TRAFFIC AND TOPOLOGY ENGINEERING IN NETWORKS: ALGORITHMS, MODELS AND OPTIMIZATION

by

Ahmad Askarian

APPROVED BY SUPERVISORY COMMITTEE:

András Faragó, Chair

Mehrdad Nourani

Neeraj Mittal

Farokh B. Bastani

To my beloved Niloofar.

TRAFFIC AND TOPOLOGY ENGINEERING IN NETWORKS: ALGORITHMS, MODELS AND OPTIMIZATION

by

AHMAD ASKARIAN, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

TELECOMMUNICATIONS ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

May 2017

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Professor Faragó for the continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my PhD study.

March 2017

TRAFFIC AND TOPOLOGY ENGINEERING IN NETWORKS:

ALGORITHMS, MODELS AND OPTIMIZATION

Ahmad Askarian, PhD
The University of Texas at Dallas, 2017

Supervising Professor: András Faragó

Traffic engineering (TE) helps to use network resources more efficiently. Network operators use TE to obtain different objectives such as load balancing, congestion avoidance and average delay reduction. We approach the issue using optimization theory and a network design game model. Also routing methods for optimal distribution of traffic in data networks that can also provide quality of service (QoS) for users is one of the challenges in recent years' research on next generation networks. The major QoS requirement in most cases is an upper bound on end-to-end path delay. In this dissertation, we address the problem of parallelizing some network algorithms on a practical distributed memory system. Also we use approximation approach for analyzing NP hard problems in networks.

TABLE OF CONTENTS

Acknowledgments v
List of Tables ix
List of Figures x
CHAPTER 1 INTRODUCTION 1
CHAPTER 2 OPTIMIZATION METHODS FOR NETWORK TRAFFIC ENGINEERING 4
2.1 Problem Statement 6
2.2 The Dual Problem 8
2.3 Practical Requirements 10
2.4 Simulation Results 14
2.5 Low Structural Congestion via Game Theory and Linear Programming 23
2.6 Related Work 25
2.7 Structural Congestion 26
2.8 Network Design Model 29
2.9 Simulation 35
CHAPTER 3 AN OPTIMAL TRAFFIC DISTRIBUTION METHOD SUPPORTING END-TO-END DELAY BOUND 37
3.1 Traffic Distribution Model 40
3.2 Solving the Optimization Problem 45
3.3 Simulation 50
3.4 Mathematical Analysis 53
CHAPTER 4 PARALLELIZING LARGE SCALE GRAPH ALGORITHMS USING THE APACHE SPARK DISTRIBUTED MEMORY SYSTEM 63
4.1 Overview of Big Data Processing 64
4.2 Apache Spark Vs. Hadoop MapReduce 66
4.3 Resilient Distributed Dataset (RDD) 70
4.4 GraphX API for Spark 72
4.5 Graph Partitioning in Spark 73
4.6 Paper citation Network Example 76
4.7 Finding PageRank in Social Network using Apache Spark 79

4.8 Finding connected components using Apache Spark 81
4.9 Triangle counting using Apache Spark 83
4.10 Summery of distributed graph processing 86
CHAPTER 5 APPROXIMATION APPROACH FOR ANALYSING NETWORK STRUCTURE 88
5.1 How Complex Networks Inspire New Avenues to Approximation – The Case of Unsplittable Flows 88
5.2 Problem statement. 88
5.3 Related work 90
5.4 A Simple Practical Approximation 92
5.5 Utilizing Network Structure to Accelerate Markov Chain Monte Carlo Algorithms 99
5.6 Aggregation in Markov Chains 103
5.7 Convergence Analysis 107
5.8 Numerical Demonstration 115
CHAPTER 6 CONCLUSION 121
References 124
Biographical Sketch 128
Curriculum Vitae 129

LIST OF TABLES

Table 2-1 Optimum flows 15

Table 2-2 The Solution of DUAL_I 16

Table 2-3 The Solution of DUAL_II 16

Table 2-4 Solution of DUAL_II 18

Table 2-5 MLU and BWE values 20

Table 2-6 MLU and BWE values 22

Table 2-7 Optimum Weights 33

Table 3-1 Parameters of the Network links 51

Table 3-2 Simulation results of step 1 54

Table 3-3 Simulation results of step 2 54

Table 3-4 Final results of Step3 for 1000 iterations and step size 0.008 55

Table 4-1 Slashdot social network statistics 85

LIST OF FIGURES

Figure 2-1 Maximum Load Balancing Flow Chart........ 13

Figure 2-2 Simulation Network Topology........ 15

Figure 2-3 IP Traffic drop in default protocol and new method........ 17

Figure 2-4 Simulation Network Topology........ 18

Figure 2-5 Scenario2 IP Traffic Dropped........ 21

Figure 2-6 Scenario2 IP Traffic Received........ 21

Figure 2-7 Node Vi decision strategies........ 30

Figure 2-8 Topological Sorting........ 32

Figure 2-9 Algorithm Flowchart........ 34

Figure 2-10 Network Topology........ 35

Figure 2-11 Degree ratio for each node using different alpha........ 36

Figure 3-1 A network graph with three sessions........ 42

Figure 3-2 The flowchart of flow distribution algorithm........ 48

Figure 3-3 The flow of links per 200 iterations........ 55

Figure 3-4 The flow of links per 200 iterations........ 56

Figure 3-5 The Lagrangian multipliers corresponding to paths per 200 iterations........ 56

Figure 3-6 q (Λ) for one dimensional........ 60

Figure 4-1 Spark full stack........ 65

Figure 4-2 Yet another resource manager........ 66

Figure 4-3 Apache Spark efficiency and usability........ 67

Figure 4-4 Cities information........ 68

Figure 4-5 Maximum temperature for each city 68

Figure 4-6 Maximum temperature for each city 68

Figure 4-7 Output of the reduce task 69

Figure 4-8 A Multi-Iteration algorithm in MapReduce 69

Figure 4-9 Multi-Iteration Algorithm in Spark 70

Figure 4-10 RDD transformation and action 71

Figure 4-11 Filter by age in an RDD 71

Figure 4-12 Word Count example, using Spark 72

Figure 4-13 Graph partitioning strategy 73

Figure 4-14 EdgePartition1D implementation 74

Figure 4-15 EdgePartition2D implementation 75

Figure 4-16 Random Vertex Cut implementation 75

Figure 4-17 Canonical Random Vertex Cut implementation 76

Figure 4-18 Text file containing the citation information 76

Figure 4-19 Loading a graph in Spark 77

Figure 4-20 inDegree and reduce in Graphx API 78

Figure 4-21 inDegrees method in GraphX producing key-value pairs 78

Figure 4-22 reduce (Function) method in GraphX producing a single key value pair 79

Figure 4-23 vertices in the paperCitationGraph 79

Figure 4-24 PageRank transformation that creates a new property graph 80

Figure 4-25 PageRank method in Spark using GraphX library 81

Figure 4-26 The most influential paper 81

Figure 4-27 Finding connected components 82

Figure 4-28 Network construction based on RDD collections 82

Figure 4-29 Finding connected components 83

Figure 4-30 Triangle counting algorithm 84

Figure 4-31 Triangle Counting in Slashdot social network 86

Figure 5-1 Deviation from the stationary measure for many iterations 119

Figure 5-2 Very fast convergence to satisfy the error bound 120

CHAPTER 1

INTRODUCTION

In recent years, the use of the Internet as communication infrastructure for different telecommunication applications has been growing significantly. Because bandwidth is one of the most important requirements of these applications, network hardware should support bandwidth management techniques. In chapter 2 we will discuss linear optimization methods which can help Internet protocols work better. We use a duality theory to find a weight set that improve the routing protocols efficiencies. As a matter of fact, routing is the most important aspect of Internet Traffic Engineering. So, we focus on routing protocols and introduce a practical method that optimizes Link Metrics. Previous optimization methods suffer from practical issues but our method could be implemented with Routing Protocols that are based on shortest paths. Our simulation results show significant improvement on network efficiency. Also, we propose a network topology design approach that targets the reduction of structural congestion in a directed acyclic network. What we mean by structural congestion is that a node has much higher in-degree than out-degree in a directed network. We approach the issue using a network design game model. In this model, we consider multiple sources and one destination. Each node is willing to connect to other nodes but it should pay the price of whole paths it uses to send traffic to the destination. The model yields a weight for each link. We show that if these weights are used to compute shortest paths, then a network topology is obtained with a low level of structural congestion. Routing methods for optimal distribution of traffic in data networks will be addressed in chapter 3. Recently several new services have become popular in the internet qualities of which depend on the end-to-end delay experienced by the packets in the network [1]. For an acceptable QoS it is required that the end-to-end delay is kept under a threshold level. Providing QoS is not an easy task in datagram networks. In new generation networks, virtual circuit switched networks such as MPLS is used to provide a better framework to implement QoS. A new method will be introduced for traffic distribution in virtual circuit switched networks which can be implemented in real networks. In this method, the input traffic of each session is distributed among the possible paths, in a manner that the total system cost is minimized at the same time as the average cost for each path is kept bounded below a required threshold level. This method is scalable as its operation is per session. It is analytically proven in this article that this algorithm converges under the assumptions that are feasible in real networks. The simulation results approve the effectiveness of the algorithm. The results obtained from the simulation are in line with the results obtained from analytical resolution of the convex optimization problem. In chapter 4, in order to analyzing large scale networks, we will discuss distributed data storage and processing system called Spark. We will address the problem of parallelizing three famous network algorithms on a practical distributed memory system. The approach is based on the Spark framework and the GraphX API which is run on top of the Hadoop distributed file system. We develop three case studies in this framework: (1) computing the PageRank in a social network, (2) finding connected components in the graph representing the network, and (3) triangle counting. A key issue for the large-scale implementation is how to partition the whole task into parallel and independent tasks that run on different machines, such that we can reduce the communication and storage overhead in the distributed cluster. In the last chapter we mathematically analyze estimation methods for NP hard network problems. The Disjoint Connecting Paths problem, and its capacitated generalization, called Unsplittable Flow problem, play an important role in practical applications, such as communication network design and routing. These problems are hard in general, but various polynomial-time approximations are known. Also, we consider the problem of estimating the measure of subsets in very large networks.

The dissertation is organized in four main chapters. In the second chapter we study linear network optimization and proposed a practical method to leverage strong duality property of a linear programming. In the third chapter we study convex optimization problems and their application in network traffic engineering. We designed an algorithm which is guaranteed end to end delay for each session as well as minimizing average delay in the whole network. As networks topology become large, traditional methods for analyzing network structure become less efficient. To avoid that lack of efficiency we study a big data processing system using their graph API to processes large scale networks. We study and implement such a method in chapter four. In the last chapter we used approximation methods to solve two important network NP-hard problems. The first one is unsplittable flow problem and the second one is estimating the measure of subsets in very large networks.

CHAPTER 2

OPTIMIZATION METHODS FOR NETWORK TRAFFIC ENGINEERING

Traffic engineering helps to use network resources more efficiently. Network operators use TE to obtain different objectives such as load balancing, congestion avoidance and average delay reduction. Plane IP routing protocols such as OSPF, a popular intradomain routing protocol, are believed to be insufficient for TE. OSPF is based on the shortest path algorithm in which link weights are usually static value without considering network load. They can be set using the inverse proportional bandwidth capacity or certain value. However, Optimization theory helps network researchers and operators to analyze the network behavior more precisely. It is not a practical approach can be implemented in traditional protocol. This chapter address the feasibility requirements, a weight set can be extracted from optimization problem use as a link metric in OSPF. We show the routes that selected in OSPF with these metrics distribute the traffic closer to optimal situation than routes from OSPF with default metric.

In recent years, the use of the Internet as communication infrastructure for different telecommunication applications has been growing significantly. Because bandwidth is one of the most important requirements of these applications, network hardware should support bandwidth management techniques. Traffic engineering (TE) is a bandwidth management technique that considers different objectives such as maximum throughput, minimum congestion and load balancing in the network. TE puts the traffic where network bandwidth is available. TE with the objective of load balancing can reduce maximum link utilization (MLU) and increase bandwidth efficiency (BWE). Because considerable delay may occur at congested links, reduction of end to end delay can be achieved as a side result of load balancing.

Destination-based routing is not flexible for TE, and so it is highly susceptible to congestion. Because of this reason the concept of TE was developed mostly in MPLS-based networks [2] [3]. MPLS-based TE can optimize traffic distribution using dedicated label switch paths (LSP). The capability of explicit routing and arbitrary traffic splitting are the most important features of MPLS TE. But the MPLS has not been widely deploy Rapid increase in network traffic especially that of new applications which require QoS guarantees, has encouraged the network providers to apply IP-based TE with different objectives. The main idea of IP-based TE is to find a set of weights that optimizes a specific objective function. If the objective function is the total link cost, the constraint of equal cost multipath (ECMP) causes the problem to be NP hard [1]. Different near-optimal heuristic algorithms based on local search were proposed to solve this problem [1].

One approach for analyzing the TE problem is formulating it with optimization theory problems. If we consider load-balancing as an objective of the optimization problem and consider the amount of traffic load on all links that belong to a specific session as the problem outcome, the solution of such problem is the path of each session that results in minimum congestion.

Measurements in [4] indicate that bottlenecks of the Internet backbone are not only located between ASs but also, they exist in intradomain links. The popular intradomain routing protocol is OSPF. In this chapter, we present a formulation of the optimization problem that object to provide maximum load balancing. This objective function is useful in a situation that network entrance is random since increase the probability of new traffic admission. In addition, we try to extract the OSPF metric from this problem and therefore reach the load balancing with OSPF routing. These attempts result in a new definition such as equivalent weight set and equivalent constraints. In this chapter, we analyze the optimization problem from feasibility perspective and show that a set of link weights that can be embedded as a link metric in OSPF protocol results in optimal or near optimal load balancing. Our simulations show that this method improves bandwidth efficiency and reduces network congestion and also leads to a substantial reduction in the end to end delay.

2.1 Problem Statement

Different TE objectives lead to different objective functions of optimization problem. We consider load balancing as an objective of traffic engineering so the objective function of the optimization problem is to minimize MLU (maximum link utilization). Consider the linear optimization problem that is called first primal problem (PRIMAL_I) with the following notation. A connected graph $G\ (N, A)$ is given. $c_{ij} : (i\ , j) \in A$ is a set of edge capacities and (s_k, d_k) is a set of source-destination pairs for each session $k \in K$. D_k is the total amount of session k traffic. The amount of traffic in link $(i, j) \in A$ that belongs to session k is X_{ij}^{k}. So, the problem is:

$$\min \mathrm{MLU} \qquad (1)$$

$$\sum_{j:(i,j)\in A} X_{i,j}^{k} - \sum_{j:(j,i)\in A} X_{j,i}^{k} = \begin{cases} D_i & i = \text{source} \\ -D_i & i = \text{destination} \\ & \text{o.w} \end{cases} \qquad (2)$$

$$\sum_{k\in K} X_{i,j}^{k} \leq C_{i,j}\mathrm{MLU} \qquad (3) \qquad\qquad (2\text{-}1)$$

$$X_{i,j\geq 0} \qquad (4)$$

Constraints (2-1-2) are flow conservation constraints that are derived from network topology. Constraint (2-1-3) ensure that link flows do not violate link capacity and (2-1-4) says that link flows are nonnegative. The PRIMAL_I solution specifies theX_{ij}^{k}, and so we have the optimal path with arbitrary splitting for all sessions that minimize MLU. Our objective is to find a practical method suited for IP networks that forces the traffic to go through a set of optimal paths. To achieve this goal, we should find a set of new link metrics such that all paths which are specified by PRIMAL_I problem can also be obtained by the shortest path algorithm in regard to the new metrics. It means that if $X_{ij}^{k} > 0$, then link (i, j) should be selected by session k according to the shortest path algorithm. Here we assume that the shortest path algorithm is OSPF that supports Equal-Cost Multi-Path (ECMP).

The load balancing methods introduced in [3] is based on primal optimization problem. In this chapter, we consider the dual optimization problem (DUAL_II) which is obtained with respect to Lagrange Multipliers. In other word, we aim to distribute traffic by determining the links weight. And It will be shown that the Lagrange Multipliers comparable with constraint (2-1-3) can be interpreted as OSPF link metrics that satisfy the load balancing objective.

Link metric in OSPF protocol must be an integer between 1 and 65535 but we will show in section IV that the Lagrange Multipliers that are obtains from the solution of DUAL_II problem and comparable with constraint (2-1-3), do not satisfy this range in general. So, the following definition gives us the choice of an alternative weight set.

Definition 1: Two weights set $\vec{W} = \{w_i\}_{i=1}^{L}$ and $\vec{W}' = \{w_i'\}_{i=1}^{L}$ are equivalent with respect to a given graph $G(N, A)$ with L links, if and only if the shortest paths between any arbitrary nodes in G are the same considering any one of these two weight sets.

2.2 The Dual Problem

Defining the Lagrange multiplayer $\{p_i\}_{i=1}^{N}$ comparable with constraint (2-1-2) and $\{w_{ij}\}_{(i,j)\in A}$ comparable with constraint (3), the Lagrange polynomial is:

$$\underset{X_{ij}^{k}\geq 0}{L}(X, MLU, P, W) = MLU + \sum_{k\in K}\sum_{i\in N} p_i (D_k - \sum_{j:(i,j)\in A} X_{ij}^{k} + \sum_{j:(j.i)\in A} X_{ji}^{k}) + \sum_{(i,j)\in A} w_{ij} (\sum_{k\in K} X_{ij}^{k} - C_{ij})$$

For more details can be referred to chapter 5 of [5]. To achieve the dual problem the following equation should be satisfied for each feasible X and MLU.

$$\mathrm{MLU} \geq \mathrm{L(X, MLU, P, W)}$$

To satisfy (6) we must have: $w_{ij} \geq 0$. Now the function $g(W,P)$ is defined as bellow:

$$g(W,P) = \min_{X,MLU} L(X, MLU, P, W)$$

So, we have:

$$g(W,P) = \min_{X,MLU} \sum\sum p_i^k D_i^k + \sum_{k\in K}\sum_{(i,j)\in A} X_{ij}^{k} (p_j - p_i + w_{ij}) + MLU(1 - \sum_{(i,j)\in A} w_{ij} C_{ij})$$

Because w_{ij} and X_{ij}^{k} are positive values, we have:

$$g(W,P)=\begin{cases}\sum_{k\in K}\sum_{i\in N} p_i^k D_i^k & p_i - p_j \le w_{ij} \ and \sum C_{ij} w_{ij} = 1 \\ -\infty & p_i - p_j \ge w_{ij} \ or \sum_{(i,j)\in A} C_{ij} w_{ij} \ne 1\end{cases}$$

The dual function is defined to maximize $g(W,P)$ when all w_{ij} are positive values. Equation (2-2) shows the DUAL_I problem that is the dual function of PRIMAL_I.

$$\max \sum_{k\in K} p_{tk}^k D^k \qquad (10)$$

$$p_i^k - p_j^k \le w_{ij} \qquad (11)$$

$$\sum_{(i,j)\in A} C_{ij} w_{ij} = 1 \qquad (12) \qquad (2\text{-}2)$$

$$p_{sk}^k = 0 \qquad (13)$$

$$w_{ij} \ge 0 \qquad (14)$$

As the primal and dual problems are linear, strong duality holds and according to complementary slackness in KKT theorem if $\hat{X}_{ij}^k$ is optimal solution of PRIMAL_I and $\{\hat{w}_{ij}, \hat{p}_{ij}^k\}$ is the optimal solution of DUAL_I we have:

$$\hat{X}_{ij}^k.(\hat{p}_i^k - \hat{p}_j^k + \hat{w}_{ij}) = 0 \qquad (2\text{-}3)$$

Equation (2-3) indicates that if session k passes link (i, j) then $p_j^k - p_i^k = w_{ij}$. According to theorem 1 in [1] if $\{\hat{w}_{ij}\}_{(i,j)\in A}$ is used as a link metric in a shortest path algorithm, all non-empty links ($X_{ij}^k > 0$) will be included among the selected paths by the shortest path algorithm procedure.

2.3 Practical Requirements

$\{w_{ij}\}_{(i,j)\in A}$ (weighs calculated from the DUAL_I) have to be equal to or greater than zero. But as we mentioned before OSPF link metrics cannot be zero. We show that there exists a weight set equivalent to $\{w_{ij}\}_{(i,j)\in A}$ that can be obtained using the new optimization problem.

Lemma 1: consider $G(N,A)$ with weight set $\{w_{ij}\}_{(i,j)\in A}$ and some scalars $\{\delta_i\}_{i=1}^{N}$ corresponding to each link and node respectively. If we change the link weight to $\overline{w}_{ij} = w_{ij} + \delta_j - \delta_i$, then the weight set $\{w_{ij}\}_{(i,j)\in A}$ and $\{\overline{w}_{ij}\}_{(i,j)\in A}$ are equivalent weight sets with respect to $G(N,A)$.

To achieve non-zero weight set we changed the weights of the links according to algorithm 1.

Algorithm 1:

Step 1: For each session $k \in K$ assign the scalar set $\{\delta_i^k\}_{i=1}^{N}$ as follows:

- If there exists at least one directed path to node i from source node of session k (s^k), then δ_i^k is equal to the length of the longest hop-count non-loopy path from s^k to i.
- Else δ_i^k is equal to zero.

Step 2: Assign the $\max_k \delta_i^k$ as the final scalar δ_i

Step 3: If the $\delta_j - \delta_i \geq 0$ then $\overline{w}_{ij} = w_{ij} + \delta_j - \delta_i$ else $\overline{w}_{ij} = w_{ij} + 1$.

So according to lemma1 and algorithm 1, there exists an equivalent weight set to $\{w_{ij}\}_{(i,j)\in A}$ that all of them are greater or equal to one. Thus, we can assume $\sum_{ij} C_{ij}.\overline{w}_{ij} = \overline{H}$.

Theorem 1: consider two optimization problems called DUAL_I and DUAL_II.

DUAL_I:

$$\max \sum_{k \in K} p_{tk}^{k} D^{k}$$

$$p_i^k - p_j^k \leq w_{ij}$$

$$\sum_{(i,j) \in A} C_{ij} w_{ij} = 1$$

$$p_{sk}^{k} = 0$$

$$w_{ij} \geq 0$$

DUAL_II:

$$\max \sum_{k \in K} p'^{k}_{tk} D^{k}$$

$$p'^{k}_{i} - p'^{k}_{j} \leq w'_{ij}$$

$$\sum_{(i,j) \in A} C_{ij} w'_{ij} = H$$

$$p'^{k}_{sk} = 0$$

$$w'_{ij} \geq 1$$

If $\{\widehat{w}_{ij}\}_{(i,j)\in A}$is an optimal solution of DUAL_I and $\{\widehat{w'}_{ij}\}_{(i,j)\in A}$ is an optimal solution of DUAL_II, the sets $\{\widehat{w}_{ij}\}_{(i,j)\in A}$ and $\{\widehat{w'}_{ij}\}_{(i,j)\in A}$ are equivalent weight sets with respect to $G(N, A)$.

Proof: Consider the optimization problem PRIMAL_II.

PRIMAL_II:

$$\max \sum_{k \in K} H.\, p_{tk}^{k} D^{k}$$

$$p_i^k - p_j^k \leq w_{ij}$$

$$\sum_{(i,j) \in A} C_{ij} w_{ij} = 1$$

$$p_{sk}^{k} = 0$$

$$w_{ij} \geq 0$$

It is clear that the X_{ij} s which minimizes the objective function of the problem PRIMALL_II are the same as the ones which cause the problem PRIMALL_I to be optimized.

The dual of PRIMAL_II is the DUAL_TEMP.

DUAL_TEMP:

$$\max \sum_{k \in K} p'^{k}_{tk} D^{k}$$

$$p'^{k}_{i} - p'^{k}_{j} \leq w'_{ij}$$

$$\sum_{(i,j) \in A} C_{ij} w'_{ij} = H$$

$$p'^{k}_{sk} = 0$$

$$w'_{ij} \geq 0$$

From complementary slackness theorem

$$\hat{X}_{ij}^{k}.(\hat{p}'^{k}_{i} - \hat{p}'^{k}_{j} + \hat{w}'_{ij}) = 0$$

Since the optimal solutions of the PRIMAL_I and PRIMAL_II are the same, thus the weight sets $\{\widehat{w}_{ij}\}_{(i,j)\in A}$ and $\{\widehat{w'}_{ij}\}_{(i,j)\in A}$ are equivalent weight sets. The weight set $\{\overline{w'_{ij}}\}_{(i,j)\in A}$ is the feasible set of the problem DUAL_TEMP and hold (15). Therefore, it is the optimal solution of this problem. So, we in this way, we were able to obtain optimal weights that do not include any link with weight 0 by limiting the constraint $w_{ij} \geq 0$ to $w'_{ij} \geq 1$. This converts the problem DUAL_TEMP to DUAL_II. Figure 2-1 shows the flow chart of our method.

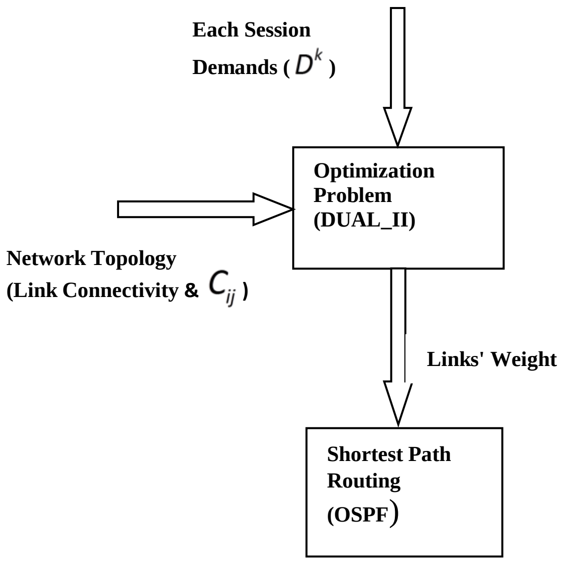

Each Session
Demands (D^k)
Optimization Problem (DUAL_II)
Network Topology
(Link Connectivity & C_{ij})
Links' Weight
Shortest Path Routing (OSPF)

Figure 2-1 Maximum Load Balancing Flow Chart

With ECMP routing a flow arriving at a node is split evenly over the links on the shortest paths from this node to the destination. It should be mentioned that arbitrary routing is not possible once ECMP in OSPF is used. So, in OSPF environment we can never obtain the optimal routing but we can get close to it as much as possible.

Objective function that is used in [6] is $\min MLU + r\sum_{k\in K}\sum_{(i,j)\in A} X_{ij}^{k}$. The second term in this objective function cause to minimizes $\sum_{(i,j)\in A} X_{ij}^{k}$ in addition to MLU. In this case, the weight set resulted from the dual function is $\{w_{ij} + r\}$ (where $\{w_{ij}\}_{(i,j)\in A}$ are Lagrange multipliers that correspond to the non-equal constraint). The routing algorithm that we use in this chapter is OSPF. This protocol splits the traffic equally among the available shortest paths, so we prefer traffic splitting as much as possible even if it passes through longer paths. As the constant r in the second term prevents the flow to go through long paths we assume that $r = 0$.

2.4 Simulation Results

In this section, we simulate the OSPF protocol with its default link metrics and with the metrics that are calculated using the optimization problem.

Scenario I: In first scenario the simulation platform is shown in figure 2-2.

All links in this network are DS3 with 44.7 Mbps rate. We suppose FIFO as a queuing policy.

The session is a VOIP with GSM quality and the average bit rate is 40 Mbps.

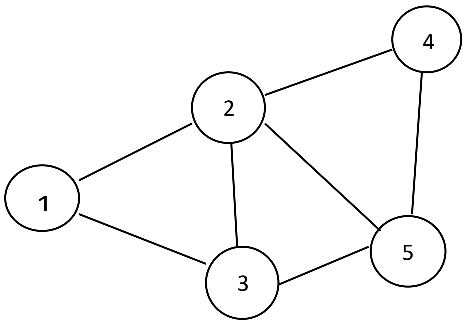


Figure 2-2 Simulation Network Topology

Node 1 is the source node and node 2 is the destination.

Table 2-1 shows the solution of primal problem (Xij) which indicate the paths that minimize maximum utilization. Solution of DUAL_I problem is shown in Table 2-2 and Table 2-3 show the solution of DUAL_II problem.

