# Efficient Randomized Algorithms for Graph Theoretic Applications

## A Thesis

*submitted in partial fulfillment of the requirements for the award of the degree of*

## Doctor of Philosophy

by

**Kuldeep Sharma**
(Reg no: 950903051)

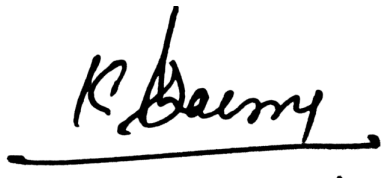
under the supervision of

**Prof. Deepak Garg**
**Computer Science and Engineering Department**
**Thapar University**

**Thapar University**
**Patiala, Punjab - 147004, India**
**June, 2016**

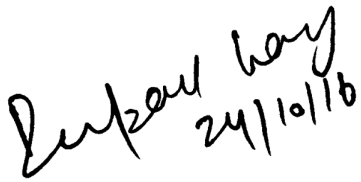
# Certificate

I hereby certify that the work which is being presented in this thesis entitled "**Efficient Randomized Algorithms for Graph Theoretic Applications**", in partial fulfillment of the requirement for the award of degree of **Doctor of Philosophy** submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Prof. Deepak Garg** and refers other research works which are duly listed in the reference section.
The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.



**Kuldeep Sharma**
Regn. No. 950903051

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.
**Prof. Deepak Garg**




Computer Science and Engineering Department
Bennett University
Greater Noida
Uttar Pradesh, INDIA.

# Acknowledgements

Firstly, I would like to express my deep gratitude to my supervisor, **Prof. Deepak Garg**, for his invaluable advice and encouragement at every step of my Ph.D program. Without his unfailing support and belief in me, this thesis would not have been possible. His contribution to this thesis goes well beyond his role as an academic supervisor and includes constant support on a personal level without which this journey might never have been completed. And for this, I am truly grateful to him. He is a great mentor for my life as well.

I would like to express my gratitude to Head & Associate Professor, Computer Science and Engineering Department, Thapar University, **Dr. Maninder Singh** and Senior Professor & Dean-RSP **Dr. O.P. Pandey** for their constant motivation and encouragement. I would like to thank **Dr. Anil Kumar Verma, Dr. Parteek Bhatia, Prof. Seema Bawa** and **Dr. Mahesh Sharma** for their support. I also wish to thank my research committee members and non-teaching staff of the institute for their help and support. I would also like to thank to my AOL teachers and friends from whom I learn the art of happiness and never give up approach.

I would like to give special acknowledgements to my fellow Ph.D scholars Dr. Gaurav Sharma, Dr. Suman Bala and Dr. Vishal Sharma for their support and motivation.

I would like to thank my friends Dr. Ravinder Kumar, Dr. Karun Verma, Dr. Sudhashu Gupta, Dr. Rattan Rana, Dr. Manoj Manuja and Er. Sahil Singla for their constant support, motivation and lots of love.

I would like to specially thank Mr. Vimal Tiwari for his valuable support and advice.

Finally, I thank almighty and would like to express my sincere and deep gratitude to my parents and family members for their love, encouragement and care. Without their support, I could not have achieved this milestone in my life.

**Kuldeep Sharma**

*... Dedicated to my respected Parents, my dear wife Neha and my loving son Atharv*

# Abstract

Randomization is currently one of the major approaches which is used to design algorithms. A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic. A randomized algorithm uses random input in the hope of achieving good performance in an average case. In addition to input, algorithm takes a source of random numbers and makes random choices during the execution. Its behavior may vary even on a fixed input.

This thesis provides comprehensive and integrated analysis of randomized algorithms over deterministic algorithms. We have also discussed the detailed literature of the multiobjective shortest path problem and multiobjective spanning tree problem. Deterministic polynomial time algorithms are extensively used for the shortest path and the minimum spanning tree problems.

In this thesis, we have presented a randomized algorithm to find Hamiltonian circuit in rectangular grid graphs with vertical size $m$ and horizontal size $n$. The algorithm firstly finds all the restricted edges in linear time and then constructs a Hamiltonian circuit by joining the sub-graphs. Algorithm uses random selection to construct the Hamiltonian cycle. The computational model is a unit cost random access machine with the restriction that only binary operation is allowed on edge selection.

Load balancing is extensively used in distributed network applications to decrease the response times. We have also discussed the backend load balancing (processor-to-dispatcher). Many algorithms are devised for front end load balancing e.g. JSQ, SQ(d) and JIQ. Join-Idle-Queue(JIQ) goes well in a distributed environment like cloud computing. In JIQ approach, at the backend, a processor joins the queue on either random or sampled basis. In both the cases, I-Queue of any dispatcher might remain empty, resulting in degrading the performance. After finishing the job, the processor should join the dispatcher whose I-Queue is empty. To achieve this, we have used a dequeue to track the dispatcher with empty I-Queue. As the processor finishes the current job and reaches the idle state, it should refer the dequeue and join the dispatcher whose I-Queue is empty.

***Keywords:*** Hamiltonian Cycle, Grid Graphs, Randomized Algorithm, Load Balancing, Distributed Networks, Secondary Load Balancing, Backend Load Balacing, Randomized Algorithm, Shortest Path, Minimum Spanning Tree, Multiobjective Shortest Path, Multiobjective Spanning Tree, Join-Idle-Queue.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Everyone in the theory of computing community is well acquainted with the concept of randomization. It is not an exaggeration to say that randomization is currently one of the major approaches to algorithm design. A randomized algorithm is an algorithm which typically uses random input in the hope of achieving good performance in an *average case*. Formally, the algorithm's performance will be a random variable determined by the random inputs with good expected value. The *worst case* is typically so unlikely to occur that it can be ignored. Consider the problem of finding an $x$ in an array of $n$ elements, given that half are $x's$ and the other half are $y's$. The obvious approach is to look at each element of the array but this would take very long ($\frac{n}{2}$ operations) if the array were ordered as $y's$ first followed by $x's$. There is a similar drawback with checking in the reverse order or checking every second element. In fact, with any strategy in which the order of the elements to be checked is fixed i.e. a deterministic algorithm (in case of deterministic algorithm $n$ operations are needed), we cannot guarantee that the algorithm will complete quickly for all possible inputs. On the other hand, if we need to check array elements at random, then we will quickly find an $x$ with high probability whatever the input is.

In the above example, the randomized algorithm always outputs the correct answer; it is just that there is a small probability of taking long to execute. At times, we want an algorithm which always completes quickly but allows a small probability of error.

## Classification of Randomized Algorithms

Two types of randomized Algorithms:

1. Las Vegas
2. Monte Carlo

Monte Carlo method is an approximation of the solution to the problems of computational mathematics by using random processes for each such problem with the parameters of the process equal to the solution of the problem. The method can guarantee that the error of Monte Carlo approximation is smaller than a given value with a certain probability. One can control the accuracy of this solution in terms of the probability error.

The Las Vegas method is the randomized method which also uses random variable or random processes but it always produces the correct result (not an approximation). The only variant is that its running time might change between executions [110]. A typical example is the well-known Quick sort method. Usually, Monte Carlo method reduces problem to the approximate calculation of mathematical expectations. Observe that any Las Vegas Algorithm can be converted into a Monte Carlo Algorithm, by having it output an arbitrary answer if it fails to complete within a specified time[109]. In addition to its input data, the randomized algorithm receives a stream of random bits that it can use for the purpose of making random choices. Even for a predetermined input, diverse runs of a randomized algorithm may give altered results. As a consequence, it is inevitable that a description of the properties of a randomized algorithm will engage probabilistic statements. For example, even when the input is preset, the execution time of a randomized algorithm is a random variable. Behavior of randomized algorithm varies from one execution to another even with a fixed input. Random variable is a function. For instance, let random variable $Y_1$ represents the result of the first die, $Y_2$ represents

the result of the second die, and $Y = Y_1 + Y_2$ represents the sum of the two. We could then ask: what is the probability that $Y = 7$ ? One property of a random variable we often care about is its expectation. For a discrete random variable $Y$ over sample space $S$, the expected value of $Y$ is:

$$E[Y] = \sum_{e \in s} P(e)Y(e) \tag{1.1}$$

Randomized algorithms are tools in computational number theory. Growth has been fuelled by the two major benefits of randomization - simplicity and speed. Numerous applications found that randomized algorithm is either the fastest algorithm available or the simplest and in most of the cases, it is both. In the analysis of a randomized algorithm, which establishes bounds on the expected value of a performance measure (e.g. the running time of the algorithm) that are valid for every input, the distribution of the performance measure is on the random choices made by the algorithm based on the random bits provided.

### 1.0.1 Complexity Analysis

Randomized algorithm $A$ may produce different results on a fixed input $x$, the output of a result $y$ is considered an (random) event. The probability that $A$ outputs $y$ for an input $x$, $P(A(x) - y)$, is the sum of all $P_{A,x}$, where C outputs y. Obliviously, the aim of the randomized algorithm designer is to achieve high $P(A(x) = y)$ if $Y$ is the correct output for the input $x$. Different runs of $A$ on an input $x$ may also have different lengths, i.e., they may have different costs. So, the time complexity becomes a random variable. Let $Time(c)$[10] be the time complexity of the run $C$ of $A$ on $x$, then the expected time complexity of $A$ on $x$ is

- $Exp - Time_A(x) = E[Time] = \sum_C P_{A,x}(C).Time(C)$,

  where the sum is taken over all runs $C$ of $A$ on $x$[11]. If one would like to consider $Exp-Time_A$ as a function of the input size, then one uses the worst case approach, i.e., the expected time complexity of $A$ is:

- $Exp-Time_A(n) = max\{Exp-Time_A(x)|x$ is an input of size $n\}$ for every $n \in \mathbb{N}$.

  It is often not easy to analyze $Exp - Time_A(n)$ for a given randomized algorithm $A$. To overcome this difficulty, one also uses the worst case approach from the beginning. This means:

- $Time_A(x) = max\{$ Time$(C)|C$ is a run of $A$ on $x\}$.

  Then the *worst case* time complexity of $A$ is :

- $Time_A(n) = max\ \{$ Time $A(x)|x$ is an input of size $n\}$.

  This definition may be misleading in some cases. This is because randomized algorithms may allow infinite runs provided that they occur with a reasonably small probability on any given input.

Randomized algorithms are better than deterministic ones. Its simplest example is Las Vegas Quick Sort algorithm.

**Algorithm 1.1** Randomized Quick Sort (RQS)

1: Input: $S = a_1, \ldots, a_n$
2: Choose an $i \in 1, \ldots, n$ uniformly at random
3: **if** $n = 1$ output(S) **then**
4: $\quad S \leq \{b \in S|b < a_i\}$
5: **else**
6: $\quad S \leq \{b \in S|b = a_i\}$
7: **end if**
8: $S \leq \{b \in S|b>a_i\}$
9: Recursively sort $S<$ and $S>$
10: Output: RQS(S) S=, RQS(S >)

In deterministic algorithm, the worst case complexity is $O(n^2)$ whereas in randomized algorithm, it is $O(n \log n)$ [2].

## 1.1 Design Paradigms of Randomized Algorithms

### 1.1.1 Abundance of Witnesses

Time and again, a computational problem required a witness or a certificate that capably authenticate a hypothesis. Say, a number $x$ has certain property i.e. $x$ is prime. To justify this property, it needs proof (witness) but for many problems, the witness lies in a search space that is too large to be searched exhaustively. However, if the search space were to contain a relatively large number of witnesses, a randomly chosen element is likely to be a witness. Further, autonomous repetition of the sampling reduces the probability that a witness is not found on any of the repetition. The most prominent examples of this phenomenon occur in number theory. Indeed, the problem of testing a given integer for primality has no known deterministic polynomial-time algorithm. There are, however, several randomized polynomial-time algorithms [3] that will correctly perform this test with high probability on any input.

### 1.1.2 Fingerprinting and Hashing

A fingerprint is an image of element mapping into another. Fingerprints obtained via random mappings have many useful properties. For example, in pattern-matching applications, it can be shown that two strings are likely to be identical, if their fingerprints are identical. Comparing the short fingerprints is considerably faster than comparing the strings themselves [4]. Another example is hashing, where the elements of a set $S$ are

stored in a table of size linear with the guarantee that the expected number of elements in $S$ mapped to a given location in the table is $O(1)$[5]. This leads to efficient schemes for deciding membership in $S$. Random fingerprints have found a variety of applications in generating pseudorandom numbers and complexity theory (for instance, the verification of algebraic identities)[6].

### 1.1.3 Foiling an Adversary

In the classical worst-case analysis of deterministic algorithms, a lower bound is established on the running time of algorithms by postulating an *adversary* that constructs an input on which the algorithm fares poorly. The input thus constructed may be different for each deterministic algorithm. With a game-theoretic interpretation of the relationship between an algorithm and an adversary, we can view a randomized algorithm as a probability distribution on a set of deterministic algorithms. This observation underlies Yao's adaptation of **Von Neumann's** *Minimax Theorem* in game theory into a technique for establishing limits on the performance improvements possible via the use of a randomized algorithm. Although the adversary may be able to construct an input that foils one (or a small fraction) of the deterministic algorithms in the set, it may be impossible to devise a single input that is likely to defeat a randomly chosen algorithm [7]. For example, consider a uniform binary $AND - OR$ tree with $n$ leaves. Any deterministic algorithm that evaluates such a tree can be forced to read the Boolean values at every one of the $n$ leaves. However, there is a simple randomized algorithm for which the expected number of leaves read on any input is $O(n^{0.794})$ [8].

### 1.1.4 Random Sampling

A pervasive theme in randomized algorithms is the idea that a small random sample from a population is representative of the population as a whole. Because computations involving small samples are inexpensive, their properties can be used to guide the computations of an algorithm attempting to determine some feature of the entire population. For instance, a simple randomized algorithm [9] based on sampling finds the $k^{th}$ largest of $n$ elements in $1.5n + O(n)$ comparison steps with high probability. In contrast, it is known that any deterministic algorithm must make at least $2n$ comparisons in the worst case.

### 1.1.5 Rapidly Mixing Markov Chains

Markov chains are probability models for trials of random experiments of great variety. Their defining characteristic is that they allow us to consider situations where the future evolution of the process of interest depends on where it is at present but not on how it got there. This contrasts with the independent trial model we have measured in the law of large numbers and the central limit theorem. For independent trial processes, the possible outcomes of each trial of the experiment are the same and occur with the same probability. Furthermore, what happens on any trial is not affected by what happens on another trial. With Markov chain models, we can generalize this to the extent that we allow the future to depend on the present [10]. In counting problems, the goal is to determine the number of combinatorial objects with a specified property. When the space of objects is large, an appealing solution is the use of the Monte Carlo approach of determining the number of desired objects in a random sample of the entire space. In number of cases, it can be shown that picking a uniform random sample is as difficult

as the counting problem itself. A particularly successful technique for dealing with such problems is to generate near uniform random samples by defining a Markov chain on the elements of the population and showing that a short random walk using this Markov chain is likely to sample the population uniformly. This method is at the core of a number of algorithms used in statistical physics. Examples include algorithms for estimating the number of perfect matchings in a graph [11][12].

# Chapter 2

# Literature Review

Myriad researchers have been fascinated by graph theory. We are concentrating on Shortest Path (SP) and Minimum Spanning Tree (MST) algorithms. A variety of algorithms are available for both the problems. Deterministic polynomial time algorithms are extensively discussed in literature. It is not an amplification to say that randomization is at present one of the major methodologies for algorithm design. In the course of last two decades, randomized algorithms have had good growth and went on from being a tool in computational theory to finding applications in many types of algorithms. Simplicity and speed, the dominant facts of randomization have necessitated this growth. Randomized algorithms got limelight due to the paper by Michael Rabin's *"Probabilistic Algorithms"*[13]. This admirable paper on probabilistic algorithms features algorithms for Primality Testing and Nearest Neighbors. Solovay and Strassen (in 1977)[3] have given the probabilistic algorithms for Primality Testing, which is another remarkable step in the field. In 1991, Karp's survey paper *"An Introduction to Randomized Algorithms"* established the existence of randomized algorithms[12]. Examples from major areas like sorting, searching, algebra, number theory, graph theory, pattern matching, selection, computational geometry, combinatorial enumeration, parallel and distributed computation have been included in that article and also proved that randomization was considerably profitable for the average case performance measure. This chapter deals with the investigation of in-practice algorithms for SP and MST (both deterministic and

randomized). A distinguishing aspect of our survey is the classification we have presented for each problem. To assimilate randomized algorithm, Quicksort is the best example. Quicksort behaves marvelously if the list of numbers to be sorted is in random order. Conversely, Quicksort behaves worse if the input is already sorted. One can devise randomized Quicksort as a two - step procedure. In the first step, the input arrangement to be sorted is randomly permuted. The deterministic Quicksort algorithm is then applied to the resulting sequence. Although, the input randomization step can be performed in linear time, but in practice, it is habitually more efficient to simply pick the pivot element randomly. The expected execution time of Quicksort (when all splitters are chosen at random) is $2n\ln(n) + O(n)$. This brief discussion narrates impartially well with the ordinary information-theoretic lower bound $n\log_2(n)$ for the amount of comparisons required to $\sqrt{n}$ items.

For the detailed study, refer *Algorithm 1.1.* We have examined deterministic as well as randomized algorithm for:

1. Shortest Path (SP)
2. Minimum Spanning Tree (MST)

In section 2.1, we have talked about deterministic as well as randomized algorithms of shortest path problems. Section 2.2 contains the literature and benchmarks of multi objective shortest path problem. In section 2.3, we have discussed the minimum spanning tree problem and their algorithms. Section 2.4 throws light on the Multiobjective Spanning Tree problem and major milestones in the field. Section 2.5 summarizes this chapter.

## 2.1 Shortest Path (SP)

Shortest path problem has been scrutinized by a large number of researchers. Therefore, shortest path algorithms have been worked over more painstakingly than any other algorithm in graph theory. More than 100 research papers and dozens of algorithms have been proposed. Several of these algorithms are working well in general, few are appropriate for a particular application or structure, whereas some are only minor modifications of earlier algorithms.

