9.44

Below, LP problem created for this specific problem can be examined closely. Equation 2.12 is used for objective function, Equations 2.16 and 2.17 are used for energy equations, Equations 2.8 and 2.9 are used for pressure constraints, Equations 2.18 and 2.19 are used for velocity constraints, Equation 2.13 is used for constraints to ensure one diameter is selected for each link and Equation 2.14 is for constraints to ensure one reservoir height is selected for each reservoir.

Objective function:

Minimize: 9600000 X_{21} +12900000 X_{22} +14400000 X_{31} +19350000 X_{32} +9600000 X_{11} +12900000 X_{12} +30000000 Y_{11} +21000000 Y_{12};

Energy equations:

-978.48 X_{21} -3.81 X_{22} +30 Y_{11} +25 Y_{12} -$\boldsymbol{P}_3$ = 0;

22.81 X_{31} +0.09 X_{32} +$\boldsymbol{P}_2$ -$\boldsymbol{P}_3$ = 0;

Pressure constraints:

$\boldsymbol{P}_2$ >= 20;

$\boldsymbol{P}_3$ >= 20;

$\boldsymbol{P}_2$ <= 60;

$\boldsymbol{P}_3$ <= 60;

Velocity Constraints:

17.79 X_{21} +1.82 X_{22} >= 0.3;

1.88 X_{31} +0.19 X_{32} >= 0.3;

18.01 X_{11} +1.84 X_{12} >= 0.3;

17.79 X_{21} +1.82 X_{22} <= 2;

1.88 X_{31} +0.19 X_{32} <= 2;

18.02 X_{11} +1.84 X_{12} <= 2;

Constraints to ensure only one pipe diameter is selected for each pipe and only one reservoir height is selected for each reservoir:

$X_{21} + X_{22} = 1$;

$X_{31} + X_{32} = 1$;

$X_{11} + X_{12} = 1$;

$Y_{11} + Y_{12} = 1$;

Integer definitions:

Integer: X_{21}, X_{22}, X_{31}, X_{32}, X_{11}, X_{12}, Y_{11}, Y_{12};

where

X_{21} : 0-1 variable for pipe 2, 80 mm diameter

X_{22} : 0-1 variable for pipe 2, 250 mm diameter

X_{31} : 0-1 variable for pipe 3, 80 mm diameter

X_{32} : 0-1 variable for pipe 3, 250 mm diameter

X_{11} : 0-1 variable for pipe 1, 80 mm diameter

X_{12} : 0-1 variable for pipe 1, 250 mm diameter

Y_{11} : 0-1 variable for reservoir, 30 m head

Y_{12} : 0-1 variable for reservoir, 25 m head

$\boldsymbol{P}_2$: variable represents pressure at node 2

$\boldsymbol{P}_3$: variable represents pressure at node 3

The developed program is fed by input file. It creates objective function and all constraints that are essential for lp_solve.

If the velocity constraints examined only, to satisfy the velocity constraints X_{22}, X_{31} and X_{12} must have value of 1; X_{21}, X_{32} and X_{11} must have value of 0. That means Pipe 1, Pipe 2 are selected as 250 mm and Pipe 3 selected as 80 mm. This is also the global optimum for this specific problem. However, this selection cannot be made because of pressure constraints. First 2 constraints on the output represent energy equations that include pressure variables. Reservoir head is selected as 30 m (Y_{11}=1 and Y_{12}=0) to be on the safe side when considering possible low pressure situation. When X_{22}, X_{31} and X_{12} are selected as 1 and X_{21}, X_{32} and X_{11} are selected as 0, $\boldsymbol{P}_3$= 26.19 m and $\boldsymbol{P}_2$= 3.38 m are obtained. Since value of $\boldsymbol{P}_2$ must be between 20 and 60 the problem becomes infeasible, and no selection can be made. Randomly selected pipe diameters leads to a discharge distribution. If these discharge values are far from the optimal solution discharge values then the program cannot make any selection to converge.

To summarize, it can be seen that velocity limits are forcing lp_solve to choose only 1 diameter for each pipe and if these diameters are selected, $\boldsymbol{P}_2$ (pressure at node 2) becomes lower than minimum pressure limit ($\boldsymbol{P}_2$= 3.38 m). Since minimum pressure constraint is 20 m, the problem becomes infeasible. Therefore, in order to get results and to test the viability of the program, runs with different constraints were made.

One more downside of the proposed model can be seen in this example. The diameter selection of pipe number 1 only relies on velocity limits. The reason is, for a loop with *n* number of pipes only *n*-1 energy equations can be written between nodes. Otherwise the created LP problem cannot be solved mathematically. This downside is valid for each loop. One pipe on each loop is only selected by considering velocity limits. If there is no velocity limits for specific problem, then proposed model is forced to select the pipe with minimum cost, usually the pipe with minimum diameter.

4.1.3 Case 2

For Case 2 allowable minimum velocity is changed from 0.3 m/s to 0 m/s. With this change, the problem is solved and a feasible result is obtained. However, this is not the optimum result.

All pipe diameters are determined as 250 mm (D_1, D_2 and D_3) and reservoir height, H_R is determined as 25 m. The program gets stuck at this step and cannot improve the solution further. If the program selects the diameter of pipe 3 as D_3=80 mm that would lead to an improved solution (which is the optimal for this case). However, this cannot be done. Because with the discharge values that are determined in the previous step, changing D_3 from 250 mm to 80 mm would cause pressure value to drop below the minimum limit which is 20 m. This is one of the biggest weaknesses of the program. Since the linear programming part is solved with the previous discharge values, this method cannot predict the change in discharge values. Thus, the global optimum may seem in the infeasible zone just because of that. Case 1 is the perfect example to show this downside. The program gets stuck in the local optimum and cannot improve solution, because global optimum is seen as it is in the infeasible zone.

4.1.4 Case 3

For Case 3, allowable minimum pressure value is changed from 20 m to 3 m. This change is made because there are no solutions that can satisfy both pressure and velocity constraints if the initial diameters of the system are selected same value (e.g. D_1, D_2 and D_3 = 250 mm). If minimum pressure value is selected above 3 m, result is found as infeasible.

Changing the lower limit of allowable pressure led us to optimal solution. It is the global optimum solution even for the original problem. However, this is just a coincidence. As stated in the Case 2, program only considers the current discharge. It only tries to get the pressure value above 3 m. However, the change in discharge causes pressure value to go above 20 m.

The downside of the program got clearer in this step: discharge values before and after each iteration may vary significantly. Therefore, selection of optimum diameters, which will lead to global optimum, may seem out of feasible zone with current discharge values. Linear programming algorithm that optimizes pipe diameters only considers current discharge values. It cannot predict the changes in discharge values for next iteration.

4.1.5 Case 4

In the Case 4 some significant changes in the constraint list has been made in order to test the viability of the program. Minimum allowable pressure is changed from 20 m to 3 m (as in the Case 3) and also velocity constraint is taken out of the formulation by changing its limits. Minimum velocity is changed from 0.3 m/s to 0 m/s and maximum velocity limit is changed from 2.0 m/s to 20.0 m/s. So the only difference between Case 3 and Case 4 is the velocity limits.

In normal cost optimization problems as limits get wider the solutions get cheaper since the feasible range gets bigger. However, this is not valid for our program and it is proven in Case 4.

In Case 3 total cost of objective function is found as 61200000 units after 2 iterations. In Case 4 the total cost of objective function is found as 62850000 units which is greater than Case 4 and this was unexpected. After investigating the steps deeply, it is found that in Case 4 the program gets stuck at a local minimum and cannot further improve the solution after first the step. In Case 3, velocity limits prevent the program to do significant changes at each iteration, thus, program does not get stuck at local minimum at the first iteration and it can improve solution at the second iteration.

The main reason of getting stuck at local minimum is same again: current discharge values prevent further improvement. Program cannot change diameter values because linear programming part only considers current discharge (discharge at that significant step) and cheaper solutions (also global optimal solution) does not satisfy constraints with current discharge at that step.

4.2 Hanoi Network

4.2.1 Problem Definition of Hanoi Network

The Hanoi network includes 34 pipes and 32 nodes as it can be seen on Figure 4.2. There is only 1 reservoir that feeds the whole system and it has 100 m constant head. Pipe lengths are given in Table 4.7 and nodal demands are given in Table 4.8.

The minimum pressure constraint is given as 30 m and there is no constraint for maximum pressure. Also there is no constraint for maximum or minimum velocity. All pipes are assumed to have constant Hazen - Williams roughness coefficient of 130. Commercially available pipe diameters and their costs are given in Table 4.9 for that specific network. The problem is to select pipe diameters from commercially available ones to satisfy constraints by considering minimum cost as the objective function.

The Hanoi network has significant importance in network optimization history. It has been solved by different solution techniques by many researchers such as Savic and Walters (1997), Abebe and Solomanite (1998), Cunha and Sousa (1999) and Liong and Atiquzzman (2004). Moreover, it was also studied by Samani and Zanganeh (2010) who has used linear programming method that is used and improved in this paper.

