ABSTRACT

BARADWAJ, NIKHIL. Traffic Quantization and its Application to QoS Routing. (Under the direction of Professor George N. Rouskas).

In today's high speed networks, the data transmission rate available is usually a multiple of some base rate. An example is that of the T-carrier circuits having transmission rates that are a multiple of the fractional T1 system which is 64 Kbps. The base rate usually depends on the characteristics of the data being transmitted such as audio, video or other types of data. Our objective is to design efficient heuristics to quantize the user traffic demands based on bandwidth into multiples of such base rates such that the excess bandwidth spent due to quantization is minimized. In doing so, our contribution is twofold. First, there is always a penalty that one has to pay to reconfigure the system for each transmission. This reconfiguration time could be due to several factors such as memory operations and table lookups. Our heuristics try to minimize such control/management costs. Secondly, our schemes have the flexibility on the values that the base rate can take which is consequently not hardwired into the system. Our work is greatly influenced by the extensive study in the field of location theory. We achieve our goal by modifying the p-median problem and solving it efficiently using dynamic programming and exploiting the properties of totally monotone matrices. In spite of having an additional constraint of base rate to quantize the traffic, we show that our heuristics have similar bandwidth losses when compared to previous study in the area of traffic quantization. In addition, we save more on control and management costs which are incurred during the reconfiguration of each flow. We also demonstrate through simulations that the blocking probability, defined as the ratio of number of flows blocked due to unavailability of the residual bandwidth in the network to the total number of flows, using our scheme is comparable to that of the continuous network.

Traffic Quantization and its Application to QoS Routing

by

NIKHIL BARADWAJ

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2005

Approved By:

Dr. Steffen Heber	Dr. Khaled Harfoush

Dr. George N. Rouskas
Chair of Advisory Committee

To Mom and Dad ...

Biography

Nikhil Baradwaj was born in New Delhi and brought up in Mangalore, India. After finishing his high school in Chennai, he graduated with a bachelors degree in Engineering (B.E) at Nitte Mahalinga Adhyanta Memorial Institute of Techonogy, Nitte, majoring in Computer Science and Engineering. He then worked as an assistant lecturer at the National Institute of Technology, Mangalore, Karnataka for a year before he came to the US to pursue his Masters degree. At the time of this writing, he is working towards his Master of Science (M.S) in Computer Science at North Carolina State University, Raleigh, US. After completing his graduation, he plans to join Microstrategy Incorporated as a software design engineer.

Acknowledgements

I would like to thank Professor George N. Rouskas, my advisor, for giving me an opportunity to work on an exciting and emerging technology. His many suggestions and constant support during this research has been the inspiration for my work. I shall always admire his patience and calm nature. Never once has he discouraged me in any way or blamed me for any mistakes that I might have made during the process of my research.

I am thankful to Dr. Khaled Harfoush and Dr. Steffan Heber for expressing their interest in my research and agreeing to be members of my committee. I am also thankful to Dr. Laura Jackson for sharing her knowledge through the early days of this research with me. She has been very kind to take some time off from her busy schedule to give her views and results that she found before I took up this research. I would like to acknowledge the National Science Foundation for supporting this research.

I wish to thank the following people who have made my MS experience memorable: Gopal and Erwin for all the help and ideas they offered and the several philosophical discussions we had together. Saila for being a wonderful friend. Arun, Chitty, Dinesh, Kaka, Krish and Sudhakar for all the good times we had together in the last two years.

Finally, I am grateful to my parents for their unconditional *love*. Without their consistent support this research work or my MS education would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Related Work

The traditional p-median problem asks us to find, for a given set of n demand points, a set of p supply points such that the distance between each demand point and its nearest service point is minimized. This problem, which plays a significant role in the study of location theory, has been addressed many times in literature previously and has several applications. [3] explains the variations in such a problem and describes a solution to the p-median problem on the real line. [5] shows that the rectilinear and Euclidean versions of the p-median problem in the plane is NP-hard. The traditional p-median problem and the simple plant location models are also solved by implementing the computational geometry approach of [2].

[4] extends the traditional p-median problem by adding a restriction, requiring the selected supply point for a demand point to be equal to or to the right of it in the real line. They call this the directional p-median problem and show that it can be solved in polynomial time in one dimension. This variant of the p-median problem arises naturally in certain quantization applications where it is important to quantize a set of inputs taking continuous values to a quantization level with equal or higher value. [4] gives examples for such applications.

One such application is in the control and management of packet switched net-

works. A transmission link in a typical backbone network operate on data rates between 2.5 and 10 Gbps and carry thousands of independent traffic flows. Each flow could be characterized by several traffic parameters such as bandwidth, burst size, delay bound etc. Accommodating such flows gives rise to a serious scalability issue for a number of control and management functions. The service provider would like to group (quantize) similar flows into a single service level to simplify to control and management tasks. Each flow would then receive *at least* the requested amount of each resource but not necessarily the exact amount.

Another application is in the preemptive scheduling of periodic tasks on multiprocessor systems. Each task is characterized by its computation time which is the number of unit length sub tasks and its period within which the processing of sub tasks is to be completed. A processor works with at most one task at a time and each task is processed by no more than one processor at a time. When the number of tasks is very large, such as a web server of a popular web site, even the fastest scheduling algorithm may not be appropriate. In such cases, we quantize the set of n tasks into a small, fixed set of offered rates with each task guaranteed to receive at least the required amount of computation time for each period. A quicker algorithm can then be devised which is independent of the number of tasks.

In both the above applications, while quantization has the disadvantage of requiring more resources (processor time, bandwidth) than a continuous rate system, the tradeoff is paid by the resulting gains in speed and simplicity.

1.2 Thesis Organization

In our work, we extend the directional p-median problem by adding an additional constraint. In Chapter 2, we formally define the directional p-median problem and give a solution in $O(np)$ time. In Chapter 3, we introduce an additional constraint to the directional p-median problem where we require each of the supply points chosen to be a multiple of some basic unit r. We extend the solution presented in Chapter 2 to solve this problem and also give heuristic approaches to improve on the time complexity at the cost of accuracy. We present several numerical results and discussions on the various heuristics in Chapter 4. Chapter 5 is a simulation study which gives an insight on the blocking

probabilities when the different algorithms are used, followed by the summary of our work and the direction of our work that can be taken in future in Chapter 6.

Chapter 2

p-Median Problem and Its Solution

This chapter looks into the problem of quantization and the approach to solve it by mapping the problem to an instance of the directional p-median problem on the number line. This idea which has been presented by [4] can then be solved using dynamic programming to get efficient complexity bounds. We first present the concept of location problem specifically defining the traditional p-median problem. We then look at a special case of the problem, the directional p-median problem and how it solves the quantization problem. Presenting a straightforward dynamic programming algorithm, we give an improvement to the algorithm by introducing the concept of totally monotone arrays and present the implementation details.

2.1 The Location Model on the Number Line

In the general facility location model, given n points in [0,1], identified as a set of **demand** points, the objective is to select a subset of these points as the location to setup facilities, called the **service** or **supply** points, that will serve the set of demand points. The **utility** of a demand point is defined as a function of the distance between it and the closest facility that services it. The objective of such models is to minimize the overall utility of the system by choosing appropriate locations for setting up the facilities. This model extends to several well known classical location problems like the p-median problem,

p-coverage problem etc., whose detailed description can be found in literature.

2.1.1 The Traditional p-median Problem

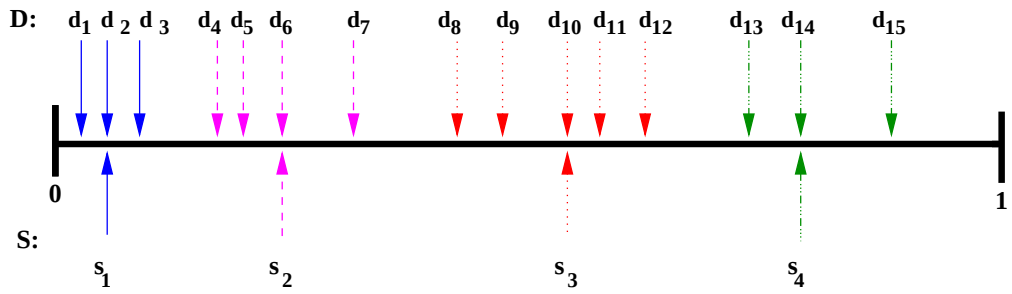

Figure 2.1: Mapping a set of demand points D to a set of service points S in traditional p-median problem

The traditional p-median problem can be defined as follows: Given a set of n demands points $D = \{d_1, d_2, \ldots, d_N\}$, find the set of p service points $S = \{s_1, s_2, \ldots, s_P\}$ such that the total distance between each demand point and its nearest service point is minimum. Formally, the p-median problem can be formulated as follows:

$$Minimize \sum_{i=1}^{n} \bar{\omega}(d_i, S)$$

subject to

$$d_1 \leq d_2 \leq \ldots \leq d_N$$

$$d_1, d_2, \ldots, d_N \in [0, 1]$$

$$\bar{\omega}(d_i, S) = \min\{\mid d - d_i \mid : d \in S\}$$

$$S \subseteq D$$

$$|S| = p$$

$$d_i \in D$$

$$1 \leq p \leq n$$

where $\bar{\omega}(d_i, S)$ is distance between d_i and its nearest supply point

