IMPLEMENTATION & VISUALIZATION OF EDMOND KARP MAXIMUM FLOW FINDING ALGORITHM

By

Waleed Ahmed

Submitted to the Department of Computer Engineering
in fulfillment of the requirements for the degree of
Masters of Science
In
Computer Software Engineering

Thesis Supervisor

Brig. Dr. Muhammad Younus Javed

College of Electrical & Mechanical Engineering
National University of Sciences & Technology
2009

Implementation and Visualization of Edmond Karp Maximum Flow Finding Algorithm

By

Waleed Ahmed

2003-NUST-MS-PhD-CSE-230

Submitted to the Department of Computer Engineering
in fulfillment of the requirements for the degree of
Masters of Science
In
Computer Software Engineering

Thesis Supervisor

Brig. Dr. Muhammad Younus Javed

College of Electrical & Mechanical Engineering
National University of Sciences & Technology
2009

ACKNOWLEDGEMENTS

Firstly, I thank to Almighty ALLAH for enabling me to complete my research work.

Secondly, I would like to express my deepest gratitude to my dissertation supervisor and mentor Brig. Dr. Muhammad Younus Javed, without his guidance, encouragement, patience, and inspiration, the research for this dissertation never would have taken place. I am also grateful to my thesis committee. I am also especially grateful to Dr. Rafique and Dr. Shoaib khan. This exceptional faculty has taught me so much over the years, and has contributed significantly to my intellectual and professional development.

I am extremely grateful to my parents and my loving family for all the love and support they have given me over the years. Without their support, I would never have the chance to succeed.

I will always have many fond memories of wonderful events during the study period over the past years. They have provided lasting friendship, enlightenment, encouragement, and entertainment. I am also indebted to Naeem Akbar and Amir Bukharri for their computer wizardry support. I wish to thank to all those people who support me to achieve the glory.
Last, but certainly not the least, I would like to thank my best friends.

Dedicated to my parents and teachers

ABSTRACT

Graph is an effective data structure to solve the complex problems in Computer Sciences. Maximum Flow Problem is one of those problems, which are based on Graph data structure. The basic problem of finding a maximal flow in a network occurs not only in transportation and communication networks but also in currency arbitrage, image enhancement, machine scheduling and many other applications.

There are many algorithms designed to solve the maximum flow problems. The Edmond Karp algorithm is also included in the list of those algorithms, which provides the efficient and optimal solution for the maximum problems. It is upgraded version of Ford Fulkerson Method. The performance of the Edmond Karp algorithm is better than the Ford Fulkerson Method regarding to searching of paths in network graph. Edmond Karp algorithm uses the Breadth-first search (BFS) algorithm to find the augmenting paths in the network graph [6].

In this research, the performance of the Edmond Karp algorithm is analysed and evaluated through a given conditions in the form of input data. For this purpose, simulation is designed to monitors the efficiency of the algorithm in respect of different output parameters, like running time, number of paths and maximum flow.

The concept of the busy node is also elaborated in this research work. Busy node is the particular node in the network graph, which is existed in maximum number of augmenting paths. Normally it contains maximum share of the flow in the graph. The idea of the busy node can be utilized for optimal solutions of image processing and network related problems.

The simulation uses datasets for the experimental evaluation of the Edmond Karp algorithm. These datasets are collected through different methods: first one includes “Auto Generated Random Graphs” and other is “User Defined Graphs”.

The simulation is also stored the experiment results to observe the behaviour of the algorithm. At the end of this report, cumulative average time is calculated from the experiment results. These cumulative average times indicate the best as well as worst case scenarios of the algorithm. In best case, it produces 2.851ms and in worst case 19.7143ms.

TABLE OF CONTENTS

PART 1: INTRODUCTION

PART 2: LITERATURE SURVEY

PART 3: NETWORK FLOW PROBLEMS AND EDMOND KARP ALGORITHM

PART 4: SIMULATION AND EXPERIMENT RESULTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Motivation

The study of maximum flow problem is a matter of interest to a wide range of researchers. It is due to the widespread occurrence of such networks in practice and challenges involved in solving of associated problems. In many real-time systems such as airports, production facilities, highways and data networks, the costs of finding maximum flow may be very high.

Although a vast literature exists on the analysis of maximum flow finding algorithms yet its solution exists only for a very small set of problems, as several assumptions are required to make a network graph mathematically tractable to obtain exact solutions or near to exact solutions. Since many real-world systems are dynamic in nature so that their evaluation study is difficult. Therefore, it motivates the need to present a visual simulator to observe and analyse the performance of maximum flow finding algorithm on different data inputs.

Objectives of the Study

Graph is a pervasive data structure in computer sciences, and algorithms are fundamental to the relevant field for working with them. There are hundreds of interesting computational problems defined in terms of graphs. One of them is elaborated in this research work.

In this work, analysis about the computation of maximum flow of the material in a network graph having a specified source of material, a sink, and capacities for the mount of material that can traverse each directed edge through implementation of Edmond-Karp Algorithm. The problem of maximum flow arises in many forms and a good algorithm for computing maximum flow can be used to solve these problems efficiently in terms of outputs, i.e maximum flow, augmenting paths and running time of the algorithm.

CHAPTER 1

Introduction

1.1. Introduction

There are many different types of networks used in our everyday life; including electrical circuits, telephone exchange, cable, highways, railways and computer networks. Networks consist of special points called nodes and links connecting pairs of nodes called arcs. Some examples of networks are shown in the following Table–1.1. In all these networks, it is desired to send some material, which is generically called as 'flow', from one node to another and do so as efficiently as possible. Network flow theory is the study of designing efficient algorithms to solve such problems.

Networks	Nodes	Arcs	Flows
Communication	Telephone, Exchanges, computers, satellites.	Cables, fiber optics, Microwave relays.	Voice, Video, packets.
Circuits	Gates, Registers, Processors	Wires	Current
Mechanical	Joints	Rods, Beams, springs	Heat, energy
Hydraulic	Reservoirs, Pumping stations, lakes	Pipelines	Fluid, Oil
Financial	Stocks, Currency	Transactions	Money
Transportation	Airports, Rail yards, Street intersections.	Highways, Railbeds, Airway routes.	Freight, Vehicles, passengers.
Chemical	Sites	Bonds	Energy

Table – 1.1: Some Network Flows Examples

For this purpose, the performance and efficiency of the Edmond Karp Algorithm is analysed through simulation. Typical simulation consists of visual environment development, algorithm execution and data results analysis. Visual environment development stage involves the creation of graph canvas, where directed graphs are designed and visualized. It also shows the map of augmenting paths built after the execution of the algorithm. Simulation includes datasets analysis and results logging facility. The potential use of this facility is to observe the behavior and performance of the maximum flow finding algorithm on different datasets given by the user or generated by the simulator.

Performance of the simulation depends upon visual presentation, correct implementation of the algorithm, which produces required results and maintains the results in the data file.

1.2. Design and Implementation

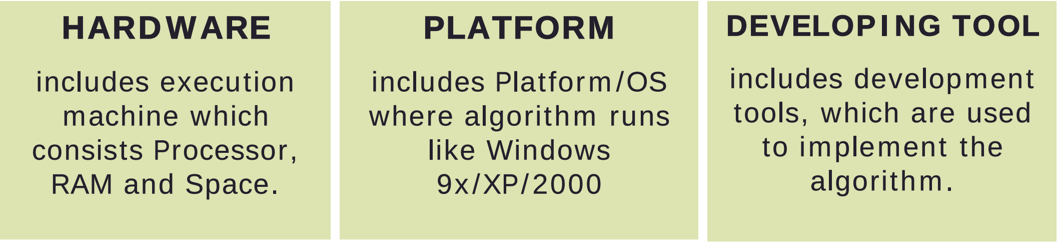
It is very important to understand the concept of the algorithm before its successful implementation. First of all, understand the concept of Edmond Karp maximum flow finding algorithm. It establishes a number of algorithmic techniques: augmenting paths, residual networks and cuts [2]. There are many applications that benefit from this solution, including network routing, highway design and circuit design. The Edmond Karp Algorithm builds on other algorithm and data structures such as breadth-first search algorithm, queues and graphs.

After the conceptual understanding, Simulation is designed to run **Edmond Karp** Algorithm under a set of given conditions in the form of input data. At run time, it monitors and records the performance of algorithm. Resultantly it generates data which is used for analysis and performance evaluation of the algorithm. Experimentation with the system and resultant analysis of the data has confirmed that the results are in conformity with the established facts and principles of the science of algorithms.

The design of algorithm is developed for its best implementation which contains the development of searching function, queue and constraints of the subjected algorithm. Next step is the selection of developing tool for the implementation of designed structure of the algorithm. After the development phase it is tested through different datasets to analyses the produced results by the simulation. Subsequently, if some bottlenecks are observed, its design structure is refined through readjustment in its implementation phase. It is also strived to improve its working mechanism, so that, it can extract more efficient form of the maximum flow finding algorithm.

1.3. Initial Requirements of the Simulation

The goal of the research is to provide an analysis of the results that elaborates the efficiency of Edmond Karp Algorithms for the applications in the vision. It compares the running time of several datasets, as well as visual presentation of the said algorithm. The performance of simulation depends on different parameters like selection of graph datasets, machine/hardware where algorithm should be executed for its implementation methodology. Three main external factors affect the performance of the simulation in respect of its running time as shown in figure -1.1. These factors are described as under:



HARDWARE

includes execution machine which consists Processor, RAM and Space.

PLATFORM

includes Platform/OS where algorithm runs like Windows 9x/XP/2000

DEVELOPING TOOL

includes development tools, which are used to implement the algorithm.

Figure – 1.1: Implementation Factors

1. **Hardware** includes execution machine, which consists of Processor, RAM, and Space etc.

2. **Platform** includes operating system, where simulation runs like Windows 9x/XP/2000. GUI based operation systems (OS) have more visual presentation support than other platforms. Our simulation tested on the Windows XP/2000.

3. **Development Tools** include, simulation development tools, which are used to implement the algorithm. In this scenario, Visual Basic 6.0 is used for the implementation of Edmond Karp maximum flow finding algorithm.

1.4. Objectives of the Simulation

Presently no system is available to offer an efficient, graphical and user friendly simulation environment on a PC based Platform for the maximum flow find algorithm. There is need for such a system, which could analysis of Maximum Flow Finding Algorithm, either for the purpose to design a system for the real world problems or as a tool to study the science of design and theory of Algorithms. The system has been designed to fulfill the gap and it offers GUI base environment for the analysis of the Edmond Karp algorithm.

The main objectives of the simulation are the visual representations of the work flow and performance analysis of the Edmond Karp maximum flow finding algorithm on different datasets. With the help of GUI based environment, it can be observed the selection criteria of the augmenting paths in the directed graphs without violating the constraints of the algorithm.

CHAPTER 2

Algorithms

2.1. Introduction

An algorithm is a recipe or a well-defined procedure for transforming some input into a desired output [2]. Perhaps the most familiar algorithms are those used for the adding and multiplying integers. In this scenario, Edmond Karp algorithm runs on the directed graphs, which contains nodes |V| and edges |E| to calculate the flow *f* as maximum as possible. There are few basic questions about algorithm as mentioned below:

2.1.1. Does it halt?

It is observed whether the algorithm halts with required results. It means that the algorithm should end its execution or not, when it achieves its core objective, like here it's desired to calculate the maximum flow in the graph as,

Value of flow f : $|f| = f(s,V) = f(V,t)$

2.1.2. Is it correct?

Analysis of the algorithm is to find that whether it correctly computes the maximum flow *f* without violation of constraints defined in Edmond Karp algorithm. It means that all the flows *f* from *u* to *v* (where $u \neq s,t$) will be initialized by 0 and constraints that satisfy the Flow $f : V \times V \rightarrow R$ are:

Capacity	:	$f(u,v) \leq c(u,v)$ *for all* u,v
Skew Symmetry	:	$f(u,v) = -f(v,u)$ *for all* u,v
Value of flow f	:	$|f| = f(s,V) = f(V,t)$

Thus, the algorithm is just the doing of the standard behavior, which is required for the desired outputs. It is most important that implemented algorithm should produce optimal results or close to optimal solutions.

2.1.3. How much time does it take?

It is observed that the algorithm is as fast as the standard algorithm. (How does it implement the search technique, which seems to play vital role in the algorithm). Execution time of the algorithm is also inspired by the implementation technique, hardware and software used to developing and testing of the algorithm.

Ford and Fulkerson were presented a basic algorithm for maximum flow problem which offer $O(V^3)$ time for the large networks. Edmond and Karp were made improvements in the Ford Fulkerson algorithm to enhance its performance regarding running time. It offers $O(VE^2)$ for the large networks which making some unexpected applications possible [6].

2.1.4. How much memory does it use?

(When cache-aware algorithms are observed, then question arises that what kind of memory is used, e.g. cache, main memory, etc.)

Memory consumption depends on the implementation technique and presentation of the application. As implementation of Graph data structure is required a large capacity of memory. Graph data-structure is implemented through two methods, one is array based and another is pointer based. Array based structure required more memory consumption then the pointer based structure.

2.2. Measuring Efficiency of Algorithm

Efficiency of the algorithm is measured on the basis of *worst-case complexity* [1],[2] i.e., the maximum number of machine operations that the algorithm requires to complete any problem instance of a given size. For network flow problems, the size depends on the number of edges *m*, the number of vertices *n*, and the integer value C used to represent capacities of the edges. A network flow algorithm is called a *polynomial time* algorithm if its worst-case complexity is bounded by a polynomial function of m, n, and $\log_2$ C [2]. It uses $\log_2 C$ because it represents the number of bits needed to store the integer C on a computer.

Comparing the performance of algorithms based on their worst-case complexity has gained widespread acceptance over the past three decades. For a given problem, the goal is to design a polynomial-time algorithm with the smallest worst-case complexity. There are many reasons to justify such a goal. First, this provides a mathematical framework in which it can compare different algorithms. Second, there is strong computational evidence suggesting a high correlation between an algorithm's worst-case complexity and its practical performance [2].

There are two approaches used to categorize the algorithms, which are mentioned under:

2.2.1. Approximation Approach

For many practical optimization applications, it is often satisfied with solutions that may not be optimal, but are guaranteed to be “close" to optimal. For example, if the input data to the in a problem is only known to a certain level of precision, then it is often acceptable to produce a solution of the same level of precision. A second important reason is that it can often tradeoff solution quality for computational speed; in many applications it can find a provably high quality solution in substantially less time than it would take to find an optimal solution.

Researchers are designed both exact and approximation algorithms for the generalized maximum flow problem. It presents a family of ϵ -approximation algorithms for every $\epsilon > 0$. It means that to find nearly optimal solutions of any prescribed level of precision. For example, when $\epsilon = 0.01$ our approximation algorithms are faster than their exact algorithms by roughly a factor of m, where m is the number of arcs in the underlying network.

2.2.2. Combinatorial Approach

Since the generalized flow problem can be formulated as a linear program, it can be solved by general purpose; linear programming methods include simplex, ellipsoid, and interior point methods. These continuous optimization methods are grounded in linear algebra.

This problem can also be solved by *combinatorial methods*. Combinatorial methods exploit the discrete structure of the underlying network, often using graph search, shortest path, maximum flow, and minimum cost flow computations as sub-routines. These methods have led to superior algorithms for many traditional network flow problems including the shortest path, maximum flow, minimum cost flow, and matching problems [2]. More recently, combinatorial methods have been used to develop fast approximation algorithms for packing and covering linear programming problems, including maximum flow and multi-commodity flow.

CHAPTER 3

Searching Techniques

3.1. Introduction

Searching is a process of considering possible sequences of actions, first it has to formulate a goal and then use the goal to formulate a problem.

A **problem** consists of four parts: the **initial state**, a set of **operators**, a **goal test** function and a **path cost** function. The environment of the problem is represented by a **state space**. A **path** through the state space from the initial state to a goal state is a **solution** [2].

In real life most problems are ill-defined, but with some analysis, many problems can fit into the state space model. A single general search algorithm can be used to solve any problem; specific variants of the algorithm embody different strategies.

Search algorithms are judged on the basis of **completeness**, **optimality**, **time complexity** and **space complexity**. Complexity depends on *b*, the **branching factor** in the state space, and *d*, the **depth** of the shallowest solution [1], [2].

Completeness : Is the strategy guaranteed to find a solution when there is one?

Time complexity : How long does it take to find a solution?

Space complexity : How much memory does it need to perform the search?

Optimality : Does the strategy find the highest-quality solution when there are several different solutions?

3.2. Searching Techniques

Some common searching techniques are given below:

3.2.1. Breadth-first search (BFS) expands the shallowest node in the search tree first. It is complete, optimal for unit-cost operators, and has time and space complexity of O(b^d). The space complexity makes it impractical in most cases [1], [2].

Using BFS strategy, the root node is expanded first, and then all the nodes generated by the root node are expanded next, and their successors, and so on. In general all the nodes at depth *d* in the search tree are expanded before the nodes at depth *d+1*.

3.2.2. Uniform-cost search (UCS) expands the least-cost leaf node first. It is complete, and unlike breadth-first search is optimal even when operators have differing costs. Its space and time complexity are the same as for BFS [1].

BFS finds the shallowest goal state, but this may not always be the least-cost solution for a general path cost function. UCS modifies BFS by always expanding the lowest-cost node on the fringe [1], [2].

3.2.3. Depth-first search (DFS) expands the deepest node in the search tree first. It is neither complete nor optimal, and has time complexity of O(b^m) and space complexity of O(bm), where m is the maximum depth. In search trees of large or infinite depth, the time complexity makes this impractical [1], [2].

DFS always expands one of the nodes at the deepest level of the tree. Only when the search hits a dead end (a non-goal node with no expansion) then the search goes back and expands nodes at shallower levels.

3.2.4. Depth-limited search (DLS) places a limit on how deep a depth-first search can go. If the limit happens to be equal to the depth of shallowest goal state, then time and space complexity are minimized.

DLS stops to go any further when the depth of search is longer than defined depth limits.

3.2.5. Iterative deepening search (IDS) calls depth-limited search with increasing limits until a goal is found. It is complete and optimal, and has time complexity of O(b^d).

IDS is a strategy that sidesteps the issue of choosing the best depth limit by trying all possible depth limits: first depth 0, then depth 1, then depth 2, and so on. In effect, IDS combines the benefits of DFS and BFS.

3.2.6. Bidirectional search (BDS) can enormously reduce time complexity, but is not always applicable. Its memory requirements may be impractical.