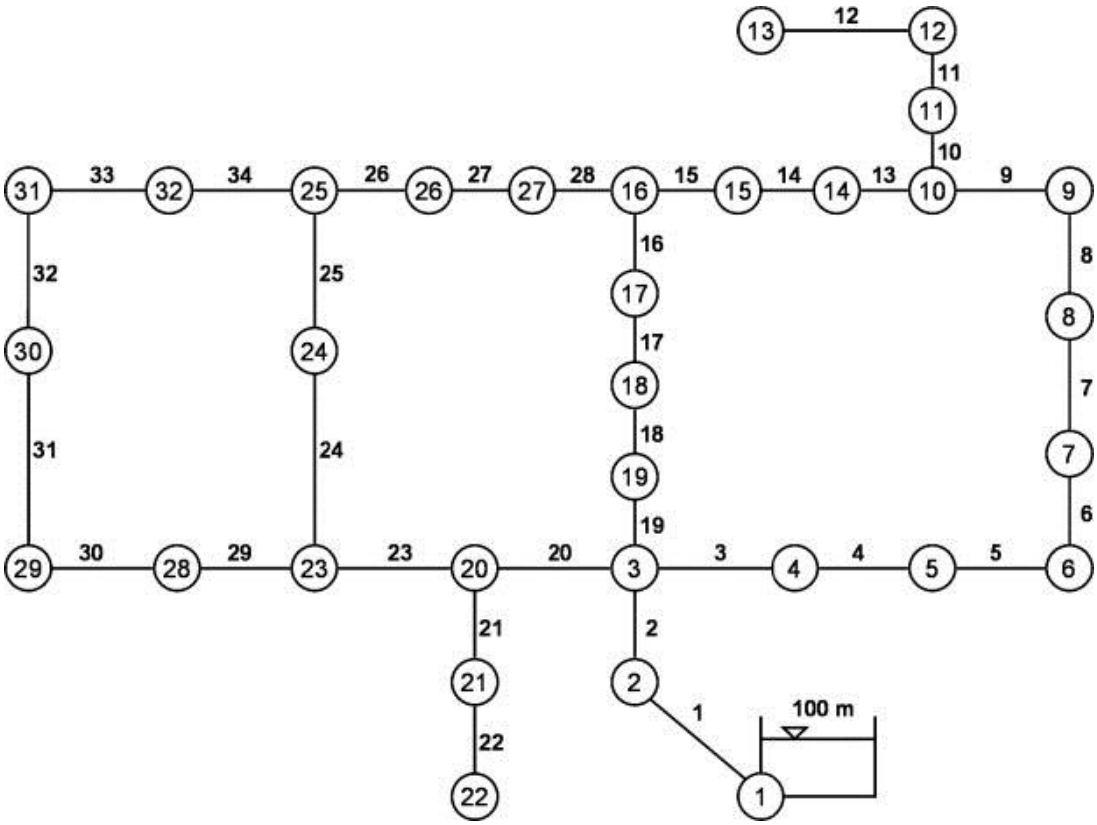

Figure 4.2 Hanoi water distribution network

Table 4.7 Lengths of pipes for Hanoi Network

Pipe number	**Pipe length (m)**	**Pipe number**	**Pipe length (m)**	**Pipe number**	**Pipe length (m)**
1	100	13	800	25	1300
2	1350	14	500	26	850
3	900	15	550	27	300
4	1150	16	2730	28	750
5	1450	17	1750	29	1500
6	450	18	800	30	2000
7	850	19	400	31	1600
8	850	20	2200	32	150
9	800	21	1500	33	860
10	950	22	500	34	950
11	1200	23	2650	–	–
12	3500	24	1230	–	–

Table 4.8 Demand of nodes for Hanoi network

Node number	Demand (L/s)	Node number	Demand (L/s)	Node number	Demand (L/s)
1	0	12	155.56	23	290.28
2	247.22	13	261.11	24	227.78
3	236.11	14	170.83	25	47.22
4	36.11	15	77.78	26	250
5	201.39	16	86.11	27	102.78
6	279.17	17	240.28	28	80.56
7	375	18	373.61	29	100
8	152.78	19	16.67	30	100
9	145.83	20	354.17	31	29.17
10	145.83	21	258.33	32	223.61
11	138.89	22	134.72	–	–

Table 4.9 Commercially available pipe diameters and their costs for Hanoi

Diameter (in.)	40	30	24	20	16	12
Diameter (mm)	1016	762	609.6	508	406.4	304.8
Cost (units/m)	278.28	180.74	129.33	98.39	70.4	45.73

4.2.2 Solution with Proposed Model by Samani and Zanganeh (2010)

As an initial trial, the method proposed by Samani and Zanganeh (2010) is used. Initial pipe diameters are selected same for all pipes and 4 different runs are made with 4 different initial pipe diameters (600 mm, 800 mm, 1000 mm and 1200 mm). All 4 solutions with different initial pipe diameters converged to the same result, infeasible. If each iteration step is examined closely, there is a trend on objective function values. In Table 4.10 it can be seen that an extreme low value is obtained in

the first iteration. Then program tries to ensure pressure constraints stay in the limits while discharge values are updated by EPANET. Cost rises in each step as the pressure values predicted by lp_solve are not confirmed by EPANET. Then lp_solve tries to solve problem with updated discharge values. At last step, program gets stuck and cannot find any solution to satisfy constraints just like the situation that happened in section 4.1.2. At that step, there is no solution to problem with the current discharge values.

Table 4.10 Value of objective function for each iteration

Iteration number	**Value of objective function (million units)**
1	5.929
2	6.098
3	6.198
4	6.299
5	6.589
6	6.963
7	7.654
8	infeasible

As it is explained in section 4.1.5 imposing no velocity limitations may lead program to make very drastic changes at early steps and this may lead to an infeasible solution. To prevent it, maximum velocity limit is changed to 10 m/s. This also was not enough for feasible solution, so minimum pressure constraint is lowered from 30 m to 29 m and 28 m. These changes forced program to get feasible solutions. Nodal pressure head values can be seen on Table 4.11 and selected pipe diameters can be seen on Table 4.12 for both *Pmin*=29 m and *Pmin*=28 m values.

For the study of Savic and Walters (1997), GA1 represents relaxed head-loss equation and GA2 represents restrictive head-loss equitation. Both are done with genetic algorithm. Cunha and Sousa (1997) used simulated annealing algorithm to solve this problem. The results of Abebe and Solomanite (1998) are gathered from the published paper of Samani and Zanganeh (2010) just for comparison.

Table 4.11 Comparison of pressure head values (m) with proposed model for Hanoi network

Node no	Savic and Walters (1997)		Abebe and Solomanite (1998)		Cunha and Sousa (1997)	Samani and Zanganeh (2010)	Proposed Model	
	GA1	GA2	GA	ACCOL			*Pmin*= 29m	*Pmin*= 28m
1	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
2	97.18	97.18	97.18	97.18	97.18	97.14	97.14	97.14
3	61.66	61.66	61.66	61.66	61.66	61.63	61.67	61.67
4	56.86	57.29	58.59	57.67	56.85	57.66	55.44	55.33
5	50.91	51.88	54.81	52.75	50.89	52.74	47.68	47.44
6	44.63	46.23	39.46	47.65	44.59	47.66	39.28	38.89
7	43.16	44.93	38.66	42.97	43.11	46.51	37.22	36.79
8	41.39	43.42	37.87	41.68	41.34	45.23	34.54	34.04
9	39.98	42.24	35.66	40.70	39.92	41.27	32.29	31.73
10	38.94	38.80	34.30	32.48	38.87	38.51	30.54	29.92
11	37.38	37.23	32.74	32.09	37.31	36.95	30.15	29.53
12	33.94	36.07	31.58	30.93	33.87	33.51	29.87	28.38
13	29.73	31.86	30.16	30.58	29.66	29.60	29.52	28.03
14	35.07	33.20	36.37	30.58	34.94	31.03	30.06	29.41
15	33.07	32.90	37.18	30.72	32.88	31.66	29.88	29.22

Table 4.11 Comparison of pressure head values (m) with proposed model for Hanoi network (continued)

Node no	Savic and Walters (1997)		Abebe and Solomanite (1998)		Cunha and Sousa (1997)	Samani and Zanganeh (2010)	Proposed Model	
	GA1	GA2	GA	ACCOL			*Pmin*= 29m	*Pmin*= 28m
16	30.17	33.02	37.64	30.77	29.81	32.81	29.74	29.07
17	30.27	40.76	48.11	46.19	29.98	40.83	29.16	28.49
18	43.92	51.15	58.61	54.41	43.82	47.89	29.1	28.24
19	55.55	58.06	60.63	60.57	55.51	56.94	49.82	49.52
20	50.41	50.66	53.87	49.22	50.46	50.77	49.89	50.15
21	41.06	41.31	44.51	47.92	41.10	41.42	40.54	40.8
22	35.87	36.12	44.08	47.86	35.92	36.24	35.38	35.63
23	44.17	44.63	39.84	41.96	44.26	44.86	43.2	43.67
24	38.85	39.56	30.56	40.18	38.52	41.13	39.6	40.17
25	35.50	36.41	30.54	38.95	34.81	35.01	33.76	34.56
26	31.47	32.94	32.16	36.01	30.89	31.83	31.55	29.84
27	30.04	32.19	32.64	35.93	29.60	31.36	30.67	29.23
28	35.43	36.01	33.55	36.48	38.59	37.92	38.37	39.13
29	30.67	31.38	31.49	36.45	29.66	32.40	33.92	35.02
30	29.67	30.48	30.48	36.55	29.92	30.65	31.96	30.73
31	30.13	30.96	30.43	36.65	30.20	30.37	31.41	30.27
32	31.37	32.25	30.21	36.76	32.66	30.00	29.22	28.51

Table 4.12 Comparison of diameters (in.) and total costs with proposed model for Hanoi network