2.1.2 The Directional *p*-median Problem

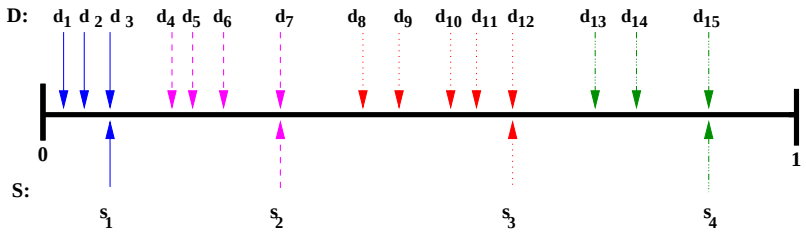

Figure 2.2: Mapping a set of of demand points D to a set of service points S in directional p-median problem

Here we look at a slightly modified version of the p-median problem where the nearest service point for a given demand point has to be greater than or equal to the demand point under consideration. In other words, the problem is directional. Our new objective function can then be formulated as

$$Obj(S) = \sum_{j=1}^{p} \sum_{d_i \in D_j} (s_j - d_i)$$

subject to

$$d_1, d_2, \ldots, d_N \in [0, 1]$$

$$S \subseteq D$$

$$|S| = p$$

$$d_1 \leq d_2 \leq \ldots \leq d_N$$

$$s_1 \leq s_2 \leq \ldots \leq s_P$$

$$s_{j-1} < d_i \leq s_j$$

$$D_j \in D$$

As an example for an application of such a problem, if a network user requests a certain amount of bandwidth, the network provider should provide at least the requested bandwidth. Hence when all the user bandwidths, which correspond to the demand points, are quantized to a set of service bandwidths, they have to be mapped to a service bandwidth that is greater than or equal to itself. The directional p-median problem fits appropriately for our problem of quantizing user demand bandwidths. In our problem of quantizing the requested demand points, the above defined $Obj(S)$ is the excess resources that is to be spent in order to service the demand points.

2.2 The $O(n^2p)$ Dynamic Programming Algorithm

The quantization problem has an input of the set $D = \{d_1, d_2, \ldots, d_N\}$ of demand points that are sorted in non-decreasing order of their traffic demands d_i. In order to describe the dynamic programming algorithm to find the optimal set S of service points, we

define the cost function $\Psi(R_n, p)$ as the minimum cost for mapping all points in set R_n to p service points where R_n is the set of requests from D with the n smallest traffic demands.

$$Obj(S) = \Psi(R_N, p) - \sum_{i=1}^{n} d_i$$

In practical terms, the cost function $\Psi(R_N, p)$ is the total bandwidth spent in the network after quantization where as the objective function $Obj(S)$ is the excess bandwidth spent in the network due to quantization. $R_N = D$ and $R_n = \bigcup_{i=1}^{n}\{d_i\}$,$n = 1, 2, \ldots, N$. We compute $\Psi(R, P)$ recursively using

$$\Psi(R_n, 1) = nd_n, n = 1, \ldots, N$$

$$\Psi(R_1, p) = d_1, p = 1, \ldots, P$$

$$\Psi(R_n, p+1) = \min_{q=l,\ldots,n-1}\{\Psi(R_q, p) + (n-q)d_n\},\ p = 1, \ldots, P-1; n = 2, \ldots, N$$

As can be seen from the above equations, when $p = 1$, all the requested points are mapped to the last demand point. The cost of mapping only one point to p service points is the demand point itself as there are no other points. The recursive equation can be explained by noting that service level $p+1$ must be the demand point d_n and the remaining p optimal service levels must be assigned values from d_i. If the p^{th} service level is d_q where q is between p and $n-1$, the optimal cost is given by the right hand side of the recursive equation by taking the minimum over all the possible values of q. It can be seen that the above algorithm has an overall time complexity of $O(n^2 p)$.

2.3 A Better Solution - The DPM1 Algorithm

In this section we present an algorithm with a better complexity bound when compared to the previous section. We exploit the property of totally monotone matrices and Monge property in our problem to formulate a better dynamic programming algorithm and present its implementation details.

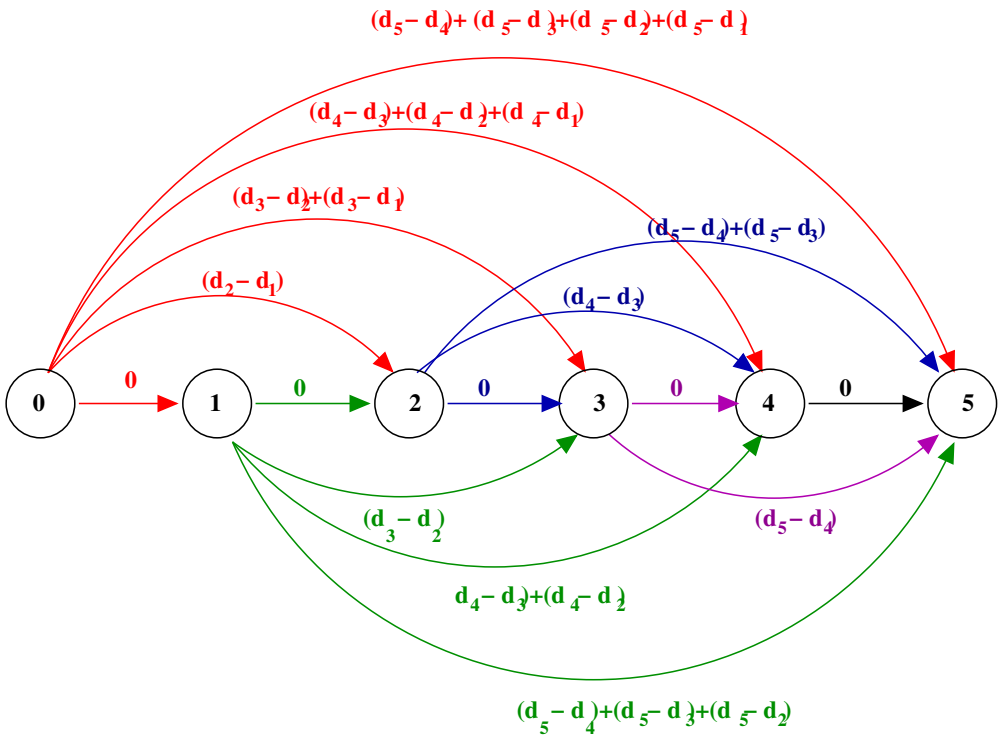

Figure 2.3: DAG representation of an instance of directional p-median problem with $n = 5$

2.3.1 Monge Condition and Totally Monotone Matrices

A matrix **M** having real entries is said to be **monotone** if $i_1 > i_2$ implies that $j(i_1) \geq j(i_2)$ where $j(i)$ is the index of the leftmost column containing the maximum value in row i of **M**. **M** is said to be **totally monotone** if its sub matrices are monotone [6].

The above directional p-median problem can be represented as a directed acyclic graph as shown in Figure 2.3, where each arc weight $\omega(i,k)$ is the sum of the distances between the demand points $d_{i+1}, d_{i+2}, \ldots, d_k$ and the demand point d_k.

Hence

$$\omega(i,k) = \begin{cases} 0 & \text{if } k = i+1 \\ (k-i-1)d_k - \sum_{j=i+1}^{k-1} d_j & \text{otherwise} \end{cases} \tag{2.1}$$

A complete and weighted DAG satisfies the concave Monge condition if

$$\omega(i,j) + \omega(i+1,j+1) \leq \omega(i,j+1) + \omega(i+1,j)$$

for all $0 < i+1 < j < n$. It has been shown that the graph representing the directional p-median problem obeys the concave Monge condition [4]. [1] showed that a two dimensional Monge array is totally monotone on their notes on searching in multidimensional monotone arrays.

2.3.2 Minimum Elements in Each Column of Totally Monotone Matrices

[6] exploits the property of totally monotone matrices to find the maximum entry in each row of such matrices in $\Theta(n)$ time. This elegant matrix searching algorithm has many geometric applications. We describe the algorithm to find the minimum entry in each column of the matrix and show that this $\Theta(n)$ algorithm can be applied to our quantization problem.

The algorithm to find the minimum in each column of a totally monotone $n \times m$ matrix, $n \geq m$, contains two parts:

The **Reduce** procedure takes an $n \times m$ totally monotone matrix as an input, $m \leq n$, and reduces the matrix to an $m \times m$ matrix by eliminating all the rows, also called the dead rows, that are assured not to contain the minimum element in any of the columns.

The **MinCompute** procedure is a recursive procedure that calls the Reduce procedure to

Algorithm Reduce (A)

C = A; k=1

While C has more than m rows do

case

C(k,k)<=C(k+1,k) & k<n: k=k+1

C(k,k)<=C(k+1,k) & k=n: Delete row C^{k+1}

C(k,k) > C(k+1,k): Delete row C^{k}

if k>1:k=k+1

end case

Algorithm MinCompute (A)

B=Reduce (A)

if n=1 then output the minimum and return

C = B[1,2,...,n;2,4,...,2*floor(n/2)]

Mincompute (C)

From the known positions of the minima in the even rows of B, find the minimum in the odd rows

find all the minimum elements in all the columns of the matrix.