Table 2-1 Optimum flows

X12	20
X13	20
X23	0
X24	20
X25	0
X35	20
X54	20

Table 2-2 The Solution of DUAL_I

w12	0.0056
w13	0.0017
w23	0
w24	0.0056
w25	0
w35	0.0077
w54	0.0017

Table 2-3 The Solution of DUAL_II

W12	2.0177
W13	1.3567
W23	1.0000
W24	1.9823
W25	1.0000
W35	1.2843
W54	1.3591

Table 2-2 shows that the optimum paths are 1->2->4 and 1->3->5. These paths can obtain in a shortest path algorithm regarding to the weights that show in table 2-2. Table 2-2 and Table 2-3 are the equivalent weights with respect to the graph that shows in Figure 2-2. So, we use the suboptimal weigh set in table 2-3 instead of default OSPF link metrics. Figure 2-3 show that in recent method packet drop decrease significantly.

In following scenarios the simulation platform is shown in Figure 2-4. We compare MLU, BWE (Bandwidth Efficiency), Number of Over Utilized Links, IP Traffic Dropped and IP Traffic Received for all scenarios.

All links in our network are DS1 with 1.5 Mbps rate. We suppose FIFO as a queuing policy. All interfaces have a limited buffer size of 100 packets.

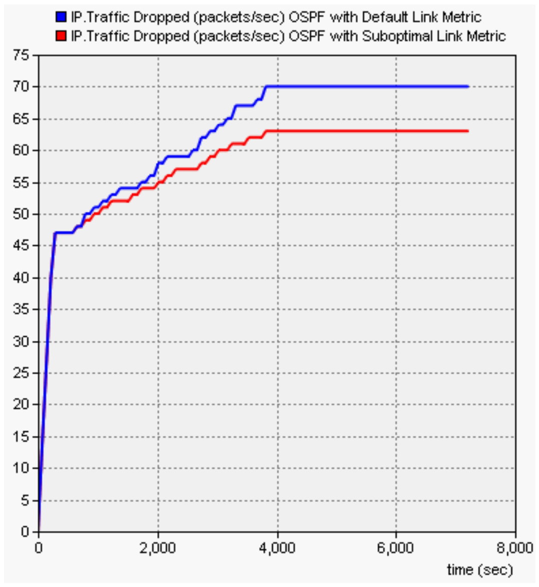


Figure 2-3 IP Traffic drop in default protocol and new method

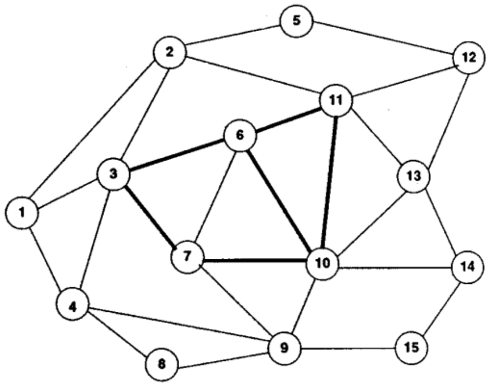


Figure 2-4 Simulation Network Topology

Scenario 2: In this scenario R1 is the traffic source and R13 is the traffic destination. Table 2-4 and table 2-4 show the Suboptimal Link Weights that obtained by DUAL_II.

Table 2-4 Solution of DUAL_II

W1_2	56.84	W3_6	1.000
W2_1	55.56	W6_3	1.000
W1_3	1.000	W3_7	1.000
W3_1	1.000	W7_3	1.000
W1_4	1.000	W4_9	1.000
W4_1	1.000	W9_4	1.000
W2_5	1.000	W4_8	1.000
W5_2	1.000	W8_4	1.000

W2_11	1.000	W5_12	1.000
W11_2	1.000	W12_5	1.000
W2_3	1.000	W6_11	1.000
W3_2	1.000	W11_6	1.000
W3_4	2.000	W6_10	1.000
W4_3	1.000	W10_6	1.000
W6_7	55.84	W10_13	1.000
W7_6	1.000	W13_10	1.000
W7_10	1.000	W10_14	1.000
W10_7	1.000	W14_10	1.000
W7_9	1.283	W11_12	1.000
W9_7	1.000	W12_11	1.000
W8_9	1.000	W11_13	1.000
W9_8	1.000	W13_11	1.000
W9_10	1.000	W12_13	1.000
W10_9	1.000	W13_12	1.000
W9_15	2.000	W13_14	1.000
W15_9	1.000	W14_13	1.000
W10_11	1.000	W14_15	1.000
W11_10	1.000	W15_14	1.000

MLU in new method decreases from 91.3 percent to 36.3 percent and BWE increases from 7.6 percent to 10 percent as shown in Table 2-5.

Table 2-5 MLU and BWE values

	Default Algorithm	Suboptimal Algorithm
MLU	91.3	36.3
BWE	7.6	10
Number of Over Utilized Link	0	0

IP Traffic Dropped and IP traffic Received do not change in this scenario because there is no congestion.

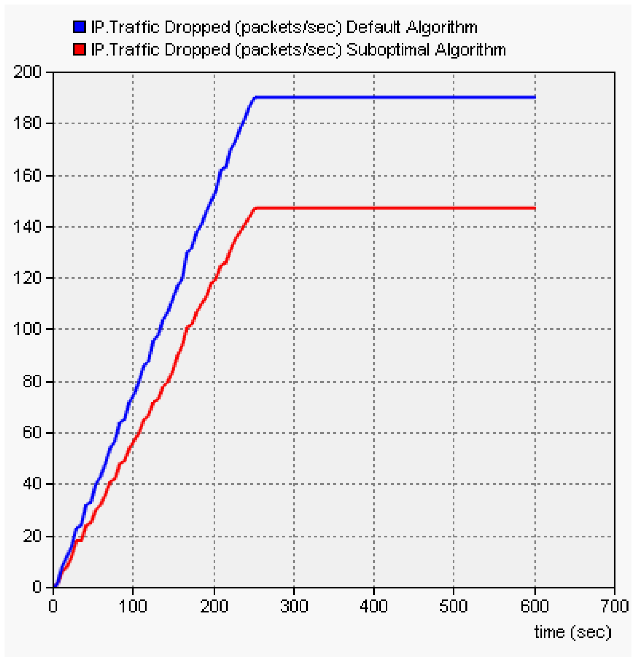


Figure 2-5 Scenario2 IP Traffic Dropped

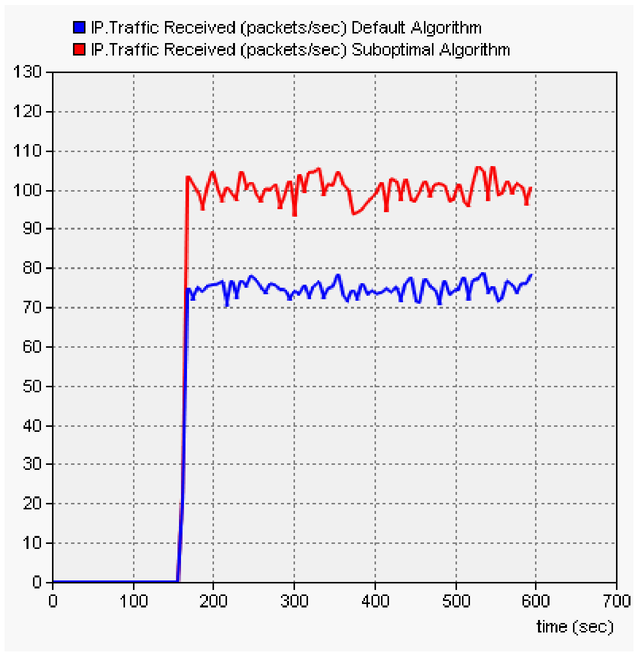


Figure 2-6 Scenario2 IP Traffic Received

Scenario 2: in this scenario we have three source-destination pairs (s_1, d_1) , (s_2, d_2) , (s_3, d_3) that are originated from R1 to R13, R5 to R9 and R4 to R2 respectively. MLU in the new method decreases from 137 percent to 91.3. The Number of over-utilized links also decreases from eight links to two links. BWE increases from 26.6 percent to 31.4 percent, table 2-6. Figure 2-5 and Figure 2-6 show the comparison of IP Traffic Dropped and IP Traffic Received

Table 2-6 MLU and BWE values

	Default Algorithm	Suboptimal Algorithm
MLU	137	91.3
BWE	29.6	31.4
Number of Over Utilized Link	8	2

In this chapter, we discus linear optimization methods which can helps Internet protocols work better. We use a duality theory to find a weight set that improve the routing protocols efficiencies. As a matter of fact, routing is the most important aspect of Internet Traffic Engineering. So, we focus on routing protocols and introduce a practical method that optimizes Link Metrics. Previous optimization methods suffer from practical issues but our method could be implemented with

Routing Protocols that based on shortest paths. Our simulation results show significant improvement on network efficiency [7].

2.5 Low Structural Congestion via Game Theory and Linear Programming

We propose a network topology design approach that targets the reduction of structural congestion in a directed acyclic network. What we mean by structural congestion is that a node has much higher in-degree than out-degree in a directed network. We approach the issue using a network design game model. In this model, we consider multiple sources and one destination. Each node is willing to connect to other nodes but it should pay the price of whole paths it uses to send traffic to the destination. The model yields a weight for each link. We show that if these weights are used to compute shortest paths, then a network topology is obtained with a low level of structural congestion.

The proposed method has two phases. In Phase I, we solve a linear optimization problem in order to find the optimum link weights. In Phase II, each node optimizes its own individual objective function, which is based on the weights computed in Phase I. We show that there exists a Nash Equilibrium which is also the global optimum. In order to measure the penalty incurred by the selfish behavior of nodes, we use the concept called price of anarchy. Our results show that the price of anarchy is zero.

Problem statement

Communication network design methods and algorithms are approached with various types of design goals. Minimum vulnerability, fault tolerance and quality of services are often used in this context [8].

As network nodes become more intelligent, distributed algorithms become increasingly dominant. Although a centralized algorithm which optimizes the entire network configuration would maximize efficiency and utilization, it is not as stable as distributed algorithm. Stability in a network means that if some nodes fail, other nodes have the capability to reconfigure themselves and recover from the failure. This idea can lead to a decision making algorithm that is executed in each node separately to optimize the global benefit.

One step further in this direction is when a node does not know the global benefit or does not care about it. In this situation a network involves selfish agents, making decisions to optimize their own benefit [9]. Social and biological networks are examples of such selfishly behaving agents that form a network. Game theory is a useful tool to analyze and predict the behavior of this kind of networks.

In this work, we study a directed acyclic network design game in the light of structural congestion consideration. Each node in a network which has a high in-degree is a bottleneck. It is desirable to avoid such a structural bottleneck, as it can easily lead to traffic congestion.

Our main objective here is to show that there exists a well-defined utility function in which the selfish behavior of each node leads to a network topology with minimum structural congestion. To do that first we convert a minimum structural congestion problem into a shortest path routing problem, in which link weights are obtained as the output of a linear optimization task. Then we construct a utility function in order to encourage each node to use paths with minimum overlap. The path set forms a new network which has a minimal structural congestion.

The rest of this chapter is organized as follows. After discussing related work in section II, we define the concept of structural congestion and optimization framework for analyzing network topology in section III. In section IV we derive a condition in which selfish behavior of each node can lead to an optimum. Finally, conclusion is presented in section V.

2.6 Related Work

The design of various networks has been studied in sociology, natural sciences and engineering for a long time [10]. Optimization and graph theory was the most useful tool in this field, since Myerson introduced a new network design model using game theory for social and economic networks. After that, the concept of game theoretic models has been used in different communication networking contexts, such as routing , flow control and dynamic access control in wireless networks [11].

Nash Equilibrium has been considered as a way to quantify the performance associated with selfish behavior of each player. Such equilibria are inefficient [11]. The lack of global control can lead to suboptimal network performance. The "price of anarchy" is a concept in game theory which measures the inefficiency of a system due to selfish behavior of each player.

A comprehensive study of game theory based communication network design is [8], which involves three important design considerations, namely the price of establishing a link, path delay, and path proneness to congestion. They showed that there exists an equilibrium point which is a global optimum.

The cost function which they considered in [8] for each player in a network design game considering path congestion is:

$$C(v_i) = \max_{v_k \in V} \max_{v_j \in l(v_i, v_k)} \eta_G^{in}(v_k) \qquad (2\text{-}4)$$

In which $\eta_G^{in}(v_k)$ is the input degree of a node v_k in a graph G, and $l(v_i, v_k)$ denotes the path connecting v_i and v_k. In this method, each node is required to connect to all other nodes and they show that a directed ring is both an optimum and equilibrium.

In this study, we focus on the structural congestion of the network. For our purposes, the network can be represented by a weighted directed acyclic graph.

2.7 Structural Congestion

A path in a network is a sequence of links, each link (except the first) having the same start node as the end node of the previous link in the sequence. Each link has a utilization factor, which we call Link Utilization (LU). If we view the network topology as a set of paths from a source to a destination, it contains several links which have different LU. A path's proneness to congestion is depending on the maximum LU on the path from a source to a destination. Let us look at a node v_i in the network, which is described using the graph $G(N, E)$. Let η_i^{in} and η_i^{out} be the input and output degree of v_i. We define the Degree Ratio (DR) for each node as follow:

Definition: The degree ratio of a node $i \in N$ is $DR_i = \dfrac{\eta_i^{in}}{\eta_i^{out}}$.

Assuming all links have unit capacity, the quantity DR_i shows the structural congestion at the node. High DR_i means node $i \in N$ is a bottleneck and can be a point of congestion. There is a direct relationship between DR_i in a network and the Maximum Link utilization (MLU) which is described in the following conjecture.

Conjecture: A set of paths in a directed acyclic network which minimizes MLU, will form a new network which minimizes (at least approximately) DR_i for all $i \in N$ and carries the same amount of traffic.

Minimizing MLU means finding a set of paths between a source and a destination, such that these paths split the input traffic as much as possible and, at the same time, have a minimum overlap. First, we analyze the problem of minimizing MLU, because it is a linear optimization problem. Consider a directed acyclic graph $G(N, E)$ which represents the network. $C_{ij} : (i, j) \in E$ is a set of edge capacities and (s_k, t_k) is a set of source-destination pairs for each session $k \in K$. The percentage of traffic on a link $(i, j) \in E$ that belongs to session k is X_{ij}^{k}. With these notations, the formulation is [8]:

$$\begin{aligned}
&\min_{X} MLU \\
&\sum_{j:(i,j)\in E} X_{ij}^{k} - \sum_{j:(j,i)\in E} X_{ji}^{k} = \begin{cases} 1 & i\varepsilon S \\ -1 & i\varepsilon T \\ 0 & otherwise \end{cases} \\
&\sum_{k\in K} X_{ij}^{k} \le C_{ij}.MLU \\
&X_{ij} \ge 0 \in \varepsilon
\end{aligned}$$

Using duality theory, we can write the dual optimization problem as follows:

$$
\begin{aligned}
&\max_{P,W} \sum_{k\in K} p_{t_k}^k \\
&\sum_{(i,j)\in E} C_{ij} w_{ij} = 1 \\
&p_i^k - p_j^k \le w_{ij} \\
&p_{s_k}^k \\
&w_{ij} \ge 0
\end{aligned}
$$

Because the primal and dual problems are both linear, strong duality holds and according to complementary slackness in the KKT theorem if $\hat{X}_{ij}^k$ is an optimal solution for the primal problem, and $\{\hat{w}_{ij}, \hat{p}_{ij}^k\}$ is an optimal solution for the dual, then we have:

$$\hat{X}_{ij}^k.(\hat{p}_i^k - \hat{p}_j^k + \hat{w}_{ij}) = 0$$

This equation indicates that if session k uses link $(i,j)\in E$ then $p_j^k - p_i^k = w_{ij}$. According to the Duality Theorem, if $\{\hat{w}_{ij}\}_{(i,j)\in E}$ is used as link metric in a shortest path algorithm, all non-empty links $(X_{ij}^k > 0)$ will be selected in a shortest path algorithm procedure. As a result if any shortest path algorithm uses $\{\hat{w}_{ij}\}_{(i,j)\in E}$ as link weights we will have set of paths between a source and destination which has an important character. The path set splits the input traffic as much as possible through the network and at the same time has minimum number of overlap links.

A network topology with minimum structural congestion means that DR_i is close to one. Let us consider a weighted directed acyclic graph which represents a network with only one source-destination pair and the capacity of all links are 1. Weights are calculated on the basis of the dual optimization problem discussed above. If we run any shortest path algorithm over such a network we obtain a set of paths Ω. If we delete any link $(i,j)\in E$ which is not on a member of Ω we will have a new weighted acyclic graph which represents a new network. Based on the following theorem, the new network has minimum DR_i.

2.8 Network Design Model

In this section, we study the performance of a non-cooperative network. This means, each node (player) tries to maximize its own benefit. The network design goal is minimizing the structural congestion. Node v_i gains α_i by connecting to any node in the network. So, each node tries to make a connection to as many nodes as possible. By connecting to each node, it must calculate the length of a path from itself to a destination. The gain a node can achieve by connecting to others minus the summed length of all paths heading to destination form the utility function of each node as follows.

Node Utilization:

$$u_G(v_i) = \alpha_i |S_i| - \sum_{p \in P} d^{p}_{G(W)}(v_i, v_t)$$

Which $|S_i|$ is the number of output links in node v_i and $d^{p}_{G(W)}(v_i, v_t)$ is a distance between node v_i and the destination using path $p \in P$ in the designed network using links weight W.

The network utilization is the sum of all node utilization functions.

Network Utilization:

$$U_G = \sum_{i \in N} u_G(v_i)$$

Optimum solution for such a game happens when we have a maximum $U_G = \sum_{i \in N} u_G(v_i)$. But in order to find equilibrium point we need to analyze the selfish behavior of each node. For that purpose, consider Figure. 2-7 as a part of a network. Node v_i is deciding to stay on its current strategy (connection to other nodes) or deviate (drop a connection or make a new one) based on the maximum utilization function.

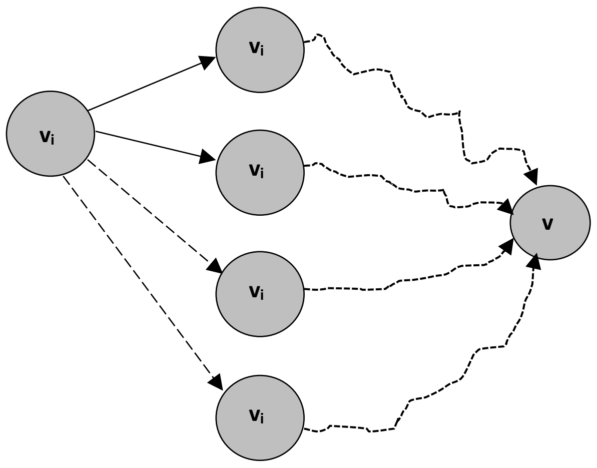


Figure 2-7 Node Vi decision strategies

Consider node v_i in the network. It is already connected to nodes v_{i+1} and v_{i+2}. It should decide to connect to nodes v_{i+3} and v_{i+4} or not. The current topology is represented by the graph G_1, if it connect to v_{i+2} the graph will be G_2 and if it connect to both v_{i+1} and v_{i+2} the graph will be G_3. Based on the weight system in the network the distance from nodes $v_{i+1}, v_{i+2}, v_{i+3}$ and v_{i+4} to destination are $l_{i+1}, l_{i+2}, l_{i+3}$ and l_{i+4} respectively. The utility of node v_i is:

$$u_{G_1}(v_i) = 2\alpha_i - (w_{i,i+1} + l_{i+1} + w_{i,i+2} + l_{i+2})$$

$$u_{G2}(v_i) = 3\alpha_i - (w_{i,i+1} + l_{i+1} + w_{i,i+2} + l_{i+2} + w_{i,i+3} + l_{i+3})$$

$$u_{G3}(v_i) = 4\alpha_i - (w_{i,i+1} + l_{i+1} + w_{i,i+2} + l_{i+2} + w_{i,i+3} + l_{i+3} \\ + w_{i,i+4} + l_{i+4})$$

Suppose that based on the weight system, links $(i, i+1)$, $(i, i+2)$ and $(i, i+3)$ are on the shortest paths. So, we have:

$$l^i_{Sh-P} = w_{i,i+1} + l_{i+1} = w_{i,i+2} + l_{i+2} = w_{i,i+3} + l_{i+3}$$

If we want that the selfish behavior of the node v_i leads to optimum topology, then the following conditions must hold:

$$u_{G_1}(v_i) < u_{G2}(v_i)$$

$$u_{G2}(v_i) > u_{G3}(v_i)$$

So, we have:

$$w_{i,i+1} + l_{i+1} < \alpha_i < w_{i,i+4} + l_{i+4}$$

If we consider λ^i_{Sh-P} as the length of a second shortest path from the node v_i to the destination we have:

$$l^i_{Sh-P} < \alpha_i < \lambda^i_{Sh-P}$$

$$l^i_{Sh-P} < \alpha_i < \lambda^i_{Sh-P}$$

This is the condition in which selfish behavior of each node in the network will lead to optimum topology with minimum structural congestion. Now the question is if there is any upper and lower bound for α in general. Using topological sorting theorem [12] we can find such a bound. Based on topological sorting theorem a directed acyclic graph can be represented in way that nodes index increase when they get closer to the destination and there is no link (m,n) if $m > n$. For example, a directed acyclic graph with 4 nodes after topological sorting is shown in figure 2-8.

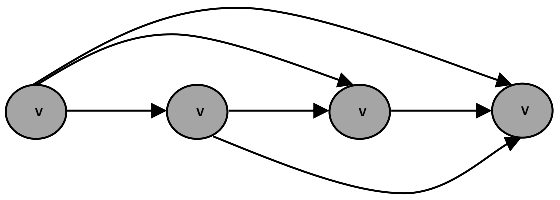


Figure 2-8 Topological Sorting

After topological sorting, we suppose that node 1 is the source and node N is the destination. Now it is clear that after using weight set which is the solution of dual optimization problem in section III we have $l^i_{Sh-P} > l^{i+1}_{Sh-P}$. So, the lower bound for α is l^1_{Sh-P} which is the shortest path from source to the destination. Also, we have $\lambda^i_{Sh-P} > \lambda^{i+1}_{Sh-P}$. So, the upper bound for α is:

$$\lambda^{N-2}_{Sh-P} = \max\left\{w_{N-2,N-1} + w_{N-1,N}, w_{N-2,N}\right\}$$

So, we have:

$$l^1_{Sh-P} < \alpha < \lambda^{N-2}_{Sh-P}$$

Now consider the network in figure 2-7. The question is what is the upper and lower bound for α in this network. Table I shows the optimal weights which calculate using dual optimization problem in section III.

Table 2-7 Optimum Weights

$p_1 - p_2 = w_{12}$	1
$p_1 - p_3 = w_{13}$	1
$p_1 - p_4 = w_{14}$	2
$p_2 - p_3 = w_{23}$	3
$p_2 - p_4 = w_{24}$	1
$p_3 - p_4 = w_{34}$	1

In this case upper and lower bound is:

$$l^{l}_{Sh-P} = 2$$

$$\lambda^{N-2}_{Sh-P}=4$$

So $\alpha = 3$ satisfies the condition. After applying $\alpha = 3$ in the node utility function, node v_2 can improve its utility function by deviate from current strategy to the one which has no connection to node v_3. As a result, we have network with better structural congestion. Applying this method to all nodes the result would be a network topology with minimum structural congestion.

In order to analyze the price of selfish behavior there is two important concepts which are price of stability and price of anarchy. The price of stability is the ratio between best objective function value in equilibrium point and the optimum network utilization function. On the other hand, price of anarchy is the ratio between worse objective function value in the equilibrium and the optimum network utilization function [13]. In this section, we showed that price of stability is one and anarchy is free if each node applies the node utilization function. Otherwise price of anarchy is depending on α and $\{w_{ij}\}_{(i,j)\in E}$.

Figure 2-9 shows how the optimization method provides inputs for our network design game.

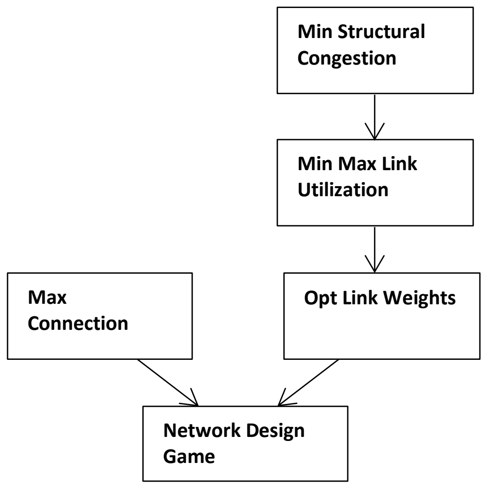


Figure 2-9 Algorithm Flowchart

It is worth mentioning that the described method can be implemented in a network using distributed algorithms like the Bellman-Ford Algorithm [14]. It means that it is not necessary for each node to have information about the whole network. It is only needed to know the parameter α, the weights of its outgoing links and the distance of its neighbors to the destination. Having this information is sufficient to find an optimum strategy.

2.9 Simulation

For the simulation, we consider a directed acyclic network with 20 nodes. All links have a capacity one and we consider node 1, 2 and 3 as a source of traffic and node 20 as the destination. Figure 2-10 shows the network topology. Maximum degree ratio is 19 in this network. Each node minimizes its own objective function based on optimum link weights and its desire to make more connection.

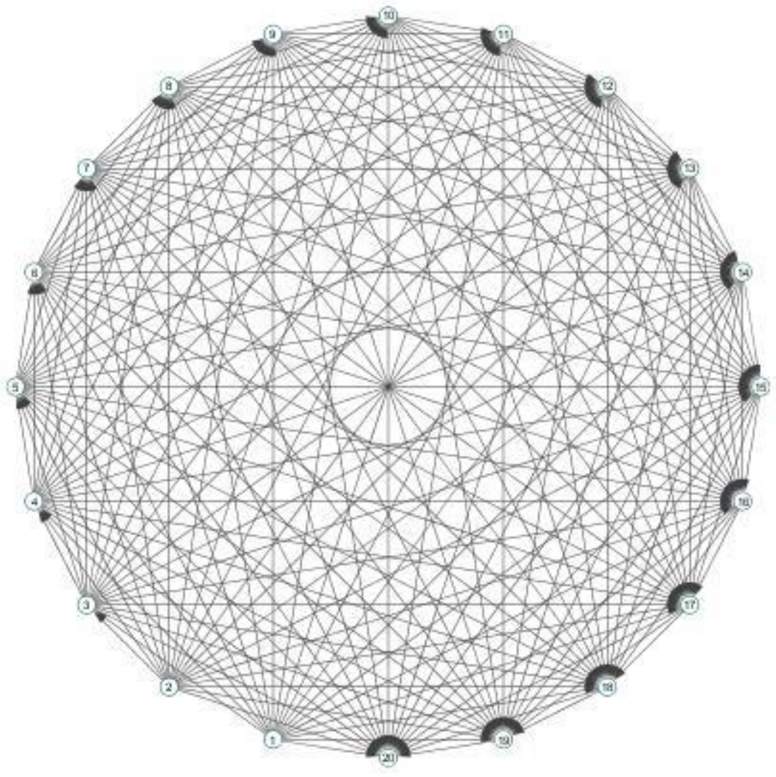


Figure 2-10 Network Topology

After solving the dual optimization problem, we have $l^{l}_{Sh-P} = 4.3$ and λ^{18}_{Sh-P}=14.5. Figure 2-11 shows that no structural congestion is a result of choosing $4.3 < \alpha < 14.5$, it means that $DR_i = 1$ for all $i \in N$. As α deviates from the constraint each node is more willing to make a new connection and it leads to more structural congestion. For example, if we choose $\alpha = 20$ degree ratio of nodes 16 and 17 are high and they can be considered as a network bottleneck.

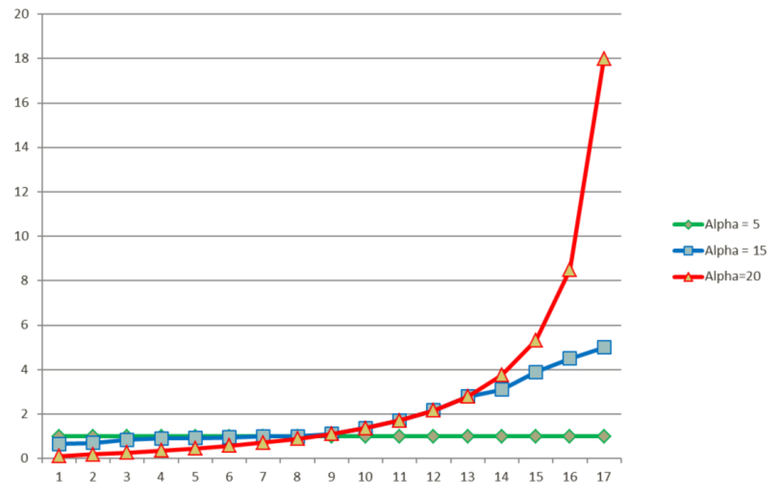


Figure 2-11 Degree ratio for each node using different alpha

This section investigates the question "how non-cooperative nodes in a network can create an efficient network?" We have studied the result of the selfish behavior of nodes, and compares it to the situation in which there is a central control unit in the network. Central control can force all nodes to use a predefined strategy in which the network utilization is optimum.