Pipe no	Savic and Walters (1997)		Abebe and Solomanite (1998)		Cunha and Sousa (1997)	Samani and Zanganeh (2010)	Proposed Model	
	GA1	GA2	GA	ACCOL			*Pmin*= 29m	*Pmin*= 28m
1	40	40	40	40	40	40	40	40
2	40	40	40	40	40	40	40	40
3	40	40	40	40	40	40	40	40
4	40	40	40	40	40	40	40	40
5	40	40	30	40	40	40	40	40
6	40	40	40	30	40	40	40	40
7	40	40	40	40	40	40	40	40
8	40	40	30	40	40	30	40	40
9	40	30	30	24	40	30	40	40
10	30	30	30	40	30	30	40	40
11	24	30	30	30	24	24	40	30
12	24	24	30	40	24	24	40	40
13	20	16	16	16	20	12	40	40
14	16	16	24	16	16	12	40	40
15	12	12	30	30	12	16	40	40
16	12	16	30	12	12	20	40	40
17	16	20	30	20	16	24	40	30
18	20	24	40	24	20	24	12	12
19	20	24	40	30	20	24	12	12
20	40	40	40	40	40	40	40	40
21	20	20	20	30	20	20	20	20
22	12	12	20	30	12	12	12	12

Table 4.12 Comparison of diameters (in.) and total costs with proposed model for Hanoi network (continued)

Pipe no	Savic and Walters (1997)		Abebe and Solomanite (1998)		Cunha and Sousa (1997)	Samani and Zanganeh (2010)	Proposed Model	
	GA1	GA2	GA	ACCOL			*Pmin*= 29m	*Pmin*= 28m
23	40	40	30	40	40	40	40	40
24	30	30	16	40	30	30	30	30
25	30	30	20	40	30	24	24	24
26	20	20	12	24	20	20	24	20
27	12	12	24	30	12	12	16	16
28	12	12	20	12	12	12	12	12
29	16	16	24	16	16	20	24	24
30	16	16	30	40	12	20	24	24
31	12	12	30	16	12	20	24	20
32	12	12	30	20	16	12	16	16
33	16	16	30	30	16	12	16	16
34	20	20	12	24	24	16	12	12
Cost (million units)	**6.073**	**6.195**	**7.000**	**7.800**	**6.056**	**6.220**	**8.437**	**8.073**

As it can be seen on Table 4.12 the calculated objective functions are 8.437 million units for *Pmin*= 29 m and 8.073 million units for *Pmin*= 28 m. These values are high if we compare them with the previous studies and also they are different from the objective function that Samani and Zanganeh (2010) have claimed to obtain. Moreover, it should be noted that Samani and Zanganeh have managed to get a result with a nodal pressure below 30 m. Pressure at Node 13 is found as 29.60 m in

their study. It is quite impossible for linear programming technique to find a feasible solution by violating constraints. Even though they claim that 30 m pressure limit is strictly used.

After further investigation of solution steps, it is seen that program is get stuck at local minimum and cannot improve the solution. This flaw of solution procedure is already examined in previous sections from 4.1.2 to 4.1.5. Detailed information about the solution of Samani and Zanganeh (2010) (such as initial pipe diameters) could not be found on the paper they published so Samani is directly contacted via E-mail. He stated that they only used method mentioned in the paper and the initial pipe diameters have no significance on the solution of the problem. However, the results obtained in this paper are significantly different from Samani and Zanganeh's even though exact same solution procedure is used.

4.2.3 Solution with Improved Model

The proposed method has failed to give feasible solution for original problem and it also has given unsatisfactory results for lowered minimum pressure head values. As it is proven in section 4.1.5, the program tends to give better result when the maximum velocity limit is lowered. For Hanoi network, the maximum velocity limit cannot be selected lower than *Vmax*= 7 m/s because main pipes' (pipe 1 and 2) velocity can go as low as 6.83 m/s even when the biggest diameter is selected for them. By considering this, an improvement on proposed model has been made. With that improvement, user can define different velocity limits for specific pipes. After numerous tests, best results are obtained with *Vmax*= 7 m/s for pipe 1 and pipe 2, *Vmax*= 3 m/s for other pipes. Selected pipe diameters can be seen on Table 4.14 and nodal pressure values can be seen on Table 4.13. Tests that are done with initial diameters 600 mm, 800 mm, 1000 mm and 1200 mm have given the same results. Improved model is also tested with *Pmin*= 30 m (as original problem), *Pmin*= 29 m and *Pmin*= 28 m.

Table 4.13 Comparison of pressure values (m) with improved model for different *Pmin* values for Hanoi network

Node no	**Minimum allowable pressure (*Pmin*)**		
	28 m	**29 m**	**30 m**
1	100.00	100.00	100.00
2	97.14	97.14	97.14
3	61.67	61.67	61.67
4	57.06	57.08	57.09
5	51.34	51.39	51.41
6	45.34	45.41	45.44
7	43.94	44.02	44.05
8	42.28	42.38	42.42
9	40.97	41.08	41.12
10	37.08	37.23	40.18
11	35.52	35.67	38.62
12	34.36	34.51	35.19
13	30.16	30.30	30.98
14	34.10	34.34	31.74
15	29.49	30.08	30.72
16	28.62	29.37	30.07
17	34.14	34.47	34.79
18	53.29	53.34	53.39
19	58.79	58.81	58.83
20	50.49	50.41	50.37
21	41.14	41.07	41.02
22	35.97	35.90	35.86
23	44.29	44.16	44.09
24	35.62	35.58	40.85
25	31.33	31.35	35.87

Table 4.13 Comparison of pressure values (m) with improved model for different *Pmin* values for Hanoi network (continued)

Node no	**Minimum allowable pressure (*Pmin*)**		
	28 m	**29 m**	**30 m**
26	29.71	29.67	31.69
27	28.42	29.29	30.04
28	39.07	38.69	39.39
29	34.17	33.52	35.08
30	31.93	31.12	33.21
31	31.26	30.36	32.70
32	28.45	29.28	30.70

Table 4.14 Comparison of diameters (in.) and total costs with improved model for different *Pmin* values for Hanoi network

Pipe no	**Minimum allowable pressure (*Pmin*)**		
	28 m	**29 m**	**30 m**
1	40	40	40
2	40	40	40
3	40	40	40
4	40	40	40
5	40	40	40
6	40	40	40
7	40	40	40
8	40	40	40
9	30	30	40
10	30	30	30
11	30	30	24

Table 4.14 Comparison of diameters (in.) and total costs with improved model for different *Pmin* values for Hanoi network (continued)

Pipe no	**Minimum allowable pressure (*Pmin*)**		
	28 m	**29 m**	**30 m**
12	24	24	24
13	20	20	16
14	12	12	16
15	12	12	12
16	12	12	12
17	16	16	16
18	24	24	24
19	24	24	24
20	40	40	40
21	20	20	20
22	12	12	12
23	40	40	40
24	24	24	30
25	24	24	24
26	24	24	20
27	12	16	12
28	12	12	12
29	24	24	24
30	24	24	24
31	24	24	24
32	16	16	16
33	16	20	16
34	12	12	12
Cost (million units)	**6.271**	**6.302**	**6.314**

With the new improved method, solution for Hanoi network has improved significantly. For *Pmin*= 28 m, total cost is improved from 8.073 to 6.271 million units and for *Pmin*= 29 m total cost is improved from 8.437 to 6.302 million units. Also with that improvement, the optimization problem can be solved for original problem where *Pmin*= 30 m and the value of objective function is calculated as 6.314 million units. This value is close or even better than some previous studies (Table 4.12). Whole solution procedure is done in 14 iterations and it only takes 6 seconds with quad-core 3.4 GHz processor. Savic and Walters (1997) has spent approximately 60 hours of computation time with their genetic algorithm method just to get one solution on Hanoi network. Even though computers of today are significantly faster than 20 years ago, still 6 seconds of computation time is a noteworthy improvement.

Extra network information about results for *Pmin*= 30 m can be found on Table 4.15.

Table 4.15 Pipe related results of improved model for *Pmin*= 30 m

Pipe no	Length (m)	Diameter (mm)	Discharge (L/s)	Velocity (m/s)
1	100	1016	5538.90	6.83
2	1350	1016	5291.68	6.53
3	900	1016	2182.00	2.69
4	1150	1016	2145.89	2.65
5	1450	1016	1944.50	2.40
6	450	1016	1665.33	2.05
7	850	1016	1290.33	1.59
8	850	1016	1137.55	1.40
9	800	1016	991.72	1.22
10	950	762	555.56	1.22
11	1200	609.6	416.67	1.43
12	3500	609.6	261.11	0.89

Table 4.15 Pipe related results of improved model for *Pmin*= 30 m (continued)

Pipe no	Length (m)	Diameter (mm)	Discharge (L/s)	Velocity (m/s)
13	800	406.4	290.33	2.24
14	500	406.4	119.50	0.92
15	550	304.8	41.72	0.57
16	2730	304.8	51.30	0.70
17	1750	406.4	291.58	2.25
18	800	609.6	665.19	2.28
19	400	609.6	681.86	2.34
20	2200	1016	2191.71	2.70
21	1500	508	393.05	1.94
22	500	304.8	134.72	1.85
23	2650	1016	1444.49	1.78
24	1230	762	716.18	1.57
25	1300	609.6	488.40	1.67
26	850	508	345.87	1.71
27	300	304.8	95.87	1.31
28	750	304.8	6.91	0.09
29	1500	609.6	438.03	1.50
30	2000	609.6	357.47	1.22
31	1600	609.6	257.47	0.88
32	150	406.4	157.47	1.21
33	860	406.4	128.30	0.99
34	950	304.8	95.31	1.31

4.3 Two-loop Network

4.3.1 Problem Definition of Two-loop Network

The last example studied in this thesis is two-loop one reservoir water distribution network. This network has been studied by many researchers. Alperovits and Shamir (1977), Quindry et al. (1981), Fujiwara et al. (1987), Kessler and Shamir (1989), Bhave and Sonak (1992), Sonak and Bhave (1993), Savic and Walters (1997) and Samani and Mottaghi (2006) have studied this network. As it can be seen on Figure 4.3 there is one reservoir feeding the system with 210 m constant head and total number of 8 pipes arranged in 2 loops.