- **Bellman-Ford Algorithm**: This algorithm solves the single source shortest path problem (where weights may be negative). In this algorithm, source vertex is initialized as 0 and rest with $\infty$. This algorithm needs $V - 1$ (V total number of vertices) passes over all edges relaxing or updating the distance to the terminus of each edge. Finally, it checks each edge again to determine whether the negative weight cycle exists or not.

- **Dijkstra's Algorithm (with list)**: Dijkstra's algorithm is a graph search algorithm that cracks the single-source shortest path problem for a graph with non-negative edge path costs, constructing a shortest path tree. This algorithm is time and again used in routing and as a subroutine in other graph algorithms.

- **Dijkstra's algorithm (with Fibonacci heap)**: The original Dijkstra's algorithm does not use a min-priority queue and runs in $O|V|^2$ times. The implementation based on a min-priority queue as used by a Fibonacci heap and running in $O(|E| + |V| \log |V|)$ is due to Fredman and Tarjan. This is asymptotically the fastest acknowledged single source shortest path algorithm for arbitrary directed graphs with unbounded non-negative weights[14]. $E$ & $V$ are edges and vertices of a graph respectively.

- **Floyd-Warshall Algorithm**: The Floyd-Warshall algorithm is devised by Floyd [15] based on the theorem of Warshall [16] "How to figure the transitive closure of Boolean matrices?". The Floyd-Warshall algorithm is known as a graph analysis algorithm. Using this algorithm, shortest path can be obtained for a weighted graph even if the edges have negative weights. The transitive closure of a relation R can also be found. The lengths of the shortest paths between all pairs of vertices can be determined by single execution, though it does not return information of the paths themselves.

- **Johnson's Algorithm**: Johnson proposed an algorithm which is better than the Floyd-Warshall's algorithm [17]. This algorithm is capable to solve APSP in directed and sparse weighted graph. In initial step, a new node is added with zero weight edges from this to all other nodes. Then Bellman-Ford algorithm is used to inspect negative weight cycles. It also examines $h(v)$, the least weight of a path from the new node to node $v$. Then it re-weights the edges using the nodes $h(v)$ values. In conclusion, it executes Dijkstra's algorithm for each node. It stores the measured least weight to other nodes, re-weighted using the nodes $h(v)$ values, as the concluding weight.

Shortest path can be obtained using matrix multiplication and all pair shortest path problem can be solved using fast matrix multiplication, another league of researchers goes with these ideas. Michael Fredman devises an algorithm which computes in $O(V^3(\lg\lg V/\lg V)^{\frac{1}{3}})$ time [18]. This result can be obtained by using $O(V^{\frac{5}{2}})$ comparisons amongst sum of edge weights. The Fredman's algorithm is somewhat superior to the running time of the Floyd-Warshall algorithm. If $A = a_{ij}$ is an $m \times n$ matrix and $B = b_{jk}$ is an $n \times p$ matrix, then

$C = AB$ is $m \times p$ matrix $C = c_{ik}$ .

$$C_{ik} = \sum_{j=1}^{n} A_{ij} B_{jk}$$

for $i = 1, 2, ..., m$ and $k = 1, 2, ..., p$. Normally, it computes in $n^3$ and $n^2(n-1)$ additions, thus worst case complexity is $O(n^3)$. Then Strassen used diverse approach which necessitates only 7 recursive multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices and $O(n^2)$ subtractions and scalar additions, yielding the recurrence:

$$\begin{aligned} T(n) &= 7T(\frac{n}{2}) + O(n^2) \\ &= O(n^{\log 7}) \\ &= O(n^{2.81}) \end{aligned}$$

Coppersmith and Winogard [19] have devised an asymptotically efficient algorithm for multiplying $n \times n$ matrices with running time of $O(n^{2.376})$. Let $O(n^{\omega})$ be the running time of the fastest algorithm for product of square matrices. Galil, Margalit and Seidel [20] came up with algorithms that resolve the APSP for undirected, un-weighted graphs in $(V^{\omega} p(V))$ time. Here $p(n)$ denotes a specific function that is poly-logarithmically bounded in $n$. These algorithms are comparatively faster than the $O(VE)$ time needed to perform $|V|$ for breadth-first searches specially in dense graphs. Different researchers have extended these results to devise algorithms for solving the APSP problem in undirected graphs, in which the edge weights are as follow $1, 2, \ldots, W($ integers in the range). The asymptotically fastest algorithm is given by Shoshan and Zwick [21] which runs in time $O(WV^{\omega} p(VM))$. Karger, Koller and Phillips [22] and independently McGeoch [23] have established improvements over Dijkstra algorithm $|V|$ time when $|E^*| = O(E)$. Devised

**Table 2.1:** Complexity of various Shortest Path algorithms

| **Algorithm** | **Time Complexity** |
|---|---|
| Bellman-Ford Algorithm | $O(VE)$ |
| Dijkstra's Algorithm (with list) | $O(V^2)$ |
| Dijkstra's algorithm (with binary heap) | $O((E+V)\log V)$ |
| Dijkstra's algorithm (with Fibonacci heap) | $O(E+V\log V)$ |
| Floyd-Warshall Algorithm | $O(\|V\|^3)$ |
| Johnson's Algorithm | $O(V^2\log V+VE)$ |

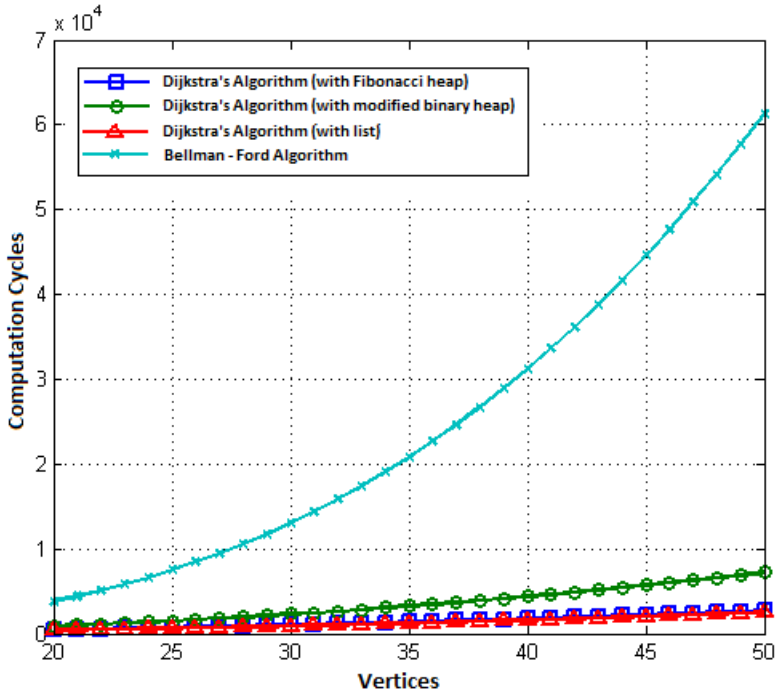



**Figure 2.1:** Comparison of single-pair shortest path algorithms

algorithm takes $O(VE^* + V^2 \log V)$ time. The set of edges in $E$ that take part to find shortest path is the time bound which depends on $E^*$.

Figure 2.1 and Figure 2.3 show the analysis of well-known deterministic algorithms for shortest path problem. In both figures, comparison is done on complete graphs, vertices (V) = $\{20, 21, 22, ....., 50\}$ are shown on x-axis whereas the number of cycles (in thousands) taken by each algorithm are shown on the y-axis. Figure 2.1 shows comparison for single pair shortest path problem. Dijkstra's Algorithm (with list) and Dijkstra's Algorithm (with Fibonacci heap) are almost taking the same number of cycles as curves

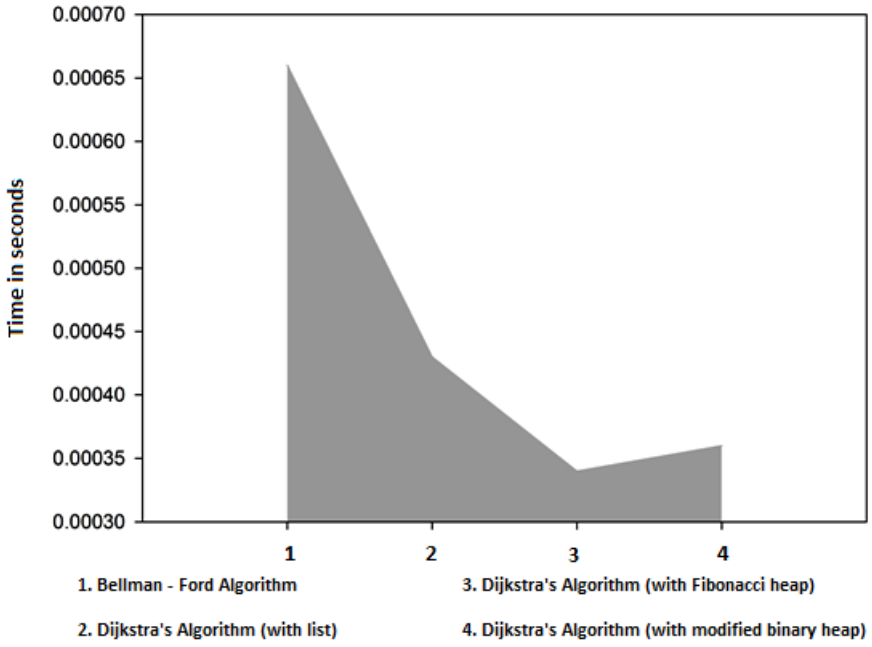



**Figure 2.2:** Execution time taken by single-pair shortest path algorithms

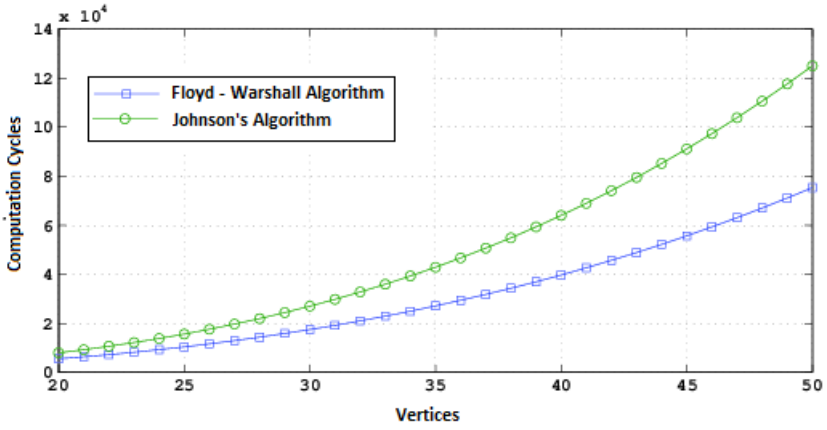



**Figure 2.3:** Comparison of all-pair shortest path algorithms

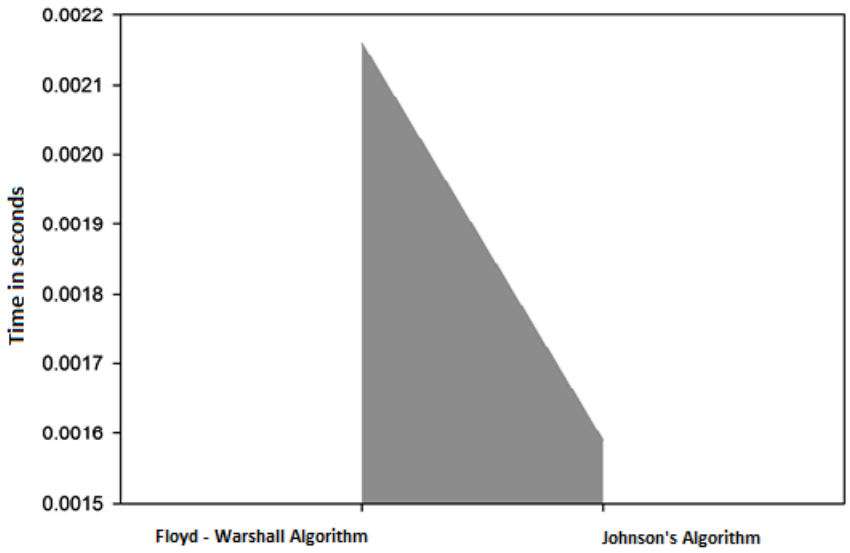



**Figure 2.4:** Execution time taken by all-pair shortest path algorithms

are overlapping in figure 2.1. Figure 2.3 shows analysis of all pair shortest path problems. Figure 2.2 and figure 2.4 show the execution time taken by various algorithms. It shows mean after one thousand executions of each algorithm.

### 2.1.1 Randomized Algorithm to find Shortest Path

**Problem Definition:** Let an undirected and unweighted graph $G = (V, E)$, where $V = 1, 2, ..., n$ and $A$ is the adjacency matrix of graph $G$. $D$ is a $n \times n$ distance matrix where $D_{ij} \notin I^- = L$ ($I^-$ is a collection of non-negative integers and L is the length of the shortest path between $i$ and $j$). Shortest path can be found by using the following steps:

1. All Pairs Distance (APD) is used to construct the distance matrix D.
2. All Pairs Shortest Paths (APSP) to find the shortest path between each pair of vertices.

Distance matrix for all pair shortest path problems of graph G (as stated for undirected, unweighted $n$-vertex graph) can be computed in time $O(M(n)logn)$, where $M(n)$ is the

time taken by the product of computing time for $n \times n$ matrices (of small integers)[24]. Seidel uses $O(n^{2.376})$ time for matrix multiplication (this matrix multiplication algorithm is given by Coppersmith and Winogard). A randomized algorithm finds a shortest path between each pair of vertices. The devised algorithm matches APD and its expected running time as well.

#### 2.1.1.1 Algorithm APD

1. Assign the value $F \leftarrow A^2$.
2. Compute the matrix $T$ ( $n \times n$) $0-1$ matrix, where $T_{ij} = 1$ iff $i \neq j$ and ($A_{ij} = 1$ or $F_{ij} > 0$).
3. If the value of $T_{ij} = 1 \; \forall \; i \neq j$ then return $D = 2T - A$. // D is $n \times n$ matrix
4. $\hat{D}$=$APD(T)$.
5. Find the product of $\hat{D}\&A$ and store into matrix $Y$.
6. If $Y_{ij} \geq \hat{D}_{ij}F_{ii}$ then compute $2\hat{D}_{ij}$ else $2\hat{D}_{ij} - 1$ and store in matrix D.

$G_1(V, E_1)$ is the graph obtained from $G$ by placing an edge between every two vertices where $i \neq j \in V$ iff path-length is 1 or 2 in $G$. The matrix $T$ computed in APD algorithm is adjacency matrix. $O(M(n))$ is the time required to compute Matrix $F$. If we know $A$ and $F$, we can determine matrix $T$ in $O(n^2)$ time.

### 2.1.2 Boolean Product Witness Matrix (BPWM)

APD algorithm is extended to solve the APSP problem; this is where randomization evidences beneficial. BPWM algorithm is a Las Vegas algorithm which solves the problem with expected running time $O(MM(n) \log^2 n)$. Here extended APD will be employed to find the witnesses for the Boolean matrix multiplication.

Given two $n \times n$ Boolean matrices $A$ & $B$, a matrix $C = AB$ (product under Boolean matrix multiplication).

$$C_{ij} = \begin{cases} 1 & if \Sigma_k A_{jk} B_{kj} > 0 \\ 0 & Otherwise \end{cases}$$

$C_{ij} = 1$ iff there exist an integer $k$ such that $A_{ik} = 1 = B_{kj}$. Such an index is called a witness for the pair $(i, j)$. It can be observed that for each $(i, j)$ with $C_{ij} > 0$, the number of witnesses can be any from 1 to $n$. The objective is to compute one witness for each pair$(i, j)$ with $P_{ij} = 1$.

**BPWM Problem Definition:** Given any two $n \times n$ Boolean matrices $A$ and $B$ and their Boolean product matrix $P$. Compute a witness matrix $W$ where $W_{ij}$ stores a witness for$(i, j)$ if $P_{ij} = 1$ and stores 0 otherwise. If an integer $k$ is a witness for a pair $(i, j)$, it may be confirmed in $O(1)$. So witness for any pair$(i, j)$, if exists, can be found in $O(n)$ time. Make a simultaneous scan of $i^{th}$ row of $A$ and $j^{th}$ column of $B$ and find $k \in A_{ik} = B_{kj} = 1$. This deterministic algorithm solves BPWM in $O(n^3)$ time.

#### 2.1.2.1 Las Vegas algorithm for the BPWM

1. Compute Matrix $W = -AB$
2. For $t = 0, 1, ..., \lfloor \log n \rfloor$ do
   (a) Compute $r = 2^t$
   (b) Repeat $\lceil 3.77 \log n \rceil$times
      i. select random $R \subseteq 1, \ldots, n$ with $|R| = r$.
      ii. Determine $A^R$ and $B^R$ and return the product in $Z$.
      iii. $\forall (i, j)$ do

         If $Z_{ij}$ is witness and $W_{ij} < 0$ then $W_{ij} = Z_{ij}$.

3. For all$(i, j)$ do

   Apply brute force to find witness $W_{ij} \forall (i, j)$ if $W_{ij} < 0$.