BDS simultaneously search both forward form the initial state and backward from the goal, and stop when the two searches meet in the middle, however search like this is not always possible [2].

Edmond Karp Algorithm uses the Breadth-first Search technique to search the augmenting paths in the directed graphs. Thus working mechanism of the BFS is elaborated onward.

3.3. Breadth First Search (BFS)

Breadth-first search expands the shallowest node in the search tree first. It is complete, optimal for unit-cost operators, and has time and space complexity of O(b^d). The space complexity makes it impractical in most cases [2].

Using BFS strategy, the root node is expanded first, and then all the nodes generated by the root node are expanded next, and their successors, and so on. In general, all the nodes at depth *d* in the search tree are expanded before the nodes at depth *d+1*.

3.3.1 Algorithm:

```
BFS(G,s) {
    initialize vertices;
    Q = {s};
    while (Q not empty) {
        u = Dequeue(Q);
        for each v adjacent to u do {
              if (color[v] == WHITE) {
                color[v] = GRAY;
                d[v] = d[u]+1; // compute d[]
                p[v] = u;  // build BFS tree
                Enqueue(Q,v);
                                        }
                                      }
 color[u] = BLACK;
}
```

BFS runs in O(V+E)

Note: BFS can compute d[v] = shortest-path distance from s to v, in terms of minimum number of edges from s to v (un-weighted graph). Its breadth-first tree can be used to represent the shortest-path.

3.3.2 BFS Function in Simulation

```
Public Function BFS(START As Integer, TARGET As Integer) As Long          ‘BFS function for search
  ‘
  Dim U, V As Integer                          ‘COUNTER VARIABLES

  For U = 0 To Val(CMBNODES.Text) – 1          ‘INITIALIZATION OF NODES COLOR FLAG
    COLORS(U) = 0
  Next U

  HEAD = 0
  TAIL = 0

  ENQUEUE (START)

  PRED(START) = -1

  Do While (HEAD <> TAIL)                      ‘EXECUTE TILL ALL NODES ARE VISITED
    U = DEQUEUE()
    For V = 0 To Val(CMBNODES.Text) – 1
                                               ‘AGLORITHM CONSTRAINT IS CHECKED HERE.
      If COLORS(V) = 0 And (CAPACITY(U, V) - FLOW(U, V)) > 0 Then
        ENQUEUE (V)
        PRED(V) = U
      End If
    Next V
  Loop

  If COLORS(TARGET) = 2 Then                   ‘IF ALL NODES ARE VISITED
    BFS = COLORS(TARGET)                       ' RETURN VALUE
  End If
‘
 End Function
```

3.3.3 Working Mechanism

The following steps explain the working mechanism of the Breadth-first search.

- *Form a one-element queue consisting of the root node.*

Until the queue is empty or the goal has been reached, determine if the first element in the queue is the goal node do nothing (or may stop, depends on the situation). If the first element is not the goal node, remove the first element from the queue and add the first element's children, if any, to the **back** of the queue. ENQUEUE and DEQUEUE procedures are used for adding and removing node from the queue, which are mentioned below:

```
Sub ENQUEUE(X As Integer)            ‘Procedure for add nodes in the Queue.
   Q(TAIL) = X
   TAIL = TAIL + 1
   COLORS(X) = 1       'GRAY
End Sub

Public Function DEQUEUE() As Long    ‘Function for pop-up node from the Queue.
   Dim X As Integer
   X = Q(HEAD)
   HEAD = HEAD + 1
   COLORS(X) = 2                             ' BLACK
   DEQUEUE = X                               ' RETURN VALUE
End Function
```

- *If the goal node has been found, announce success, otherwise announce failure.*

Note: This implementation differs with DFS in insertion of first element's children; DFS starts from **FRONT,** while BFS starts from **BACK**. The worst case for DFS is the best case for BFS and vice versa.

```
If COLORS(TARGET) = 2 Then
      BFS = COLORS(TARGET)                    ' RETURN VALUE – Target node reached.
   End If
```

- *The side affects of BFS:*

1. Memory requirements are a bigger problem for BFS than the execution time.
2. Time is still a major factor, especially when the goal node is at the deepest level.

The working mechanism of the BSF technique as shown in the Table-3.1

S #	Breadth-first Searching Technique
INPUT	
1	• A graph G = (V, E) (directed or undirected) • A **source** vertex s ∈ V
IDEA	
2	• Explore the edges of G to "discover" every vertex reachable from s, taking the ones closest to s first
OUTPUT	
3	• d[v] = distance (smallest # of edges, or shortest path) from s to v, for all v ∈ V • BFS tree

Table 3.1: Breadth-first Searching Technique

3.3.4. Complexity

Whereas depth first search required spaces proportional to the number of decisions, breadth first search requires space exponential in the number of choices. If there are c choices at each decision and k decisions have been made, then there are c^k possible boards that will be in the queue for the next round. This difference is quite significant to the given space restrictions of some programming environments.

CHAPTER 4

Maximum Flow Problem

4.1. Network Flows

A **network flow** graph G = (V, E) is a **directed** graph with two special vertices: the source vertex s, and the sink (destination) vertex t. Each vertex represents a point, where material is sent or received. An edge (u,v) in the graph means that there is a link from u to v. Each edge has an associated **capacity** c(u,v), always finite, representing the amount of capacity available on this edge/arc. For simplicity, it is assumed that there can only one edge (u,v) for vertices u and v, but it does allow reverse edges (v,u), [3]. The concept of the network flow can be explained with a practical example (as explained in 4.3).

4.2. Some Traditional Network Flow Problems

In this section, it is formally defined the shortest path, minimum mean cycle, maximum flow, minimum cut, and minimum cost flow problems. It is also stated the best known complexity bounds. These subroutines are used in generalized flow algorithms. Some traditional network flow problems are mentioned below:

4.2.1 Shortest Path Problem

In the *shortest path problem*, the goal is to find a simple path between two nodes, so as to minimize the total length. An instance of the shortest path problem is a network $G = (V,E, s,l)$, where s is a distinguished node called the *source*, and l is a length function. The problem is NP-hard if negative length cycles are allowed [2]. In networks with no negative length cycles, there are a number of polynomial time algorithms for the problem, e.g. Bellman-Ford [1], [2]. There are faster specialized algorithms for networks with nonnegative arc lengths, e.g., Dijkstra. We let SP(m,n) denote the complexity of solving a shortest path problem in a network with m arcs, and n nodes, and nonnegative lengths.

4.2.2. Minimum Mean Cycle Problem

In the *minimum mean cycle problem*, the goal is to find a cycle whose ratio of length to number of arcs is minimum. That is, want to find a cycle that minimizes length function. An instance of the minimum mean cost cycle problem is a network $G = (V,E,l)$, where l is a length function. Although it is NP-hard to find a cycle of minimum length, it is possible to find a minimum mean cycle in polynomial-time [2]. Virtually all known algorithms are based upon a shortest path computation in a network where negative length arcs are allowed.

4.2.3. Minimum Cut Problem

The *s-t minimum cut problem* is intimately related to the maximum flow problem. The input is the same as for the maximum flow problem. The goal is to find a partition of the nodes that separates the source and sink, so that the total capacity of arcs going from the source side to the sink side is minimum. Formally, it is defined an *s-t cut* $[S,T]$ to be a partition of the nodes $V = [S,T]$ so that $s\ 2\ S$ and $t\ 2\ T$. The *capacity of a cut* is defined to be the sum of the capacities of \forward" arcs in the cut: $u[S,T] = \text{X}\ v2S,w2T\ u(v,w)$: (cut capacity). The goal is to find an *s-t* cut of minimum capacity. It is easy to see that the value of any flow is less than or equal to the capacity of any *s-t* cut. Any flow sent from s to t must pass through every *s-t* cut, since the cut disconnects s from t. Since flow is conserved, the value of the flow is limited by the capacity of the cut. A cornerstone result of network flows is the much celebrated max-flow min-cut theorem of Ford and Fulkerson [6]. It captures the fundamental duality between the maximum flow and minimum cut problems.

Theorem: *The maximum value of any flow from the source s to the sink t in a capacitated network is equal to the minimum capacity among all s-t cuts.*

Proof: It is sufficient to show that the capacity of some *s-t* cut equals the value of some flow. Let f be a maximum flow. Choose S to be the set of nodes reachable from the source using only residual arcs in G_f. It is showed that $[S,T]$ is an *s-t* cut of capacity. By the definition of S, flow f saturates every forward arc in the cut, and does not send flow along any backward arcs in the cut. Thus, the net flow crossing the cut is $u[S,T]$. By flow conservation, the net flow sent across any s-t cut is equal to the value of the flow.

4.2.4. Minimum Cost Flow Problem

In the *minimum cost flow problem*, the goal is to send flow from supply nodes to demand nodes as cheaply as possibly, subject to arc capacity constraints. An instance of the minimum cost flow problem is a network $G = (V,E, b, u, c)$, where $b : V$ is a *supply function*, u is a capacity function, and c is a cost function. We say node v 2 V has supply if $b(v) > 0$ and demand if $b(v) < 0$. We assume that the total supply equals the total demand, otherwise the problem is infeasible. Let f be a flow. If there exists a negative cost residual cycle in G_f, then it can improve f by sending flow around the cycle.

4.2.5. Maximum Flow Problem

In the *maximum flow problem*, the goal is to send as much flow as possible between two nodes, subject to arc capacity limits. An instance of the maximum flow problem is a network $G = (V,E, s, t, u)$, where s is a distinguished node called the *source*, t is a distinguished node called the *sink*, and u is a capacity function. A *flow* is a pseudo flow that satisfies and the *flow conservation constraints* as mentioned in [2], [3].

This says that for all nodes except the source and sink, the net flow leaving that node is zero. It does not have to distinguish between flow entering and leaving node v because of the anti-symmetry constraints. The *value* of a flow f is the net flow into the sink.

The objective is to find a flow of maximum value. An *augmenting path* is a residual s-t path. Clearly if there an augmenting path exists in graph G_f, then it can improve f by sending flow along this path.

4.3. Maximum Flow Problem Analogy

Imagine that a courier service wants to deliver some cargo from one city to another. Company can deliver them using various flights from cities to cities, but each flight has a limited amount of space that company can use. An important question is, how much of cargo can be shipped to the destination using the different flights available? To answer this question, it explores what is called a network flow graph, and show how it model different problems using such a graph as shown in figure – 4.1.

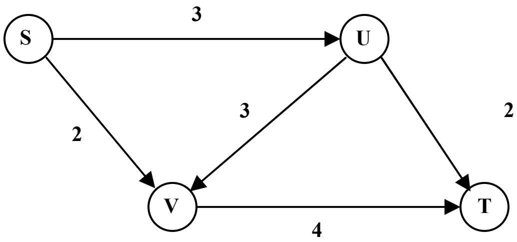


Figure – 4.1: A simple capacity network flow graph.

In context of the above example, company wants to know how much maximum cargo they can ship from s to t. Since the cargo "flows" through the graph from s to t, it is called as **maximum flow problem**. A straight forward solution is to do the following: keep finding paths from s to t where it can send flow along, send as much flow *f* as possible along each path $f_{p,}$, and update the flow graph afterwards to account for the used space. The following figure-4.2 shows an arbitrary selection of a path on the above graph.

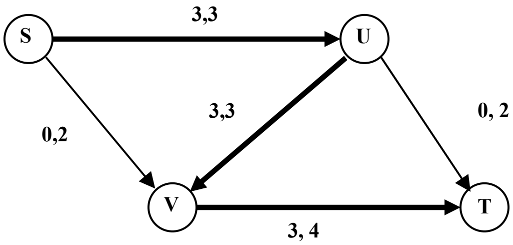


Figure – 4.2: The number on each edge is the flow, and the second is the capacity.

In Figure 4.2, it is picked a path s → u → v → t. The capacities along this path are 3, 3 and 4 respectively, which means it has a bottleneck capacity of 3 – it can send at most 3 units of flow along this path. Now it sent 3 units of flow along this path, and tries to update the graph. How should it do? An obvious choice would be to decrease the capacity of each edge used by 3 – it has used up 3 available spaces along each edge, so the capacity on each edge must decrease by 3. Updating this way, the only other path left from s to t is s → v → t. The edge (s,v) has capacity 2, and the edge (v,t) now has capacity 1, because of a flow of 3 from the last path. Hence, with the same update procedure. Algorithm execution is ends, but with non-optimal solution as shown in figure-4.3.

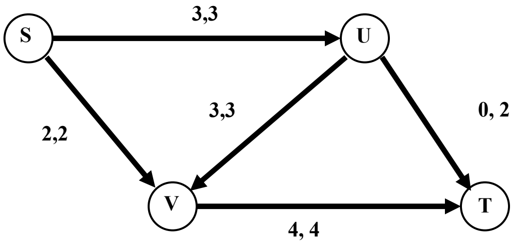


Figure - 4.3: Using path s → v → t. Algorithm ends, but this is not optimal.

Algorithm now ends, because it cannot find anymore augmenting paths from s to t (remember, an edge that has no free capacity cannot be used). However, can it do better? It turns out it can. If it only sends 2 units of flow (u,v), and diverge the third unit to (u,t), then it opens up a new space in both the edges (u,v) and (v,t). It can now send one more unit of flow on the path s → v → t, increasing our total flow to 5, which is obviously the maximum possible flow in the residual network graph. The optimal solution is shown in the following figure-4.4:

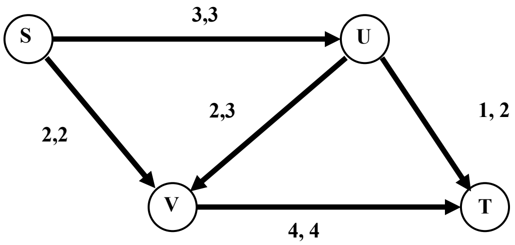
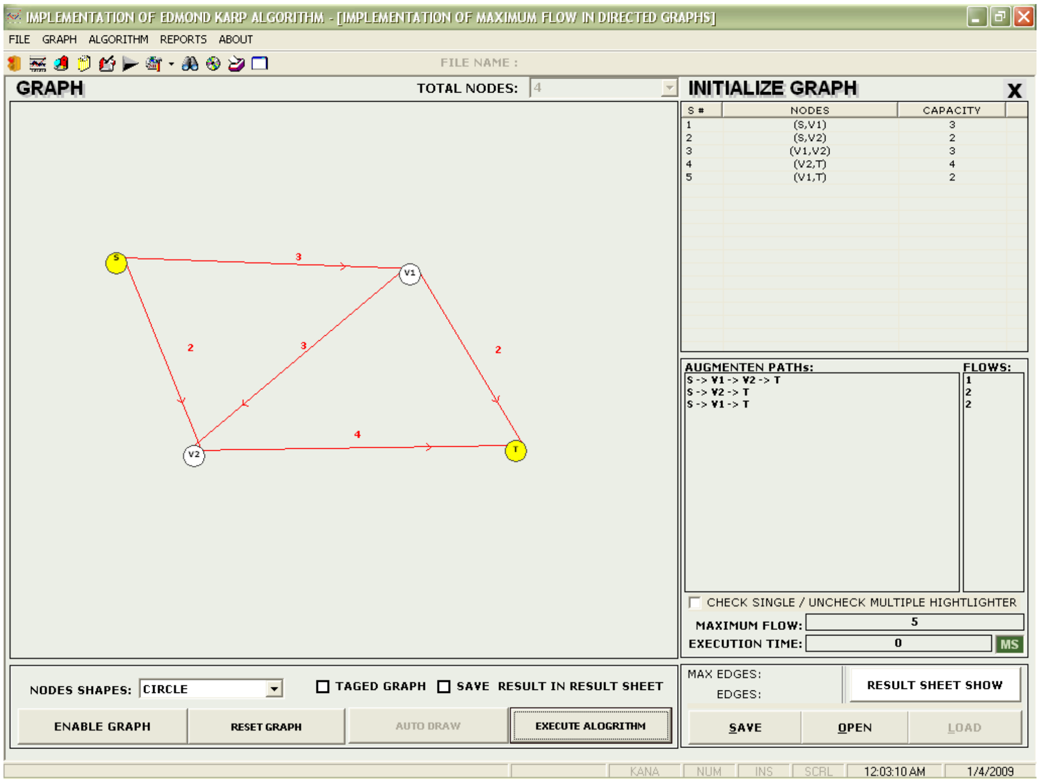


Figure – 4.4: Optimal solution.

So, what is wrong with working mechanism of algorithm? One problem was that it picked the paths in the wrong order. If it had picked the paths s → u → t first, then pick s → u → v → t, then finally s → v→ t, it would have ended up with the optimal solution. One solution is to always pick the right ordering or paths; but this can be difficult. Can it resolve this problem without worrying about which paths it picks first and which ones it picks last?

Comparing Figure-4.3 and Figure-4.4, it is observed that the difference between these two are in the edges (s,v), (v,u) and (u,t). In the optimal solution in Figure 4.4, (s,v) has one more unit of flow, (u,v) has one less unit, and (u,t) has one more unit. If it is examined that these three edges and form a path s → v → u → t, then it can interpret the path like this: first try to send some flow along (s,v), and there are no more edges going away from v that has free capacity. Now, it can **push back** flow along (u,v), telling others that some units of flow that originally came along (u,v) can now be **taken over** by flow coming into v along (s,v). After it pushes flow back to u, it can explore for new paths, and the only edge can be used is (u,t). The three edges have a bottleneck capacity of 1, due to the edge (s,v), and so it pushes one unit along (s,v) and (u,t), but push **back** one unit on (u,v). Think of pushing flow backwards as using a backward edge that has capacity equal to the flow on that edge.

It turns out that this small fix yields a **correct solution** to the maximum flow problem. First associates a **capacity** function along each edge c(u,v), that tells how many units of flow can go from u to v. Initially, c(u, v) is set to the maximum capacity for each edge (u, v). Now, it keeps finding paths from s to t (it refers to these paths as **augmenting paths**). When an augmenting path is found, it adjusts the capacity for each edge in the path. The algorithm ends when no more paths are found. Intelligent selection of the augmenting paths depend on the searching technique, which used to choose the more appropriate paths, like the worst scenario of the DFS is the best for the BFS search. Here analysis the preliminaries of the network flow problems. Now execute this example on the simulator to observe the maximum possible flow available in this graph as shown in figure – 4.5.