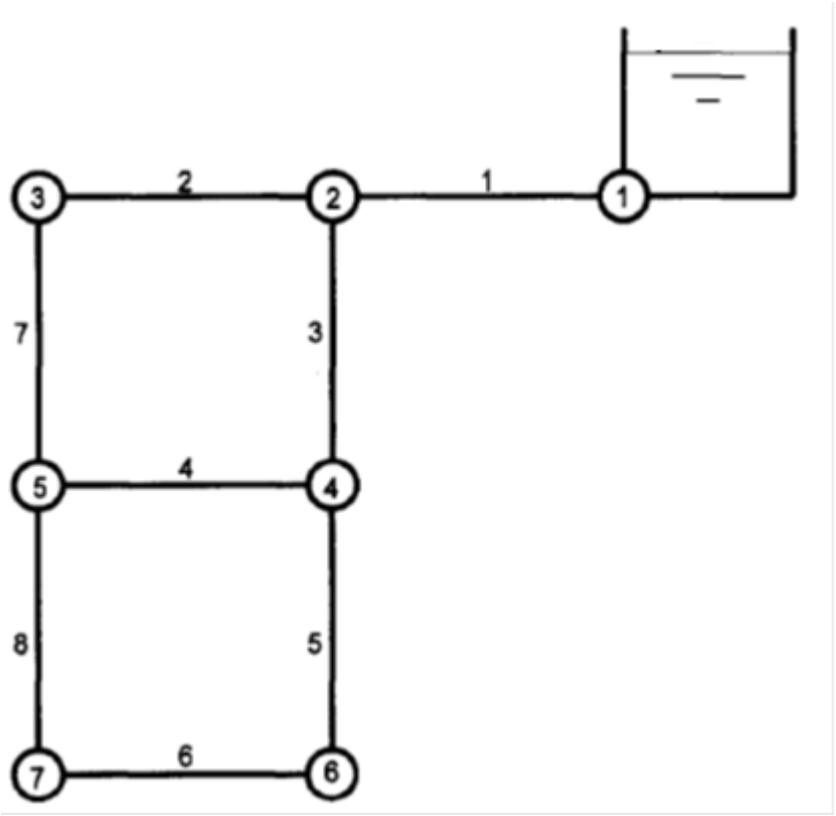

Figure 4.3 Two loop network

The node data for this network, node elevations and demands, are summarized in Table 4.16. Available pipe diameters and their unit costs can be found on Table 4.17. The minimum acceptable pressure requirement is selected as 30 m for all nodes. There is no upper pressure limit or any restriction on flow velocity. All pipes are 1000 m long and they have same Hazen-Williams roughness coefficient of 130.

Table 4.16 Elevations and demands for nodes for Two-loop Network

Node number	**Elevation (m)**	**Demand (L/s)**
1	210	-311
2	150	28
3	160	28
4	155	33
5	150	75
6	165	92
7	160	55

Table 4.17 Available pipe diameters and their unit costs for Two-loop Network

Diameter (in.)	**Cost (units/m)**	**Diameter (in.)**	**Cost (units/m)**
1	2	12	50
2	5	14	60
3	8	16	90
4	11	18	130
6	16	20	170
8	23	22	300
10	32	24	550

4.3.2 Solution of Two-loop Network

Many tests and many runs have been made with different parameters and different initial pipe diameters. However, all runs converged to the same solution. Results including pipe related data can be seen on Table 4.18 and results including node related data can be seen on Table 4.19. Improved model, which gives user the ability to assign different velocity limits for specific pipes, has not developed a better solution for this case. It has given same results as the proposed model. Comparison of the results with previous studies is summarized in Table 4.20. If total costs are compared, solution obtained from this study has not been an improvement.

As previously mentioned in Section 4.1.2 both proposed and improved model have to select minimum diameter for 1 pipe for each loop when there is no velocity limits. This is a big downside of the solution procedure. This situation also occurred in this problem and 2 pipe diameters are selected as minimum (pipe no 7 and 8). However, Samani and Mottaghi (2006) have managed to deal with this problem, somehow. Even though the same solution procedure is used in this study, their result contains only 1 minimum diameter for whole system. This situation is quite impossible since LP optimization selects the parameter with minimum cost when there is no constraint related with specific decision variable.

Table 4.18 Pipe related results for Two-loop Network

Pipe no	Diameter (mm)	Diameter (in.)	Discharge (L/s)	Velocity (m/s)	Headloss (m)
1	457.2	18	311.00	1.89	6.75
2	203.2	8	28.20	0.87	4.11
3	457.2	18	254.80	1.55	4.67
4	254	10	74.75	1.48	8.43
5	406.4	16	147.05	1.13	2.99
6	254	10	55.05	1.09	4.79
7	25.4	1	0.20	0.40	8.99
8	25.4	1	0.05	0.09	0.66

Table 4.19 Node related results for Two-loop Network

Node no	Elevation (m)	Demand (L/s)	Head (m)	Pressure (m)
1	210	-311	210.00	0.00
2	150	28	203.25	53.25
3	160	28	199.14	39.14
4	155	33	198.59	43.59
5	150	75	190.15	40.15
6	165	92	195.59	30.59
7	160	55	190.81	30.81

Table 4.20 Result comparison with previous studies for Two-loop Network

	Diameter (in.)		
Pipe no	Savic and Walters (1997)	Samani and Mottaghi (2006)	Proposed Model
1	18	18	18
2	10	10	8
3	16	16	18
4	4	4	10
5	16	16	16
6	10	10	10
7	10	10	1
8	1	1	1
Cost (units)	419000	419000	441000

CHAPTER 5

CONCLUSION

5.1 Summary of the Research

In this study, optimization of water distribution networks is performed by mixed integer linear programming method combined with an iterative procedure. The objective function is determined as total initial cost which can be summarized as summation of pipe, reservoir and pump costs. The parameters subjected to optimization are pipe diameters, reservoir heights and pump characteristic. All these parameters should be discrete in the problem.

The optimization process also ensures to satisfy nodal demands, nodal pressure limits and pipe velocity limits. Linear programming method also ensures that created sub-problems are always optimized to global minimum.

5.2 Conclusion and Comments

The optimization method that originally proposed by Samani and Zanganeh (2010) is coded into Java based computer program. Hydraulic solver (EPANET) and linear programming solver (lp_solve) are implemented into the program.

The proposed method is tested on three different water distribution networks. The results are not promising as discussers have foreseen. The solution procedure tends to get stuck on local minimum before getting close to the global optimum value. The main reason behind it is quite clear. The program cannot predict the change in discharge values as it operates hydraulic solver and linear programming solver successively. The linear programming solver always finds the global optimum for

the created sub-problem; however these problems are always created with the previously calculated discharge values. So, getting global optimum for sub-problems do not have any significance when the sub-problems cannot represent the real problem properly. Also this does not ensure to find the global optimum for the real problem.

There is one more downside of the proposed method. One pipe diameter for each loop is selected by considering only velocity limits, due to the nature of algorithm. This becomes a huge problem when there is no velocity limit in the problem. In this case, program just selects the diameter with minimum cost for one pipe for each loop.

The proposed model is improved by giving the program the ability to assign different velocity limits for different pipes. By doing that, the optimization algorithm is prevented from selecting very small diameters for specific pipes. This allowed program to achieve optimum value with smaller steps. This ensures smaller discharge changes in each iteration step. The result obtained with this method is significantly improved the result obtained by proposed method. Moreover, it is quite close to the values that are achieved with meta-heuristic algorithms such as genetic algorithm.

The proposed method has one big advantage against meta-heuristic methods. Iteration number and time it requires to converge are significantly low. It only requires 14 iterations for Hanoi network which has 34 pipes and 3 loops. It takes roughly 6 seconds to get result with 3.4 GHz quad core processor for this specific network. Same network has been solved with genetic algorithm by Savic and Walters (1997) and it has taken approximately 60 hours just for computation. Even though today's computers are significantly faster than 20 years ago, still 60 hours to 6 seconds is such a big improvement.

To sum up, the solution algorithm tested in this research is not a significant improvement if it is compared with the previous deterministic methods for optimization. It has the same weaknesses as its predecessors. However, it may be

quite useful for big and complex networks since it requires very little computation time.

REFERENCES

Abebe, A. J. and Solomanite, D. P., (1998). "Application of global optimization to the design of pipe networks", *3rd International Conference on Hydroinformatics*, Copenhagen, Denmark, pp. 989–995.

Alperovits, E. and Shamir, U., (1977). "Design of optimal water distribution system", *Water Resources Research* **13**(6), pp. 885-900.

Bhave, P. R. and Sonak, Y. Y., (1992). "A critical study of the linear programming gradient method of optimal design of water supply networks", *Water Resources Research* **28**(6), pp. 1577-1584.

Cunha, M. C. and Ribeiro, L., (2004). "Tabu search algorithms for water network optimization", *European Journal of Operatinal Research* **157**(3), pp. 746-758.

Cunha, M. C. and Sousa, J., (1999). "Water distribution network design optimization: Simulated annealing approach", *Journal of Water Resources Planning and Management* **125**(4), pp. 215-221.

Cunha, M. C. and Sousa, J., (2001). "Hydraulic infrastructures design using simulated annealing", *Journal of Infrastructure Systems* **7**(1), pp. 32-39.

Dandy, G. C., Simpson, A. R. and Murphy, L. J., (1996). "An improved genetic algorithm for pipe network optimization", *Water Resources Research* **32**(2), pp. 449-458.