The integer matrix multiplication of $A$ and $B$ bearing in mind their entries as 0 and 1, produces a matrix $C$ whose item $C_{ij}$ corresponds accurately to the number of witnesses for the Boolean matrix entry $P_{ij}$. If $A = B$ is the adjacency matrix of a graph $G$, then $P_{ij} = 1$ iff there exists a path of length-2 from $i$ to $j$ and $C_{ij}$ is the number of such paths. A witness $K$ for $P_{ij}$ is the in-between vertex on a length-2 path from $i$ to $j$, finding witnesses for Boolean matrix multiplication is closely related to the issue of extending the APD algorithm for finding the shortest paths. For $P_{ij}$, the number of witnesses for this entry has been determined to be $w$. We may find the number of witnesses $w$ by using integer matrix multiplication to compute $C = AB$ and looking at the entry $C_{ij}$. Assume that $w \geq 2$, since it is easy to find witness, otherwise. Let $r$ be an integer such that $\frac{n}{2} \geq wr \geq n$[1]. The set $R$ comprises a unique witness for $P_{ij}$. $\hat{A}$ is $\hat{A}_{ik} = kA_{ik}$, shows that the integer matrix multiplication of $\hat{A}$ and B produces a matrix that contains the witness for all entries in the matrix P that have a unique witness. In particular, if each entry of P has unique witness, then $W = \hat{A}B$ is a solution of BPWM problem. Suppose that $R$ is represented as an incidence vector that has $R_k = 1$ for $k \in R$ and $R_k = 0$ for $k \notin R$. $A^R$ be the matrix attained from $A$ where $A^R_{ik} = kR_kA_{ij}$ and $B^R$ be the matrix got by setting $B^R_{kj} = R_kB_{kj}$. $A^R \times B^R$ produce witnesses for all entries in $P$ that have unique witness in $R$. The probability that a random set $R$ of size $r$ has a unique witness for an entry in $P$ with **w** witnesses where $\frac{n}{2r} \leq w \leq \frac{n}{r}$. Repeating for $O(\log n)$ for $R$ shows that witnesses are not found for such entries in $P$ and missing ones can be found using brute-force[1]. $MM(n)$ time taken to multiply $n \times n$ matrices by step 1. $O(\log_2 n)$ are iterations of the innermost loop body step 1. $O(\log_2 n)$ are iterations of the innermost loop body in step 2. This proves that the brute-force computations in

Step 3 are not too expensive. For any non-zero $P_{ij}$, a witness is found in step 2 with probability at least $1 - \frac{1}{n}$. The expected number of witnesses remaining to be found at the start of step 3 is n and since each of these is then found by brute force in $O(n)$ time, it follows the expected cost of Step 3 is $O(n^2)$. There will be at least one iteration of the outer loop with a value **r** such that $\frac{n}{2} \leq wr \leq n$,$1 - \frac{1}{2e}$ is the at most probability that $R$ does not have a unique witness for $P_{ij}$. So the inner loop is repeated $3.77 \log n$ times, the probability that no witness is found for the entry before the end of step 2 is at most $(1 - \frac{1}{2e})^{3.77 \log n} \leq \frac{1}{n}$.

#### 2.1.2.2 Algorithm APSP

1. Compute the distance matrix $D = APD(A)$.
2. For $S = \{0, 1, 2\}$ do
   (a) compute $0 - 1$ matrix $D^{(s)}$ such that $D_kj^{(s)} = 1$ iff $D_{kj} + 1 \equiv S(3)$
   (b) compute the witness matrix $W(S) = BPWM(A, D^{(S)})$.
3. Compute successor matrix $S$ such that $S_{ij} = W_{ij}^{(D_{ij} mod 3)}$.

The above discussed APD and BPWM are used to solve the all pair shortest path problem. The sole concern with this strategy is that the entire operation must be repeated for the $n$ different values of $d$, directing to a super-cubic algorithm for APSP. A simple observation leads to a decrease in the number of witness matrix computations from $n$ down to 3.

## 2.2 Multiobjective Shortest Path Problem

In the previous segment,various shortest path algorithms have been discussed. In this section, we will spill the beans about some other phase of shortest path problems with multiple constraints or we may have multiple objectives. Generally, the issue of discov-

ering the shortest path from a predetermined starting node to another node has been considered in the structure of the single objective optimization. The intent is to decide the achievable path for which either the aggregate travel time or the aggregate distance is minimized. In numerous genuine applications, it is regularly found that a single objective function is not enough to address the matter. Formally, it is known as Multiobjective Shortest Path Problem (MSPP). MSPP has been remarkably studied by many researchers; it is an elongation of the traditional shortest path. For instance, the problem of finding optimal routes in communication systems includes minimizing delay while amplifying throughput or discovering productive paths in transportation domain such that at the same time travel cost, track length and travel time are minimized. The idea of streamlining in the MSPP or in a multiobjective problem is unique in relation to the single-objective optimization problem, wherein the errand is to get an answer that optimizes a single objective function. The assignment in a multiobjective problem is not to locate an ideal answer for every objective function, but to locate an ideal arrangement that at the same time optimizes all objectives. Multiobjective techniques for the evaluation of transportation planning alternatives firstly showed up in the mid 1960's. From that period onwards, there has been an increment in research in this field. In 1986, Current et al. [25] presented a review paper on multiobjective design of transportation networks, after a while the review was updated with routing problems [26]. These two are the extensive reviews in the field. In 1978, Aneja et al. [27] gave a parametric approach to the shortest path problem between two specified nodes. They have taken the extra constraint and objective and articulated a bicriteria linear program. The optimal answer to the original problem was indicated to be a peculiar sort of extreme point of the bicriteria problem. Algorithm was devised for obtaining such an extreme point. At each iteration of the algorithm, a shortest path that minimizes a positively weighted average of the two objectives was seen. It showed that one asks to work out, on the average, at most $n$

such problems, where $n$ is the number of variables, to obtain the desired non-dominated extreme points.

**Table 2.2:** Growth of Multiobjective Shortest Path Approaches

| **Author** | **Year** | **Approach** |
|---|---|---|
| Andrzej P. Wierzbick | 1977 | He presented $D-$ monotonicity and $D_e$ approximation properties of scalarizing functionals to solve a multi objective optimization problem like partial ordering or preordering induced by a cone D. A few methods for developing functionals with these properties are also talked about.[28] |
| Hansen | 1980 | In his article, few bicriterion path problems are defined and their computational complexity is also examined. Algorithms were devised for MAXMIN-MAXMIN and MINSUM-MAXMIN problems. A fully polynomial approximation scheme and a pseudo-polynomial for the MINSUM-MINSUM problem is also presented in the article.[29] |
| Climaco and Martins | 1982 | They presented an algorithm for bicriterion shortest path problem. The set of Pareto optimal paths is determined. Also one objective used for any Pareto optimal path cannot be upgraded without deteriorating the other one.[30] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Martins | 1984 | He presented two algorithms for the multicriteria shortest path problems. One is to generalize multiple labelling scheme (as proposed by Hansen) for bicriteria case, which proves that any pair of nondominated paths can be connected to nondominated paths. This outcome is the support of an algorithm that can be seen as a variation of the simplex method utilized in continuous linear multiobjective programming.[31] |
| Current et al. | 1987 | They introduced a median shortest path for a bicriterion path problem to minimize the total path length and the total travel time. Devised algorithm is suitable for the transportation network design problems where the trade-off between operator costs and user costs is significant. The presented approach is to identify non inferior solutions to the median shortest path problems and it incorporates a $K$ shortest path algorithm.[32] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Warburton | 1987 | He has devised a method for approximating the set of Pareto optimal paths in multiple-objective, shortest-path problems. Approximation techniques has been given in such a way that can estimate the Pareto optima to any required amount of accuracy and avoid speedily expanding computational and storage demands as problem size increments. It is also discussed that approximation methods can be useful to produce fully polynomial approximation schemes for a variety of NP-complete, single-objective problems.[33] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Batta and Chiu | 1988 | Optimized paths for routing an objectionable vehicle on a network embedded on a Euclidean plane have been determined. Demand points are discretely distributed at nodes and continuously distributed on straight-line links of the network. They have not included the probability of accidental leakage of hazardous material, a path that minimizes the weighted sum of lengths over which this vehicle is within a threshold distance $\lambda$ of population centers. By appropriately redefining link lengths, a shortest-path algorithm is used to solve the problem. Special properties of the objective function allow us to efficiently calculate the modified link lengths. The optimal routing strategy and the objective values were also discussed along with an economic interpretation of the case when $\lambda$ varies.[34] |
| Smith and Shier | 1989 | Their article presented various implementations of a general label correcting algorithm for bicriterion networks. Then the outcome is compared on networks of varying size and degrees of correlation between the two criteria. So the experimental findings give a direction to solve bicriterion problems and also throw light on the factors that impact the computational effort.[35] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Current et al. | 1990 | They have recommended a method to produce an approximation of the noninferior solution set of two objective shortest path problems. It assists the decision maker to decide on the preferred or best compromise solution from among the noninferior solutions. The authors proposed an interactive approach based on a NISE-like procedure to search for nondominated supported solutions and using auxiliary constrained shortest path problems to carry out the search inside the duality gaps.[36] |
| Carraway et al. | 1990 | They gave implementations of dynamic programming (DP) to multicriteria sequential decision problems, including the optimization of a multicriteria inclination function which are uncommon. In such cases, it is not difficult to disrupt the DPs monotonicity assumption, whereas Generalized DP guarantees optimality. A prototypical multicriteria DP problem is solved using this methodology. The problem is also named as a multicriteria version of the shortest path problem.[37] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Azvedo and Marins | 1991 | In this article, a ranking based path procedure algorithm is proposed for the determination of the set of nondominated paths in an acyclic network. It is also supported by the principle of optimality.[38] |
| Mote et al. | 1991 | They have devised an algorithm for bicriterion shortest path problems (BSP) in this article. Proposed approach first relaxes the integrality conditions and resolves a bicriterion network problem. It is then solved parametrically, exploiting properties aligned to the adjacent basis trees. Pareto-optimal paths are attained using a label correcting method. This article also presents the computational results comparing the *K-th* shortest path approach and the parametric approach to the label setting technique. The results show that the parametric approach is orders of magnitude faster than the *K-th* shortest path.[39] |
| Tung and Chew | 1992 | A refined approach for finding one Pareto-optimal path for each efficient objective vector is described. It is a generalization of an earlier devised approach.[40] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Fujimura | 1996 | They have described it as vital to deal with numerous factors, exemplified by path planning, where a mobile robot finds the path through the navigation. The navigated path is considered only if it fulfills multiple objectives, such as energy consumption, time and safety.[41] |
| Modesti and Sciomachen | 1998 | Their article presents the problem about finding Origin-Destination ($O-D$) shortest paths in urban multimodal transportation networks. They minimize multiple objectives- the overall cost, time and users' discommodity associated with the needed paths. They make use of an ad hoc utility function for weighing the arcs together with their cost and time and seeing at the same time the liking of the users associated to all the possible transportation modalities.[42] |
| Coutinho et al. | 1999 | They have offered an interactive approach to search for unsupported nondominated solutions based on a k-shortest path procedure.[43] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Murthy and Olson | 1999 | They have proposed a method for the bicriterion shortest path problem. Devised method takes the assumption of inherent utility function which is quasi-concave and non-increasing and network consists of non-negative, integer valued arc lengths. Pairwise comparisons of alternatives develop domination cones which are used to fathom a large number of Pareto-optimal solutions. It is able to converge to the optimal solution in a sensibly small number of pairwise comparisons, even for those problems with a large amount of Pareto-optimal solutions.[44] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Zitzler and Thiele | 1999 | In their article, they have ushered in the Strength Pareto Evolutionary Algorithms (SPEA) approach for multicriteria optimization where four multiobjective EAs are compared and they consider an extended 0/1 knapsack problem as a footing. The suggested approach has characteristics like storing nondominated solutions externally in a second, continuously updated population; assessing an individual's fitness depending on the number of external nondominated points; conserving population diversity using the Pareto dominance relationship; integrating a clustering technique in order to minimize the nondominated set without abolishing its features. SPEA can be very efficient in sampling from along the whole Pareto-optimal front and sharing away the generated solutions over the tradeoff surface.[45] |
| ANTUNES et al. | 1999 | They have offered an algorithmic approach to handle the routing in networks, a multiple objective shortest path problem clearly in prospect of different metrics and corresponding QoS requirements.[46] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Skriver and Andersen | 2000 | They have devised a fast algorithm for the bi-criterion shortest-path problem (BSP). By imposing domination conditions, they have cut the number of iterations to obtain all the Pareto optimal paths in the mesh. They have observed the previously devised algorithms for the BSP and devised a ranking based on their computational complexity. A program for generating random networks is also introduced.[47] |
| Muller and Weihe | 2001 | It is based on identification of key characteristics which are appraised along the train networks. The authors take scenarios like travel time, fare, and number of train changes on a randomized model. The quantity of Pareto optima on each visited node is limited by a small constant. [48] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Guerriero and Musmanno | 2001 | They have passed a class of labeling methods to produce the whole arrangement of Pareto-optimal path length vectors from starting node to every single other node in a multicriteria network. The proposed methods are upheld hypothetically by the rule of optimality and they are characterized on the basis of different innovative node and label selection procedures. Computational results indicate that the projected methods outpace all the other codes in various instances.[49] |
| Granat and Guerriero | 2003 | They have devised the reference point guided labeling algorithm which finds the Pareto-optimal shortest path. This article provided the interactive technique of examining multicriteria shortest path by the reference point approach.[50] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Rajeev and Nilanjan | 2003 | They have proposed Evolutionary Algorithms (EAs) to solve multi-criteria multi-constrained network design problems. A framework for a bi-criteria bi- constrained communication network topology design is submitted. A hybrid approach and EA are used together for this NP-hard problem. They have also compared the achieved results with the exhaustive search and branch exchange heuristics.[51] |
| Gandibleux et al. | 2006 | They introduced the expanded edition of the label setting algorithm devised by Martins in 1984. They have computed all the efficient paths from a given source vertex to all the other vertices of the network. The major changes are related to the dominance test and the procedure for identifying efficient paths in multi-objective shortest path problem.[52] |

Continued on next page

**Table 2.2 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Tsaggouris and Zaroliagis | 2009 | Fully polynomial-time approximation scheme (FPTAS) is provided for multiobjective shortest paths. Likewise, a novel method is discussed for finding FPTAS to any multiobjective optimization problem having non-linear objectives. Approximate solutions are proposed for multiobjective optimal path and non-additive shortest path, which have significant applications in traffic optimization and QoS routing. The path to find FPTAS to numerous realistic setups in transportation and communication networks is also talked about.[53] |

From the above table, we can see that the extensive work has been done in the field. Many researchers have devised algorithms to obtain effective solutions for bi-objective multicriteria shortest path problem. Different researchers have suggested several approaches to reach out the optimal solution. Some are problem specific and some are generalized.

## 2.3 Minimum Spanning Tree

Minimum Spanning Tree (MST) has a rich and long history. MST is most suitable for real life applications. It is an optimal answer for connecting locations with minimal cost,

travelling salesman problem in optimization, cable network, etc. In this part, we will discuss deterministic algorithms and randomized linear time algorithm to find minimum spanning trees. We will also discuss major breakthrough in the study. The leading algorithm to find MST was due to Czech scientist Otakar Borůvka in 1926. He invented this algorithm for economical electrical coverage of power line of Moravia. The algorithm was arguable and rediscovered by many scientists. At first, it was rediscovered by a civil engineer Choquet in 1938. Further it was amended by a league of anthropologists Florek, Zubrzycki, Lukaziewics, Stienhaus and Perkal in 1951 and again by a western computer scientist Sallin in early 1960s[54]. All used common methodology *"Add all the safe edges and recurse"*.

### 2.3.1 Borůvka's Algorithm

- Select the smallest adjacent edge for every vertex and edges are added to MST escaping cycles.
- Connected components are named as Super-vertices, they can be computed by contracting the graph on the edges added to the MST.
- Repeat this until one connected component(super-vertex) is left i.e. $n-1$ edges contracted.
- The outcome of the union of these edges is a minimum spanning tree.

Here each step of Borůvka's algorithm diminishes the quantity of vertices by a factor of at least two. The same process will be executed at most $O(\log n)$ times. Entire contraction can be done in $O(m)$ time on each iteration and in totality, it takes $O(m \log n)$ time to form an MST. MST was captivated by several researchers in the commencement of the computer age in 1950s. The two adequate and commonly used textbook algorithms for

MST are Prims algorithm and Kruskals algorithm, both of them cited Borůvkas paper. Moreover, Prim's algorithm was a rediscovery of Vojtech Jarnik. Computer scientist Robert Prim devised Prim's algorithm in 1957[55].

### 2.3.2 Prim's Algorithm

- Start from a random vertex, look at all edges connecting to the vertex and choose the one with the lowest weight and add this to the tree.
- Look at all edges associated with the tree, pick the one (edge should not be in the present MST) with the least weight and add to the tree.
- It develops till the tree spans all the vertices in the input graph.

Devised algorithm is greedy in nature at each step of the partial spanning tree. The tree is improved with an edge that is the smallest among all possible adjoining edges. Huge amount of time in this algorithm is spent to find the smallest edge. A straightforward method finds the smallest edge by searching the adjacency lists of the vertices in V; then each repetition costs $O(m)$ time accrue a total running time of $O(mn)$, by expending binary heaps this can be improved to $O(m \log n)$. By means of Fibonacci heaps, Prims algorithm runs in $O(m + n \log n)$ time. Joseph Kruskal gave an algorithm which is known as Kruskals algorithm in 1956[56].

