ON BOUNDED-DELAY MINIMUM-COST PATH PROBLEM

By

YUANBING DU

Bachelor of Art

Jinan University

Guangzhou, P.R.China

1996

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2003

ON BOUNDED-DELAY MINIMUM-COST PATH

PROBLEM

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

[illegible]

Dean of the Graduate College

PREFACE

Real time applications over the Internet require guaranteed end-to-end quality of service (QoS). The task of QoS-based routing is to find a route in the network which has sufficient resources to satisfy the QoS requirements. One of the QoS-based routing problems --- the Bounded-Delay Minimum-Cost Path Problem (BDMCP) is NP-hard. Several heuristic algorithms including LHWHM algorithm and BFM-BDMCP algorithm have been proposed to find suboptimal solutions. In this thesis, we propose a new heuristic algorithm called K-BFM-BDMCP which has better chance to find optimal solution than BFM-BDMCP and still runs in polynomial time. We also do a comparative study of the new algorithm with BFM-BDMCP algorithm.

ACKNOWLEDGEMENTS

Many thanks are given to my advisor, Dr. H.K. Dai, for his continuous support and help on all aspects from technical to material. His guidance and advice helped me overcome many technical obstacles which otherwise would take much more efforts.

Special thanks are given to Ruibin for his love and constant support, which make this long journey full of joy.

I would like to thank Dr. Nohpill Park and Dr. Douglas Heisterkamp for serving on my thesis committee and for their invaluable input.

Thanks also go to Iker Gondra, Shim Chow, Yong Hu and Jing Ding for their help and support during my preparation for thesis defense.

The thesis is dedicated to my father and mother. It is their love, encouragement, belief and understanding that make everything I have possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

The Internet has been around for nearly twenty years and it keeps growing rapidly in terms of speed, size and user. But so far the underlying protocols have not changed and the Internet can still only provide "best-effort" service, which means it will try its best to forward network traffic, giving better service to some traffic at the expense of giving worse to the rest. While traditional applications (such as File Transfer Protocol and email) work fine with this kind of service, newly emerged real time applications (such as video-conferencing and Video on-Demand) are having a hard time because they require high bandwidth, low delay and small jitter. In other words, these new applications require better transmission service than "best-effort". Quality-of-Service (QoS) is a framework that is being developed to bring down the disparity and unfairness in the usage of Internet resources such that a set of service requirements by the users can be satisfied. To provide QoS in the Internet, many techniques have been proposed and studied, one of which is QoS-based routing. The aim of QoS routing is to determine paths based on Qos parameters such as delay, jitter, bandwidth, loss probability and other end-to-end quality features. The proposed thesis will be dealing with QoS-based routing problems.

1.1 Quality-of-Service

Definitions

The notion of Quality-of-Service has been proposed to capture the qualitatively or quantitatively defined performance contract between the service provider and the user applications. As defined in [1], QoS is "a set of service requirements to be met by the network while transporting a flow". The service requirements are expressed in some measurable QoS metrics. Well-known metrics include bandwidth, delay, jitter, monetary cost, etc.

QoS Metrics

Bandwidth

Bandwidth is defined as the difference between the highest and lowest frequencies of a transmission channel. It is used to measure how fast data flows on a given transmission path and is expressed as data speed in bits per second (bps). Now, what we are interested most is the available bandwidth, which refers to the residual bandwidth that could be used to route traffic on a particular link.

Delay

Delay refers to the time taken for a packet to travel from the source to the destination. The most direct way to determine the delay is to send a packet from the source to the destination and the destination is required to send back immediately. By measuring the round-trip time and dividing it by two, we can get the delay.

Jitter

Jitter is defined as the amount of variation in the end-to-end packet transit time. It happens due to the varying sizes of packets of a given flow, which result in differences in delay while the packets are delivered. Jitter plays a very important role in real time applications such as audio and video transmission. A high jitter will give an uneven quality to sound or image.

Monetary Cost

Monetary cost is the cost incurred when the users are required to pay for using the Internet resources.

Usually, different metrics may have different features. There are three types of metrics: additive, multiplicative, and concave [9]. They are defined as follows.

Let $m(n_1,n_2)$ be a metric for link(n_1,n_2). For any path $P = (n_1, n_2, ..., n_{j-1}, n_j)$, the metric m is: (Note here $n_1, n_2, n_3 ..., n_{j-1}, n_j$ represent network nodes.)

- Additive: if $m(P) = m(n_1,n_2) + m(n_2,n_3) + ... + m(n_{j-1},n_j)$

  Examples are delay, jitter, cost and hop-count. For instance, the delay of a path is the sum of the delay of every hop.

- Multiplicative: if $m(P) = m(n_1,n_2) \times m(n_2,n_3) \times ... \times m(n_{j-1},n_j)$

  An example is reliability, in which case $0 \leq m(n_i, n_{i+1}) \leq 1$ for $1 \leq i \leq j-1$.

- Concave: if $m(P) = \min\{\ m(n_1,n_2)\ ,\ m(n_2,n_3)\ ,\ ...\ ,\ m(n_{j-1},n_j)\ \}$

  An example is bandwidth, which means that the bandwidth of a path is determined by the link with the minimum available bandwidth.

1.2 QoS-Based Routing

Definitions

QoS-based routing is defined in [1] as: "A routing mechanism under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of the flows." Here, the QoS requirement of a flow is given as a set of constraints such as available bandwidth, cost, delay, link and end-to-end path utilization, and induced jitter. The basic function of QoS-based routing is to find a network path to satisfy the given constraints. In addition, most QoS-based routing algorithms consider the optimization of resource utilization.

Figure 1-1 shows a simple example of QoS-based routing. Suppose that there is a traffic flow from node A to node C which requires 4M bps bandwidth. As we can see, although path A→B→C is shorter, which has 2 hops, it will not be selected because it does not have enough bandwidth. Instead, path A→D→E→C is selected.

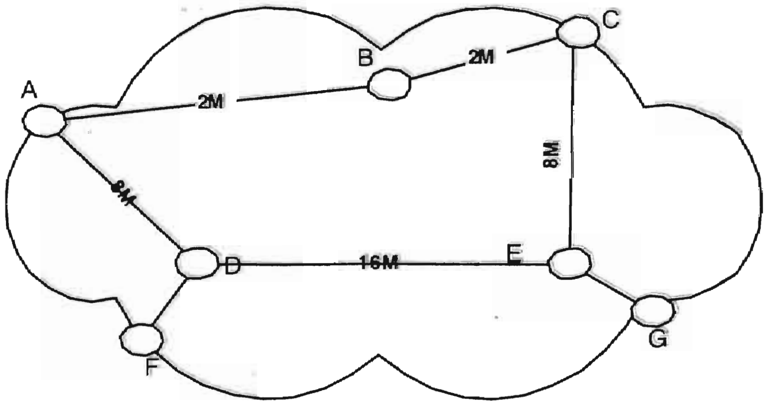

Figure 1-1: A QoS-based routing example.

Objectives of QoS-Based routing

The main objectives of QoS-Based routing are:

- First, to find a path that has a good chance of meeting the QoS requirements of end users. At the same time, this should be done dynamically so that enable the routing to select a path among several feasible paths based on some policy constraints.
- Second, to optimize the network resource usage. A network QoS-based routing scheme can aid in the efficient utilization of network resources by improving the total network throughput.
- Third, to degrade network performance gracefully when things like congestion happen. When network is in heavy load, QoS-based routing is expected to give better performance (e.g., better throughput) than best-effort routing, which can degrade the performance dramatically.

Requirements for QoS-Based Routing Algorithm

The current Internet routing protocols are based on two routing algorithms – Distance Vector algorithm and Link-State algorithm. In Distance Vector algorithm, neighboring routers exchange routing information periodically. Thus every router can learn the routing information from others. Based on that information, the shortest path to every destination can be computed. An example is the well-known Routing Information Protocol (RIP). This is also called Bellman-Ford-Moore algorithm.

While in Link-State algorithm, every router advertises its link state information to the whole network, thus every router can receive the link-state information. Such information is maintained in a local database in every router, from which the routing table is calculated using Dijkstra's algorithm. The advertising is triggered by events, and it also happens periodically. An example is the Open Shortest Path First (OSPF) protocol.

Below are some basic requirements for QoS-based routing algorithms:

- The algorithms should be efficient and scalable enough that they can be used for large network.
- The algorithms should not be so complicated that they can be implemented easily.
- The algorithms should be suitable to current Internet architecture.

Note that some of them are conflicting with each other, so that compromise has to be made. We have to make some trade off between efficiency and complexity.

QoS-based routing algorithms are expected to be employed in current Internet. They must be easy to implement and compatible with the current "best-effort" routing protocols.

QoS-Based Routing Problems

The QoS-based routing problems can be divided into two major classes: unicast routing and multicast routing. The unicast routing problem refers to finding the best feasible path between a single source and a single destination, which satisfies a set of QoS requirements. On the other hand, multicast routing problem refers to finding the best feasible tree covering a single source and a set of destinations, which satisfy a set of QoS requirements. This thesis deals with solving a unicast routing problem.

Unicast routing problems can be further partitioned into sub-classes based on the type of QoS metric that is used for routing. For the concave QoS metrics such as available bandwidth, the state of a path is determined by the state of the bottleneck link. For example, in Figure1-2, the bandwidth of the path S→A→B→C is 1, which is the bandwidth of the bottleneck link (A, B). For these metrics, two basic routing problems can be defined. One is called link-optimization routing. An example is the bandwidth-optimization routing, which is to find a path that has the largest bandwidth on the bottleneck link. The other problem is called link-constrained routing. An example is the bandwidth-constrained routing, which is to find a path whose bottleneck link bandwidth is above a required value.

For additive metrics such as delay, delay jitter and cost, the state of a path is determined by the combined state over all links on the path. For example, in Figure1-2, the delay of the path S→A→B→T is 7, which is the total delay of all links on the path. For these metrics, two basic routing problems could be defined. One is path-optimization routing. An example is the least-cost routing, which is to find a path whose total cost is minimized. The other problem is path-constrained routing. An example is the delay-constrained routing, which is to find a path whose delay is bounded by a required value. The above four basic problems can be combined into many different routing problems. For example, the bandwidth-constrained least-delay routing problem belongs to the link-constrained path-optimization problem. The delay-constrained least-cost routing problem belongs to the path-constrained path-optimization problem. There are some other problems such as link-constrained path-constrained routing and path-constrained link-optimization routing.

The problem with which we are dealing in this thesis is a path-constrained path-optimization problem, which we call the Bounded-Delay Minimum-Cost Path Problem.

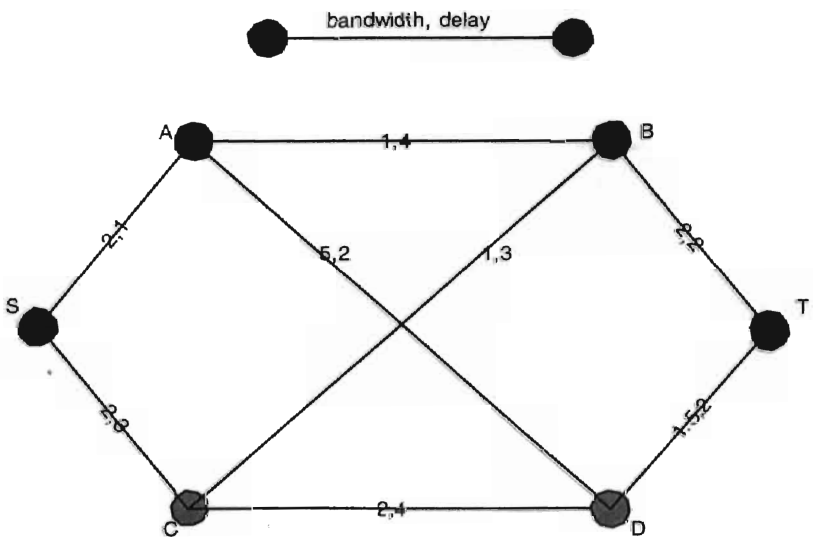

Figure 1-2: An example of network with bandwidths and delays.

1.3 Bounded-Delay Minimum-Cost Path Problem (BDMCP)

In this section, we will introduce the Bounded-Delay Minimum-Cost Path Problem (BDMCP).

Let R^+ denote the set of all positive real numbers. Given a network N represented by a directed graph $G(V,E)$, a source vertex s, a destination vertex t, a cost function c: $E \rightarrow R^+$, a delay function d: $E \rightarrow R^+$, and a constant $T \in R^+$, an instance of BDMCP problem BDMCP(N,s,t,T) is to find a path p from s to t (p is thus called an s-t path) such that $d(p) \leq T$ with smallest $c(p)$ if such a path exists.

An s-t path p which satisfies $d(p) \leq T$ is called a feasible path. We assume that both functions c and d are additive in the sense that the value of function c (d, respectively) of a path is equal to the summation of the values of function c (d, respectively) of all edges on the path.

An undirected graph may be viewed as a directed graph with each link $e = (u, v)$ replaced by two oppositely oriented links $e_1 = (u, v)$ and $e_2 = (v, u)$. In this case $c(e_1) = c(e_2)$ and $d(e_1) = d(e_2)$. For $e = (u, v)$, $c(e)$ and $d(e)$ are also denoted as $c_{u,v}$ and $d_{u,v}$ respectively.

Figure 1-3 shows an example of BDMCP problem with the delay constraint of 50.

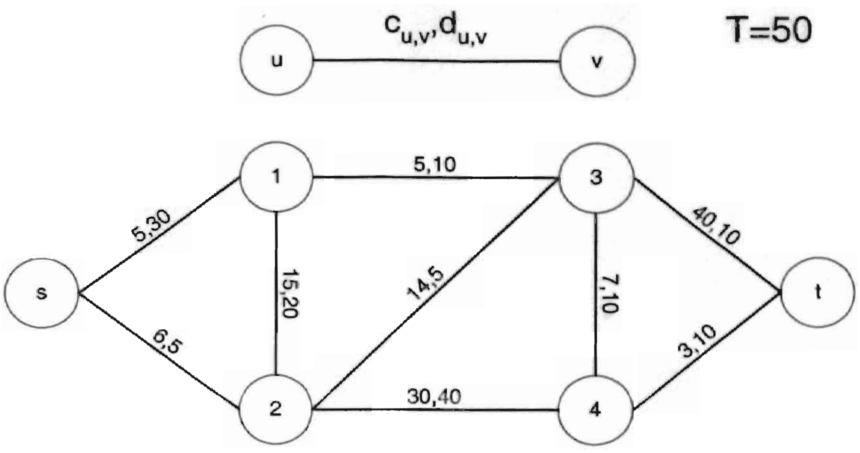

Figure 1-3: An example of BDMCP problem.

In the above example, there exist a number of feasible s-t paths such as $s \xrightarrow{5,30} 1 \xrightarrow{5,10} 3 \xrightarrow{40,10} t$ and $s \xrightarrow{6,5} 2 \xrightarrow{14,5} 3 \xrightarrow{7,10} 4 \xrightarrow{3,10} t$, with cost 50 and 30 respectively. The latter is the feasible minimum-cost path.

Throughout the thesis, we use n to denote $|V|$ and m to denote $|E|$ unless otherwise stated.

1.4 Related Work

Wang and Crowcroft [5] proved that the problem of finding paths subject to two or more additive constraints is NP-complete. The corresponding decision problem of BDMCP is the problem of finding paths subject to two additive constraints. So BDMCP problem is NP-hard. Wang and Crowcroft also presented a centralized algorithm and two distributed algorithms for finding paths in a network satisfying bandwidth and delay constraints.

Widyono [24] proposed a routing method that gives the optimal path having the lowest possible cost without violating the delay constraint, unfortunately with an exponential running time. The constrained Bellman-Ford (CBF) routing algorithm performs a breadth-first search, discovering paths of monotonically increasing delay while recording and updating lowest cost path to each node it visits. It stops, when the highest constraint is exceeded, or there is no more possibility of improving the paths. It also has exponential worst-case running time.