Fujiwara, O. and Khang, D. B., (1990). “A two-phase decomposition method for optimal design of looped water distribution networks”, *Water Resources Research* **26**(4), pp. 539-549.

Fujiwara, O., Jenchaimahakoon, B. and Edirisinghe, N. C. P., (1987). “A modified linear programming gradient method for optimal design of looped water distribution networks”, *Water Resources Research* **23**(6), pp. 977-982.

Geem, Z. W., Kim, J. H. and Loganathan, G. V., (2001). “A new heuristic optimization algorithm: Harmony search”, *Simulation* **76**(2), pp. 60-68.

Goldberg, D. E. and Kuo, C. H., (1987). “Genetic algorithms in pipeline optimization”, *Journal of Computing in Civil Engineering* **1**(2), pp. 128-141.

Hoag, L.N. and Weinberg, G., (1957). “Pipeline network analysis by electronic digital computer”, *Journal of the American Water Works Association* **49**(5), pp. 517-524.

Jesperson, K., (2001). “A brief history of drinking water distribution.” *On Tap*, pp. 18-46

Kessler, A. and Shamir, U., (1989). “Analysis of the linear programming gradient method for optimal design of water supply networks”, *Water Resources Research* **25**(7), pp. 1469-1480.

Liong, S. and Atiquzzaman, M., (2004). “Optimal design of water distribution network using shuffled complex evolution”, *Journal of Instrumentation Engineering,* **44**(1), pp. 93–107.

Martínez, J., (2008). “Discussion of ‘Optimization of water distribution networks using integer linear programming’ by Hossein M. V. Samani and Alireza Mottaghi”, *Journal of Hydraulic Engineering* **134**(7), pp. 1023-1024.

Murphy, L. J., Simpson, A. R. and Dandy, G. C., (1993). “Pipe network optimization using an improved genetic algorithm”, *Specialty Conference*

of Water Resources Planning and Management Division, American Society of Civil Engineers, Seattle, Washington, USA.

Quindry, G., Brill, E. and Liebman, J., (1981). "Optimization of looped water distribution systems", *Journal of Environmental Engineering* **107**(4), pp. 665-679.

Samani, H. M. V. and Mottaghi, A., (2006). "Optimization of water distribution networks using integer linear programming", *Journal of Hydraulic Engineering* **132**(5), pp. 501-509.

Samani, H. M. V. and Zanganeh, A., (2010). "Optimisation of water networks using linear programming", *Proceedings of the Institution of Civil Engineers - Water Management*, **163**(9), pp. 475-485.

Savic, D. and Cunha, M. C., (2008). "Discussion of 'Optimization of water distribution networks using integer and linear programming' by Hossein M. V. Samani and Alireza Mottaghi", *Journal of Hydraulic Engineering* **134**(7), pp. 1024-1025.

Savic, D. and Walters, G., (1997). "Genetic algorithms for least-cost design of water distribution networks", *Journal of Water Resources Planning and Management* **123(**2), pp. 67-77.

Schaake, J. and Lai, D., (1969). "Linear programming and dynamic programming applications to water distribution network design", Research Report No: 116, Department of Civil Engineering, Massachusetts Institute of Technology, USA.

Simpson, A. R., Dandy, G. C. and Murphy, L. J., (1994). "Genetic algorithms compared to other techniques for pipe optimization", *Journal of Water Resources Planning and Management* **120**(4), pp. 423-443.

Sonak, V. V. and Bhave, P. R., (1993). "Global optimum tree solution for single-source looped water distribution networks subjected to a single loading pattern", *Water Resources Research* **29**(7), pp. 2437-2443.

Templeman, A., (1982). "Discussion of 'Optimization of looped water distribution systems' by Quindry et al.", *Journal of Environmental Engineering* **108**(3), pp. 599-602.

Varma, K. V. K., Narasimhan, S. and Bhallamudi, S. M., (1997). "Optimal design of water distribution systems using an NLP method", *Journal of Environmental Engineering* **123**(4), pp. 381-388.

Walski, T.M., Chase, D.V., Savic, D., Grayman, W. M., Beckwith, S., Koelle,E., (2003). "Advanced Water Distribution Modeling and Management", *Civil and Environmental Engineering and Engineering Mechanics Faculty Publications*, paper 18.

Walters, G. A., Halhal, D., Savic, D. and Quazar, D., (1999). "Improved design of 'Anytown' distribution network using structured messy genetic algorithm", *Urban Water* **1**(1), pp. 23-28.