### 2.3.3 Kruskal's Algorithm

- Initially, each vertex in the graph is processed as a separate tree and creates a forest.
- Sort all the edges in the graph. For each edge $(u, v)$ in sorted order, repeat if vertices $u$ and $v$ belong to two diverse trees. Then $add(u, v)$ to the forest, merging two trees

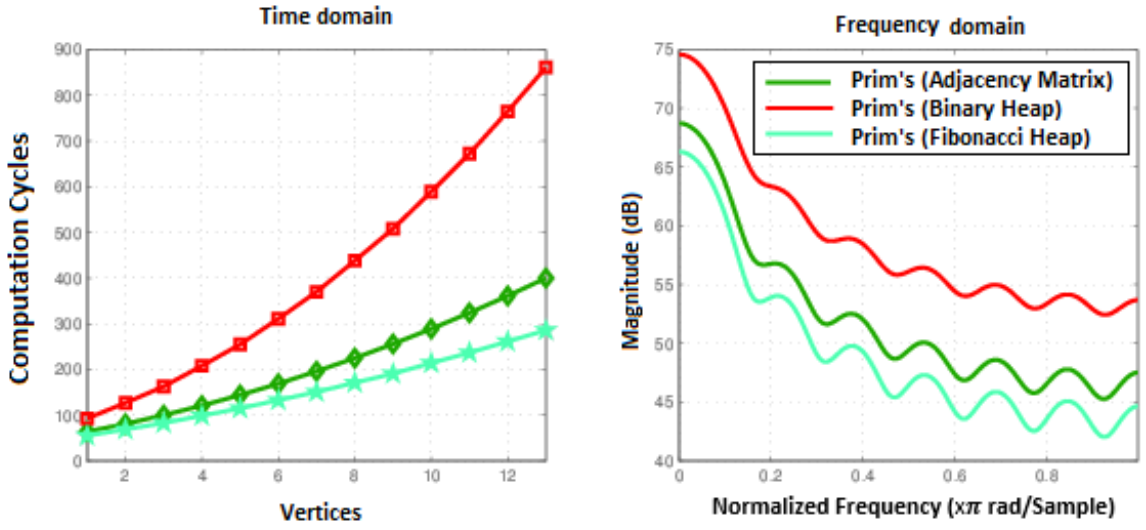



**Figure 2.5:** Comparison of Prim's algorithm

into a single one.

- Above process continues until all the edges have been processed. Sorting all edges to non-decreasing order takes $O(m \log m)$ time.

If the edges join two distinct trees in the forest, then the running time is $O(m\alpha(m,n))$, where $\alpha$ is the functional inverse of Ackermanns function. Hence the asymptotic running time algorithms are $O(m \log m)$, which is the same as $O(m \log n)$ since $\log m = \Theta(\log n)$ by observing that $m = O(n^2)$ and $m = \Omega(n)$. Textbook algorithms take $O(m \log n)$ to compute. Andrew Chi-Chih Yao , D Cheriton and R Tarjan independently made enhancements to $O(m \log \log n)$[57]. Fredman and Tarjan use Fibonacci heaps and scale down its complexity to $O(m\beta(m,n))$ where $\beta(m,n) = min\{i|log^n \leq \frac{m}{n}\}$ [58]. In the worst case, $m = O(n)$ and the running time is $O(m \log^* m)$. Gabow, Galil, Spencer and Tarjan have further lowered the complexity to $O(m \log \beta(m,n))$ [59]. How fast can one verify whether a given tree is minimum? By using path compression, Tarjan [60] gave a linear time algorithm. The linear number of comparisons can be used to verify the MST but with nonlinear overhead to decide which comparisons to make. This approach was

devised by Janos Komlos[61]. The first linear time MST verification algorithm, to discard edges that cannot be presented in MST, was given by Dixon, Rauch and Tarjan[62]. In continuation, a simple linear time verification algorithm was proposed by Valerie King [63]. All these methods use the fact that a spanning tree is a minimum spanning tree iff the weight of each non-tree edge $(u, v)$ is at least the weight of the heaviest edge in the path in the tree between $u$ and $v$. Bernard Chazelle [64] took a significant step towards an open problem, "Can we find MST in linear time?" His algorithm runs in $O(m\alpha(m, n))$ time where $\alpha$ is the functional inverse of Ackermanns function [65]. The key concept is to figure out sub-optimal independent sets in a non-greedy fashion and then progressively ameliorate them until the best solution is reached. An approximate priority queue called soft heap is in use to create a sub-optimal spanning tree whose equality is increasingly refined until an MST is finally produced [66]. "The complexity of the MST is equal to its decision-tree's complexity". This is established by Seth and Vijaya [67], [68], [69]. A deterministic comparison-based MST algorithm is devised by them. This algorithm computes in $O(T^*(m, n))$ time where $T^*(m, n)$ is the number of edge-weight comparisons required to shape the MST. The source of their algorithm's mysterious running time and its optimality is the use of recomputed MST decision trees whose exact depth is unknown but nonetheless provably optimal. An insignificant lower bound is $\Omega(m)$ and the best upper bound is $O(m\alpha(m, n))$ [64]. In 1994, Philip, Klein and Tarjan[70] proposed a randomized linear-time algorithm to construct MST in connected graph. The full version of the article was published in 1995 with Karger [71]. This was a modification of the Karger's Algorithm, who obtained an expected time bound of $O(m + n \log n)$. This algorithm uses the random sampling technique devised by Karger and verification algorithm proposed by Dixon [62]. The devised algorithm runs in $O(m)$ expected time. Both the cycle and the cut properties are used in this. To obtain an approximated result, it uses a randomly sampled fraction of edges. Then, by means of MST verification,

**Table 2.3:** Complexity of various algorithms available for MST

| **Algorithm** | **Data Structure Used** | **Time Complexity** |
|---|---|---|
| Borůvka Algorithm | Disjoint set union | $O(m \log n)$ |
| Kruskal(sorted) | Disjoint set union | $O(m\alpha(m, n))$ |
| Kruskal(not sorted) | Disjoint set union and heap | $O(m \log n)$ |
| | Binary heap | $O(m \log n)$ |
| Prim | d- Heap | $O(nd \log_d n + m \log_d n)$ |
| | F-Heap | $O(n \log n + m)$ |
| Yao | Heaps | $O(m \log \log n)$ |
| Cheriton and Tarjan | Doubly linked queue, disjoint set union | $O(m \log \log n)$ |
| Fredman and Tarjan | F-Heap, doubly linked queue | $O(m\beta(m, n))$ |
| Gabow et al. | F-Heap, disjoint set union | $O(m \log \beta(m, n))$ |
| Karger | Randomized | $O(n \log n + m)$ |
| Karger et al. | Randomized, recursion | $O(m)$ |

finds the misplaced edges and recalculates the true MST in linear expected time. Here randomization plays a vital role to eliminate edges that are guaranteed not to fit into the MST.

### 2.3.4 The Linear-Time MST Algorithm

Input: weighted, undirected graph G with V vertices and E edges.

Output: Minimum Spanning Forest $\delta$ for G.

1. Run Borůvkas Step 2 times.

   a) If there were originally n nodes, how many in reduced problem

2. Random Sampling

   a) Make a smaller subgraph by choosing each edge with probability 1/2.

   b) Recursively compute minimum spanning tree (or perhaps forest) $F$ on this reduced graph

   c) Use verification algorithm to eliminate any edges $(u, v)$ not in $F$ whose weight is

more than weight of $F(u, v)$.

3. Apply algorithm recursively to the remaining graph to compute a spanning tree $\delta$

   a) Knit together tree from step 1 with F to form spanning tree

Step 1: Run 2 Borůvka Steps it will return contracted Graph G1 and List of edges E1. In step 2(a) construct G2 by sampling 1/2 edges (G2 has at most 1/2 the edges of G0). In 2(b) recursively construct T2 for graph G2. Step 2(c) use T2 to identify edges E2 that cannot be in MST for G1 (G3 edges bounded by 1/2 nodes of G0). Finally in step 3 recursively construct T3 for graph G3 (Graph $G3 = G1 - E2$). Return T3 melded with edges E1 as final tree T4 (Tree T4 = T3 $\cup$ E1)[1].

## 2.4 Multiobjective Minimum Spanning Tree

The multiobjective minimum spanning tree issue is not just an extension of the MST from the single objective to multiple objectives. Most of the time, it is hard to find an ideal organization of the subject because of multiple objectives. More than one objective generally create conflict with each other. The actual result of the problem is a set of Pareto optimal solutions. This problem is commonly solved by making multiple objectives into a single objective since those polynomial-time MST algorithms are only effective to the MST with a single objective. On the other hand, only one single-point solution in the sense of Pareto optimality can be obtained, not to refer the difficulty in properly converting the multiple objectives into a single objective. The choice maker can possibly use the better one (Pareto optimal) point over the others depending on the situation. Consequently, it could be good to have (all) other promising Pareto optimal solutions.

**Table 2.4:** Growth of Multiobjective Minimum Spanning Tree

| Author | Year | Approach |
|---|---|---|
| Andrew W. Shogan | 1983 | His paper gives the way out to construct a minimal-cost spanning tree with multiple constraints. In a network, limited consumption of the scarce resources and the requirement that there exists a flow from the source satisfying the demands at the sinks without exceeding any arc capacity were the constraints taken care off. A branch and bound algorithm based on Lagrangian relaxation was used to solve the problem.[72] |

Continued on next page

**Table 2.4 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Hutson and Charles | 1989 | They have expanded the network design coverage problem. In this paper, maximal covering tree problems were shown to expand the applicability of the minimal spanning tree, which identified the minimal length connection of all nodes in a network. Maximal covering tree problem relaxes the constraint that all nodes must be connected. Instead, maximal covering tree problems identify the best choices for subtrees in the spanning tree network based on the relative benefits and costs of connecting nodes. Two-objective IP models were articulated and worked for the maximal direct covering the tree. One on spanning networks in which arcs currently exist and second for the general spanning tree graph.[73] |
| Hutson and Charles | 1993 | Minimum Cost Covering Subtree (MCCS) and Maximal Indirect Covering Subtree (MICS) problems using spanning tree were introduced. MCCS search for the minimum cost collection of arcs that form a subtree and satisfy covering constraints for nodes of the network. MICS selects that subtree which maximizes the demand within a distance standard of nodes of the subtree.[74] |

Continued on next page

**Table 2.4 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Hamacher and Ruhe | 1994 | Two different kinds of multiple objective minimum spanning tree issues characterized on a system with vectorial weights. First is to minimize the max of linear objective functions in set of spanning trees. Second is to find the efficient multi-criteria spanning trees. A ranking based algorithm is developed to resolve bicriterian spanning tree. Neighbourhood search is applied to choose a serial publication of solutions having the space between two continuous solutions.[75] |
| Melamed and Sigal | 1995 | A generic method is proposed for the reliable solution found by the linear complication and computational complexity is examined by performing an experiment for covering trees and assignment problems. [76] |

Continued on next page

**Table 2.4 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Andersen et al. | 1996 | They focused to compute the set of effective spanning trees for a given network where each arc carries two attributes. A new heuristic approach, adjacent search (new method) is introduced and neighbourhood search is discussed. In neighbourhood search, all spanning trees which are adjacent to at least one spanning tree in the current approximation set are taken into account. Adjacent search differs to neighbourhood search that only spanning trees which are adjacent to at least two spanning trees in the current approximation set are considered. It was established that the adjacent search is more effective for large problems.[77] |
| Melamed and Sigal | 1998 | Their paper discussed the solutions to the three criteria tree and assignment problems. It is discovered that the relative number of effective solutions found using linear convolution for two criteria is less than the three criteria.[78] |

Continued on next page

**Table 2.4 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Ramos et al. | 1998 | An algorithm is designed to establish the set of efficient trees for uniobjective problem of finding optimal spanning trees of a connected graph. Two processes are taken to solve; the foremost is the building up of all the spanning trees of such a connected graph and second is the building up of the perfect set of spanning trees involving minimum cost. [79] |
| Zhou and Gen | 1999 | Their article proposed genetic algorithm based approach to MST problem. Multiple Criteria Decision Making method and non-dominated sorting method is applied by adopting the Prüfer number as the tree encoding. The Genetic Algorithm search generates all Pareto Optimal resolutions which are dispersed all through Pareto frontier or concentrated towards the area near ideal point. The analysis depicted the efficiency of this method for the multi criteria Minimum spanning tree problem. [80] |
| Knowles and Corne | 2001 | They have given an evolutionary approach based on Knowles and Corne's PAES algorithm and also modified Prims iterated approach for the multiobjective MST. [81] |

Continued on next page

**Table 2.4 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Syarif et al. | 2002 | In their article, the problem of determining the least cost of the physical distribution flow in logistics system is discussed. They look at the logistic chain network problem framed by 0-1 mixed integer linear programming example. The suggested solution is a spanning tree based genetic algorithm. Repairing technique for the infeasible Prüfer number is designed to make it run for huge size problems. By comparing the experimental solutions, it is showcased that its effectiveness is more than the traditional algorithms. [82] |
| Steiner and Radzik | 2003 | They have measured a two-phase method to calculate the complete circle of solutions of the biobjective minimum spanning tree problem. In the first phase, the efficient solutions by expanding the convex hull of the possible space is figured. In the second phase, a k-best minimum spanning tree algorithm to find all nonextreme efficient solutions is applied. The "k-best" technique performs significantly better in their case.[83] |

Continued on next page

**Table 2.4 – continued from previous page**

| Author | Year | Approach |
|---|---|---|
| Perny and Spanjaard | 2005 | They have devised preference-based search algorithms for the preferred spanning tree problem and the preferred paths problem, both are the generalizations of their class. An axiom name independence is introduced for preference relations and several solutions for multicriteria spanning trees. [84] |
| FrankNeumann | 2004 2007 | Neumann has presented a runtime analysis of multi-objective evolutionary algorithm for minimal spanning tree. He also gave upper bounds on the expected time till the time evolutionary algorithm has produced a population, including for each extremal point of the Pareto front a corresponding spanning tree. These details are of special interest as they give a 2-approximation of the Pareto front. [85] [86] |
| Neumann and Witt | 2010 | They have examined multi-objective evolutionary algorithms (MOEAs) on multi-objective minimum spanning tree problem. They have revealed that stochastic search algorithms are able to compute MST in expected polynomial time.[87] |

The existing literature shows that the straightforward MST problem can be solved efficiently even in linear time, but the degree constrained and multiobjective versions are NP-hard.

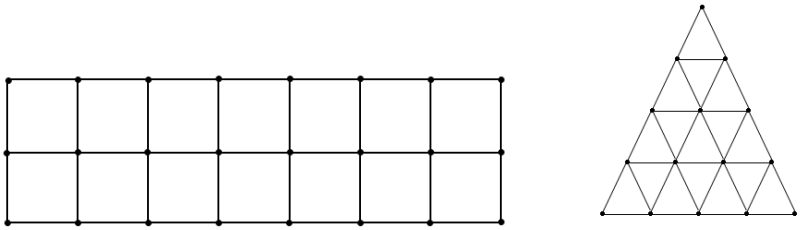
## 2.5 Summary

In this chapter we have seen that for such applications, randomized approach is far better than the deterministic approach and the same is true for many more applications in the field. Observations proved that by using randomization, one can solve MST in linear time ($O(m)$ with high probability). All Pair Shortest Path Problem using randomization can be solved in $O(MM(n) \log n)$, where $O(MM(n))$ is necessary time taken by $n$ x $n$ matrix. Different researchers have proposed various approaches to accomplish the optimal solution for multiobjective SP and MST. However, there is major gap where randomization can play an important part. Randomization is helpful where one has to take random decision to save computation so that the functional complexity may reduce.

# Chapter 3

# Randomized Algorithm to Find Hamiltonian Circuit in Rectangular Grid Graph

A Hamiltonian Circuit (Hamiltonian Cycle) is a circuit/cycle that passes through every single vertex of graph precisely once and finishes at the beginning. Graph having Hamiltonian circuit is called Hamiltonian graph. Hamiltonian path or Traceable path joins all the vertex precisely once. Hamiltonian cycle is named after *Sir Willam Rowan Hamilton (1805 -1865)* who devised a puzzle to discover Hamiltonian cycle along the edges of a dodecahedron such that every vertex is visited once and the ending point is the same as the starting point. Countless efforts have been made to portray the graphs which comprise Hamiltonian circuits. While providing the classifications in numerous special cases, none of these has led to an efficient algorithm for classifying such graphs in general. In fact, a league of researches have proved this problem to be "NP-Complete", indicating that no simple computational oriented classification is promising. For this motive, interest has moved to special cases with further restricted structure, for which such a classification may still be possible in grid graphs. A grid graph is a vertex-induced constrained sub graph of the infinite grid. It is a subset of a Graph G ($g(v) \subset G(v)$) together with edges whose both termination points are in this subset. There are many types of grid graphs like Rectangular grid graphs, Triangular grid graph as shown in figure 3.1 respectively.



(a) Rectangular grid graph (b) Triangular grid graph

**Figure 3.1:** Types of grid graphs.

The problem of determining Hamiltonian cycle in rectangular grid graph is considered here. Rectangular grid graph $G(m, n)$ is a two-dimensional graph with horizontal size $m$ and vertical size $n$. In other words, it can be treated as a two-dimensional rectangular grid graph with width $m$ and height $n$. Rectangular grid graphs were studied by Luccio and Mugania in 1978, while they were making an effort to solve the Hamiltonian path problem aimed at chess board [88]. A solid grid graph is a type of rectangular grid graph without holes. The study of Hamiltonian cycle in solid grid graphs came into existence due to Itai et al.[89]. In 1982, they provided evidence that finding Hamiltonian cycle in general grid graphs is NP-Complete and devised an algorithm for rectangular solid grid graph. In 1994, Afrati presented a linear time algorithm for restricted grids (a sub-class of solid grid graphs) which were fundamentally left-aligned stacks of decreasing-length grid stack[90]. Both methods discussed above work on general solid grid graphs. Further in 1997, Umas et al. contributed a polynomial time algorithm for the Hamiltonian cycle problem for solid grid graphs[91]. They used the cycle merging technique in which the 2-factor spanning subgraph (where all the vertices having degree two) was found and the makeover of the 2-factor was done, that reduced the number of components. They claimed that the algorithm can be applied to a generalization of solid grid graphs. In 2002, Chen et al. enhanced the algorithm presented by Itai et al. They decreased the number of partitioning steps from $O(m+n)$ to a constant[92]. They further claimed that the algorithm can be optimally parallelized to obtain a constant time parallel algorithm

on the parallel machine without need of inter-process communication in meshes. In 2011, Kohjerdi et al. devised a linear time algorithm for finding longest path between any two given vertices in a rectangular grid graph[93]. They presented that constructing a Hamiltonian circuit in rectangular grid graphs could be treated as a special case of finding the longest path between any given pair of distinct vertices $\{x, y\}$ in $G(m, n)$. Namely if $\{x, y\}$ are starting and ending vertices of the longest path, then the edge between $x$ and $y$ forms a Hamiltonian circuit of $G(m, n)$.