Chen and Nahrstedt[11] have done a comparison study of algorithms that have been proposed for QoS-based routing for both unicast and multicast routing.

Hassin[23] presented a full polynomial approximation scheme for the BDMCP problem one of which runs in time $O(log\ log(U/L)[|E||V|\varepsilon^{-1}+log\ log(U/L)])$ where U and L denote the upper bound and lower bound of the cost respectively. Later works improved Hassin's algorithm with better complexity properties. However, these approximation algorithms are computationally expensive and not easy to implement.

Luo et al. [5] proposed a simple heuristic algorithm called LHWHM algorithm for the BDMCP problem. This algorithm is based on Dijkstra's algorithm and performs well on sparse graphs.

Most recently, Ravi et al. [8] designed a new heuristic algorithm called BFM-BDMCP and it is based on Bellman-Ford-Moore algorithm. It is also suitable for distributed implementations. They have also done a comparative evaluation and the experimental results showed that LHWHM algorithm and BFM-BDMCP algorithm are serious candidates for implementation in real network environment.

1.5 Scope of This Thesis

This thesis is concerned with the design and analysis of heuristic algorithms for the BDMCP problem. The thesis is organized as follows.

In Chapter 2, we review two classic algorithms for shortest path problem and two major existing heuristic algorithms, LHWHM algorithm and BFM-BDMCP algorithm, for BDMCP problem. We also review two pseudopolynomial time algorithms for BDMCP. In Chapter 3, we discuss the drawback of BFM-BDMCP algorithm and propose an improved version of BFM-BDMCP algorithm called K-BFM-BDMCP. Then we prove that K-BFM-BDMCP runs in polynomial time and discuss its advantage over BFM-BDMCP. In Chapter 4, we present our experimental results.

Chapter 2 Review of Related Algorithms

In this chapter, we review several existing algorithms that are related to BDMCP problem. Dijkstra's algorithm and Bellman-Ford-Moore algorithm are classic algorithms for shortest-path problem. LHWHM algorithm and BFM-BDMCP are heuristic algorithms for BDMCP problem. They are based on Dijkstra's algorithm and Bellman-Ford-Moore algorithm respectively. At the end of this chapter, we review two pseudopolynomial time algorithms for BDMCP problem which we will use for evaluating our algorithms.

2.1 Dijkstra's Algorithm

Dijkstra's algorithm solves the single-source shortest-path problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are nonnegative. In this section, we assume that we are to solve a problem related to the two major heuristic algorithms for BDMCP presented in this chapter: given an instance of BDMCP(N,s,t,T), find the minimum delay path from destination t to every vertex in V.

First we construct a network N^* by reversing the directions of all links in N and treat destination node t as the source node in N^*. We will apply Dijkstra's algorithm to N^*.

Dijkstra's algorithm proceeds as follows. The algorithm associates three attributes with each node u, namely, *DLABEL*(u), *PERM*(u) and *PRED*(u). At any step in this algorithm *DLABEL*(u) represents the delay of a path from node t to node u, *PERM*(u) indicates whether a node u has been permanently labeled or not, and $PRED(u)$ indicates the node from which node u has received its latest *DLABEL* value.

Dijkstra's algorithm uses the technique of relaxation. The process of relaxing an edge (u,v) consists of testing whether we can improve the minimum-delay path to v found so far by going through u and, if so, updating $DLABEL(v)$ and $PRED(v)$. A relaxation step may decrease the value of the minimum delay path estimate $DLABEL(v)$ and update v's predecessor field $PRED(v)$. The following code performs a relaxation step on edge (u,v).

```
PROCEDURE relax(u,v)
1. if $DLABEL(v)>DLABEL(u)+ d_{u,v}$ then
2.          $DLABEL(v):=DLABEL(u)+ d_{u,v}$
3.          $PRED(v):=u$
END
```

Figure 2-1: Procedure relax(u,v).

Initially, we have $DLABEL(t) = 0$ and $DLABEL(u) = \infty$ for $u \neq t$, $PERM(t) = 1$ and $PERM(u) = 0$ for $u \neq t$ and $PRED(u) = u$ for all $u \in V$. The algorithm repeatedly selects a node, say u, from the nodes not yet permanently labeled which has the minimum *DLABEL* value, set $PERM(u) = 1$ and relax all edges leaving u. This continues until all the nodes are permanently labeled.

```
ALGORITHM Dijkstra($N^*$,$t$)
1.  $DLABEL(t)$:=0
2.  $PERM(t)$:=1
3.  for each $v \in V(N^*)$
4.          if $v \neq t$ then
5.                  $DLABEL(v)$:=$\infty$
6.                  $PERM(v)$:=0
7.          $PRED(v)$:=$v$
8.  $u$:=$t$
9.  for each vertex $v$ adjacent to $u$
10.         relax($u$,$v$)
11. from all the nodes which are not yet permanently labeled, pick a node $u$ with smallest
    $DLABEL$ value and set $PERM(u)$:=1
12. If no such $u$ exists, HALT
13. goto 9
END
```

Figure 2-2: Dijkstra's algorithm.

At termination, the predecessor array *PRED* can be used to trace the minimum delay path generated by the algorithm. The main feature of Dijkstra's algorithm is that the label *DLABEL*(u) of a permanently labeled node u is the delay of a minimum delay t-u path. So once a node is labeled permanently, no further labeling of this node is necessary.

From a recently permanently labeled node *u*, the algorithm extends the minimum delay path t-*u* to a t-*v* path where *v* is a neighbor of *u*. Dijkstra's algorithm runs in $O(n^2)$.

A formal description of Dijkstra's algorithm is given in Figure 2-2.

Let us apply Dijkstra's algorithm to the example of BFMCP problem in Figure 1-3. First we obtain the network N^* showed in Figure 2-3:

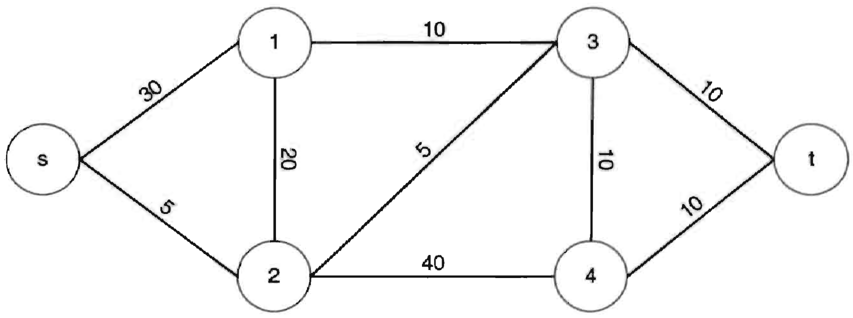

Figure 2-3: Network N^*.

Then we proceed as follows:

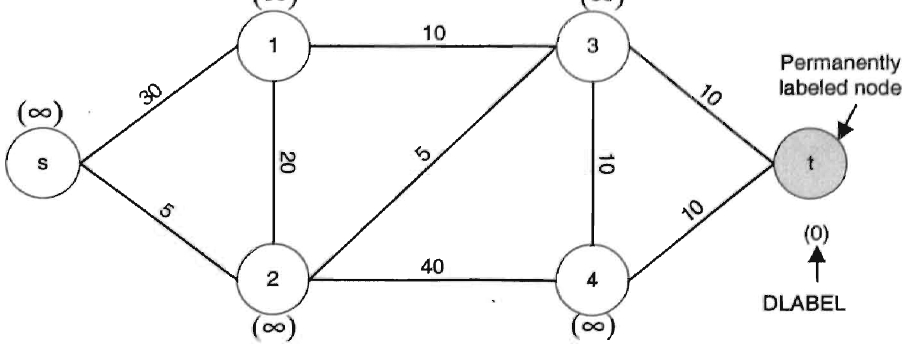

Figure 2-4: Initialization.

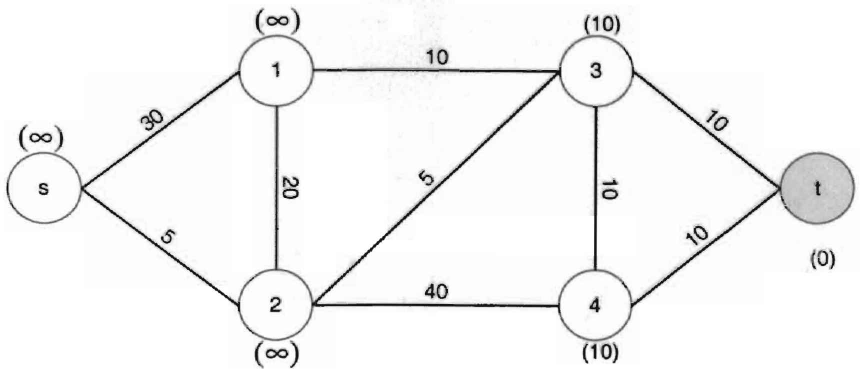

Figure 2-5: Iteration 1.

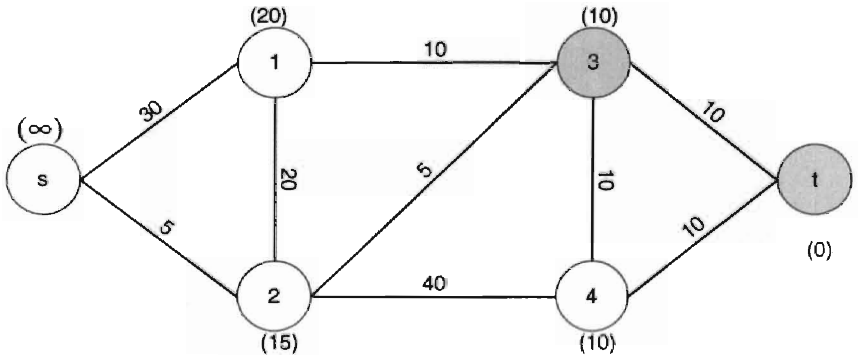

Figure 2-6: Iteration 2.

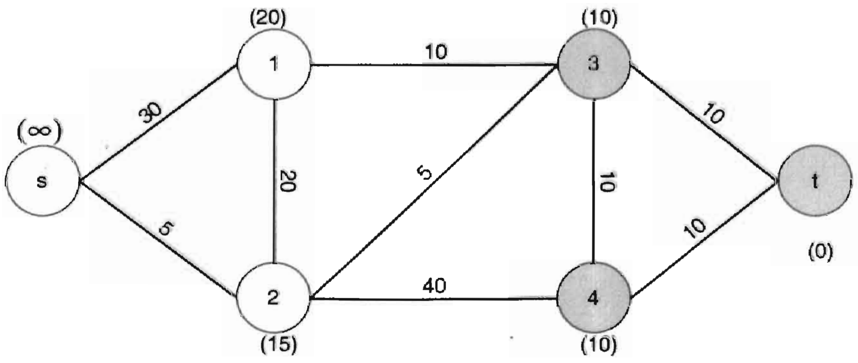

Figure 2-7: Iteration 3.

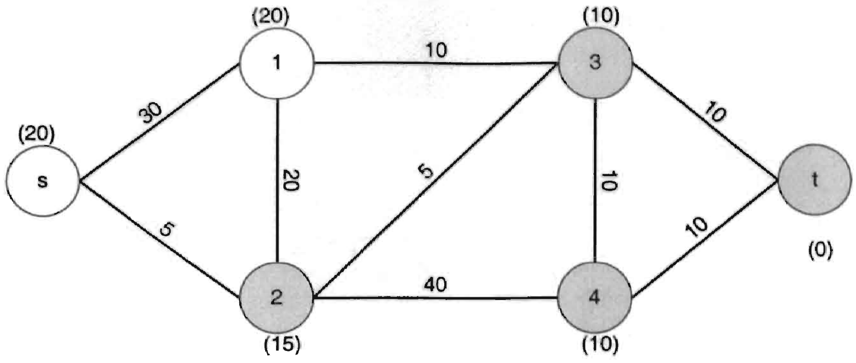

Figure 2-8: Iteration 4.

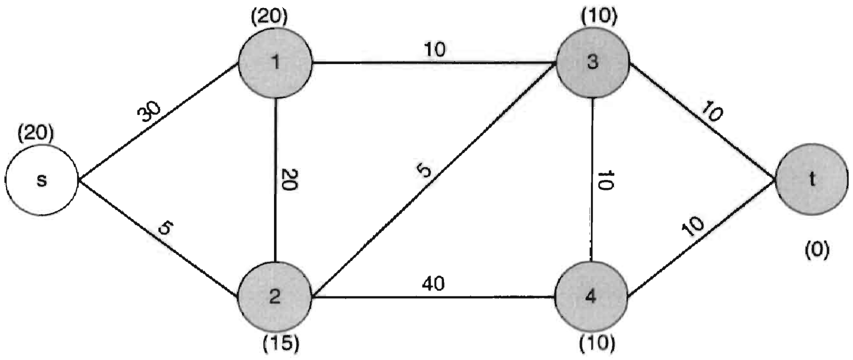

Figure 2-9: Iteration 5.

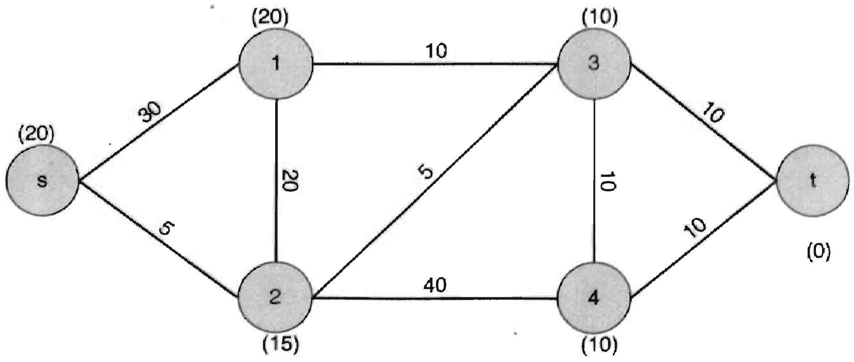

Figure 2-10: Iteration 6.

2.2 LHWHM Algorithm

The LHWHM algorithm is based on Dijkstra's algorithm. It was proposed by Luo, Huang, Wang, Hobbs and Munter in [5]. It finds suboptimal solution to BDMCP problem.

Definition 2.1. A cost-delay label is a 2-tuple (c,d) where c and d are nonnegative real numbers and denote cost and delay respectively.

The algorithm associates four attributes with each node u, namely, *LABEL*(u), *PERM*(u), *D*(u), and *PRED*(u). The attribute *D*(u) stores the delay of a minimum delay u-t path which can be done easily by applying Dijkstra's algorithm to the network N^*, which is obtained by reversing the directions of all links in the given network, using the link delays and using node t as the source. We have shown how this is done in the previous section. At any step in this algorithm *LABEL*(u) (is a cost-delay label) represents the cost of a path from node s to node u and the delay of a path from node s to node u. The attribute *PERM*(u) indicates whether the node u has been permanently labelled or not, and *PRED*(u) indicates the node from which node u has received its latest *LABEL* value.

LHWHM algorithm uses a modification of **relax** procedure used in Dijkstra's algorithm. It is showed in Figure 2-11.

```
PRECEDURE modified-relax($u$,$v$)
1. if $LABEL(u).delay+d_{u,v}+D(v)\leq T$ and then $LABEL(u).cost+c_{u,v} < LABEL(v).cost$ then
2.         $LABEL(v).cost:=LABEL(u).cost+ c_{u,v}$
3.         $LABEL(v).delay:=LABEL(u).delay+ d_{u,v}$
4.         $PRED(v):=u$
END
```

Figure 2-11: Modified-relax procedure.