Based on the discussion in section IV if we fix the benefit of establishing a new link for each node, α, in a way that satisfies the condition $l_{Sh-P}^{l} < \alpha < \lambda_{Sh-P}^{N-2}$, the price of stability will be one and also the price of anarchy will be zero in this network design game [15] [16].

CHAPTER 3

AN OPTIMAL TRAFFIC DISTRIBUTION METHOD SUPPORTING END-TO-END DELAY BOUND

Routing methods for optimal distribution of traffic in data networks that can also provide quality of service (QoS) for users is one of the challenges in recent years' research on next generation networks. The major QoS requirement in most cases is an upper bound on end-to-end path delay. In multipath virtual circuit switched networks each session distributes its traffic among a set of available paths. If all possible paths are considered available, then the source's decision on its traffic distribution can be considered as routing. A model of the routing function as a mathematical problem which distributes the input traffic over possible paths for each session is proposed here. A distributed and iterative algorithm which will keep the average end-to-end delay for individual paths below a required bound is introduced. This algorithm minimizes the total average delay of all packets in the network. The convergence of the algorithm is illustrated [17].

Computer networks have evolved into a new generation where a wide range of new services are provided to various network users. For many of these new services, such as VOIP, IPTV, Network Games, etc., it is not sufficient just to transfer the information to the destination, but for the users' satisfaction it is necessary to guarantee their required QoS as well. In this manner, the new services with arbitrary QoS require ments can be deployed in the network. Providing the QoS must be achieved by utilizing the least possible resources of the network such that the network can be optimized in terms of resource utilization [18]. Network optimization algorithms determine traffic distribution for a given traffic demand so that the optimum re- source utilization can be achieved.

But the research results so far show that providing QoS in cases where routing is performed without paying attention to the QoS requirements is difficult. Therefore, considering the required QoS in the optimization algorithms and determining the routes accordingly is one of the challenges of the next generation networks.

Recently several new services have become popular in the internet qualities of which depend on the end-to- end delay experienced by the packets in the network. For an acceptable QoS it is required that the end- to-end delay is kept under a threshold level. Providing QoS is not an easy task in datagram networks. In new generation networks, virtual circuit switched networks such as MPLS is used to provide a better framework to implement QoS.

Most of the QoS provisioning algorithms in the literature exploit certain mechanisms to guarantee the delay for a given path. Nen Jin, et.al show that for providing QoS in a DiffServ network, the price per unit of traffic rate for each traffic class can be adjusted. They assume a given path for a user. The satisfaction of the user is modeled through a convex function of the traffic passing through that given path and the QoS level of the assigned traffic class. In [19] QoS is proposed to be provided by adjusting the capacity allocated to each DiffServ class. The QoS measure is the exact proportion of the average delay of two different traffic classes. Each user's traffic is routed through a predetermined path and depending on the amount of traffic of each class, the traffic over this path experiences a delay which is considered as its cost. In [20] a dynamic method is used to adjust the users' traffic rate in a manner that a minimum rate and a maximum delay threshold are guaranteed. A predetermined path used for routing the traffic and its rate is determined by solving a convex optimization problem which satisfies the user's delay requirements.

Most of the articles that study the traffic distribution in virtual circuit switched networks assume a set of known paths for each source-destination pair. To simplify the problem, usually, a small set of paths is selected from all possible paths beforehand. In the articles that find routes based on QoS requirements, the QoS is mostly measured based on parameters. Each QoS parameter for a path is sum of the QoS parameters of its links. The links are modeled by an m-dimensional weight vector W = (w_1, ..., w_m) the components of which represent the QoS parameters of links. Paths with QoS parameters lower than the threshold levels will satisfy the required QoS and can be selected. In this manner, the QoS-based routing problem is modeled as a multi-constraint (optimal) problem. Since these problems are NP-hard, in most cases heuristic methods are adopted in solving them.

Here the objective is to introduce a scalable method in terms of the number of sessions, in order to dis- tribute the network's traffic over available paths in a virtual circuit switched network that would minimize the average delay for all packets as the total cost of the network, while guaranteeing a bounded end-to-end path delay as the users, QoS requirement. The proposed method in this article is based on the analysis of the traffic distribution problem with delay constraints. As a result, this problem is modeled as a constrained convex optimization problem and the routing algorithm is provided in accordance to the analytical solution of this problem.

In Section 2 an analytical model for distributing traffic is introduced where the traffic distribution is modeled as a constrained convex optimization problem. In Section 3 the Lagrangian dual method is adopted for solving this problem. An algorithm that can be realized in a data network based on the dual method is proposed here. In Subsection 3.1 the implementation method of the proposed algorithm in real networks is explained. In section 4 the simulation results are provided expressing that this proposed method con- verges and can achieve its objective in an effective manner. This article will be concluded in Section 5. The analysis of the proposed model is provided in the Appendix A.

3.1 Traffic Distribution Model

The objective in common for all the routing algorithms is to determine the appropriate paths for carrying the users' traffic from source to destination. All or part of each user's traffic is assigned to each selected path; therefore, a direct output of a routing algorithm is the amount of traffic allocated to each path. In fact, routing can be modeled as a mathematical problem which determines the distribution of all sessions' traffic over the network graph.

In this article source-destination pairs are assumed to be known and are presented by the set W. Each source-destination pair w 2 W is considered as a session and its average input traffic is presented by r_W. A data network is modeled as a stationary and directed graph G(A, V). The graph nodes, represented by set V model the network routers or gateways and graph links represented by set A, model the physical links between the routers. Some of the nodes of the graph are source or destination of the sessions in the network.

A session path is a set of links that connects the source of the session to its destination. The set of the paths of each session is called $P_{W,}$ Figure 3-1. Thus, the routing problem is similar to finding the distribution of each session's traffic over its paths.

The parameters and notations which are used in the rest of this article are introduced in the following Nomenclature:

• W: The set of all existing sessions, where N_W shows the total number of these sessions

• P : The set of available paths of all sessions w 2 W in G(A, V),where N_P shows the total number of these paths

• P_W: The set of available paths of session w

• r_W: Average traffic rate of session w

• x_p: Traffic assigned to path p 2 P

• X: A vector of N_P components whose pth component is the assigned traffic to path p, x_p

• $_p$: The lagrangian multiplier according to the delay constraint of path p

• ⇤: A vector of N_P components whose pth component is the $_p$

• th_p: The threshold level of average delay of packets in path p

• Th : A vector of N_P components whose pth component is th_p

• f_{ij} : The flow crossing from link (i, j) of G(A, V)

• $h_p(X)$: The cost function associated with path p

• H(X): A vector of N_p components whose pth component is $h_p(X)$

- $D_{ij}(f_{ij})$: The cost function associated with link (i,j)

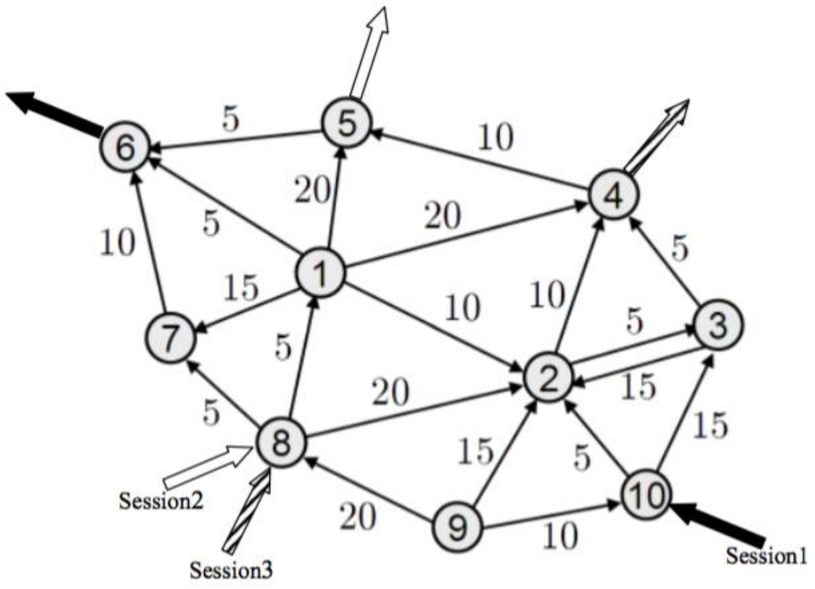


Figure 3-1 A network graph with three sessions

Based on the above definitions the following relations hold:

$$x_p \geq 0 \qquad \forall p \in P \quad (1) \qquad (1)$$

$$\sum_{p \in P_w} x_p = r_w \qquad \forall w \in W, \forall_p \in P_w \qquad (2)$$

$$f_{ij} = \sum_{p|(i,j)} x_p \qquad \forall (i,j) \in \qquad (3) \qquad\qquad (3\text{-}1)$$

$$h_p(X) = \sum_{(i,j) \in p} D_{ij}(f_{ij}) \qquad \forall p \in P \qquad (4)$$

If the average delay of the packets over a link is considered as the link's cost function, $D_{ij}(f_{ij})$, and the messages are delayed only by the links of the network, then (3-2) expresses the expected delay for all packets over the network [3]. Equation (3-2) indicates the average time that packets remain in the network and use network resources; thus, it can be considered as the overall system cost.

$$D = \sum_{(i,j)\in A} D_{ij}(f_{ij}) \qquad \text{(3-2)}$$

Even in a virtual circuit network minimizing (3-2) can be a good objective for traffic distribution since it can improve network resource utilization [21] [22]. In the virtual circuit switched networks, each session's traffic is distributed among the available paths. By assuming a stable network and assuming that the traffic of the sessions is stationary, this problem is modeled and analyzed as the problem of distributing the average input traffic of each session r_W , over the set of session's paths P_W, which will result in the sessions' path flows x_p, for all sessions. Thus, f_{ij}, the total flow of link (i, j), can be expressed by the different path flows. As a result, f_{ij} equals the sum of all path flows traversing link (i, j), (3-1-3). Here each session represents a customer. The expectation of each customer from the network is defined based on the customer's traffic's delay tolerance. In this case, the customer will be satisfied if the average delay is bounded to a certain threshold. Therefore, considering the delay of each link as its cost is deemed to be appropriate. In this model, the sum of the cost function of the links which compose a path, is considered as the path cost, $h_p(X)$, which is equal to the sum of the costs of the path's links

(4). Considering (3-2) as the overall cost function of the network and (3-1-4) as the customer cost, the limitation of which is required by the customers, the routing in the network can be modeled as

Problem 1.

Problem 1.

$$\min_{x} D(X) = \sum_{(i,j)\in A} D_{ij}\left(\sum\nolimits_{p|ij\in p} x_p\right) \qquad (1)$$

$$\sum_{p\in P_w} x_p = r_w \qquad \forall w \in W, \forall p \in P_w \qquad (2) \qquad\qquad (3\text{-}3)$$

$$x_p \geq 0 \qquad \forall p \in P \qquad (3)$$

$$h_p(X) \leq th_p \qquad \forall p \in P \qquad (4)$$

In this problem, the path flows x_p, are the variables. The objective function D(X) is considered as the overall system cost. The purpose of this problem is to find the distribution of the traffic among the available paths in order to minimize the overall system cost while the constraints (3-3-2) to (3-3-4) are satisfied. Constraints (3-3-2) and (3-3-3) guarantee the acceptable allocation of the traffic over the session's paths, and constraint (3-3-4) guarantees the delay limitation or users' expectation. If constraint (3-3-4) is ignored, Problem 1 is converted to Problem 2.

Since the cost functions $h_p(X)$ are convex, Problem 1 is a constrained convex optimization problem [5], which can be solved using any of the existing methods, such as Projected Gradient, Interior Point, etc. But here the objective is to find a solution that can also be implemented in real networks. In this regard, the Lagrange dual problem is formulated and solved. In other words, since Problem 1 is a convex optimization problem the duality theorem is adopted in solving it. The fact that strong duality holds is presented in Proposition 1.

Problem 2.

$$minimize D(X) = \sum_{(i,j)\in A} D_{ij}(\sum_{p|ij\in p} x_p)$$

$$\sum_{p\in P_w} x_p = r_w \qquad \forall w \in W, \forall p \in P_w \tag{3-4}$$

$$x_p \geq 0 \qquad \forall p \in P$$

3.2 Solving the Optimization Problem

Usually the cost function $D_{ij}(f_{ij})$ is expressed as a convex, non-decreasing, continuous and differentiable function; therefore, the path cost will have the above characteristics. Since the cost functions $h_p(X)$ are con- vex, Problem 1 is a constrained convex optimization problem [5], which can be solved using any of the existing methods, such as Projected Gradient, Interior Point, etc. But here the objective is to find a solution that can also be implemented in real networks. In this regard, the Lagrange dual problem is formulated and solved. In other words, since Problem 1 is a convex optimization problem the duality theorem is adopted in solving it. The fact that strong duality holds is presented in Proposition 1. Since there is a practical solution to solve Problem 2 [15], the dual problem is described using the Lagrange multipliers related to (3-3-4). Thus the Lagrangian is (3-5) where only constraint (3-3-4) is relaxed by introducing Lagrange multiplier $_p$ for each path. The resultant partial dual function is Problem 3.

$$L(x,\Lambda) = D(X) + \sum_{p\in P} \lambda_p.\left(h_p(X) - th_p\right) \qquad \forall \Lambda \geq 0 \tag{3-5}$$

Problem 3.

$$q(\Lambda) = minimizeL(x, \Lambda)$$

$$\sum_{p\in P_w} x_p = r_w \qquad \forall w \in W, \forall p \in P_w \qquad (3\text{-}6)$$

$$x_p \geq 0 \qquad \forall p \in P$$

Considering Problem 3 as the dual function of Problem 1, the dual problem will be Problem 4.

Problem 4.

$$\max q(\Lambda)$$

$$\lambda_p \geq 0 \qquad \forall p \in P$$

As mentioned in Proposition 2, the q(⇤) is a con- vex function which is not necessarily differentiable in general, but it is sub-differentiable at all points. There- fore, Problem 4 can be solved iteratively by adopting the sub-gradient method [22]. In this method, an initial value is given to variable Λ, (Λ^0), and in each iteration according to (11) a new value is calculated which will be closer to the optimum value.

$$\Lambda^{k+1} = [\Lambda^k + \alpha^k . g^k]^+ \qquad (3\text{-}7)$$

To calculate the new value of ⇤ in the kth iteration, first a sub-gradient of function -q(Λ) called -g^k is calculated at Λ^k, and then Λ^{k+1} is calculated by using (3-7) where, α^k is a positive step size and ”+” denotes projection on the set R^+. As result-4 indicates, in order to find a vector g^k the traffic must be distributed based on Problem 3 solution according to $\Lambda = \Lambda^k$, denoted by $X^{⇤}(\Lambda^k)$. In this case, the deviation of the cost of a path from its threshold th_p, is equal to the associated component of g^k, (3-8).

$$g^k = h_p\left(X^*\left(\Lambda^k\right)\right) - th_p \qquad (3\text{-}8)$$

Eventually, the iterative algorithm finds $\Lambda^{⇤}$ which is the best solution for Problem 4. Obviously in this iteration the input traffic is distributed similar to that of the path flows which are the solution of Problem 3 for the amount of $\Lambda = \Lambda^{⇤}$. Since the conditions for strong duality exists according to Proposition 1, this distribution will be the optimum solution of Problem 1 as well. In the following section the proposed algorithm is explained.

Algorithm Steps:

Step1: A feasible value is given to Λ. Since in Problem 4 every $\Lambda >= 0$ is acceptable, the $\Lambda^0 =0$ is used as the initial value. In this step, the initial value of q^{best} is 0.

Step2: In iteration k, Problem 3 must be solved based on the value of Λ^k, leading to the optimum value $q(\Lambda^k)$ and the optimum point $X^{⇤}(\Lambda^k)$. The components of this vector are represented by $x^{⇤}{}_p(\Lambda^k)$. In other words, a mechanism must be used to determine path flows, for the optimal routing problem when (3-9) is considered as the cost function of each link. There- fore the Lagrange multipliers can be interpreted as the bottleneck indicators of the paths.

$$D_{ij}^k = \left(1 + \sum\nolimits_{p|ij\in p} \lambda_p^k\right).D_{ij}\Big(\sum_{p|ij\in p} x_p^*(\Lambda^k)\Big) \qquad (3\text{-}9)$$

Step3: In iteration k with respect to the value of X ⇤ (Λ^k) which is calculated in step2, the deviation of each path's cost from the threshold level of the same path is calculated. Considering the Proposition 3, the negative of this value can be considered as the pth component of the sub-gradient vector of q(Λ) at Λ^k or $-g^k$. After calculating the deviation for all paths, the p value of Λ for next iteration or Λ^{k+1} can be calculated using (3-8).

Step4: The value of $q^{best} = \max\{q^{best}, q(\Lambda^k)\}$is calculated and k is increased by one. Then if the condition of ending the algorithm is met, the algorithm terminates, otherwise, it goes back to step2 for next iteration.

Condition of ending the algorithm: In a simple case, the condition which leads to the algorithm termination can be the maximum number of iterations (Figure 3-2).

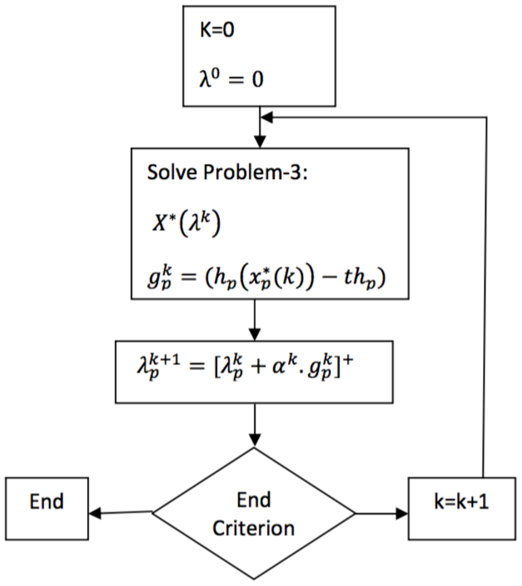

K=0
$\lambda^0 = 0$
Solve Problem-3:
$X^*(\lambda^k)$
$g_p^k = (h_p(x_p^*(k)) - th_p)$
$\lambda_p^{k+1} = [\lambda_p^k + \alpha^k . g_p^k]^+$
End
End Criterion
k=k+1

Figure 3-2 The flowchart of flow distribution algorithm

Matching the algorithm with real networks

As mentioned before, the main objective of this article is to distribute the input traffic of a session over its known paths. A session can be equivalent of a source and destination pair in virtual circuit switched networks such as ATM and MPLS, or in general in any network that uses explicit routing or source routing. Even a certain DiffServ class traversing the same LSP in these networks can be considered as a session. In practice this proposed algorithm is implemented for each session iteratively and in parallel for all sessions.

Here each iteration of the algorithm is assumed to be performed in one time slot. At the end of a time slot, destination nodes calculate the deviation of the average delay for each path from the required delay bound. The bottleneck multiplier of the path is calculated based on its cost deviation and is sent to the source node. The average delay of packets in each iteration can be determined by the destination either using analytical modeling or just by measurement. In a case where the path delay is estimated by using measurement methods, based on the assumptions about the link cost in this article, this proposed algorithm will definitely converge according to the Proposition 4. During each time slot the source nodes distribute the input traffic according to the optimal point of Problem 3. In each iteration, the Problem 3 is an optimal routing problem where the cost function of each link is defined by (13). This problem can be solved by one of the existing methods.

Each time slot can be in the order of the end-to-end trip time in the network. The algorithm is scalable because it is implemented independently for each session. If the set of the paths for each session can be assumed to include all possible paths for the session based on the topology of the network, the algorithm will practically select the routes; therefore, a separate method for determining the possible routes will not be necessary.

3.3 Simulation

The algorithm for two sessions is simulated over the network graph in Figure 3-3. The algorithm is executed independently for each session in an iterative and synchronized manner. All possible paths for session 1 are P1(14a), P2(14b) and P3(14c) and for session 2 are P4(14d), P5(14e) and P6(14f).

$$P1 = \{(1,2),(2,4)\} \quad (14a)$$
$$P2 = \{(1,2),(2,3),(3,4)\} \quad (14b)$$
$$P3 = \{(1,3),(3,4)\} \quad (14c)$$
$$P4 = \{(2,3),(3,4),(4,5)\} \quad (14d)$$
$$P5 = \{(2,4),(4,5)\} \quad (14e)$$
$$P6 = \{(2,5)\} \quad (14f)$$

In this simulation, the average delay of the links is modeled as (15) which is a convex, continuous, and differentiable function of its average traffic. In this equation, C_{ij} is the capacity of the link (i, j) and K_{ij} is a positive coefficient of the link. The domain of this function covers the traffic flows between 0 and C_{ij} only and as the flow gets closer to C_{ij} the delay increases exponentially. The function is undefined for values equal to or above C_{ij}. The coefficient and capacity of the links of Figure 3-3 are proposed in Table 3-1.

$$D_{ij}(f_{ij}) = \frac{(K_{ij} * f_{ij}^2)}{(C_{ij} - f_{ij})} \quad (3\text{-}10)$$

Table 3-1 Parameters of the Network links

Link(i,j)	K(i,j)	C(i,j)
(1,2)	2	44.7
(1,3)	4	16
(2,3)	3	44.7
(2,4)	1	44.7
(2,5)	8	44.7
(3,4)	4	44.7
(4,5)	2	16

The constant input traffics are used in the simulation as the expected values of the sessions traffics in general. The average input traffic for each session is assumed to be 20 Mbps. In this simulation, the attempt is made to clarify two important points: to show that the iterative algorithm converges to the optimal point of Problem 1 and that this algorithm achieves its objective in limiting the end-to-end delay of the paths in addition to minimizing the total network delay. Since the main objective of this proposed model is similar to the optimal routing problem, the Problem 2, the results of the proposed algorithm are compared with the Problem 2, for the above scenario.

In the first step, the path flows for each session are calculated based on solving the optimal routing problem, the Problem 2, by applying CVX package in MATLAB. In this case, the end-to-end delay for each path as well as the expected delay of packets are calculated (see Table 3-2).

In the second step, the path flows for each session are calculated based on the optimal routing problem with end-to-end delay constraint, Problem 1. The end-to-end delay bound for each path is assumed to be 76 units in this simulation. The path flows are calculated by solving Problem 1 applying CVX package in MATLAB (see Table 3-3).

The total cost of the network in step 2 is slightly higher than the optimum total cost in step 1. Yet in step 1 the individual path cost, for paths 1 and 6, is beyond the end-to-end delay bound. This means that this proposed algorithm is able to limit the delay with a minimum increase in the total cost. Also it can be seen that based on the Complementary Slackness condition, x_p of paths 1 and 6 is decreased from the optimum values of step 1, down to a point that their average delays are decreased to the threshold level. As such, the optimum dual variable, DV, of these two paths is expected to be higher than zero while DV of the other paths expected to be zero. It can be interpreted that the marginal cost of the paths 1 and 6 should be lower compared to that of path 3 for the calculated traffic.

In the final step, the proposed algorithm is simulated through MATLAB. Here the step size is 0.008. The simulation finishes after 1000 iterations. The final results of the algorithm are presented in Table 3-4. The stepwise results of the algorithm for Lagrange multipliers and two of the link flows as a sample are presented in Figure 3-4 and Figure 3-5.

The results in Table 3-4 are the same as the results in Table 3-3. This means that the iterative algorithm con- verges to the same results of the centralized solution [23].

Figure 3-4 shows that the path flows converge to the same results as the results of the case where the Problem 1 is solved in a central manner [24].

Figure 3-5 shows that the Lagrangian multipliers of the distributed solution converge to the optimal dual variable values obtained from the centralized solution of the Problem 1.

3.4 Mathematical Analysis

In this section, the analysis of the proposed algorithm is provided. First some parameters used in this section are defined

- x_p:Flowofthepathpthatisheldin Assumption1

- $H(X)$: Cost vector of all sessions with N_P components where the pth component represents the cost of the pth path

- $A(X)$: Deviation vector with N_P components where the pth component represents the deviation of the pth path from its threshold

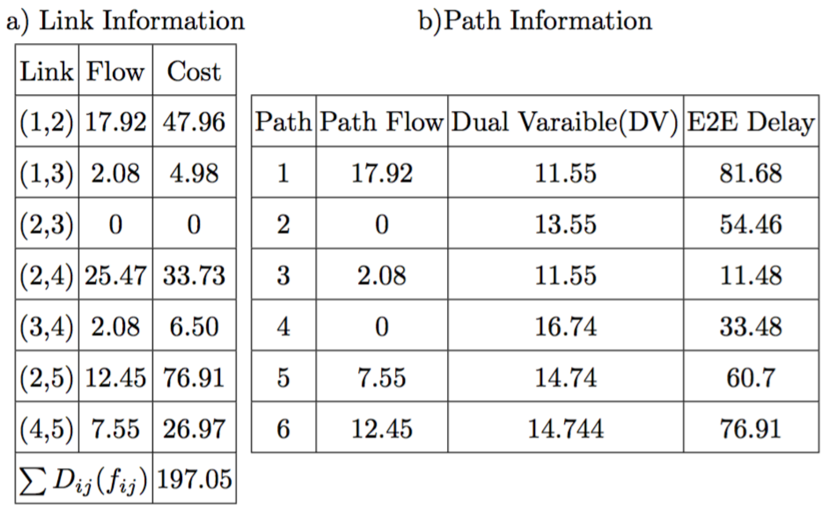
- Th: Threshold vector with N_P components and the pth component represents the maximum delay bound of the path p

- $\Lambda^{\leftarrow}$: Optimum solution of Problem 4 which is a vector with N_P components

- $\Lambda^{\leftarrow}{}_p$: The pth component of the optimum vector Λ which is the optimum Lagrange multiplier of the pth path.

Result-1: Since the feasible set of the Problem 1 is not empty, this problem has at least one optimal point. Proposition 1. The optimum solution of Problem 4 is equal to the optimum solution of Problem 2.