A randomized algorithm for determining Hamiltonian Circuit for rectangular grid graph has been presented in this chapter. It runs in $O(mn)$ time with high probability. Section 3.1 covers preliminaries on which devised algorithm is based upon. Section 3.2 presents Randomized algorithm for finding Hamiltonian Circuit for rectangular grid graph. Section 3.3 contains the analysis and discussion. Section 3.4 summarizes this chapter.

## 3.1 Foundation

In this section, some characterizations and related Lemmas are introduced and discussed, on which proposed algorithm is established.

### Definition - I

$G_{HC}$ is Hamiltonian Cycle of $G(m, n)$ iff number of Vertices (V) are equal to number of Edges (E) and degree of each vertex is 2 in a connected graph.

$$V = m \times n$$

In rectangular grid graph, total number of edges can be counted by the factor $E = 2(m \times n) - (m + n)$.

Total number of vertical edges in a rectangular grid graph will be :

$$m(n-1) \tag{3.1}$$

Total number of horizontal edges in a rectangular grid graph will be :

$$n(m-1) \tag{3.2}$$

on adding 3.1 and 3.2, total edges in a graph :

$$= m(n-1) + n(m-1)$$

$$= mn - m + nm - n$$

$$= 2mn - (m+n)$$

$$E = 2V - (m+n)$$

where $E$ and $V$ are total number of edges and vertices respectively. $d_i^\circ$ is used to represent the degree of vertex ,where $i$ is degree.

**Lemma 3.1.1.** *Determine the number of vertices with degree* $d_2^\circ, \eta = d_3^\circ, \beta = d_4^\circ$ *in* $G(m,n)$.

*Proof.* In rectangular grid graph, $\beta$ is the total number of vertices having $d^\circ = 4$.

Number of vertices except first and last row:

$$m - 2 \tag{3.3}$$

Number of vertices except first and last column:

$$n - 2 \tag{3.4}$$

by multiplying 3.1 and 3.4

$$\beta = (m - 2)(n - 2)$$

$\eta$ vertices will have $d^\circ = 3$ where:

$$\eta = V - (d_2^\circ + d_4^\circ)$$

$$\eta = V - (4 + d_4^\circ)$$

In the graph $G(m, n)$, 4 vertices (corner vertices) have degree 2. $\beta$ is the number of vertices which have $d_4^\circ$ where $\beta = ((m - 2) * (n - 2))$. $\eta$ is the number of vertices which have $d_3^\circ$ where $\eta = V - (\beta + 4)$. So for $G(8, 3)$, we have number of vertices with $d_2^\circ = 4$, $\eta = 14$ and $\beta = 6$, same is represented in figure 3.2. Vertices with $d_2^\circ$ are shown in red color, vertices with $d_3^\circ$ are shown in blue color and vertices with $d_4^\circ$ are shown in color green. □

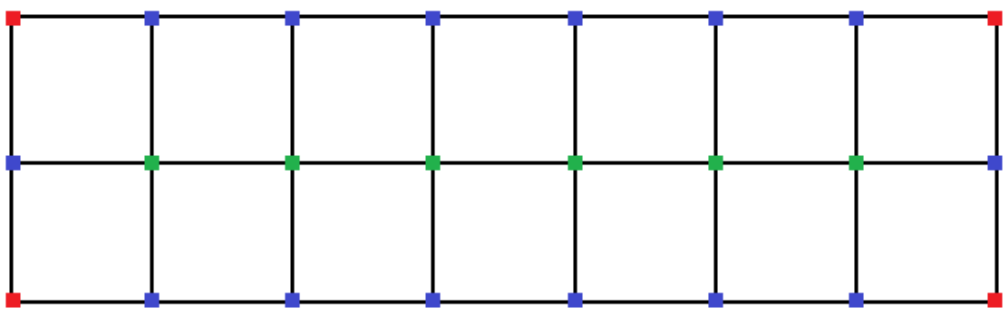


**Figure 3.2:** Rectangular Grid Graph $G(8, 3)$.

## Definition - II

An edge $\acute{E}$ is said to be **Dense Edge** if $\omega > 5(\omega = d_x^\circ + d_y^\circ)$ whereas $d_x^\circ$ and $d_y^\circ$ are degree of $(x, y)$ coordinate of $\acute{E}$ respectively. An edge with $\omega = 4$ is represented as **Restricted Edge**. In figure 3.2, the edges connected with red color vertices are considered as Restricted Edges. Figure 3.3 represents Dense Edge.

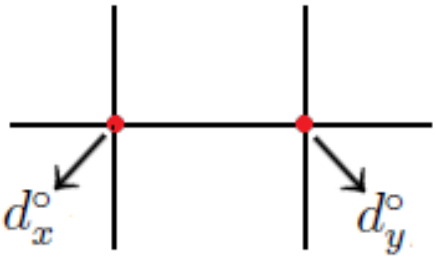



**Figure 3.3:** Dense Edge where $\omega > 5$.

To form $G_{HC}$ from $G(m, n)$, $V$ edges are required, while $\alpha$ edges are needed to be discarded (edges that don't take part in $G_{HC}$), where $\alpha = V - (m + n)$. In cycle graph, $V \equiv E$ and every vertex have degree 2. Every *Cycle Graph* is Hamiltonian. For constructing $G_{HC}$ from $G(m, n)$, $V$ number of vertices and edges are required. Figure 3.2 shows rectangular grid graph where $V = 24$ and $E = 37$. Figure 3.4 shows possible Hamiltonian circuits, where $V \equiv E$. Here maximum edges we may hide off in a grid graph is $\alpha = 13$. In a complete undirected graph of $n$ vertices, we may possibly construct $\frac{(n-1)!}{2}$ number of diverse Hamiltonian cycles, whereas in case of directed graph it is $(n - 1)!$.

**Lemma 3.1.2.** *$G_{HC}$ of $G(m, n)$ is possible iff $V \equiv 0$(mod 2) and $(m, n > 1)$.*

Chen et al. and Kohjerdi et al. proved this lemma with graph coloring [92][93].

*Proof.* To find Hamiltonian cycle for any rectangular grid graph, $V$ edges are needed. The Hamiltonian cycle is possible to find iff $V \equiv E$. Odd number of edges cannot make

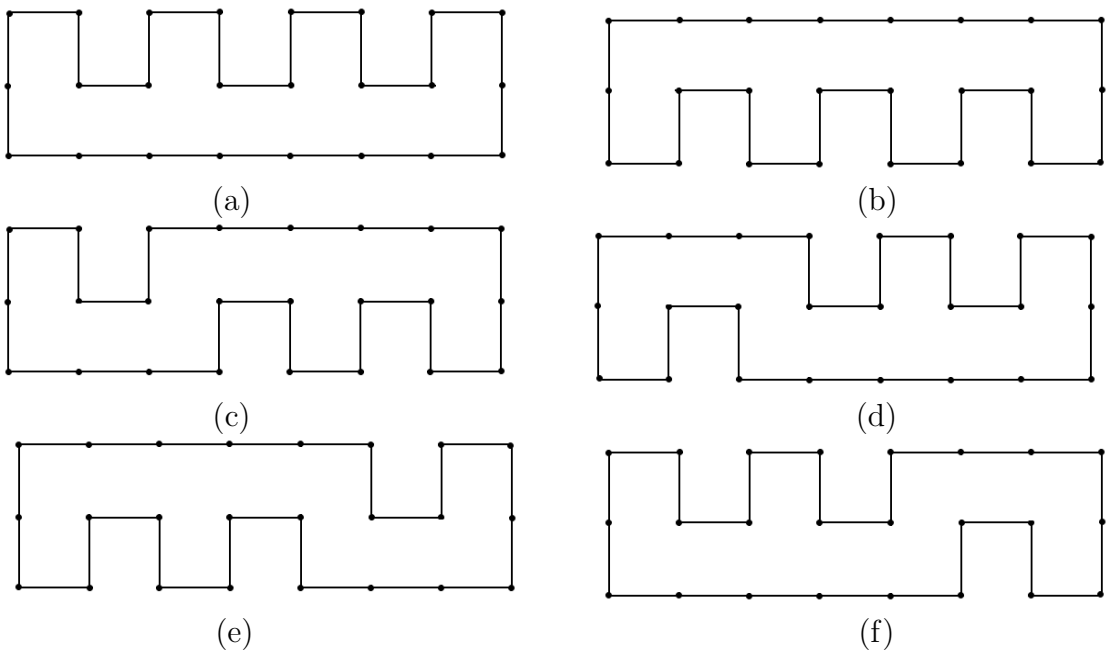



**Figure 3.4:** Possible Hamiltonian Circuits ($G_{HC}$) for $G(8,3)$ rectangular grid graph.

$d_2^\circ$ for each vertex in a rectangular grid graph. If $(m \times n)$ mod 2 is non-zero then after edge selection step, following cases may arise:

- At least one vertex will have $d^\circ = 0$ which means an isolated vertex.
- $d^\circ = 1$ or 3 means cycle cannot be formed.

  Both the cases are not favorable.

Figure 3.5(a) is an input graph $G(5,3)$. It has $V = 15$, $E = 22$ and $\alpha = 7$. After completion of edge selection step, the possible outputs are shown in figure 3.5(b), 3.5(c) or 3.5(d). Figure 3.5(b) has one vertex with $d^\circ > 2$ and one vertex with $d^\circ < 2$. Figure 3.5(c), 3.5(d) are left with vertex $d^\circ = 0$. By inspecting the figure, it is proved if $V$ is odd then Hamiltonian circuit is not possible. □

## 3.2 The Algorithm

The Algorithm 3.1 for finding Hamiltonian circuit has two phases. The first phase chooses

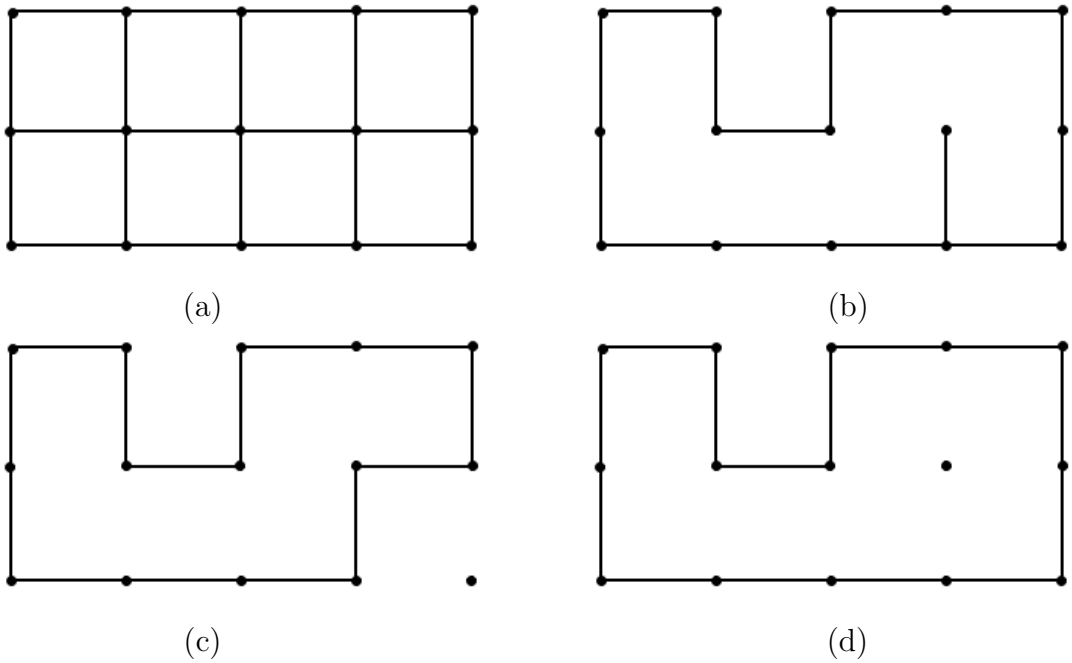



**Figure 3.5:** The possible outputs after selection step in $G(5, 3)$.

the contributing edges for Hamiltonian circuit. This edge selection is honestly done at random. The deterministic tactic may be used to find the contributing edges, that increases combinations so complexity may rise; however in case of randomized approach, this never exceeds $O(n^2)$. As a result of first step, we may get Hamiltonian cycle as an output or we get disconnected cycle graphs. In case cycles are found in graph, the second phase joins the cycle graphs by replacing the adjacent edges and hence completes the Hamiltonian cycle. In edge selection step, the devised algorithm firstly marks all the *Restricted Edges*(corner edges). Then starting from any vertex, it selects two edges randomly originating from that vertex. If one edge is already marked as restricted, then it selects the other one; if two edges are marked as restricted edges then it visits the next vertex. This step makes the degree two of each vertex. The expected results of this step may be any one of those shown in figure 3.6. This step may result in Hamiltonian cycle or disconnected cycle graphs. Disconnected cycle graphs have to join into a single graph by replacing the adjacent edges.

**Algorithm 3.1** Randomized Algorithm to find Hamiltonian cycle.

1: **if** $m \times n$ *is odd* **then**
2: Exit() {Hamiltonian cycle not possible}
3: **end if**
4: Mark all the restricted edges. {edge with $\omega = 4$}
5: $\delta \leftarrow G$.{Copy the $G(m, n)$ to $\delta$}
6: Discard $\alpha$ edges at random with Step 5 $\delta(m, n)$
7: ***EdgeSelection*** $\delta(m, n)$
8: **for** 1 to $\alpha$ **do**
9: **if** edge is dense **then**
10: Discard the edge & update the restricted list. {one discarded edge will influence the degree of two vertices so update is compulsory before next move.}
11: **else**
12: mark as restricted
13: **end if**
14: **end for**
15: **if** any vertex have degree not equal to 2 **then**
16: **goto** *Step 3* {start again}
17: **end if**
18: Find cycles
19: **if** $no\_of\_cycle \equiv 1$ **then**
20: **Exit()** {Hamiltonian cycle}
21: **else**
22: join cycles by replacing the adjacent edges.
23: **end if**

### 3.2.1 Deterministic Approach

In figure 3.7(a) and 3.7(b), two edges are marked (in dotted lines) such that these edges should be replaced with adjacent edges to complete the cycle in figure 3.7(c) and figure 3.7(d). This leads to the formation of a cycle by removing the marked edges and joining the original edges. This is because of the beauty of its structure. All edges in this type of graph are on an equal Euclidian distance, so it hardly matters which one of the edges is chosen. Choose any two adjacent edges that may join the two cycles. We must carry on this cycle merging process, until we are left with a single connected graph.

In algorithm 3.2, we have mentioned deterministic approach which may lead to combinatorial problem (combination of edges). The complexity of this step in best case will be

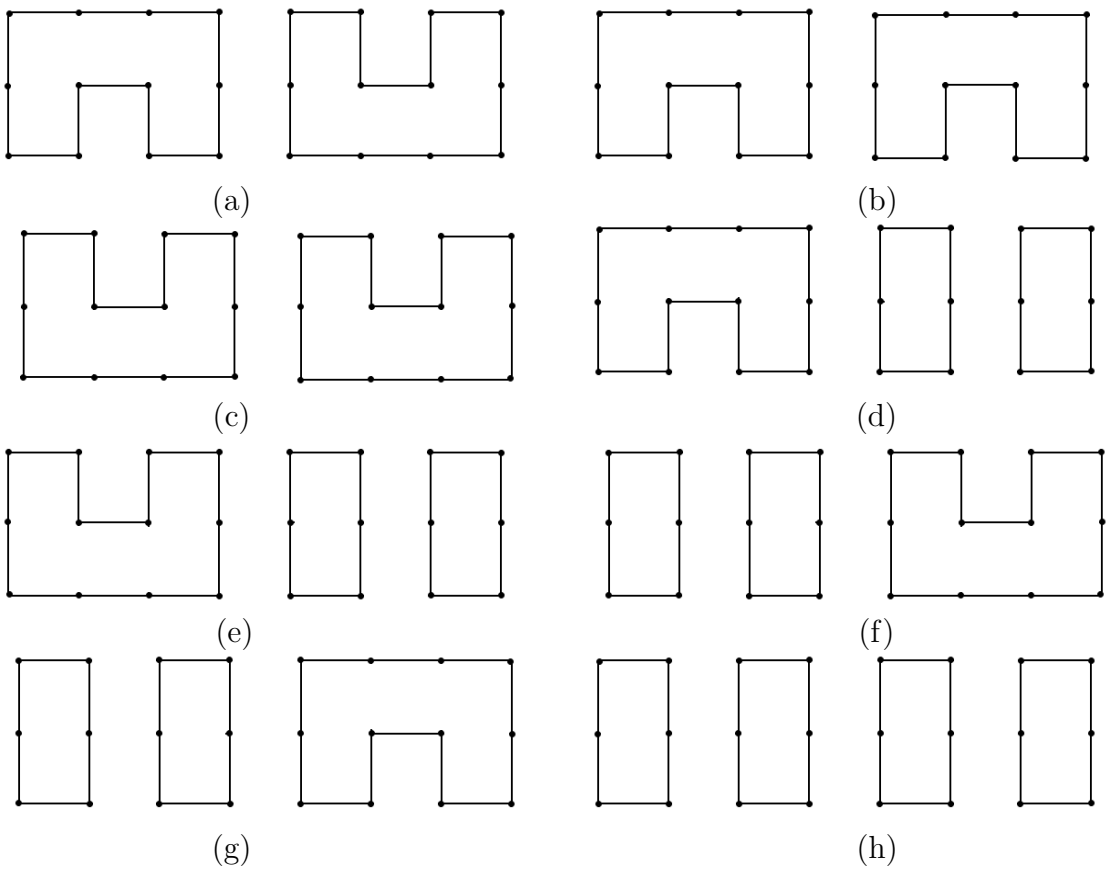



**Figure 3.6:** Possible Hamiltonian Circuits ($G_{HC}$) for $G(8,3)$ rectangular grid graph.

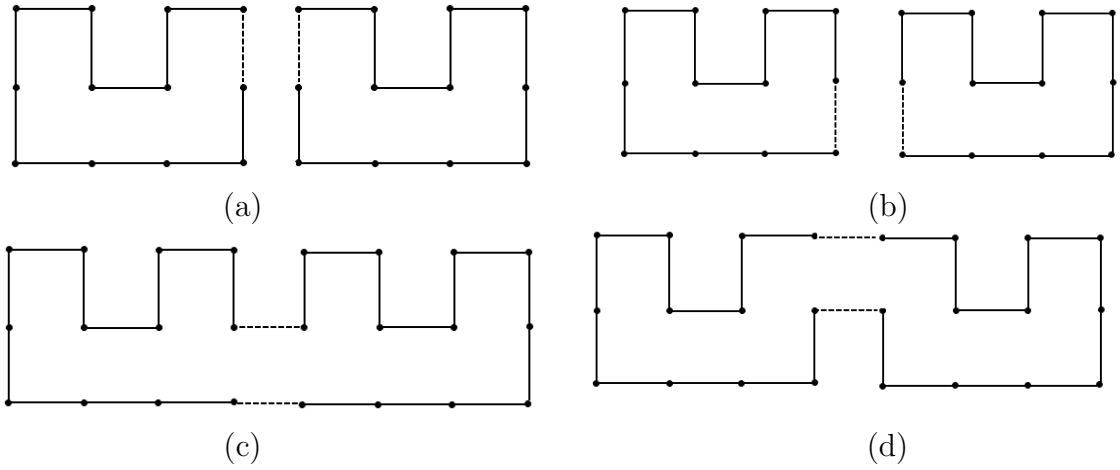



**Figure 3.7:** Joining Cycles by replacing the adjacent edges.

$O(n^3)$ which is why randomized approach is the best suitable for the discussed problem. Random selection is helpful to reduce the functional complexity as well.