APPENDIX: DEVELOPED PROGRAM

```
package optimization;

import java.io.File;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.Set;
import java.util.logging.Logger;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import lpsolve.LpSolve;
import lpsolve.LpSolveException;

import org.addition.epanet.hydraulic.HydraulicSim;
import org.addition.epanet.hydraulic.io.AwareStep;
import org.addition.epanet.hydraulic.io.HydraulicReader;
import org.addition.epanet.network.Network;
import org.addition.epanet.network.PropertiesMap;
import org.addition.epanet.network.io.input.InpParser;
import org.addition.epanet.network.io.output.InpComposer;
import org.addition.epanet.network.structures.Link;
import org.addition.epanet.network.structures.NUConvert;
import org.addition.epanet.network.structures.Node;
import org.addition.epanet.util.ENException;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

public class Solver {

        public static void main(String[] args) {

                Configuration conf;
                Network net;
                EpaNetResult simResult;
```

```
		//Read Configuration File
		conf = parseConfigurationFile();

		final List<Integer> reservoirIds = new ArrayList<Integer>();
		reservoirIds.addAll(conf.getReservoirIds());

		//Parse Network
		net = parseNetworkFile(conf.getInputFileLocation());


		// Main Loop
		int iterationNo = 1;
		Queue<double[]> prevPtrVarsList = new
LimitedQueue<double[]>(conf.getStopSteps());
		double[] ptrVars = null;
		int runCounter = 0;

		do{
			// Simulate with EPANet and Gather Analysis Results
			simResult = simulateEpa(net, reservoirIds);
			//simResult.prettyPrint();

			// Solve the LP Problem with LPSolve
			ptrVars = lPSolve(simResult, conf, iterationNo);

			// Update the Network with Solution Values
			updateNetwork(net, ptrVars, simResult, conf);

			composeNetworkFile(net,
conf.getOutputFileLocation()+"_"+iterationNo+".inp");
			iterationNo++;

			//Compose solution network to file
			composeNetworkFile(net, conf.getOutputFileLocation());
			net = parseNetworkFile(conf.getOutputFileLocation());

			runCounter++;

		//Check if last solution is contained in last
[StopIfNoImprovementSteps]
		}while(!addToListAndCheckContains(prevPtrVarsList, ptrVars));

		System.out.println(String.format("\t\t\t\t\t\t\t\t\t\t\t\t\t<<< Run is
finished after %d iterations.. >>>", runCounter));
```

```
	}


	private static boolean addToListAndCheckContains(Queue<double[]> prevPtrVarsList, double[] ptrVars) {

		Iterator<double[]> ptrIter = prevPtrVarsList.iterator();
		while(ptrIter.hasNext()){
			double[] currentElement = ptrIter.next();
			boolean equals = true;
			for(int i=0; i<currentElement.length; i++)
				if(currentElement[i] != ptrVars[i])
					equals = false;
			if(equals)
				return true;
		}

		prevPtrVarsList.add(ptrVars);
		return false;
	}


	private static void composeNetworkFile(Network net, String outputFileLocation) {
		InpComposer comp = new InpComposer();
		try {
			comp.composer(net, new File(outputFileLocation));
		} catch (ENException e) {
			System.out.println("Network compose failed..");
			e.printStackTrace();
		}
	}


	private static void updateNetwork(Network net, double[] ptrVars, EpaNetResult simResult, Configuration conf) {
		int count = 0;
		for(NetLink ll : simResult.getLinks()){
			for(Double dia : conf.getAvailablePipeDiameters().keySet()){
				if(ptrVars[count] == 1)
					for(Link l : net.getLinks())
						if(Integer.parseInt(l.getId())==ll.getId())

	l.setNUDiameter(PropertiesMap.UnitsType.SI, dia/1000);
					count++;
			}
```

```
                }

                for(NetReservoir rr : simResult.getReservoirs()){
                        for(Double ele :
conf.getAvailableReservoirElevations().keySet()){
                                if(ptrVars[count] == 1)
                                        for(Node n : net.getNodes())

        if(rr.getId()==Integer.parseInt(n.getId()))

        n.setElevation(NUConvert.convertDistance(PropertiesMap.UnitsType.SI,
ele));
                                        count++;
                                }
                        }
                }


        private static double[] lPSolve(EpaNetResult simResult, Configuration conf,
int iterationNo) {

                double[] ptrVars = null;
                try {
                        // Create the problem
                        int linkPipe = simResult.getLinks().size() *
conf.getAvailablePipeDiameters().size();
                        int reservoirElevation = simResult.getReservoirs().size() *
conf.getAvailableReservoirElevations().size();
                        int variableNo = linkPipe + reservoirElevation +
simResult.getNodes().size();

                        LpSolve solver = LpSolve.makeLp(0, variableNo);

                        //Set variables to be Integers
                        int count=1;
                        for(int i=0; i<linkPipe; i++){
                                //solver.setInt(count, true);
                                solver.setBinary(count, true);
                                count++;
                        }
                        for(int i=0; i<reservoirElevation; i++){
                                //solver.setInt(count, true);
                                solver.setBinary(count, true);
                                count++;
                        }
```

```
                        // Add constraints
                        List<Constraint> cons = generateConstraints(simResult, conf);
                        for(Constraint c : cons)
                                solver.strAddConstraint(c.getConsStr(),  c.getType(),
c.getRhs());

                        // Set Objective function
                        String objFuncStr = constructObjFunct(simResult.getLinks(),
conf.getAvailablePipeDiameters(),  simResult.getReservoirs(),
conf.getAvailableReservoirElevations(), simResult.getNodes());
                        solver.strSetObjFn(objFuncStr);
                        solver.setMinim();

                        // Write constructed problem to file in LP format
                        solver.writeLp(String.format("resources/lp_problem%d.txt",
iterationNo));
                        // Solve the problem
                        solver.solve();

                        // Print solution
                        System.out.println("Iteration No: " + iterationNo);
                        System.out.println("Value  of  objective  function:  "  +
solver.getObjective());
                        double[] var = solver.getPtrVariables();
                        for (int i = 0; i < var.length; i++) {
                                System.out.println("Value  of  var["  +  i  +  "]  =  "  +
var[i]);
                        }
                        System.out.println();

                        ptrVars = solver.getPtrVariables();

                        // Delete the problem and free memory
                        if(solver.getLp()!=0)
                                solver.deleteLp();
                }
                catch (LpSolveException e) {
                        System.out.println("LP Solve Failed..");
                        e.printStackTrace();
                }

                // Return solution set
                return ptrVars;
        }
```

```
	private static List<Constraint> generateConstraints(EpaNetResult simResult,
	Configuration conf) {
		List<Constraint> result = new ArrayList<Constraint>();
		result.add(generateReservoirConstraints(simResult, conf));
		result.addAll(generateNodePressureConstraints(simResult, conf));
		result.addAll(generatePressureLimitConstraints(simResult, conf));
		result.addAll(generateVelocityConstraints(simResult, conf));
		result.addAll(generateIntegerConstraints(simResult, conf));
		return result;
	}


	private static List<Constraint> generateIntegerConstraints(EpaNetResult
simResult, Configuration conf) {
		List<Constraint> result = new ArrayList<Constraint>();

		for(NetLink l : simResult.getLinks()){
			Constraint cons = new Constraint();

			//Constraint string construction
			String consStr = new String();

			for(NetLink nl : simResult.getLinks()){
				for(@SuppressWarnings("unused") Double dial :
conf.getAvailablePipeDiameters().keySet()){
					if(nl.getId() == l.getId())
						consStr += "1 ";
					else
						consStr += "0 ";
				}
			}

			for(@SuppressWarnings("unused") NetReservoir rr :
simResult.getReservoirs()){
				for(@SuppressWarnings("unused") Double ele :
conf.getAvailableReservoirElevations().keySet())
					consStr += "0 ";
			}

			for(@SuppressWarnings("unused") NetNode nn :
simResult.getNodes())
				consStr += "0 ";

			consStr = consStr.substring(0, consStr.length()-1);
```

```
                cons.setConsStr(consStr);
                cons.setRhs(1);
                cons.setType(LpSolve.EQ);
                result.add(cons);

            }


            for(@SuppressWarnings("unused") NetReservoir r : simResult.getReservoirs()){
                Constraint cons = new Constraint();

                //Constraint string construction
                String consStr = new String();

                for(@SuppressWarnings("unused") NetLink nl : simResult.getLinks())
                    for(@SuppressWarnings("unused") Double dial : conf.getAvailablePipeDiameters().keySet())
                        consStr += "0 ";

                for(@SuppressWarnings("unused") NetReservoir rr : simResult.getReservoirs()){
                    for(@SuppressWarnings("unused") Double ele : conf.getAvailableReservoirElevations().keySet())
                        consStr += "1 ";
                }

                for(@SuppressWarnings("unused") NetNode nn : simResult.getNodes())
                    consStr += "0 ";

                consStr = consStr.substring(0, consStr.length()-1);


                cons.setConsStr(consStr);
                cons.setRhs(1);
                cons.setType(LpSolve.EQ);
                result.add(cons);
            }



            return result;
        }
```

```
	private static List<Constraint> generateVelocityConstraints(EpaNetResult simResult, Configuration conf) {
		List<Constraint> result = new ArrayList<Constraint>();

		//Minimum Velocity
		for(NetLink l : simResult.getLinks()){

			Constraint cons = new Constraint();

			//Constraint string construction
			String consStr = new String();

			for(NetLink nl : simResult.getLinks()){
				for(Double dial : conf.getAvailablePipeDiameters().keySet()){
					if(nl.getId() == l.getId())
						consStr += Math.abs(calculateVelocity(nl.getFlow(), dial)) + " ";
					else
						consStr += "0 ";
				}
			}

			for(@SuppressWarnings("unused") NetReservoir rr : simResult.getReservoirs()){
				for(@SuppressWarnings("unused") Double ele : conf.getAvailableReservoirElevations().keySet())
					consStr += "0 ";
			}

			for(@SuppressWarnings("unused") NetNode nn : simResult.getNodes())
				consStr += "0 ";

			consStr = consStr.substring(0, consStr.length()-1);

			cons.setConsStr(consStr);
			LinkVelocityConstraint lvc = conf.getLinkVelocityConstraint(l.getId());
			cons.setRhs(lvc==null ? conf.getNetworkConstraints().get("MinVelocity") : lvc.getMinVelocity());
			cons.setType(LpSolve.GE);
			result.add(cons);
```

```
		}
		//Maximum Velocity
		for(NetLink l : simResult.getLinks()){

			Constraint cons = new Constraint();

			//Constraint string construction
			String consStr = new String();


			for(NetLink nl : simResult.getLinks()){
				for(Double dial :
conf.getAvailablePipeDiameters().keySet()){
					if(nl.getId() == l.getId())
						consStr +=
Math.abs(calculateVelocity(nl.getFlow(), dial)) + " ";
					else
						consStr += "0 ";
				}
			}

			for(@SuppressWarnings("unused") NetReservoir rr :
simResult.getReservoirs()){
				for(@SuppressWarnings("unused") Double ele :
conf.getAvailableReservoirElevations().keySet())
					consStr += "0 ";
			}
			for(@SuppressWarnings("unused") NetNode nn :
simResult.getNodes())
				consStr += "0 ";

			consStr = consStr.substring(0, consStr.length()-1);

			cons.setConsStr(consStr);
			LinkVelocityConstraint lvc =
conf.getLinkVelocityConstraint(l.getId());
			cons.setRhs(lvc==null ?
conf.getNetworkConstraints().get("MaxVelocity") : lvc.getMaxVelocity());
			cons.setType(LpSolve.LE);
			result.add(cons);
		}

		return result;
	}
```

```