A formal description of the LHWHM algorithm is given below.

```
ALGORITHM LHWHM($N$,$s$,$t$,$T$)
1. Construct a network $N^*$ by reversing the directions of all links in $N$. Apply Dijkstra's
   algorithm on $N^*$ (using delays, instead of costs) and calculate for each node $u$ the
   value of $D(u)$, the delay of a minimum delay path from $t$ to $u$
2. $LABEL(s):=(0,0)$
3. $PERM(s):=1$
4. for each $v \in V$
5.         if $v \neq s$ then
6.                 $LABEL(v):=(\infty,0)$
7.                 $PERM(v):=0$
8.         $PRED(v):=v$
9. $u:=s$
```

```
10. for each vertex $v$ adjacent to $u$
11.         modified-relax($u$,$v$)
12. from all the nodes which are not yet permanently labeled, pick a node $u$ with the
    smallest cost of $LABEL$ value and set $PERM(u)$:=1
13. If no such $u$ exists, HALT
14. goto 10
END
```

Figure 2-12: LHWHM algorithm.

2.3 Bellman-Ford-Moore Algorithm

The Bellman-Ford-Moore algorithm (BFM for short) solves the single-source shortest-paths problem in the more general case in which edge weights can be negative. Since for BDMCP problem, all the edge weights (costs and delays) are nonnegative, BFM seems no better than Dijkstra's algorithm. But BFM does have some properties that make the heuristic algorithm for BDMCP based on it outperform LHWHM algorithm. We will come back to this in the next section.

Again, we present BFM by using it to solve the same problem as in Section 2.1. The BFM algorithm associates with each node u two attributes $DLABEL(u)$ and $PRED(u)$. Initially $DLABEL(t)$ =0, $DLABEL(u) = \infty$,for all $u \in V$ and $u \neq t$ and $PRED(u)$ =u for all $u \in V$.

Like Dijkstra's algorithm, the BFM algorithm uses the technique of relaxation, progressively decreasing an estimate *DLABEL*(v) on the delay of a minimum-delay path from the source t to each vertex v in V until it achieves the actual minimum-delay path delay. On the other hand, unlike Dijkstra's algorithm in which just relaxing all the outgoing edges of a particular node during one iteration, the BFM algorithm relaxes all outgoing edges of all nodes (we call relaxing all outgoing edges for a node when scanning that node) during one iteration, which is called a sweep. It keeps sweeping until no relaxation in a sweep succeeds to update the *DLABEL* value of any node.

A formal description of the BFM algorithm (asynchronous version) is given below:

```
ALOGORITHM BFM($N^*$,$t$)
1.  $DLABEL(t)$:=0
2.  for each $v \in V$
3.          if $v \neq t$ then
4.                  $DLABEL(v)$:=$\infty$
5.          $PRED(v)$:=$v$
6.  for each edge $(u,v) \in E$
7.          relax($u$,$v$)
8.  If no node's $DLABEL$ value is updated during the previous sweep, HALT
9.  goto 6
END
```

Figure 2-13: BFM algorithm.

It can be shown that after performing n-1 sweeps the BFM algorithm will terminate, resulting in the time complexity of O(mn). The above implementation of the BFM algorithm permits two choices which are presented below.

Version 1 (Asynchronous)

While scanning a node u during a sweep, use the current value of $DLABEL(u)$ to label the neighbors of node u.

Version 2 (Synchronous)

While scanning node u during a sweep, use the value of $DLABEL(u)$ at the end of the previous sweep for updating the labels of the neighbors of node u.

2.4 BFM-BDMCP

The BFM-BDMCP algorithm [7] for the BDMCP problem is based on the BFM algorithm. As in the case of the LHWHM algorithm, we first compute $D(u)$ for each node u where $D(u)$ is the minimum delay of any u-t path. Again each node u is associated with the attributes $LABEL(u)$ and $PRED(u)$.

The BFM-BDMCP algorithm also uses the technique of modified relaxation as in LHWHM.

A formal presentation of the BFM-BDMCP algorithm (asynchronous version) is as follows:

```
ALGORITHM BFM-BDMCP($N,s,t,T$)
1.  Construct a network $N^*$ by reversing the directions of all links in $N$. Apply Dijkstra's
    algorithm on $N^*$ (using delays, instead of costs) and calculate for each node $u$ the
    value of D(u), the delay of a minimum delay path from $t$ to $u$
2.  $LABEL(s):=(0,0)$
3.  for each $v \in V$
4.          if $v \neq s$ then
5.                  $LABEL(v):=(\infty,0)$
6.          $PRED(v):=v$
7.  for each $(u,v) \in E$
8.          modified-relax($u,v$)
9.  if no node's LABEL value is updated during the previous sweep, HALT
10. goto 7
END
```

Figure 2-14: BFM-BDMCP algorithm.

Let us apply BFM-BDMCP algorithm (synchronous version) to the example in Figure 1-3.

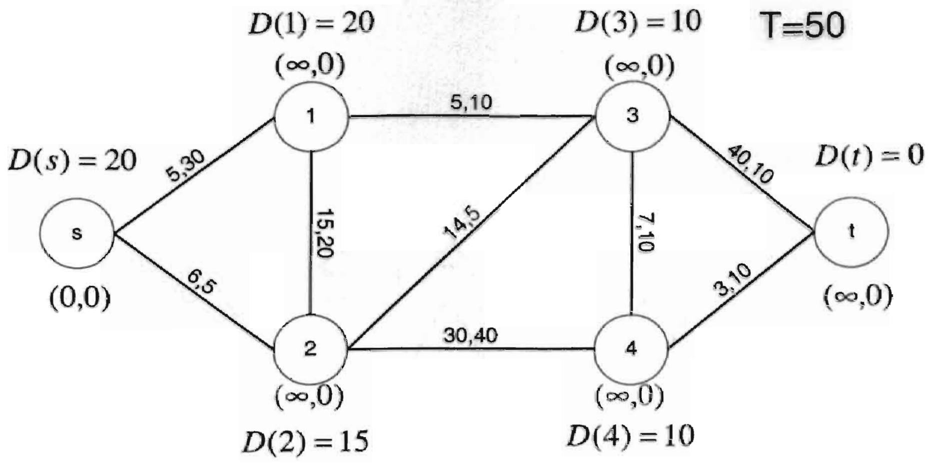

Figure 2-15: Initialization.

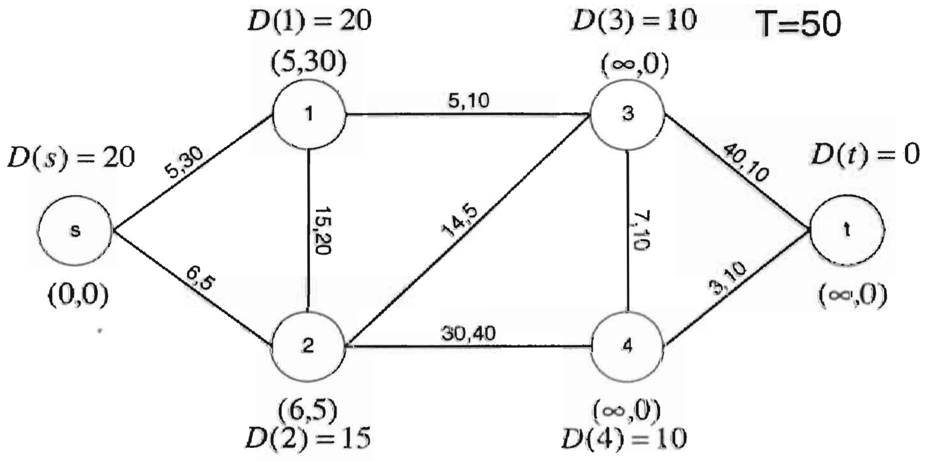

Figure 2-16: Sweep 1.

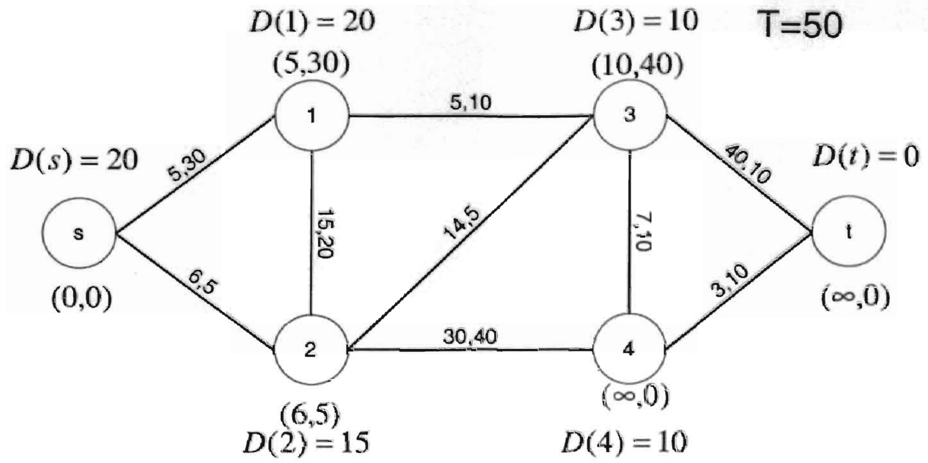

Figure 2-17: Sweep 2.

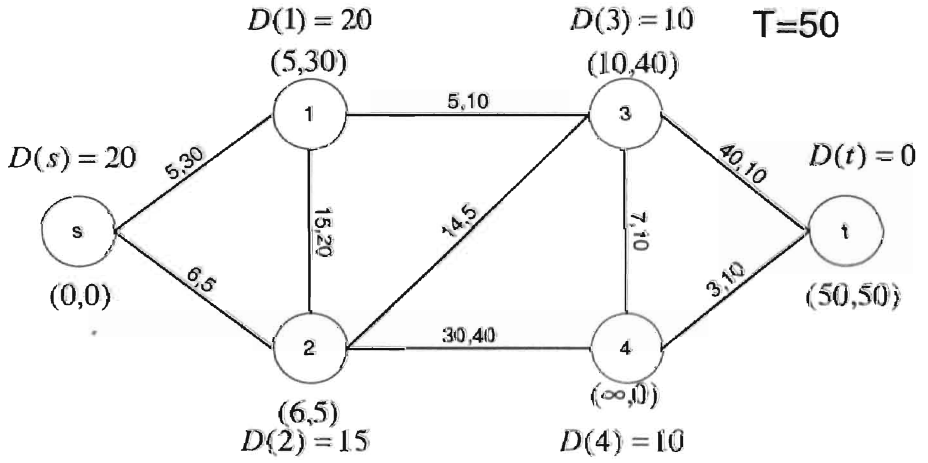

Figure 2-18: Sweep 3.

Suppose while labeling from node u, the $LABEL(v)$ of neighbor node v is updated . We can view this as "node u initiating a wave to node v". Basically this means that node u has extended a current s-u path to an s-v path.

In the case of the LHWHM algorithm, during an iteration only one node (the most recently permanently labeled node) is scanned. Thus at most n new waves (equivalently, paths) are initiated. And only one of these waves gets chosen for propagation of waves in the subsequent iteration. In fact, at most n-1 waves will get a chance to propagate new waves during the entire algorithm. On the other hand, in the case of the BFM-BDMCP algorithm, an iteration corresponds to a sweep, that is, the scanning of all the nodes of the network. This means that during a sweep a number of waves are initiated. Therefore a large number of paths become considered for further extensions to feasible s–t paths. This is because every node v whose *LABEL*(v).cost gets updated during a sweep gets a chance to initiate a wave (a new path) for exploration and does not have to wait until it becomes permanently labeled as in the case of the LHWHM algorithm. It is for this reason that the BFM-BDMCP algorithm is expected to outperform the LHWHM algorithm in terms of the cost of the solution.

Ravi[7] proved that the BFM-BDMCP algorithm terminates with a feasible s-t path if one such path exists.

2.5 Pseudopolynomial Time Algorithms for BDMCP

Although BDMCP problem is NP-hard, it can be solved by pseudopolynomial algorithm[16]. Thus if the input numbers are uniformly bounded it is polynomially solvable. The pseudopolynomial algorithm uses the idea of dynamic programming.

Dynamic programming can be applied to problems which can be solved by combining the solutions to subproblems and the subproblems are not independent. A dynamic-programming algorithm solves every subproblem just once and saves its answer in a table, therefore avoiding the work of recomputing the answer every time the subproblem is encountered.

There are two dynamic programming algorithms for BDMCP:

Algorithm A.

Let $f_j(x)$ denote the cost of a shortest path from s to j, such that the delay of the path is at most x. Assume that $d_{i,j} > 0$ for all $(i,j) \in E$.

```
ALGORITHM Dynamic-Programming-A(N,s,t,T)
1. for each vertex $j \in V$ do
2.         if $j \neq s$ then $f_j(0) := \infty$
3. for $x$:=0 to $T$
4.         $f_s(x) := 0$
5. for $x$:=1 to $T$
6.         for each vertex $j \in V$ do
7.                 if $j \neq s$ then
8.                         $f_j(x) := \min\{ f_j(x-1) , \min_{k \in V | x \geq d_{k,j}} \{ f_k(x - d_{k,j}) + c_{k,j} \} \}$
9. return $f_t(T)$
END
```

Figure 2-19: Dynamic-Programming A.

This algorithm runs in O(nT).

Algorithm B.

Let $g_j(c)$ denote the time of a quickest path from s to j, such that the cost of the path is at most c.

```
ALGORITHM Dynamic-Programming-B($N,s,t,T$)
1. for each vertex $j \in V$ do
2.         if $j \neq s$ then $g_j(0) := \infty$
3. for $c$:=0 to $OPT$
4.         $g_s(c)$:=0
5. for $c$:=1 to $OPT$
6.         for each vertex $j \in V$ do
7.                 if $j \neq s$ then
8.                         $g_j(c) := \min\{ g_j(c-1), \min_{k \in V | c \geq c_{k,j}} \{ g_k(c - c_{k,j}) + d_{k,j} \} \}$
END
```

Figure 2-20: Dynamic-Programming-B.

Here *OPT* is the optimal solution of BDMCP problem. In the above algorithm, *OPT* is not know a priori, but it satisfies that $OPT = \min\{c | g_t(c) \leq T\}$. Thus *OPT* is the first value of c for which $g_t(c) \leq T$. This algorithm runs in time $O(m \cdot OPT)$.

Chapter 3 A New Heuristic Algorithm

According to [8], LHWHM algorithm performs well on sparse networks and BFM-BDMCP algorithm generally outperforms LHWHM algorithm. But still, they fail to find the optimal solutions for some networks since they are just heuristic algorithms for BDMCP problem. In this chapter, we present an improved version of BFM-BDMCP, called K-BFM-BDMCP, which has better chance than BFM-BDMCP to find optimal solutions yet still runs in polynomial time.

3.1 *The Drawback of BFM-BDMCP*

Why does BFM-BDMCP algorithm fail to find the optimal solution for some networks? To answer this question, let us first look at the following simple example of BDMCP problem:

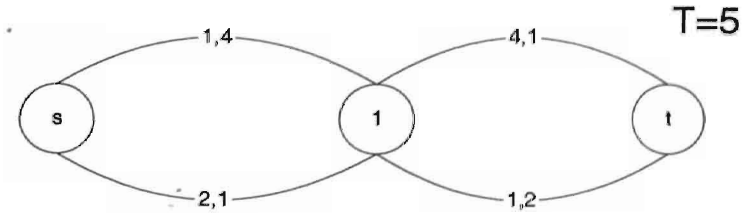

Figure 3-1: A simple example of BDMCP problem.

When applying BFM-BDMCP algorithm to this network, we have the following result:

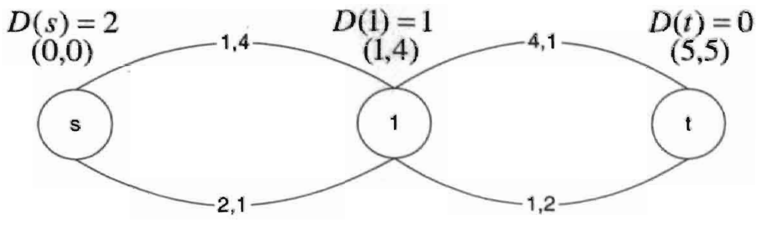

Figure 3-2: Solution found by BFM-BDMCP.