**Algorithm 3.2** Deterministic Algorithm for *EdgeSelection*

```
1: Procedure EdgeSelection δ(m, n).
2: for 1 to α do
3:    M ← check_compulsory_edge()
4:    if M is not NULL then
5:       Delete one dense edge of the vertex M
6:    else
7:       Delete the last dense edge traversed.
8:    end if
9: end for
10: Procedure check_compulsory_edges()
11: for j ← 1 : V do
12:    Count ← 0
13:    if degree_of_vertex_j > 2 then
14:       for k ← 1 : V do
15:          if edge(j − k)exists then
16:             if degree_of_edge(j − k) > 2 then
17:                Count ← Count + 1
18:             end if
19:          end if
20:       end for
21:    end if
22:    if Count == degree_of_vertex_j − 2 then
23:       return j;
24:    end if
25: end for
26: return NULL
```

## 3.3 Analysis and Discussion

Proposed algorithm relies on a random sampling step. At first, mark the restricted edges that cannot be included in Hamiltonian cycle. No Dense edge can take part in $G_{HC}$. Now all Dense edges can be computed in linear time.

**Lemma 3.3.1.** *The expected number of restricted edges are n by the end of computation.*

*Proof.* This computation begins with marking 8 restricted edges. Process the remaining edges in increasing order by $\omega$. If the edge selected is a Dense edge then flip a coin that has probability $P$ heads upturn. Discard the edge if and only if the coin faces head. Now

edge $\alpha$ will not be a part of $G_{HC}$. $\square$

**Lemma 3.3.2.** *Suppose a graph contains $E$ number of edges of which $\alpha$ edges are to be eliminated and $E-\alpha$ are desired edges. Consider choosing $e$ edges at random (without replacement), where $E/2 \leq \alpha e \leq E$. Then $P$[exactly one eliminated edge is chosen] $\geq \frac{1}{2}e$.*

*Proof.*

$$\frac{\binom{\alpha}{1}\binom{E-\alpha}{e-1}}{\binom{E}{e}} = \alpha \frac{e!}{(e-1)!} \frac{(E-\alpha)!)}{E!} \frac{(E-e)!}{(E-\alpha-e+1)!}$$

$$= \alpha e \left( \prod_{i=0}^{\alpha-1} \frac{1}{E-i} \right) \left( \prod_{j=0}^{\alpha-2} (E-e-j) \right)$$

$$= \frac{\alpha e}{E} \left( \prod_{j=0}^{\alpha-2} \frac{E-e-j}{E-1-j} \right)$$

$$\geq \frac{\alpha e}{E} \left( \prod_{j=0}^{\alpha-2} \frac{E-e-j-(\alpha-j-1)}{E-1-j-(\alpha-j-1)} \right)$$

$$= \frac{\alpha e}{E} \left( \prod_{j=0}^{\alpha-2} \frac{E-\alpha-(e-1)}{E-\alpha} \right)$$

$$= \frac{\alpha e}{E} \left( 1 - \frac{e-1}{E-\alpha} \right)^{\alpha-1}$$

$$\geq \frac{1}{2} \left( 1 - \frac{1}{\alpha} \right)^{\alpha-1}$$

The last inequality follows from the observations that $\frac{\alpha e}{e} \geq \frac{1}{2}$ and $\frac{e-1}{E-\alpha} \leq \frac{1}{\alpha}$, which in turn follows from the assumption that $\frac{E}{2} \leq \alpha e \leq E$. Finally, applying this lemma, the probability of desired edges is bounded by $\frac{1}{2}e$. $\square$

### 3.3.1 Complexity Analysis

This section analyzes the complexity of the devised algorithm. Worst case analysis established the behaviour of the algorithm where random selection step may need to be run

again. In figure 3.8, $G(5,4)$ rectangular grid graph is chosen. Table 1 represents some of the special cases that may provoke the worst execution of algorithm. In case 1, all the given edges are discarded in Random Selection step then node 2 will be in a fix. In other words, it is not possible to make $d_2^\circ$, similarly in cases $2,3,4$, nodes $2,9,9$ will be in a fix (respectively). This will happen if all the edges in a particular row or column are discarded, which is fairly possible in deterministic approach. In case of random selection, probability to choose edge column wise or row wise is negligible.

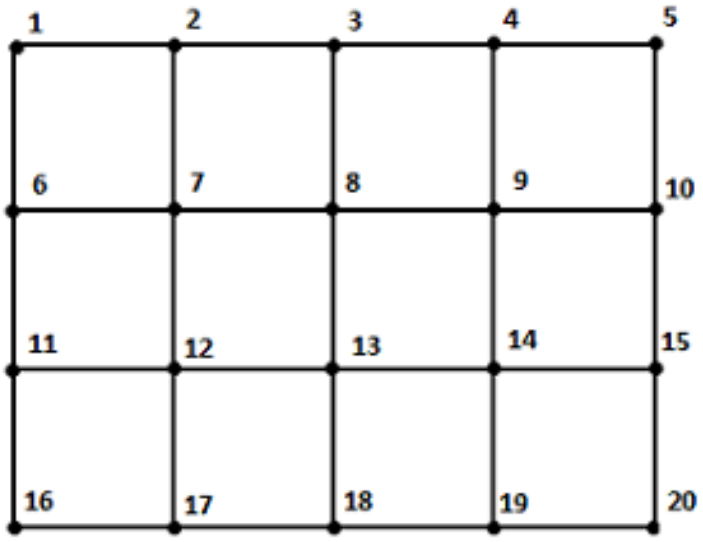



**Figure 3.8:** Rectangular Grid Graph $G(5,4)$.

**Table 3.1:** Example of Some Special Cases where selection step may need to run again.

| Case 1 | Case 2 | Case 3 | Case 4 |
|---|---|---|---|
| 9 - 10 | 9 - 10 | 14 - 15 | 15 - 14 |
| 8 - 9 | 15 - 14 | 13 - 14 | 14 - 13 |
| 7 - 8 | 9 - 8 | 3 - 4 | 12 - 13 |
| 6 - 7 | 8 - 7 | 12 - 13 | 17 - 18 |
| 3 - 4 | 6 - 7 | 17 - 18 | |
| | 3 - 4 | | |

### 3.3.2 Expected Running Time Analysis

We initiate our analysis by making some interpretations about the expected running time of the algorithms which is $O(mn)$. We show that the complexity of our algorithm in worst case is $O(n^2)$. Consider single call of the algorithm:

- **Step 1:** In this step, we discard the Dense edges by making $d_2^\circ$ of each vertex which takes $O(mn)$ time using state forward graph algorithm technique.
- **Step 2:** By traversing $\delta$ (as defined in Algorithm 3.1)we get all the cycles. This step takes $O(n)$ time, where $n = \alpha$.
- **Step 3:** Combining cycles found in step 2. This step takes $O(mn)$ time where $m$ and $n$ are number of vertices and edges in $\delta$.

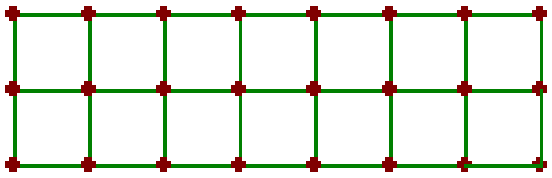
In worst case the aggregate time may go up to $O(n^2)$. The average case complexity of the proposed algorithm is $O(mn)$. The expected running time of the Hamiltonian circuit algorithm is $O(mn)$. Let number of edges are $n$, the number of edges to be discarded are $n-m$. To discard desired edges, procedure requires maximum $m$ iterations. Expectation comes out to be $E = [\frac{n-m}{m}]$.

## 3.4 Summary

Finally, the devised algorithm takes $O(mn)$ time to determine Hamiltonian Circuit. The results after the implementation of the devised algorithm are as follows:

- Figure 3.9(a) is input where $m = 8$ and $n = 3$.
- Figure 3.9(b) states the edges which need to be discarded.
- Figure 3.9(c) depicts the different disconnected graphs after edge selection step.

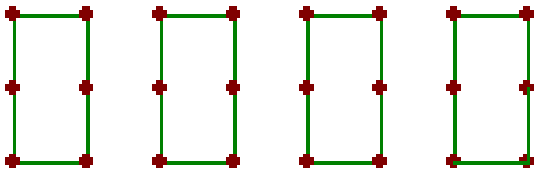
- Figure 3.9(d) shows the Hamiltonian Cycle of $G(8,3)$ after merging four disjoint cycles into single one.



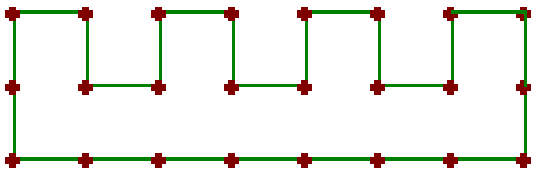
(a)

```
-----output-----
 vertex 1 edges 2 1---2 1---9
 vertex 2 edges 2 2---1 2---10
 vertex 3 edges 2 3---4 3---11
 vertex 4 edges 2 4---3 4---12
 vertex 5 edges 2 5---6 5---13
 vertex 6 edges 2 6---5 6---14
 vertex 7 edges 2 7---8 7---15
 vertex 8 edges 2 8---7 8---16
 vertex 9 edges 2 9---1 9---17
 vertex 10 edges 2 10---2        10---18
 vertex 11 edges 2 11---3        11---19
 vertex 12 edges 2 12---4        12---20
 vertex 13 edges 2 13---5        13---21
 vertex 14 edges 2 14---6        14---22
 vertex 15 edges 2 15---7        15---23
 vertex 16 edges 2 16---8        16---24
 vertex 17 edges 2 17---18       17---9
 vertex 18 edges 2 18---17       18---10
 vertex 19 edges 2 19---20       19---11
 vertex 20 edges 2 20---19       20---12
 vertex 21 edges 2 21---22       21---13
 vertex 22 edges 2 22---21       22---14
 vertex 23 edges 2 23---24       23---15
 vertex 24 edges 2 24---23       24---16
```

(b)



(c)



(d)

**Figure 3.9:** Output of devised algorithm

# Chapter 4

# Load Balancing in Distributed Networks

Computing has passed through many transformations since the inception of the first computing machines. Load balancing refers to efficiently distributing/routing incoming network traffic across a group of backend servers or computing resources. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process. A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. They are used to maximize the capacity utilization, speed and reliability. They improve the overall performance of applications by decreasing the burden on servers. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it. A centralized solution has one component that is shared by users all the time[94][95]. All resources are accessible, but there is a single point of control as well as a single point of failure. A distributed system is a group of autonomous computers connected to a computer network, which appears to the clients of the system as a single computer [96][108].

Load balancing is an important issue in parallel and distributed systems. So far, extensive research work has been done to propose various load balancing approaches. Load balancing strategy for optimal peak hour performance in Cloud data centers is proposed

by Kulkarni et al.[97][98]. An approach for dynamic load balancing of mobile agents is described by Omer [99].

In standard web server farm, a concentrated hardware load balancer is utilized to send jobs equally to the front end servers [100]. The most famous and result oriented algorithms are Join-the-shortest-queue (JSQ) algorithm, Shortest Queue of randomly chosen $d$ queues(SQ(d)) algorithm and Join-Idle-Queue (JIQ) algorithm. JSQ is the most prevalent technique and used in processor sharing server farms like in CISCO local Director, F5 Application Delivery Controller, Microsoft SharePoint and IBM Network Dispatcher [101][102]. In JSQ, a new request is allocated to the server which has the least count of unprocessed requests. In this manner, JSQ endeavors to adjust stack over the servers, diminishing the probability of a server having a few jobs whereas other servers are idle. From the aspect of incoming request, it is a greedy strategy for the instance of processor sharing servers because the incoming request would have a preference to share a server with as small number of jobs as possible [103].

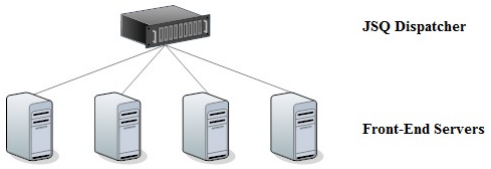



**Figure 4.1:** Join-the-shortest-queue (JSQ) algorithm

In JSQ algorithm, all incoming job requests come through a centralized load balancer and similar is true for the responses. The load balancer is now conscious of all the arrival and departure requests to an individual front end server, making it simple with no extra communication required for tracking. A customary web server ranch contains just a couple of servers, whereas appropriated server farms have hundreds or a great many processors for the front end alone. The capacity to scale horizontally in and out to adapt to the elasticity of demand is highly valued in data centers. A solitary equipment load

balancer, that accommodates many processors, is both expensive and inefficient as it increases the granularity of scaling[104]. Later, need for distributed dispatcher was felt in which centralized load balancing did not prove to be efficient enough [105]. Every server has its own list of arrivals and departures, so the actual load of the system could not be determined. To overcome this, it was required to have a communication channel which in turn, could decrease the overhead of communication and the load [106]. $\lambda(n)$ denotes the conditional arrival rate into the first queue. Specifically, we define:

$$\lambda(n) = \lim_{t \to \infty} \frac{A_n(t)}{T_n(t)}$$

where $A_n(t)$ is the count of arrivals amid the time interval $[0, t]$ finding $n$ occupations in the queue 1, where $T_n(t)$ is the aggregate time amid $[0, t]$ during which there are $n$ occupations in the queue 1. Every dispatcher autonomously adjusts its jobs since just a small amount of jobs use a specific dispatcher. The dispatcher has no information of the present number of jobs in every server which makes the execution of the JSQ algorithm troublesome.

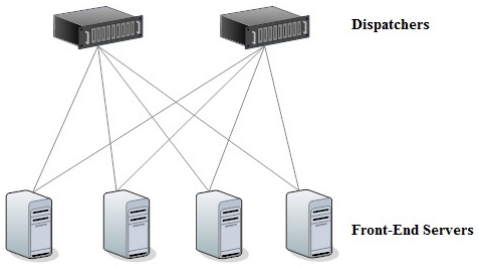



**Figure 4.2:** Distributed Dispatchers

## 4.1 Motivation

Following are the few limitations of backend load balancing which stimulated the researcher to define a new approach:

1. A dispatcher receives a job for processing but its I-Queue is empty since no processor is available in idle state. Due to this, job will be assigned to another server on random basis. It will send request, but this process will degrade the performance and moreover the idle processors in other I-Queues are underutilized.
2. When the I-Queue is empty, the request is sent to another server randomly but on the other hand, this server is associated with the I-Queue of other dispatcher which shows its (server) state as idle, whereas it is not. The server is already processing the job and will be in idle state once the job is accomplished.
3. After completing the job, the processor should join the dispatcher with an empty I-Queue.

To overcome the limitations of backend load balancing in JIQ, the devised approach introduces dequeue to be associated in the secondary load balancing of JIQ. When I-Queue of any dispatcher is exhausted, its entry is recorded in the dequeue. So as soon as the server finishes the job, it will refer this dequeue to be associated with the dispatcher (whose I-Queue is empty). All the limitations (as said above) are due to the emptiness of I-Queue. So dequeue can be effectively used here to ensure the availability of server in every I-Queue.

JIQ had been considered as the fundamental part for the proposed approach. The commitment is towards the improvement of backend load balancing (secondary load balancing) by guaranteeing the accessibility of server in every I-Queue.

This chapter proposed the exit plan for backend load balancing which is also referred as secondary load balancing (processor-to-dispatcher). Section 4.2 covers the Join-Idle-Queue algorithm [107]. Section 4.3 presents the system model. In section 4.4, the proposed approach is discussed. Section 4.5 embodies analysis. Section 4.6 summarizes this chapter.

## 4.2 Relevant Work

Join-Idle-Queue algorithm is suitable for the distributed environment where various dispatchers are put. In this approach, I-Queue is maintained by each dispatcher in which related server's entrance is kept. As an incoming request lands at the dispatcher, it sends to the processor referring the I-Queue and erase the entry of the processor from the I-Queue. If the I-Queue is unfilled then the job is moved to any available server on random basis. This is introduced as primary load balancing or dispatcher-to-processor load balancing. At the point when any processor completes the job, it needs to join any I-Queue. This is designated as secondary load balancing or processor-to-dispatcher load balancing. In JIQ, two methodologies are proposed for secondary balancing: JIQ-Random and JIQ-SQ(d). In JIQ-Random, processor may join any of the dispatcher while in JIQ-SQ(d), it joins only out of the $d$ sampled I-Queues.