TOTAL NODES: 4

S #	NODES	CAPACITY
1	(S,V1)	3
2	(S,V2)	2
3	(V1,V2)	3
4	(V2,T)	4
5	(V1,T)	2

AUGMENTEN PATHs:	FLOWS:
S -> V1 -> V2 -> T	1
S -> V2 -> T	2
S -> V1 -> T	2

□ CHECK SINGLE / UNCHECK MULTIPLE HIGHTLIGHTER

MAXIMUM FLOW: 5

EXECUTION TIME: 0 MS

NODES SHAPES: CIRCLE

□ TAGED GRAPH □ SAVE RESULT IN RESULT SHEET

MAX EDGES:

EDGES:

12:03:10 AM 1/4/2009

Figure – 4.5: Simulation result for optimal solution

Many practical examples are available to understand maximum flow problem, like transferring of data from one computer node to another in computers network, transferring of oil from source well to destination sink through pipelines, sending signals from one point to another through cables in telecom network, etc.

4.4. Preliminaries of Network Flow Problems

In this part, several fundamental of network flow problems are reviewed. It formally defines the generalized maximum flow problem and reviews some basic facts that are used in the design and analysis of Edmond Karp maximum flow finding algorithm.

4.4.1. Basic Definitions

All of the problems are defined on a directed graph (V,E) where V is an n-set of nodes and E is an m-set of directed arcs. For notational convenience, it is assumed that the graph has no parallel arcs; this allows to uniquely specify an arc by its endpoints. Edmond Karp algorithm easily extends to allow for parallel arcs, and the complexity bounds are remained valid. It considers only simple directed paths and cycles.

4.4.1.1. Capacities:

The maximum flow, minimum cost flow, and generalized maximum flow problem use a *capacity function* [2]. The capacity $c(v,u)$ limits the amount of flow; it is permitted to send into arc (v,u). There is capacity constraints, which should not violated for the accurate execution of the Edmond Karp maximum flow finding algorithm as:

Proof of capacity constraint:

Here: $f' = f + f_p$ is valid flow in G (f a flow in G, f_p a flow in residual network) with a value $|f'| = |f| + |f_p|$.

Thus,

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p \\ -c_f(p) & \text{if } (v,u) \text{ is on } p \\ 0 & \text{otherwise} \end{cases}$$

$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is on } p\}$$

$$c_f(u,v) = c(u,v) - f(u,v)$$

For all $u,v \in V$, we require $(f+f_p)(u,v) \leq c(u,v)$

Proof: $f_p(u,v) \leq c_f(u,v) = c(u,v) - f(u,v)$

$(f+f_p)(u,v) = f(u,v) + f_p(u,v) \leq c(u,v)$

4.4.1.2. Symmetry:

For the maximum flow problems, it assumes the input network is *symmetric*. This is without loss of generality, since it could always add the opposite arc and assign it zero capacity. Without loss of generality, it is also assumed that costs are *antisymmetric*, i.e., $c(v,u) = -c(u, v)$ for every arc (v,u).

It means that all the flows f from u to v (where $u \neq s,t$) will be initialized by 0 and constraints that satisfy the Flow $f : V \times V \rightarrow R$ are:

a) Skew symmetry: For all $u,v \epsilon V$, it requires, $(f+f_p) = -(f+f_p)(v,u)$

b) Flow conservation: For all $u \epsilon V - \{s,t\}$, it requires, $\sum v\ v(f+f^p)(u,v) = 0$

Proof of (a):

$$\begin{aligned}(f+f_p)(u,v) &= f(u,v) + f_p(u,v)\\ &= -f(v,u) - f_p(v,u)\\ &= -(f(v,u) + f_p(v,u))\\ &= -(f+f_p)(v,u), \text{ done.}\end{aligned}$$

Proof of (b): Let $u \epsilon V - \{s,t\}$, then

$$\begin{aligned}\sum_v v\ (f+f_p)(u,v) &= \sum_v v\ (f(u,v) + f_p(u,v))\\ &= \sum_v v\ f(u,v) + \sum_v v\ f_p(u,v)\\ &= 0 + 0 = 0, \text{ done}\end{aligned}$$

The main objective of the algorithm is to maximize the flow:

Value of flow f: $|f| = f(s,V) = f(V,t)$

4.4.1.3. Residual Networks:

With respect to a pseudo-flow f in network G_f, the *residual capacity function* $c_f : E\ !<0$ is defined by $c_f(v,u) = c(v,u) - f(v,u)$. The *residual network* is $G_f = (V,E,c_f)$. Note that the residual network may include arcs with zero residual capacity, and still satisfies the symmetry assumption [2].

For example, if $c(v,w) = 20;\ c(w, v) = 0$, and $-f(w,v) = 17$, then arc $(v.w)$ has 3 units of residual capacity, and arc (w, v) has 17 units of residual capacity. It defines $E_f = f(v,w)$, where E: $c_f(v,w) > 0$ to be the set of all arcs in G_f with positive residual capacity. A *residual arc* is an arc with positive capacity. A *residual path (cycle)* is a path (cycle) consisting entirely of residual arcs.

Residual capacities : $c_f(u,v) = c(u,v) - f(u,v)$

Residual network : $G_f = (V,E_f)$, where

$E_f = \{(u,v) \in V \times V : c_f(u,v) > o\}$

4.4.1.4. Augmenting Path:

If there is some flow in the residual network and can be find a path p from source s to sink/destination t, in such a way that, there is units of flow a, which is greater then zero $(a > 0)$, and for each edge (u,v) in p, we can add units are added in flow: $f(u,v) + a \leq c(u,v)$, such path p is called augmenting path [3],[4].

CHAPTER 5

EDMOND KARP MAXIMUM FLOW ALGORITHM

5.1. Maximum Flow in Network

The basic problem of finding a maximal flow in a network occurs not only in transportation and communication networks, but also in currency arbitrage, image enhancement, machine scheduling and many other applications. Improvements in the basic algorithm are presented by Richard Edmonds and Karp [6]. The improved form of algorithm offers $O(VE^2)$ time which is better for large networks, making some of these unexpected applications possible.

The algorithms used to determine maximal flow are based on *Max-Flow/Min-Cut theorem* [2]. It is the property that defines the minimum cut in a flow network of a directed graph as a bottleneck that limits the maximum flow. Ford and Fulkerson have proved mathematically that these two forms were equivalent – the maximum flow goes through the minimum cut, to maximize the flow [5].

5.2. Max-Flow / Min-Cut Theorem

A cut is a set of edges separating s and t such that $s \in X$, $t \in \underline{X},$ V = {X+$\underline{X}$) and the capacity of the cut is c(X,$\underline{X}$). Suppose a cut is a bottleneck or choke point, the maximum flow cannot be greater than the minimum of all the cuts; hence the intuition behind the min cut is to maximize the flow as in [6].

For any network the maximum flow from source s to sink t is equal to the minimum cut capacity of all cuts separating s and t.

.

Lemma : $f(X,\underline{X}) - f(\underline{X},X) \leq c(X,\underline{X})$

Cormen : - The following statements are equivalent

- f is a maximal flow in G

- The residual network G_f has no augmenting path

- $|f| = c(X,\underline{X})$ for some cut $(X,\underline{X})$ of G

5.3. Maximum Flow as a Linear Programming Problem:

Richard Edmonds and Karp noted that the network flow problem could be formulated as a linear programming problem, either in Primal or Dual form [6].

Maximize: v the value of the total flow.

Subject to: $\Sigma f(u,v) - \Sigma f(v,u) = 0$ for all u,v in {G-s,t}

$\Sigma f(s,u) - v = 0$ for all successor nodes to source s

$\Sigma f(u,t) - v = 0$ for all predecessor nodes to sink t

$0 \leq f(u,v) \leq c(u,v)$ for all edges (u,v)

The network flow algorithm has much better performance for any large network application. In other words, some linear problems can be solved more effectively with flow algorithms.

5.4. Edmond Karp Maximum Flow Finding Algorithm

The Edmond Karp Algorithm represents a basic insight formalized into a maximum flow finding algorithm [6] that guarantees finding of a maximal flow for networks with non-negative capacities ($c(u,v) \geq 0$), while capacities are rational numbers.

Edmond Karp finds augmenting flows on simple path from source to sink and increases flow along the path up to the minimum of $c(u,v) - f(u,v)$ for all u,v in the path.

The original algorithm did not deal with negative or irrational valued flows in edges. Its performance depends on the value of the maximum flow, since each augmenting path might add only a small increment of flow. Performance of the algorithm is $O(E|f^*|)$, where f^* represents the flow of the graph.

The analysis of the running time of the algorithm on a given graph **G = (V,E)** usually measures the size of the input in terms of the number of vertices |**V**| and the number of edges |**E**| of the graph. There are two parameters describing the size of the input, not just one.

5.4.1. Basic Concept

Edmond-Karp Algorithm uses an approach that tags each vertex and scans neighbouring vertices to find a possible existing path (**p**) from source (**s**) to sink (**t**). If it finds a path (p) without violating constraint f(u,v) < c(u,v) for all edges in the path, it uses that additional flow to **augment** the current flow value [6]. When no augmenting flow path can be found among source (s) and sink (t) then algorithm terminates. Edmond-Karp proved that the algorithm only terminates when a maximum flow has been found [6]. Edmond-Karp Algorithm is also known as upgraded version of the Ford Fulkerson method. Its performance is better than that of Ford Fulkerson Method by using of **BFS** searching technique (BFS algorithm is greedy in nature, it always explore the shortest paths first).

5.4.2. Algorithm

```
Edmond-Karp(G,s,t)          (G = Graph, s = Source node, t = Destination node)
    Initialize flow f to 0 everywhere
    while there is an augmenting path p do
        augment flow f along p
    return f
```

Edmond Karp maximum flow finding algorithm can be expanded as:

EdmondKarp(*G,s,t*)

```
For each edge $(u,v) \in E(G)$
do $f(u,v) = 0$
   $f(v,u) = 0$
while there exist a path p from s to t in the residual network $G_f$
    do $c_f(p) = \min\{c_f(u,v): (u,v) \in p\}$
        for each edge (u,v) in p
            do $f(u,v) = f(u,v) + c_f(p)$
               $f(v,u) = -f(u,v)$
```

5.4.3. Function of Edmond Karp Algorithm in Simulator

The function used in the Edmond Karp Algorithm is mentioned below:

```
Public Function FF(Source As Integer, SINK As Integer) As Long
   Dim I, J, U, W, L As Integer              'Counter Variables
   Dim INCRE As Long                                   'Incremental Counters
   Dim MAX_FLOW As Long
   Dim T1, T2                                ‘Execution time checker variables
   MAX_FLOW = 0    L = 0
T1 = DateTime.Timer
   For W = 0 To 499
      AGT(W) = -1
   Next W
 W = 0
   For I = 0 To Val(CMBNODES.Text) - 1
          For J = 0 To Val(CMBNODES.Text) – 1
```

```
                 FLOW(I, J) = 0
       Next J
       SKEW(I) = 0
Next I
   If CMBSEARCH.ListIndex = 0 Then
  Do While (BFS(Source, SINK) = 2)
    INCRE = 1000000
    U = Val(CMBNODES.Text) - 1
    Do While (PRED(U) >= 0)
      INCRE = MINI(INCRE, (CAPACITY(PRED(U), U) - FLOW(PRED(U), U)))
      U = PRED(U)
      AGT(W) = U
      W = W + 1
    Loop
    U = Val(CMBNODES.Text) - 1
    Do While (PRED(U) >= 0)
      FLOW(PRED(U), U) = FLOW(PRED(U), U) + INCRE
      FLOW(U, PRED(U)) = FLOW(U, PRED(U)) - INCRE
      U = PRED(U)
    Loop
    MAX_FLOW = MAX_FLOW + INCRE
    SKEW(L) = INCRE
    L = L + 1
  Loop
ElseIf CMBSEARCH.ListIndex = 1 Then
  Do While (DFS(Source, SINK) = 2)
    INCRE = 1000000
    U = Val(CMBNODES.Text) - 1
    Do While (PRED(U) >= 0)
      INCRE = MINI(INCRE, (CAPACITY(PRED(U), U) - FLOW(PRED(U), U)))
      U = PRED(U)
      AGT(W) = U
   W = W + 1
    Loop
    U = Val(CMBNODES.Text) - 1
    Do While (PRED(U) >= 0)
      FLOW(PRED(U), U) = FLOW(PRED(U), U) + INCRE
      FLOW(U, PRED(U)) = FLOW(U, PRED(U)) - INCRE
```

```
        U = PRED(U)
      Loop
      MAX_FLOW = MAX_FLOW + INCRE
      SKEW(L) = INCRE
    L = L + 1
   Loop
  End If
  T2 = DateTime.Timer
  LBLTIME.Caption = Round(Abs(T2 - T1), 8) * 100
  AUG_PATH
  PATH_FLOW
  FF = MAX_FLOW
End Function
```

5.4.4. Approach

Edmond-Karp maximum flow finding algorithm conceptually is divided into three portions, as:

i. **Initialization and constraints**

First, initialize the flow *f* in the *G = (v.u)* in such a way that;

$$\sum_{v \in V} f(u,v) = f(u,V) = 0 \text{ for all } u \neq s,t$$

It means that all the flows *f* from *u* to *v* (where $u \neq s,t$) will be initialized by *0* and constraints that satisfy the Flow $f : V \, x \, V \rightarrow R$ are:

Capacity	:	$f(u,v) \leq c(u,v)$ *for all u,v*
Skew Symmetry	:	$f(u,v) = -f(v,u)$ *for all u,v*
Value of flow f	:	$\lvert f\rvert = f(s,V) = f(V,t)$

ii. **Searching for augmenting paths in residual network of vertices/nodes**

Edmond-Karp Algorithm searches for the shortest augmenting paths from source s to sink t by using Breadth First Searching technique (BFS), which makes its performance more efficient than Ford Fulkerson Method [6].

Using BFS strategy firstly, the root node (source node) is expanded, and then all the nodes generated by the root node are expanded next and their successors and so on. In general, all the nodes at depth *d* in the search tree are expanded before the nodes at depth *d+1*. BFS can compute d[v] = shortest-path distance from source s to edge v, in terms of minimum number of edges from s to v.

iii. Calculating Maximum Flow

The main objective of algorithm is to get the maximize value of the flow |f|. Therefore, after each iteration or searching augmenting path of the accessed flow f should be added in Flow |f| as:

$$|f| = \sum_{v \in V} f(s,v) = f(s,V) = f(V,t)$$

5.5. Complexity

By adding the flows of augmenting paths to the maximum flow already established in the graph, the maximum flow will be reached when no more flows of augmenting paths can be found in the graph. However, there is no certainty that this situation will ever be reached, so the best scenario is when it is guaranteed that answer will be correct if an algorithm terminates. If the algorithm runs forever, the flow might not even converge towards the maximum flow.

This situation only occurs with irrational flow values. When the capacities are integers, the runtime of Edmond Karp algorithm is bounded by O(E*f), where E is the number of edges in the graph and f is the maximum flow in the graph. This is because each augmenting path can be found in O(E) time and increases the flow by an integer amount which is at least 1.

Example:

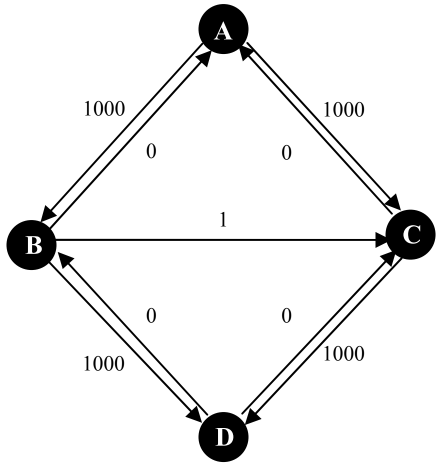


FIGURE – 5.1: Complexity

According to figure-5.1, a Graph contains 4 nodes, where A represents the source node and D represents the sink node. The inefficient behaviour of the Ford-Fulkerson Method can be observed by executing the method on the graph (shown in figure-5.1). Ford Fulkerson Method uses depth-first searching (DFS) technique for the searching of augmenting paths in the residual network [2],[5]. The augmenting paths are found with a depth-first-search, where neighbouring nodes are visited in alphabetical order (DFS searches the deepest nodes first) and find minimum capacity edge on priority basis. So it picks first path as A→B→C→D, which has the minimum capacity of 1. Second augmenting path is selected as A→C→B→D, which again contains minimum capacity flow 1 due to symmetry of the edge (C,B). This process is continued till no augmenting path is found in the directed graph.

This example shows the worst-case behavior of the Ford Fulkerson Method. In each step, only a flow of 1 is sent across the network. At the end, after 2000 augmenting paths maximum flow is calculated, which is 2000. On the other hand in Edmond Karp Algorithm, this utilizes the breadth-first search technique [6] for searching of 2 augmenting paths, to find out the maximum flow of 2000.

5.6. Busy Node

Busy node indicates that specific node in the network (excluding source and destination nodes), which exists in maximum number of augmenting paths of the directed graph. If two or more nodes exist in equal number of augmenting paths then criteria for selection of busy node depends upon the share in maximum flow of these nodes. It means the node which contains maximum share of the flow will be declared as busy node in the network. It can be explained with the help of an example (Graph is adopted from [8]).

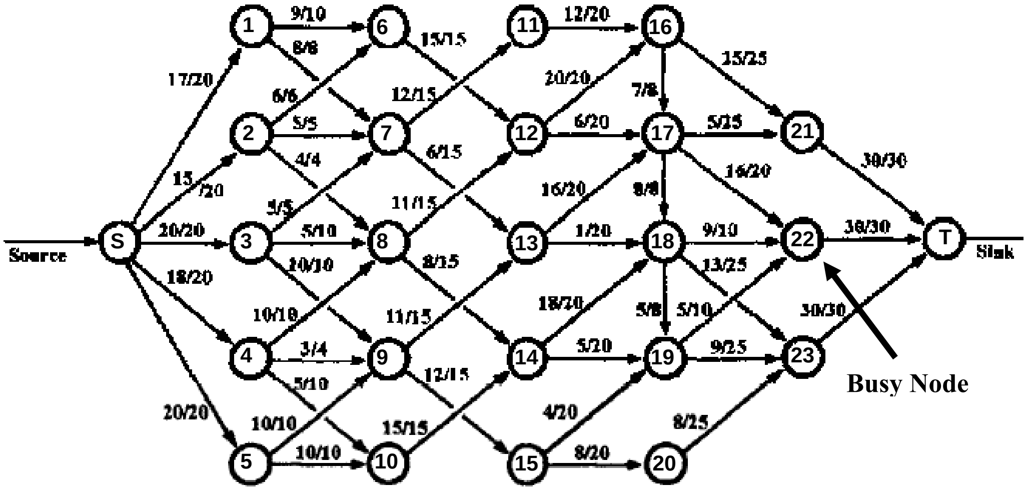


Figure – 5.2: Busy node Identification

The concept of busy node can be understand by implementing the Edmond-Karp algorithm on graph as shown in figure-5.2, which contains 25 vertices (V) with 80 edges (E). After execution, it is observed that 23 augmenting paths are existed in the graph with maximum flow of 90. According to the definition of crucial node (as mentioned above), vertex number 22 exist in 10 augmenting paths and vertex 17 exists in 9 augmenting paths with maximum flow share of 30 and 29 respectively. It indicates that vertex 22 is the busy node in the graph. The concept of busy node can be utilized for optimization of many image processing and network based applications.