BFM-BDMCP algorithm found the solution of (5,5) and the corresponding *s*-*t* path is $s \xrightarrow{1,4} 1 \xrightarrow{4,1} t$. But obviously the optimal solution for this example is (3,3) and the optimal *s*-*t* path is $s \xrightarrow{2,1} 1 \xrightarrow{1,2} t$. The reason why it fails to find the optimal solution is that it measures the goodness of a cost-delay label only by cost and does not take delay into account. In the above example, two possible cost-delay labels for node 1 are (1,4) and (2,1), but the label (2,1) which would lead to the optimal solution was discarded because its cost value is greater than that of the label (1,4), though its delay value is less than that of the label (1,4).

Note that the LHWHM algorithm also suffers from the same problem, namely, when a node has obtained several cost-delay labels that all have chance to get to the optimal solution, it just blindly gambles on the one that has the smallest cost value.

3.2 Improving BFM-BDMCP

The cause of the problem with BFM-BDMCP algorithm is that each node holds one cost-delay label. Intuitively, the possible remedy for this problem would be increasing the memory of each node so that more labels could be stored in each node to initiate waves to the neighbors. For instance, if we allow each node to store up to three cost-delay labels in BFM-BDMCP, and during each sweep every label stored in each node needs to initiate waves to the neighbor nodes, we expect to find better solutions. Let us apply this strategy to the example in Section 3.1:

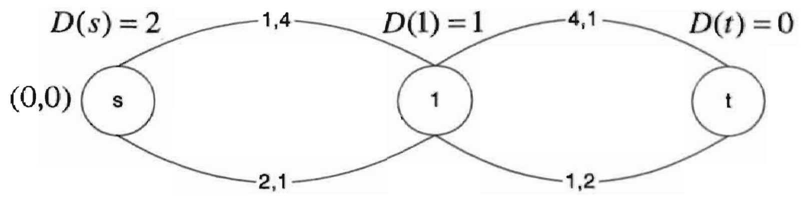

Figure 3-3: Initialization.

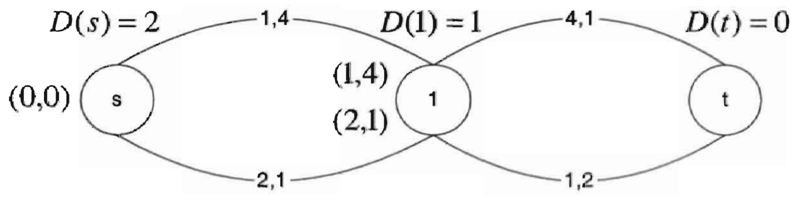

Figure 3-4: Sweep 1.

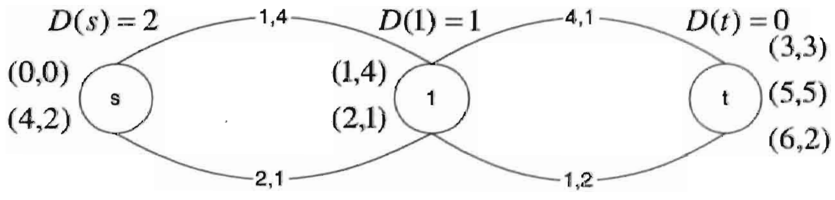

Figure 3-5: Sweep 2.

In this case, we found not only better solution than that of BFM-BDMCP algorithm, but also the optimal solution. In fact, if each node had infinite memory to remember as many cost-delay labels as it should remember, the optimal solution would guarantee to be found. Based on this idea, an optimal algorithm based on BFM-BDMCP algorithm for BDMCP problem can be obtained. Note that we use "optimal" only to mean that this algorithm is able to find the optimal solution.

In this optimal algorithm, we associate each node with a set of cost-delay labels instead of just one cost-delay label as in BFM-BDMCP. The algorithm takes n-1 sweeps and stores all the cost-delay labels for node v generated during the ith sweep in $LABEL(v,i)$. Initially, $LABEL(s,0)=\{(0,0)\}$ and $LABEL(v,0)=\phi$ for $v \neq s$. Every label in $LABEL(v,i)$ needs to initiate waves to the neighbor nodes of v during the $(i+1)$st sweep.

Below is the formal description of the optimal algorithm.

```
ALGORITHM OPTIMAL-BFM-BDMCP(N,s,t)
1.  use Dijkstra's algorithm or BFM algorithm to compute $D(v)$ for every $v \in V$
2.  for each $v$ in $V$ do
3.          $LABEL(v,0) := \phi$
4.  $LABEL(s,0) := \{(0,0)\}$
5.  for $i$:=1 to $n$-1 do
6.          for each $v \in V$ do
7.                  $LABEL(v,i) := LABEL(v,i-1)$
8.          for each edge$(u,v) \in E$ do
9.                  for each cost-delay label $(c,d)$ in $LABEL(u,i-1)$ do
```

```
10.                 $(c',d'):=(c+c_{u,v}, d+d_{u,v})$
11.                 if $d'+D(v) \le T$ then
12.                         $LABEL(v,i):=LABEL(v,i) \cup \{(c',d')\}$
13. return the cost-delay label with the smallest cost in LABEL($t$,$n$-1)
END
```

Figure 3-6: An optimal algorithm for BDMCP.

Obviously, this optimal algorithm uses the brute-force approach. Thus when it terminates, *LABEL*(t,n-1) contains all the feasible s-t paths.

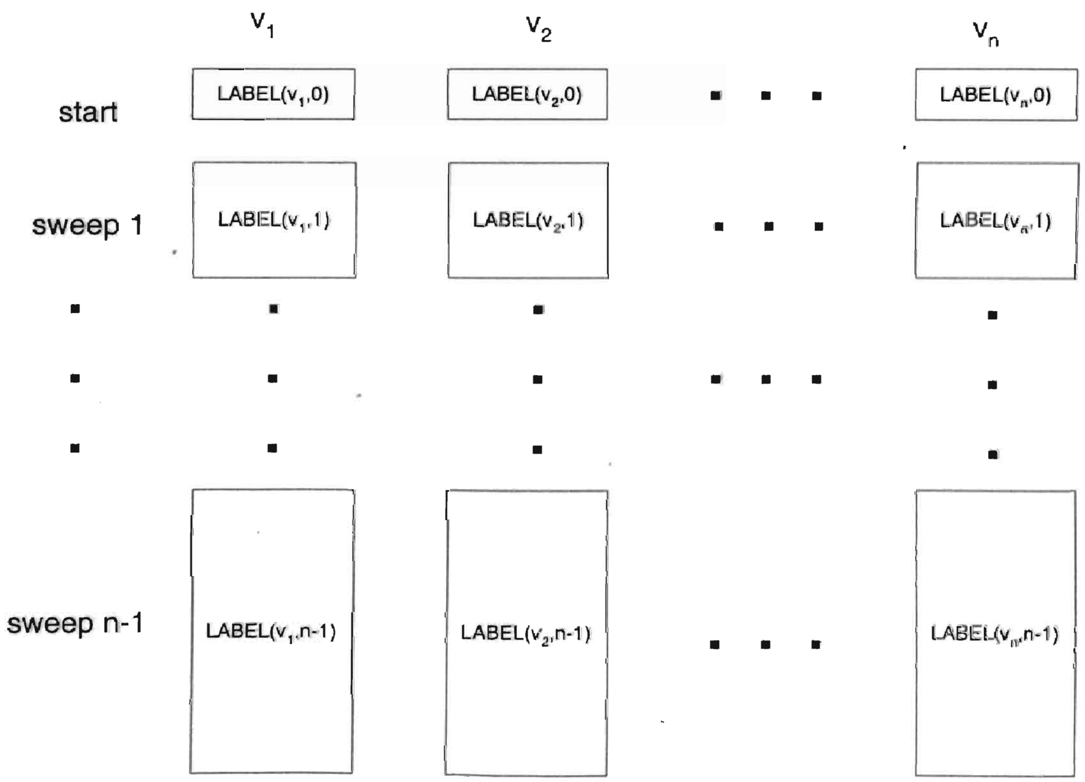

Figure 3-7: The execution of OPTIMAL-BFM-BDMCP.

Although this optimal algorithm gives us the optimal solution, it is impractical. The problem with it is that remembering all the possible cost-delay labels for each node might lead to exponential growth of the number of the cost-delay labels stored in some nodes and the running time would be exponential. In fact, this is an exponential time algorithm and it is intractable.

A natural solution for the above problem is to limit the size of the memory in each node to a constant, say *k*. This modified version of BFM-BDMCP algorithm is called K-BFM-BDMCP algorithm. Later we will see that BFM-BDMCP algorithm is a special case of K-BFM-BDMCP with *k*=1. Intuitively, if *k*>1, which means every node can remember more labels than BFM-BDMCP, K-BFM-BDMCP is more likely to yield better solutions than BFM-BDMCP. Also, since the number of labels in each label set is at most *k*, a polynomial number of labels are scanned during each sweep, thus K-BFM-BFMCP should still run in polynomial time.

In the following section, we will present K-BFM-BDMCP algorithm and show that it runs in polynomial time.

3.3 K-BFM-BDMCP

The K-BFM-BDMCP algorithm can be derived from the optimal algorithm from the previous section. The main task is to throw away certain cost-delay labels in the label sets after each sweep so that each node does not run out of memory during the execution. Now the problem is what labels we should throw away.

Definition 3.1. Let (c_1,d_1) and (c_2,d_2) be two cost-delay labels. We say that (c_1,d_1) equals (c_2,d_2) (denoted by $(c_1,d_1) = (c_2,d_2)$) if $c_1=c_2$ *and* $d_1=d_2$. We say that (c_1,d_1) dominates (c_2,d_2) (denoted by $(c_1,d_1) \geq (c_2,d_2)$) if $c_1\leq c_2$ *and* $d_1\leq d_2$, and (c_1,d_1) strictly dominates (c_2,d_2) (denoted by $(c_1,d_1) > (c_2,d_2)$) if $(c_1,d_1)\geq(c_2,d_2)$ and $(c_1,d_1)\neq(c_2,d_2)$. (The relations "$>$" and "$\geq$" on cost-delay labels are called dominance and strict dominance.)

The relation (strict) dominance on a set of cost-delay labels is clearly a partial ordering on the set. Any two distinct cost-delay labels might be compared based on the dominance relation. For two cost-delay labels l_1 and l_2, if l_1 dominates l_2 , then l_1 is better than l_2 since the cost and the delay of l_1 are no greater than the corresponding values of l_2. But if no one can dominate the other, then we cannot tell which one is better except for the destination node. The reason is that even though the cost-delay label with smaller cost value seems better than the other one with smaller delay value, later the latter one would be able to get to the destination through certain low cost and high delay path to obtain a better solution because the former one cannot get through the same path due to delay constraint. For the labels in the destination node, since we have already achieved the goal, so the one with smallest cost value is better than the others.

Now what if we had more than k cost-delay labels for a node and none of them is dominated by one another? Due to the memory limit, we need some criteria to compare these cost-delay labels and throw away some "not-so-good" ones.

Definition 3.2. Let (c_1,d_1) and (c_2,d_2) be two cost-delay labels. We say that (c_1,d_1) is biassedly better than (c_2,d_2) (denoted by $(c_1,d_1) \succ (c_2,d_2)$) if $c_1<c_2$ *and* $d_1>d_2$.

The relation "$\succ$" enforces a total ordering on a set of cost-delay labels in which no one is dominated by another. Here we have bias towards labels with larger cost value since our goal is to find the path with smallest cost and without any knowledge of future, the labels with smaller cost value would be more likely to achieve our goal.

Below is the formal description of K-BFM-BDMCP.

```
ALGORITHM K-BFM-BDMCP($N,s,t,T,k$)
1.  use Dijkstra's algorithm or BFM algorithm to compute $D(v)$ for every $v \in V$
2.  for each vertex $v \in V$ do
3.          $LABEL(v,1)$: $=\phi$
4.  $LABEL(s,1)$:=$\{(0,0)\}$
5.  for $i$: =1 to $n$-1 do
6.          for each vertex $v \in V$ do
7.                  $LABEL(v,i)$: =$LABEL(v,i-1)$
8.          for each vertex $(u,v) \in E$ do
9.                  for each cost-delay label $(c,d) \in LABEL(u,i-1)$ do
10.                         $(c',d')$:=$(c+c_{u,v}, d+d_{u,v})$
11.                         if $d'+D(v) \leq T$ then
```

```
12.                         LABEL(v,i):=LABEL(v,i) ∪ {(c',d')}
13.         for each vertex $v \in V$ do
14.                 remove all cost-delay labels in LABEL(v,i) that are dominated
                    by some other label in the same set
15.         if |LABEL(v,i)|>k then
16.                 trim LABEL(v,i) only to keep the k biassedly better cost-delay labels
17. return the cost-delay label with the smallest cost in LABEL(t,n-1)
END
```

Figure 3-8: K-BFM-BDMCP algorithm.

In the K-BFM-BDMCP algorithm, each node is associated with a set of cost-delay labels. The sweep is taken n-1 times. *LABEL*(v,i) denotes the set of cost-delay labels associated with vertex v during the ith sweep. Each cost-delay label in *LABEL*(v,i) is either copied from *LABEL*(v,i-1) or computed from some cost-delay label in *LABEL*(u,i-1) where $u \neq v$. In other words, each cost-delay label in *LABEL*(v,i) is due to some cost-label existing in the previous sweep which we call predecessor. Following the relation of predecessor, every cost-delay label l except for (0,0) in *LABEL*(s,0) can be traced back to the (0,0) label in *LABEL*(v,0). Thus we can reconstruct the paths we have already found.

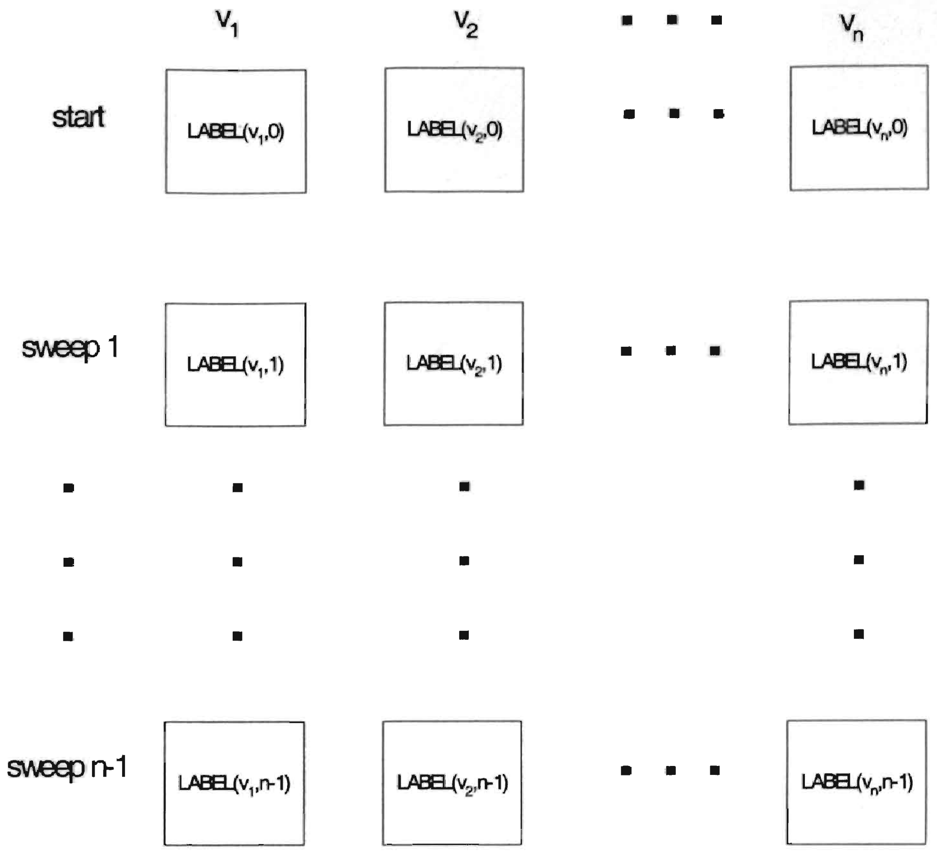

Figure 3-9: The execution of K-BFM-BDMCP.