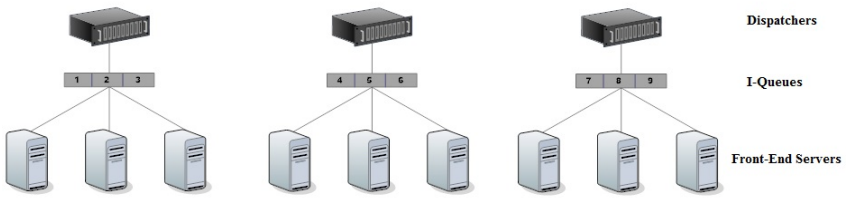



**Figure 4.3:** Join-Idle-Queue

## 4.3 System Model

Let $L$ be the length of the queue computed over $N$ number of dispatchers with $M$ number of processors. Now, length of the queue for a dispatcher-queue model is given by:

$$L = \frac{M}{N} \tag{4.1}$$

For a system, let $J$ be the jobs to be allocated to this dispatcher queue model such that process $\gamma$ can be allocated using JIQ approach. But this method can affect the processor allocation terribly as at time $t$, there can be queue which is left empty that adds to the cost of the system and affect its performance. Therefore, a system $S$ is required such that

$$S \longrightarrow f\left(N, M, J, \gamma\right) \tag{4.2}$$

where

$$L^s = \left\{x : x = length(Q^t) | x \neq 0\right\} \tag{4.3}$$

Here, $Q$ is the queue. For ideal state, $L^s = 0$. A system closer to ideal state will be treated as a solution to the problem of secondary load balancing provided the below given constraints hold.

### 4.3.1 System Constraint

Probability of queue $P$ being empty must follow a global minima principle and should be least at any given instance i.e. at time $t$, for a defined state,

$$\lim_{x \longrightarrow \max} P_i^t \longrightarrow \min \tag{4.4}$$

The probability $P_i^t$ for the $i^{th}$ queue at time $t$ per dispatcher is computed over the deviation of system from its normal state such that

$$P_i^t = \frac{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t - Q_E^t)^2}}{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t - Q_0^t)^2}} \tag{4.5}$$

where $Q_E$ is the empty queue slots, $Q_K$ is the actual queue slot, and $Q_0$ is the actual operating queue. For an ideal state, $Q_o^t = 0$, hence,

$$P_{i,ideal}^t = \frac{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t - Q_E^t)^2}}{\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_K^t)^2}} \tag{4.6}$$

Now, for a continuous process

$$\int_{0,t\in T}^{t} P_i^t dt \leq \min((4.5),(4.6)) \tag{4.7}$$

Here, minimum is defined as either (4.5) or (4.6) depending upon the requirement.

### 4.3.2 Operating Cost

For the considered system, jobs are considered to be following Poisson distribution. According to this statement, operating cost i.e. time to allocate the job should be minimum or equal to job processing time only. For job $y$, $y \in J$, time required for completion will be computed as:

$$T_y = t_y(A) + t_y(P) \tag{4.8}$$

where $t_y(A)$ is the allocation time and $t_y(P)$ is the processing time. Let $T_h$ be the threshold defined for the considered model above for which the $P_i$ may increase due to

overflow of queue. Therefore, for the defined system,

$$T_y \leq T_h \tag{4.9}$$

Hence, processor following Poisson distribution will be defined as:

$$P_f^t = \frac{J(t)^{M_a} e^{-J(t)}}{M_a!} \tag{4.10}$$

where $M_a$ denotes the actual processors available such that $M_a \in M$ and $J(t)$ is the number of jobs at time $t$. Now, the total time will be computed as:

$$T_T = \sum_{i=1}^{k} (P_f^t) \times t + \sum_{i=1}^{k} (t(A)) \tag{4.11}$$

For the overall system,

$$\frac{T_T}{I} \leq T_h \tag{4.12}$$

where $I$ is the number of iterations.

## 4.4 Our Approach

We have proposed the solution for load balancing in the inverse direction from processor to dispatcher. In JIQ, a job can arrive at dispatcher whose I-Queue is vacant, while there exist idle processors in neighboring I-Queues. In Secondary load balancing situation, JIQ-Random and JIQ-SQ(d) have ample chances that a processor will not join a vacant I-Queue. Our rationale is to guarantee that each I-Queue ought to have idle processors. To accomplish this, we are utilizing following architecture which is inspired by JIQ. In JIQ, an I-Queue is maintained by every dispatcher. The I-Queue is empty when all

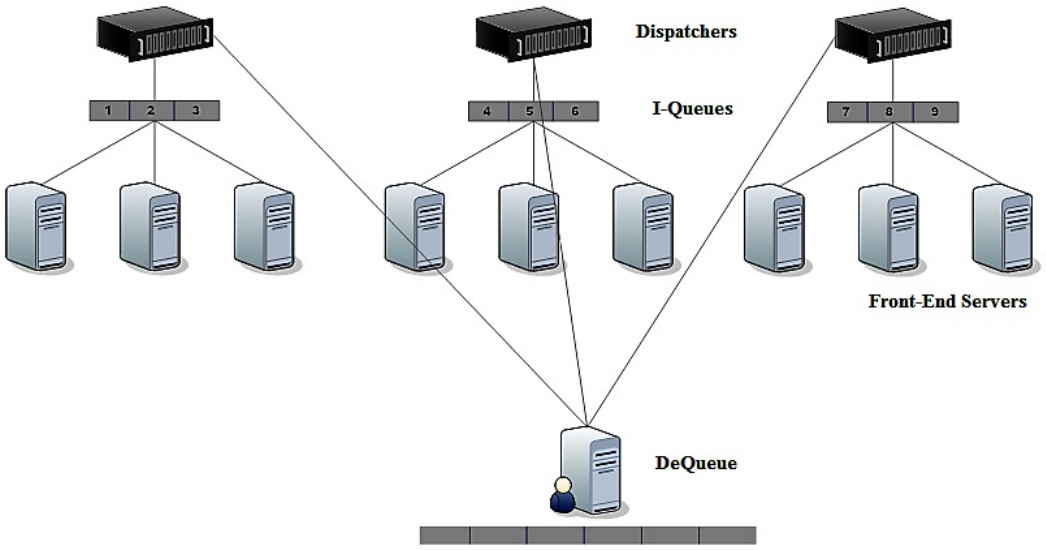



**Figure 4.4:** JIQ-JSQ

the servers are occupied. To keep up this consistency, when a server completes its job, it should join the dispatcher with a void I-Queue. A dequeue is maintained to track the dispatchers whose I-Queue is vacant. When any processor joins any dispatcher by advising the dequeue, if the I-Queue is void, the dispatcher ID is entered into the dequeue. As soon as the processor completes the current job, it refers the dequeue and joins the relevant dispatcher.

## 4.5 Analysis

The main challenge in join-idle queue-algorithm is sending processor back to dispatcher. Sending idle servers to I-Queues consequently helps in primary load balancing. Till date, two well established algorithms Random and SQ(d) have been used. In our methodology, we have used a server to make it work like JSQ to fill the empty I-Queue.

$$JSQ. \lim_{n\to\infty} \lambda_0, R_d = \frac{\lambda}{1-\lambda}$$

$$Random.\lambda_{0,R_1} = \lambda$$

$$SQ(d).\lim_{n\to\infty} \lambda_{0,R_d} = \lambda\frac{1-\lambda^d}{1-\lambda} = \lambda + \lambda^2 + \cdots + \lambda^d$$

### 4.5.1 Backend Load Balancing

When a front end server becomes free, it sends its ID to the dispatcher which then sends the processor to the I-Queue. We name this algorithm as JIQ-JSQ by which there is an improvement in secondary load balancing system. At a point when a front end server becomes idle, it sends its ID to the server which then sends the idle processor to the I-Queue.

Let $\rho$ be the proportion of occupied I-Queues in a system with $n$ servers in equilibrium. For JIQ-Random:

$$\frac{\rho}{1-\rho} = r(1-\lambda) \tag{4.13}$$

For JIQ-SQ(d):

$$\sum_{i=1}^{\infty} \rho^{\frac{d^i-1}{d-1}} = r(1-\lambda) \tag{4.14}$$

Similarly by putting the factor $d$ in JIQ-JSQ, $\rho = r(1-\lambda)$ (proportion of occupied I-Queues). Arrival rate to idle server in case of JIQ-JSQ

$$= \frac{\lambda\rho}{1-\lambda} + \frac{\lambda(1-\rho)}{1-\lambda} \tag{4.15}$$

$$= \frac{\lambda}{1-\lambda}(\rho + 1 - \rho) \tag{4.16}$$

$$= \frac{\lambda}{1-\lambda} \tag{4.17}$$

Hence, the arrival ratio of occupied server is $\frac{\lambda(1-\rho)}{1-\lambda}$ which is $\frac{1}{1-\rho}$ times greater to idle server than occupied server. We observe a substantial reduction in proportion of empty I-Queue, $n = 500$, $m = 50$, $r = 10$ and $\lambda = 0.6$

**Table 4.1:** Proportion of empty I-Queues

| Algorithm | Values |
|---|---|
| Random | 0.2 |
| SQ(2) | 0.027 |
| JSQ | 0 |

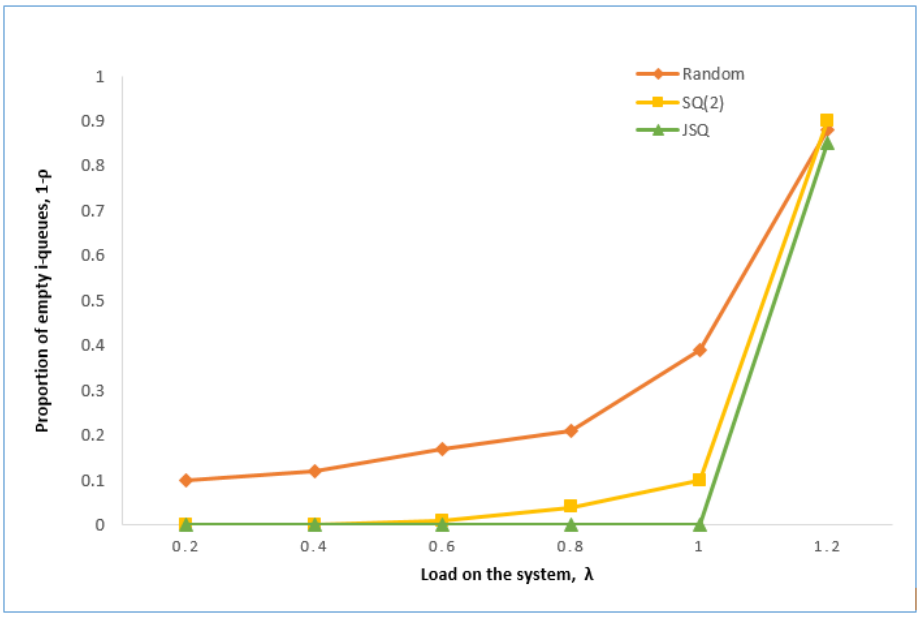



**Figure 4.5:** JIQ-JSQ

In JIQ literature, SQ(2) methodology is suggested well than the random methodology. Yet, Table 1 demonstrates improvement in the extent of empty queues with our JIQ-JSQ approach as compared to other available ones. At less or sensible load, it demonstrates the proportion to be null. Till the load lesser than 0.9, Figure 4.5 shows null with respect to the empty queues. For load till 0.9, there is no empty I-Queue. If in case it is greater than 0.9, when there are less number of idle processors than the number of I-Queue, our approach will follow the random algorithm to 1. Proportion of empty I-Queues with $r = 10$ ($m = 50$ & $n = 500$) and load is varied from 0.2 to 1. It compares the proportion of empty queues of Random, SQ(2) and JSQ. Note that the proportion of empty queues is nearly nil in case of JSQ till load 0.9. Figure 4.6 shows the analysis between the JIQ-JSQ (proposed approach) and the JIQ(SQ2). For mean response time, we have enhanced

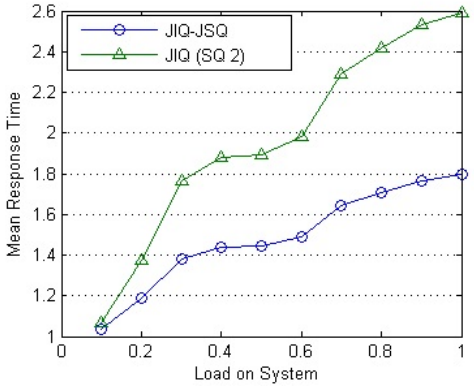



**Figure 4.6:** Mean Response Time: JIQ-JSQ and JIQ (SQ 2)

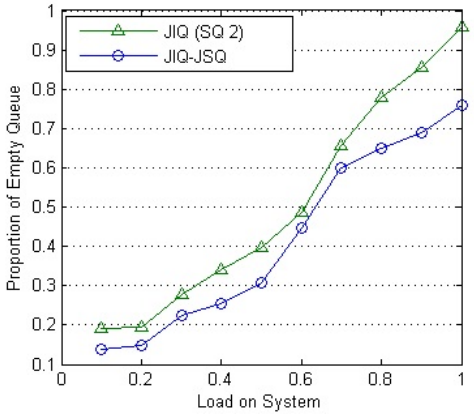



**Figure 4.7:** Empty I-Queue: JIQ-JSQ and JIQ (SQ 2)

with respect to the load on the system. The proposed approach provides a proactive solution to the JIQ problem thus, offers solution with 27% less response time. Figure 4.7 presents the analysis for proportion of empty queue after the applicability of the JIQ-JSQ approach. Comparison shows that the JIQ(SQ 2) has 19% larger proportion of the queue being empty than the proposed approach. This causes untimely delays in selection of processor and job has to wait for longer duration.

## 4.6 Summary

We propose the JIQ-JSQ algorithm for the backend load balancing in distributed network. JIQ-JSQ is better than both JIQ-Random and JIQ-SQ(d). This algorithm turns out to

be helpful at high load. By guaranteeing the availability of server in each I-Queue, the approach will likewise enhance the backend (dispatcher-to-processor) load balancing.

# Chapter 5

# Conclusion and Future Scope

## 5.1 Conclusion

The thesis addresses various important issues in the field . We conclude with this chapter by giving some final remarks about the addressed problems.

**Randomized Algorithm to Find Hamiltonian Circuit in Rectangular Grid Graph**[§]

The devised algorithm will take $O(mn)$ time with high probability to determine Hamiltonian Circuit. After implementing the devised algorithm, the results are shown in figure 3.9. The algorithm 3.1 has two phases. Firstly, it chooses the contributing edges for Hamiltonian circuit. This edge selection is done randomly. In edge selection step, the devised algorithm at first marks all the *Restricted Edges* (corner edges). Then starting from any vertex, it selects two edges randomly originating from that vertex. If one edge is already marked as restricted, then it selects the another one, if two edges are marked as restricted edges then it visits the next vertex. This step makes the degree two of each vertex. As a result of first step, we may get Hamiltonian cycle as an output or disconnected cycle graphs. In case disconnected cycles graphs are found, the second phase joins the disconnected cycle graphs by replacing the adjacent edges and completes the Hamiltonian cycle.

[§]Kuldeep Sharma and Deepak Garg, "Randomized Algorithm to Find Hamiltonian Circuit in Rectangular Grid Graph", International Journal of Applied Mathematics and Computer Science [under review].

**Load Balancing in Distributed Networks**[¶]

We propose the JIQ-JSQ algorithm for the dispatcher-to-processor load balancing in web server ranches. JIQ-JSQ altogether beats the JIQ-Random and JIQ-SQ (d). This algorithm turns out to be helpful at high load. By guaranteeing the availability of server in each I-Queue, it will likewise enhance the dispatcher-to-processor load balancing also. By proposing D-queue, we have also enhanced mean response time with respect to the load on the system. The proposed approach provides a proactive solution to the JIQ problem. Thus JIQ-JSQ offers solution with 27% less response time. We have presented the analysis for proportion of empty queue after applicability of the JIQ-JSQ approach. Comparison shows that the JIQ(SQ 2) has 19% larger proportion of the queue being empty than the proposed approach.

## 5.2 Thesis Contribution

The major contributions of this thesis are:

1. A comprehensive survey of existing deterministic as well as randomized algorithms.
2. Monte - Carlo algorithm for finding Hamiltonian Circuit in Rectangular Grid Graph.
3. An algorithm for secondary load balancing approach which is Processor-to-Dispatcher.

[¶]Sharma, K. and Garg, D., 2016. Approach for Processor-to-Dispatcher Load Balancing in Distributed Networks. International Journal of Next-Generation Computing, 7(1).

## 5.3 Future Scope

As we have observed from review of different graph theoretic applications, there is still scope of improvements in following aspects:

1. There are numerous issues whose deterministic algorithm exist, but there is scope to devise the randomized algorithm and various randomized algorithms are available where the lower bound has been proven, but algorithm of that lower bound complexity are not available, so there is scope to achieve that gap.
2. There is even scope to invent an algorithm for the all-pairs shortest paths problem that does not use matrix multiplication and runs in time $O(n^{3-\epsilon})$ for a positive constant $\epsilon$.
3. There are large number of multiobjective algorithms exist for SP and MST, there is huge scope to devise Las Vegas and Monte Carlo algorithms for both the problems.
4. One may devise randomized algorithm for triangular grid graphs.

# Appendix

## A.1 List of Publications

### Refereed Journals: Published/Accepted

1) Sharma, K. and Garg, D., 2016. Approach for Processor to Dispatcher Load Balancing in Distributed Networks. International Journal of Next-Generation Computing, 7(1).
2) Sharma, K. and Garg, D., 2011. Randomized Algorithms: Methods and Techniques. International Journal of Computer Applications IJCA, pp.29-32.
3) Kuldeep Sharma and Deepak Garg, "Multiobjective - Shortest Path and Spanning Tree", Journal of Engineering & Technology, [Accepted].