CHAPTER 6

Simulation Design and Implementation

6.1. Introduction

The goal of this research is to provide an experimental analysis of Edmond Karp Maximum Flow Finding Algorithm. It compares the running time of given datasets as well as visual presentation of the outputs produced by simulation. The focus of the Edmond Karp algorithm is to find optimal solution for the maximum flow problem.

6.2. Design and Implementation

It is most important to understand the concept of the algorithm before its successful implementation. First of all, understand the concept of the Edmond Karp maximum flow finding algorithm. It establishes a number of algorithmic techniques: augmenting paths, residual networks, and cuts. There are many applications that benefit from this solution, including network routing, highway design, path finding for multiple units, and circuit design. The Edmond Karp algorithm builds on algorithms and data structures that can be observed as breadth-first search, queues (used in BFS), and graphs.

The simulator for maximum flow finding algorithm has been designed with a view to develop a software tool, which can be used for the study and evaluation of maximum flow in the network graph. This software has been developed as a comprehensive software package, which runs a simulation, generates useful data to be used for performance evaluation of algorithm and provides a user friendly environment. Software design strategy is functional oriented and design is modular in nature. The system is designed to run on a personal Computer as a windows application. The system is required to simulate creation of directed graphs and execution of the algorithm. It should maintain data of the created graphs in data files. It is also required to record the outputs of the simulation. The system should use the data to compute algorithm evaluation parameters and to ascertain behavior of the algorithm. A user friendly and mouse driven graphical user interface (GUI) to be integrated, in order to provide a user the opportunity to run the algorithm from the displayed menu as shown in figure-6.5.

6.2.1. Design of Edmond Karp Algorithm

In Edmond Karp Algorithm first initialize the flows for all the edges as:

Initialize the flow f in the $G = (v,u)$ in such a way that;

$$\sum_{v \in V} f(u,v) = f(u,V) = 0 \text{ for all } u \neq s,t$$

After the initialization of the flows, algorithm searches for the augmenting paths in the residual network, without violating the constraints of the algorithm as $c_f(u,v) \leq c(u,v)$, where u,v ≠ s,t. Edmond Karp algorithm utilizes Breadth-first Searching algorithm to search augmenting paths in residual network. With each augmenting path algorithm calculate the minimum flow capacity of the path through minimization function. At the end, it updates the flows for all the edges and returns the final calculated maximum flow |f|. The module of the Edmond Karp algorithm in the simulation is given below:

```
Public Function FF(Source As Integer, SINK As Integer) As Long
  Do While (BFS(Source, SINK) = 2)
       INCRE = 1000000
       U = Val(CMBNODES.Text) - 1
       Do While (PRED(U) >= 0)
         INCRE = MINI(INCRE, (CAPACITY(PRED(U), U) - FLOW(PRED(U), U)))
         U = PRED(U)
         AGT(W) = U    W = W + 1
       Loop
       Do While (PRED(U) >= 0)
         FLOW(PRED(U), U) = FLOW(PRED(U), U) + INCRE
         FLOW(U, PRED(U)) = FLOW(U, PRED(U)) - INCRE
         U = PRED(U)
       Loop
       MAX_FLOW = MAX_FLOW + INCRE
       SKEW(L) = INCRE     L = L + 1
    Loop
   LBLTIME.Caption = Round(Abs(T2 - T1), 8) * 100
   FF = MAX_FLOW
End Function
```

Working mechanism of the Edmond Karp algorithm is also demonstrated by the following flow chart (Figure-6.1).

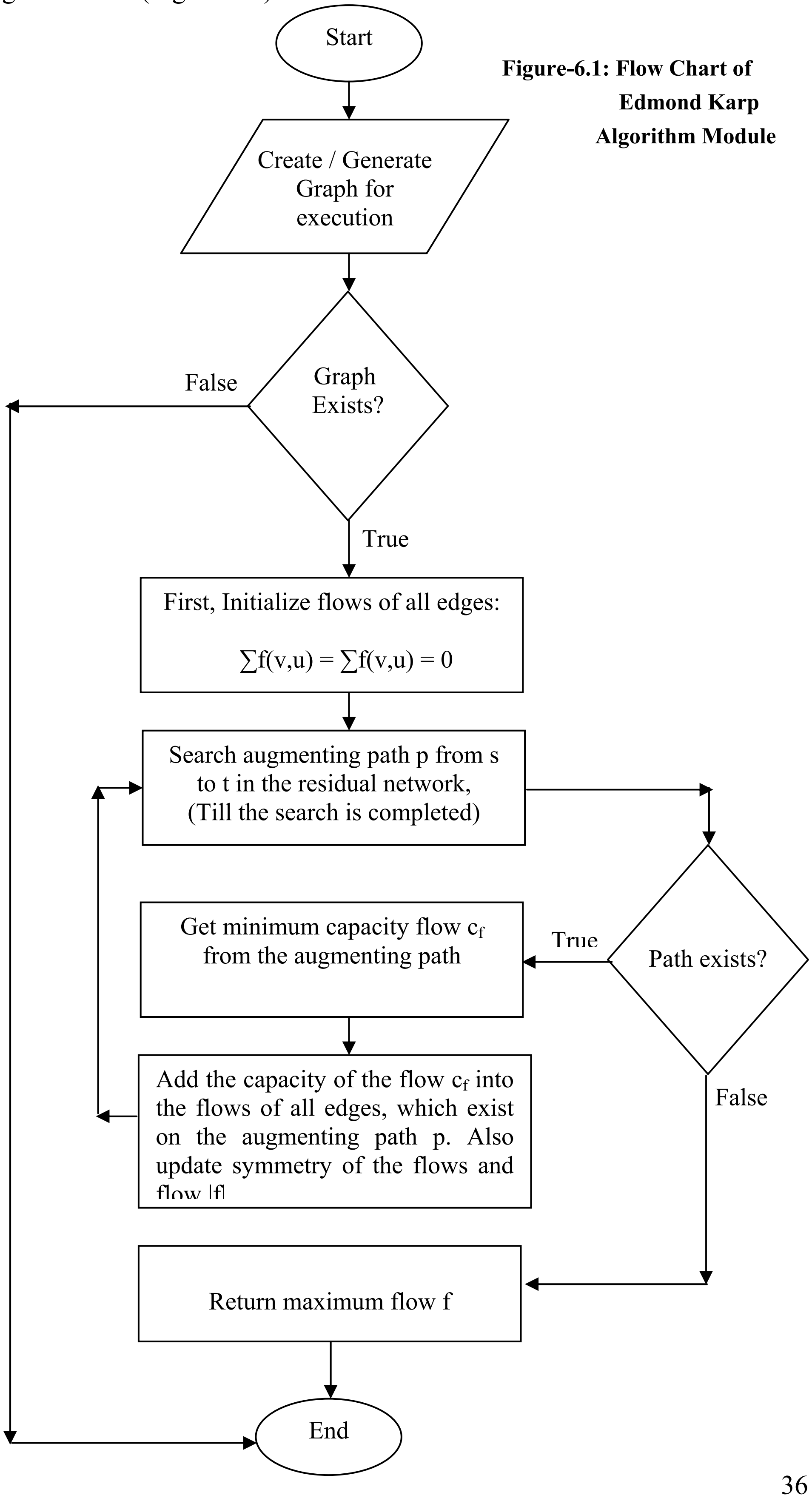

Start
Create / Generate Graph for execution
Graph Exists?
False
True
First, Initialize flows of all edges:
∑f(v,u) = ∑f(v,u) = 0
Search augmenting path p from s to t in the residual network, (Till the search is completed)
Path exists?
True
Get minimum capacity flow c_f from the augmenting path
Add the capacity of the flow c_f into the flows of all edges, which exist on the augmenting path p. Also update symmetry of the flows and flow |f|
False
Return maximum flow f
End

Figure-6.1: Flow Chart of Edmond Karp Algorithm Module

6.2.2. Working Mechanism of Breadth-first Search (BFS) in Simulation

Edmond Karp algorithm uses the Breadth-first Search (BFS) technique to find out the possible augmenting paths in the residual network of the graph. BFS technique utilizes queue structure to store and retrieve the nodes. For this purpose, it performs with the special functions ENQUEUE and DEQUEUE in the simulation. The work flow of the BFS is also explained through flow chart as shown in the figure-6.3. The module for BFS in the simulation is mentioned below:

- *Main module of the Breadth-first search in Simulation:*

```
Public Function BFS(START As Integer, TARGET As Integer) As Long        ‘BFS function for search
  ‘
  Dim U, V As Integer                         ‘COUNTER VARIABLES

  For U = 0 To Val(CMBNODES.Text) – 1         ‘INITIALIZATION OF NODES COLOR FLAG
    COLORS(U) = 0
  Next U

  HEAD = 0
  TAIL = 0

  ENQUEUE (START)

  PRED(START) = -1

  Do While (HEAD <> TAIL)                     ‘EXECUTE TILL ALL NODES ARE VISITED
    U = DEQUEUE()
    For V = 0 To Val(CMBNODES.Text) – 1
                                              ‘AGLORITHM CONSTRAINT IS CHECKED HERE.
      If COLORS(V) = 0 And (CAPACITY(U, V) - FLOW(U, V)) > 0 Then
        ENQUEUE (V)
        PRED(V) = U
      End If
    Next V
  Loop

  If COLORS(TARGET) = 2 Then                  ‘IF ALL NODES ARE VISITED
    BFS = COLORS(TARGET)                      ' RETURN VALUE
  End If

 End Function
```

- *Form a one-element queue consisting of the root node.*

Until the queue is empty or the goal has been reached, it determines if the first element in the queue as the goal node do nothing (or may stop depending on the situation). If the first element is not the goal node, remove the first element from the queue and add the first element's children (if any) to the **tail** of the queue. ENQUEUE and DEQUEUE procedures are used to add and remove node from the queue, which are mentioned as under:

```
Sub ENQUEUE(X As Integer)              ‘Procedure for add nodes in the Queue.
‘
   Q(TAIL) = X
   TAIL = TAIL + 1
   COLORS(X) = 1       'GRAY
‘
End Sub
```

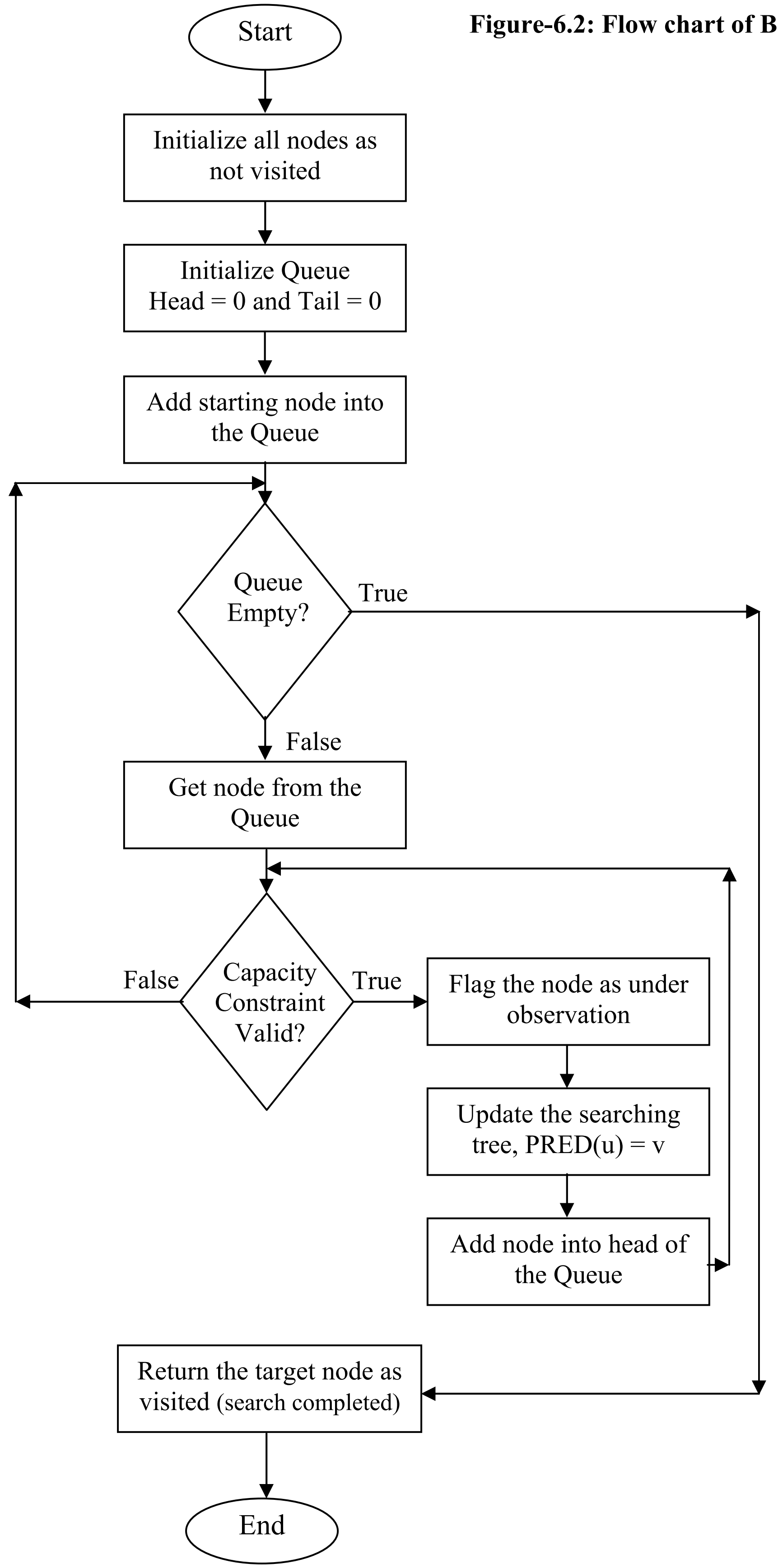
```
Public Function DEQUEUE() As Long      ‘Function for pop-up node from the Queue.
‘
  Dim X As Integer
   X = Q(HEAD)
   HEAD = HEAD + 1
   COLORS(X) = 2                           ' BLACK
   DEQUEUE = X                             ' RETURN VALUE
‘
End Function
```

- *If the goal node has been found, announce success, otherwise announce failure as.*

```
If COLORS(TARGET) = 2 Then
     BFS = COLORS(TARGET)                  ' RETURN VALUE – Target node reached.
  End If
```

BFS function can be explained through flow chart, shown in figure-6.2

Figure-6.2: Flow chart of BFS



Start
Initialize all nodes as not visited
Initialize Queue Head = 0 and Tail = 0
Add starting node into the Queue
Queue Empty?
True
False
Get node from the Queue
Capacity Constraint Valid?
False
True
Flag the node as under observation
Update the searching tree, PRED(u) = v
Add node into head of the Queue
Return the target node as visited (search completed)
End

6.2.3. Minimization function in Simulation

Edmond Karp Algorithm searches the augmenting path in the residual network graph. If algorithm gets a path p from s to t, then it tries to get the minimum value of the capacity flow c_f to maintain the constraint validation of the algorithm.

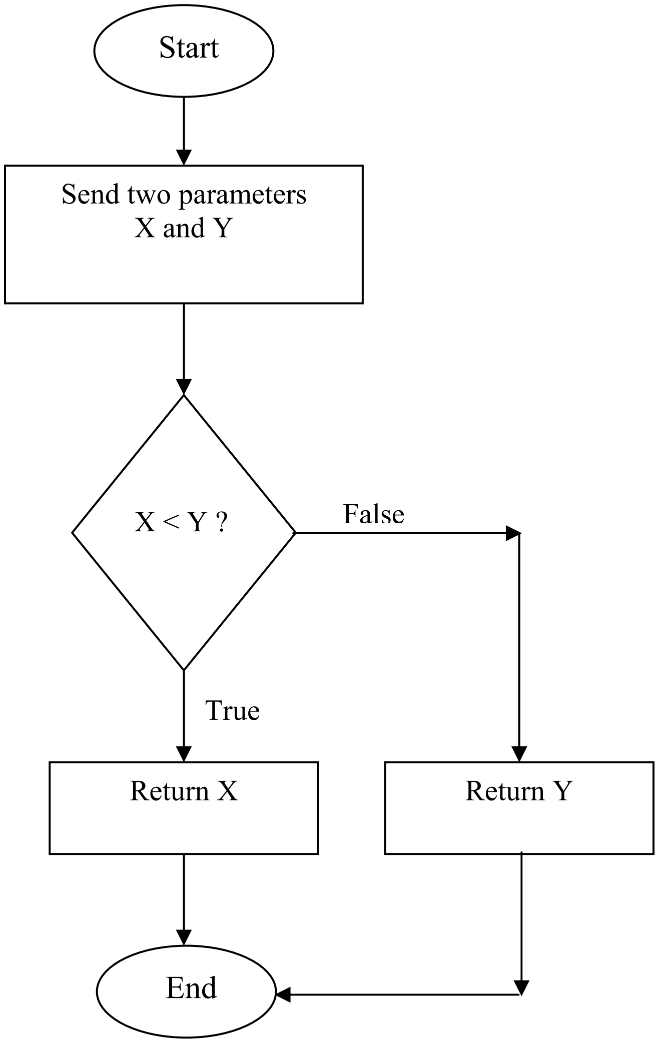


Figure – 6.3: Flow chart of minimization function

According to the flow chart mentioned in figure-6.3, two parameters (X and Y) are sent to the function. The function checks the minimum value between these two parameters and stops its execution with return of the minimum value.