Table 3-2 Simulation results of step 1



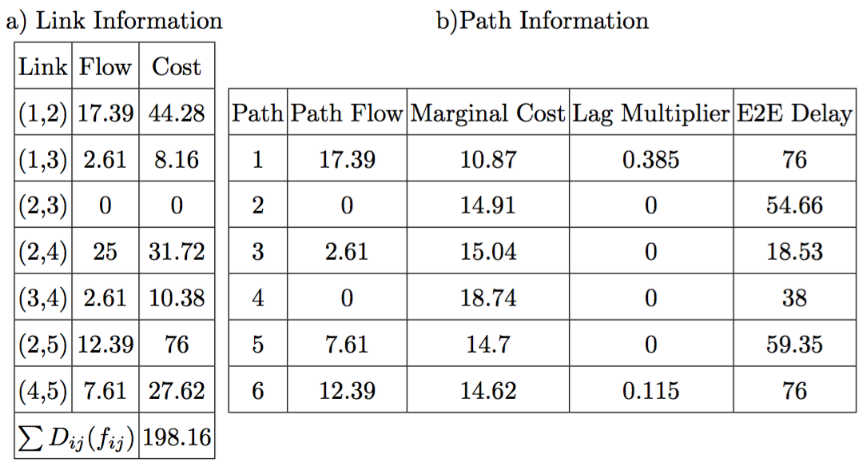
a) Link Information

Link	Flow	Cost
(1,2)	17.92	47.96
(1,3)	2.08	4.98
(2,3)	0	0
(2,4)	25.47	33.73
(3,4)	2.08	6.50
(2,5)	12.45	76.91
(4,5)	7.55	26.97
$\sum D_{ij}(f_{ij})$		197.05

b)Path Information

Path	Path Flow	Dual Varaible(DV)	E2E Delay
1	17.92	11.55	81.68
2	0	13.55	54.46
3	2.08	11.55	11.48
4	0	16.74	33.48
5	7.55	14.74	60.7
6	12.45	14.744	76.91

Table 3-3 Simulation results of step 2



a) Link Information

Link	Flow	Cost
(1,2)	17.39	44.28
(1,3)	2.61	8.16
(2,3)	0	0
(2,4)	25	31.72
(3,4)	2.61	10.38
(2,5)	12.39	76
(4,5)	7.61	27.62
$\sum D_{ij}(f_{ij})$		198.16

b)Path Information

Path	Path Flow	Marginal Cost	Lag Multiplier	E2E Delay
1	17.39	10.87	0.385	76
2	0	14.91	0	54.66
3	2.61	15.04	0	18.53
4	0	18.74	0	38
5	7.61	14.7	0	59.35
6	12.39	14.62	0.115	76

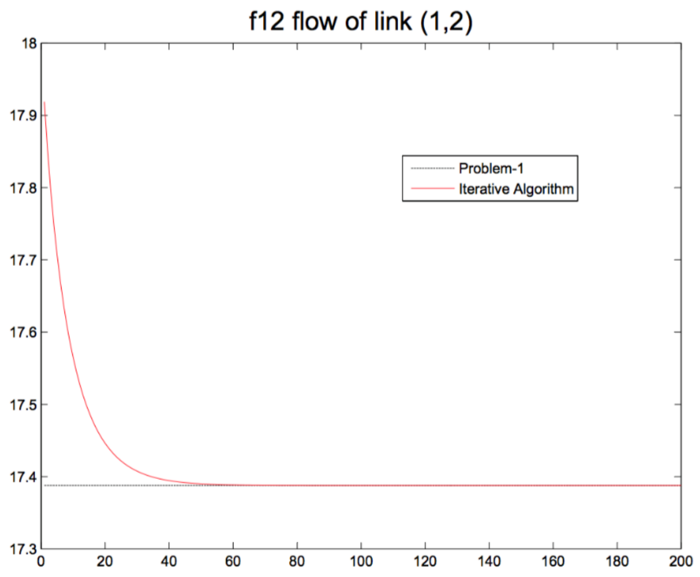
Table 3-4 Final results of Step3 for 1000 iterations and step size 0.008

a) Link Information

Link	Flow	Cost
(1,2)	17.39	44.28
(1,3)	2.61	8.16
(2,3)	0	0
(2,4)	25	31.72
(3,4)	2.61	10.38
(2,5)	12.39	76
(4,5)	7.61	27.62
$\sum D_{ij}(f_{ij})$		198.16

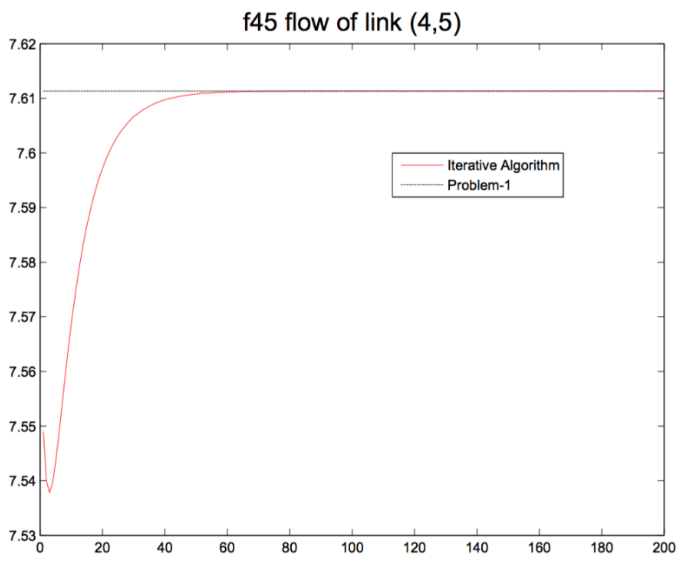
b)Path Information

Path	Path Flow	Marginal Cost	Lag Multiplier	E2E Delay
1	17.39	10.87	0.385	76
2	0	14.91	0	54.66
3	2.61	15.04	0	18.53
4	0	18.74	0	38
5	7.61	14.7	0	59.35
6	12.39	14.62	0.115	76



(a) Link (1,2)

Figure 3-3 The flow of links per 200 iterations



(b) Link (4,5)

Figure 3-4 The flow of links per 200 iterations

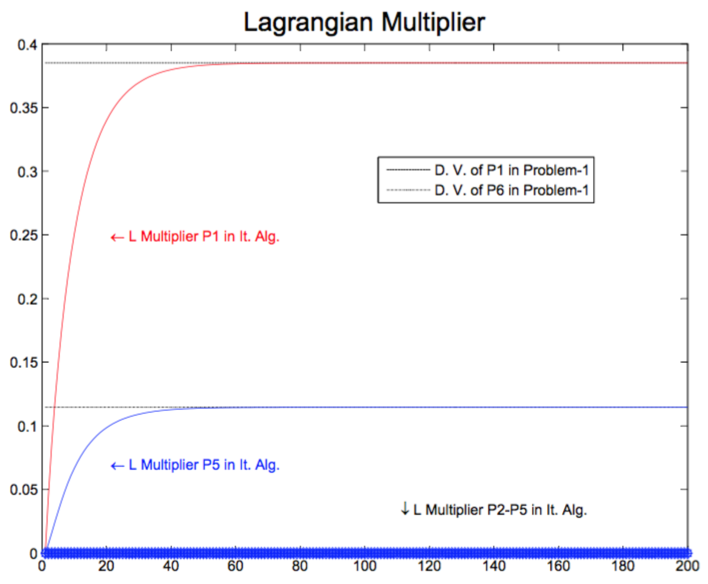


Figure 3-5 The Lagrangian multipliers corresponding to paths per 200 iterations

Assumption 1. The value of r_W's is such that Problem 1 has at least one strictly feasible point, in other words (16) is held.

$$\exists\bar{X}| \sum_{p\in P_w} \bar{x}_p = r_w \text{ and } \bar{x}_p \geq 0 \text{ and } h_p(\bar{X}) < th_p \qquad \forall w \in W, \forall p \in P_w \tag{3-11}$$

Result-1: Since the feasible set of the Problem 1 is not empty, this problem has at least one optimal point. Proposition 1. The optimum solution of Problem 4 is equal to the optimum solution of Problem 2.

Proof. Since Problem 2 is a convex optimization problem, if the Slater conditions apply then the strong duality will also apply. According to Assumption 1 the Slater condition is held; therefore, strong duality is held.

Result-2: Assuming that the input traffic of sessions w meet (3-11), a strong duality exists and the optimum solution of Problem 4 is equal to the optimum solution of Problem 2.

Result-3: Because of strong duality, (3-12) should hold for the optimum points of Problem 2 and Problem 4 as follow:

$$\lambda_p^*.\left(h_p\left(x_p^*\right)\right) = 0 \equiv \begin{cases} h_p\left(x_p^*\right) - th_p < 0 => \lambda_p^* = 0 \\ h_p\left(x_p^*\right) - th_p = 0 => \lambda_p^* \geq 0 \end{cases} \tag{3-12}$$

According to (3-12), at the optimum point of Problem 4, the Lagrange Multiplier of the paths with lower costs than that of the threshold level is 0, and for the paths with Lagrange Multipliers greater than 0, the final traffic amount assigned to them will be such that the cost of these paths will be exactly equal to the threshold level.

Proposition 2. A) The function -q(Λ) defined in Problem 4 is a convex function of Λ.

B) This function has sub-gradient at all of the points in its domain [25] [26].

Proof. If

$$C = \{(x_1 \dots x_2) | \sum_{p \in P_w} x_p = r_w, x_p \geq 0 \ \forall p \in P_w\}$$

Then:

$$-q(\Lambda) = \max_{X \in C}\{-L(X, \Lambda)\}$$

A) q is a convex function: Defining vector A(X) and function b(X) by (3-13-1) and (3-13-2), L(X, Λ) can be considered as a linear function of Λ for a given value of vector X, as in (3-13-3)

$$A(X) = Th - H(x) \qquad (1)$$

$$b(X) = \sum_{p \in P} h_p(x_p) \qquad (2) \qquad\qquad (3\text{-}13)$$

$$-L(X, \Lambda) = ((A(X)^T . \Lambda + \text{b(X)}) \qquad (3)$$

Taking into account the definition given in (20) for function L(X, Λ), q(Λ) can be considered as the point-wise maximum of the family of linear functions at all points ⇤ according to (3-14);

therefore q(Λ) is a convex function.

$$-q(\Lambda)|_{\Lambda^1} = \max_{X\in C}\{(A(X)^T.\Lambda + b(X))|_{\Lambda^1}\} \qquad (3\text{-}14)$$

B) Function q(Λ) has sub-gradient at all points Λ:

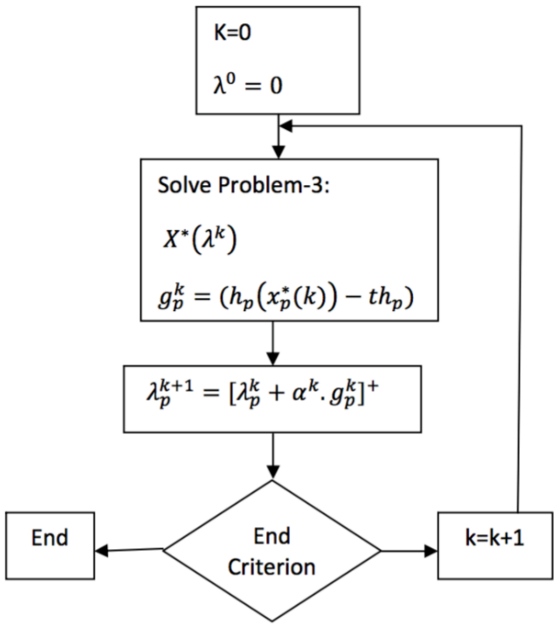
The q(Λ) is differentiable at all points Λ where only one X , X ⇤ (Λ), maximizes (3-14), i.e. at these values of ⇤, only one of the functions $A(X)^T . \Lambda + b(X))$ is greater than the others; therefore, at these points, the sub-gradient of the function is unique and is equal to its gradient which is calculated through (3-15).

$$\frac{\partial - q(\Lambda)}{\partial\Lambda} = \nabla\big(-q(\Lambda)\big) = A\big(X^*(\Lambda)\big) = Th - H\big(X^*(\Lambda)\big)$$
$$X^*(\Lambda) = \arg(\max_{x\in C}\{A(X)^T.\Lambda + b(X)\}) \qquad (3\text{-}15)$$

The q(Λ) is not differentiable at the points Λ where (3-15) is at its maximum at some points. At these Λ some of the functions $(A(X)^T . \Lambda + b(X))$ have the greatest value at the same time. In this case, although q(Λ) is not differentiable, it has sub-gradient which is calculated through (3-16).

$$\frac{\partial - q(\Lambda)}{\partial\Lambda} = Convexhull_{x_i}\{-A\big(X_i^*(\Lambda)\big)^T\}$$
$$X^*(\Lambda) = \arg(\max_{X\in C}\{A(X)^T.\Lambda + \mathrm{b(X)}\}) \qquad (3\text{-}16)$$

According to Proposition 2, the function q(Λ) is the point-wise infimum of a family of affine functions (21); hence, it is concave and sub-differentiable at any point.



K=0
$\lambda^0 = 0$
Solve Problem-3:
$X^*(\lambda^k)$
$g_p^k = (h_p(x_p^*(k)) - th_p)$
$\lambda_p^{k+1} = [\lambda_p^k + \alpha^k . g_p^k]^+$
End
End Criterion
k=k+1

Figure 3-6 q (Λ) for one dimensional

According to Proposition 2, the function q(Λ) is the point-wise infimum of a family of affine functions (21); hence, it is concave and sub-differentiable at any point.

(Figure 3-6). In Proposition 3 an equation is provided to calculate one of the sub-gradient vectors of function - q(Λ) that can be used in the algorithm in figure 3-2.

Proposition 3. At each point Λ (24) gives the sub- gradient of -q(Λ) at that point.

Proof**.** According to (22,23) for a given Λ^b, each optimal solution of (21), $X_i^{\leftarrow}$, $A(X_i^{\leftarrow})$, is one of the sub-gradient vectors of -q(Λ) at point Λ. According to (3-15), the optimum point of this equation at point Λ can be obtained by solving problem 3 based on Λ.

$$-g(\widehat{\Lambda}) = (Th - H(X^*)) \in \frac{\partial_q(\Lambda)}{\partial \Lambda}|_{\widehat{\Lambda}}$$

$$X^*(\Lambda) = \arg(\max_{X \in C}\{A(X)^T.\Lambda + \mathrm{b(X)}\}) \tag{3-17}$$

In other words, $X^{\leftarrow}(\Lambda)$ is an optimal point of Problem 3 based on Λ.

Result-4: Considering (24) the number of components of vector g(Λ) is equal to the total number of paths of session w. The pth component of this vector is equal to the deviation of the cost of path p from its threshold level. In this equation, the path cost should be calculated when the traffic is the optimum solution of Problem 3 for vector Λ . To calculate the sub gradient vector at point Λ, solving Problem 3 at vector Λ and finding its optimum solutions suffices. Following this, the cost of each path is calculated for this traffic and its deviation from the threshold level is considered as the component of the sub-gradient vector.

Proposition 4. The algorithm introduced in Section 3 converges:

Proof. As shown in Figure 3-2, this algorithm describes the steps of the sub-gradient method in solving Problem 4. According to the proof given in [21], if the value of the sub-gradient of function q(Λ) in all points has an upper bound such as G and if the distance from the initial point of the algorithm and the optimum point is less than R, the sub-gradient method converges [21]. To prove the convergence of the algorithm, first, an upper bound for the distance of the initial point of this algorithm and the optimum point is introduced, and then the upper bound for the value of the sub-gradient vector of function q(Λ) at all acceptable points is calculated.

A) Upper bound for the distance between the initial point Λ^0 and optimal point Λ(*).

The initial point of the proposed algorithm in this article is $\Lambda^0 = 0$. Assume a component λ_p is infinite. Considering Assumption-1 the amount of $L(X, \Lambda^{⇤})$ and also $g(\Lambda^{⇤})$ is -inf. The optimal value of Problem 3 will be -inf, while the optimal values of Problem 3 and Problem 1 were expected to be equal. Considering Assumption-1 the optimal value of Problem 1 is finite (a contradiction); therefore, all components of $\Lambda^{⇤}$ are finite, hence $|\Lambda^{⇤} - \Lambda^0|$ is bounded.

B) The norm of the sub-gradient vector in all iterations is upper bounded:

In iteration k, the component p of the sub-gradient vector is equal to the difference of

$h_p(X^{⇤}(\Lambda^k))$ with th_p . Considering Assumption-1, $(X^{⇤}(\Lambda^k))$ is a finite vector and since the optimal value of Problem 1 is finite then $h_p(X^{⇤}(\Lambda^k))$ must be finite, hence, the norm of the vector is finite. Based on the maximum distance between the initial and the optimal points of the algorithm and the upper bound calculated for the sub-gradient at every step of the algorithm, the sub-gradient method for solving this problem will converge.

CHAPTER 4

PARALLELIZING LARGE SCALE GRAPH ALGORITHMS USING THE APACHE SPARK DISTRIBUTED MEMORY SYSTEM

The rapidly emerging area of Social Network Analysis is typically based on graph models. They include directed/undirected graphs, as well as a multitude of random graph representations that reflect the inherent randomness of social networks. A large number of parameters and metrics are derived from these graphs. Overall, this gives rise to two fundamental research/development directions: (1) advancements in models and algorithms, and (2) implementing the algorithms for huge real-life systems. The model and algorithm development part deals with finding the right graph models for various applications, along with algorithms to treat the associated tasks, as well as computing the appropriate parameters and metrics. In this chapter, we would like to focus on the second area: on implementing the algorithms for very large graphs.

It is worth noting that the sheer size of real-life social networks leads to a type of hardness that the theoretical approaches rarely appreciate. This lies in the fact that even very simple algorithmic tasks can become hard if we want to execute them on huge graphs. For example, counting the triangles in a graph is a simple looking, yet important task; it plays a role in often used parameters, such as the clustering coefficient and the transitivity ratio. For smaller graphs, triangle counting can be carried out by a trivial polynomial-time algorithm: we can just exhaustively check all triples of nodes, and count how many of them are triangles. For a very large graph, however, this does not lead to a scalable approach. While various nontrivial improvements to this exhaustive search are known, they still face the scalability problem on huge graphs.

In this chapter, we address the problem of parallelizing three famous network algorithms on a practical distributed memory system. The approach is based on the Spark framework and the GraphX API which is runs on top of the Hadoop distributed file system. We develop three case studies in this framework: (1) computing the PageRank in a social network, (2) finding connected components in the graph representing the network, and (3) triangle counting. A key issue for the large-scale implementation is how to partition the whole task into parallel and independent tasks that run on different machines, such that we can reduce the communication and storage overhead in the distributed cluster. In our case studies four different partition strategies are introduced, they are called Canonical Random Vertex Cut, Edge Partition 1D, Edge Partition 2D, and Random Vertex Cut. All these partition strategies are based on Resilient Distributed Dataset (RDD) which is introduced in Apache Spark. RDD is a new representation of data which is stored in different physical location that can be recovered in a case of failure. RD. As a graph point of view, edges and vertices information can be stored as an RDD. Graph algorithms could be break into independent tasks which processing a corresponding partition of edges or vertices data. Apache Spark tries to keep graph RDD in memory as much as possible to speed up the processing. This new paradigm of graph processing is scalable and fault tolerant.

4.1 Overview of Big Data Processing

Spark is an open source, in-memory big data processing framework in a distributed environment. It started as a research program in 2009 and became an open source project in 2010. In 2014, it was released as an Apache incubator projec.

Spark is evolved from Hadoop MapReduce so it can be run on Hadoop cluster and data in the Hadoop distributed File System (HDFS). It supports a wide range of workloads, such as Machine Learning, Business Intelligence, streaming and batch processing. Spark was created to complement, rather than replace Hadoop. The Spark core is accompanied by a set of powerful, higher-level libraries which can be used in the same application. These libraries currently include SparkSQL, Spark Streaming, MLlib (for machine learning), and GraphX, as shown in Figure 4-1.

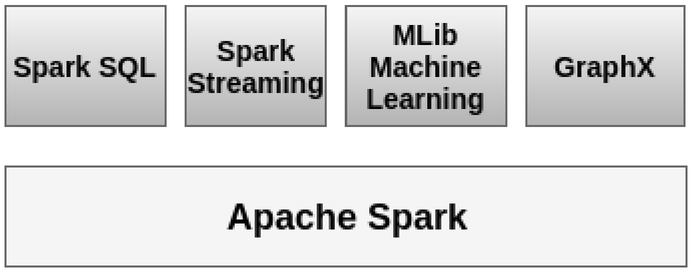


Figure 4-1 Spark full stack

In order to efficiently use the processing resources of a cluster, Spark needs a cluster resource manager. Yet Another Resource Negotiator (YARN) is a Hadoop processing layer that contains a resource manager and a job scheduler. Yarn allows multiple applications to run on a single Hadoop Cluster. Figure 4-2 illustrates how Spark uses Yarn as a distributed resource manager.

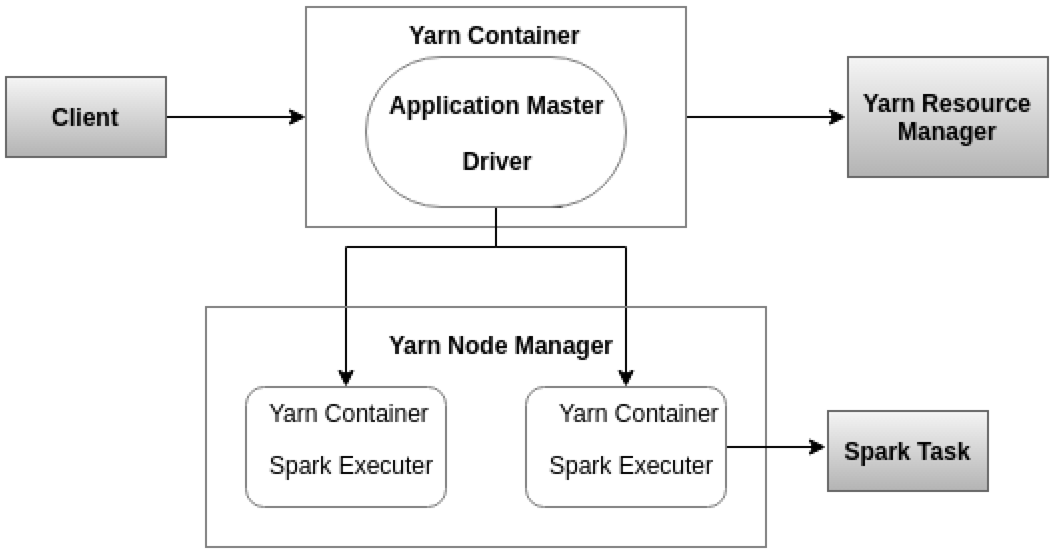


Figure 4-2 Yet another resource manager

Although Spark is designed for in-memory computation, it is capable of handling workloads larger than the cluster aggregate memory. Almost all the Spark built-in functions automatically split to local disks when the working data set does not fit in memory. In the next two section, we outline the difference between Spark and MapReduce, as well as the concept of Resilient Distributed Dataset (RDD) in Spark.

4.2 Apache Spark Vs. Hadoop MapReduce

Apache Spark improvements over Hadoop MapReduce are characterized by efficiency and usability, as shown in Figure 4-3. In order to improve efficiency, it offers in-memory computing capability, which can provide a fast running environment for applications that need to reuse and share data across computations. Having different languages with integrated APIs, such as Java, Scala, Python and R, improve Spark's usability, as compared to MapReduce.

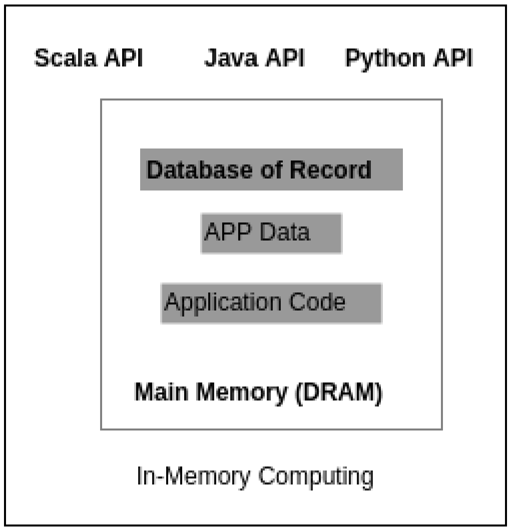
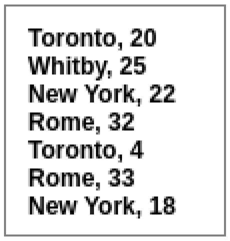


Figure 4-3 Apache Spark efficiency and usability

Next, we explain the Hadoop Mapreduce with an example, and then discuss how Spark can improve the efficiency for implementing more complex algorithms.

MapReduce is a programming model, and an associated implementation, which allows massive scalable data processing across hundreds or thousands of servers. MapReduce refers to two separate and distinct tasks, needed for big data processing. The first one is the map task, which converts a set of data to another set of data called tuples (key/value pairs). The reduce task takes the map output and combines those data tuples into smaller sets of tuples. In the MapReduce processing model the reduce task always runs after the map task. A simple MapReduce example is as follows.

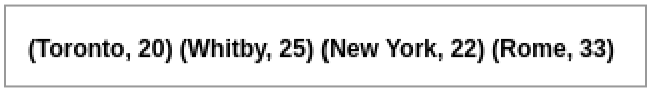
Let say we have five data sets, each of them contains two columns that represent a city and the temperature in that city (Figure 4-4).



Toronto, 20
Whitby, 25
New York, 22
Rome, 32
Toronto, 4
Rome, 33
New York, 18

Figure 4-4 Cities information

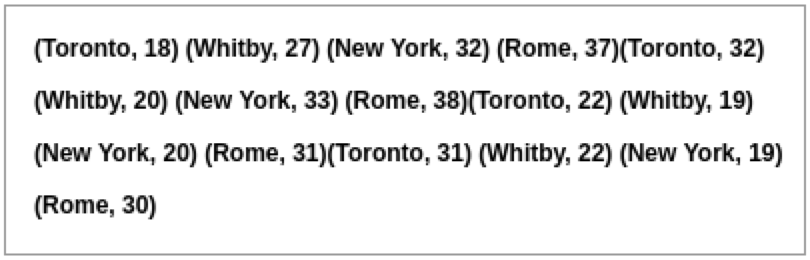
The goal is to find the maximum temperature for each city. In MapReduce, we split the task into five map tasks. Results from one mapper task are illustrated in Figure 4-5.



(Toronto, 20) (Whitby, 25) (New York, 22) (Rome, 33)

Figure 4-5 Maximum temperature for each city

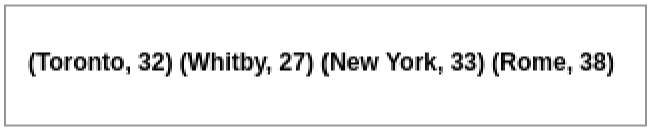
We can assume other mapper tasks create the intermediate results which are shown in Figure 4-6.



(Toronto, 18) (Whitby, 27) (New York, 32) (Rome, 37)(Toronto, 32)
(Whitby, 20) (New York, 33) (Rome, 38)(Toronto, 22) (Whitby, 19)
(New York, 20) (Rome, 31)(Toronto, 31) (Whitby, 22) (New York, 19)
(Rome, 30)

Figure 4-6 Maximum temperature for each city

Reduce tasks will combine input and output results for each city (Figure 4-7).



```
(Toronto, 32) (Whitby, 27) (New York, 33) (Rome, 38)
```

Figure 4-7 Output of the reduce task

Some applications, such as implementing large scale graph algorithms in a social network, are more complex than just one path of Map and Reduce. They require multiple operations over a same data sets. In MapReduce no sharing data across time stamps or iteration is available . Let's take a look at the PageRank example in MapReduce which requires multiple iteration, see Figure 4-8.

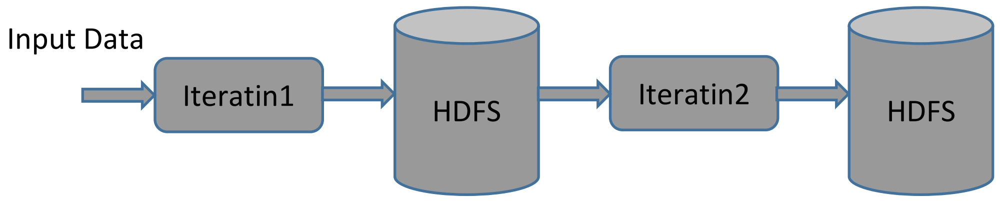


Figure 4-8 A Multi-Iteration algorithm in MapReduce

The algorithm starts with data in HDFS and then does one step on MapReduce (iteration 1). Then to share the data with the next step it has to write it back into HDFS again. After that, in the next iterations of PageRank, the data must be loaded back, and the algorithm is continued.

Spark has a computation model in which after each iteration the data will be stored in memory, and it is available to be processed in the next steps as illustrated in Figure 4-9.

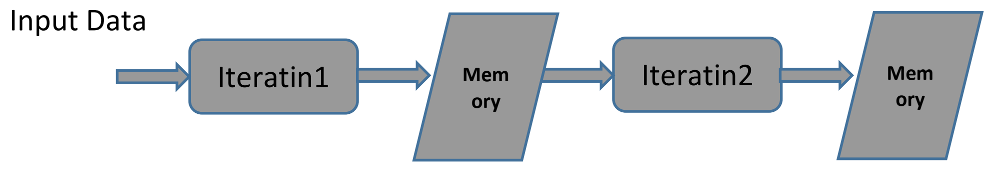


Figure 4-9 Multi-Iteration Algorithm in Spark

In Spark, instead of thinking in terms of map and reduce functions, we think in terms of distributed data sets. This is what essentially distinguishes Spark from Hadoop Mapreduce. The data abstraction in Spark is called Resilient Distributed Dataset (RDD), consisting of parallel collections of Scala objects. In the next section, RDD is explained in more details.

4.3 Resilient Distributed Dataset (RDD)

RDD is a logical reference of a dataset which is paralleled among many processors in the cluster. RDD is resilient, meaning that if data in memory is lost, then it can be recovered. It is distributed, which means processing across the cluster, and the dataset can come from a file, or be created by a program. Basically, RDD is a fundamental unit of data in Spark, forming an immutable dataset. It contains two different operations, called Transformation and Action. Transformation creates a new RDD based on an existing one, while Action returns a value from an RDD. Figure 4-10 illustrates Transformation and Action for an existing RDD.

RDD Transformation includes parts called filter, map, union, and others. Actions includes reduce, collect, count, etc. The main advantage of RDDs is that they are simple and well understood, because they deal with concrete classes, providing a familiar object-oriented programming style with compile-time type-safety.