### Refereed Journals: Under Revision/Review

4) Kuldeep Sharma and Deepak Garg, "Randomized Algorithm to Find Hamiltonian Circuit in Rectangular Grid Graph", International Journal of Applied Mathematics and Computer Science [under review].

# Bibliography


[1] R. Motwani and P. Raghavan, *Randomized algorithms*. Chambridge University Press, 2010.

[2] C. A. R. Hoare, "Algorithm 65: Find," *Commun. ACM*, vol. 4, pp. 321–322, July 1961.

[3] R. Solovay and V. Strassen, "A fast monte-carlo test for primality," *SIAM journal on Computing*, vol. 6, no. 1, pp. 84–85, 1977.

[4] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987.

[5] L. Carter and J. Vogel, "Man," *Universal Classes of Hash Functions. J. Computer and System Sciences*, vol. 18, pp. 143–154, 1979.

[6] R. Freivalds, "Probabilistic machines can use less running time.," in *IFIP congress*, vol. 839, p. 842, 1977.

[7] A. C. C. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pp. 222–227, IEEE, 1977.

[8] R. W. Floyd and R. L. Rivest, "Expected time bounds for selection," *Communications of the ACM*, vol. 18, no. 3, pp. 165–172, 1975.

[9] M. Snir, "Lower bounds on probabilistic linear decision trees," *Theoretical Com-*

*puter Science*, vol. 38, pp. 69–82, 1985.

[10] A. Sinclair, *Algorithms for random generation and counting: a Markov chain approach*. Springer Science & Business Media, 2012.

[11] W. Ching and M. Ng, "Markov chains: Models, algorithms and applications, ser. international series in operations research & management science," 2006.

[12] R. M. Karp, "An introduction to randomized algorithms," *Discrete Applied Mathematics*, vol. 34, no. 1, pp. 165–201, 1991.

[13] M. O. Rabin, "Probabilistic algorithm for testing primality," *Journal of number theory*, vol. 12, no. 1, pp. 128–138, 1980.

[14] M. L. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pp. 338–346, Oct 1984.

[15] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[16] S. Warshall, "A theorem on boolean matrices," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 11–12, 1962.

[17] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 1–13, 1977.

[18] M. L. Fredman, "New bounds on the complexity of the shortest path problem," *SIAM Journal on Computing*, vol. 5, no. 1, pp. 83–89, 1976.

[19] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progres-

sions," *Journal of symbolic computation*, vol. 9, no. 3, pp. 251–280, 1990.

[20] Z. Galil and O. Margalit, "All pairs shortest paths for graphs with small integer length edges," *Journal of Computer and System Sciences*, vol. 54, no. 2, pp. 243–254, 1997.

[21] A. Shoshan and U. Zwick, "All pairs shortest paths in undirected graphs with integer weights," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pp. 605–614, IEEE, 1999.

[22] D. R. Karger, D. Koller, and S. J. Phillips, "Finding the hidden path: Time bounds for all-pairs shortest paths," *SIAM Journal on Computing*, vol. 22, no. 6, pp. 1199–1217, 1993.

[23] C. C. McGeoch, "All-pairs shortest paths and the essential subgraph," *Algorithmica*, vol. 13, no. 5, pp. 426–441, 1995.

[24] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *Journal of computer and system sciences*, vol. 51, no. 3, pp. 400–403, 1995.

[25] J. Current and H. Min, "Multiobjective design of transportation networks: Taxonomy and annotation," *European Journal of Operational Research*, vol. 26, no. 2, pp. 187–201, 1986.

[26] J. Current and M. Marsh, "Multiobjective transportation network design and routing problems: Taxonomy and annotation," *European Journal of Operational Research*, vol. 65, no. 1, pp. 4–19, 1993.

[27] Y. P. Aneja and K. Nair, "The constrained shortest path problem," *Naval Research*

*Logistics Quarterly*, vol. 25, no. 3, pp. 549–555, 1978.

[28] A. P. Wierzbick, "Basic properties of scalarizmg functionals for multiobjective optimization," *Optimization*, vol. 8, no. 1, pp. 55–60, 1977.

[29] P. Hansen, "Bicriterion path problems," in *Multiple criteria decision making theory and application*, pp. 109–127, Springer, 1980.

[30] J. C. N. Climaco and E. Q. V. Martins, "A bicriterion shortest path algorithm," *European Journal of Operational Research*, vol. 11, no. 4, pp. 399–404, 1982.

[31] E. Q. V. Martins, "On a multicriteria shortest path problem," *European Journal of Operational Research*, vol. 16, no. 2, pp. 236–245, 1984.

[32] J. R. Current, C. S. Revelle, and J. L. Cohon, "The median shortest path problem: a multiobjective approach to analyze cost vs. accessibility in the design of transportation networks," *Transportation Science*, vol. 21, no. 3, pp. 188–197, 1987.

[33] A. Warburton, "Approximation of pareto optima in multiple-objective, shortest-path problems," *Operations research*, vol. 35, no. 1, pp. 70–79, 1987.

[34] R. Batta and S. S. Chiu, "Optimal obnoxious paths on a network: transportation of hazardous materials," *Operations Research*, vol. 36, no. 1, pp. 84–92, 1988.

[35] J. Brumbaugh-Smith and D. Shier, "An empirical investigation of some bicriterion shortest path algorithms," *European Journal of Operational Research*, vol. 43, no. 2, pp. 216–224, 1989.

[36] J. Current, H. Min, and D. Schilling, "Multiobjective analysis of facility location decisions," *European Journal of Operational Research*, vol. 49, no. 3, pp. 295–307,

1990.

[37] R. L. Carraway, T. L. Morin, and H. Moskowitz, "Generalized dynamic programming for multicriteria optimization," *European Journal of Operational Research*, vol. 44, no. 1, pp. 95–104, 1990.

[38] J. A. d. Azevedo and E. Martins, "An algorithm for the multiobjective shortest path problem on acyclic networks," *Investigacao Operacional*, vol. 11, no. 1, pp. 52–69, 1991.

[39] J. Mote, I. Murthy, and D. L. Olson, "A parametric approach to solving bicriterion shortest path problems," *European Journal of Operational Research*, vol. 53, no. 1, pp. 81–92, 1991.

[40] C. T. Tung and K. L. Chew, "A multicriteria pareto-optimal path algorithm," *European Journal of Operational Research*, vol. 62, no. 2, pp. 203–209, 1992.

[41] K. Fujimura, "Path planning with multiple objectives," *Robotics & Automation Magazine, IEEE*, vol. 3, no. 1, pp. 33–38, 1996.

[42] P. Modesti and A. Sciomachen, "A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks," *European Journal of Operational Research*, vol. 111, no. 3, pp. 495–508, 1998.

[43] J. M. Coutinho-Rodrigues, J. C. Clímaco, and J. R. Current, "An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions," 1999.

[44] I. Murthy and D. L. Olson, "An interactive procedure using domination cones for bicriterion shortest path problems," *European Journal of Operational Research*,

vol. 72, no. 2, pp. 417–431, 1994.

[45] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *evolutionary computation, IEEE transactions on*, vol. 3, no. 4, pp. 257–271, 1999.

[46] C. H. Antunes, J. Craveirinha, J. Climaco, and C. Barrico, "A multiple objective routing algorithm for integrated communication networks," *Teletraffic science and engineering*, pp. 1291–1300, 1999.

[47] A. J. Skriver and K. A. Andersen, "A label correcting approach for solving bicriterion shortest-path problems," *Computers & Operations Research*, vol. 27, no. 6, pp. 507–524, 2000.

[48] M. Müller-Hannemann and K. Weihe, "Pareto shortest paths is often feasible in practice," in *Algorithm Engineering*, pp. 185–197, Springer, 2001.

[49] F. Guerriero and R. Musmanno, "Label correcting methods to solve multicriteria shortest path problems," *Journal of optimization theory and applications*, vol. 111, no. 3, pp. 589–613, 2001.

[50] J. Granat and F. Guerriero, "The interactive analysis of the multicriteria shortest path problem by the reference point method," *European Journal of Operational Research*, vol. 151, no. 1, pp. 103–118, 2003.

[51] R. Kumar and N. Banerjee, "Multicriteria network design using evolutionary algorithm," in *Genetic and Evolutionary ComputationGECCO 2003*, pp. 2179–2190, Springer, 2003.

[52] X. Gandibleux, F. Beugnies, and S. Randriamasy, "Martins' algorithm revisited for

multi-objective shortest path problems with a maxmin cost function," *4OR*, vol. 4, no. 1, pp. 47–59, 2006.

[53] G. Tsaggouris and C. Zaroliagis, "Multiobjective optimization: Improved fptas for shortest paths and non-linear objectives with applications," *Theory of Computing Systems*, vol. 45, no. 1, pp. 162–186, 2009.

[54] R. L. Graham and P. Hell, "On the history of the minimum spanning tree problem," *Annals of the History of Computing*, vol. 7, no. 1, pp. 43–57, 1985.

[55] R. C. Prim, "Shortest connection networks and some generalizations," *Bell system technical journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[56] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[57] D. Cheriton and R. E. Tarjan, "Finding minimum spanning trees," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 724–742, 1976.

[58] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[59] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[60] R. E. Tarjan, "Applications of path compression on balanced trees," *Journal of the ACM (JACM)*, vol. 26, no. 4, pp. 690–715, 1979.

[61] J. Komlós, "Linear verification for spanning trees," in *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pp. 201–206, IEEE, 1984.

[62] B. Dixon, M. Rauch, and R. E. Tarjan, "Verification and sensitivity analysis of minimum spanning trees in linear time," *SIAM Journal on Computing*, vol. 21, no. 6, pp. 1184–1192, 1992.

[63] V. King, "A simpler minimum spanning tree verification algorithm," *Algorithmica*, vol. 18, no. 2, pp. 263–270, 1997.

[64] B. Chazelle, "A minimum spanning tree algorithm with inverse-ackermann type complexity," *Journal of the ACM (JACM)*, vol. 47, no. 6, pp. 1028–1047, 2000.

[65] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 215–225, 1975.

[66] B. Chazelle, "The soft heap: an approximate priority queue with optimal error rate," *Journal of the ACM (JACM)*, vol. 47, no. 6, pp. 1012–1027, 2000.

[67] S. Pettie and V. Ramachandran, "An optimal minimum spanning tree algorithm," in *Automata, Languages and Programming*, pp. 49–60, Springer, 2000.

[68] S. Pettie and V. Ramachandran, "An optimal minimum spanning tree algorithm," *Journal of the ACM (JACM)*, vol. 49, no. 1, pp. 16–34, 2002.

[69] S. Pettie and V. Ramachandran, "Randomized minimum spanning tree algorithms using exponentially fewer random bits," *ACM Transactions on Algorithms (TALG)*, vol. 4, no. 1, p. 5, 2008.

[70] P. N. Klein and R. E. Tarjan, "A randomized linear-time algorithm for finding min-

imum spanning trees," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 9–15, ACM, 1994.

[71] D. R. Karger, P. N. Klein, and R. E. Tarjan, "A randomized linear-time algorithm to find minimum spanning trees," *Journal of the ACM (JACM)*, vol. 42, no. 2, pp. 321–328, 1995.

[72] A. W. Shogan, "Constructing a minimal-cost spanning tree subject to resource constraints and flow requirements," *Networks*, vol. 13, no. 2, pp. 169–190, 1983.

[73] V. A. Hutson and C. S. ReVelle, "Maximal direct covering tree problems," *Transportation Science*, vol. 23, no. 4, pp. 288–299, 1989.

[74] V. A. Hutson and C. ReVelle, "Indirect covering tree problems on spanning tree networks," *European Journal of Operational Research*, vol. 65, no. 1, pp. 20–32, 1993.

[75] H. W. Hamacher and G. Ruhe, "On spanning tree problems with multiple objectives," *Annals of Operations Research*, vol. 52, no. 4, pp. 209–230, 1994.

[76] I. Melamed and I. SIGAL, "An investigation of linear convolution of criteria in multicriteria discrete programming," *Computational mathematics and mathematical physics*, vol. 35, no. 8, pp. 1009–1017, 1995.

[77] K. A. Andersen, K. Jörnsten, and M. Lind, "On bicriterion minimal spanning trees: An approximation," *Computers & Operations Research*, vol. 23, no. 12, pp. 1171–1182, 1996.

[78] I. Melamed and I. K. Sigal, "Numerical analysis of tricriteria tree and assignment problems," *Computational mathematics and mathematical physics*, vol. 38, no. 10,

pp. 1707–1714, 1998.

[79] R. Ramos, S. Alonso, J. Sicilia, and C. González, "The problem of the optimal biobjective spanning tree," *European Journal of Operational Research*, vol. 111, no. 3, pp. 617–628, 1998.

[80] G. Zhou and M. Gen, "Genetic algorithm approach on multi-criteria minimum spanning tree problem," *European Journal of Operational Research*, vol. 114, no. 1, pp. 141–152, 1999.

[81] J. D. Knowles and D. W. Corne, "A comparison of encodings and algorithms for multiobjective minimum spanning tree problems," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, pp. 544–551, IEEE, 2001.

[82] A. Syarif, Y. Yun, and M. Gen, "Study on multi-stage logistic chain network: a spanning tree-based genetic algorithm approach," *Computers & Industrial Engineering*, vol. 43, no. 1, pp. 299–314, 2002.

[83] S. Steiner and T. Radzik, "Solving the biobjective minimum spanning tree problem using a k-best algorithm," tech. rep., Technical Report TR-03-06, Department of Computer Science, Kings College London, 2003.

[84] P. Perny and O. Spanjaard, "A preference-based approach to spanning trees and shortest paths problems," *European Journal of Operational Research*, vol. 162, no. 3, pp. 584–601, 2005.

[85] F. Neumann, "Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1620–1629, 2007.

[86] F. Neumann, "Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem," in *Parallel Problem Solving from Nature-PPSN VIII*, pp. 81–90, Springer, 2004.

[87] F. Neumann and C. Witt, "Multi-objective minimum spanning trees," in *Bioinspired Computation in Combinatorial Optimization*, pp. 149–159, Springer, 2010.

[88] F. Luccio and C. Mugnia, "Hamiltonian paths on a rectangular chessboard," in *Proceedings of the 16th Annual Allerton Conference*, pp. 161–173, 1978.

[89] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamilton paths in grid graphs," *SIAM Journal on Computing*, vol. 11, no. 4, pp. 676–686, 1982.

[90] F. Afrati, "The hamilton circuit problem on grids," *Informatique théorique et applications*, vol. 28, no. 6, pp. 567–582, 1994.

[91] C. Umans and W. Lenhart, "Hamiltonian cycles in solid grid graphs," in *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pp. 496–505, IEEE, 1997.

[92] S. Dong Chen, H. Shen, and R. Topor, "An efficient algorithm for constructing hamiltonian paths in meshes," *Parallel Computing*, vol. 28, no. 9, pp. 1293–1305, 2002.

[93] F. Keshavarz-Kohjerdi, A. Bagheri, and A. Asgharian-Sardroud, "A linear-time algorithm for the longest path problem in rectangular grid graphs," *Discrete Applied Mathematics*, vol. 160, no. 3, pp. 210–217, 2012.

[94] C.-Y. Lee, S. Piramuthu, and Y.-K. Tsai, "Job shop scheduling with a genetic algorithm and machine learning," *International Journal of production research*, vol. 35,

no. 4, pp. 1171–1191, 1997.

[95] S. C. Park, S. Piramuthu, N. Raman, and M. J. Shaw, "Intelligent scheduling with machine learning," in *Intelligent Scheduling Systems*, pp. 193–214, Springer, 1995.

[96] S. M. Thampi and C. S. K, "Survey of search and replication schemes in unstructured p2p networks," *arXiv preprint arXiv:1008.1629*, 2010.

[97] A. K. Kulkarni and B. Annappa, "Load balancing strategy for optimal peak hour performance in cloud datacenters," in *Signal Processing, Informatics, Communication and Energy Systems (SPICES), 2015 IEEE International Conference on*, pp. 1–5, IEEE, 2015.

[98] L. Thomas and B. Annappa, "Utilization of map-reduce for parallelization of resource scheduling using MPI: PRS," in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pp. 415–420, ACM, 2011.

[99] C. Georgousopoulos and O. F. Rana, "Combining state and model-based approaches for mobile agent load balancing," in *Proceedings of the 2003 ACM symposium on Applied computing*, pp. 878–885, ACM, 2003.

[100] T. C. K. Chou and J. A. Abraham, "Load balancing in distributed systems," *IEEE Transactions on Software Engineering*, no. 4, pp. 401–412, 1982.

[101] V. Gupta, M. H. Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms," *Performance Evaluation*, vol. 64, no. 9, pp. 1062–1081, 2007.

[102] Y.-T. Wang and R. J. T. Morris, "Load sharing in distributed systems," *IEEE Transactions on computers*, vol. 100, no. 3, pp. 204–217, 1985.

[103] M. S. Squillante and R. D. Nelson, *Analysis of task migration in shared-memory multiprocessor scheduling*, vol. 19. ACM, 1991.

[104] M. Bramson, Y. Lu, and B. Prabhakar, "Randomized load balancing with general service time distributions," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, pp. 275–286, ACM, 2010.

[105] H. L. Applewhite, R. Garg, E. D. Jensen, J. D. Northcutt, and L. Sha, "Decentralized resource management in distributed computer systems.," tech. rep., DTIC Document, 1982.

[106] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, "Queueing system with selection of the shortest of two queues: An asymptotic approach," *Problemy Peredachi Informatsii*, vol. 32, no. 1, pp. 20–34, 1996.

[107] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg, "Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services," *Performance Evaluation*, vol. 68, no. 11, pp. 1056–1071, 2011.

[108] S. M. Thampi and R. S. K, "Collaborative loadbalancing scheme for improving search performance in unstructured p2p networks," in *Proc. Of the 1 st Int. Conf. Contemporary Computing*, pp. 161–169, 2008.

[109] D. E. Knuth, *The art of computer programming: sorting and searching*, vol. 3. Pearson Education, 1998.

[110] R. Motwani and P. Raghavan, "Randomized algorithms," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 33–37, 1996.