When $n \geq m$, the MinCompute procedure solves the minimum problem in a totally monotone $n \times m$ matrix in time $O(n)$ and this has been proved by [6]. The above algorithm will be used to solve the quantization problem but will not store the totally monotone array in memory. Instead, a function will be used to generate each element of the matrix in constant time, the details of which will be presented in the next section. The deletion of rows in the Reduce algorithm will be implemented by using a linked list that points to each row of the matrix. When a row is deleted, the corresponding node is deleted from the linked list to indicate the deletion. The linked list implementation also helps in easily traversing to the next or the previous undeleted row in the array.

2.3.3 The New Dynamic Programming Formulation To Exploit Monge Condition

In order to take advantage of the Monge condition, we need to change the dynamic programming algorithm presented in Section 2.2. We introduce a new term $\varpi(i, j)$ into the recursive equation, which is defined as the cost of mapping demand points $d_{i+1}, d_{i+2}, \ldots, d_k$ to point d_k. In other words,

$$\varpi(i,k) = \begin{cases} 0 & \text{if } k = i+1 \\ (k-i-1)d_k - \sum_{j=i+1}^{k-1} d_j & \text{otherwise} \end{cases} \tag{2.2}$$

As seen in the previous section, this term follows the Monge condition. By incorporating this new term, our dynamic programming algorithm can be defined as

$$F(i,j) = \begin{cases} 0 & \text{if } i = j \\ \varpi(j,i) & \text{if } j = 1 \\ \min_{k=j-1}^{i-1}\{F(k,j-1) + \varpi(k+1,i)\} & \text{if } j < i, j \neq 1 \end{cases} \tag{2.3}$$

Note that the recursive function $F(i, j)$ defined here calculates the objective function unlike the cost function $\Psi(R_n, p)$ that was defined for the dynamic programming algorithm in the previous section.

Figure 2.4 shows the table that is filled using the Equation (2.3) using dynamic programming one column at a time from left to right. The term $F(N, P)$ gives us the

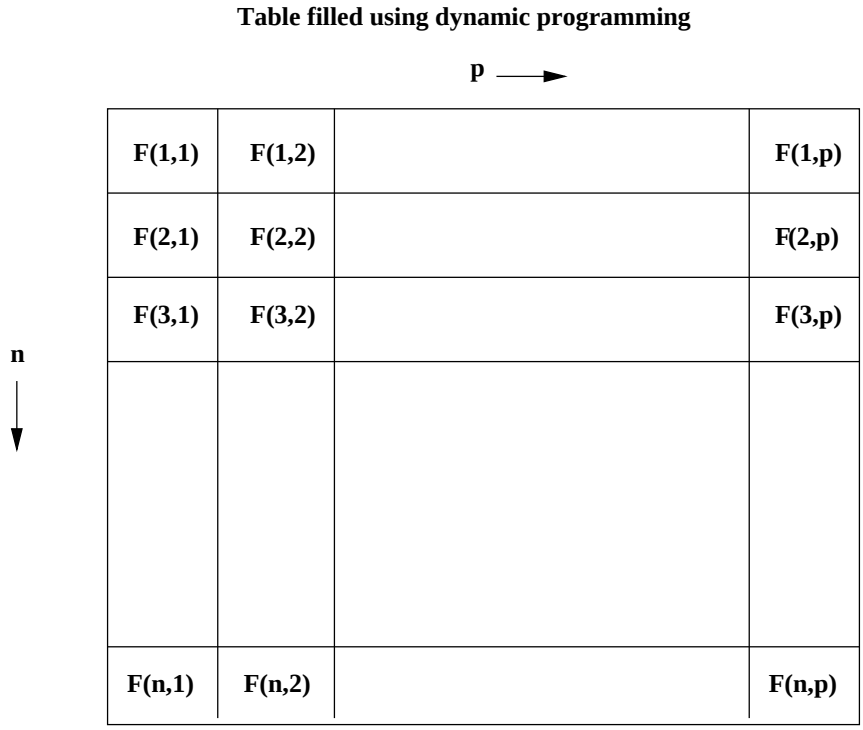

Figure 2.4: Table filled using dynamic programming

solution to the quantization problem. Let $F(k)$ represent column k of Figure 2.4. Our objective is to fill each column of the Figure 2.4 in $O(n)$. In order to achieve this, we introduce a new function

$$\Gamma(i,j) = F(i,k-1) + \varpi(i+1,j) \tag{2.4}$$

We can fill column k of Figure 2.4 by finding the minimum elements in each column of the $n \times n$ matrix represented by function $\Gamma(i,j)$. The above equation is similar to the set of equations that are represented by [7] to solve the concave least-weight subsequence problem. We see from the above equation that in order to calculate the elements of $F(k)$, $\Gamma(i,j)$ depends on the values of $F(k-1)$ that has already been calculated.

We now show that the function $\Gamma(i,j)$ obeys the concave Monge condition.
We know that

$$\varpi(i+1,j) + \varpi(i+2,j+1) \leq \varpi(i+1,j+1) + \varpi(i+2,j) \tag{2.5}$$

If we add $F(i,k-1) + F(i+1,k-1)$ to both sides of Equation (2.5) and apply Equation (2.4), we get

$$\Gamma(i,j) + \Gamma(i+1,j+1) \leq \Gamma(i,j+1) + \Gamma(i+1,j) \tag{2.6}$$

Equation (2.6) shows that $\Gamma(i,j)$ exhibits the concave Monge condition and hence the matrix represented by $\Gamma(i,j)$ is totally monotone. In solving the above dynamic programming algorithm, to fill each column $F(k)$ in the table shown in Figure 2.4, we form an $n \times n$ matrix containing $\Gamma(i,j)$ values that depend on the values of $F(i,k-1)$ using Equation (2.4). Finding minimum elements in each column of this matrix gives us the n values needed to fill $F(k)$. This can be solved in $O(n)$ using the Reduce and MinCompute algorithms presented in the previous section. The time to fill all the p columns would then take $O(np)$.

The only issue now is that the array of $\Gamma(i,j)$ is not actually represented in memory. Hence the value of $\Gamma(i,j)$, given the value of i and j has to be evaluated in constant time. $\Gamma(i,j)$ depends on the value of $\varpi(i+1,j)$ and $F(i,k-1)$ each of which needs to be evaluated in constant time. In order to evaluate $\varpi(i+1,j)$ in constant time, we perform a simple trick initially even before starting to solve the dynamic programming algorithm. Define, for

$j = 1, \ldots, N$

$$AV(j) = \sum_{t=1}^{j} d_t$$

Defining $AV(j)$ for each $d_j \in$D would take time $O(n)$. Once this preprocessing is done, one can find the value of $\varpi(i, j)$ in constant time using the formula

$$\varpi(j, k) = -AV(k) + AV(j-1) + (k - j + 1)d_k$$

for $j \leq k$.

The value of $F(i, k-1)$ is already computed and hence can be referred to, in constant time. Hence the value of $\Gamma(i, k)$ can be calculated in constant time for any two values i and j where $i \leq n$ and $j \leq n$. Having determined how to evaluate $\Gamma(i, k)$ in constant time given i and j and using the algorithm to determine the smallest element in each column in totally monotone matrices, we can solve the dynamic programming algorithm in $O(np)$ which is an improvement over the algorithm presented in Section 2.2.

Chapter 3

Quantization Involving Additional Constraints

In this chapter, we look at an extension to the directional p-median problem by adding an additional constraint to the existing problem. We find the service points for the given set of demand points such that all of the service points are a multiple of some basic unit r. We explain the purpose of describing such a problem and extend the algorithm given in the previous chapter to solve it.

3.1 The Additional Constraint

In certain applications, it is necessary to have the solution as a multiple of some basic parameter. A good example is that of the memory system. A page is the smallest set of data moved between main memory and the disk. When bringing the block of memory from main memory to cache, the size of the cache block is a multiple of a page unit.

In today's high speed networks, the data transmission rate available is usually a multiple of a base rate r. For example the T-carrier circuits have transmission rates that are a multiple of the fractional T1 system which is 64 Kbps. Optical transmission systems such as the SONET lines have bandwidths that are a multiple of the basic SONET line rate

(OC-1) of 51.84 Mb/s. The base rate usually depends on the characteristics of the data being transmitted such as audio, video or other types of data. Hence when we quantize the user requested demands, our objective is to make sure that they are a multiple of some base unit so that the residual bandwidth after allocating the requested bandwidth is still a multiple of that unit. This would prevent links from having residual bandwidths that cannot be provided to any user's request. Additionally, our scheme should provide the flexibility on the values that the base rate can take which is consequently not hardwired into the system. Hence we extend the new problem of quantization by adding an additional constraint to the directional p-median problem.

Given a set of demand points $D = \{d_1, d_2, \ldots, d_N\}$, find a set of p service points $S = \{s_1, s_2, \ldots, s_P\}$ such that the total distance between each demand point and its nearest service point is minimum and in addition, each service point should be a multiple of some basic unit r. In other words, $s_i = k_i \times r, \forall s_i \in S$.