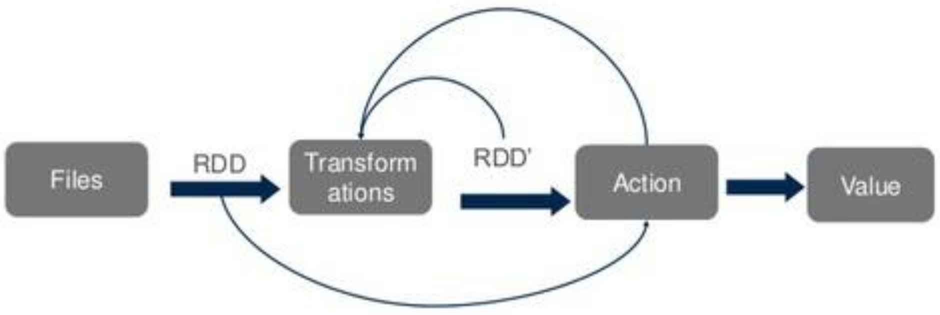
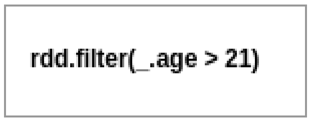


Figure 4-10 RDD transformation and action

For example, given an RDD containing instances of Person we can filter by age by referencing the age attribute of each Person object as illustrated in Figure 4-11.



```
rdd.filter(_.age > 21)
```

Figure 4-11 Filter by age in an RDD

The transformations are only computed when an action requires a result to be returned. In this example, when an action like count is called, we will be returned the Persons objects belonging to persons older than 21.

Pair RDD is a special form of RDD, in which each element must have a key-value pair (two element tuple). Pair RDD is important because of the traditional map-reduce algorithms for parallel processing which is based on key and value pairs. Figure 4-12 shows the word count example which is implemented using pair RDD in Spark. First step is creating an RDD based on input data then split the RDD based on the words space. After having an RDD corresponding to each word, a pair RDD can be created with the word as the key and number one as the value. Then reducing the pair RDDs based on the key returns the word count.

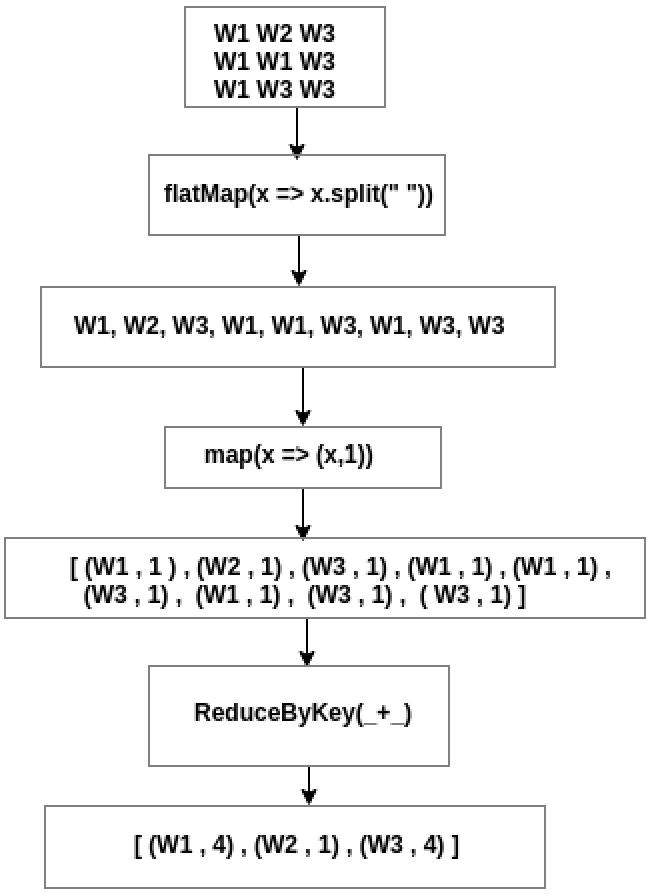

W1 W2 W3
W1 W1 W3
W1 W3 W3
flatMap(x => x.split(" "))
W1, W2, W3, W1, W1, W3, W1, W3, W3
map(x => (x,1))
[(W1 , 1) , (W2 , 1) , (W3 , 1) , (W1 , 1) , (W1 , 1) , (W3 , 1) , (W1 , 1) , (W3 , 1) , (W3 , 1)]
ReduceByKey(_+_)
[(W1 , 4) , (W2 , 1) , (W3 , 4)]

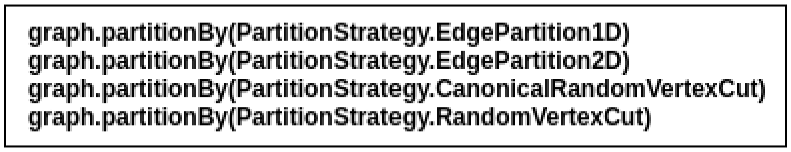
Figure 4-12 Word Count example, using Spark

4.4 GraphX API for Spark

Graph-based algorithms are becoming very important for solving numerous problems in data-intensive applications, including search engines, recommendation systems, financial analysis, and many others. As these problems grow in scale, computational and memory requirements of the processing algorithms rapidly become a bottleneck. To avoid such a bottleneck, parallel computing resources are required. Graphx is a new component in Apache Spark for graph parallel processing, which extends the Spark RDD by introducing the concept of Property Graph. The Property Graph is a directed multigraph, that is, a directed graph with potentially multiple parallel edges sharing the same end vertices. There also properties attached to each vertex and edge. GraphX is a native property graph processor. It allows all vertices and edges to have their own properties.

4.5 Graph Partitioning in Spark

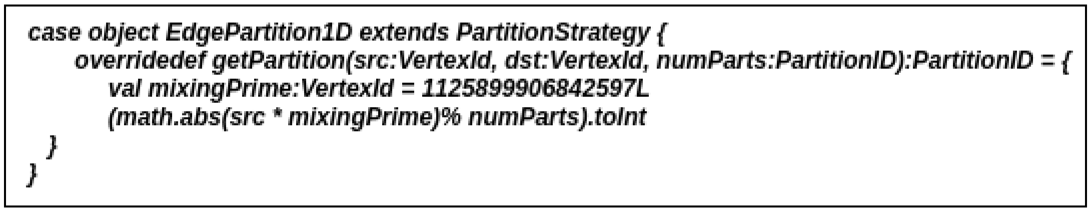
Graph partitioning algorithms are designed to minimize communication and balance the computation among multiple processors. Partitioning the graph data and balancing the computation on a distributed cluster of machines is a common approach to scale-out computations for large scale input graph data. Iterative computations on input graph data, for instance the PageRank systems, are well known use cases for graph partitioning. The quality of graph partitioning depends on balancing the processing load across machines and minimizing the communication cost inside the cluster. There are two main approaches for partitioning a graph among different machines. They are called vertex cut and edge cut. Graphx implements the vertex cut approach to ensure one edge is assigned to one partition. In this case one vertex can be shared across partitions. This strategy moves the network communication from edges to vertices. In order to ensure vertices are partitioned in a most efficient way for a particular algorithm, Graphx provides a number of strategies, which are illustrated in Figure 4-13.



```
graph.partitionBy(PartitionStrategy.EdgePartition1D)
graph.partitionBy(PartitionStrategy.EdgePartition2D)
graph.partitionBy(PartitionStrategy.CanonicalRandomVertexCut)
graph.partitionBy(PartitionStrategy.RandomVertexCut)
```

Figure 4-13 Graph partitioning strategy

The choice between the partition strategies is based upon the algorithm and the graph structure. The strategy called EdgePartition1D ensures that all edges with the same source are partitioned together, so the edges that belong to a particular partition have the same source. For applications, such as counting the outgoing edges which the operation aggregated to the source, each partition has all the data needed on an individual machine. In this case the network traffic among different machines is minimized. On the other hand, for graphs with power-law structure, a few partitions may receive a significant proportion of the total number of edges. Figure 4-14 shows the EdgePartition1D source code in Scala.

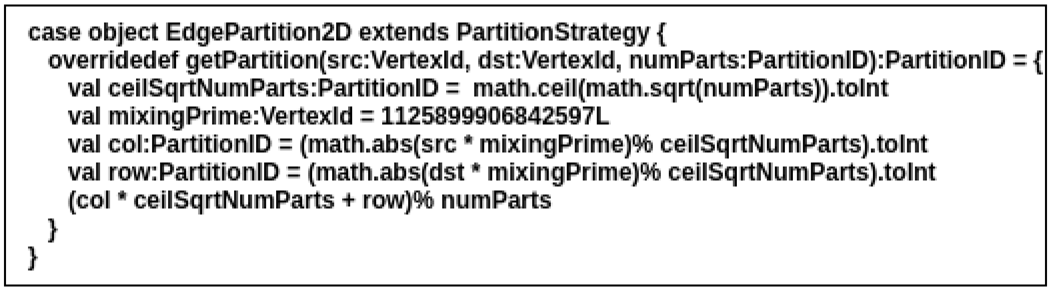


```
case object EdgePartition1D extends PartitionStrategy {
    overridedef getPartition(src:VertexId, dst:VertexId, numParts:PartitionID):PartitionID = {
        val mixingPrime:VertexId = 1125899906842597L
        (math.abs(src * mixingPrime)% numParts).toInt
  }
}
```

Figure 4-14 EdgePartition1D implementation

A large number called mixingPrime is used to balance the partitions. EdgePartition2D uses both source vertices and destination vertices to calculate partitions. Figure 4-15 shows the source code for EdgePartition2D.

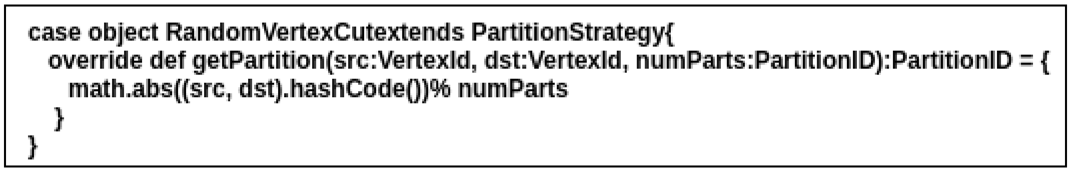
The RandomVertex Cut strategy splits the graph based on both source and destination vertices, which can help to create a more balanced partition. This strategy may affect the runtime performance due to the increase in the amount of network communication



```
case object EdgePartition2D extends PartitionStrategy {
  overridedef getPartition(src:VertexId, dst:VertexId, numParts:PartitionID):PartitionID = {
    val ceilSqrtNumParts:PartitionID =  math.ceil(math.sqrt(numParts)).toInt
    val mixingPrime:VertexId = 1125899906842597L
    val col:PartitionID = (math.abs(src * mixingPrime)% ceilSqrtNumParts).toInt
    val row:PartitionID = (math.abs(dst * mixingPrime)% ceilSqrtNumParts).toInt
    (col * ceilSqrtNumParts + row)% numParts
  }
}
```

Figure 4-15 EdgePartition2D implementation

The RandomVertex Cut strategy splits the graph based on both source and destination vertices, which can help to create a more balanced partition. This strategy may affect the runtime performance due to the increase in the amount of network communication. In this case even the edge that connects the same pair of nodes may be spread among two machines based on the direction of the edge. Figure 4-16 illustrates the source code of Random Vertex Cut.

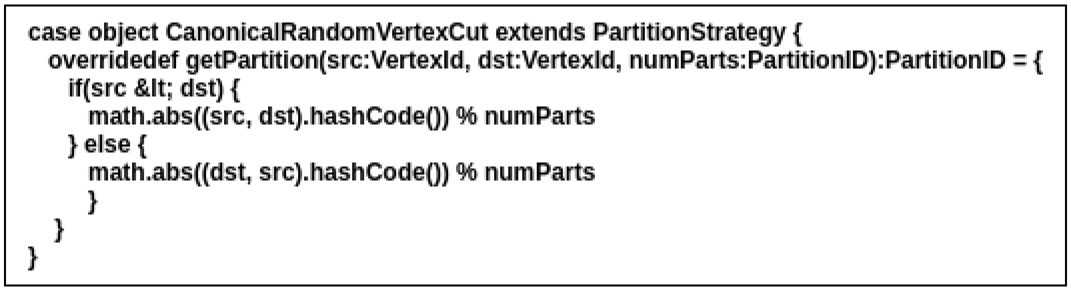


```
case object RandomVertexCutextends PartitionStrategy{
  override def getPartition(src:VertexId, dst:VertexId, numParts:PartitionID):PartitionID = {
    math.abs((src, dst).hashCode())% numParts
   }
}
```

Figure 4-16 Random Vertex Cut implementation

The strategy Canonical Random Vertex Cut partitions the edges regardless to the direction, so the edges sharing both a source and a destination will be partitioned together. Figure 4-17 shows the source code for this partitioning strategy.

In the next section we show an example, which is based on a property graph in a paper citation network (see http://snap.stanford.edu/data/cit-HepTh.html). The network is created from the publication information available for ArXiv High Energy Physics Theory category.



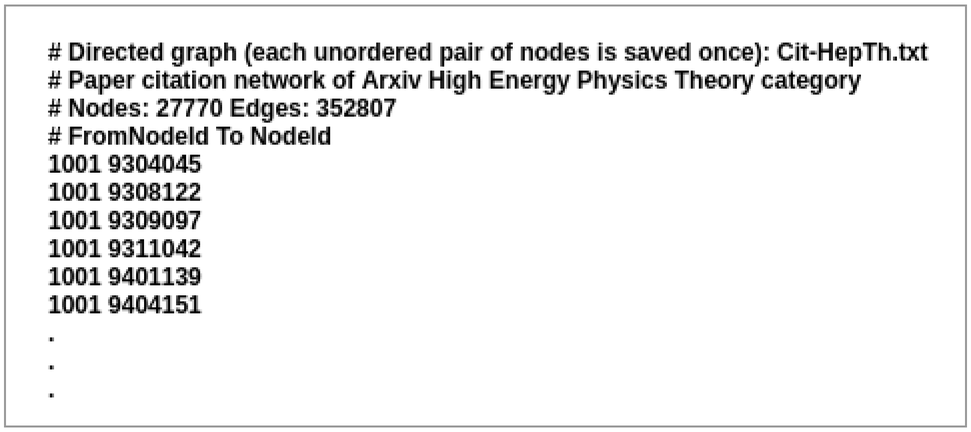
```
case object CanonicalRandomVertexCut extends PartitionStrategy {
  overridedef getPartition(src:VertexId, dst:VertexId, numParts:PartitionID):PartitionID = {
    if(src &lt; dst) {
      math.abs((src, dst).hashCode()) % numParts
    } else {
      math.abs((dst, src).hashCode()) % numParts
      }
  }
}
```

Figure 4-17 Canonical Random Vertex Cut implementation

In the next section we show an example, which is based on a property graph in a paper citation network (see http://snap.stanford.edu/data/cit-HepTh.html). The network is created from the publication information available for ArXiv High Energy Physics Theory category.

4.6 Paper citation Network Example

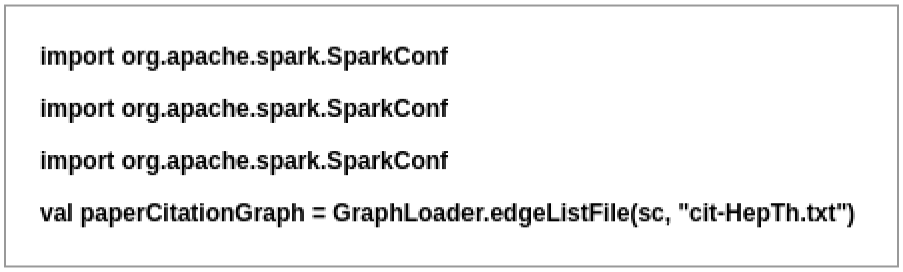
The first few lines in the text file containing all the citation information is shown in Figure 4-18. After the comment lines, which begin with #, each line represents one edge of the graph. For example, the first edge starts from a vertex identified by 1001 to another one identified by 9304045. Each vertex is keyed by a unique 64-bit long identifier (VertexID).



```
# Directed graph (each unordered pair of nodes is saved once): Cit-HepTh.txt
# Paper citation network of Arxiv High Energy Physics Theory category
# Nodes: 27770 Edges: 352807
# FromNodeId To NodeId
1001 9304045
1001 9308122
1001 9309097
1001 9311042
1001 9401139
1001 9404151
.
.
.
```

Figure 4-18 Text file containing the citation information

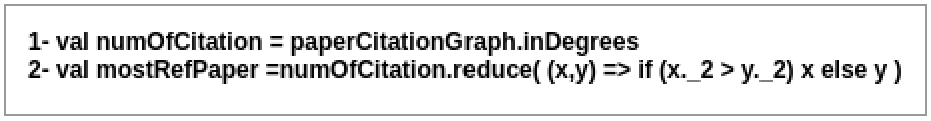
Similarly, edges have corresponding source and destination vertex identifiers. In the context of a paper citation network, the second paper in the older one being cited by the newer paper. This format of a text file is recognized by GraphX. The next step is the creation of the RDD, based on the edges and vertices data. For this purpose, SparkContext should be constructed. Figure 4-19 shows the source code for loading the graph data, and creates an immutable value called paperCitationGraph.



```
import org.apache.spark.SparkConf
import org.apache.spark.SparkConf
import org.apache.spark.SparkConf
val paperCitationGraph = GraphLoader.edgeListFile(sc, "cit-HepTh.txt")
```

Figure 4-19 Loading a graph in Spark

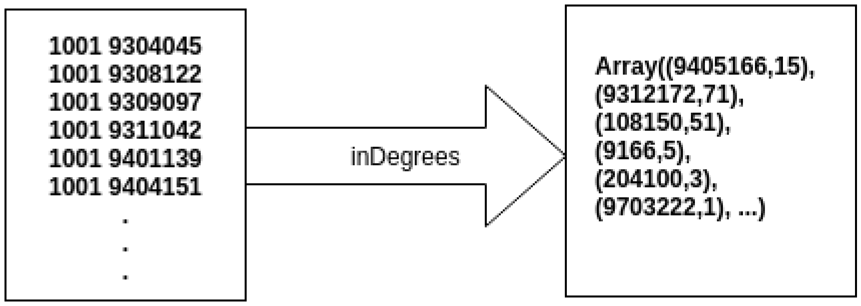
GraphLoader is an object in GraphX library which contains a method called edgeListFile. This can load a graph from a text file in edge-list format and it uses two methods. The first one is SparkContext (sc) and the second one is the file that contains the graph property information. Now the graph is ready to be processed in a distributed environment. Finding the most-referenced paper is a well-known problem in a paper citation network. The following steps, as illustrated in Figure 4-20, can find such a paper in a distributed way using Spark/Graphx library by calling inDegree and reduce methods.



```
1- val numOfCitation = paperCitationGraph.inDegrees
2- val mostRefPaper =numOfCitation.reduce( (x,y) => if (x._2 > y._2) x else y )
```

Figure 4-20 inDegree and reduce in Graphx API

In the first step, an RDD of vertexID and in-degree pairs proceeds from the inDegree method. Figure 4-21 shows the details.



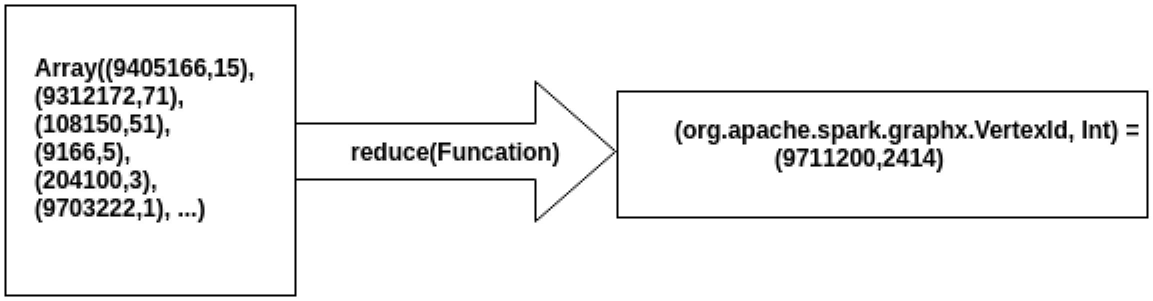
```
1001 9304045
1001 9308122
1001 9309097
1001 9311042
1001 9401139
1001 9404151
.
.
.
```

```
Array((9405166,15),
(9312172,71),
(108150,51),
(9166,5),
(204100,3),
(9703222,1), ...)
```

Figure 4-21 inDegrees method in GraphX producing key-value pairs

In the second step, each RDD uses a reduce method which takes a function as an input. The function receives two elements from the RDD and returns a single value. The function would be called repeatedly on pairs of elements (RDD) from the reduce method, until only a single value is left. The single value is returned from the reduce method in step 2. Figure 4-22 illustrates the second step.

The PageRank algorithm can be used to measure the influence of vertices in any social network, although originally it was developed to support Google search. We are using the same data set as the previous section. First let us see how vertices data look like, figure 4-23.



```
Array((9405166,15),
(9312172,71),
(108150,51),
(9166,5),
(204100,3),
(9703222,1), ...)
```

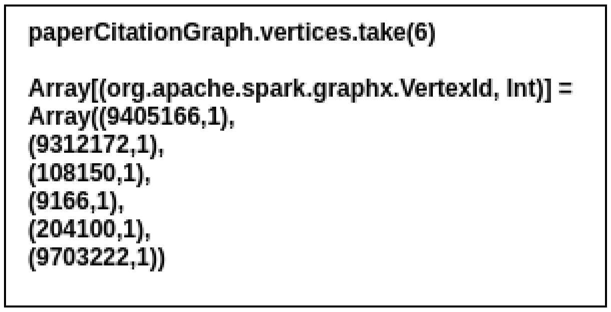
```
(org.apache.spark.graphx.VertexId, Int) =
        (9711200,2414)
```

Figure 4-22 reduce (Function) method in GraphX producing a single key value pair

The result is: Paper ID 9711200 was cited by 2414 other papers, making it the most cited paper.

4.7 Finding PageRank in Social Network using Apache Spark

The PageRank algorithm can be used to measure the influence of vertices in any social network, although originally it was developed to support Google search. We are using the same data set as the previous section. First let us see how vertices data look like, figure 4-23.



```
paperCitationGraph.vertices.take(6)

Array[(org.apache.spark.graphx.VertexId, Int)] =
Array((9405166,1),
(9312172,1),
(108150,1),
(9166,1),
(204100,1),
(9703222,1))
```

Figure 4-23 vertices in the paperCitationGraph

As illustrated in Figure 4-23, all vertices are key-value pairs, where key is the vertex ID and the value is 1. In GraphX all vertices and edges have their own properties, and the arising graph is called property graph. In the considered case the value 1 is the property for all vertices, which is attached to them by the GraphLoader.edgeListFile() method. Now in order to calculate the rank of each vertex, we have to change each vertex property to match the corresponding PageRank. The idea of immutable data set implies that the graph property does not change. So, a new property graph must be created to express the PageRank property. This is a key Spark concept, the existing RDDs (in this case graph structure) are not updated. Instead, a transformation takes place on an existing RDD to create a new RDD. Figure 4-24 illustrates two graph properties before and after applying the PageRank method (transformation).

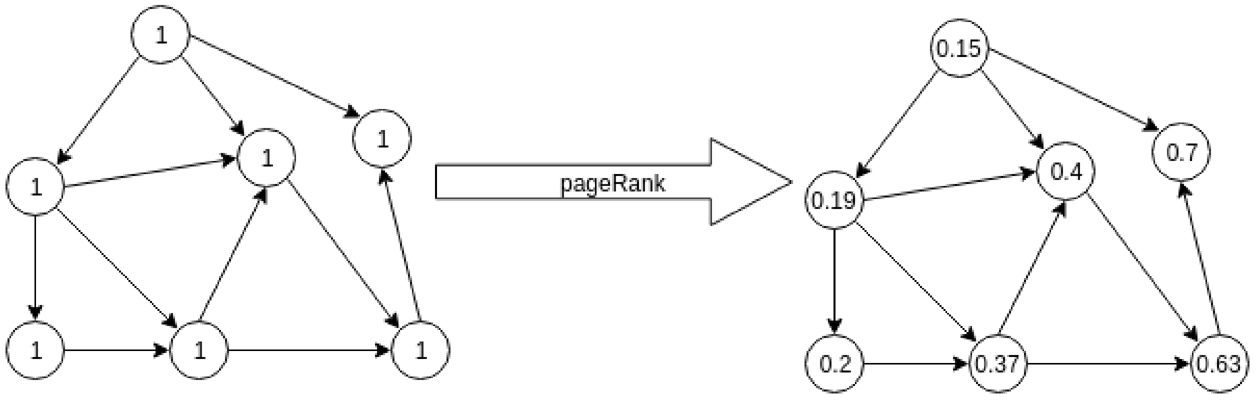
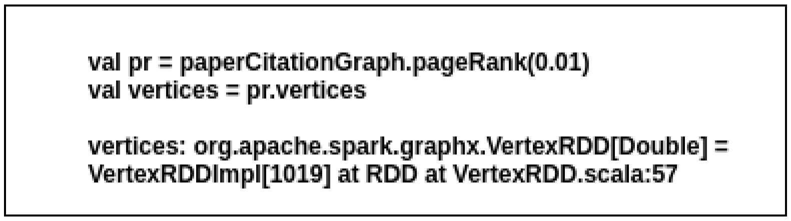


Figure 4-24 PageRank transformation that creates a new property graph

PageRank is a link analysis algorithm that outputs a probability distribution, which can be used to represent the likelihood of a page being referenced. So the new graph has a property of type Double. The code for PageRank calculation in the paper citation network is shown in Figure 4-25.

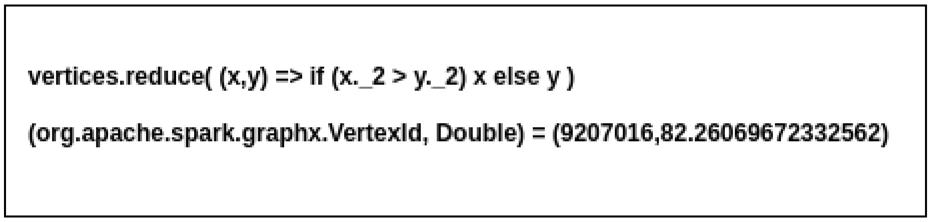


```
val pr = paperCitationGraph.pageRank(0.01)
val vertices = pr.vertices

vertices: org.apache.spark.graphx.VertexRDD[Double] =
VertexRDDImpl[1019] at RDD at VertexRDD.scala:57
```

Figure 4-25 PageRank method in Spark using GraphX library

The next step is to run the reduce method with an appropriate function to find the vertex with the highest PageRank, see Figure 4-26.



```
vertices.reduce( (x,y) => if (x._2 > y._2) x else y )

(org.apache.spark.graphx.VertexId, Double) = (9207016,82.26069672332562)
```

Figure 4-26 The most influential paper

Finally, we get the result that, according to the PageRank algorithm, paper ID 9207016 is the most influential one.

4.8 Finding connected components using Apache Spark

A connected component of a graph is a set of vertices such that every vertex is reachable from every other vertex. Connected component can identify isolated members in social networks, and can also approximate clusters. Figure 4-27 shows an example of connected components in a graph.