We first prove the correctness of K-BFM-BDMCP.

Theorem 3.1. K-BFM-BDMCP returns a feasible *s*-*t* path if one such path exists.

Proof. Let the label sets in the optimal algorithm in Section 3.2 be denoted by *LABEL* and the label sets in K-BFM-BDMCP denoted by *LABEL'*. First we prove that (1) $LABEL'(v,i) \subseteq LABEL(v,i)$; (2) if $LABEL(v,i) \neq \phi$, then $LABEL'(v,i) \neq \phi$ where $v \in V$ and $0 \leq i \leq n-1$.

This can be done by induction on i.

The base case for which i=0 is obviously true since $LABEL(v,0) = LABEL'(v,0)$ for $v \in V$.

In the inductive step, we assume that the claim holds for $LABEL(v,i)$ and $LABEL'(v,i)$ for $v \in V$. Consider $LABEL(v,i+1)$ and $LABEL'(v,i+1)$ where $v \in V$. At first, $LABEL(v,i)$ is copied to $LABEL(v,i+1)$ and $LABEL'(v,i)$ is copied to $LABEL'(v,i+1)$. According to the induction hypothesis, $LABEL'(v,i+1) \subseteq LABEL(v,i+1)$ at this point. Afterwards, for any cost-delay label l that is added to $LABEL'(v,i+1)$, let l' be its predecessor. Then $l' \in LABEL'(u,i)$ for some $u \in V$. According to the induction hypothesis, $l' \in LABEL(u, i)$. Therefore l will be also added to $LABEL(v,i+1)$ through the same edge. Thus before line 13 in K-BFM-BDMCP algorithm, We have $LABEL'(v,i+1) \subseteq LABEL(v,i+1)$. After that, we only remove labels from $LABEL'(v,i+1)$, so in the end we still have $LABEL'(v,i+1) \subseteq LABEL(v,i+1)$.

In addition, if $LABEL(v,i+1) \neq \phi$ where $v \in V$, let $l \in LABEL(v,i+1)$ and l' be its predecessor, then $l' \in LABEL(u,i)$ for some $u \in V$ which means $LABEL(u,i) \neq \phi$. According to the induction hypothesis, $LABEL'(u,i) \neq \phi$. Thus the cost-delay label in $LABEL'(u,i)$ will generate a cost-delay label in $LABEL'(u,i+1)$ through the same edge through which l' generated l. Therefore $LABEL'(u,i+1) \neq \phi$.

Since *LABEL*$(t, n\text{-}1)$ contains all the feasible s-t paths, if there exists a feasible path in the network, then *LABEL*$(t, n\text{-}1) \neq \phi$ which means *LABEL'*$(t, n\text{-}1) \neq \phi$ and *LABEL'*$(t, n\text{-}1)$ $\subseteq$ *LABEL*$(t, n\text{-}1)$. In other words, it returns a feasible path.

The time complexity of K-BFM-BDMCP depends on how to implement it. In a careful implementation, we use red-black tree to implement label sets. Red-black trees are one of many search-tree schemes that are nearly balanced in order to guarantee that basic set operations such as search, insertion and deletion take $O(\lg n)$ time in the worst case where n is the number of elements in the set. The cost-delay labels can be ordered on cost values with delay values being the tie-breakers.

Theorem 3.2. K-BFM-BDMCP algorithm runs in $O(n^3 k\log(nk))$ time with space complexity $O(kn^2)$.

Proof. From the formal description of K-BFM-BDMCP algorithm, Line 1 takes $O(mn)$ if we use BFM algorithm. Line 2-3 takes $O(n)$. There are n-1 sweeps. During the ith sweep, copying label sets takes $O(nk)$ (line 6-7). After that we need to scan all the m edges. For each edge (u,v), we need to scan all the cost-delay label in LABEL$(u,i\text{-}1)$ which is of size $O(k)$. *LABEL*(v,i) can be of size $O(nk)$ in the worst case, so line 12 takes $O(\log(nk))$. Line 13-16 contains n iteration. In each iteration, line 14 actually computes the maxima of *LABEL*(v,i) which can be done in $O(nk\log(nk))$ [25]. Trimming is relatively cheaper and only takes $O(k)$. So the total running time is:

$$O(mn+n+n(nk+mk\log(nk)+n^2k\log(nk)))= O(n^3k\log(nk)).$$

The space for *LABEL* is O(kn^2) and it is the dominant factor. So the space complexity is O(kn^2).

Let us apply K-BFM-BDMCP algorithm to the example showed in Section 1.3.

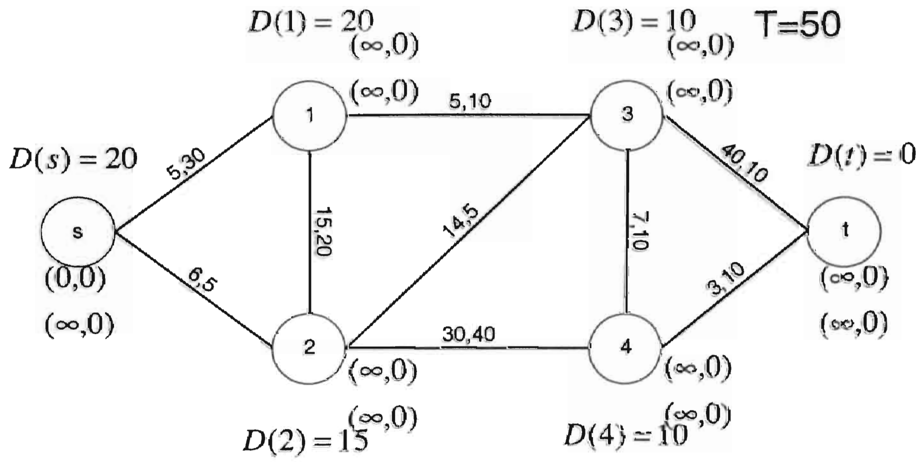

Figure 3-10: Initialization.

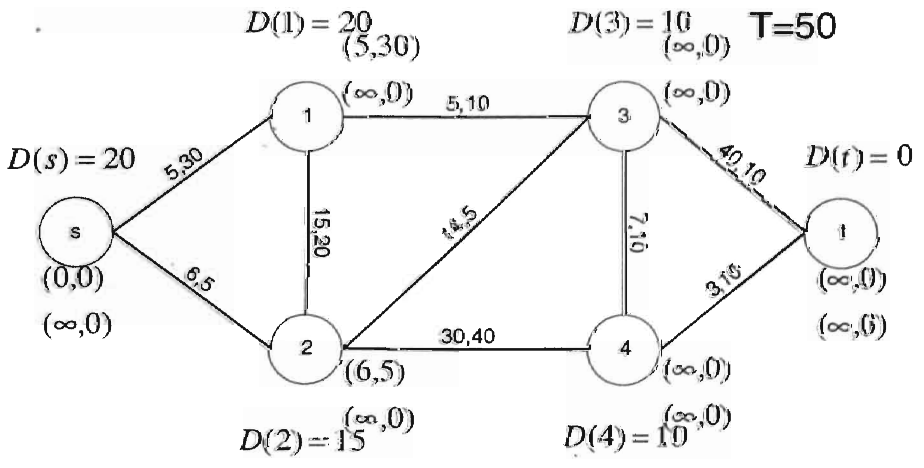

Figure 3-11: Sweep 1.

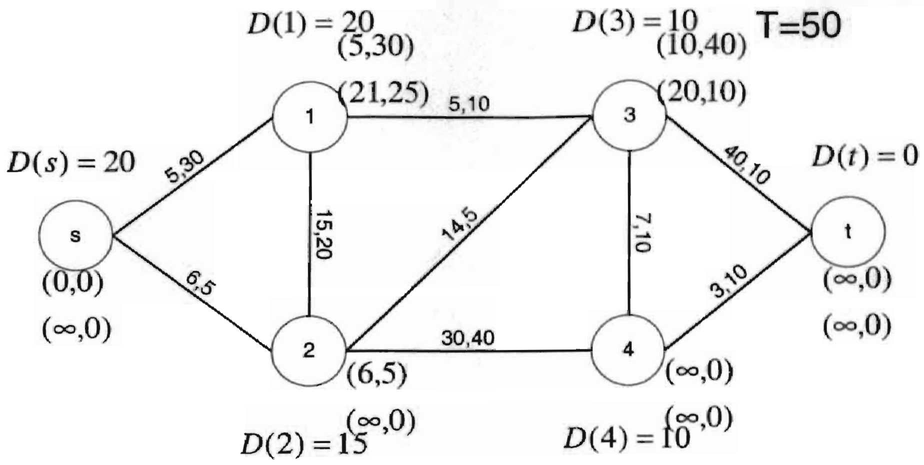

Figure 3-12: Sweep 2.

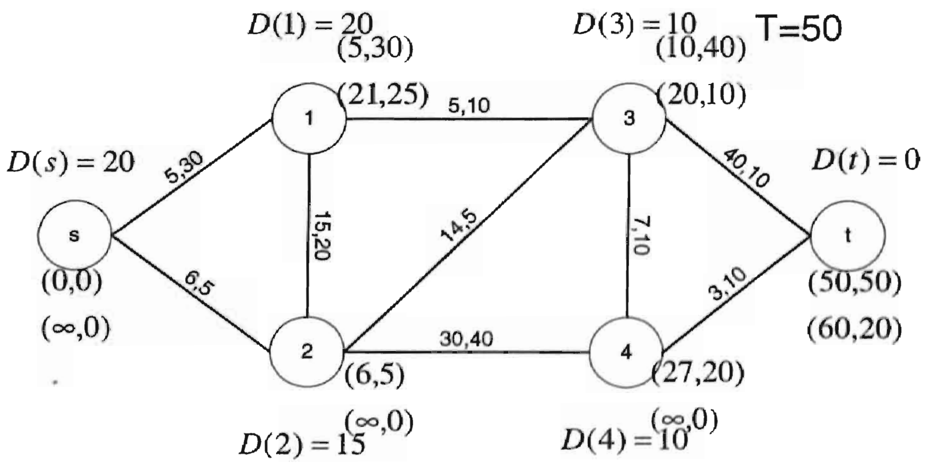

Figure 3-13: Sweep 3.

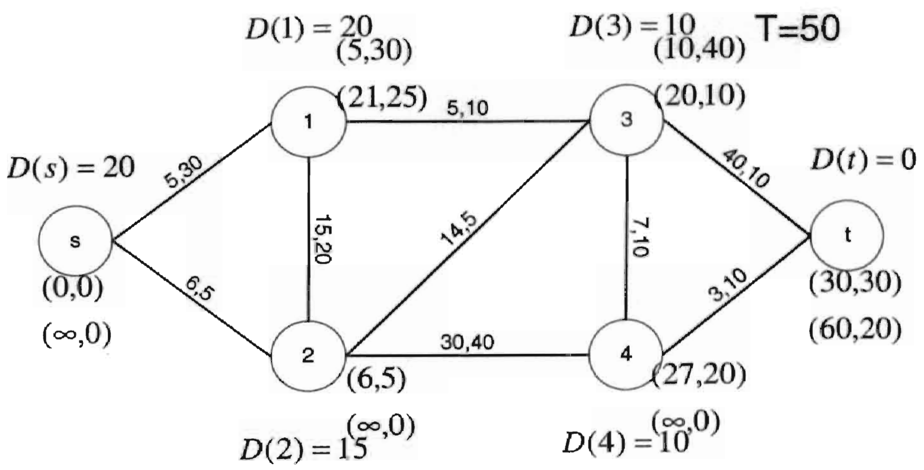

Figure 3-14: Sweep 4.

3.4 The advantage of K-BFM-BDMCP

The main advantage of K-BFM-BDMCP is its scalability. In fact from the analysis of the previous section, we can see BFM-BDMCP algorithm is a special case of K-BFM-BDMCP algorithm with k=1. Thus the idea of K-BFM-BDMCP is to trade off time with quality of solution. The greater the value of k that is fed into K-BFM-BDMCP, the longer the running time but the better chance to find better solutions. The running time is still polynomial so it is tractable. Thus this algorithm embodies the trade-off and balance between time and the quality of solution.

When employing K-BFM-BDMCP in the real world, we can make the value of k adaptive, which means the value of k is adjustable over time. A monitor program can be set up to monitor the topology change and traffic change to adjust the k value fed to the routing program that answers routing requests. Furthermore, we can use different k

values for different runs of the algorithm. In this case, each request contains the expectation of the quality of routes. If the quality of routes is very important, a bigger *k* value can be used in order to answer this request. If the request just needs a good feasible path, a lower *k* value can be used.

Another advantage of K-BFM-BDMCP is that it is also easy to implement in a distributed fashion because of the inherent nature of distributedness of BFM algorithm. In fact, Ravi [7] has proposed a distributed version of BFM-BDMCP algorithm that is easy to implement. Since K-BFM-BDMCP algorithm is based on BFM-BDMCP algorithm, its distributed version could be obtained similarly. This is also our future work.

Chapter 4 Experimental Evaluation

In this chapter, we first present our experimental results on the relationship between the value of k and the performance of K-BFM-BDMCP algorithm. Then we do a comparative evaluation with BFM-BDMCP algorithm and K-BFM-BDMCP algorithm.

4.1 Test Graphs

We use two kinds of graphs to do our experiments. One is regular graph and the other is irregular graph.

4.1.1 Regular Graph

The kind of regular graph we use is proposed by Harary[22]. A regular graph $H_{k,n}$, where k refers to the vertex degree and n refers to the number of vertices, is defined as follows.

- *Case* 1: k even. Let $k = 2r$. Then $H_{2r,n}$ has vertices $v_0, v_1, v_2, \ldots, v_{n-1}$ and two vertices v_i and v_j are adjacent if $i - r \leq j \leq i + r$, where addition is modulo n.
- *Case* 2: k odd, n even. Let $k = 2r + 1$. $H_{2r+1,n}$ is constructed by first constructing $H_{2r,n}$ and then adding edges joining vertex v_i to vertex v_j for $i = 1, 2, \ldots, n/2$ and $j = i + n/2$ (mod n).
- *Case* 3: k odd, n odd. Let $k = 2r + 1$. $H_{2r+1,n}$ is constructed by first constructing $H_{2r,n}$ and then adding edges joining v_i and v_j for $i = 0, 1, 2, \ldots, (n-1)/2$ and $j = i + (n+1)/2$ (mod n).

As an example, graph $H_{4,8}$ is shown in Fig. 4-1.

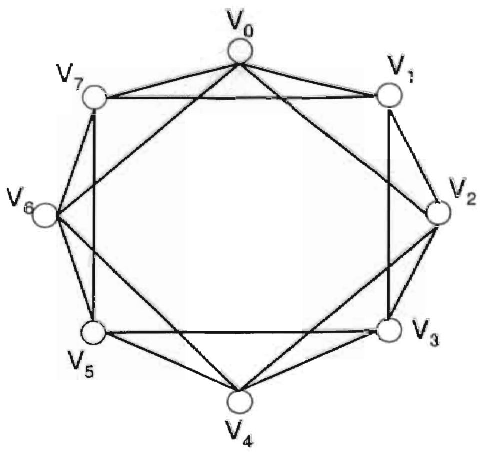

Figure 4-1: Regular graph $H_{4,8}$.