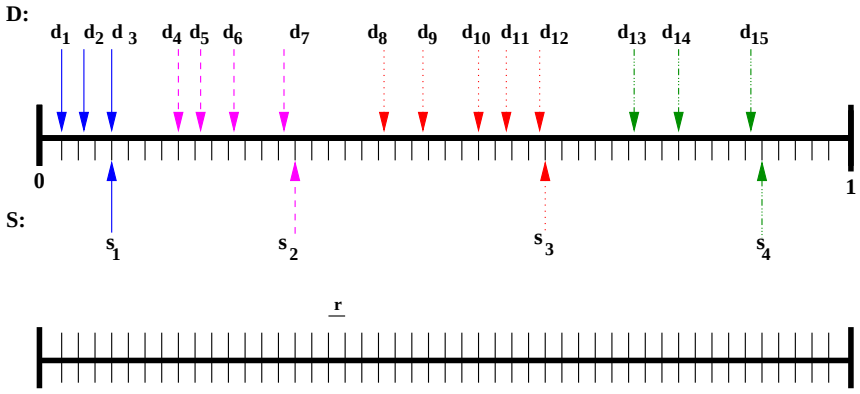

Figure 3.1: The directional p-median problem with service points being a multiple of a basic unit r

Considering the above problem, our new objective function $Objective(S)$ to be minimized will then be

$$Objective(S) = \sum_{j=1}^{p} \sum_{d_i \in D_j} (r \times k_j - d_i) \tag{3.1}$$

where D_j is the set of demand points mapped to the service point s_j and r is the basic unit.

Let the cost function $Cost(S)$ be defined as the total bandwidth that is provided for all the user demands in the system. Then $Cost(S)$ for the above problem is

$$Cost(S) = \sum_{j=1}^{p} (r \times k_j \times n_j) \tag{3.2}$$

where n_j is the number of demand points mapped to the service point s_j. In other words,

$$Cost(S) = Objective(S) + \sum_{i=1}^{n} (d_i) \tag{3.3}$$

where $d_i \in D$. It is intuitive from the above equation that the smaller the basic unit r, the smaller and better is the cost function $Cost(S)$.

There is always a penalty that one has to pay to reconfigure the system for each transmission. This reconfiguration time could be due to several factors such as memory operations and table lookups. Let us call this extra penalty, the control/management cost.

Let us define a penalty term $\Phi(r)$ as the rate that is lost in reconfiguration given that the basic unit of flow is r bits per second. In other words, it represents the rate of flow that could be accepted into the system if there was no time spent for reconfiguration. Let T be the transmission speed of the system. To consider the worst case penalty, let us assume that the system contains only flows with the basic rate. Hence, the actual time it takes to transmit r bits would be r/T seconds. Let C_1 seconds be the configuration time after each transmission. So a time of $C_1 + r/T$ is spent on each flow. If we consider a time frame of one second, the number of transmissions of size r bits in one second would be

$$\frac{T}{T \times C_1 + r}$$

This is also the number of times reconfiguration takes place. Hence, the number of bits that could have been transmitted during all the time that spent on reconfiguration is

$$\Phi(r) = \frac{T}{T \times C_1 + r} \times C_1 \times T$$

As we can see from the above equation, the penalty term that should be minimized is inversely proportional to the basic unit r.

$$\Phi(r) = \frac{c}{\alpha + r}$$

for some constant c and α where

$$c = C_1 \times T^2$$

and

$$\alpha = C_1 \times T$$

This penalty term should also be considered when we try to find the objective function and the minimum cost function expressed in Equations (3.1) and (3.2) respectively. Hence we define a new objective function $Obj(S)$ as follows:

$$Obj(S) = \frac{c}{\alpha + r} + Objective(S) \tag{3.4}$$

where $\frac{c}{\alpha+r}$ is the control/management cost for reconfiguration and $Objective(S)$ defined in Equation (3.1) is the excessive bandwidth used. We can infer from the above equation that the value of the basic unit r can neither be very small nor very large. Making the basic unit r very small would increase the penalty function thereby increasing the objective function. Making the basic unit very large would increase the $Objective(S)$. Hence our goal is to find the optimum r for which the value of the objective function $Obj(S)$ is minimum.

3.2 The Behavior Of The Objective Function

Figure 3.2 is a plot of a sample objective function for $p = 10$, $n = 1000$, $c = 0.01$ and the n demand points generated as a uniform distribution between 0 and 1. The value of the basic unit r is varied from 0 to $1/p$ in increments of 10^{-5} (the details about the range and the granularity of the basic unit are presented in the next section). Figure 3.3 is a plot of a sample objective function for $p = 10$, $n = 1000$, $c = 0.005$ and the n demand points generated as a decreasing distribution (refer the next chapter for the behavior of these distributions) between 0 and 1. We observe from these figures that there is an initial period of exponential decrease in the objective function as the value of r increases. This is where the control/management cost factor dominates the objective function due to the very

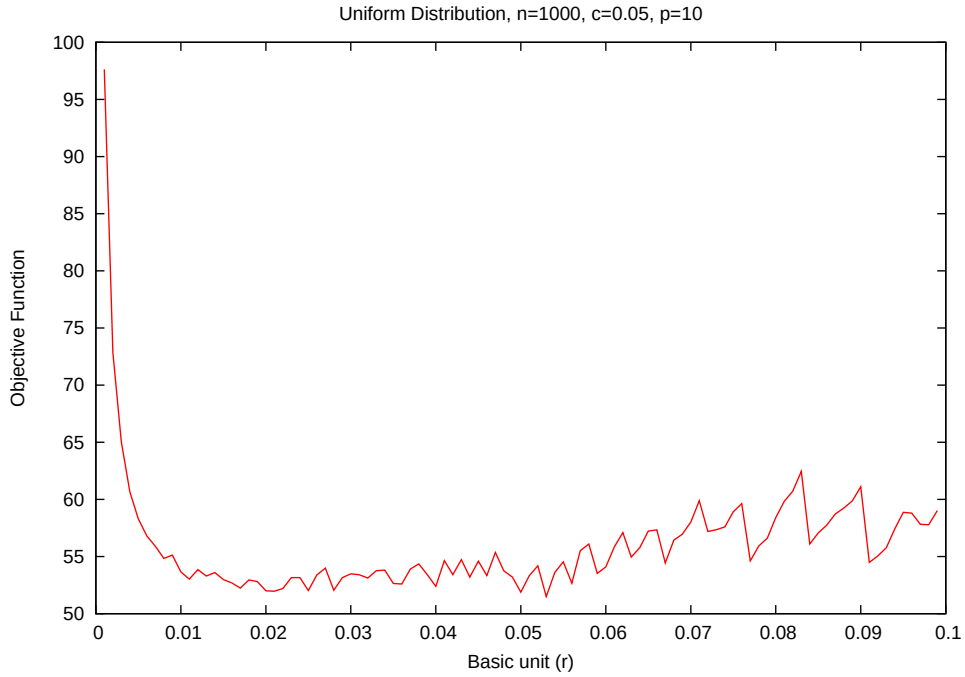

Figure 3.2: Objective function for demand points following a Uniform distribution

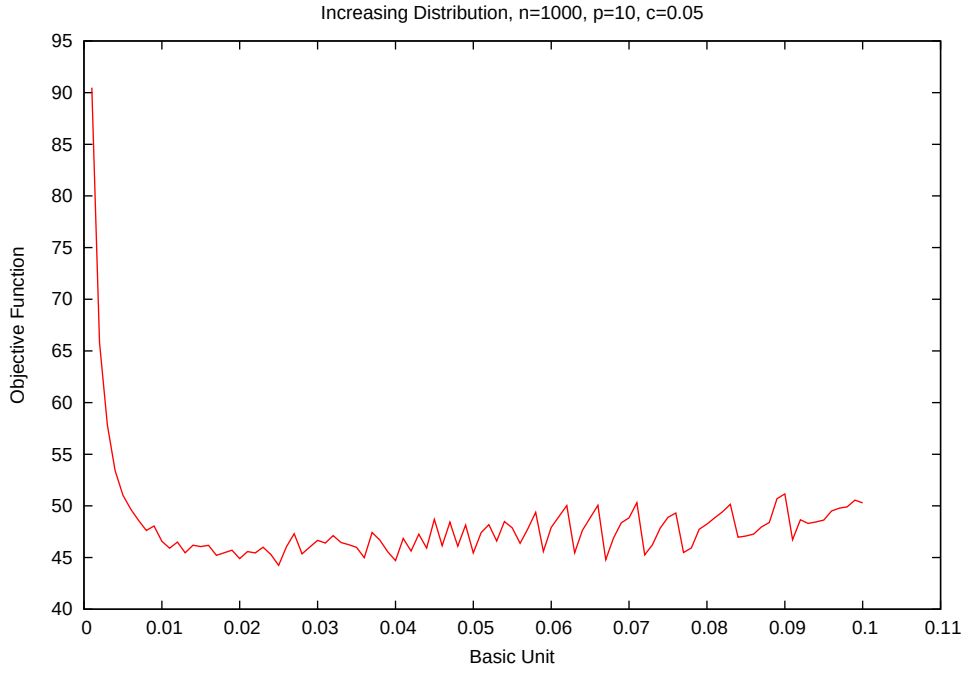

Figure 3.3: Objective function for demand points following a Decreasing distribution

small values of r. As the value of r increases considerably, both the factors of the objective function defined in Equation (3.4) come into play.

Also evident from the graphs is that the objective function is non-convex containing several trenches at irregular intervals. The non-convex nature makes the optimization of the above objective function difficult as it is very easy to get trapped in a local minima. Since there is no easy way out to find the appropriate r whose objective function is minimum using the standard optimization techniques known, we develop an exhaustive search on all the possible values of r to find the optimum r.