6.2.4. Working of the Simulation

Simulator is designed on GUI based environment, so the user can easily handle and control its operations. The flow chart of the simulator is shown in the figure-6.4. According to the flow chart of the simulator, it is started with main window, where user has three main options for the operations:

a. New
b. Open
c. Result Sheet

a. New: This option is used to create or generate the network graph on the graph canvas. New option is further divided into sub-options, one is to select the number of nodes from the list and draw the graph on the graph canvas, and other is used to generate the graph through 'Auto Generation Interface'. When the algorithm is executed on the graph, it produces desired outputs as:

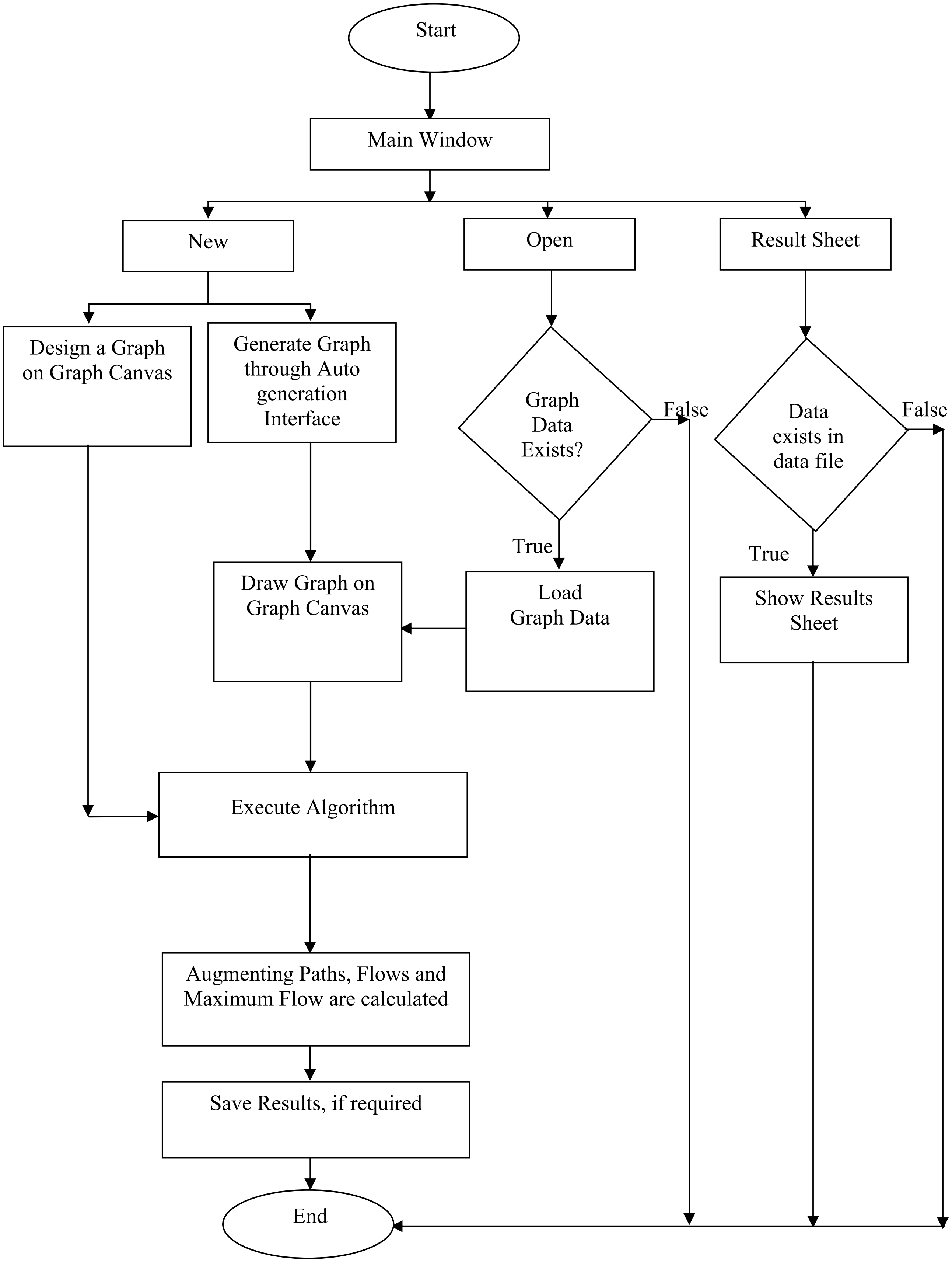
- Augmenting paths (with their respective flows)
- Execution time (in milliseconds)
- Identify busy node
- Value of maximum flow of the graph

The data of Graphs and experiment results can be saved in the data files (if required) for the analysis purpose.

b. Open: It is used to open the existing saved graphs from the data files. If graph data exists then it is loaded into the memory. Otherwise it is ended with error message. The rest of the procedure is the same as performed in the 'New' option.

c. Result Sheet: This option is used for showing the experimental results as well as for saving of them in result data file. If the proper results data file does not exist, then it will show the error message and ends.

Figure-6.4. - Flow Chart of Simulation


Start
Main Window
New
Open
Result Sheet
Design a Graph on Graph Canvas
Generate Graph through Auto generation Interface
Graph Data Exists?
False
True
Data exists in data file
False
True
Draw Graph on Graph Canvas
Load Graph Data
Show Results Sheet
Execute Algorithm
Augmenting Paths, Flows and Maximum Flow are calculated
Save Results, if required
End

6.3. Performance of the Simulation

The performance of the simulation depends on following parameters, which are as under:

- Selection of graph datasets
- Hardware/Machine (where simulation is executed)
- Platform (for visual support)
- Developing tools.

- **Selection of graphs datasets:** Simulation is designed to run Edmond Karp Algorithm under a set of given conditions in the form of input data. These inputs are also used to evaluate the performance of the simulation in respect of workload. For this purpose three different datasets are selected as input data, which are mentioned below:

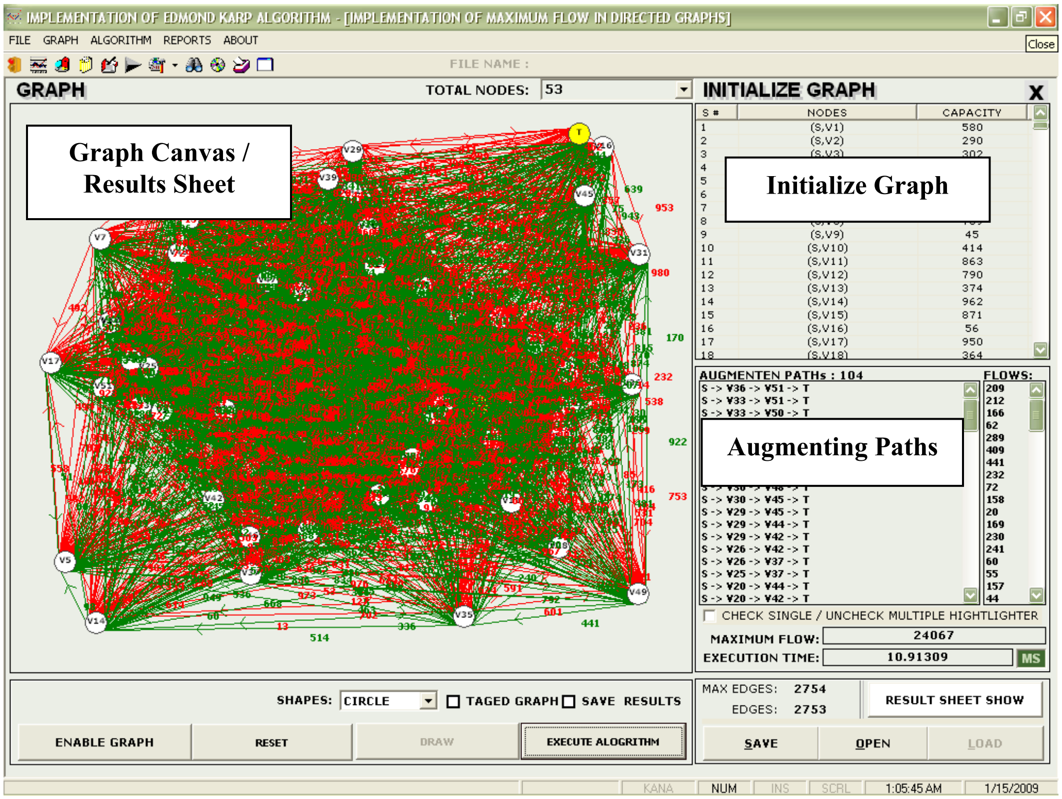
    - User-defined Graphs
    - Auto generated Graphs (with random number of Edges)
    - Auto generated Graphs (with maximum number of Edges)

- **Hardware:** Performance of the simulation is also depends on the hardware where it is executed. Hardware includes execution machine, which consists of processor, RAM and space etc. Execution time and efficient visual support depends on the fast and high speed processor, large memory capacity and huge space. Onward follow experiment results have been collected by Simulator, which is executed on 1.7 GHz Processor, 256 MB RAM and 20 GB hard disk based-machine.
- **Platform:** Platform provides the supportive services for the Simulator. It can also affect the performance of the simulation. Platform includes operating system, where simulation runs like Windows 9x/XP/2000. GUI based on operation systems (OS) has more graphical support than other platforms. The simulation is tested on the Windows XP/2000. The experiments results are mentioned in this report are collected on the Window XP based operating system.

- **Development Tool:** Includes Visual Basic 6.0, which is used to implement the Edmond Karp maximum flow finding algorithm. File system is used for the preservation of experiment results log file. These log files and graph data sheets are saved in a plain text file.

6.4. Simulation Features

Simulation is designed to run Edmond Karp Algorithm under a set of given conditions in the form of input data. Simulator is provided a user friendly and mouse driven graphical user interface (GUI) to be integrated, in order to provide a user the opportunity to run the algorithm from the displayed menu. Simulator design is based on menu-driven interface as shown in figure-6.5.



TOTAL NODES: 53

S #	NODES	CAPACITY
1	(S,V1)	580
2	(S,V2)	290
3	(S,V3)	302
4	[illegible]	[illegible]
5	[illegible]	[illegible]
6	[illegible]	[illegible]
7	[illegible]	[illegible]
8	[illegible]	[illegible]
9	(S,V9)	45
10	(S,V10)	414
11	(S,V11)	863
12	(S,V12)	790
13	(S,V13)	374
14	(S,V14)	962
15	(S,V15)	871
16	(S,V16)	56
17	(S,V17)	950
18	(S,V18)	364

AUGMENTEN PATHs : 104	FLOWS:
S -> V36 -> V51 -> T	209
S -> V33 -> V51 -> T	212
S -> V33 -> V50 -> T	166
[illegible]	62
[illegible]	289
[illegible]	409
[illegible]	441
[illegible]	232
[illegible]	72
S -> V30 -> V45 -> T	158
S -> V29 -> V45 -> T	20
S -> V29 -> V44 -> T	169
S -> V29 -> V42 -> T	230
S -> V26 -> V42 -> T	241
S -> V26 -> V37 -> T	60
S -> V25 -> V37 -> T	55
S -> V20 -> V44 -> T	157
S -> V20 -> V42 -> T	44

□ CHECK SINGLE / UNCHECK MULTIPLE HIGHTLIGHTER

MAXIMUM FLOW: 24067

EXECUTION TIME: 10.91309 MS

SHAPES: CIRCLE □ TAGED GRAPH □ SAVE RESULTS

MAX EDGES: 2754

EDGES: 2753

1:05:45 AM 1/15/2009

Figure 6.5 : Main Window of the Simulator

The main window of the simulator is divided into separate portions according to their functions, which are described as:

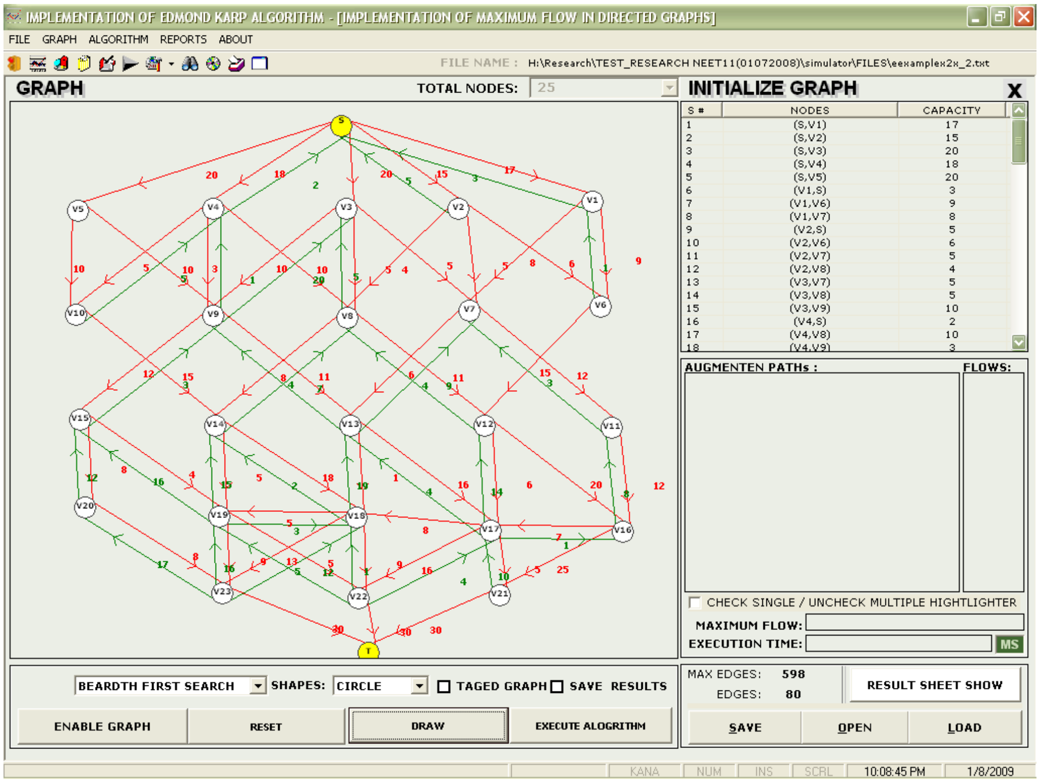
- **Graph Canvas / Result Sheet** is used to create graphs or display the Results Sheet. Graphs can be designed by user or generated by the 'Auto Generation Interface'.
- **Initialize Graph** list is populated when graph data is loaded into the memory. It shows the list of all present edges in the graph along with their respective capacities.
- **Augmenting Paths** list is used to indicate all possible augmenting paths existing in the graph. Augmenting Paths list also shows the respective flows of the available paths.

6.4.1. Input of the System

System takes input data in three different methods, which are explained below:

6.4.1.1. Input from User:

In this method, user creates graph on the Graph Canvas with selected number of nodes. These graphs contain two special nodes, which are highlighted in yellow colour as shown in figure – 6.6. The system also ensures the correctness of the data by imposing checks on input values. The capacities of the edges should be some positive values. User can program the system to simulate the hundreds of nodes-based graphs.



IMPLEMENTATION OF EDMOND KARP ALGORITHM - [IMPLEMENTATION OF MAXIMUM FLOW IN DIRECTED GRAPHS]

FILE NAME : H:\Research\TEST_RESEARCH NEET11(01072008)\simulator\FILES\eexamplex2x_2.txt

GRAPH

TOTAL NODES: 25

INITIALIZE GRAPH

S #	NODES	CAPACITY
1	(S,V1)	17
2	(S,V2)	15
3	(S,V3)	20
4	(S,V4)	18
5	(S,V5)	20
6	(V1,S)	3
7	(V1,V6)	9
8	(V1,V7)	8
9	(V2,S)	5
10	(V2,V6)	6
11	(V2,V7)	5
12	(V2,V8)	4
13	(V3,V7)	5
14	(V3,V8)	5
15	(V3,V9)	10
16	(V4,S)	2
17	(V4,V8)	10
18	(V4,V9)	3

AUGMENTEN PATHs :

FLOWS:

☐ CHECK SINGLE / UNCHECK MULTIPLE HIGHTLIGHTER

MAXIMUM FLOW:

EXECUTION TIME: MS

BEARDTH FIRST SEARCH SHAPES: CIRCLE ☐ TAGED GRAPH ☐ SAVE RESULTS

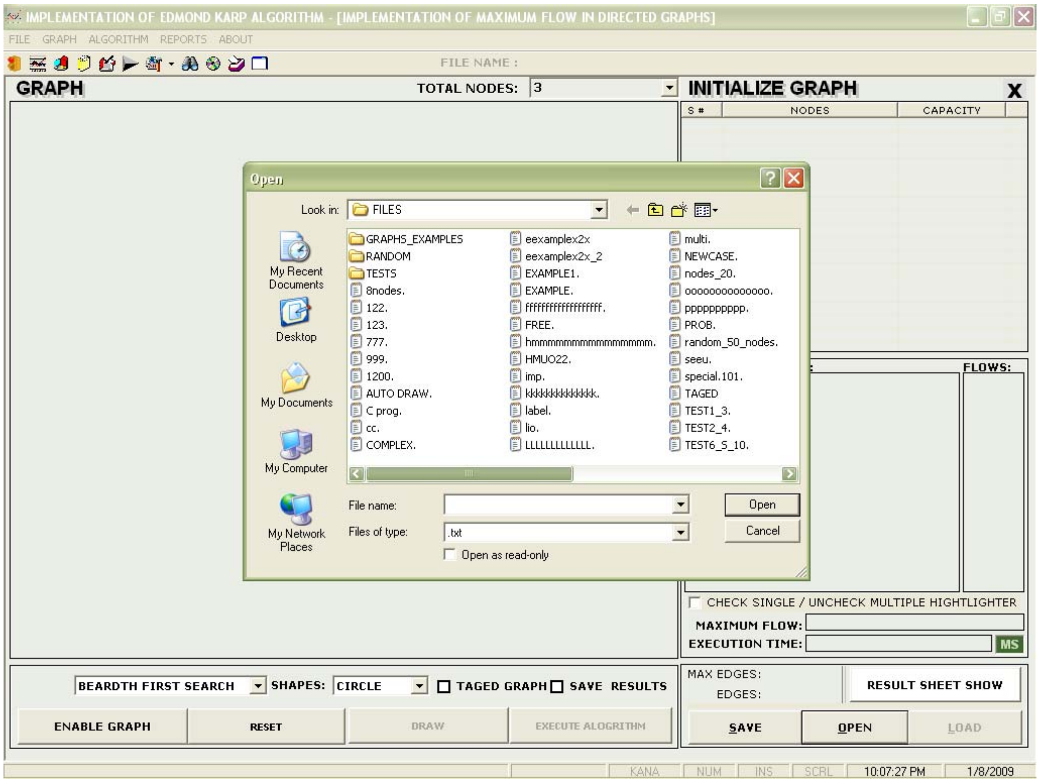
MAX EDGES: 598

EDGES: 80

Figure – 6.6: Draw a Graph from Input

6.4.1.2. Input from Saved Files

System also takes data from the saved graph data files as shown in Figure-6.7. These data files are based on plain text files structure, which contain the capacities of the edges and coordinates of the nodes. If the correct graph data file is selected from the files list, then it is loaded into the memory and initialize Graph list of the main window. Otherwise error message (Graph data not found) will be popped-up on the screen.