Regular graphs can be used to model interconnection networks and dense networks.

4.1.2 Irregular Graph

Irregular graphs are like random graphs in the sense that they have no well-defined structures yet abide by some desired properties. Since BDMCP problem is one of the QoS-based routing problems, we need to experiment our algorithms on the Internet. Due to the immense scale of the Internet, deploying an Internet-wide experiment is nearly impossible. So we evaluate our algorithms using randomly generated network topologies.

Currently, there are five Internet topology generators available: Waxman[17], Tiers[18], GT-ITM[19], Inet[20] and BRITE[21]. We choose Inet because it is based on Autonomous System (AS) connectivity in the Internet and it generates topologies that best approximate the actual Internet AS topology.

Empirical evidence shows that Internet AS topology exhibits power-law connectivity which may also hold for route topologies. Power-law graph structure induces "centers" where connectivity is concentrated on a few nodes with most vertices possessing sparse connectivity.

4.1.3 Generating Parameters

For an instance of BDMCP problem, we need not only the underlying graph, but also the other parameters such as link costs and delays, source node, destination node and the delay constraint.

Link costs and delays are selected in one of the following two ways:

1. Randomly generated costs and delays in the range 1 to *max*;
2. Randomly generated costs and delays such that $d_{i,j} + c_{i,j} = max$.

Here *max* is a predefined constant. The motivation for the choice (2) above is to test the heuristics under what we believe to be a worst-case scenario, and to capture the fact that delay and cost are inversely related.

For the source node and destination node, our criterion is the longer distance between them, the better. Thus first we pick the node with the smallest degree as the source node. Then we apply Dijkastra's algorithm to obtain the minimum delay from the source node to the others node and pick the one with the maximum minimum delay as the destination node. Finally, we assign twice this maximum minimum delay as the delay constraint.

4.2 The Relationship between k and Performance of K-BFM-BDMCP

The results of the experiments are presented in Table 4-1. Here we choose regular graphs with random costs and delays as the test graphs. As expected, the quality of solution is improved in terms of cost when we increase the k value, and the running time is also increasing but the increments are very steady.

From Figure 4.2, we can see that there is a dramatic improvement from *k*=2 to *k*=5, but the running time is only increased by a small number, as shown in Figure 4.3. So in this case *k*=5 seems to obtain the balance between the quality of solution and running time.

Table 4-1: Comparison of the performance of different k values

GRAPH	NODES	K-BFM-BDMCP								OPTIMAL
		k=2		k=5		k=7		k=9		
		cost	time	cost	tIme	cost	time	cost	time	cost
k=5	400	1659	3.0	1640	3.8	1640	4.6	1636	5.3	1636
k=5	450	2069	3.4	1890	4.8	1890	6.0	1890	6.9	1890
k=7	500	1396	3.1	1355	4.5	1355	5.5	1348	6.5	1330
k=7	550	1357	3.4	1261	4.8	1243	5.9	1235	7.4	1226
k=7	600	1882	3.6	1685	5.4	1654	6.8	1638	8.4	1417
k=9	650	1766	3.6	1321	5.4	1317	6.6	1293	8.2	1188
k=9	700	1413	3.8	1301	6.1	1296	7.4	1296	9.3	1232

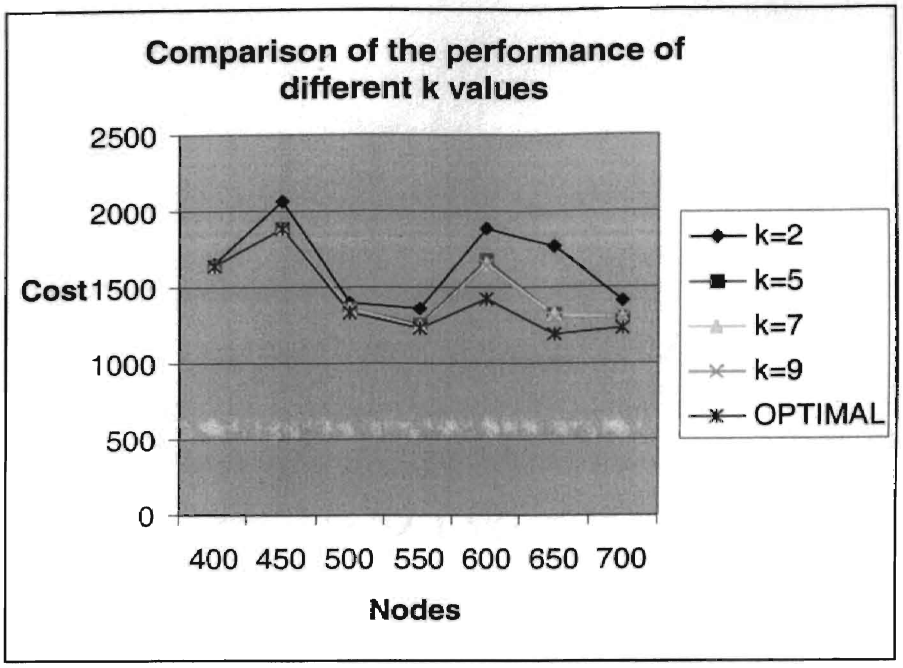

Figure 4-2: Cost comparison of the performance of different *k* values.

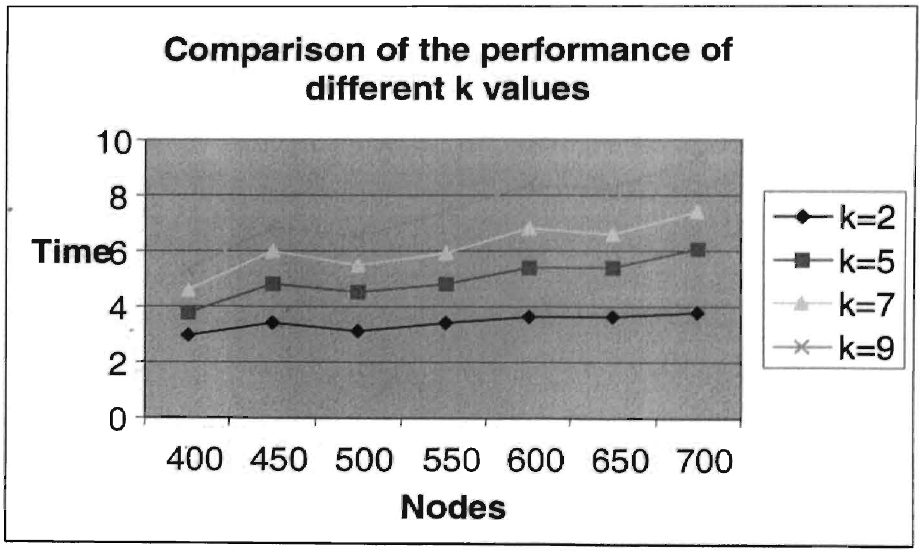

Figure 4-3: Time comparison of the performance of different *k* values.

4.3 Comparison of BFM-BDMCP Algorithm and K-BFM-BDMCP Algorithm

The comparison results are presented in Tables 4-2, 4-3, 4-4, and 4-5. From the results we can see that for regular graphs K-BFM-BDMCP algorithm does result in a considerable improvement over BFM-BDMCP at the cost of a small amount extra time, as shown in Figures 4-4, 4-5, 4-6, and 4-7. But for irregular graphs the performance of K-BFM-BDMCP is worse than that of BFM-BDMCP because they both obtain the optimal solution most of time and the running time of K-BFM-BDMCP is longer, as shown in Figures 4-8 and 4-9. This is due to the fact that there are more routing choices in regular graphs than in irregular graphs. Recall that we use the Inet Internet topology generator to generate irregular graphs. The Internet has the power-law structure which means there are a few super routers that take control of the traffic over the Internet and there are few choices when doing the routing.

Table 4-2: Comparison using regular graph and random costs and delays

GRAPH	NODES	EDGES	BOUND	BFM-BDMCP			K-BFM-BDMCP			OPTIMAL	
				cost	delay	time	cost	delay	time	cost	delay
k=5	400	1000	2610	1659	2610	1.1	1640	2602	4.0	1636	2602
k=5	450	1125	2946	2081	2895	1.2	1890	2937	4.8	1890	2937
k=7	500	1750	1848	1703	1848	1.2	1355	1845	4.6	1330	1845
k=7	550	1925	1986	1378	1971	1.2	1261	1968	4.8	1226	1977
k=7	600	2100	2250	1897	2250	1.2	1685	2241	5.4	1417	2246
k=9	650	2925	1436	1881	1434	1.4	1321	1435	5.2	1188	1432
k=9	700	3150	1563	1413	1561	1.4	1301	1539	6.1	1232	1556

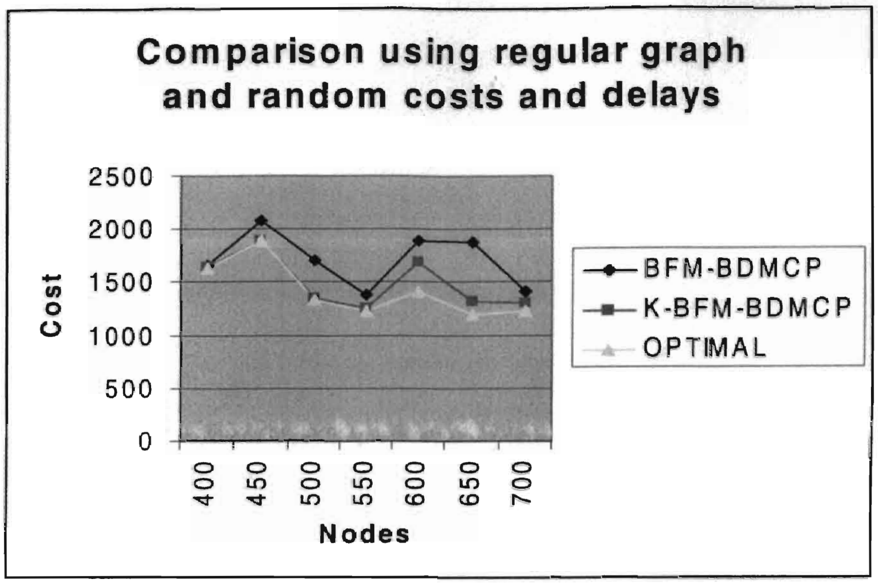

Figure 4-4: Cost comparison using regular graph and random costs and delays.

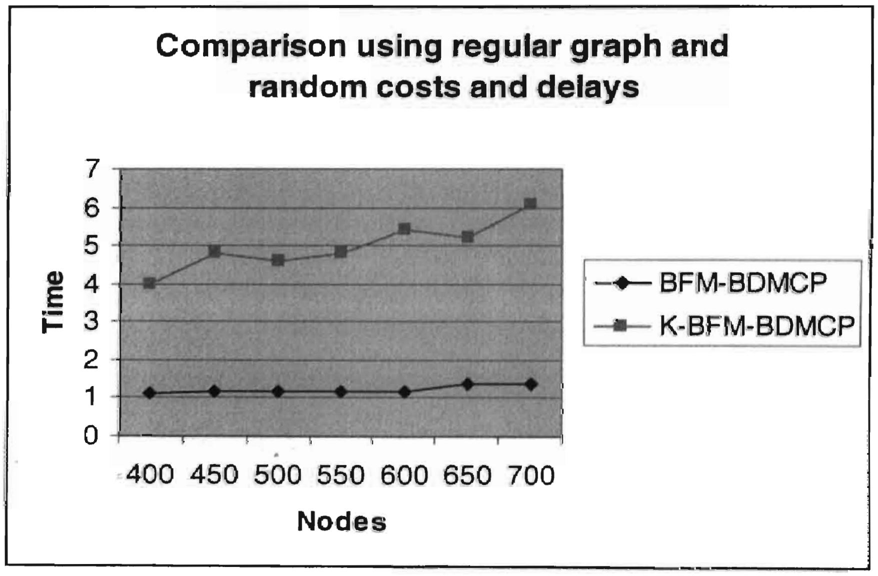

Figure 4-5: Time comparison using regular graph and random costs and delays.

Table 4-3: Comparison using regular graph and related costs and delays

GRAPH	NODES	EDGES	BOUND	BFM-BDMCP			K-BFM-BDMCP			OPTIMAL	
				cost	delay	time	cost	delay	time	cost	delay
k=5	400	1000	5161	4694	4906	1.1	4694	4906	2.7	4639	5161
k=5	450	1125	5891	5088	5712	1.1	4924	5876	3.3	4901	5891
k=7	500	1750	3260	6356	3244	1.2	5943	3257	3.0	5340	3260
k=7	550	1925	4006	5794	4006	1.2	5196	4004	3.2	4594	4006
k=7	600	2100	4081	7721	4079	1.2	6723	4077	3.6	6217	4081
k=9	650	2925	2925	6078	2922	1.3	6076	2924	3.4	5075	2925
k=9	700	3150	2801	8201	2799	1.4	8199	2801	3.6	6399	2801

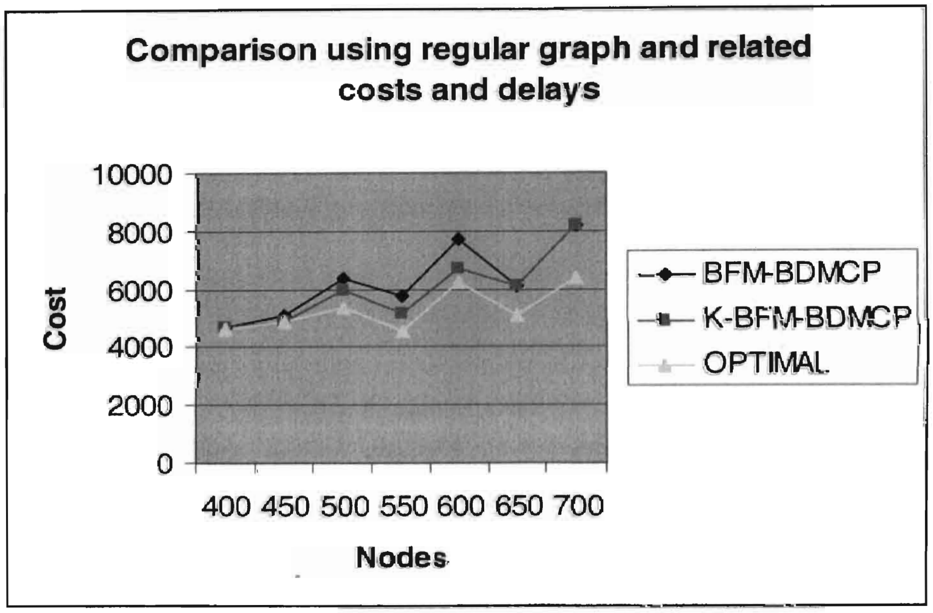

Figure 4-6: Cost comparison using regular graph and related costs and delays.

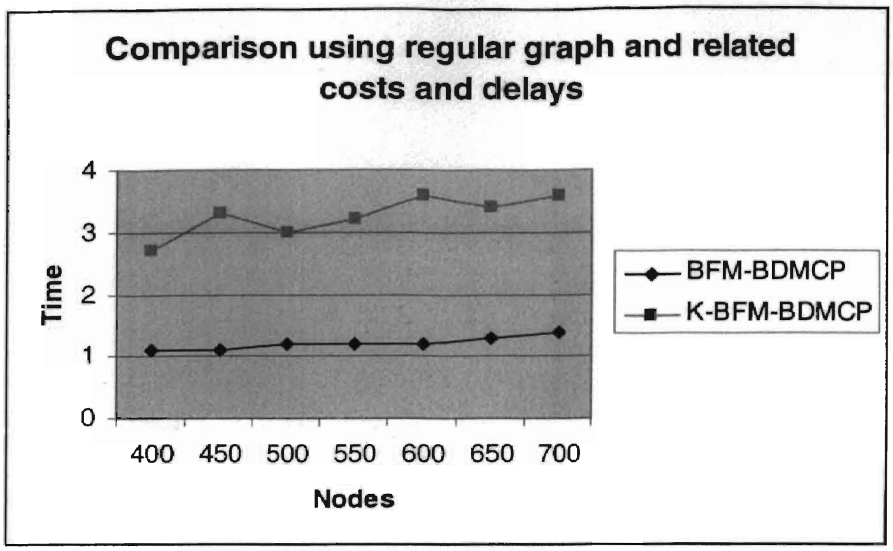

Figure 4-7: Time comparison using regular graph and related costs and delays.