3.3 The Exhaustive Search For The Optimum r

To solve the above mentioned problem, we will modify the algorithm presented in the previous chapter used for solving the directional p-median problem. The idea is that for each value r_i in the range of values that the basic unit r can take, apply the algorithm to find the p service points for the given n demand points such that all the p service points are a multiple of r_i. Calculate the objective function $Obj_i(S)$ for each value of r_i and find the optimum r_i that has the minimum objective function.

Since the p service points to be selected are a multiple of r, they no longer need to be one of the demand points. In the case of the directional p-median problem, the set S of p service points selected were a subset of the demand point set D. In other words, $S \subseteq D$. This is no longer true with the addition of the new constraint which means that any of the p service points could be out of the set of the given demand point set D.

We define the granularity of r as the difference between r_i and r_{i-1}. We require the granularity of r in order to find out the values r_i of r for which the objective function has to be calculated. The granularity of the function is important as it determines the accuracy with which we find the minimum objective function among all values of r. As the granularity increases, the accuracy also increases. Considering today's high speed network capacities of around 2.5 Gbps and the fact that each flow provided is a multiple of 64 Kbps, we can estimate the granularity of the basic unit r as $64Kbps/2.5Gbps \approx 0.00001$ when all the demand points are normalized to values between 0 and 1. For a given value of p, we know that the maximum value r can take is $1/p$ which makes it the upper bound of r. Hence we linearly find the objective function for all values $r_i \in r$ such that $0 \leqslant r_i \leqslant 1/p$

with increments of 0.00001 (in the normalized scale).

Having known the range of values that r can take and the granularity of r, for each value of $r_i \in r$, we find the set of potential service points by taking multiples of r_i between 0 and $1/p$. We find the objective function using the DPM1 algorithm presented in the previous chapter on this set. We do this for all possible values of r_i to find the optimum objective function, i.e the objective function with the lowest value. This is then the desired solution. Hence this algorithm resorts to an extensive and tedious linear search on all the possible values of r_i to find the an r_{opt} that has the optimum objective function. The service points obtained by using this value r_{opt} would then be the best set of service points for the given set of demand points.

3.4 The Optimization Heuristics

As we infer from the previous section, there is a lot to pay with respect to time when we conduct an exhaustive search to find the r_{opt} which gives the optimum objective function. Due to the granularity of r, there could be up to 10^5 possible values in the worst case (when $p = 1$), and for each of these possible values of r we need to run the DPM1 algorithm which has a time complexity of $O(np)$, where n here would be all the multiples of r between 0 and $1/p$ which could again have up to 10^5 possible values. Hence the time complexity for the whole search would be $const \times p$ where $const$ is a very large value. The exhaustive search algorithm hence has a very large time complexity that is independent on the value of n. Hence even if the value of n is very small, it costs a lot to conduct such a search to find the best set of service points. An improvement in running time can be made by trading off the accuracy with which the minimum objective function is found. To satisfy this goal, we present a few optimization heuristics in this section.

3.4.1 Demand Driven Heuristic (DDH)

In the exhaustive search algorithm, we considered all the multiples of r_i between 0 and $1/p$ as the set of potential service points. In this heuristic, we find the multiples of r_i that are closest to and greater than the n given demand points. These new n points

would then be the potential set of service points thereby considerably decreasing the time complexity of the algorithm.

In order to satisfy this, we define a set $V = \{v_1, v_2, \ldots, v_n\}$ of points such that $v_i = r \times \lceil d_i/r \rceil$ for each $d_i \in D$. Calculating this set V given the demand set D has time complexity $O(n)$. We change the definition of $\varpi(i,k)$ in Equation (2.2) as the sum of the distance between demand points $d_{i+1}, d_{i+2}, \ldots, d_k$ and point v_k (instead of d_k). Hence

$$\varpi(i,k) = \begin{cases} 0 & \text{if } k = i+1 \\ (k-i-1)v_k - \sum_{j=i+1}^{k-1} d_j & \text{otherwise} \end{cases} \tag{3.5}$$

In order to use the above equation in the DPM1 algorithm we need to prove that this new function $\varpi(i,k)$ obeys the concave Monge condition. In other words, we have to prove that

$$\varpi(i,k) + \varpi(i+1,k+1) \leq \varpi(i+1,k) + \varpi(i,k+1)$$

We prove this as follows:

$$\begin{aligned} L.H.S &= \varpi(i,k) + \varpi(i+1,k+1) \\ &= (k-i-1)v_k - \textstyle\sum_{j=i+1}^{k-1} d_j + (k-i-1)v_{k+1} - \sum_{j=i+2}^{k} d_j \\ R.H.S &= \varpi(i+1,k) + \varpi(i,k+1) \\ &= (k-i-2)v_k - \textstyle\sum_{j=i+2}^{k-1} d_j + (k-i)v_{k+1} - \sum_{j=i+1}^{k} d_j \\ R.H.S - L.H.S &= v_{k+1} - v_k + d_k - d_k = v_{k+1} - v_k \geq 0 \end{aligned}$$

Hence $\varpi(i,k) + \varpi(i+1,k+1) \leq \varpi(i+1,k) + \varpi(i,k+1)$ thereby obeying the concave Monge condition. We can now use the DPM1 algorithm presented in the previous chapter to find the p service points and the corresponding objective functions for each value of r_i. We then continue this process on all the possible values that r_i can take between 0 and $1/p$ until we get an r_{opt} that gives the optimum objective function.

For each value $r_i \in r$, finding the objective function and the p service points has a time complexity of $O(np)$. Depending on the value of p, there could be around 10^5 values of r in the worst case (when $p = 1$). Let the number of values of r be denoted by x. Then the total time complexity would be $O(nxp)$.

3.4.2 Service Driven Heuristics

The DDH explained in the previous section has a time complexity that depends on the number of values of r and both n and p. For large values of n, the DDH heuristic could turn out to be a slow algorithm. In order to save further on time, we present a few other optimization heuristics that have faster time complexity at the expense of approximate results with a higher value of objective function.

The idea behind the heuristic approach here is is driven by the fact that the p service points to be found might be close to the service points that are found using the DPM1 algorithm in Chapter 2. Hence, we first find the p service points for the given set of demand points without considering the r factor. Hence this problem reduces to a simple directional p-median problem that can be solved in $O(np)$. Once the p service points $S = \{s_1, s_2, \ldots, s_P\}$ are found using the algorithm presented in the previous chapter, for each value of the basic unit r, we find two sets

$$S' = \{s'_1, s'_2, \ldots, s'_P\}$$

such that $s'_i = \lceil s_i/r \rceil \times r$ and

$$\vec{S} = \{\vec{s_1}, \vec{s_2}, \ldots, \vec{s_P}\}$$

such that $\vec{s_i} = \lfloor s_i/r \rfloor \times r$. The $2p$ elements of the above two sets would collectively become the potential candidates for being one of the p service points. We follow a dynamic programming approach to find the best p service points. Let us define $\eta(i, j)$ as the cost of mapping all the demand points between i and j to j. We then define function F recursively as follows:

$$F(1, 0) = \eta(d_1, s'_1)$$

$$F(1, 1) = \eta(d_1, \vec{s_1})$$

$$F(r, 0) = \min\{F(r-1, 0) + \eta(s'_{r-1}, s'_r), F(r-1, 1) + \eta(\vec{s_{r-1}}, s'_r)\}$$

$$F(r, 1) = \min\{F(r-1, 0) + \eta(s'_{r-1}, \vec{s_r}), F(r-1, 1) + \eta(\vec{s_{r-1}}, \vec{s_r})\}$$

for all $1 < r \leq p$. Solving the above recursive equations using the dynamic programming algorithm, we can calculate the objective function as $F(p, 1) + \frac{c}{\alpha + r}$ as well as the p best

service points. We name the above described approach, the **Bidirectional Service Driven Heuristic (BSDH)**.

Another approach is to just consider only those points that are a multiple of r and greater than the p service points found using the DPM1 algorithm. To achieve this, we compute just one set

$$S' = \{s'_1, s'_2, \ldots, s'_P\}$$

such that $s'_i = \lceil s_i/r \rceil \times r$. This set would then be the required service points. It is then a straightforward task to compute the objective function. We name this the **Unidirectional Service Driven Heuristic (USDH)**.

Both of the above heuristic approaches are faster than the DDH scheme by a factor of n which is a significant improvement at a cost of having a slightly higher objective function.

3.4.3 The Power of Two Heuristic (PTH)

Here we derive another scheme where the service points do not depend on the values of the demand points. Hence the service points only depend on the value of p, the number of quantization levels required. Hence irrespective of the values of the demand points or the number of demand points, we always select the same set of service points. We select the service points to be negative powers of 2 from 0 to $p-1$. Hence the set S of service points would be

$$S = \{\tfrac{1}{2^{p-1}}, \tfrac{1}{2^{p-2}}, \tfrac{1}{2^{p-3}}, \ldots, 1\}$$

We see that obtaining these service points would take only $O(p)$ and calculating the value of the objective function then would take $O(n)$. Although this is a very fast heuristic when compared to the other heuristics presented prior to this, the value of the objective function and therefore the excess amount of bandwidth spent for the system would be considerably higher when compared to the other heuristics. We choose the service points to be powers of two as this scheme could then be applied in other systems where the power of 2 factor plays an important role.

Chapter 4

Numerical Results

We now present numerical results comparing the performance of several algorithms and heuristics discussed in the previous chapter for traffic quantization. The schemes we shall consider are:

1. The DPM1 algorithm, explained in Chapter 2 that quantizes traffic using directional p-median problem in one dimension presented in [4].