TOTAL NODES: 3

Look in: FILES

Files of type: .txt

BEARDTH FIRST SEARCH

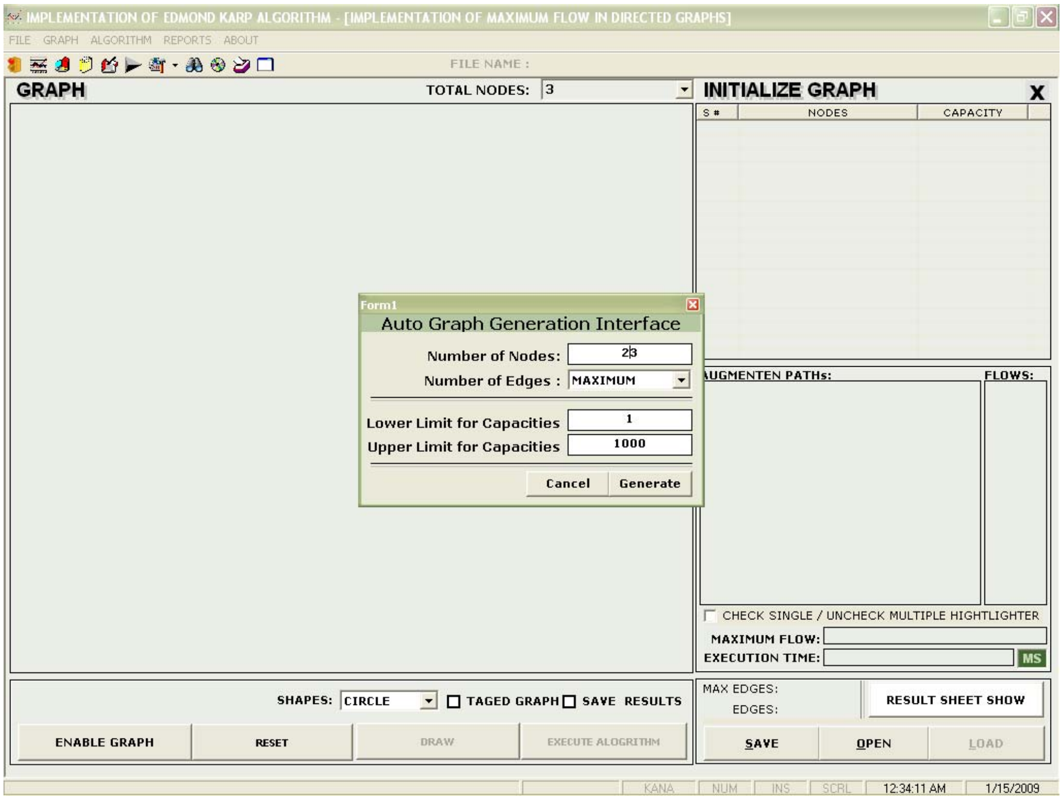
SHAPES: CIRCLE

Figure – 6.7: Open a Graph from a data file.

6.4.1.3. Input from System

The performance of the Edmond Karp Algorithm is also tested on random generated datasets. The inputs of data graphs are generated by the system through "Auto Generation Interface" as shown in figure-6.8. In auto generation interface, user defines some parameters as under:

- Number of nodes
- Selection criteria of edges (maximum or random number of edges)
- Range of the edges capacities.



IMPLEMENTATION OF EDMOND KARP ALGORITHM - [IMPLEMENTATION OF MAXIMUM FLOW IN DIRECTED GRAPHS]

TOTAL NODES: 3

Form1

Auto Graph Generation Interface

Number of Nodes: 23

Number of Edges : MAXIMUM

Lower Limit for Capacities 1

Upper Limit for Capacities 1000

SHAPES: CIRCLE

Figure – 6.8: Interface for Random Generation of Graph

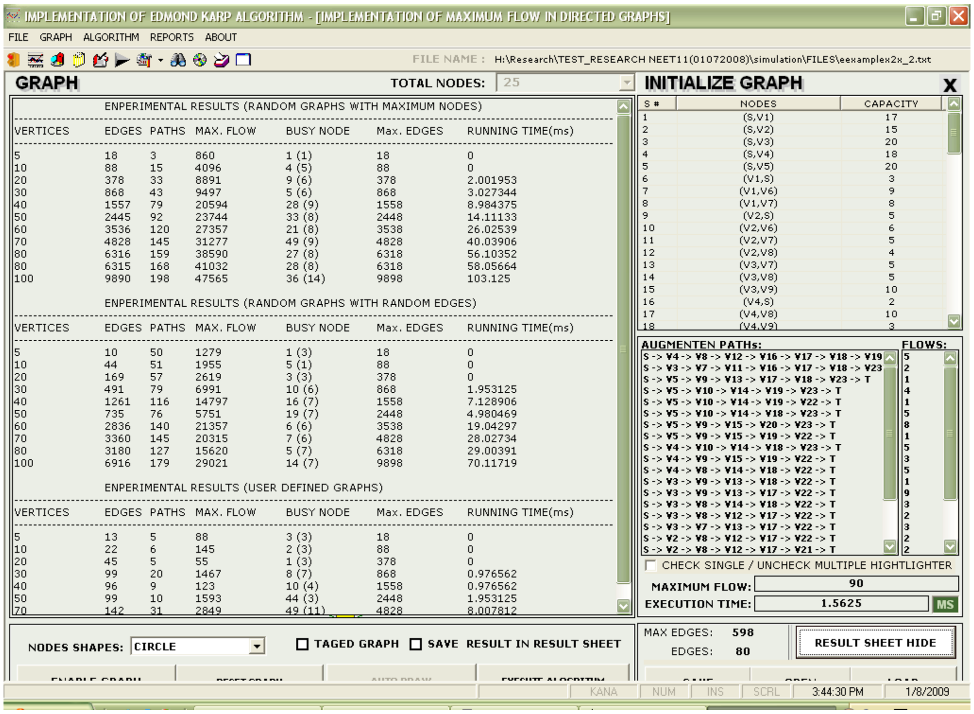
6.4.2. Output of System

The main objective of the simulation is to produce outputs for performance evaluation of the Edmond Karp Maximum Flow Finding Algorithm. System can preserve simulation data at run time in data files. A user can store complete graph history as well as performance factor in the data file. The data is essential for:

- The performance evaluation of the algorithm
- Detailed analysis of the algorithm for the purpose of research
- User can run any saved simulation any time

There are two types of data files which are used to save simulation data:

1. User can save the executed outputs of the simulation in the form of Results Sheet which contains the tabular presentation of the results for analysis purpose, as shown in the figure-6.9.


IMPLEMENTATION OF EDMOND KARP ALGORITHM - [IMPLEMENTATION OF MAXIMUM FLOW IN DIRECTED GRAPHS]
FILE GRAPH ALGORITHM REPORTS ABOUT
GRAPH
TOTAL NODES:
INITIALIZE GRAPH
X
NODES SHAPES:
TAGED GRAPH
SAVE RESULT IN RESULT SHEET
RESULT SHEET HIDE
KANA NUM INS SCRL

FILE NAME : H:\Research\TEST_RESEARCH NEET11(01072008)\simulation\FILES\eexamplex2x_2.txt

TOTAL NODES: 25

```
                    ENPERIMENTAL RESULTS (RANDOM GRAPHS WITH MAXIMUM NODES)
---------------------------------------------------------------------------------------
VERTICES            EDGES  PATHS  MAX. FLOW    BUSY NODE    Max. EDGES   RUNNING TIME(ms)
---------------------------------------------------------------------------------------
5                   18     3      860          1 (1)        18           0
10                  88     15     4096         4 (5)        88           0
20                  378    33     8891         9 (6)        378          2.001953
30                  868    43     9497         5 (6)        868          3.027344
40                  1557   79     20594        28 (9)       1558         8.984375
50                  2445   92     23744        33 (8)       2448         14.11133
60                  3536   120    27357        21 (8)       3538         26.02539
70                  4828   145    31277        49 (9)       4828         40.03906
80                  6316   159    38590        27 (8)       6318         56.10352
80                  6315   168    41032        28 (8)       6318         58.05664
100                 9890   198    47565        36 (14)      9898         103.125

                    ENPERIMENTAL RESULTS (RANDOM GRAPHS WITH RANDOM EDGES)
---------------------------------------------------------------------------------------
VERTICES            EDGES  PATHS  MAX. FLOW    BUSY NODE    Max. EDGES   RUNNING TIME(ms)
---------------------------------------------------------------------------------------
5                   10     50     1279         1 (3)        18           0
10                  44     51     1955         5 (1)        88           0
20                  169    57     2619         3 (3)        378          0
30                  491    79     6991         10 (6)       868          1.953125
40                  1261   116    14797        16 (7)       1558         7.128906
50                  735    76     5751         19 (7)       2448         4.980469
60                  2836   140    21357        6 (6)        3538         19.04297
70                  3360   145    20315        7 (6)        4828         28.02734
80                  3180   127    15620        5 (7)        6318         29.00391
100                 6916   179    29021        14 (7)       9898         70.11719

                    ENPERIMENTAL RESULTS (USER DEFINED GRAPHS)
---------------------------------------------------------------------------------------
VERTICES            EDGES  PATHS  MAX. FLOW    BUSY NODE    Max. EDGES   RUNNING TIME(ms)
---------------------------------------------------------------------------------------
5                   13     5      88           3 (3)        18           0
10                  22     6      145          2 (3)        88           0
20                  45     5      55           1 (3)        378          0
30                  99     20     1467         8 (7)        868          0.976562
40                  96     9      123          10 (4)       1558         0.976562
50                  99     10     1593         44 (3)       2448         1.953125
70                  142    31     2849         49 (11)      4828         8.007812
```

S #	NODES	CAPACITY
1	(S,V1)	17
2	(S,V2)	15
3	(S,V3)	20
4	(S,V4)	18
5	(S,V5)	20
6	(V1,S)	3
7	(V1,V6)	9
8	(V1,V7)	8
9	(V2,S)	5
10	(V2,V6)	6
11	(V2,V7)	5
12	(V2,V8)	4
13	(V3,V7)	5
14	(V3,V8)	5
15	(V3,V9)	10
16	(V4,S)	2
17	(V4,V8)	10
18	(V4,V9)	3

AUGMENTEN PATHs:	FLOWS:
S -> V4 -> V8 -> V12 -> V16 -> V17 -> V18 -> V19	5
S -> V3 -> V7 -> V11 -> V16 -> V17 -> V18 -> V23	2
S -> V5 -> V9 -> V13 -> V17 -> V18 -> V23 -> T	1
S -> V5 -> V10 -> V14 -> V19 -> V23 -> T	4
S -> V5 -> V10 -> V14 -> V19 -> V22 -> T	1
S -> V5 -> V10 -> V14 -> V18 -> V23 -> T	5
S -> V5 -> V9 -> V15 -> V20 -> V23 -> T	8
S -> V5 -> V9 -> V15 -> V19 -> V22 -> T	1
S -> V4 -> V10 -> V14 -> V18 -> V23 -> T	5
S -> V4 -> V9 -> V15 -> V19 -> V22 -> T	3
S -> V4 -> V8 -> V14 -> V18 -> V22 -> T	5
S -> V3 -> V9 -> V13 -> V18 -> V22 -> T	1
S -> V3 -> V9 -> V13 -> V17 -> V22 -> T	9
S -> V3 -> V8 -> V14 -> V18 -> V22 -> T	3
S -> V3 -> V8 -> V12 -> V17 -> V22 -> T	2
S -> V3 -> V7 -> V13 -> V17 -> V22 -> T	3
S -> V2 -> V8 -> V12 -> V17 -> V22 -> T	2
S -> V2 -> V8 -> V12 -> V17 -> V21 -> T	2

□ CHECK SINGLE / UNCHECK MULTIPLE HIGHTLIGHTER

MAXIMUM FLOW: 90

EXECUTION TIME: 1.5625 MS

NODES SHAPES: CIRCLE

□ TAGED GRAPH □ SAVE RESULT IN RESULT SHEET

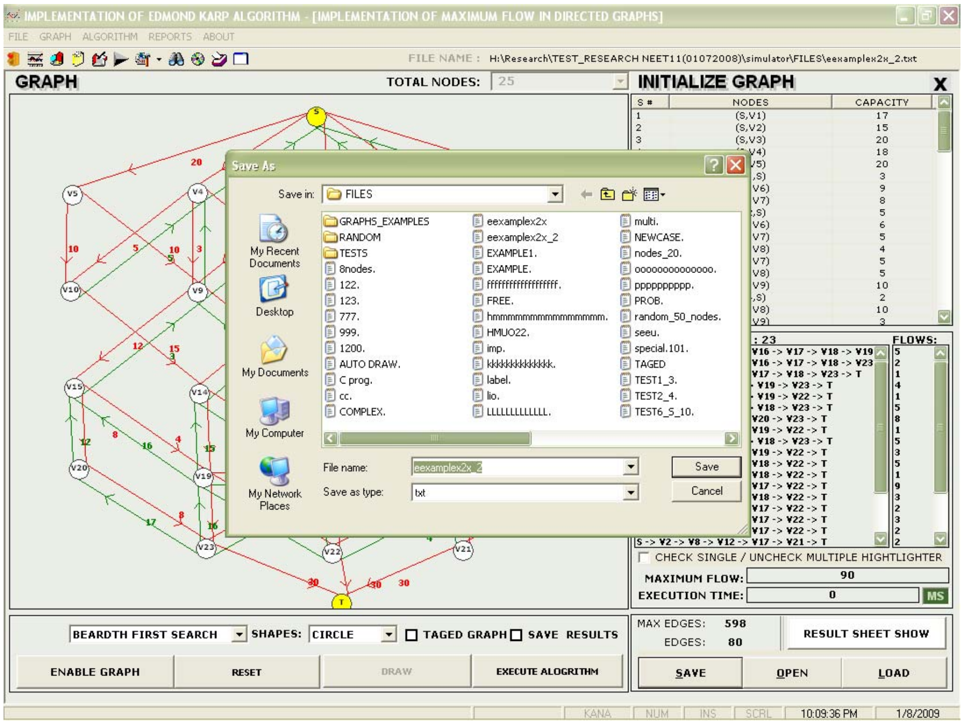
MAX EDGES: 598

EDGES: 80

3:44:30 PM 1/8/2009

Figure – 6.9: Experiment Result Sheet View (Results Log file)

2. Designed/Generated Graph data can also be stored into a data as shown in figure-6.10. Graph data contain the capacities of the edges alongwith the coordinates of the vertices.



FILE NAME : H:\Research\TEST_RESEARCH NEET11(01072008)\simulator\FILES\eexamplex2x_2.txt

TOTAL NODES: 25

Save in: FILES

File name: eexamplex2x_2

Save as type: txt

MAXIMUM FLOW: 90

EXECUTION TIME: 0 MS

BEARDTH FIRST SEARCH SHAPES: CIRCLE □ TAGED GRAPH □ SAVE RESULTS

MAX EDGES: 598

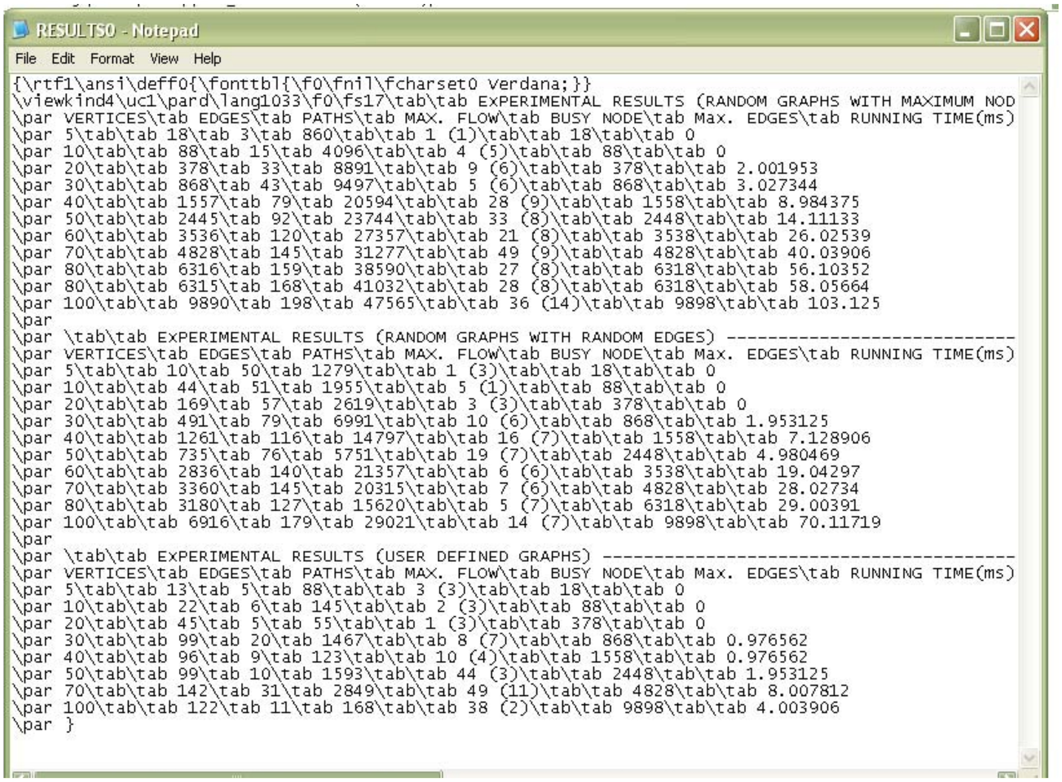
EDGES: 80

10:09:36 PM 1/8/2009

Figure – 6.10: Save the Graph in graph data file.