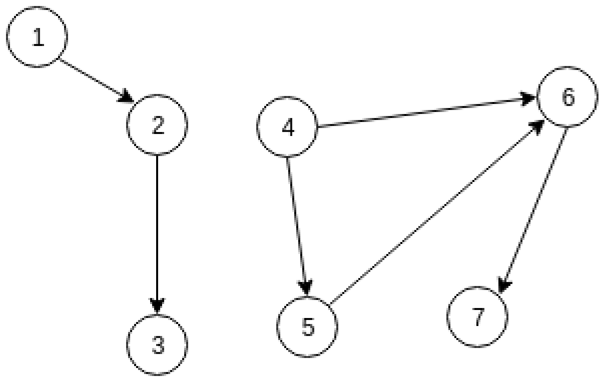
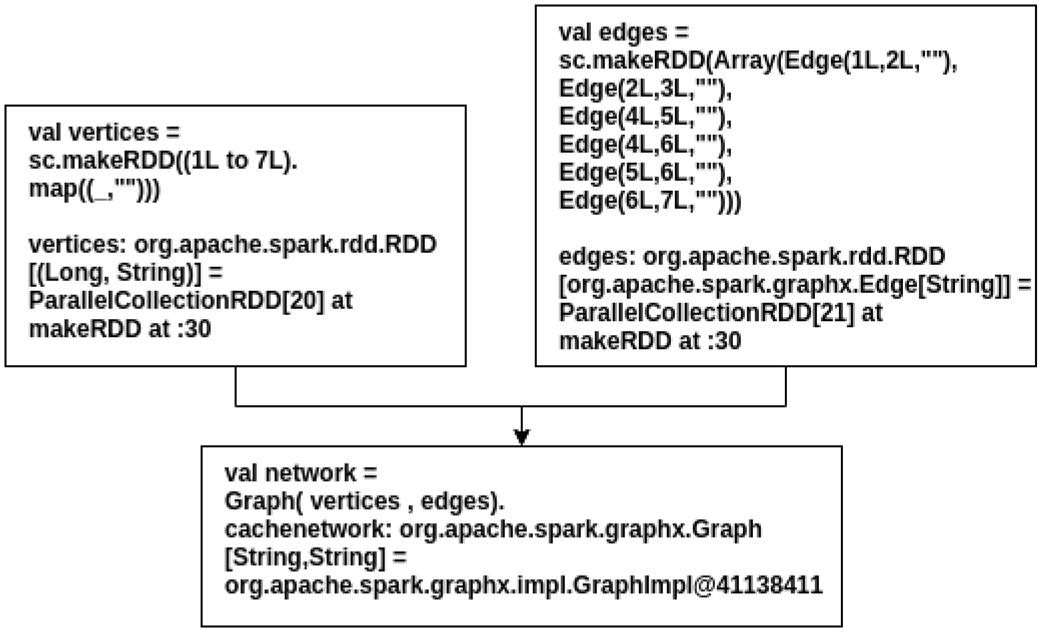


Figure 4-27 Finding connected components

Figure 4-28 illustrates the network construction, based on RDD collections (vertices and edges).



```
val vertices =
sc.makeRDD((1L to 7L).
map((_,"")))

vertices: org.apache.spark.rdd.RDD
[(Long, String)] =
ParallelCollectionRDD[20] at
makeRDD at :30
```

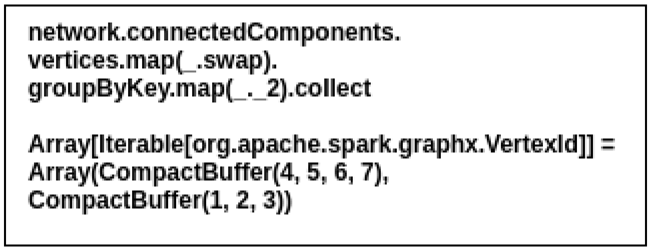
```
val edges =
sc.makeRDD(Array(Edge(1L,2L,""),
Edge(2L,3L,""),
Edge(4L,5L,""),
Edge(4L,6L,""),
Edge(5L,6L,""),
Edge(6L,7L,"")))

edges: org.apache.spark.rdd.RDD
[org.apache.spark.graphx.Edge[String]] =
ParallelCollectionRDD[21] at
makeRDD at :30
```

```
val network =
Graph( vertices , edges).
cachenetwork: org.apache.spark.graphx.Graph
[String,String] =
org.apache.spark.graphx.impl.GraphImpl@41138411
```

Figure 4-28 Network construction based on RDD collections

Using the code provided in Figure 4-29 allows the detection of connected components in the network above.



```
network.connectedComponents.
vertices.map(_.swap).
groupByKey.map(_._2).collect

Array[Iterable[org.apache.spark.graphx.VertexId]] =
Array(CompactBuffer(4, 5, 6, 7),
CompactBuffer(1, 2, 3))
```

Figure 4-29 Finding connected components

4.9 Triangle counting using Apache Spark

Counting the number of triangles in a large graph is frequently used in complex network analysis such as spam detection and uncovering hidden structures in link recommendation. A triangle consists of three vertices that all connected with edges. A social network which contains more triangles usually has tighter connections. The TriangleCount method in spark counts triangles passing through each vertex using the following steps.

Step 1: Find the set of neighbors for each vertex.

Step 2: For each edge find the intersection of the sets and send the count to both vertices

Step 3: Find the sum at each vertex and divide by two since each triangle is counted twice.

Figure 4-30 below illustrates the process.

In order to use the TriangleCount method in Spark, the graph has to meet two requirements. First, the graph has to be partitioned by one of the partition strategy options, described in Section 3-1. Second, if there are any duplicate edges, they have to point in the same direction. To ensure the latter requirement, all edges must be in canonical order, pointing from the lower-numbered vertex ID to the higher numbered vertex ID.

Let us consider the Slashdot social network to find the number of triangles in a large scale network. Slashdot is a technology news related website which has a specific user community. The website features user-submitted and editor-evaluated current, primarily consisting of technology related news. In 2002 Slashdot introduced the Slashdot Zoo feature which allows users to tag each other as friends or foes. The network contains friend/foe links between the users. The network was recorded in November 2008. Table 4-1 shows the data set statistics.

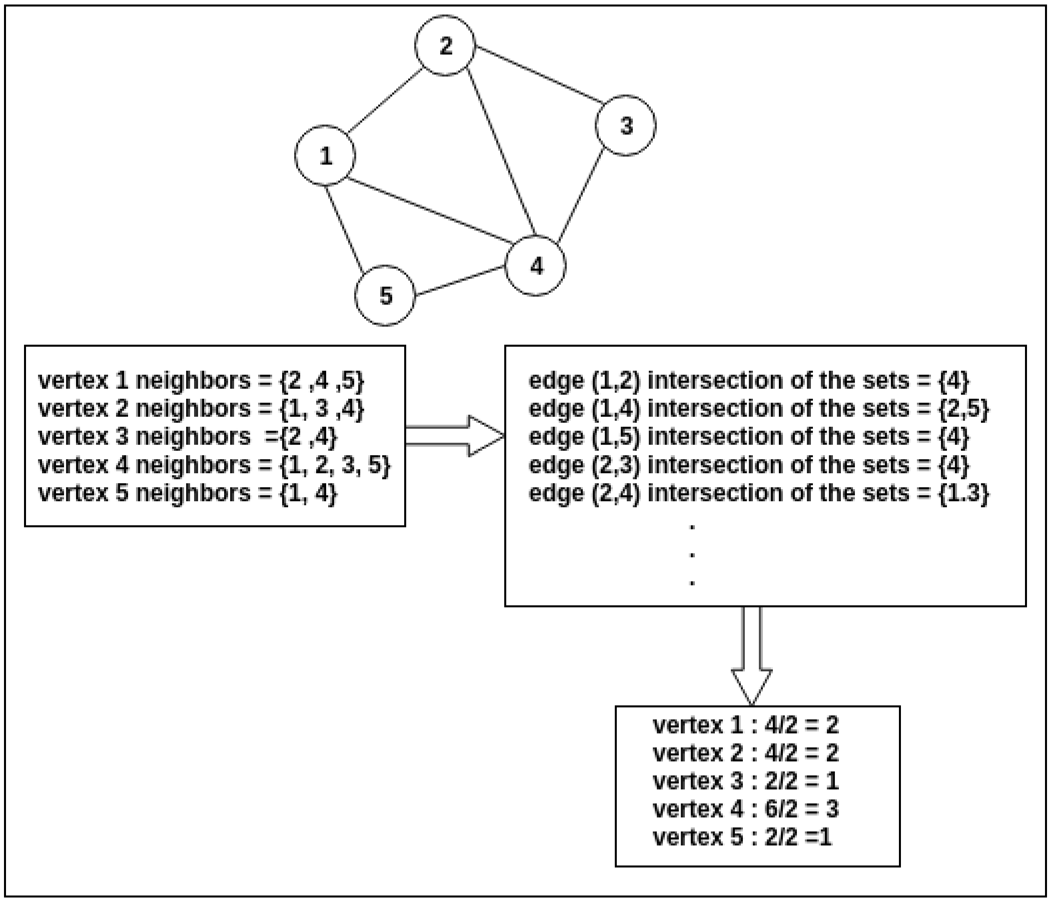

2
3
1
4
5
vertex 1 neighbors = {2 ,4 ,5}
vertex 2 neighbors = {1, 3 ,4}
vertex 3 neighbors =={2 ,4}
vertex 4 neighbors = {1, 2, 3, 5}
vertex 5 neighbors = {1, 4}
edge (1,2) intersection of the sets = {4}
edge (1,4) intersection of the sets = {2,5}
edge (1,5) intersection of the sets = {4}
edge (2,3) intersection of the sets = {4}
edge (2,4) intersection of the sets = {1.3}
vertex 1 : 4/2 = 2
vertex 2 : 4/2 = 2
vertex 3 : 2/2 = 1
vertex 4 : 6/2 = 3
vertex 5 : 2/2 =1

Figure 4-30 Triangle counting algorithm

Table 4-1 Slashdot social network statistics

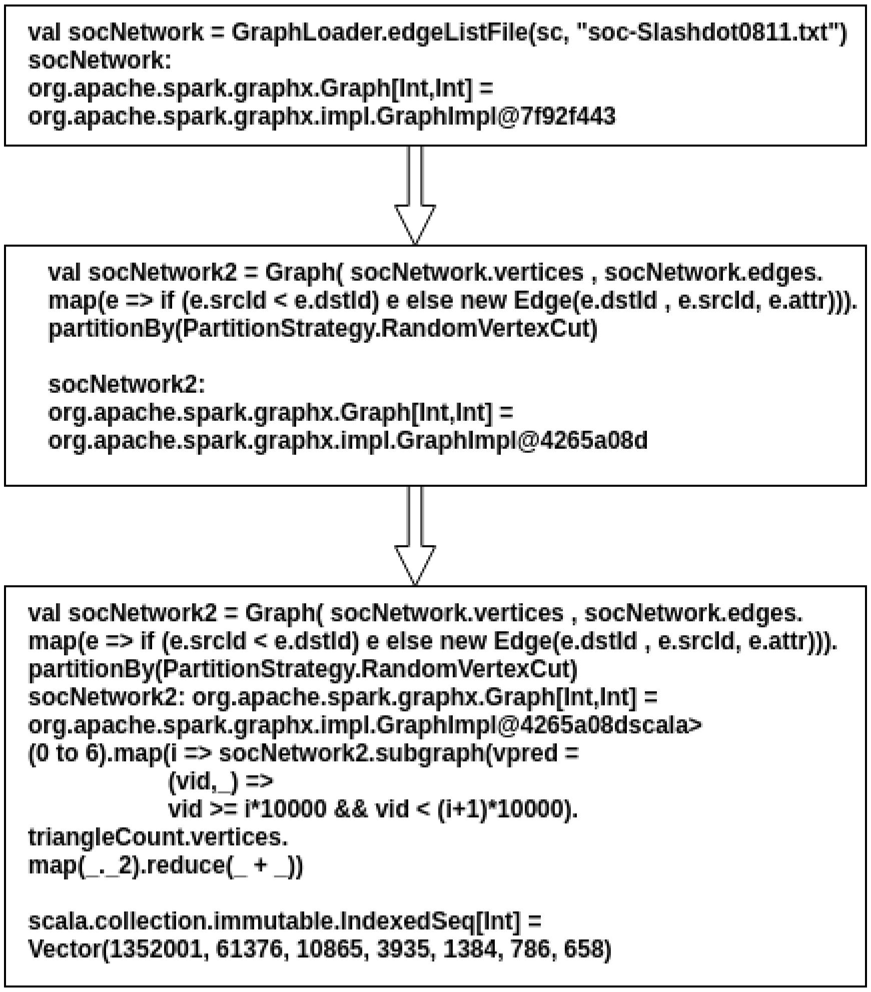
Number of Nodes	77360
Number of Edges	905468
Nodes in largest WCC	77360 (1.000)
Edges in largest WCC	905468 (1.000)
Nodes in largest SCC	70355 (0.909)
Edges in largest SCC	888662 (0.981)
Average clustering coefficient	0.0555
Number of triangles	551724
Fraction of closed triangles	0.008184
Diameter (longest shortest path)	10
90-percentile effective diameter	4.7

Figure 4-31 shows the steps including the source code for counting number of triangles in the Slashdot social network.

As shown in the figure above, the number of triangles are 1352001, 61376, 10865, 3935, 1384, 786 and 658 for each of seven subgraphs ((0 to 6).map), respectively, in the social network.

4.10 Summery of distributed graph processing

In this chapter we introduced Apache Spark as a replacement for Hadoop Mapreduce. Efficiency of Spark, as a result of in-memory processing, makes it a popular big data processing engine. It also has high usability, due to different programming language APIs.



```
val socNetwork = GraphLoader.edgeListFile(sc, "soc-Slashdot0811.txt")
socNetwork:
org.apache.spark.graphx.Graph[Int,Int] =
org.apache.spark.graphx.impl.GraphImpl@7f92f443
```

```
val socNetwork2 = Graph( socNetwork.vertices , socNetwork.edges.
map(e => if (e.srcId < e.dstId) e else new Edge(e.dstId , e.srcId, e.attr))).
partitionBy(PartitionStrategy.RandomVertexCut)

socNetwork2:
org.apache.spark.graphx.Graph[Int,Int] =
org.apache.spark.graphx.impl.GraphImpl@4265a08d
```

```
val socNetwork2 = Graph( socNetwork.vertices , socNetwork.edges.
map(e => if (e.srcId < e.dstId) e else new Edge(e.dstId , e.srcId, e.attr))).
partitionBy(PartitionStrategy.RandomVertexCut)
socNetwork2: org.apache.spark.graphx.Graph[Int,Int] =
org.apache.spark.graphx.impl.GraphImpl@4265a08dscala>
(0 to 6).map(i => socNetwork2.subgraph(vpred =
                (vid,_) =>
                vid >= i*10000 && vid < (i+1)*10000).
triangleCount.vertices.
map(_._2).reduce(_ + _))

scala.collection.immutable.IndexedSeq[Int] =
Vector(1352001, 61376, 10865, 3935, 1384, 786, 658)
```

Figure 4-31 . Triangle Counting in Slashdot social network

There is a large collection of algorithms that cannot be implemented using only one iteration of Map and Reduce functions. Notably, graph processing algorithms fall in this category. Apache

Spark improves efficiency of implementing such algorithms using in-memory processing. Essentially, after one iteration of Map and Reduce, the results are ready, and available in memory for the next iteration. Spark can be 100 times faster than Hadoop Mapreduce for machine learning algorithms, such logistic regression [6]. A fundamental processing unit in Spark is RDD, instead of one path of Map and Reduce functions. RDD is an immutable distributed data set across the cluster which is resilient to data storage failure. Data processing algorithms can be implemented using transformations and actions on each RDD. Transformations will create series of RDDs. As a result of immutability, each one of them can be recalculated from the previous one.

Graphx is a new component in Spark for implementing graph algorithms in a distributed environment. Graphx extends the Spark RDD by introducing a new graph abstraction in terms of a distributed dataset, attached to vertices and edges. In this chapter three important social network algorithms have been introduced using Graphx library in Apache Spark. The first one is finding PageRank in a social network, the second one is finding connected components, and the last but not least is the triangle counting algorithm. In all the three applications, we have illustrated the steps via appropriate examples [27].

CHAPTER 5

APPROXIMATION APPROACH FOR ANALYSING NETWORK STRUCTURE

5.1 How Complex Networks Inspire New Avenues to Approximation – The Case of Unsplittable Flows

The Disjoint Connecting Paths problem, and its capacitated generalization, called Unsplittable Flow problem, play an important role in practical applications, such as communication network design and routing. These problems are hard in general, but various polynomial-time approximations are known. Nevertheless, the approximations tend to be rather complicated, often rendering them impractical in large, complex networks. Therefore, our goal is to present a solution that provides a simple, efficient algorithm for the unsplittable flow problem in large directed graphs. The simplicity is achieved by sacrificing a small part of the solution space. This also represents a novel paradigm of approximation: rather than giving up finding an exact solution, we restrict the solution space to a subset that is the most important for applications, and exclude those that are marginal in some sense. Specifically, the sacrificed part (i.e., the marginal instances) only contains scenarios where some edges are very close to saturation. Therefore, the excluded part is not significant, since the almost saturated solutions are typically undesired in practical applications, such as network design [28].

5.2 Problem statement.

The Disjoint Connecting Paths problem is the following decision task. Input: a set of node pairs $(s_1, t_1), \dots, (s_k, t_k)$ in a graph. Task: Find edge disjoint paths $P_1, \dots, P_k$, such that P_i connects

s_i with t_i for each i.

This is one of the classical NP-complete problems that appears already at the sources of NP-completes theory, among the original problems of Karp. It remains NP-complete both for directed and undirected graphs, as well as for the edge disjoint and vertex disjoint paths version. The corresponding natural optimization problem, when we are looking for the maximum number of terminator pairs that can be connected by disjoint paths is NP-hard.

There is also a capacitated version of the Disjoint Connecting Paths problem, also known as the Unsplittable Flow problem. In this task, a flow demand value is given for each origin- destination pair (s_i, t_i), as well as a capacity value is known for each edge. The requirement is to find a system of paths, connecting the respective source-destination pairs, such that the capacity constraint of each edge is obeyed, i.e., the sum of the flows of paths that traverse the edge cannot be more than the capacity of the edge. The name Unsplittable Flow expresses the requirement that between each source-destination pair the flow must follow a single route, it cannot split. Note that here the disjointness of the paths themselves is not required a priori, but can be enforced by the capacity constraints. The Unsplittable Flow problem is important in communication network design and routing applications.

In this chapter, after reviewing some existing results, we show that the Unsplittable Flow problem, which is NP-complete, becomes efficiently solvable by a relatively simple algorithm if we impose a mild and practically well justifiable restriction on the instance.

5.3 Related work

Considerable work was done on the Disjoint Connecting Paths problem, since its first appearance as an NP-complete problem in [14].

One direction of research deals with finding the “heart” of the difficulty: which are the simplest restricted cases that still remain NP-complete (Or NP-hard if the optimization version is considered, where we look for the maximum number of connecting paths, allowing that possibly not all source-destination pairs will be connected). [29] proves, motivated by VLSI layout design, that the problem remains NP-complete even for graphs as regular as a two-dimensional mesh. If we restrict ourselves to undirected planar graphs with each vertex having degree at most three, the problem also remains NP- complete, as proven by Middendorf and Pfeiffer [30]. The optimization version remains NP- hard for trees with parallel edges, although there the decision problem is already solvable in polynomial time.

The restriction that we only allow paths which connect each source node with a dedicated target is essential. If this is relaxed and we are satisfied with edge disjoint paths that connect each source s_i with some of destinations t_j but not necessarily with t_i, then the problem becomes solvable with classical network flow techniques. Thus, the prescribed matching of sources and destinations causes a dramatic change in the problem complexity. Interestingly, it becomes already NP-complete if we require that just one of the sources is connected to a dedicated destination, the rest is relaxed as above [31].

Another group of results produces polynomial time algorithmic solutions for finding the paths, possibly using randomization, in special classes of graphs. For example, Middendorf and Pfeiffer

[30] proves the following. Let us represent the terminator pairs by demand edges. These are additional edges that connect a source with its destination. If this extended graph is embeddable in the plane such that the demand edges lie in a bounded number of faces of the original graph, then the problem is solvable in polynomial time. (The faces are the planar regions bordered by the curves that represent the edges in the planar embedding, i.e., in drawing the graph in the plane). Thus, this special case requires that, beyond the planarity of the extended graph, the terminators are concentrated in a constant number of regions (independent of the graph size), rather than spreading over the graph.

A deep theoretical result, due to Robertson and Seymour [32], is that for general graphs the problem can be solved in polynomial time if the number k of paths to be found is constant (i.e. cannot grow with the size of the graph). Broder, Frieze, Suen and Upfal [33] consider the case of random graphs and provide a randomized algorithm that, under some technical conditions, finds a solution with high probability in time $O(nm^2)$ for a graph of n vertices and m edges.

Another line of research aims at finding approximations to the optimization version. An algorithm is said to be an f(n)-approximation if it can connect a subset of the terminator pairs by disjoint paths such that this subset is at most f(n) times smaller than the optimum in a graph of n vertices. For example, in this terminology a 2-approximation algorithm always reaches at least the half of the optimum, or an O(logn)-approximation reaches at least a c/ log n fraction of the optimum, for $n > n_0$ with some constants c, n_0.

Various approximations have been presented in the literature. For example, Garg, Vazi- rani and Yannakakis [34] provide a 2-approximation for trees with parallel edges. Aumann and Rabani [35] gives an O(log n)-approximation for the 2-dimensional mesh. Kleinberg and Tardos [36] present an O(log n)-approximation for a larger subclass of planar graphs, they call “nearly Eulerian, uniformly high-diameter planar graphs” (the rather technical definition is omitted here). For the general case an approximation factor of min $\{\sqrt{m}, m/opt\} = O(\sqrt{m})$ is known to be achievable , where m is the number of edges and opt is the optimum, i.e., the maximum number of disjoint connecting paths between the source- destination pairs. Similar bounds apply for the Unsplittable Flow problem, as well. Bounds have been also found in terms of special (less trivial) graph parameters.

5.4 A Simple Practical Approximation

The various above referenced solutions are rather complicated, which is certainly not helpful for practical applications, in particular in large, complex networks. Our approach for providing a simple solution to the unsplittable flow problem based on the following idea. We “cut down” a small part of the solution space by slightly reducing the edge capacities. In other words, we exclude solutions that are close to saturating some edge, as explained below.

Let V_i be the given flow demand of the i^{th} connecting path. We normalize these demands such that $V_i <= 1$ for every i. Let C_j be the capacity of edge j. The graph is assumed directed and the edges are numbered from 1 through m. Recall that a feasible solution of the problem is a set of connecting (directed) paths that satisfy the edge capacity constraints, that is, on each edge j the sum of the V_i values of those paths that traverse the edge does not exceed C_j. As mentioned earlier, deciding whether a feasible solution exist at all is a di cult (NP-complete) problem.

On the other hand, not all feasible solutions are equally good from the practical viewpoint. For example, if a route system in a network saturates or nearly saturates some links, then it is not preferable because it is close to being overloaded. For this reason, let us assign a parameter

$0 < \rho_j < 1$ to each edge j, such that ρ_j will act as a "safety margin" for the edge. More precisely, let us call a feasible solution a safe solution with parameters ρ_j , j = 1, . . . , m, where m is the number of edges, if it uses at most $\widetilde{C_j} = \rho_j C_j$ capacity on edge j.

Now, the interesting thing is that if we restrict ourselves to only those cases when a safe solution exists, then the hard algorithmic problem becomes solvable by a relatively simple randomized algorithm. With very high probability the algorithm finds a solution in polynomial time, whenever there exists a safe solution.

The price is that we exclude those cases when a feasible solution still possibly exists, but there is no safe solution. This means, in these cases all feasible solutions are undesirable, in the sense that they make some edges nearly saturated. In these marginal cases the algorithm may find no solution at all. This approach constitutes a new avenue to approximation, in the sense that instead of giving up finding an exact solution, we rather restrict the search space to a (slightly) smaller one. When, however, the algorithm finds any solution, then it is an exact (not just approximate) solution.

Now let us choose the safety margin ρ_j for a graph of m edges as

$$\rho_j = 1 - (e-1)\sqrt{\frac{ln\, 2m}{C_j}} \approx 1 - 1.71\sqrt{\frac{ln\, 2m}{C_j}} \qquad (5\text{-}1)$$

Where ln denotes the natural logarithm $\log_e$. Note that j tends to 1 with growingC_j, even if the graph also grows, but C_j grows faster than the logarithm of the graph size, which is very reasonable (note that doubling the number of edges will increase the natural logarithm by less than 1). For example, if in a graph each edge capacity is 1000 units, measured in relative units, such that the maximum path flow demand is 1, and the graph has 200 edges, then $\rho \approx 0{,}97$.

Now we outline how the algorithm works. To make it even closer to practical applications, we also assume that cost factors are assigned to the edges and we are looking for a feasible solution with small cost, where the cost incurred on an edge is proportional with the demand routed through it.

Algorithm

Step 1 Initialization

Compute the $\widetilde{C_j} = \rho_j C_j$ values with $\rho_j$set according to (5-1).

Step 2 Flow relaxation

Solve the continuous minimum cost multicommodity flow relaxation of the problem, using the $\widetilde{C_j}$capacities. This can be done by standard linear programming. In case the flow problem has no solution then declare “no safe solution exists” and STOP.

Step 3 Randomized Rounding via Random Walk

For each source-destination pair u_i, v_i find a path via the following randomized rounding procedure. Start at the source and take the next node such that it is drawn randomly among the successor neighbors of the source, with probabilities proportional to the i^{th} commodity flow values on the edges from u_i to the successor neighbors in the directed graph. Continue this in a similar way: at each node choose the next one among its successor neighbors randomly, with probabilities that are proportional to the i^{th} commodity flow values. Finally, upon arrival at v_i, we store the found (u_i, v_i) path.

Step 4 Feasibility Check and Repetition

Having found a system of paths in the previous steps, check whether it is a feasible solution. If so, then STOP, else repeat from Step 2.If after repeating r times (r is a fixed parameter) none of the runs are successful then declare "No solution is found" and STOP.

It is clear from the above informal description that the algorithm has practically feasible complexity, since the most complex part of it is solving a multicommodity flow problem that can be done by linear programming. It is repeated r times where r is a parameter, chosen by us. The main property of the algorithm is shown in the following theorem.

Theorem 1: If a safe solution exists, the algorithm finds a feasible solution with probability at least $1 - 2^{r}$

Proof. Since a safe solution is also a feasible solution of the multicommodity flow relaxation, therefore, if there is no flow solution in Step 2, then no safe solution can exist either.

Step 3 transforms the flow solution into paths. To see that they are indeed paths, observe that looping cannot occur in the randomized branching procedure, because if a circle arises on the way, that would mean a circle with all positive flow values for a given commodity, which could be canceled from the flow of that commodity, thus contradicting to the minimum cost property of the flow. Furthermore, since looping cannot occur, we must reach the destination via the procedure in at most n steps, where n is the number of nodes.

Now a key observation is that if we build the paths with the described randomization between the i^{th} source-destination pair, then the expected value of the load that is put on any given edge by these paths will be exactly the value of the i^{th} commodity flow on the link. This follows from the fact that the branching probabilities are flow-proportional.

From the above we know that the total expected load of an edge, arising from the randomly chosen paths, is equal to the total flow value on the edge. What we have to bound is the deviation of the actual load from this expected value. Let F_j be the flow (=expected load) on edge j. This arises in the randomized procedure as

$$F_j = E(\sum_i V_i X_i)$$

where X_i is a random variable that takes the value 1 if the i^{th} path contributes to the edge load, otherwise it is 0. The construction implies that these random variables are independent. Now consider the random variable

$$\psi_j = \sum_i V_i X_i$$

We have $\psi_j = F_j$. The probability that ψ_j deviates form its expected value by more than a factor of can be bounded by the tail inequality found in [10]:

$$Pr(\psi_j > (1+\delta)F_j) < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{F_j}$$

It can be calculated from this [10] that if we want to guarantee that the bound does not exceed a given value $\epsilon > 0$, then it is sufficient to satisfy

$$\delta \le (e-1)\sqrt{\frac{\ln\left(\frac{1}{\epsilon}\right)}{F_j}}$$

Let us choose $\epsilon = 1/2m$. Then we have

$$Pr\left(\psi_j > \left(1+(e-1)\sqrt{\frac{ln\,2m}{F_j}}\right)F_j\right) < \frac{1}{2m}$$

Since the bound that we do not want to exceed is the edge capacityC_j, therefore, if

$$C_j \ge \left(1+(e-1)\sqrt{\frac{ln\,2m}{F_j}}\right)F_j \qquad (5\text{-}2)$$

is satisfied, then we have

$$Pr(\psi_j > C_j) < \frac{1}{2m}$$

If this holds for all edges, that yields

$$Pr(\exists_j : \psi_j > C_j) \leq \sum_{j=1}^{m} Pr(\psi_j > C_j)$$

$$< m\frac{1}{2m}$$

$$= \frac{1}{2}$$

Thus, the probability that the found path system is not feasible is less than 1/2. Repeating the procedure r times with independent randomness, the probability that none of the trials provide a feasible solution is bounded by $1/2^r$, that is, the failure probability becomes very small, already for moderate values of r.

Finally, expressing F_j form (5-2) we obtain

$$F_j \leq C_j\left(1-(e-1)\sqrt{\frac{ln\, 2m}{C_j}}\right) = \rho_j C_j$$

which shows that the safety margin is correctly chosen, thus completing the proof.

We have presented a simple, efficient solution for the NP-complete Unsplittable Flow problem in directed graphs. The simplicity and efficiency is achieved by sacrificing a small part of the solution space. The sacrificed part only contains scenarios where some edges are very close to saturation. Therefore, the loss is not significant, since the almost saturated solutions are typically undesired in practical applications, such as network design. The approach constitutes a new avenue to approximation, in the sense that instead of giving up finding an exact solution, we rather restrict the search space to a (slightly) smaller one. When, however, the algorithm finds any solution, which happens with high probability, then it is an exact (not just approximate) solution.