2. The DDH heuristic for quantizing traffic based on the multiples of base unit lying close to the demand points

3. The USDH heuristic that quantizes traffic by considering only points that are a multiple of the base unit, close to and ahead of the p supply points found using DPM1.

4. The BSDH heuristic that quantizes traffic by considering only points that are a multiple of the base unit and close to the p supply points found using DPM1 in both directions.

5. The PTH heuristic which has fixed service points and are negative powers of 2 between 0 and $p-1$.

Six different distributions were used to generate the demand sets in the simulations: uniform, increasing, decreasing, triangle, unimodal and bimodal. The mathematical expressions for the probability density functions (pdf) and cumulative density functions

(cdf) of each of these distributions are given in Table 4.1. Figures 4.1 - 4.6 show the graph of the pdf of these distributions.

Distribution	pdf	cdf	Domain
Uniform	1	x	$0 < x < 1$
Increasing	$2x$	x^2	$0 < x < 1$
Decreasing	$-2x + 2$	$-x^2 + 2x$	$0 < x < 1$
Triangle	$4x$	$2x^2$	$0 < x < 0.5$
	$-4x + 4$	$-2x^2 + 4x - 1$	$0.5 \leq x < 1$
Unimodal	4/9	$4x/9$	$0 < x < 0.25$
	6	$6x - 25/18$	$0.25 \leq x < 0.35$
	4/9	$4x/9 + 5/9$	$0.35 \leq x < 1$
Bimodal	1/4	$x/4$	$0 < x < 0.25$
	4	$4x - 15/16$	$0.25 \leq x < 0.35$
	1/4	$x/4 + 3/8$	$0.35 \leq x < 0.65$
	1	$4x - 33/16$	$0.65 \leq x < 0.75$
	1/4	$x/4 + 3/4$	$0.75 \leq x < 1$

Table 4.1: Formulae for pdf and cdf input distribution

4.1 Relative Performance of the Algorithms

Figures 4.7 - 4.10 show graphs describing the nature of the objective function for DDH, BSDH and USDH heuristics drawing the demand points from various distributions. The nature of the graphs presented in the figures is non-convex with very irregular trenches as described in the previous chapter. Initially, when the value of r is very small, the objective function is very high because of Equation (3.4). This explains the need for taking the cost/management costs into consideration when selecting the appropriate r for traffic quantization. As the value of r increases, there is an exponential decrease in the objective function for some time after which it gets irregular.

The graphs show that DDH has the lowest objective curve in most cases, followed closely by BSDH. The USDH gives a higher objective function by a small percentage but is compensated by a much faster running time. Hence we infer that there is a trade off between the value of the objective function and the running time complexity of the algorithms.

Figure 4.11 shows the performance for all the algorithms. We observe that the PTH heuristic performs the worst as it has a predetermined set of service points which does