Table 4-4: Comparison using irregular graph and random costs and delays

NODES	EDGES	BOUND	BFM-BDMCP			K-BFM-BDMCP			OPTIMAL	
			cost	delay	time	cost	delay	time	cost	delay
3050	4810	626	303	419	2.2	303	419	6.3	303	419
3100	4904	610	263	407	2.1	263	407	6.3	263	407
3150	4998	555	218	423	2.2	218	423	5.1	218	423
3200	5094	615	250	407	2.4	250	407	5.6	250	407
3250	5189	733	185	450	2.2	185	450	6.7	185	450
3300	5284	700	261	560	2.4	261	560	5.2	261	560
3350	5379	503	234	394	2.4	234	394	5.7	234	394
3400	5474	465	149	283	2.4	149	283	4.7	149	283

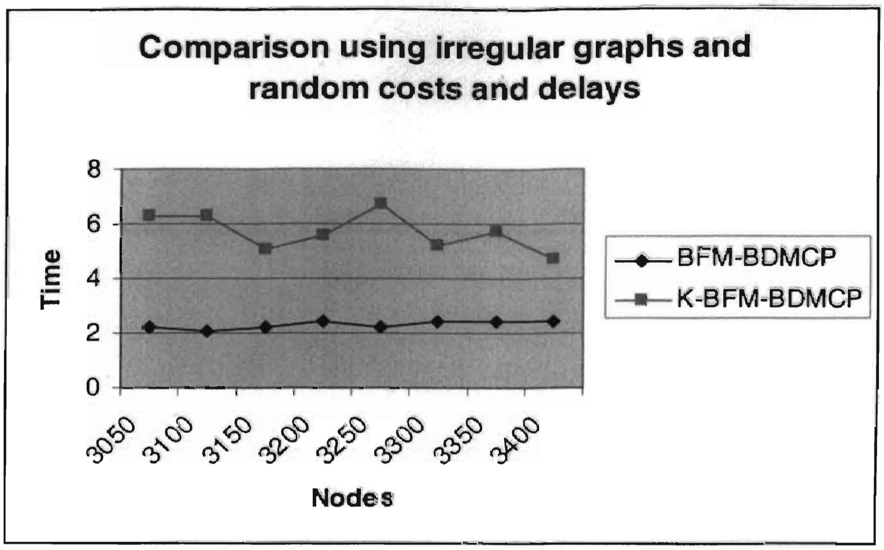

Figure 4-8: Time comparison using irregular graphs and random costs and delays.

Table 4-5: Comparison using irregular graph and related costs and delays

NODES	EDGES	BOUND	BFM-BDMCP			K-BFM-BDMCP			OPTIMAL	
			cost	delay	time	cost	delay	time	cost	delay
3050	4810	1506	311	1489	2.2	294	1506	6.0	294	1506
3100	4904	1464	297	1303	2.2	297	1303	5.8	297	1303
3150	4998	1336	286	1114	2.3	286	1114	6.2	286	1114
3200	5094	1472	323	1277	2.2	323	1277	6.5	323	1277
3250	5189	1758	16	1584	2.3	16	1584	6.8	16	1584
3300	5284	1684	12	1612	2.3	12	1612	6.3	12	1612
3350	5379	1206	308	1092	2.6	308	1092	6.1	308	1092
3400	5474	1118	245	755	2.4	245	755	5.6	245	755

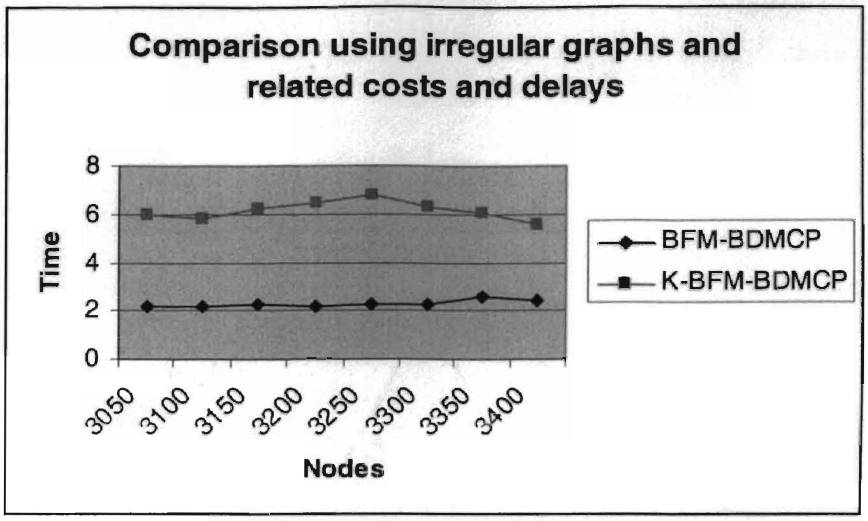

Figure 4-9: Time comparison using irregular graphs and related costs and delays.

Chapter 5 Conclusion and Future Work

In this thesis, we have studied the problem of finding the minimum cost path from a source node to a destination node satisfying a delay constraint in a network represented by a directed graph. We call this problem the Bounded-Delay Minimum-Cost Path Problem (BDMCP). This problem is NP-hard and people have been taking two approaches to attack this problem: heuristics scheme and approximation scheme. We adopt the heuristics scheme.

We began by reviewing several important algorithms related to BDMCP problem. Among them BFM-BDMCP is of our most interest because of its efficiency and effectiveness.

We then discussed the drawback of BFM-BDMCP and proposed a new algorithm based on the BFM-BDMCP algorithm. We call this new algorithm the K-BFM-BDMCP algorithm. K-BFM-BDMCP is expected to find better solutions than BFM-BDMCP at the cost of longer, but still polynomial, running time. We proved its correctness.

An experimental evaluation has been carried out to compare the performance of K-BFM-BDMCP with different k values and comparing the performance of the BFM-BDMCP and K-BFM-BDMCP. The comparison is with respect to the cost of the resulting path, and the time taken for the execution of the algorithms on different types of graphs. The experimental results show that K-BFM-BDMCP algorithm gives paths with better costs when compared to the BFM-BDMCP algorithm and that their times of execution are comparable.

Future work would be improving K-BFM-BDMCP algorithm by modifying the way it trims the cost-delay label sets associated with the nodes when they are out of space. Also, we plan to design a distributed version of K-BFM-BDMCP algorithm.

References

[1] E. Crawley, R.Nair, B.Rajagopalan, H.Sandick, "A Framework for QoS-based Routing in the Internet", *RFC2386*, Aug. 1998, 37 pages. Available: http://www.ietf.org/rfc/rfc2386.txt

[2] Wei Sun, "QoS/Policy/Constraint Based Routing", Dec. 1999. Available: http://www.cis.ohio-state.edu/~jain/cis788-99/qos_routing/index.html

[3] Z. Wang and J. Crowcroft, "Quality-of-Service Routing for Supporting Multimedia Application", *IEEE on Selected Areas in Communications*, vol. 14, no.7, pp.1228-1234, Sep. 1996.

[4] K. M. Chandy and J. Mishra, "Distributed Computation on Graphs: Shortest Path Algorithms", *CACM*, vol. 25(11), pp. 833-837, Nov. 1982.

[5] Gang Luo, Kaiyuan Huang, Jianli Wang,Chris Hobbs and Ernst Munter, "Multi-Qos Constraints Based Routing for IP and ATM Networks", *Proceedings of IEEE Workshop on QoS Support for Real-Time Internet Applications*, June 1999, Vancouver, Canada.

[6] Shigang Chen, Klara Nahrstedt, "On Finding Multi-Constrained Paths", *Proceedings of IEEE International Conference on Communications (ICC 98)*, pp.874-879, June 1998, Atlanta, Georgia.

[7] Ravi Ravindran, "Distributed and Sequential Heuristics for QoS Routing in Communication Networks", *M.S. Thesis*, June 2000, Computer Science Dept. University of Oklahoma, Norman, Oklahoma.

[8] Ravi Ravindran, Krishnaiyan Thulasiraman, Anindya Das, Kaiyuan Huang, Gang Luo, Guoliang Xue, "Quality of Service Routing: Heuristics and Approximation Schemes with a Comparative Evaluation", *ISCAS 2002*, May 2002, Scottsdale, Arizona.

[9] Shigang Chen, "Routing Support for Providing Guaranteed End-to-End Quality-of-Service", *Ph.D. thesis*, May 1999, University of Illinois at Urbana-Champaign.

[10] S.Shenker, C.Patridge, R.Guerin, "Specification of Guaranteed Quality of Service", *Network Working Group, RFC 2212.*

[11] S.Chen and K.Nahrsedt, "An Overview of Quality of Service Routing for NextGeneration High Speed Networks: Problems and Solutions", *IEEE Network Magazine,* vol. 12 (6), 1998.

[12] Douglas S.Reeves, Hussein F.Salama, "A Distributed algorithm for Delay-Constrained Unicast Routing". *IEEE/ACM transactions on Networking*, vol. 8, no 2, pp. 239-250, Apr. 2000.

[13] Jaffery M.Jaffe, "Algorithms for finding Paths with Multiple Constraints", *Networks*, vol. 14 (1984), pp. 95-116.

[14] Roch A.Gue̍rin, "QoS Routing in Networks with Inaccurate Information: Theory and Algorithms,", *IEEE/ACM Transactions on Networking,* vol.7, no.3, June 1999.

[15] Q.Ma and P.Steenkiste, "Quality of Service Routing with Performance Guarantees," *Proceedings of 4th Int'l IFIP Workshop QoS*, May 1997.

[16] R.K.Ahuja, T.Magnanti and J.B.Orlin, "Network Flows Theory, Algorithms and Applications", *Prentice Hall*, 1993.

[17] B.M. Waxman. Routing of Multipoint Connections. *IEEE Journal of Selected Areas in Communication*, vol. 6(9), pp. 1617-1622, Dec. 1988.

[18] M. Doar. A Better Model for Generating Test Networks. *Proceedings Of IEEE COLBECOM*, Nov. 1996.

[19] K. Calvert, M.B. Doar, and E.W.Zegura. "Modeling Internet Topology". *IEEE Communications Magazine*, June 1997.

[20] C. Jin, Q. Chen, and S.Jamin. Inet: Internet topology generator. *Tech.rep.cse-tr-433-00*, University of Michigan EECS Department, 2000.

[21] Alberto Medina and Ibrahim Matta. Brite: A flexible generator of internet topologies. *Technical Report BU-CS-TR-2000-005*, Boston University, Boston, Massachusetts, 2000.

[22] K. Thulasiraman and M.N.S.Swamy, Graphs: Theory and Algorithms. *JohnWiley and Sons*, 1992.

[23] Hassin R, "Approximation Schemes for the Restricted Shortest Path Problem", *Mathematics of Operation Research*, vol. 17(1), pp. 36-42.

[24] R. Widyono, "The design and evaluation of routing algorithms for real-time channels", *Technical Report TR-94-024*, University of California at Berkeley, June 1994.

[25] Kung, H.T., "On the computational complexity of finding the maxima of a set of vectors", *Proceeding of 15th Annual IEEE Symposium on Switching and Automata Theory*, pp. 117-121, Oct. 1972.

Appendices

Code Listings

This appendix contains the code listings of the Java programs that we used to do our experimental evaluations.

BFM.java

```
//Usage: java BFM network_filename
class BFM
{
   public static void main(String args[]){
      Network N=null;
      try {
         N=new Network(args[0]);
      } catch (Exception e){
         System.out.println(e.getMessage());
         System.exit(1);
      }

      int prevCLABEL[]=new int[N.numNode];
      int CLABEL[]=new int[N.numNode];
      int prevDLABEL[]=new int[N.numNode];
      int DLABEL[]=new int[N.numNode];
      int D[]=new int[N.numNode];
      int PRED[]=new int[N.numNode];

      //initialization
      LHWHM.compute_D(N,D);

      for (int i=0;i<N.numNode;i++) {
         CLABEL[i]=Network.INFINITY;
         DLABEL[i]=0;
         PRED[i]=i;
      }
      CLABEL[N.s]=0;

      boolean hasUpdate;

      while (true) {
         for (int u=0;u<N.numNode;u++){
            prevCLABEL[u]=CLABEL[u];
            prevDLABEL[u]=DLABEL[u];
         }

         hasUpdate=false;

         for (int u=0;u<N.numNode;u++)
      for (int j=0;j<N.node[u].edge.length;j++) {
        int edge=N.node[u].edge[j];
        int v;
        if (u==N.edge[edge].vertex[0]) v=N.edge[edge].vertex[1];
        else v=N.edge[edge].vertex[0];

        if (prevCLABEL[u]+N.edge[edge].c<CLABEL[v] &&
            prevDLABEL[u]+N.edge[edge].d+D[v]<=N.T ) {
            CLABEL[v]=prevCLABEL[u]+N.edge[edge].c;
            DLABEL[v]=prevDLABEL[u]+N.edge[edge].d;
            hasUpdate=true;
        }
      }
    if (!hasUpdate) break;
    }

    System.out.println(CLABEL[N.t]+","+DLABEL[N.t]);
  }
}
```

Convert.java

```
// Convert a network file generated by Inet to a file we need
// Usage: java Convert inet_file random_seed method max
// method=1: cost=1..max and delay=1..max
// method=2: cost+delay=max
import java.util.*;
import java.io.*;

class Convert
{
  public static void main(String args[]) throws Exception {
    String inet_file=args[0];
    int random_seed=Integer.parseInt(args[1]);
    int method=Integer.parseInt(args[2]);
    int max=Integer.parseInt(args[3]);

    BufferedReader in =
      new BufferedReader(
        new FileReader(inet_file));
    String s = new String();
    s = in.readLine();
    StringTokenizer st = new StringTokenizer(s);
    int numNode=Integer.parseInt(st.nextToken());
    int numEdge=Integer.parseInt(st.nextToken());
    System.out.println(numNode+" "+numEdge);

    Random ran=new Random(random_seed);

    for (int i=0;i<numNode;i++) s=in.readLine();

    for (int i=0;i<numEdge;i++) {
      s=in.readLine();
      st = new StringTokenizer(s);
      int v1=Integer.parseInt(st.nextToken());
```

```
            int v2=Integer.parseInt(st.nextToken());
            int weight=Integer.parseInt(st.nextToken());
            int cost,delay;
            delay=1+(max-1)*weight/9999;
            //delay=ran.nextInt(max-1)+1;
            if (method==1)
               cost=ran.nextInt(max-1)+1;
            else cost=max-delay;

            System.out.println(v1+" "+v2+" "+cost+" "+delay);
         }
         in.close();
      }
   }
```