6.4.3. Data Storage

Simulator uses the file system to store the data of the created or generated graphs as well as the data of the result data file. File system helps to enhance the performance of the simulator. If database storage technique is compared with file system technique, then it is observed that database requires connectivity drivers, database engine and extra time to access the data from concerned database (it means system becomes depended). But in case of file system, it is easy to store and get the data from the text file without any kind of dependency. The structure of the file is shown in figure-6.11.



```
{\rtf1\ansi\deff0{\fonttbl{\f0\fnil\fcharset0 Verdana;}}
\viewkind4\uc1\pard\lang1033\f0\fs17\tab\tab EXPERIMENTAL RESULTS (RANDOM GRAPHS WITH MAXIMUM NOD
\par VERTICES\tab EDGES\tab PATHS\tab MAX. FLOW\tab BUSY NODE\tab Max. EDGES\tab RUNNING TIME(ms)
\par 5\tab\tab 18\tab 3\tab 860\tab\tab 1 (1)\tab\tab 18\tab\tab 0
\par 10\tab\tab 88\tab 15\tab 4096\tab\tab 4 (5)\tab\tab 88\tab\tab 0
\par 20\tab\tab 378\tab 33\tab 8891\tab\tab 9 (6)\tab\tab 378\tab\tab 2.001953
\par 30\tab\tab 868\tab 43\tab 9497\tab\tab 5 (6)\tab\tab 868\tab\tab 3.027344
\par 40\tab\tab 1557\tab 79\tab 20594\tab\tab 28 (9)\tab\tab 1558\tab\tab 8.984375
\par 50\tab\tab 2445\tab 92\tab 23744\tab\tab 33 (8)\tab\tab 2448\tab\tab 14.11133
\par 60\tab\tab 3536\tab 120\tab 27357\tab\tab 21 (8)\tab\tab 3538\tab\tab 26.02539
\par 70\tab\tab 4828\tab 145\tab 31277\tab\tab 49 (9)\tab\tab 4828\tab\tab 40.03906
\par 80\tab\tab 6316\tab 159\tab 38590\tab\tab 27 (8)\tab\tab 6318\tab\tab 56.10352
\par 80\tab\tab 6315\tab 168\tab 41032\tab\tab 28 (8)\tab\tab 6318\tab\tab 58.05664
\par 100\tab\tab 9890\tab 198\tab 47565\tab\tab 36 (14)\tab\tab 9898\tab\tab 103.125
\par
\par \tab\tab EXPERIMENTAL RESULTS (RANDOM GRAPHS WITH RANDOM EDGES) -----------------------------
\par VERTICES\tab EDGES\tab PATHS\tab MAX. FLOW\tab BUSY NODE\tab Max. EDGES\tab RUNNING TIME(ms)
\par 5\tab\tab 10\tab 50\tab 1279\tab\tab 1 (3)\tab\tab 18\tab\tab 0
\par 10\tab\tab 44\tab 51\tab 1955\tab\tab 5 (1)\tab\tab 88\tab\tab 0
\par 20\tab\tab 169\tab 57\tab 2619\tab\tab 3 (3)\tab\tab 378\tab\tab 0
\par 30\tab\tab 491\tab 79\tab 6991\tab\tab 10 (6)\tab\tab 868\tab\tab 1.953125
\par 40\tab\tab 1261\tab 116\tab 14797\tab\tab 16 (7)\tab\tab 1558\tab\tab 7.128906
\par 50\tab\tab 735\tab 76\tab 5751\tab\tab 19 (7)\tab\tab 2448\tab\tab 4.980469
\par 60\tab\tab 2836\tab 140\tab 21357\tab\tab 6 (6)\tab\tab 3538\tab\tab 19.04297
\par 70\tab\tab 3360\tab 145\tab 20315\tab\tab 7 (6)\tab\tab 4828\tab\tab 28.02734
\par 80\tab\tab 3180\tab 127\tab 15620\tab\tab 5 (7)\tab\tab 6318\tab\tab 29.00391
\par 100\tab\tab 6916\tab 179\tab 29021\tab\tab 14 (7)\tab\tab 9898\tab\tab 70.11719
\par
\par \tab\tab EXPERIMENTAL RESULTS (USER DEFINED GRAPHS) ----------------------------------------
\par VERTICES\tab EDGES\tab PATHS\tab MAX. FLOW\tab BUSY NODE\tab Max. EDGES\tab RUNNING TIME(ms)
\par 5\tab\tab 13\tab 5\tab 88\tab\tab 3 (3)\tab\tab 18\tab\tab 0
\par 10\tab\tab 22\tab 6\tab 145\tab\tab 2 (3)\tab\tab 88\tab\tab 0
\par 20\tab\tab 45\tab 5\tab 55\tab\tab 1 (3)\tab\tab 378\tab\tab 0
\par 30\tab\tab 99\tab 20\tab 1467\tab\tab 8 (7)\tab\tab 868\tab\tab 0.976562
\par 40\tab\tab 96\tab 9\tab 123\tab\tab 10 (4)\tab\tab 1558\tab\tab 0.976562
\par 50\tab\tab 99\tab 10\tab 1593\tab\tab 44 (3)\tab\tab 2448\tab\tab 1.953125
\par 70\tab\tab 142\tab 31\tab 2849\tab\tab 49 (11)\tab\tab 4828\tab\tab 8.007812
\par 100\tab\tab 122\tab 11\tab 168\tab\tab 38 (2)\tab\tab 9898\tab\tab 4.003906
\par }
```

Figure – 6.11: Text File for data storage

CHAPTER 7

RESULTS AND DISCUSSION

7.1. Results

This section contains the analysis of experimental results of Edmond Karp maximum flow finding algorithm. These results are prepared on the desktop (Window XP based platform) computer with a 1.7 GHz Pentium IV processor and 1GB of RAM.

7.1.1 Dataset Selection Criteria

The inputs used in experiments are combination of user-defined and auto generated datasets. These datasets are instances of the maximum flow optimization problem, and correspond to the sets of data forms.

In addition, auto generated datasets are further divided into two sub-datasets on the basis of number of edges (random and maximum). The simulator computes the number of augmenting paths existed in the different graph datasets. It also calculates the optimal solution of maximum flow by using Edmond Karp algorithm. The purpose of these datasets is to examine whether any underlying structure of the problems is affecting the relative running time of the algorithm. Cumulative Average Time is used to analyse the average running time of the algorithm in different data inputs. The data selection criteria can be defined as:

- User-Defined Graphs Data
- Auto Generated Graph Data (with Random Number of Edges)
- Auto Generated Graph Data (with Maximum Number of Edges)

7.1.2. Experiment Results

The results of experiments of datasets are mentioned below:

7.1.2.1. Experiment 1:

EXPERIMENT RESULTS (USER DEFINED GRAPHS)

S #	Number of Vertices	Number of Edges	Number of Paths	Maximum Flow	*Busy Node	Number of Maximum Edges	Running Time (ms)
1	5	13	5	88	3 (3)	18	0
2	10	22	6	145	2 (3)	88	0
3	20	45	5	55	1 (3)	378	0
4	30	99	20	1467	8 (7)	868	1.5625
5	40	106	14	358	10 (7)	1558	1.6075
6	50	121	21	270	30 (9)	2448	3.125
7	60	158	32	785	4 (17)	3538	4.6875
8	70	170	30	511	12 (12)	4828	6.25
9	100	205	23	1326	22 (4)	9898	7.2250

*Node (Number of maximum paths)

Table-7.1: Experiment Results of User-defined Graphs – Experiment 1

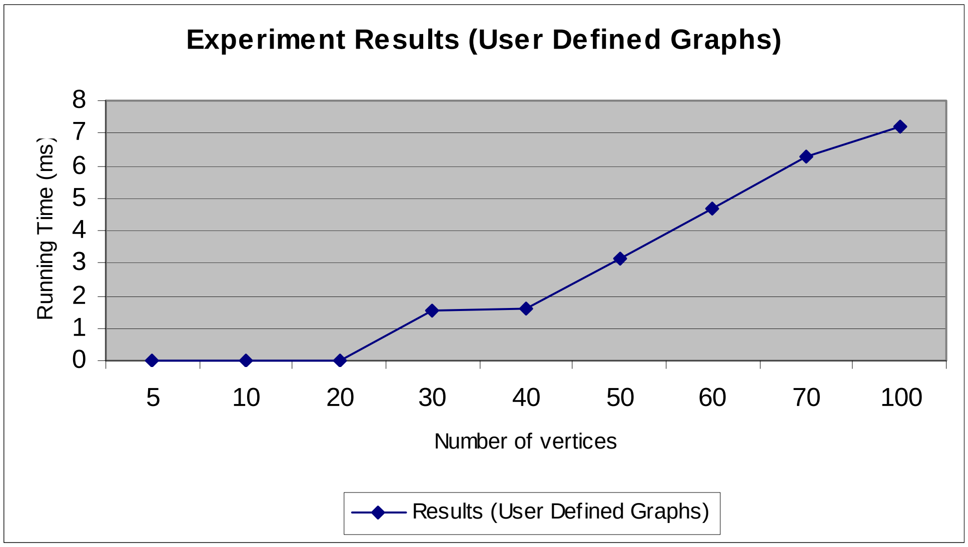


Figure-7.1: Graphical representation of User-defined Graphs – Experiment 1

EXPERIMENT RESULTS (AUTO GENERATED GRAPHS WITH RANDOM EDGES)

S #	Number of Vertices	Number of Edges	Number of Paths	Maximum Flow	*Busy Node	Number of Maximum Edges	Running Time (ms)
1	5	10	3	1279	1 (6)	18	0
2	10	44	4	1955	5 (1)	88	1.4625
3	20	164	13	2141	12 (12)	378	2.0625
4	30	497	36	8243	13 (66)	868	3.0255
5	40	1266	71	15157	38 (9)	1558	4.6875
6	50	1846	73	18095	2 (18)	2448	7.8125
7	60	2823	96	20697	4 (7)	3538	14.0625
8	70	3394	99	19473	4 (6)	4828	18.75
9	80	3499	87	22256	62 (8)	6318	20.3125
10	90	4394	119	23076	16 (8)	8008	32.8125
11	100	6899	154	31065	39 (16)	9898	43.75

* Node (Number of maximum paths)

Table-7.2: Experiment Results of Auto Generated Graphs with Random Edges)

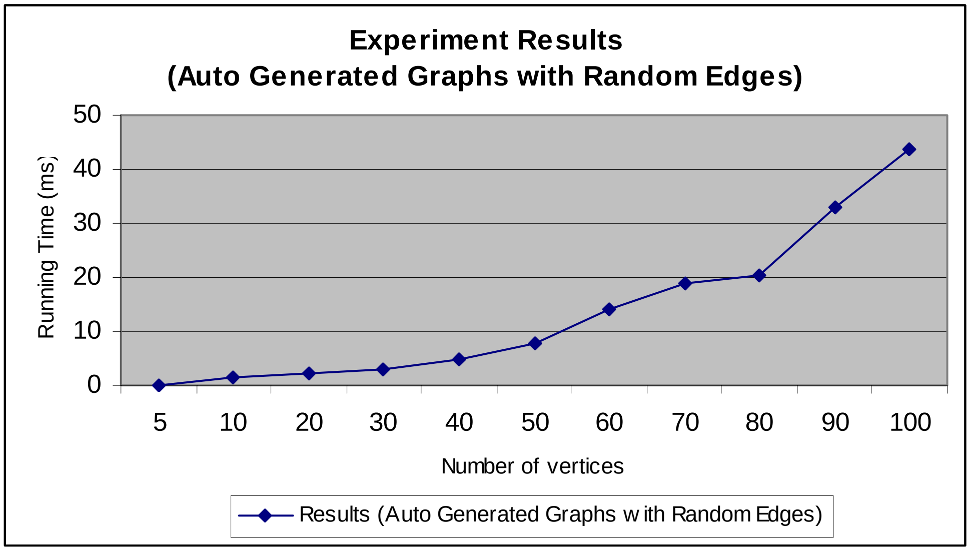


Figure-7.2: Graphical representation of Auto Generated Graphs with Random Edges

EXPERIMENTAL RESULTS (AUTO GENEREATED GRAPHS WITH MAXIMUM EDGES)

S #	Number of Vertices	Number of Edges	Number of Paths	Maximum Flow	*Busy Node	Number of Maximum Edges	Running Time (ms)
1	5	18	3	860	1 (1)	18	0
2	10	88	15	4096	4 (5)	88	0.5625
3	20	378	40	9147	14 (7)	378	1.5662
4	30	868	55	12015	20 (9)	868	3.125
5	40	1557	79	20594	28 (9)	1558	6.25
6	50	2446	100	23327	36 (7)	2448	10.9375
7	60	3536	109	24939	46 (6)	3538	15.625
8	70	4824	120	28325	25 (8)	4828	21.875
9	80	6315	172	39485	69 (9)	6318	39.0625
10	90	8005	172	42190	6 (8)	8008	46.875
11	100	9894	207	51651	2 (7)	9898	70.312

* Node (Number of maximum paths)

Table-7.3: Experiment Results of Auto Generated Graphs with Maximum Edges)

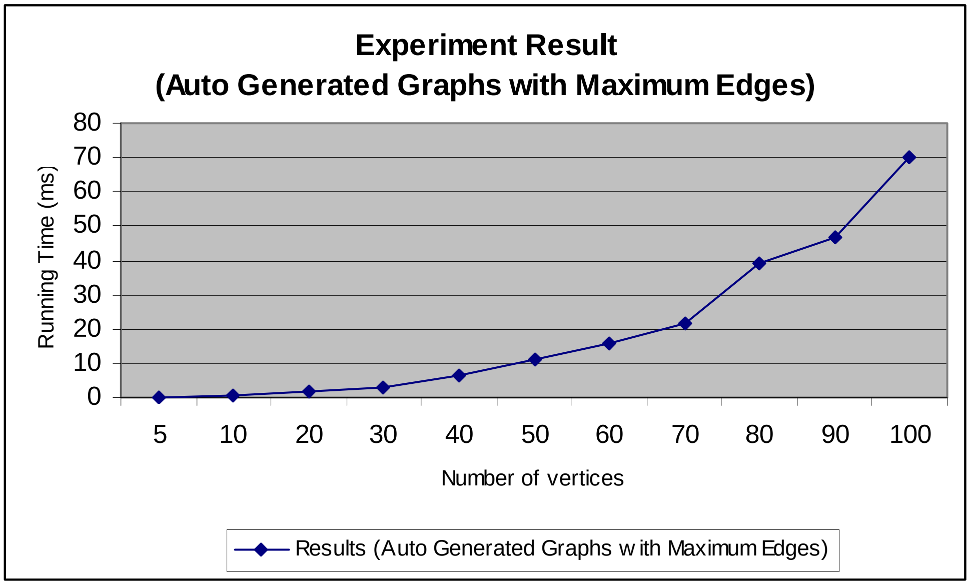


Figure-7.3: Graphical representation of Auto Generated Graphs with Maximum Edges

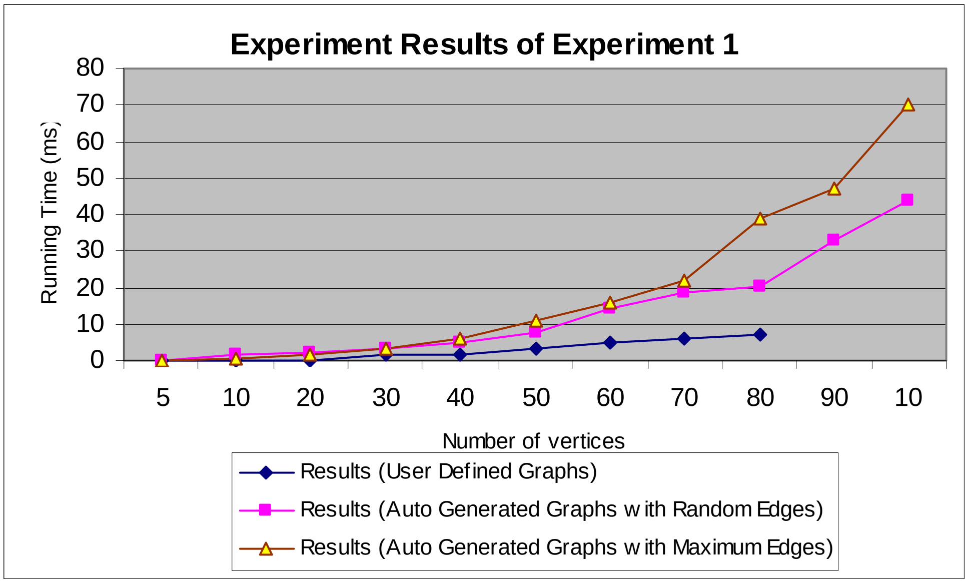


Figure-7.4: Graphical representation of Experiment 1

7.1.2.2. Experiment 2:

EXPERIMENTAL RESULTS (USER DEFINED GRAPHS)

S #	Number of Vertices	Number of Edges	Number of Paths	Maximum Flow	*Busy Node	Number of Maximum Edges	Running Time (ms)
1	5	10	3	439	1 (4)	18	0
2	10	24	6	861	7 (4)	88	0
3	20	53	11	1006	8 (7)	378	0.8625
4	30	107	16	2549	1 (16)	868	1.5625
5	40	116	18	861	9 (16)	1558	1.8625
6	50	129	22	3060	48 (9)	2448	3.342
7	60	182	29	3749	3 (9)	3538	4.6875
8	70	145	30	524	2 (18)	4828	6.1534
9	100	198	27	1357	21 (6)	9898	7.8125

* Node (Number of maximum paths)

Table-7.4: Experiment Results of User-defined Graphs) – Experiment 2

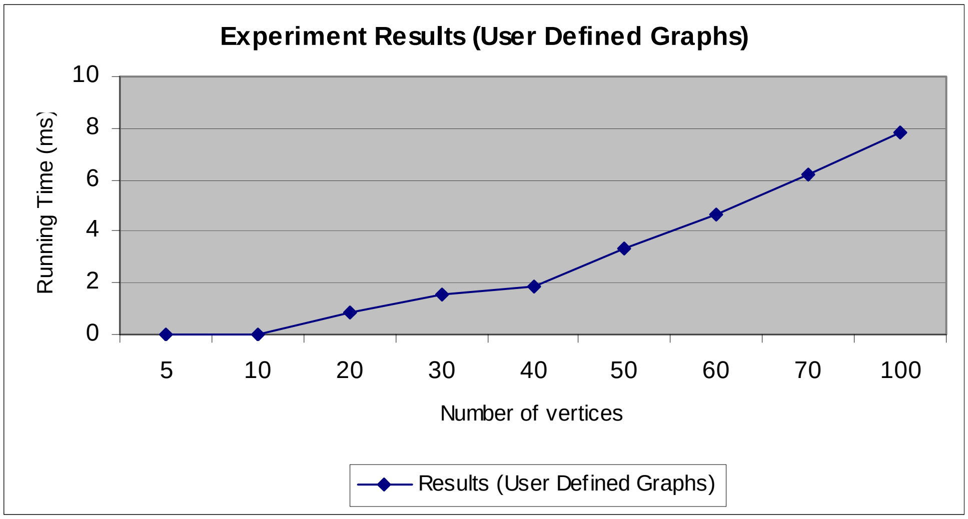


Figure-7.5: Graphical representation of User-defined Graphs – Experiment 2

EXPERIMENTAL RESULTS (AUTO GENERATED GRAPHS WITH RANDOM EDGES)