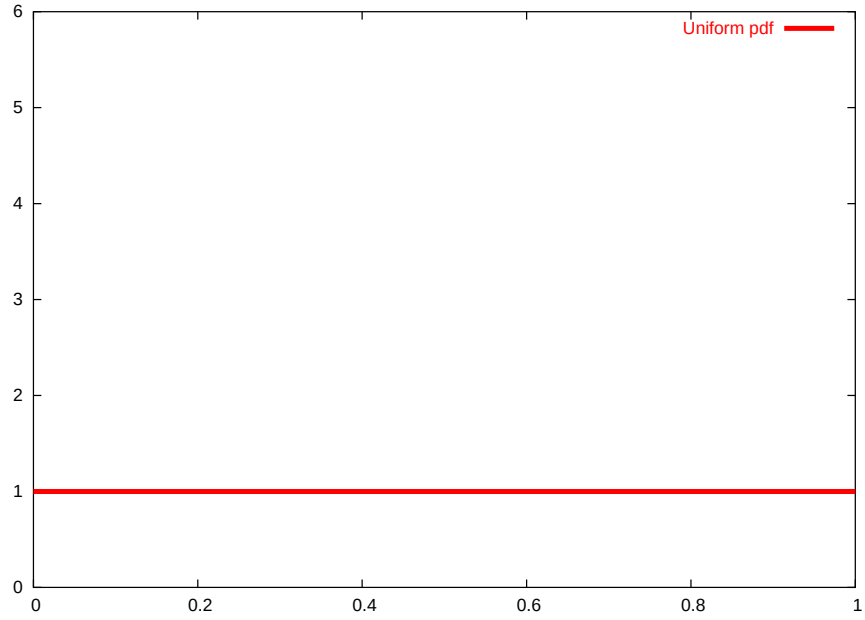

Figure 4.1: pdf of Uniform distribution

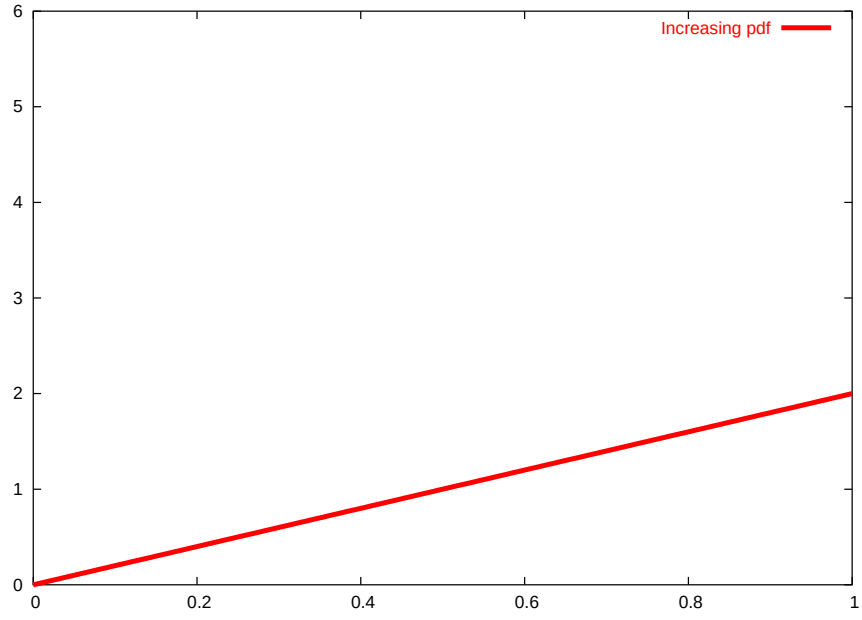

Figure 4.2: pdf of Increasing distribution

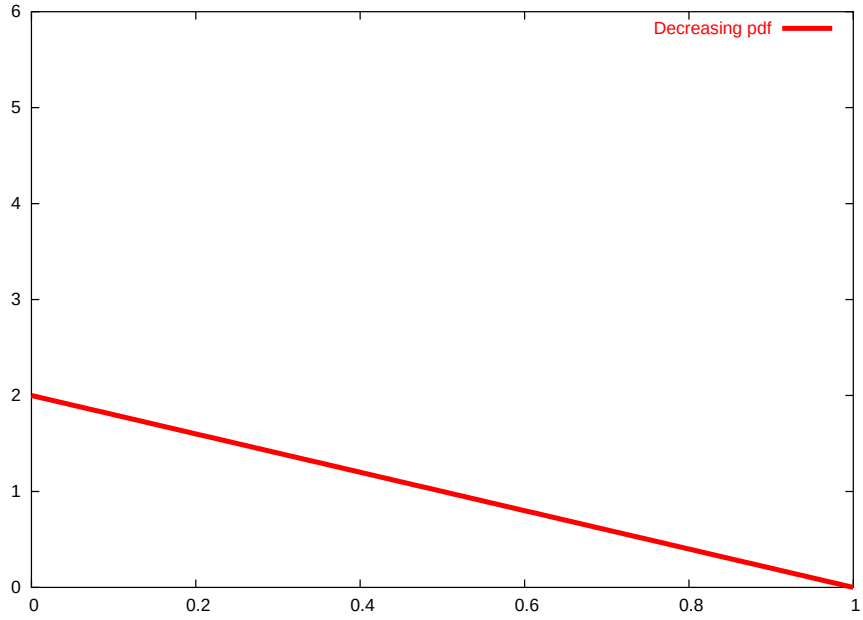

Figure 4.3: pdf of Decreasing distribution

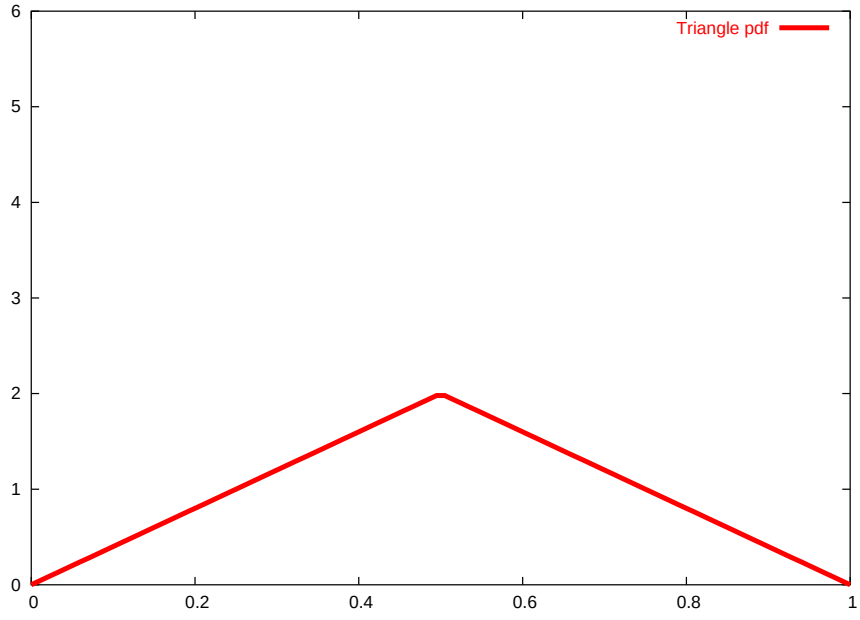

Figure 4.4: pdf of Triangle distribution

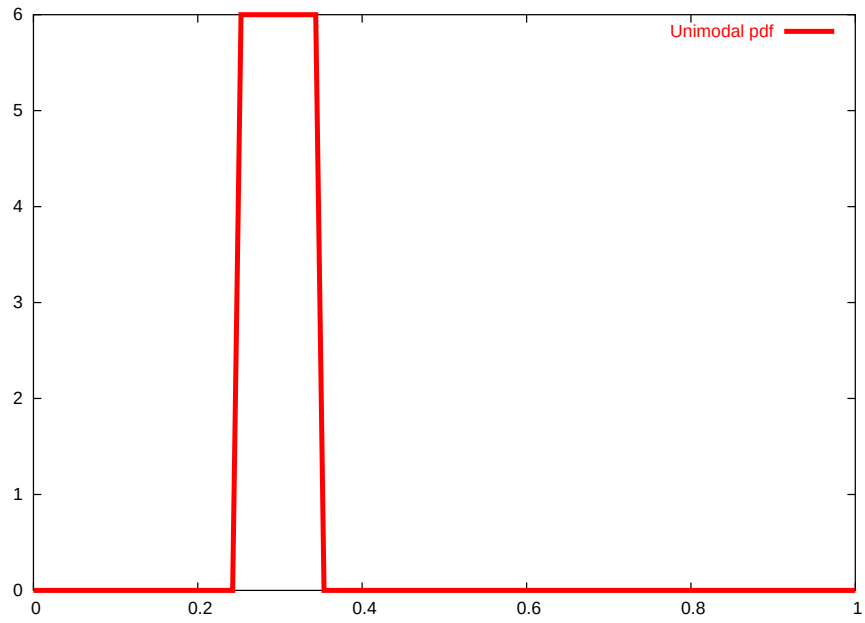

Figure 4.5: pdf of Unimodal distribution

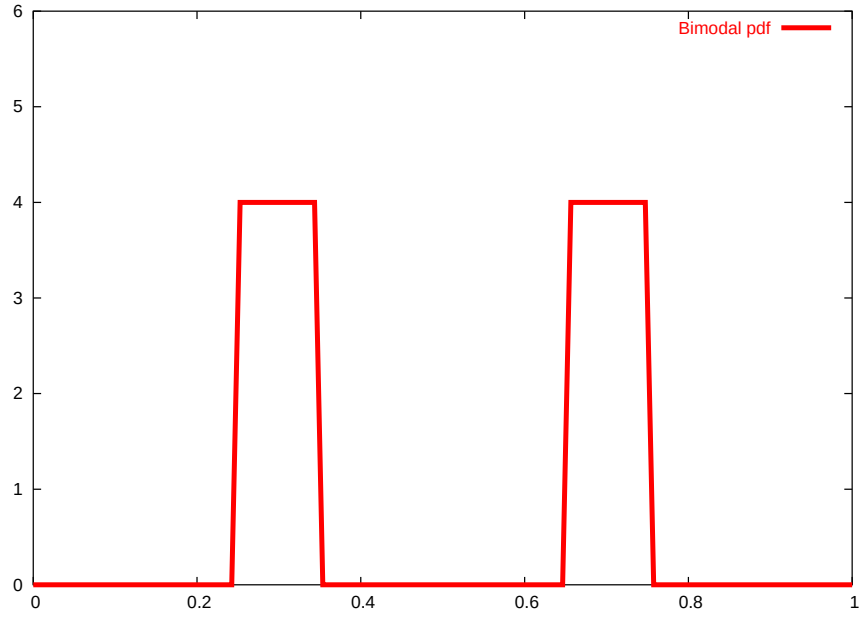

Figure 4.6: pdf of Bimodal distribution

not depend on the demand points. The DPM1 algorithm has the lowest curve but there is no control/management cost accounted for.

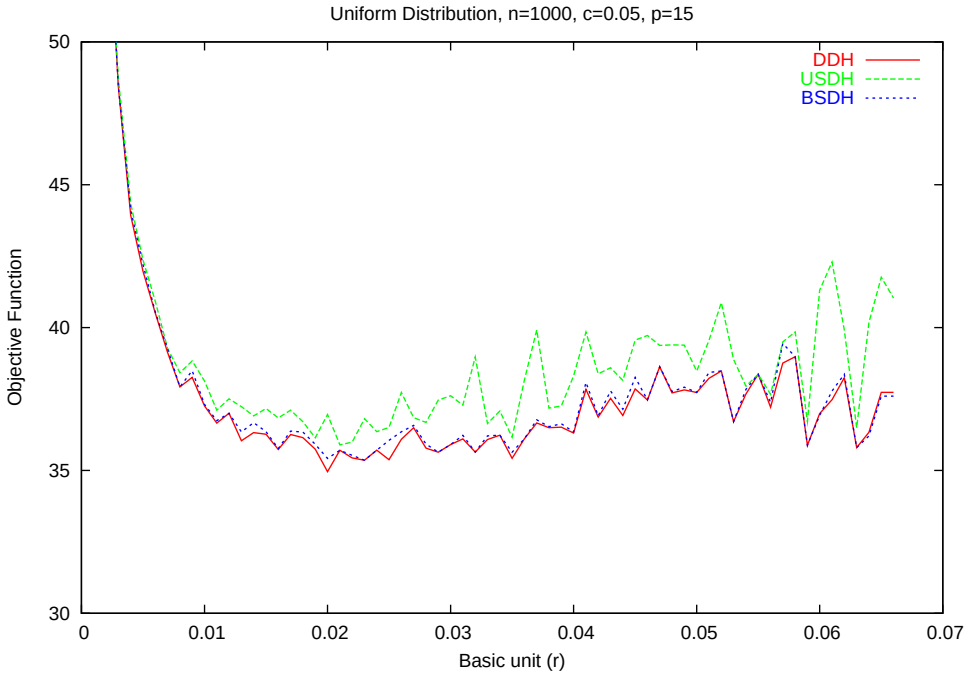

Figure 4.7: Relative Performance of DDH, BSDH and USDH for a uniform distribution

Figures 4.12 and 4.13 show the nature of the objective function for the DDH heuristic when the values of p and c are changed respectively. We observe from the graphs that as p increases, the value of the objective function decreases. This is obvious because as p increases, excess bandwidth spent due to quantization decreases. As the value of p tends to n, i.e a continuous network, the objective function tends to zero. Also shown in the graph is that as c increases, the objective function increases which is in accordance to Equation (3.4).

Figures 4.15 - 4.20 are a plot of the optimum objective function with respect to the simulation instance number. Each instance of the simulation has a new set of demand points generated from the respective distribution. For each instance of the simulation, the heuristics are run for all values of the basic unit r to determine the r where the objective function hits the global minimum. The value of the objective function is termed the optimum objective function for that instance of the simulation. All graphs are a plot of 30

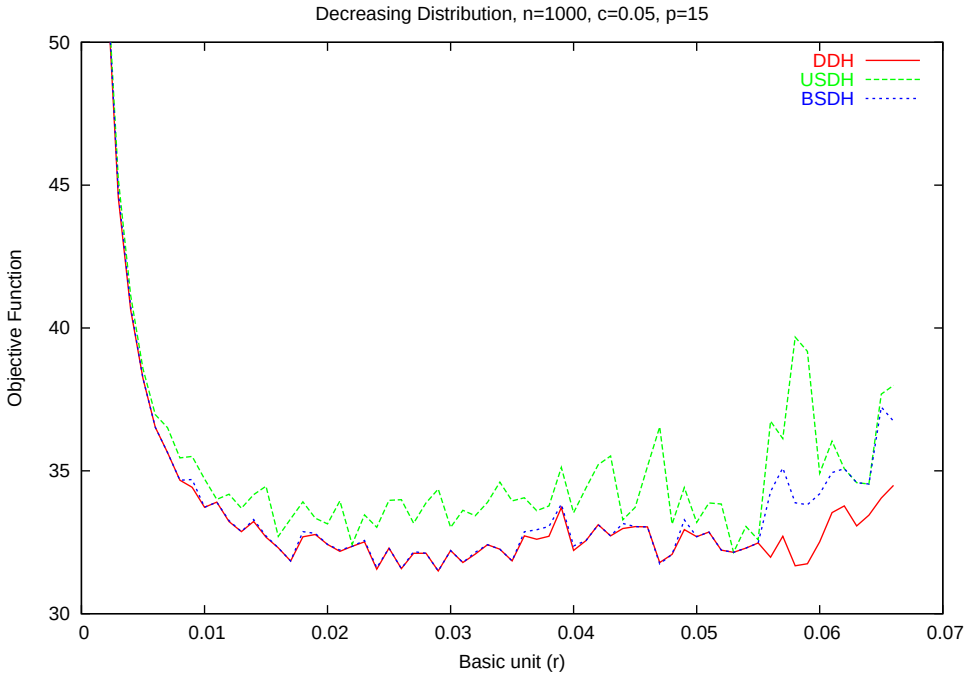

Figure 4.8: Relative Performance of DDH, BSDH and USDH for a decreasing distribution