5.5 Utilizing Network Structure to Accelerate Markov Chain Monte Carlo Algorithms

We consider the problem of estimating the measure of subsets in very large networks. A prime tool for this purpose is the Markov Chain Monte Carlo (MCMC) algorithm. This algorithm, while extremely useful in many cases, still often suffers from the drawback of very slow convergence. We show that in a special, but important case, it is possible to obtain significantly better bounds on the convergence rate. This special case is when the huge state space can be aggregated into a smaller number of clusters, in which the states behave approximately the same way (but their behavior still may not be identical). A Markov chain with this structure is called quasi-lumpable. This property allows the aggregation of states (nodes) into clusters. Our main contribution is a rigorously proved bound on the rate at which the aggregated state distribution approaches its limit in quasi-lumpable Markov chains. We also demonstrate numerically that in certain cases this can indeed lead to a significantly accelerated way of estimating the measure of subsets. The result can be a useful tool in the analysis of complex networks, whenever they have a clustering that aggregates nodes with similar (but not necessarily identical) behavior [37].

The Markov Chain Monte Carlo (MCMC) method is one of the most frequently used algorithms to solve hard counting, sampling and optimization problems. This is relevant for many areas, including complex networks, physics, communication systems, computational biology, optimization, data mining, big data analysis, forecast problems, prediction tasks, and innumerable others. The success and influence of the method is shown by the fact that it has been selected as one of the top 10 of all algorithms in the 20th century.

The MCMC algorithm also plays an important role in large, complex networks. In this chapter, we consider the following regularly occurring application of the MCMC method:

Consider a very large graph G, with node set S, and let A⊆S be a subset of the nodes. We would like to estimate the relative size of A, that is, the goal is to obtain a good estimate of the value

$$p = \frac{|A|}{|S|} \qquad (5\text{-}3)$$

More generally, if a random walk is considered on the graph, with stationary distribution π, then we would like to estimate π(A), the stationary probability of being in A. In the special case when π is the uniform distribution, we get back the formula (5-3).

If we can take random samples from S, according to the stationary distribution, then an obvious estimate with good properties is the relative frequency of the event that the sample falls in A. Unfortunately, in most nontrivial cases of interest, this sampling task is not feasible. The reason is that often the large set S is defined implicitly. Examples are the set of all cliques in a graph, or the set of all feasible solutions to an optimization problem, and many others. No efficient general method is known to sample uniformly at random from such complex sets.

An important application in telecommunication networks is to estimate blocking probabilities. More generally, if we have a large system, with an enormous state space, we may want to estimate that the actual state falls in a specific subset. For example, if the state space consists of all possible load values of the network links, which leads to a state space of astronomical size, we may want to know what the probability is that at most k links are overloaded, for some value of k.

At this point, the MCMC does a very good service. If we define a Markov chain in which the states are the elements of Sand the transitions are based on simple local operations, then we can very often obtain a Markov chain with uniform, or some other simple stationary distribution over S. Then, if we run this chain long enough so that it gets close to the stationary distribution, then the state where we stop the chain will be a good approximation of a random sample over S, distributed according to the stationary distribution. Then by repeating the process sufficiently many times, and by counting the relative frequency that the random sample falls in A, we can get a good estimate of the probability measure of A.

The key difficulty is, however, that we should run the chain long enough to get sufficiently close to the stationary distribution. This time is often referred to as mixing time. If the mixing time grows only polynomially with the size of the problem, e.g. with the size of the graph, then we say that the chain is rapidly mixing. Unfortunately, in many cases of interest the mixing time grows exponentially with the problem parameters, so in many important cases the Markov chain is mixing very slowly.

What we are interested in is whether it is possible to speed up the running time. It is clear that if we want to estimate the size of any possible subset, then we really need to get close to the stationary distribution, since only this distribution can guarantee that the probability of the random state falling in the set is really the relative size of the set. On the other hand, if we only want to estimate the relative size of a specific subset A, then it is enough for us if we reach a distribution in which the measure of A is close to the stationary measure, but this does not have to hold for every other set. In other words, if πtdenotes the state distribution after t steps and π is the stationary distribution, then we want to choose t such that $|\pi t(A) - \pi(A)|$ is small, but the same does not have to hold for all other sets. This makes it possible to reduce the required value of t, that is, to speed up the algorithm. In this chapter we investigate under what conditions it is possible to obtain such a speed-up.

The main result is that the structure of the chain, that is, the network structure, can significantly help, if it has some special properties. Specifically, if the Markov chain is close to a so called lumpable chain, then remarkable speedup is possible. In other words, in this case we can indeed capitalize on the particular network structure to accelerate the method. Below we informally explain what the concept of lumpability means. The formal definition will follow in the next section.

The concept of lumpability stems from the following observation: it is very useful if the state space can be partitionedsuch that the states belonging to the same partition class “behave the same way,” in the sense defined formally in the next section. This is the concept of lumpability. Informally speaking, it means that some sets of states can be lumped together (aggregated) and replaced by a single state, thus obtaining a Markov chain which has a much smaller state space, but its essential behavior is the same as the original.

In some cases, the lumpability of the Markov chain can have a very significant effect on the efficiency of the model. A practical example is discussed in [38], where the authors present a fast algorithm to compute the PageRank vector, which is an important part of search engine algorithms in the World Wide Web. The PageRank vector can be interpreted as the stationary distribution of a Markov chain. This chain has a huge state space, yielding excessive computation times. This Markov chain, however, is lumpable. Making use of the lumpability, the computation time can be reduced to as low as 20% of the original, according to the experiments presented in [38].

Unfortunately, it happens relatively rarely that the Markov chain satisfies the definition of lumpability exactly. This motivates the concept of quasi-lumpability. Informally, a Markov chain is quasi-lumpable if its transition matrix is obtainable by a small perturbation from a matrix that exactly satisfies the lumpability condition (see the formal definition in the next section).

In this chapter we are interested in the following problem, which is often encountered in applications: how long do we have to run the Markov chain if we want to get close to the stationary distribution within a prescribed error? While the general question is widely discussed in the literature, we focus here on a less researched special case: how much gain can the convergence speed enjoy, if we can capitalize on the special structure of quasi-lumpability.

5.6 Aggregation in Markov Chains

We assume the reader is familiar with the basic concepts of Markov chains. We adopt the notation that a Markov chain M is given by a set S of states and by a transition probability matrix P, so we write$M = (S, P)$. This notation does not include the initial distribution, because it is assumed arbitrary.

Let us first define the concept lumpability of a Markov chain. Informally, as mentioned in the Introduction, a chain is lumpable if its states can be aggregated into larger subsets of S, such that the aggregated (lumped) chain remains a Markov chain with respect to the set-transition probabilities (i.e., it preserves the property that the future depends on the past only through the present). Note that this is generally not preserved by any partition of the state space. Let us introduce now the formal definition.

Definition 1.

(Lumpability of Markov chain) Let $M = (S, P)$ be a Markov chain. Let $Q = \{A1, \dots, Am\}$ be a partition of S. The chain M is called lumpable with respect to the partition Q if for any initial distribution, the relationship

$$Pr(Xt \in Aj \mid Xt - 1 \in Ai1, \dots, Xt - k \in Aik) \qquad (5\text{-}4)$$
$$= Pr(Xt \in Aj|Xt - 1 \in Ai1)$$

holds for any $t, k, j, i_1, \dots, i_k$, whenever these conditional probabilities are defined (i.e., the conditions occur with positive probability). If the chain is lumpable, then the state set of the lumped chain is Q and its state transition probabilities are defined by

$$\tilde{p}_{ij} = Pr(Xt \in Aj|Xt - 1 \in Ai) \qquad (5\text{-}5)$$

Checking whether a Markov chain is lumbable would be hard to do directly from the definition. That is why it is useful to have the following characterization, which is fundamental result on the lumpability of Markov chains. For simple description, we use the notation $p(x, A)$ to denote the probability that the chain moves into a set $A \subseteq S$ in the next step, given that currently it is in the state $x \in S$. Note that x itself may or may not be in A [31].

Theorem 1. (Necessary and sufficient condition for lumpability) A Markov chain $M = (S, P)$ is lumpable with respect to a partition $Q = \{A1, \dots, Am\}$ of S if and only if for any i,j the value of $p(x, Aj)$ is the same for every x∈Ai. These common values define the transition probabilities $p\hat{}(Ai, Aj)$ for the lumped chain, which is a Markov chain with state set Q and state transition probabilities

$$p\hat{}(Ai, Aj) = p(x, Aj) = Pr(Xt \in Aj|Xt - 1 \in Ai)$$

where x is any state in Ai.

Informally, the condition means that a move from a set $Ai \in Q$ to another set $Aj \in Q$ happens with probability $p(x, Aj)$, no matter which $x \in Ai$ is chosen. That is, any $x \in Ai$ has the property that the probability of moving from this x to the set Aj in the next step is the same for every $x \in Ai$. The sets Ai, Aj are partition classes of Q. We also allow $i = j$, so they may coincide.

Whenever our Markov chain is lumpable, we can reduce the number of states by the above aggregation, and it is usually advantageous for faster convergence (a specific bound will be proven in Section 3).

It is worth noting that lumpability is a rather special property, and one has to be quite lucky to encounter a real-life Markov chain that actually has this property. Sometimes it happens (see, e.g., the example in the Introduction about PageRank computation), but it is not very common. Therefore, let us now relax the concept of lumpability to broaden the family of the considered Markov chains. The extended condition, as explained below, is called quasi-lumbability.

Informally, a Markov chain is called quasi-lumpable or ϵ-quasi-lumpable or simply ϵ-lumpable, if it may not be perfectly lumpable, but it is "not too far" from that. This "ϵ-closeness" is defined in [10,11] in a way that the transition matrix can be decomposed as $P = P^{-} + P^{\epsilon}$. Here P^{-} is a component-wise non-negative lower bound for the original transition matrix P, such that P^{-} satisfies the necessary and sufficient condition of Theorem 1. The other matrix, P^{ϵ}, represents a perturbation. It is an arbitrary non-negative matrix in which each entry is bounded by ϵ. This definition is not very easy to visualize, therefore, we use the following simpler but equivalent definition.

Definition 2. (ϵ-lumpability) Let $\epsilon \geq 0$. A Markov chain $M = (S, P)$ is called ϵ-lumpable with respect to a partition $Q = \{A1, \ldots, Am\}$ of S if

$$|p(x, Aj) - p(y, Aj)| \leq \epsilon$$

holds for any $x, y \in Ai$ and for any $i, j \in \{1, \ldots, m\}$.

Note that if we take $\epsilon = 0$, then we get back the ordinary concept of lumpability. Thus, quasi-lumpability is indeed a relaxation of the original concept. It can also be interpreted in the following way. If $\epsilon > 0$, then the original definition of lumpability may not hold. This means, the aggregated process may not remain Markov. i.e., it does not satisfy (2). On the other hand, if ϵ is small, then the aggregated process will be, in a sense, “close" to being Markov, that is, to satisfying (2).

What we are interested in is the convergence analysis of quasi-lumpable Markov chains, typically for a small value of ϵ. For the analysis we need to introduce another definition.

Definition 3. (Lower and upper transition matrices) Let $M = (S, P)$ be a Markov chain which is ϵ-lumpable with respect to a partition $Q = \{A1, \ldots, Am\}$. The lower and upper transition matrices $L = [l_{ij}]$ and $U = [u_{ij}]$ are defined as $m \times m$ matrices with entries

$$l_{ij} = \min_{x \in A_i} p(x, A_j) \quad and \quad u_{ij} = \max_{x \in A_i} p(x, A_j)$$

respectively, for $i, j = 1, \dots, m$.

Note that it always holds (component-wise) that $L \leq U$. If the chain is lumpable, then these matrices coincide, so then $L = U = \tilde{P}$, where $\tilde{P}$ is the transition matrix of the lumped chain. If the chain is ϵ-lumpable, then L and U differ at most by ϵ in each entry.

Generally, L and U are not necessarily stochastic matrices (A vector is called stochastic if each coordinate is non-negative and their sum is 1. A matrix is called stochastic if each row vector of it is stochastic.), as their rows may not sum up to 1.

5.7 Convergence Analysis

An important concept in Markov chain convergence analysis is the ergodic coefficient, see, e.g., [12]. It is also called coefficient of ergodicity.

Definition 4. (Ergodic coefficient) Let $P = [pij]$ be an $n \times n$ matrix. Its ergodic coefficient is defined as

$$\rho(P) = \frac{1}{2} \max_{i,j} \sum_{k=1}^{n} |p_{ik} - p_{jk}|$$

The ergodic coefficient is essentially the largest L1 distance that occurs between different row vectors of the matrix P. That is, in a sense, it captures how diverse are the row vectors of the matrix. The 1/2 factor is only for normalization purposes. For stochastic matrices two important properties of the ergodic coefficient are the following:

(i) $0 \leq \rho(P) \leq 1$

(ii) $\rho(AB) \leq \rho(A)\rho(B)$

The importance of the ergodic coefficient lies in its relationship to the convergence rate of the Markov chain. It is well known that the convergence rate is determined by the second largest eigenvalue of the transition matrix (that is, the eigenvalue which has the largest absolute value less than 1), If this eigenvalue is denoted by λ_1, then the convergence to the stationary distribution happens at a rate of $O(\lambda_i^t)$, where t is the number of steps. It is also known [12] that the ergodic coefficient is always an upper bound on this eigenvalue, it satisfies $\lambda_1 \leq \rho(P) \leq 1$. Therefore, the distance to the stationary distribution is also bounded by $O(\rho(P)^t)$. Thus, the smaller is the ergodic coefficient, the faster convergence we can expect. Of course, it only provides any useful bound if $\rho(P) < 1$. If $\rho(P) = 1$ happens to be the case, then it does not directly provide a useful bound on the convergence rate, since then $\rho(P)$t remains 1. In this situation, a possible way out is considering the k-step transition matrix P^k for some constant integer k. If k is large enough, then we can certainly achieve $\rho(P^k) < 1$, since it is known [39] that $\lim_{k \to \infty} \rho(P^k) = 0$.

Now we are ready to present our main result, which is a bound on how fast will an ϵ-lumpable Markov chain converge to its stationary distribution on the sets that are in the partition, which is used in defining the ϵ-lumpability of the chain. We are going to discuss the applicability of the result in the next section.

Theorem 2.

Let $\epsilon \geq 0$ and $M = (S, P)$ be an irreducible, aperiodic Markov chain with stationary distribution π. Assume the chain is ϵ-lumpable with respect to a partition $Q = \{A1, \dots, Am\}$ of S. Let ρ be any upper bound on the ergodic coefficient of the lower transition matrix L (Definition 3), that is, $\rho(L) \leq \rho$. Let π0 be any initial probability distribution on S, such that $P(Xt \in Ai) > 0$ holds for any i, and $t = 0,1,2, \ldots,$ whenever the chain starts from π_0 Then for every $t \geq 1$ the following estimation holds:

$$\sum_{i=1}^{m} |\pi_t(A_i) - \pi(A_i)| \leq 2\left(\rho + \frac{\epsilon m}{2}\right)^t + \epsilon m \frac{1 - \left(\rho + \frac{\epsilon m}{2}\right)^2}{1 - \rho - \epsilon m/2}$$

assuming $\rho + \epsilon m/2 < 1$.

Remark: Recall that the parameter ϵ quantifies how much the Markov chain deviates from the ideal lumpable case, see Definition 2. In the extreme case, when $\epsilon = 1$, every Markov chain satisfies the definition. This places an“upward pressure” on ϵ: the larger it is, the broader is the class of Markov chains to which ϵ-lumpability applies. On the other hand, a downward pressure is put on ϵ by Theorem 2: the convergence bound is only meaningful, if $\rho + \epsilon m/2 < 1$ holds. This inequality can be checked for any particular ϵ, since it is assumed that ρ and m are known parameters. Furthermore, the smaller is ϵ, the faster is the convergence. Therefore, the best value of ϵ is the smallest value which still satisfies Definition 2 for the considered state partition [40].

For the proof of Theorem 2 we need a lemma about stochastic vectors and matrices:

Lemma 1. Let x, y be n-dimensional stochastic vectors and $B1, \ldots, Bk;\ C1, \ldots, Ck$ be $n \times n$ stochastic matrices. If $\rho(Bi) \leq \rho 0$ and $\rho(Ci) \leq \rho_0$ for all $i, 1 \leq i \leq k$, then

$$\| xB1 \ldots Bk - yC1 \ldots Ck \| \leq \rho_0^k \|x - y\| + \left(\sum_{j=0}^{k-1} \rho_0^j \right) E$$

where $E = \max_i \parallel B_i - C_i \parallel$ The vector norm used is the L1-norm $\|x\| = \sum_{i=1}^{n}|x_i|$and the matrix norm is

$$\|A\| = \max_{z\neq 0}\frac{\|zA\|}{\|z\|} = \max_{i}\sum_{j=1}^{n}|a_{ij}|$$

for any $n \times n$ real matrix $A = [a_{ij}]$.

Lemma 1 can be proved via induction on k. Now, armed with the lemma, we can prove our theorem.

Proof of Theorem 2**.** Let π_0 be an initial state distribution of the Markov chain M, let πt be the corresponding distribution after t steps and $\pi = \lim_{t\to\infty}\pi_t$ be the (unique) stationary distribution of M. For a set $A \subseteq S$ of states the usual notations $\pi_t(A) = P(Xt \in A)$, $\pi(A) = \lim_{t\to\infty}\pi_t(A)$ are adopted.

Using the sets $A1, \dots, Am$ of the partition Q, let us define the stochastic vectors

$$\tilde{\pi}_t = (\pi_t(A_1), \dots, \pi_t(A_m)) \qquad (5\text{-}6)$$

for $t = 0,1,2, \dots$ and the $m \times m$ stochastic matrices

$$\tilde{\pi}_t = (\pi_t(A_1), \dots, \pi_t(A_m)) \qquad (5\text{-}7)$$

for $t = 1,2, \dots$. Let us call them aggregated state distribution vectors and aggregated transition matrices, respectively. Note that although the entries in (4) involve only events of the form $\{Xt \in Ak\}$, they may also depend on the detailed state distribution within these sets, which is in turn determined by the initial distribution $\pi 0$. In other words, if two different initial distributions give rise to the same probabilities for the events $\{Xt \in Ak\}$ for some t, they may still result in different conditional probabilities of the form $P(Xt + 1 \in Aj|Xt \in Ai)$, since the chain is not assumed lumpable in the ordinary sense. This is why the notations P˜t(π0),p(π0)t(i, j) are used. Also note that the conditional probabilities are well defined for any initial distribution allowed by the assumptions of the lemma, since then $P(Xt \in Ai) > 0$.

For any fixed t the events$\{Xt \in Ai\}, i = 1, \dots, m$, are mutually exclusive with total probability 1, therefore, by the law of total probability,

$$P(Xt + 1 \in Aj) = \sum_{i=1}^{m} P(Xt + 1 \in Aj|Xt \in Ai)P(Xt \in Ai), \qquad j = 1, \dots, m$$

holds. This implies $\widetilde{\pi}_{t+1} = \widetilde{\pi}_t \widetilde{P}_t(\pi_0)$, from which

$$\widetilde{\pi}_t = \widetilde{\pi}_0 \widetilde{P}_1(\pi_0) \dots \widetilde{P}_t(\pi_0) \qquad (5\text{-}8)$$

follows.

We next show that for any $t = 1,2, \dots$ the matrix $\widetilde{P}_t(\pi_0)$falls between the lower and upper transition matrices, i.e., $L \leq \widetilde{P}_t(\pi_0) \leq M$ holds. Let us use short notations for certain events: for any $i = 1, \dots, m$ and for a fixed $t \geq 1$ set $Hi = \{Xt \in Ai\}$, $H'i = \{Xt + 1 \in Ai\}$, and for $x \in S$ let $Ex = \{Xt = x\}$. Then $Ex \cap Ey = \emptyset$ holds for any $x \neq y$ and $\sum_{x \in S} E_x = 1$. Applying the definition of conditional probability and the law of total probability, noting that $P(Hi) > 0$ is provided by the assumptions of the lemma, we get

$$p_t^{(x_0)}(i, j) = P\left(H_j' \middle| H_i\right) \qquad = \frac{P(H'j \cap Hi)}{P(Hi)}$$

$$= \frac{\sum_{x \in S} P(H'j \cap Hi \cap Ex)}{P(Hi)}$$

$$= \frac{\sum_{x \in S}\big((H_j'\big|Hi \cap Ex)P(Hi \cap Ex)\big)}{P(Hi)}$$

$$= \sum\nolimits_{x \in S} P(H'j|Hi \cap Ex)\frac{P(H'j \cap Hi)}{P(Hi)}$$

$$= \sum\nolimits_{x \in S} P(H'j|Hi \cap Ex)\, P(E_x|H_i)$$

Whenever $x \notin Ai$ we have $P(Ex|Hi) = P(Xt = x|Xt \in Ai) = 0$. Therefore, it is enough to take the summation over Ai, instead of the entire S. For $x \in Ai$, however, $Hi \cap Ex = \{Xt \in Ai\} \cap \{Xt = x\} = \{Xt = x\}$ holds, so we obtain

$$p_t^{(\pi_0)}(i,j) = \sum_{x \in A_i} P(Xt + 1 \in Aj|Xt = x)P(Xt = x|Xt \in Ai)$$

Thus, $p_t^{(\pi_0)}(i,j)$ is a weighted average of the $P(Xt + 1 \in Aj|Xt = x)$ probabilities. The weights are $P(Xt = x|Xt \in Ai)$ so they are non-negative and sum up to 1. Further,

$$lij \leq P(Xt + 1 \in Aj|Xt = x) \leq uij$$

must hold, since lij,uij are defined as the minimum and maximum values, respectively, of

$$p(x, Aj) = P(Xt + 1 \in Aj|Xt = x)$$

over $x \in Ai$. Since the weighted average must fall between the minimum and the maximum, therefore, we have

$$l_{ij} \leq p_t^{(x_0)}(i,j) \leq u_{ij} \qquad (5\text{-}9)$$

that is,

$$L \leq \widetilde{P}_t(\pi_0) \leq M \qquad (5\text{-}10)$$

for any $t \geq 1$ and for any initial distribution π0 allowed by the conditions of the Theorem.

Let us now start the chain from an initial distribution π0 that satisfies the conditions of the Theorem. We are going to compare the arising aggregated state distribution vectors (3) with the ones resulting from starting the chain from the stationary distribution π. Note that, due to the assumed irreducibility of the original chain, $\pi(x) > 0$ for all $x \in S$, so π is also a possible initial distribution that satisfies the conditions $P(Xt \in Ai) > 0$.

When the chain is started from the stationary distribution π, then, according to (5), the aggregated state distribution vector at time t is $\tilde{\pi}\,\tilde{P}_1(\pi).\ldots\,.\tilde{P}_1(\pi)$ where $\tilde{\pi}$ is given as $\tilde{\pi}_t = \tilde{\pi}_t\tilde{P}_t(\pi_0)\ldots\tilde{P}_t(\pi_0)$. On the other hand, $P(Xt \in Ai)$ remains the same for all $t \geq 0$ if the chain starts from the stationary distribution. Therefore, we have

$$\tilde{\pi}\,\tilde{P}_1(\pi).\ldots\,.\tilde{P}_1(\pi) = \tilde{\pi} = (\pi(A_1),\ldots,(A_m)) \qquad \text{(5-11)}$$

When the chain starts from π0, then we obtain the aggregated state distribution vector

$$\tilde{\pi}_t = \tilde{\pi}_t\tilde{P}_t(\pi_0)\ldots\tilde{P}_t(\pi_0) \qquad \text{(5-12)}$$

after t steps. Now we can apply Lemma 1 for the comparison of (5-11) and (5-12). The roles for the quantities in Lemma 1 are assigned as $x = \tilde{\pi}_0$, $y = \tilde{\pi}_t$, $k = t$, $n = m$, and, for every $\tau = 1,\ldots,k$, $B_t = \tilde{P}_t(\pi_0)\ C_t = \tilde{P}_t(\pi)$. To find the value of ρ_0 recall that by (5-10) we have $L \leq \tilde{P}_t(\pi) \leq M$ and $L \leq \tilde{P}_t(\pi_0) \leq M$. Since any entry of U exceeds the corresponding entry of L at most by ϵ, therefore, by the definition of the ergodic coefficient, $\rho(\tilde{P}_t(\pi_0)) \leq \rho + \frac{\epsilon m}{2}$ and $\rho(\tilde{P}_t(\pi)) \leq \rho + \frac{\epsilon m}{2}$hold, where ρ is the upper bound on ρ(L). Thus, we can take $\rho_0 = \rho + \frac{\epsilon m}{2}$. With these role assignments we obtain from Lemma 1

$$\left\|\tilde{\pi}_0\tilde{P}_1(\pi_0)\ldots\tilde{P}_t(\pi_0) - \tilde{\pi}\tilde{P}_1(\pi)\ldots\tilde{P}_t(\pi)\right\| \leq \left(\rho + \frac{\epsilon m}{2}\right)^t \|\tilde{\pi}_0 - \tilde{\pi}\| + E$$

where $E = \max_r \|P_r(\pi_t) - P_r(\pi)\|$ and the norms are as in Lemma 1. Taking (5-11) and (5-12) into account yields

$$\|\tilde{\pi}_0 - \tilde{\pi}\| = \sum_{i=1}^{m} |\pi_t(A_i) - \pi(A_i)| \leq \left(\rho + \frac{\epsilon m}{2}\right)^2 \|\tilde{\pi}_0 - \tilde{\pi}\| + E \sum_{k=0}^{t-1} \left(\rho + \frac{\epsilon m}{2}\right)^k \tag{5-13}$$

Thus, it only remains to estimate $\|\tilde{\pi}_0 - \tilde{\pi}\|$and E. Given that$\tilde{\pi}_0$, $\tilde{\pi}$ are both stochastic vectors, we have $\|\tilde{\pi}_0 - \tilde{\pi}\| \leq \|\tilde{\pi}_0\| + \|\tilde{\pi}\| \leq 2$. Further,

$$E = \max_r \|P_r(\pi_t) - P_r(\pi)\| = \max_r \max_i \sum_{j=1}^{m} \left| P_r^{(x_0)}(i,j) - P_r^{(\pi)}(i,j) \right| \leq \epsilon m$$

since (6) holds for any considered $\tilde{\pi}_0$ (including π), and, by the definition of ϵ-lumpability, $u_{ij} - l_{ij} \leq \epsilon$. Substituting the estimations into (10), we obtain

$$\sum_{i=1}^{m} |\pi_t(A_i) - \pi(A_i)| \leq 2\left(\rho + \frac{\epsilon m}{2}\right)^2 + \epsilon m \sum_{k=0}^{t-1} \left(\rho + \frac{\epsilon m}{2}\right)^2 = 2\left(\rho + \frac{\epsilon m}{2}\right)^2 + \epsilon m \frac{1 - \left(\rho + \frac{\epsilon m}{2}\right)^2}{1 - \rho - \epsilon m/2}$$

proving the Theorem. □

If the chain happens to be exactly lumpable, then we get a "cleaner" result. Let $\tilde{\pi}_t$ be the state distribution of the lumped chain after t steps and let $\tilde{\pi}$ be its stationary distribution. For concise description let us apply a frequently used distance concept among probability distributions. If p, q are two discrete probability distributions on the same set S, then their total variation distance $D_{TV}(p, q)$ is defined as

$$D_{TV}(p,q) \leq \frac{1}{2} \sum_{x \in S} |p(x) - q(x)|$$

It is well known that $0 \leq D_{TV}(p, q) \leq 1$ holds for any two probability distributions. It is also clear from the definition of the ergodic coefficient that it is the same as the maximum total variation distance occurring between any two row vectors of the transition matrix.

Note that exact lumpability is the special case of ϵ-lumpability with $\epsilon = 0$. Therefore, we immediately obtain the following corollary.

Corollary 1**.** If the Markov chain in Theorem 2 is exactly lumpable, then in the lumped chain for any $t = 0,1,2, \ldots$ the following holds:

$$D_{TV}(\tilde{\pi}_t, \tilde{\pi}) \leq \rho^t$$

where $\rho = \rho(\tilde{P})$ is the ergodic coefficient of the transition matrix $\tilde{P}$ of the lumped chain.

Proof**.** Take the special case $\epsilon = 0$ in Theorem 2. □

5.8 Numerical Demonstration

Let us consider the following situation. Let M be a Markov chain with state space S. Assume we want to estimate the stationary measure $\pi(A)$ of a subset $A \subseteq S$. A practical example of such a situation is to estimate the probability that there is at most k blocked links, for some constant k, in a large communication network. Here the state space is the set S of all possible states of all the links. The state of a link is the current traffic load of the link, and it is blocked if the load is equal to the link capacity, so it cannot accept more traffic. Within this state space the considered subset A is the subset of states in which among all links at most k are blocked. Therefore, the relevant partition of S is $\{A, S - A\}$. This is motivated by real-world application, since the number of blocked links critically affects network performance. When considering the loss of traffic due to blocking, the models of these networks are often called loss networks. For detailed background information on loss networks [31]. Of course, we can also consider other events in the network. For example, at most a given percentage of traffic is blocked, without specifying how many links are involved in the blocking.