	private static List<Constraint> generatePressureLimitConstraints(EpaNetResult simResult, Configuration conf) {
		List<Constraint> result = new ArrayList<Constraint>();

		//Minimum Pressure
		for(NetNode n : simResult.getNodes()){
			Constraint cons = new Constraint();

			//Constraint string construction
			String consStr = new String();

			for(@SuppressWarnings("unused") NetLink nl : simResult.getLinks())
				for(@SuppressWarnings("unused") Double dia : conf.getAvailablePipeDiameters().keySet())
					consStr += "0 ";

			for(@SuppressWarnings("unused") NetReservoir rr : simResult.getReservoirs()){
				for(@SuppressWarnings("unused") Double ele : conf.getAvailableReservoirElevations().keySet())
					consStr += "0 ";
			}

			for(NetNode nn : simResult.getNodes()){
				if(nn.getId() == n.getId())
					consStr += "1 ";
				else
					consStr += "0 ";
			}

			consStr = consStr.substring(0, consStr.length()-1);

			cons.setConsStr(consStr);

	cons.setRhs(conf.getNetworkConstraints().get("MinNodalPressure"));
			cons.setType(LpSolve.GE);
			result.add(cons);
		}

		//Maximum Pressure
		for(NetNode n : simResult.getNodes()){
			Constraint cons = new Constraint();

			//Constraint string construction
			String consStr = new String();
```

```
                for(@SuppressWarnings("unused") NetLink nl :
simResult.getLinks())
                        for(@SuppressWarnings("unused") Double dia :
conf.getAvailablePipeDiameters().keySet())
                                consStr += "0 ";

                for(@SuppressWarnings("unused") NetReservoir rr :
simResult.getReservoirs()){
                        for(@SuppressWarnings("unused") Double ele :
conf.getAvailableReservoirElevations().keySet())
                                consStr += "0 ";
                }
                for(NetNode nn : simResult.getNodes()){
                        if(nn.getId() == n.getId())
                                consStr += "1 ";
                        else
                                consStr += "0 ";
                }

                consStr = consStr.substring(0, consStr.length()-1);

                cons.setConsStr(consStr);
        cons.setRhs(conf.getNetworkConstraints().get("MaxNodalPressure"));
                cons.setType(LpSolve.LE);
                result.add(cons);
        }

        return result;
    }

    private static List<Constraint>
generateNodePressureConstraints(EpaNetResult simResult, Configuration conf) {
        List<Constraint> result = new ArrayList<Constraint>();

        Set<Integer> visitedNodeIds = new HashSet<Integer>();
        for(NetReservoir rr : simResult.getReservoirs())
                visitedNodeIds.add(rr.getId());

        for(NetLink l : simResult.getLinks()){
                if(!visitedNodeIds.contains(l.getFirstNodeId())                 ||
!visitedNodeIds.contains(l.getSecondNodeId())){

                        NetNode firstNode = null, secondNode = null;
                        for(NetNode n : simResult.getNodes()){
```

```
                        if(n.getId() == l.getFirstNodeId())
                            firstNode = n;
                        else if(n.getId() == l.getSecondNodeId())
                            secondNode = n;
                    }

                    // the link connects to reservoir
                    if(firstNode == null || secondNode == null)
                        continue;

                    visitedNodeIds.add(l.getFirstNodeId());
                    visitedNodeIds.add(l.getSecondNodeId());

                    //System.out.println("Link's ID: " + l.getId() + " flow: "
+ l.getFlow());
                    //System.out.println("Link's first node ID: " +
l.getFirstNodeId());
                    //System.out.println("Link's second node ID: " +
l.getSecondNodeId());

                    Constraint cons = new Constraint();

                    //Constraint string construction
                    String consStr = new String();
                    for(NetLink nl : simResult.getLinks()){
                        for(Double dia :
conf.getAvailablePipeDiameters().keySet()){
                            if(nl.getId() == l.getId()){
                                double headLoss =
calculateHeadLoss(nl.getLength(), nl.getFlow(), dia,
conf.getRoughnessCoefficient());
                                if(nl.getFlow() >= 0)
                                    consStr += (-1) *
headLoss + " ";
                                else
                                    consStr += (+1) *
headLoss + " ";
                            }
                            else
                                consStr += "0 ";
                        }
                    }
                    for(@SuppressWarnings("unused") NetReservoir rr :
simResult.getReservoirs()){
                        for(@SuppressWarnings("unused") Double ele
: conf.getAvailableReservoirElevations().keySet()){
```

```
                                        consStr += "0 ";
                                }
                        }
                        for(NetNode nn : simResult.getNodes()){
                                if(nn.getId() == firstNode.getId())
                                        consStr += "1 ";
                                else if(nn.getId() == secondNode.getId())
                                        consStr += "-1 ";
                                else
                                        consStr += "0 ";
                        }
                        consStr = consStr.substring(0, consStr.length()-1);

                        cons.setConsStr(consStr);
                        cons.setRhs(secondNode.getElevation() -
firstNode.getElevation());
                        cons.setType(LpSolve.EQ);
                        result.add(cons);
                }
        }
        return result;
    }

    private static Constraint generateReservoirConstraints(EpaNetResult
simResult, Configuration conf) {

        //List<Constraint> result = new ArrayList<Constraint>();
        Constraint result = new Constraint();

        int resId = simResult.getReservoirs().get(0).getId();

        NetLink resLink = null;
        int resFirstNodeId = -1;
        NetNode resNode = null;

        for(NetLink l : simResult.getLinks()){
                if(l.getFirstNodeId() == resId){
                        resLink = l;
                        resFirstNodeId = l.getSecondNodeId();
                        break;
                }
                else if(l.getSecondNodeId() == resId){
                        resLink = l;
                        resFirstNodeId = l.getFirstNodeId();
                        break;
                }
```

```
		}

		for(NetNode n : simResult.getNodes()){
			if(n.getId() == resFirstNodeId){
				resNode = n;
				break;
			}
		}

		result.setRhs(resNode.getElevation());
		result.setType(LpSolve.EQ);
		String consStr = new String();
		for(NetLink ll : simResult.getLinks()){
			for(Double dia : conf.getAvailablePipeDiameters().keySet()){
				if(ll.getId() == resLink.getId())
					consStr += (-1) *
calculateHeadLoss(ll.getLength(), ll.getFlow(), dia, conf.getRoughnessCoefficient())
+ " ";
				else
					consStr += "0 ";
				}
		}

		//Reservoir
		for(@SuppressWarnings("unused") NetReservoir rr :
simResult.getReservoirs()){
			for(Double ele :
conf.getAvailableReservoirElevations().keySet()){
				consStr += ele + " ";
			}
		}
		//Nodal Pressure
		for(NetNode n : simResult.getNodes()){
			if(n.getId() == resNode.getId())
				consStr += "-1 ";
			else
				consStr += "0 ";
		}
		consStr = consStr.substring(0, consStr.length()-1);
		result.setConsStr(consStr);


		/*
		for(NetLink l : simResult.getLinks()){
			if(l.getFirstNodeId() == resId || l.getSecondNodeId() ==
resId){
```

```
                        Constraint cons = new Constraint();
                        //cons.setRhs(0);
                        cons.setType(LpSolve.EQ);
                        String consStr = new String();
                        for(NetLink ll : simResult.getLinks()){
                            for(Double dia : conf.getAvailablePipeDiameters().keySet()){
                                if(ll.getId() == l.getId())
                                    consStr += (-1) * (calculateHeadLoss(ll.getLength(), ll.getFlow(), dia, conf.getRoughnessCoefficient())) + " ";
                                else
                                    consStr += "0 ";
                            }
                        }

                        //Reservoir
                        for(Double ele : conf.getAvailableReservoirElevations().keySet()){
                            consStr += ele + " ";
                        }
                        //Nodal Pressure
                        for(NetNode n : simResult.getNodes()){
                            if(n.getId() == l.getFirstNodeId() || n.getId() == l.getSecondNodeId())
                                consStr += "-1 ";
                            else
                                consStr += "0 ";
                        }
                        consStr = consStr.substring(0, consStr.length()-1);
                        cons.setConsStr(consStr);
                        result.add(cons);
                    }
                }
                 */

                return result;
        }
        //Calculates Head Loss using Hazen-Williams Method
        private static double calculateHeadLoss(double length, double flow, double d, double c){
                double result;
                //result = (length * 10.67 * Math.pow(Math.abs(flow*0.001), 1.85)) / (Math.pow(c, 1.85) * Math.pow(d*0.001, 4.87));
                //if(flow < 0)
                //result = result * -1;
```

```
		result = Math.pow((Math.abs(flow)*0.001*Math.pow(length, 0.54)*Math.pow(4, 1.63))/(Math.PI * Math.pow(d*0.001, 2.63) * 0.85 * c), (1.0/0.54));
		return result;
	}

	//Calculates Velocity
	private static double calculateVelocity(double flow, double d){
		return (flow*0.001) / ((Math.PI / 4) * Math.pow(d*0.001, 2));
	}

	private static String constructObjFunct(List<NetLink> links, Map<Double, Double> availablePipeDiameters, List<NetReservoir> reservoirs, Map<Double, Double> availableReservoirElevations, List<NetNode> nodes) {

		String result = new String();

		//Links
		for(NetLink l : links){
			for(Double dia : availablePipeDiameters.keySet()){
				result += l.getLength() * availablePipeDiameters.get(dia) + " ";
			}
		}

		//Reservoirs
		for(@SuppressWarnings("unused") NetReservoir r : reservoirs){
			for(Double ele : availableReservoirElevations.keySet()){
				result += availableReservoirElevations.get(ele) + " ";
			}
		}
		//Nodes
		for(@SuppressWarnings("unused") NetNode n : nodes){
			result += 0 + " ";
		}

		result = result.substring(0, result.length()-1);

		return result;
	}
	private static Configuration parseConfigurationFile() {
		Configuration conf = new Configuration();
		try {
			DocumentBuilder parser = DocumentBuilderFactory.newInstance().newDocumentBuilder();
```

```
                Document doc = parser.parse("resources/Configuration.xml");

                //Input File Location

        conf.setInputFileLocation(doc.getElementsByTagName("InputFileLocation"
).item(0).getTextContent());

                //Output File Location

        conf.setOutputFileLocation(doc.getElementsByTagName("OutputFileLocati
on").item(0).getTextContent());

                // Stop If No Improvement Steps

        conf.setStopSteps(Integer.parseInt(doc.getElementsByTagName("StopIfNoI
mprovementSteps").item(0).getTextContent()));

                //Network Constraints
                org.w3c.dom.Node nc =
doc.getElementsByTagName("Network").item(0);
                for(int i=0; i<nc.getAttributes().getLength(); i++)

        conf.addNetworkConstraint(nc.getAttributes().item(i).getNodeName(),
Double.parseDouble(nc.getAttributes().item(i).getTextContent()));

                //Link Velocity Constraints
                NodeList velConstraints =
doc.getElementsByTagName("LinkVelocity");
                for(int i=0; i<velConstraints.getLength(); i++){
                        org.w3c.dom.Node nvc = velConstraints.item(i);
                        int linkId =
Integer.parseInt(nvc.getAttributes().getNamedItem("LinkId").getTextContent());
                        double linkMinVelocity =
Double.parseDouble(nvc.getAttributes().getNamedItem("MinVelocity").getTextCon