S #	Number of Vertices	Number of Edges	Number of Paths	Maximum Flow	*Busy Node	Number of Maximum Edges	Running Time (ms)
1	5	8	2	357	2 (2)	18	0
2	10	53	10	1954	4 (30)	88	0
3	20	152	15	3838	13 (5)	378	1.4225
4	30	508	30	4828	20 (6)	868	2.5635
5	40	1258	67	15585	29 (6)	1558	6.25
6	50	1846	73	18095	2 (9)	2448	7.8125
7	60	2846	91	19989	4 (8)	3538	12.5
8	70	3366	109	22982	40 (7)	4828	18.75
9	80	3499	87	22256	62 (8)	6318	20.3125
10	90	4008	105	17657	1 (7)	8008	29.6875
11	100	6904	157	35758	64 (8)	9898	51.5625

* Node (Number of maximum paths)

Table-7.5: Experiment Results of Auto Generated Graphs with Random Edges) – Expriment2

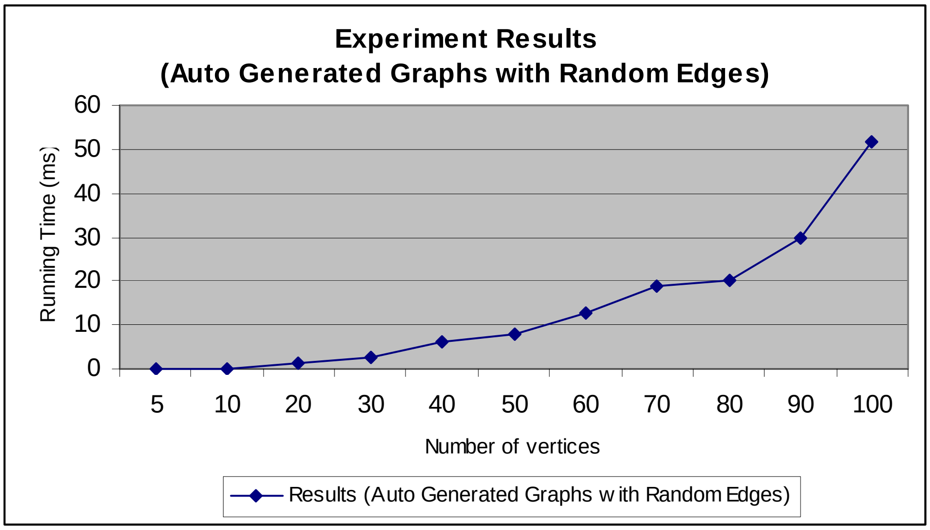


Figure-7.6: Graphical representation of Auto Generated Graphs with Random Edges

EXPERIMENTAL RESULTS (AUTO GENERATED GRAPHS WITH MAXIMUM EDGES)

S #	Number of Vertices	Number of Edges	Number of Paths	Maximum Flow	*Busy Node	Number of Maximum Edges	Running Time (ms)
1	5	18	4	1439	1 (2)	18	0
2	10	88	11	2096	3 (6)	88	0.0625
3	20	378	31	8151	13 (7)	378	1.8725
4	30	868	62	14729	13 (8)	868	3.125
5	40	1557	66	16170	19 (6)	1558	4.6875
6	50	2445	102	23731	7 (6)	2448	10.9375
7	60	3535	115	28736	13 (8)	3538	15.625
8	70	4827	139	30905	20 (9)	4828	25.0265
9	80	6315	179	37755	22 (8)	6318	39.0625
10	90	8006	170	37529	9 (8)	8008	46.875
11	100	9893	211	50864	52 (8)	9898	70.3125

* Node (Number of maximum paths)

Table-7.6: Experiment Results of Auto Generated Graphs with Maximum Edges) – Experiment2

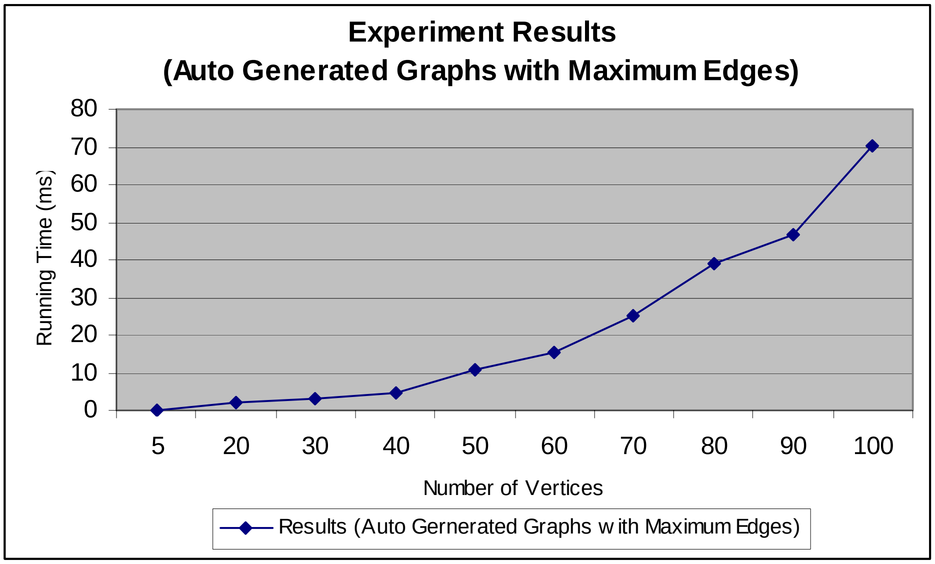


Figure-7.7: Graphical representation of Auto Generated Graphs with Maximum Edges

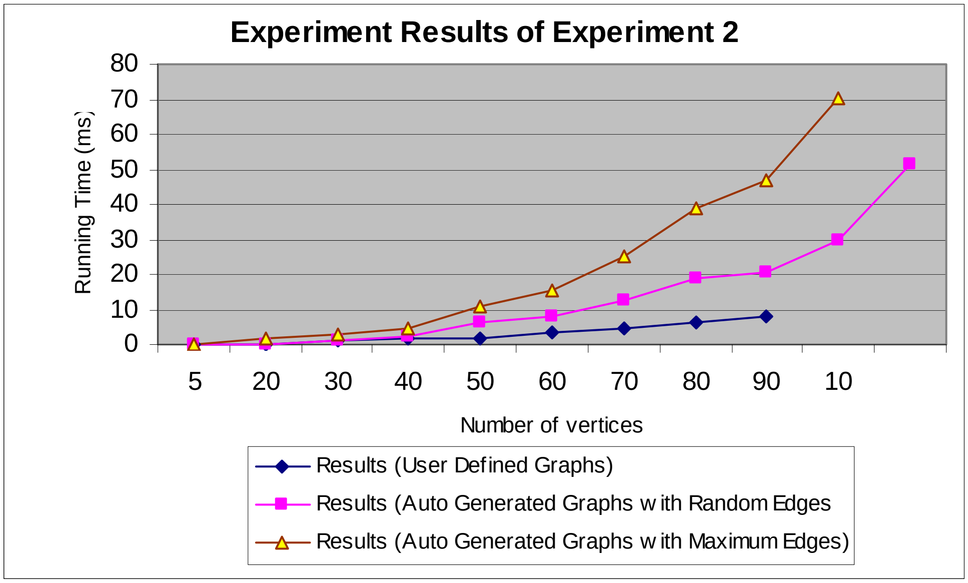


Figure-7.8: Graphical representation of Experiment 2

7.1.2.3. Cumulative Average Running Time

Dateset Types	Average Running Time (ms)		Cumulative Average Running Time (ms)
	Experiment 1	Experiment 2	
User Defined Graphs	2.7158	2.9224	2.8571
Auto Generated (with Random Edges)	13.5216	13.7146	13.6181
Auto Generated (with Maximum Edges)	19.6537	19.7749	19.7143

Table 7.7: Cumulative Average Running Time (ms)

Dateset Types	Vertices	Edges	Cum. Avg. Running Time (ms)
User Defined Graphs	5-100	10-205	2.8571
Auto Generated (with Random Edges)	5-100	10-6904	13.6181
Auto Generated (with Maximum Edges)	5-100	18-9894	19.7143

Table 7.8: All Datasets Input Ranges

The cumulative average running time of these two experiments are shown in Table-7.7. According to the cumulative average time, Edmond Karp algorithm is shown as worst case scenario in Auto generated graphs (with maximum number of edges). In these datasets algorithm traversed in maximum number of time to search out the existing paths in directed graphs.

7.2. Discussion

Following is the discussion and explanation of experimental results achieved by simulation of the Edmond Karp Algorithm.

7.2.1. Running Time Complexity

Analysis about the running time of Edmond Karp Algorithm on a given directed graph **G(V,E)**, it usually measures the size of the input in terms of the number of vertices **|V|** and the number of edges **|E|** of the graph, i.e, there are two parameters described the size of the input and not just one.

The performance of the Edmond Karp algorithm is observed through these experiment results, regarding to the number of augmenting paths, maximum flow and execution time. Simulation is behaved normally in all three categories of the two experiments.

7.2.2. Factors affect the Running Time of the Algorithm

The factors which affect the running time of the maximum flow finding algorithm are mentioned below:

7.2.2.1. Inputs of the Graph

The running time of the Edmond Karp maximum flow finding algorithm is $O(VE^2)$. It means that the running time of the algorithm is influenced by the graph inputs, which are vertices (V) and edges (E). It is explained with the help of an example. Kindly refer to experiment 1, inputs of the graphs at serial no. 7, 8 and 9 of Table-7.1, which shows the increase in running time with the increase in inputs of the graphs data G(V,E) between serial no. 7 & 8, and serial no. 8 & 9. When number of edges is increased then algorithm has to traverse among all the edges to find out the augmenting paths without violating constraints of the algorithm, which required more running time and vise-versa. It shows that the inputs of the graph are directly proportional to execution time of the algorithm to some extent. The density of the graph data is also one of the causes for the increase in execution time of the algorithm. Same behavior of running time is observed in all two experiments.

7.2.2.2. Number of Augmenting Paths

The strategy of Edmond Karp Algorithm is to search the augmenting paths, to augment alongwith these paths in residual network, to update residual capacities $c_f(u,v)$, respective edge flows f(u,v) and to delete the saturated edges (zero capacity edges) until sink is no longer reachable from the source. Total number of paths describes that how many times algorithm is traversed in the residual network for the searching of augmenting paths. It is also observed by these experiments that the running time of the algorithm is also affected due to the number of augmenting paths in the directed graphs. For example, in Table 7.3 of experiment 1 graphs at serial no. 7, 8 and 9 contain the 109, 120 and 172 augmenting paths and their running times are 15.625, 21.875, 39.062 ms respectively. It is also observed that the influence of fluctuation in the number of paths is also reflected in their respective running time.

7.2.2.3. Influence of Searching Technique

Edmond Karp utilizes breadth-first search (BFS) technique for searching of augmenting paths. BFS always tries to search the shortest paths on priority basis for achieving its target. The level of graph G(V,E) is directed breadth-first search to start with the root source vertex with sideways and back edges to search a path. The level of a destination vertex is the length of the shortest path from s to t in G(V,E). If the destination vertex is at the deepest level of the graph, then it becomes the worst case scenario of the Breadth-first Search. So the search technique also affects the performance of the Edmond Karp algorithm regarding its running time and space complexity.

7.2.2.4. Symmetry Edges

For the maximum flow problems, it is assumed that the input network is *symmetric*. If the graph G contains more symmetric edges and less shortest-paths p for the Breadth-first Search, then running time of the maximum flow finding algorithm is increased to some extent. Therefore, the execution time of the maximum flow algorithm can be increased or visa-versa.

7.2.2.5. Calculation of large capacity Edges

Arithmetic operations are also involved to calculate the maximum flow of the directed graph G(V,E). It is also observed that if graph G(V,E) contains large capacities c(u,v) values then it may affect the execution time of the algorithm in respect of its arithmetic operations, it can be analysed by results shown in Table 7.3. Graphs at serial no. 10 and 11, their maximum flows are 42190 and 51651 with the running time of 46.875 and 70.312 ms respectively.

CHAPTER 8

Conclusions and Future Enhancements

8.1. Conclusions

Through this research, behavior of the Edmond Karp maximum flow finding algorithm is observed and the idea of busy node is elaborated. These test results produced by the simulator shows the efficiency and performance of the Edmond-Karp algorithm. Breadth first Searching (BFS) technique performed the vital role in the performance of the said algorithm. Ford-Fulkerson runs in polynomial time $O(E|f^*|)$ and depends on the maximum flow f*, but Edmonds-Karp runs in $O(VE^2)$ time by modifying Ford-Fulkerson Method to use breadth first search to identify augmenting flows. This algorithm can also be analysed by using other searching techniques and simulation can be further enhanced for the concept of multiple sources and destination based graphs or the gain factor involved in maximum flow finding algorithms.

Computer network systems are using the maximum flow algorithms to control the congestion of the network traffic. One of the objectives of maximum flow finding algorithm is to maximize flow in the network to achieve optimal solution for the practical applications implementations. Efficient maximum flow finding algorithm depends on efficient techniques to search out the augmenting paths in the network.

Maximum flow is an important aspect of network traffic optimization and image processing enhancement. As network graphs are the most important resource, to analysis the performance of Edmond Karp algorithm. Many algorithms have been designed to implement maximum flows in the networks. Design methods include analytic modeling, deterministic modeling and simulations. Simulation being the most accurate of all is commonly employed for system's performance evaluation despite the fact that they require complex programming for developing an efficient simulator.

This work involves development of a simulator for proposed Edmond Karp maximum flow finding algorithm with analysis for the optimized performance of Edmond Karp algorithm in different datasets. Using this simulator deterministic evaluation of Edmond Karp algorithm can also be performed. Implementation and visual presentation of Edmond Karp can be used for the understanding and training of the students. It provides the following as outputs:

- Optimal Augmenting Paths found in the residual network.
- Indicate the Busy node in the network.
- Calculate the optimal solution for the maximum flow.
- Calculate the running time of the different datasets in milliseconds.
- Save the Graphs and experiment results for analysis purposes.

This work presents a simulator that uses graphical animation to convey the concepts of Edmond Karp maximum flow find algorithm for the different datasets. The simulator is unique in a number of respects. First, it uses a more realistic model that can be configured easily by the user. Second, it graphically highlights the each augmenting path and its respective flow obtained from it. Using this representation, it becomes much easier to understand concept and performance of the implemented algorithm for different datasets. The simulator can be used by students in algorithms and design courses or by anyone interested in learning maximum flow finding algorithms in an easier and a more effective way. The system is designed to mimic the dynamic behavior of the system over time related to calculate maximum flow. It runs self driven simulations and uses a synthetic workload that is artificially generated to resemble the expected conditions in the modeled system.

A user friendly and mouse driven Graphical User Interface (GUI) has been integrated .It provides the user an opportunity to save the input of all graphs either designed by user or generated by simulator. It also provides the user an opportunity to generate a report sheet, which shows graphical as well as analytical results. It uses file system design to save the graphs and results data, which makes the system faster as compared to the use of database for storage of results. This feature makes the system attractive for experimental analysis as academic use.

In simulation, it is difficult to impose time management, since scenarios can be changed according to the graph structure. However, in this case datasets are predefined before simulation starts, time management policy can be defined.

8.2. Future Enhancements

The following future enhancements are suggested:

- **Multiple Source and Sink Nodes**

  This simulation can be enhanced for the concept of multiple source and sink / destination nodes as shown in Figure – 8.1.

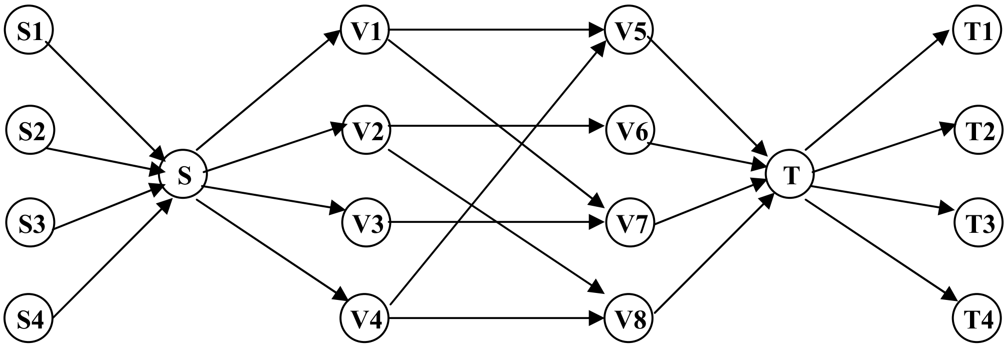


Figure – 8.1: Multiple source and sink nodes graph

- **Analysis Gain Factor (GF) in Maximum Flow**

  Simulation can further enhanced to analysis the affect of the gain factor on the maximum flow problems [9]. Gain Factor represents the lost portion of the flow during the traversal in the directed graph.

- **Utilize other Searching Techniques in Edmond Karp Algorithm**

  Edmond Karp algorithm can be tested by some other available searching techniques, like uniform-cost search or bidirectional search.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *“Introduction to Algorithms”*, Second Edition. MIT Press and McGraw-Hill, 2001.

[2] Luca Trevisan. *“Efficient Algorithms and Intractable Problems”*, Fall 2001.

[3] T.E Harris, *“Maximum Flow”,* RAND Corporation in Santa Manica, 1950

[4] L.R. Ford, Jr. and D.R. Fulkerson, *Maximal flow through a network*, Canadian Journal of Mathematics 8 (1956).

[5] L.R. Ford, Jr. and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, New Jersey, 1962.

[6] J. Edmonds and R.M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the Association for Computing Machinery 19 (1972).

[7] Robert Sedgewich, *Finding Paths in Graphs*, Princeton University.

[8] Mark F. Bramlette, *“Finding Maximum Flow with Random and Genetic Search”*, Stampede Group, 7723 Fulton Avenue, North Hollywood, California.

[9] Kevin Daniel Wayne, *“Generalized Maximum Flow Algorithms”*, Graduate School of Cornell University, 1999

Many other relevant books and websites are also concerned for the research work.