In many cases we are unable to directly compute $\pi(A)$. This task frequently has enormous complexity, for the theoretical background. Then a natural way to obtain an estimation of $\pi(A)$ is simulation. That is, we run the chain from some initial state, stop it after t steps and check out whether the stopping state is in A or not. Repeating this experiment a large enough number of times, the relative frequency of ending up in A will give a good estimation of the measure of $\pi_t(A)$. If tis chosen such that πt is close enough to the stationary distribution π for any initial state, then we also obtain a good estimation for $\pi(A)$. This is the core idea of the Markov Chain Monte Carlo approach.

Unfortunately, Markov chains with large state space often converge extremely slowly. Therefore, we may not get close enough to π after a reasonable number of steps. In such a case our result can do a good service, at least when the chain satisfies some special requirements. As an example, let us consider the following case. First we examine it using our bounds, then we also study it through numerical experiments.

Assume the set $A \subseteq S$ has the property that its elements behave similarly in the following sense: for any state $x \in A$ the probability to move out of A in the next step, given that the current state is x, is approximately the same. Similarly, if $x \notin A$, then moving into A in the next step from the given x has approximately the same probability for any $x \notin A$. To make this assumption formal, assume there are values p_0, q_0, ϵ, such that the following conditions hold:

(A) If $x \in A$ then $p_0 \leq p(x, \bar{A}) \leq p_0 + \epsilon$ where $\bar{A} = S - A$. This means, the smallest probability of moving out of Afrom any state in x∈A is at least p0, and the largest such probability is at most $p0 + \epsilon$.

(B) If $x \in \bar{A}$ then $q_0 \leq p(x, A) \leq q_0 + \epsilon$. Similarly to the first case, this means that the smallest probability of moving into A from any state in $x \notin A$ is at least q_0, and the largest such probability is at most$q_0 + \epsilon$. (We choose ϵ such that it can serve for this purpose in both directions.)

(C) To avoid degenerated cases, we require that the numbers p_0, q_0, ϵ satisify $p_0 + \epsilon < 1$, $q_0 + \epsilon < 1$ and $0 < p_0 + q_0 < 1$. The other state transition probabilities (within A and $\bar{A}$) can be completely arbitrary, assuming, of course, that at any state the outgoing probabilities must sum up to 1.

Let us now apply our main result, Theorem 2, for this situation. The parameters will be as follows: m, the number of sets in the partition, is 2, since the partition is $(A, \bar{A})$. The matrices L,U become the following:

$$L = \begin{bmatrix} 1 - p_0 - \epsilon & p_0 \\ q_0 & 1 - q_0 - \epsilon \end{bmatrix} \quad U = \begin{bmatrix} 1 - p_0 & p_0 + \epsilon \\ q_0 + \epsilon & 1 - q_0 \end{bmatrix}$$

where the distributions π˜t,π˜ are over the sets of the partition $(A, \bar{A})$, not on the original state space. Note that in our case we actually have$D_{TV}(\tilde{\pi}_t, \tilde{\pi}) = |\pi t(A) - \pi(A)|$ due to the fact that $|\pi t(A) - \pi(A)| = |\pi t(\bar{A}) - \pi(\bar{A})|$. Therefore, we obtain the estimation directly for the set A:

$$|\pi t(A) - \pi(A)| \leq (1 - p_0 - q_0)^t + \epsilon \frac{1 - (1 - p_0 - q_0)^t}{p_0 + q_0} \qquad (5\text{-}14)$$

If p0+q0 is not extremely small, then the term $(1 - p_0 - q_0)^t$will quickly vanish, as it approaches 0 at an exponential rate. Therefore, after a reasonably small number of steps, we reach a distribution πt from any initial state, such that approximately the following bound is satisfied:

$$|\pi t(A) - \pi(A)| \leq \frac{\epsilon}{p_0 + q_0} \qquad (4\text{-}15)$$

It is quite interesting to note that neither the precise estimation (5-14), nor its approximate version (5-15) depend on the size of the state space.

Now we demonstrate via numerical results that the obtained bounds indeed hold. Moreover, they are achievable after a small number of Markov chain steps, that is, with fast convergence. We simulated the example with the following parameters: n=100 states, $p_0 = q_0 = 0.25$, and ϵ=0.1. The set A was a randomly chosen subset of 50 states. The transition probabilities were also chosen randomly, with the restriction that together with the other parameters they had to satisfy conditions (A), (B), (C).

Figure 5-1 shows the relative frequency of visiting A, as function of the number of Markov chain steps. It is well detectable that the chain converges quite slowly. Even after many iterations the deviation from the stationary probability π(A) does not visibly tend to 0. On the other hand, it indeed stays within our error bound:

$$|\pi t(A) - \pi(A)| \leq \frac{\epsilon}{p_0 + q_0} = \frac{0.1}{0.25 + 0.25} = 2 \times 0.1$$

as promised. Having observed this, it is natural to ask, how soon can we reach this region that is, how many steps are needed to satisfy the bound? This is shown in Figure 5-2. We can see that after only 10 iterations, the error bound is already satisfied. Note that this is very fast convergence, since the number of steps to get within the bound was as little as 10% of the number of states.

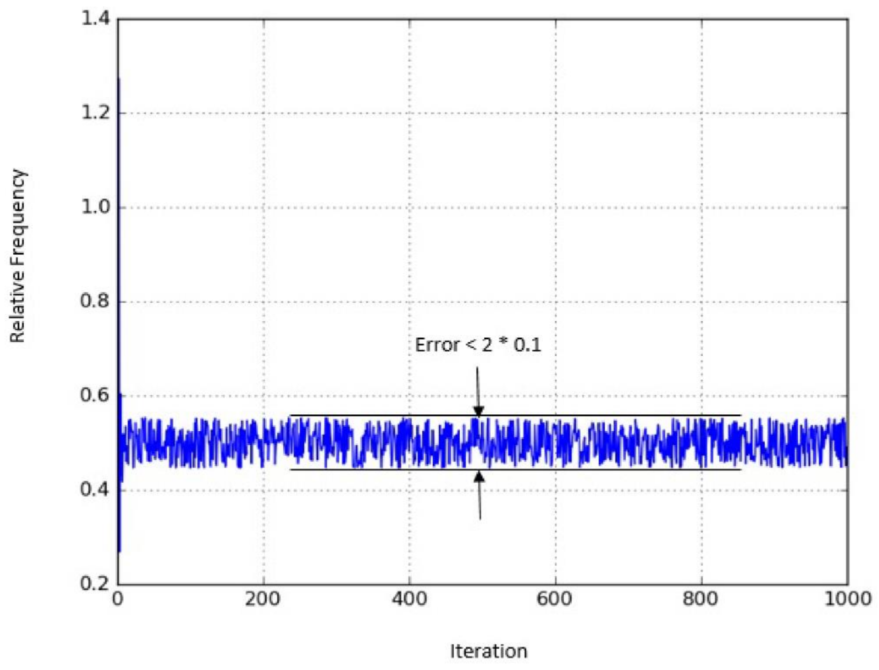


Figure 5-1 Deviation from the stationary measure for many iterations

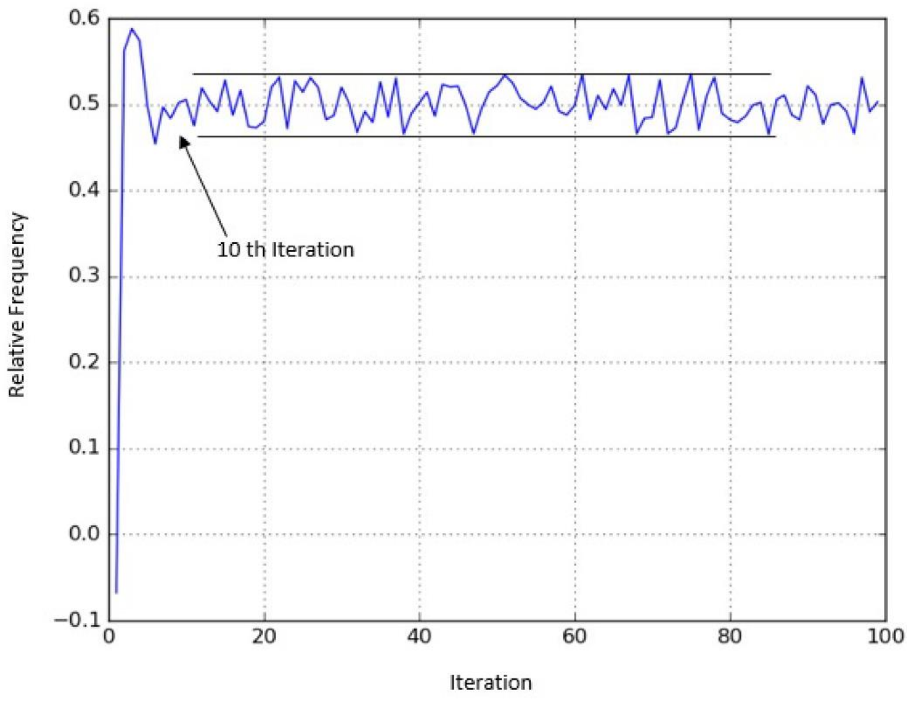


Figure 5-2 Very fast convergence to satisfy the error bound

We have analyzed the convergence rate of quasi-lumpable Markov Chains. This represents the case when the large state space can be aggregated into a smaller number of clusters, in which the states behave approximately the same way. Our main contribution is a bound on the rate at which the aggregated state distribution approaches its limit in such chains. We have also demonstrated that in certain cases this can lead to a significantly accelerated convergence to an approximate estimation of the measure of subsets. The result can serve as a useful tool in the analysis of complex networks, when they have a clustering that approximately satisfies the conditions lumpability.

CHAPTER 6

CONCLUSION

In this dissertation, we first discus linear optimization methods which can helps Internet protocols work more efficiently. We use a duality theory to find a weight set that improve the routing protocols efficiencies. As a matter of fact, routing is the most important aspect of Internet Traffic Engineering. So, we focus on routing protocols and introduce a practical method that optimizes link metrics. Previous optimization methods suffer from practical issues but our method could be implemented with Routing Protocols that based on shortest paths. Our simulation results show significant improvement on network efficiency. Second, a new method is introduced for traffic distribution in virtual circuit switched networks which can be implemented in real networks. In this method, the input traffic of each session is distributed among the possible paths, in a manner that the total system cost is minimized at the same time as the average cost for each path is kept bounded below a required threshold level. This method is scalable as its operation is per session. It is analytically proven in this article that this algorithm converges under the assumptions that are feasible in real networks. The simulation results approve the effectiveness of the algorithm. The results obtained from the simulation are in line with the results obtained from analytical resolution of the convex optimization problem. We also investigates the question "how non-cooperative nodes in a network can create an efficient network?" We have studied the result of the selfish behavior of nodes, and compares it to the situation in which there is a central control unit in the network. Central control can force all nodes to use a predefined strategy in which the network utilization is optimum.

In chapter 4 we study Big Data solution for analyzing large scale networks. We introduced Apache Spark as a replacement for Hadoop MapReduce. Efficiency of Spark, as a result of in-memory processing, makes it a popular big data processing engine. It also has high usability, due to different programming language APIs. There is a large collection of algorithms that cannot be implemented using only one iteration of Map and Reduce functions. Notably, graph processing algorithms fall in this category. Apache Spark improves efficiency of implementing such algorithms using in-memory processing. Essentially, after one iteration of Map and Reduce, the results are ready, and available in memory for the next iteration. Spark can be 100 times faster than Hadoop MapReduce for machine learning algorithms, such logistic regression. A fundamental processing unit in Spark is RDD, instead of one path of Map and Reduce functions. RDD is an immutable distributed data set across the cluster which is resilient to data storage failure. Data processing algorithms can be implemented using transformations and actions on each RDD. Transformations will create series of RDDs. As a result of immutability, each one of them can be recalculated from the previous one. Graphx is a new component in Spark for implementing graph algorithms in a distributed environment. Graphx extends the Spark RDD by introducing a new graph abstraction in terms of a distributed dataset, attached to vertices and edges. In this chapter three important social network algorithms have been introduced using Graphx library in Apache Spark. The first one is finding PageRank in a social network, the second one is finding connected components, and the last but not least is the triangle counting algorithm. In all the three applications, we have illustrated the steps via appropriate examples.

In the last chapter we study approximation approach for analyzing network structure. We investigate unsplittable flow problems and introduce a novel practical approximation. The simplicity and efficiency is achieved by sacrificing a small part of the solution space. The sacrificed part only contains scenarios where some edges are very close to saturation. Therefore, the loss is not significant, since the almost saturated solutions are typically undesired in practical applications, such as network design. The approach constitutes a new avenue to approximation, in the sense that instead of giving up finding an exact solution, we rather restrict the search space to a (slightly) smaller one. When, however, the algorithm finds any solution, which happens with high probability, then it is an exact (not just approximate) solution.

At the end we have analyzed the convergence rate of quasi-lumpable Markov Chains. This represents the case when the large state space can be aggregated into a smaller number of clusters, in which the states behave approximately the same way. Our main contribution is a bound on the rate at which the aggregated state distribution approaches its limit in such chains. We have also demonstrated that in certain cases this can lead to a significantly accelerated convergence to an approximate estimation of the measure of subsets. The result can serve as a useful tool in the analysis of complex networks, when they have a clustering that approximately satisfies the conditions lumpability.

REFERENCES

[1] Bernard Fortz and Mikkel Thorup, "Internet Traffic Engineering by Optimising OSPF Weights," in *Nineteenth annual joint conference of the IEEE computer and communications societies. Proceedings. IEEE*, 2000.

[2] Awduche, Macolm, Agogbua, O'Dell, McManus, "Requirements for Traffic Engineering Over MPLS," Internet Society, 1999.

[3] Awduche, "MPLS and traffic engineering in IP networks," *IEEE communications,* pp. 42-47, 1999.

[4] Ningning Hu, Li Erran Li, Zhuoqing Morley Mao, Peter Steenkiste, Jia Wang, "Locating Internet Bottlenecks: Algorithms, Measure-ments and Implications," in *ACM SIGCOMM*, 2004.

[5] Stephen Boyd and Lieven Vandedenberghe, Convex Optimization, Cambridge University Press, 2004.

[6] Yufei Wang, Zheng Wang, and Leah Zhang, "Internet Traffic Engineering without Full Mesh Overlaying," *INFOCOM,* 2001.

[7] Touraj Shabanian, Massoud Reza hashemi, Ahmad Askarian "Maximum Load Balancing with Optimized Link Metrics," *Journal of Software Engineering and Applications*, vol. 5.12, p. 14, 2013.

[8] Amir Nahir, Ariel Orda, and Ari Freund, "Topology Design of Communication Networks: A Game-Theoric Perspective," *IEEE/ACM TRANSACTIONS ON NETWORKING,* pp. 405-414, 2014.

[9] Zuyuan Fang and B. Bensaou, "Fair bandwidth sharing algorithms based on game theory frameworks for wireless ad-hoc networks," *INFOCOM,* 2004.

[10] Zhu Ji and Ray Liu, "Cognitive Radios for Dynamic Spectrum Access- Dynamic Spectrum Sharing: A Game Theoretical Overview," *IEEE Communications,* pp. 88-94, 2007.

[11] G. Debreu, "A Social Equilibrium Existence Theorem," in *Proceedings of the National Academy of Sciences of the United States of America*, 1952.

[12] Y.L. Varol and Doron Rotem, "An algorithm to generate all topological sorting arrangements," *The Computer Journal*, pp. 83-84, 1981.

[13] Frank H. Page Jr. and Myrna Wooders, "Strategic basins of attraction, the path dominance core, and network formation games," *Games and Economic Behavior,* pp. 462-487, 2009.

[14] Baruch Awerbuch, Amotz Bar-Noy, and Madan Gopal, "Approximate Distributed Bellman-Ford Algorithm," *IEEE TRANSACTIONS ON COMMUNICATIONS,* pp. 2515-2517, 1994.

[15] Ahmad Askarian, Andras Farago, "Designing Networks with Low Structural Congestion via Game Theory and Linear Programming," *Transactions on Networks and Communications,* vol. 3.1, p. 01, 2015.

[16] Touraj Shabanian, Ahmad Askarian, Massoud Reza hashemi, "Practical approach for traffic engineering with suboptimal OSPF routing," in *Electrical Engineering (ICEE), 2011 19th Iranian Conference on. IEEE*, 2011.

[17] Touraj Shabanian, Massoud Reza hashemi, Ahmad Askarian, Behnaz Omoomi, "An Optimal Traffic Distribution Method Supporting End-to-End Delay Bound," *Journal of Computing and Security,* vol. 1, p. 1, 2014.

[18] Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, Paul Spirakis, "The structure and complexity of Nash equilibria for a selfish routing game," *Theoretical Computer Science,* pp. 3305-3326, 2009.

[19] Seung Yeob Nam, Sunggon Kim, and Dan Keun Sung, "Measurement-based admission control at edge routers," *IEEE/ACM Transactions on Networking,* p. 410–423, 2008.

[20] K. W. Ross, Multiservice Loss Models for Broadband Telecommunication Networks, 1995.

[21] D. P. Bertsekas, Network Optimization:, Athena Scientific, 1998.

[22] D. P. Bertsekas, Nonlinear programming, Athena Scientific, 1999.

[23] Mehrdad Heydarzadeh, Mehrdad Nourani, Sarah Ostadabbas, "In-bed posture classification using deep autoencoders," in *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the. IEEE*, 2016.

[24] Mehrad Heydarzadeh, "Gear fault diagnosis using discrete wavelet transform and deep neural networks," in *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, 2016.

[25] Mehrdad Heydarzadeh, " An augmented reality platform for CABG surgery," in *Biomedical Circuits and Systems Conference (BioCAS)*, 2015.

[26] Mehrdad Heydarzadeh, "Gearbox Fault Diagnosis Using Power Spectral Analysis," in *Signal Processing Systems (SiPS),* 2016.

[27] Florian Bourse, Marc Lelarge, and Milan Vojnovic, "Balanced Graph Edge Partition," in *20th ACM SIGKDD international conference on Knowledge discovery and data mining,* New York, USA, 2014.

[28] Ahmad Askarian, Andras Farago, "Approximate State Aggregation in Markov Chain Monte Carlo Algorithms," *Technical Report UTDCS-19-16, Dept. of Computer Science, The Univ. of Texas at Dallas,* 2016.

[29] Mark Kramer, "The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits," *Elsevier* , 1984.

[30] Frank Pfeiffer, "On the complexity of the disjoint paths problem," *springer,* 1990.

[31] Andras Farago, "Speeeding Up Markov Chain Monte Carlo Algorithm," in *International Conference on Foundations of Computer Science*, Las Vegas, NV, USA, 2006.

[32] N. R. Seymour, "Graph Minors .XIII. The Disjoint Paths Problem," *Journal of Combinatorial Theory, Series B,* 1995.

[33] A. Z. Broder, "Optimal Construction of Edge-Disjoint Paths in Random Regular Graphs," *SIAM Journal on Computing 28.2,* 1998.

[34] Naveen Garg, "Primal-dual approximation algorithms for integral flow and multicut in trees," *Algorithmica 18, no. 1,* 1997.

[35] Yanatan Aumann, "An O (log k) approximate min-cut max-flow theorem and approximation algorithm," *SIAM Journal on Computing 27, no. 1* , 1998.

[36] David Kempe, "Maximizing the spread of influence through a social network," *In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining,* 2003.

[37] Ahmad Askarian, Andras Farago, "Utilizing Network Structure to Accelerate Markov Chain Monte Carlo Algorithms," *Algorithms*, vol. 9.3, p. 50, 2016.

[38] F. P. Kelly, "Loss networks," *The annals of applied probability,* 1991.

[39] Geraham Louth, "Computational complexity of loss networks," *Theoretical Computer Science 125, no. 1* , 1994.

[40] Mehrad Heydarzadeh, "A two-stage fault detection and isolation platform for industrial systems using residual evaluation," *IEEE Transactions on Instrumentation and Measurement,* vol. 65.10, pp. 2424-2432, 2016.

[41] Youngseok Lee, Biswanath Mukherej, "Traffic Engineering in Next-Generation Optical Networks," *IEEE Communications*, pp. 16-33, 2004.

[42] Marija Antic, Natasa Maksic, Petar Knezevic, and Aleksandra Smiljanic, "Two Phase Load balance Routing using OSPF," *IEEE Journal on selected areas in Communications,* pp. 2088-2096, 2010.

[43] Matthew O. Jackson and Alison Watts, "On the formation of interaction networks in social coordination games," *Games and Economic Behavior,* pp. 265-291, 2002.

[44] Robert Aumann and Roger Myerson, "Endogenous Formation of Links Between Players and of Coalitions: An Application of the Shapley Value," *Networks and Groups*, pp. 207-220, 2003.

[45] T. Roughgarden, "Selfish routing and the price of anarchy," *Cambridge,* 2005.

[46] Elias Koutsoupias, Christos Papadimitriou, "Worst-case equilibria," *Compuer Scienceview,* pp. 65-69, 2009.

[47] Ning Wang, Kin-Hon Ho, George Pavlou, and Michael P. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials,* 2008.

[48] Xavier Masip-Bruin, Marcelo Yannuzzi, Jordi Domingo-Pascual, Alexandre Fonte, Marılia Curado, Edmundo Monteiro, Fernando Kuipers, Piet Mieghem, Stefano Avallone, Giorgio Ventre,Pedro Aranda, Matthias Hollick, Ralf Steinmetz, Luigi Iannone, Keve Salamatian, "Research challenges in QoS routing," *Computer Communications,* pp. 563-581, 2006.

[49] Neda Moghim, Seyed Mostafa Safavi, and Massoud Reza Hashemi, "Performance evaluation of a new end-point admission control algorithm in ngn with improved network utilization," *International Journal of Innovative Computing, Information and Control,* 2010.

[50] Xavi Masip-Bruin, Marcelo Yannuzzi, René Serral-Gracià, Jordi Domingo-Pascual, José Enríquez-Gabeiras and María Ángeles Callejo, Michel Diaz, Florin Racaru, Giovanni Stea, Enzo Mingozzi, Andrzej Beben, Wojciech Burakowski, Edmundo Monteiro,and Luís Cordei, "The EuQoS System: A Solution for QoS," *IEEE Communications,* pp. 96-103, 2007.

[51] Mohamed E. M. Saad, Alberto Leon-Garcia, and Wei Yu, "Optimal network rate allocation under end-to-end quality-of-service requirements," *IEEE Transactions on Network and Service Management,* pp. 40-49, 2007.

[52] A. Farago, "On the Convergence Rate of Quasi Lumpable Markov Chains.," in *Proceedings of the 3rd European Performance Engineering Workshop*, Budapest, Hungary, 2006.

[53] Srikanth Kandula, Dina Katabi, Bruce S. Davie, and Anna Charny, "Walking the tightrope: responsive yet stable traffic engineering," in *SIGCOMM*, 2005.

[54] Wojciech Burakowski, Andrzej Beben, Halina Tarasiuk, Jaroslaw Sliwinski, Robert Janowski, Jordi Mongay Batalla, and Piotr Krawiec, "Provision of end-to-end qos in heterogeneous multi-domain networks," *annals of telecommunications,* p. 559–577, 2008.

[55] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney, "Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters," *Internet Mathematics,* pp. 271-276, 2009.

[56] Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis, "Efficient Triangle Counting in Large Graphs via Degree-based Vertex Partitioning," *Internet Mathematics,* pp. 161-185, 2012.

BIOGRAPHICAL SKETCH

Ahmad Askarian Received the Bachelor degree in electrical engineering from the Shahed University, Tehran, Iran, in 2008 and the M.E. degree from the Isfahan University of Technology (IUT), Isfahan, Iran, in 2010. He is currently pursuing the doctoral degree at The University of Texas at Dallas (UTD). Since 2012, he is a research assistant in the Scalable Network Engineering Techniques Laboratory, UTD, Dallas, US.

His research interest includes network optimization, graph theory and distributed processing systems.

CURRICULUM VITAE

Last update on March 2, 2017

Ahmad Askarian *May 20, 1984*

ahmad.askarian@gmail.com +1 469 236 7573 www.linkedin.com/in/ahmadaskarian

---

Experience

State Farm — Dallas, Texas

Big Data Engineer — *June '15 – present*

Data ingestion from RDBMS into HDFS and HBase. We designed and implemented a data stream-ing pipeline using Apache **Kafka**, Apache **Flume**, and **Spark Streaming**. Kafka producer and Flume interceptor have been written in Java and Spark Streaming in **S**cala.

Accounting system modernization. In this project we designed and implemented Apache Spark ap-plication to manage all financial transactions and create an online view using Apache **Hive/Impala** and **HiveContext** available in **SparkSQL**.

Designed and implement a machine learning model for automating financial auditing. In this project we used Apache Spark/MLlib, H2O and Microsoft/Mobius (**using C#**).

On-line fraud detection system for credit card transaction. In this project, using **decision tree** library in Spark Streaming, we implement near real time data analysis and incremental update of the machine learning model. **Lambda Architecture** is leveraged for this effort.

Data format conversion to Apache **Avro** and **Parquet**. In a distributed processing system, data serialization and movement between independent executors are a key for having high performance. In this regards, we convert high volume of CSV and XML file into Avro and Parquet.

Scheduling different Hadoop applications using Apache **oozie rest API**. We developed an oozie work-flow, which contains Spark, Java, FS and notification actions and acontroller using **Python** scripts.

.

UTDallas Computer Science Department — Dallas, Texas

Research Assistant — *Sep '12 – May '15*

MCMC approach for finding maximal clique in a social network. In this project we used

Graphx/Scala to implement a **probabilistic estimation algorithm** for community detection.

Convergence rate analysis of quasi-lumpable Markov Chains. In this project we study large state space, which could be aggregated into a smaller number of clusters, in which the states behave approximately the same. The simulation is implemented using **Python**

.

Skills

Machine Learning (**Spark/MLlib**, **TensorFlow**), Functional Programming (**Certified Scala Developer**), Object Oriented Programming (**Java, Python, C#**), Hadoop Ecosystem and scalable data management systems (**HDFS API, Yarn API, Oozie Rest API, Flume, Hive, Impala, Pig**)

Education

University of Texas at Dallas	Richardson, Texas
PhD in Telecommunications Engineering	*2012 – 2017*
Focused on Algorithm and Machine Learning	
Isfahan University of Technology	Isfahan, Iran
Master of Science in Electrical Engineering	*2009 – 2011*
Shahed University	Tehran, Iran
Bachelor Degree Electrical Engineering	*2004 – 2009*

List of publication

1. A. Askarian, R. Xu, and A. Farago, "Approximate State Aggregation in Markov Chain Monte Carlo Algorithms," Technical Report UTDCS-19-16, Dept. of Computer Science, The Univ. of Texas at Dallas, Dec. 2016.
2. A. Askarian, R. Xu, and A. Farago, "Finding Unsaturated Solutions of the Unsplittable Flow Problem Efficiently," Technical Report UTDCS-20-16, Dept. of Computer Science, The Univ. of Texas at Dallas, Dec. 2016.
3. A. Askarian, R. Xu, and A. Farago, "Scalable Application of the Apache Spark Distributed Memory System for Graph Algorithms," Technical Report UTDCS-22-16, Dept. of Computer Science, The Univ. of Texas at Dallas, Dec. 2016.
4. Askarian, Ahmad, and Andras Farago. "Designing Networks with Low Structural Congestion via Game Theory and Linear Programming." *Transactions on Networks and Communications* 3, no. 1 (2015): 01.
5. Shabanian, Touraj, Massoud Reza Hashemi, and Ahmad Askarian. "Maximum Load Balancing with Optimized Link Metrics." *Journal of Software Engineering and Applications* 5, no. 12 (2013): 14.
6. Askarian, Ahmad, and Andras Farago. "Designing Networks with Low Structural Congestion via Game Theory and Linear Programming." *Transactions on Networks and Communications* 3, no. 1

(2015): 01.

7. Askarian, Ahmad, Rupei Xu, and András Faragó. "Utilizing Network Structure to Accelerate Markov Chain Monte Carlo Algorithms." *Algorithms* 9, no. 3 (2016): 50.

8. Shabanian, Touraj, Massoud Reza Hashemi, Ahmad Askarian, and Behnaz Omoomi. "An Optimal Traffic Distribution Method Supporting End-to-End Delay Bound." *Journal of Computing and Security* 1, no. 1 (2014).

9. Shabanian, Touraj, Ahmad Askarian, and MasoudReza Hashemi. "Practical approach for traffic engineering with suboptimal OSPF routing." In *Electrical Engineering (ICEE), 2011 19th Iranian Conference on*, pp. 1-5. IEEE, 2011.