tent());
                        double linkMaxVelocity =
Double.parseDouble(nvc.getAttributes().getNamedItem("MaxVelocity").getTextCo
ntent());
                        conf.addLinkVelocityConstraint(linkId,
linkMinVelocity, linkMaxVelocity);
                }

                //Reservoirs
                NodeList rl = doc.getElementsByTagName("Reservoir");
                for(int i=0; i<rl.getLength(); i++){
```

```
                    int id =
Integer.parseInt(rl.item(i).getAttributes().getNamedItem("id").getTextContent());
                    conf.addReservoirId(id);
                }
                //Available Pipe Diameters
                NodeList dl = doc.getElementsByTagName("Diameter");
                for(int i=0; i<dl.getLength(); i++){
                    double length =
Double.parseDouble(dl.item(i).getAttributes().getNamedItem("length").getTextCont
ent());
                    double cost =
Double.parseDouble(dl.item(i).getAttributes().getNamedItem("cost").getTextConten
t());
                    conf.addAvailablePipeDiameter(length, cost);
                }
                //Available Reservoir Elevations
                NodeList el = doc.getElementsByTagName("Elevation");
                for(int i=0; i<el.getLength(); i++){
                    double height =
Integer.parseInt(el.item(i).getAttributes().getNamedItem("height").getTextContent()
);
                    double cost =
Double.parseDouble(el.item(i).getAttributes().getNamedItem("cost").getTextConten
t());
                    conf.addAvailableReservoirElevation(height, cost);
                }
                //Roughness Coefficient

        conf.setRoughnessCoefficient(Double.parseDouble((doc.getElementsByTag
Name("RoughnessCoefficient").item(0).getTextContent())));

            } catch (Exception e) {
                System.out.println("Parse configuration file failed..");
                e.printStackTrace();
            }
            return conf;
        }

        private static EpaNetResult simulateEpa(Network net, List<Integer>
reservoirIds){

            EpaNetResult simResult = new EpaNetResult();


            try {
                Logger log = Logger.getLogger("log");
```

```
                    HydraulicSim hydSim = new HydraulicSim(net, log);
                    File hydFile = File.createTempFile("hydSim", "bin");

                    hydSim.simulate(hydFile);
                    //hydSim.stopRunning();
                    HydraulicReader hydReader = new HydraulicReader(new
RandomAccessFile(hydFile, "r"));
                    AwareStep step = hydReader.getStep(0);
                    hydFile.delete();

                    //Links
                    int i = 0;
                    for(Link l : net.getLinks()){
                            simResult.addLink(new
NetLink(l.getNULength(PropertiesMap.UnitsType.SI),      step.getLinkFlow(i,      l,
net.getFieldsMap()),
                                            Integer.parseInt(l.getId()),
Integer.parseInt(l.getFirst().getId()), Integer.parseInt(l.getSecond().getId())));

                            i++;
                    }
                    //Nodes & Reservoirs
                    for(Node n : net.getNodes()){
                            int nodeId =  Integer.parseInt(n.getId());
                            //Reservoir
                            if(reservoirIds.contains(new Integer(nodeId))){
                                    simResult.addReservoir(new
NetReservoir(n.getNUElevation(PropertiesMap.UnitsType.SI), nodeId));
                            }
                            //Node
                            else{
                                    simResult.addNode(new
NetNode(n.getElevation(), nodeId));
                            }
                    }

            }catch(Exception e){
                    System.out.println("Simulate Epa failed..");
                    e.printStackTrace();
            }

            return simResult;
    }

    private static Network parseNetworkFile(String networkFileLocation){
            //Read input file and construct network
```

```
		Logger log = Logger.getLogger("log");
		InpParser parser = new InpParser(log);
		Network net = new Network();
		try {
			parser.parse(net, new File(networkFileLocation));

	//net.getPropertiesMap().setUnitsflag(PropertiesMap.UnitsType.SI);
			//net.updatedUnitsProperty();
		} catch (ENException e) {
			System.out.println("Network parse failed..");
			e.printStackTrace();
		}
		return net;
	}
}
package optimization;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Configuration {

	private String inputFileLocation;
	private String outputFileLocation;
	private Map<String, Double> networkConstraints;
	private List<Integer> reservoirIds;
	private Map<Double, Double> availablePipeDiameters;
	private Map<Double, Double> availableReservoirElevations;
	private Double roughnessCoefficient;
	private Map<Integer, LinkVelocityConstraint> linkVelocityConstraints;
	private int stopSteps;

	public Configuration(){
		inputFileLocation = null;
		networkConstraints = new HashMap<String, Double>(4);
		reservoirIds = new ArrayList<>();
		availablePipeDiameters = new HashMap<Double, Double>();
		availableReservoirElevations = new HashMap<Double, Double>();
		roughnessCoefficient = null;
		linkVelocityConstraints = new HashMap<Integer,
LinkVelocityConstraint>();
	}

	public void addNetworkConstraint(String name, double value){
```

```
            networkConstraints.put(name, value);
        }
        public void addReservoirId(int id){
            reservoirIds.add(id);
        }
        public void addAvailablePipeDiameter(double length, double cost){
            availablePipeDiameters.put(length, cost);
        }
        public void addAvailableReservoirElevation(double height, double cost){
            availableReservoirElevations.put(height, cost);
        }
        public void addLinkVelocityConstraint(int linkId, double minVelocity,
double maxVelocity){
            LinkVelocityConstraint velCon = new
LinkVelocityConstraint(minVelocity, maxVelocity);
            linkVelocityConstraints.put(linkId, velCon);
        }
        public LinkVelocityConstraint getLinkVelocityConstraint(int linkId){
            return linkVelocityConstraints.get(linkId);
        }
        public String getInputFileLocation() {
            return inputFileLocation;

        public void setInputFileLocation(String inputFileLocation) {
            this.inputFileLocation = inputFileLocation;
        }
        public String getOutputFileLocation() {
            return outputFileLocation;
        }
        public void setOutputFileLocation(String outputFileLocation) {
            this.outputFileLocation = outputFileLocation;
        }
        public double getRoughnessCoefficient() {
            return roughnessCoefficient;
        }
        public void setRoughnessCoefficient(double roughnessCoefficient) {
            this.roughnessCoefficient = roughnessCoefficient;
        }
        public Map<String, Double> getNetworkConstraints() {
            return networkConstraints;
        }
        public List<Integer> getReservoirIds() {
            return reservoirIds;
        }
        public Map<Double, Double> getAvailablePipeDiameters() {
            return availablePipeDiameters;
```

```
	}

	public Map<Double, Double> getAvailableReservoirElevations() {
		return availableReservoirElevations;
	}

	public void setStopSteps(int stopSteps) {
		this.stopSteps = stopSteps;
	}
	public int getStopSteps() {
		return stopSteps;
	}
}
package optimization;
public class Constraint {

	private String consStr;
	private int type;
	private double rhs;

	public Constraint(){
		super();
		consStr = new String();
	}
	public String getConsStr() {
		return consStr;
	}
	public void setConsStr(String consStr) {
		this.consStr = consStr;
	}
	public int getType() {
		return type;
	}
	public void setType(int type) {
		this.type = type;
	}
	public double getRhs() {
		return rhs;
	}
	public void setRhs(double rhs) {
		this.rhs = rhs;
	}
}

package optimization;
```

```
import java.util.ArrayList;
import java.util.List;

public class EpaNetResult {

	private List<NetNode> nodes;
	private List<NetLink> links;
	private List<NetReservoir> reservoirs;

	public EpaNetResult(){
		nodes = new ArrayList<NetNode>();
		links = new ArrayList<NetLink>();
		reservoirs = new ArrayList<NetReservoir>();
	}
	public List<NetNode> getNodes() {
		return nodes;
	}
	public List<NetLink> getLinks() {
		return links;
	}
	public List<NetReservoir> getReservoirs() {
		return reservoirs;
	}
	public void addNode(NetNode node){
		nodes.add(node);
	}
	public void addLink(NetLink link){
		links.add(link);
	}
	public void addReservoir(NetReservoir reservoir){
		reservoirs.add(reservoir);
	}
	public void clearAllData(){
		nodes.clear();
		links.clear();
		reservoirs.clear();
	}
	public void prettyPrint() {
		System.out.println("---------------------------------------");
		System.out.println("Nodes:");
		for(NetNode n : getNodes())
			System.out.println(String.format("\tId:    %d\tElevation:    %f",
n.getId(), n.getElevation()));

		System.out.println("Links:");
		for(NetLink l : getLinks())
```

```
                        System.out.println(String.format("\tId: %d\tLength: %f\tFlow:
%f\tFirstNodeId: %d\tSecondNode: %d", l.getId(), l.getLength(), l.getFlow(),
l.getFirstNodeId(), l.getSecondNodeId()));


                System.out.println("Reservoirs:");
                for(NetReservoir r : getReservoirs())
                        System.out.println(String.format("\tId: %d\tElevation: %f",
r.getId(), r.getElevation()));
                System.out.println("---------------------------------------");

        }
}

package optimization;

public class NetLink {

        private double length;
        private double flow;
        private int id;
        private int firstNodeId;
        private int secondNodeId;

        public NetLink(double length, double flow, int id, int first, int second){
                super();
                this.length = length;
                this.flow = flow;
                this.id = id;
                this.firstNodeId = first;
                this.secondNodeId = second;
        }
        public double getLength() {
                return length;
        }
        public void setLength(double length) {
                this.length = length;
        }
        public double getFlow() {
                return flow;
        }
        public void setFlow(double flow){
                this.flow = flow;
        }
        public int getId() {
                return id;
```

```
	}
	public void setId(int id) {
		this.id = id;
	}
	public int getFirstNodeId() {
		return firstNodeId;
	}
	public void setFirstNodeId(int firstNodeId) {
		this.firstNodeId = firstNodeId;
	}
	public int getSecondNodeId() {
		return secondNodeId;
	}
	public void setSecondNodeId(int secondNodeId) {
		this.secondNodeId = secondNodeId;
	}

}

package optimization;

public class NetNode {
	private double elevation;
	private int id;

	public NetNode(double elevation, int id){
		super();
		this.elevation = elevation;
		this.id = id;
	}
	public double getElevation() {
		return elevation / 3.280839895013123;
	}
	public void setElevation(double elevation) {
		this.elevation = elevation;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
}
package optimization;

public class NetReservoir {
```

```
	private double elevation;
	private int id;
	public NetReservoir(double elevation, int id){
		super();
		this.elevation = elevation;
		this.setId(id);
	}
	public double getElevation() {
		return elevation;
	}
	public void setElevation(double elevation) {
		this.elevation = elevation;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}

}
package optimization;

public class LinkVelocityConstraint {
	private double minVelocity;
	private double maxVelocity;

	public LinkVelocityConstraint(double minVelocity, double maxVelocity){
		super();
		this.minVelocity = minVelocity;
		this.maxVelocity = maxVelocity;
	}
	public double getMinVelocity() {
		return minVelocity;
	}
	public void setMinVelocity(double minVelocity) {
		this.minVelocity = minVelocity;
	}
	public double getMaxVelocity() {
		return maxVelocity;
	}
	public void setMaxVelocity(double maxVelocity) {
		this.maxVelocity = maxVelocity;
	}
}
```