DynamicProgramming.java

```
//Optimal Solution
//assuming d(i,j)>0
class DynamicProgramming
{
   public static void main(String args[]){
      Network N=null;
      try {
         N=new Network(args[0]);
      } catch (Exception e){
         System.out.println(e.getMessage());
         System.exit(1);
      }

      int D[]=new int[N.numNode];
      LHWHM.compute_D(N,D);

      //initialization
      int f[][][]=new int[N.numNode][N.T+1][3];

      for (int i=0;i<N.numNode;i++)
         f[i][0][0]=Network.INFINITY;
      f[N.s][0][0]=0;

      for (int x=1;x<=N.T;x++) {
         for (int j=0;j<N.numNode;j++) {
            if (j==N.s) continue;
            //compute f[j][x]
            int min=f[j][x-1][0];
            int minj=j;
            int minx=x-1;

            for (int i=0;i<N.node[j].edge.length;i++) {
               int edge=N.node[j].edge[i];
               int k;
               if (j==N.edge[edge].vertex[0])
                  k=N.edge[edge].vertex[1];
               else k=N.edge[edge].vertex[0];
            if (x<N.edge[edge].d) continue;
            if (f[k][x-N.edge[edge].d][0]+N.edge[edge].c<min) {
               min=f[k][x-N.edge[edge].d][0]+N.edge[edge].c;
               minj=k;
               minx=x-N.edge[edge].d;
            }
         }
         f[j][x][0]=min;
         f[j][x][1]=minj;
         f[j][x][2]=minx;
      }
   }

   int totalDelay=0;
   int j=N.t;
   int x=N.T;
   while (j!=N.s || x!=0) {
      int jj,xx;
      jj=f[j][x][1];
      xx=f[j][x][2];
      if (j!=jj) totalDelay+=delay(N,jj,j);
      j=jj;
      x=xx;
   }
   System.out.println(f[N.t][N.T][0]+","+totalDelay);
}

//assume there is at most one edge between any two vertices
static int delay(Network N,int k,int j){
   for (int i=0;i<N.node[k].edge.length;i++) {
      int edge=N.node[k].edge[i];

      if ((k==N.edge[edge].vertex[0] && j==N.edge[edge].vertex[1]) ||
          (k==N.edge[edge].vertex[1] && j==N.edge[edge].vertex[0]))
         return N.edge[edge].d;
   }
   return Network.INFINITY;
}
}
```

Edge.java

```
class Edge
{
   int vertex[]=new int[2];

   int c; //cost
   int d; //delay

   public Edge(int vertex0,int vertex1,int c,int d){
      vertex[0]=vertex0;
      vertex[1]=vertex1;
      this.c=c;
      this.d=d;
   }
}
```

KBFM.java

```
import java.util.*;

class KBFM
{
   public static void main(String args[]){
      Network N=null;
      try {
         N=new Network(args[0]);
      } catch (Exception e){
         System.out.println(e.getMessage());
         System.exit(1);
      }

      int k=Integer.parseInt(args[1]);

      Label prevLabel[]=new Label[N.numNode];
      Label label[]=new Label[N.numNode];
      int D[]=new int[N.numNode];

      //initialization
      LHWHM.compute_D(N,D);

      for (int i=0;i<N.numNode;i++)
         label[i]=new Label();
      label[N.s].add(new CostDelayLabel(0,0,null));

      boolean hasUpdated;
      int sweep=0;

      while (true) {
         sweep++;

         for (int u=0;u<N.numNode;u++)
            prevLabel[u]=(Label) label[u].clone();

         hasUpdated=false;

         for (int u=0;u<N.numNode;u++) {
            Iterator it=prevLabel[u].iterator();
            while (it.hasNext()) {
               CostDelayLabel cdp_u=(CostDelayLabel) it.next();

               if (cdp_u.hasScanned) continue;

               for (int j=0;j<N.node[u].edge.length;j++) {
                  int edge=N.node[u].edge[j];
                  int v;
                  if (u==N.edge[edge].vertex[0]) v=N.edge[edge].vertex[1];
                  else v=N.edge[edge].vertex[0];

                  int cost=cdp_u.c+N.edge[edge].c;
```

```
                int delay=cdp_u.d+N.edge[edge].d;

                if (delay+D[v]<=N.T) {
                   if (label[v].add(new CostDelayLabel(cost, delay,
cdp_u)))
                      hasUpdated=true;
                }
             }

             cdp_u.hasScanned=true;
          }//finish scanning u
       } //finish sweeping

       if (! hasUpdated) break;

       for (int u=0;u<N.numNode;u++)
          label[u].trim(k);
     }

     CostDelayLabel result=label[N.t].first();
     System.out.println(result.c+","+result.d);
   }

   static class Label implements Cloneable{
     TreeSet holder;

     public Label(){
        holder=new TreeSet();
     }

     public Object clone() {
        Label lbl=new Label();
        lbl.holder=(TreeSet) holder.clone();
        return lbl;
     }

     public Iterator iterator(){
        return holder.iterator();
     }

     public CostDelayLabel first(){
        return (CostDelayLabel) holder.first();
     }

     public int size(){
        return holder.size();
     }

     public void print(){
        Iterator it=iterator();
        while (it.hasNext()) {
           CostDelayLabel cdp=(CostDelayLabel) it.next();
           System.out.print("("+cdp.c+","+cdp.d+")");
        }
        System.out.println();
     }
```

```
    public boolean add(CostDelayLabel cdp){
      if (!holder.add(cdp))
        return false;

      CostDelayLabel tmp=null;
      SortedSet ss=holder.headSet(cdp);

      if (! ss.isEmpty()) {
        tmp=(CostDelayLabel) ss.last();
        if (tmp.isDominating(cdp)) {
          holder.remove(cdp);
          return false;
        }
      }

      //remove all cost-delay labels that are dominated by cdp

      Iterator it=holder.tailSet(cdp).iterator();
      it.next();  //skip cdp itself

      TreeSet ts=new TreeSet();
      while (it.hasNext()) {
        Object o=it.next();
        if (cdp.isDominating(o))
          ts.add(o);
        else break;
      }

      it=ts.iterator();
      while (it.hasNext())
        holder.remove(it.next());

      return true;
    }

    public void trim(int k){
      int i=holder.size()-k;
      while (i>0) {
        holder.remove(holder.last());
        i--;
      }
    }
  }

  static class CostDelayLabel implements Comparable{
    int c,d;
    CostDelayLabel predecessor;
    boolean hasScanned;

    public CostDelayLabel(int c,int d,CostDelayLabel predecessor){
      this.c=c;
      this.d=d;
      this.predecessor=predecessor;
      hasScanned=false;
    }

    public int compareTo(Object o) {
      CostDelayLabel p=(CostDelayLabel) o;

      if (c<p.c) return -1;
      if (c==p.c) {
         if (d<p.d) return -1;
         else return 0;
      }
      return 1;
    }

    public boolean isDominating(Object o) {
      CostDelayLabel p=(CostDelayLabel) o;

      if (c<=p.c && d<=p.d) return true;
      return false;
    }
  }
}
```

LHWHM.java

```
import java.util.*;
class LHWHM
{
  public static void main(String args[]){
    Network N=null;
    try {
      N=new Network(args[0]);
    } catch (Exception e){
      System.out.println(e.getMessage());
      System.exit(1);
    }

    int CLABEL[]=new int[N.numNode];
    int DLABEL[]=new int[N.numNode];
    int D[]=new int[N.numNode];
    boolean PERM[]=new boolean[N.numNode];
    int PRED[]=new int[N.numNode];

    //initialization
    compute_D(N,D);

    for (int i=0;i<N.numNode;i++) {
      CLABEL[i]=Network.INFINITY;
      DLABEL[i]=0;
      PERM[i]=false;
      PRED[i]=i;
    }
    CLABEL[N.s]=0;

    for (int i=0;i<N.numNode;i++){
      int u=0;
      int minCLABEL=Network.INFINITY;
      for (int j=0;j<N.numNode;j++)
        if (! PERM[j] && CLABEL[j]<minCLABEL) {
            u=j;
            minCLABEL=CLABEL[j];
         }

      PERM[u]=true;
      if (u==N.t) break;

      for (int j=0;j<N.node[u].edge.length;j++) {
         int edge=N.node[u].edge[j];
         int v;
         if (u==N.edge[edge].vertex[0]) v=N.edge[edge].vertex[1];
         else v=N.edge[edge].vertex[0];

         if (!PERM[v] &&
              CLABEL[u]+N.edge[edge].c<CLABEL[v] &&
              DLABEL[u]+N.edge[edge].d+D[v]<=N.T ) {
               CLABEL[v]=CLABEL[u]+N.edge[edge].c;
               DLABEL[v]=DLABEL[u]+N.edge[edge].d;
               PRED[v]=u;
         }
      }
   }

   System.out.println(CLABEL[N.t]+","+DLABEL[N.t]);
}

static void compute_D(Network N,int D[]){
   //initialization
   boolean PERM[]=new boolean[N.numNode];

   for (int i=0;i<N.numNode;i++) {
      D[i]=Network.INFINITY;
      PERM[i]=false;
   }

   //find a node with the minimum degree and name it as destination
   int minDegree=Network.INFINITY;
   for (int i=0;i<N.numNode;i++)
      if (N.node[i].edge.length<minDegree) {
         minDegree=N.node[i].edge.length;
         N.t=i;
      }
   D[N.t]=0;

   //go
   for (int i=0;i<N.numNode;i++){
      int u=0;
      int minD=Network.INFINITY;
      for (int j=0;j<N.numNode;j++)
         if (! PERM[j] && D[j]<minD) {
            u=j;
            minD=D[j];
         }

      PERM[u]=true;

      for (int j=0;j<N.node[u].edge.length;j++) {
            int edge=N.node[u].edge[j];
            int v;
            if (u==N.edge[edge].vertex[0]) v=N.edge[edge].vertex[1];
            else v=N.edge[edge].vertex[0];

            if (!PERM[v] && D[u]+N.edge[edge].d<D[v])
               D[v]=D[u]+N.edge[edge].d;
         }
      }

      int maxD=0;
      for (int i=0;i<N.numNode;i++)
         if (D[i]>maxD) {
            maxD=D[i];
            N.s=i;
         }
      //System.out.println("maxD="+maxD);
      N.T=maxD*2;
   }
}
```

Network.java

```
import java.util.*;
import java.io.*;

class Network
{
   static final int INFINITY=1000000;

   int s; //source
   int t; //destination

   int T; //bound

   int numNode;
   int numEdge;

   Node node[];
   Edge edge[];

   public Network(String fileName) throws Exception{
      BufferedReader in =
         new BufferedReader(
            new FileReader(fileName));
      String s = new String();
      s = in.readLine();
      StringTokenizer st = new StringTokenizer(s);
      numNode=Integer.parseInt(st.nextToken());
      numEdge=Integer.parseInt(st.nextToken());

      node=new Node[numNode];

      edge=new Edge[numEdge];
      for (int i=0;i<numEdge;i++) {
        s=in.readLine();
        st = new StringTokenizer(s);
        int vertex1=Integer.parseInt(st.nextToken());
        int vertex2=Integer.parseInt(st.nextToken());
        int cost=Integer.parseInt(st.nextToken());
        int delay=Integer.parseInt(st.nextToken());
        edge[i]=new Edge(vertex1,vertex2,cost,delay);
      }

      in.close();

      int degree[]=new int[numNode];

      for (int i=0;i<numNode;i++) degree[i]=0;

      for (int i=0;i<numEdge;i++) {
        degree[edge[i].vertex[0]]++;
        degree[edge[i].vertex[1]]++;
      }

      for (int i=0;i<numNode;i++) {
        node[i]=new Node();
        node[i].edge=new int[degree[i]];
      }

      for (int i=0;i<numEdge;i++)
        for (int j=0;j<=1;j++) {
          int vertex=edge[i].vertex[j];
          node[vertex].edge[--degree[vertex]]=i;
        }
    }

    public static void main(String args[]) throws Exception{
      Network N=new Network(args[0]);
      int D[]=new int[N.numNode];
      LHWHM.compute_D(N,D);
      System.out.print(N.T+","+N.numEdge);
    }
}
```

Node.java

```
class Node
{
    int edge[]; //all edges connected to this node
}
```

RegularGraph.java

```
// Usage: java RegularGraph k n random_seed method max
// k --- connectivity
// n --- the number of nodes
// cost+delay=max
import java.util.*;

class RegularGraph
{
  public static void main(String args[]) throws Exception {
    int k=Integer.parseInt(args[0]);
    int n=Integer.parseInt(args[1]);
    int random_seed=Integer.parseInt(args[2]);
    int method=Integer.parseInt(args[3]);
    int max=Integer.parseInt(args[4]);

    int r=k/2;

    TreeSet edge=new TreeSet();

    for (int i=0;i<n;i++) {
      for (int j=i-r;j<=i+r;j++) {
        if (i==j) continue;
        int jj=j % n;
        if (jj<0) jj+=n;
        edge.add(new Edge(i,jj));
      }
    }

    if (k!=r*2)    //r is odd
      if ((n/2)*2==n)  {     //n is even
        for (int i=1;i<=n/2;i++)
          edge.add(new Edge(i,(i+n/2) % n));
      } else {
        for (int i=0;i<=(n-1)/2;i++)
          edge.add(new Edge(i,(i+(n+1)/2) % n));
      }

    System.out.println(n+" "+edge.size());
    Random ran=new Random(random_seed);

    Iterator it=edge.iterator();
    while (it.hasNext()) {
      Edge e=(Edge) it.next();
      int cost,delay;
      if (method==1) {
        cost=ran.nextInt(max-1)+1;
        delay=ran.nextInt(max-1)+1;
      } else {
        cost=ran.nextInt(max-1)+1;
        delay=max-cost;
      }
      System.out.println(e.v0+" "+e.v1+" "+cost+" "+delay);
    }
  }

  static class Edge implements Comparable{
    int v0,v1;

    public Edge(int v0,int v1){
      if (v0<v1) {
            this.v0=v0;
            this.v1=v1;
         } else {
            this.v0=v1;
            this.v1=v0;
         }
      }

      public int compareTo(Object o) {
         Edge e=(Edge) o;

         if (v0==e.v0 && v1==e.v1)
            return 0;
         if (v0<e.v0 || (v0==e.v0 && v1<e.v1))
            return -1;
         return 1;
      }
   }
}
```

VITA 2

Yuanbing Du

Candidate for the Degree of

Master of Science

Thesis: ON BOUNDED-DELAY MINIMUM-COST PATH PROBLEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Shaoguan, Guangdong, China, January 1, 1974, daughter Of Tianlv Long and Hanning Du.

Education: Graduated from Shaoguan No.1 Middle School, Shaoguan, Guangdong, China, in July 1992; received Bachelor of Economics in International Finance from Jinan University, Guangzhou, China, in June 1996; completed the requirements for the degree of Master of Science in Computer Science at the Computer Science Department of Oklahoma State University in May 2003.

Experience: Employed by Industrial and Commercial Bank of China, China, as an Accountant from July 1996 to August 1999; employed by Midland Realty, China, as a Consultant from August 1999 to May 2000.

Name: Yuanbing Du Date of Degree: May 2003

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: ON BOUNDED-DELAY MINIMUM-COST PATH PROBLEM

Pages in Study: 74 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: Real time applications over the Internet require guarenteed end-to-end quality of service (Qos). The task of QoS-based routing is to find a route in the network which has sufficient resources to satisfy the QoS requirements. One of the QoS-based routing problems – the Bounded-Delay Minimum-Cost Path Problem (BDMCP) is NP-hard. There are two directions to solving NP-hard problems: approximation scheme and heuristic approaches. In this thesis, we focus on heuristic approaches. Several heuristic algorithms which are based on Dijkstra's algorithm or Bellman-Ford-Moore algorithm have been proposed to find suboptimal solutions, but they share the same drawback -- nonscalability. We propose a new heuristic algorithm called K-BFM-BDMCP in which the memory of each node in networks is increased to predefined constant k so that more potentially good intermediate results could be stored to reach a better final solution. We prove that the new algorithm still runs in polynomial time which means it is feasible in practice. An experimental study is conducted to find out the relationship between the value of k and the performance of K-BFM-BDMCP. We also do a comparative study of the new algorithm with BFM-BDMCP algorithm.

Findings and Conclusions: With high probability K-BFM-BDMCP finds better solution than BFM-BDMCP with a tolerable amount of extra time. The main advantage is its scalability and is very suitable for real network enviroments.

ADVISOR'S APPROVAL: [illegible]