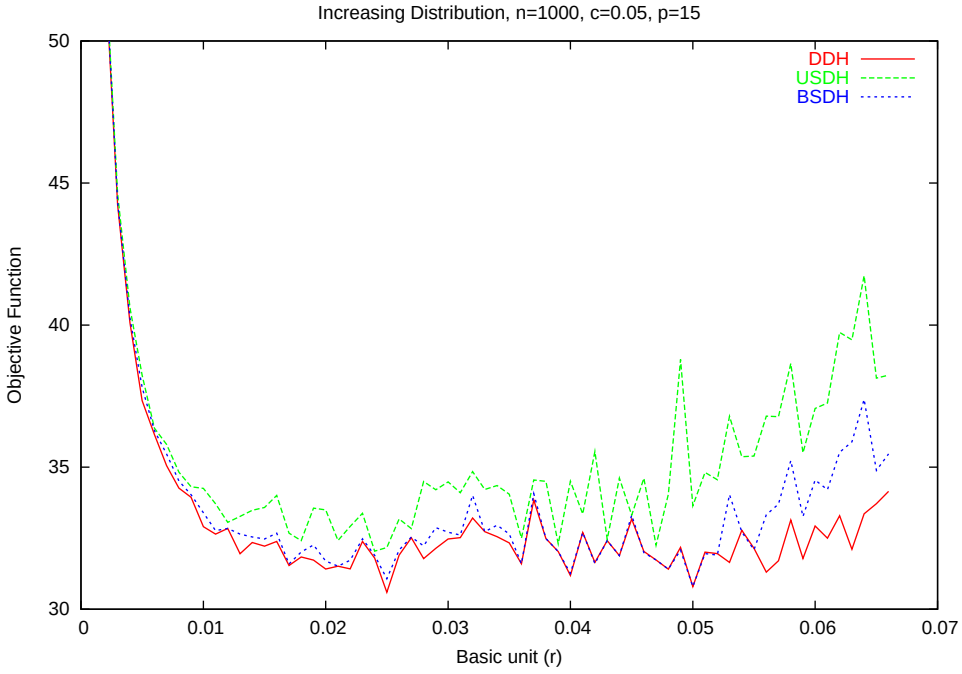

Figure 4.9: Relative Performance of DDH, BSDH and USDH for a increasing distribution

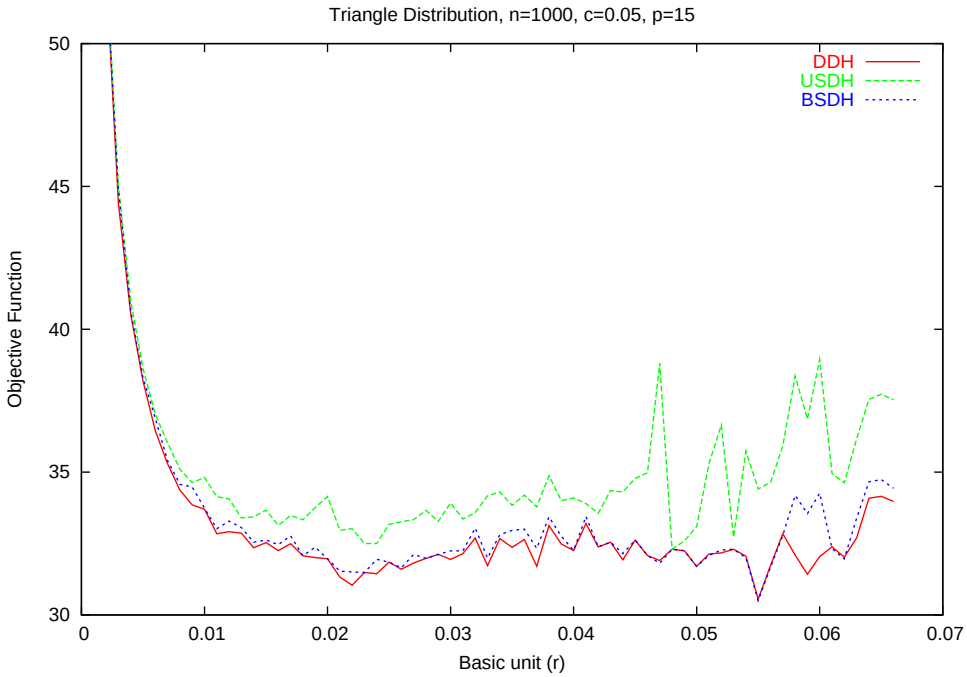

Figure 4.10: Relative Performance of DDH, BSDH and USDH for a triangle distribution

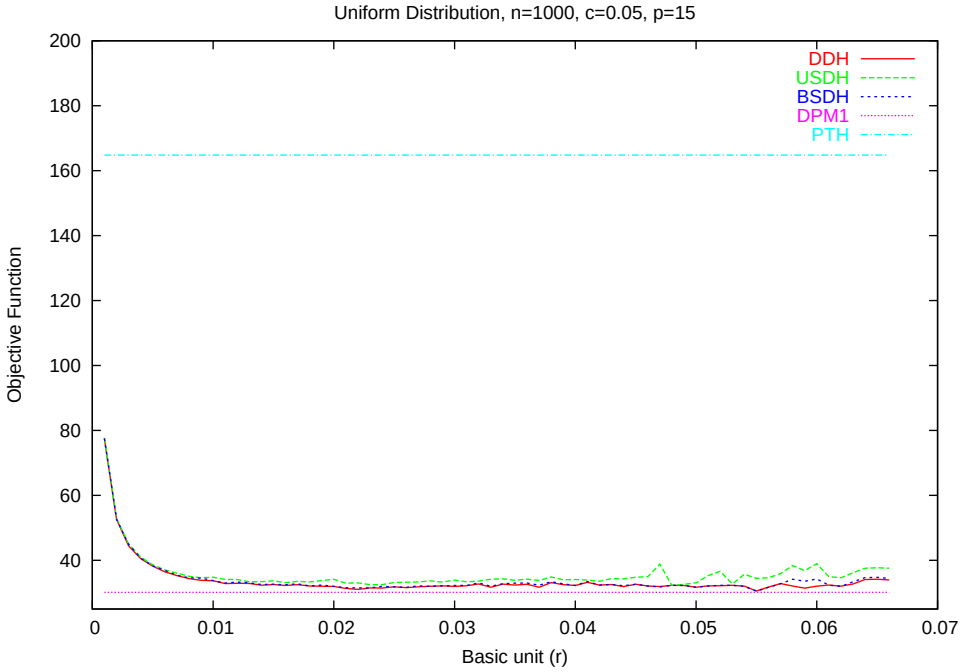

Figure 4.11: Relative Performance of DDH, BSDH, USDH, PTH and DPM1 for a uniform distribution

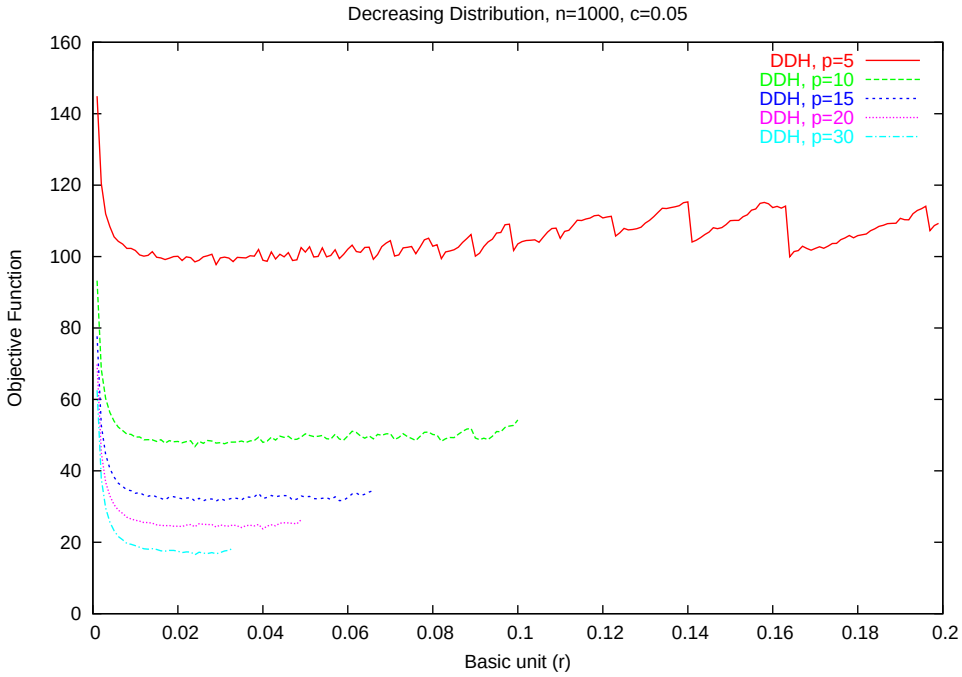

Figure 4.12: Nature of the Objective function by varying p

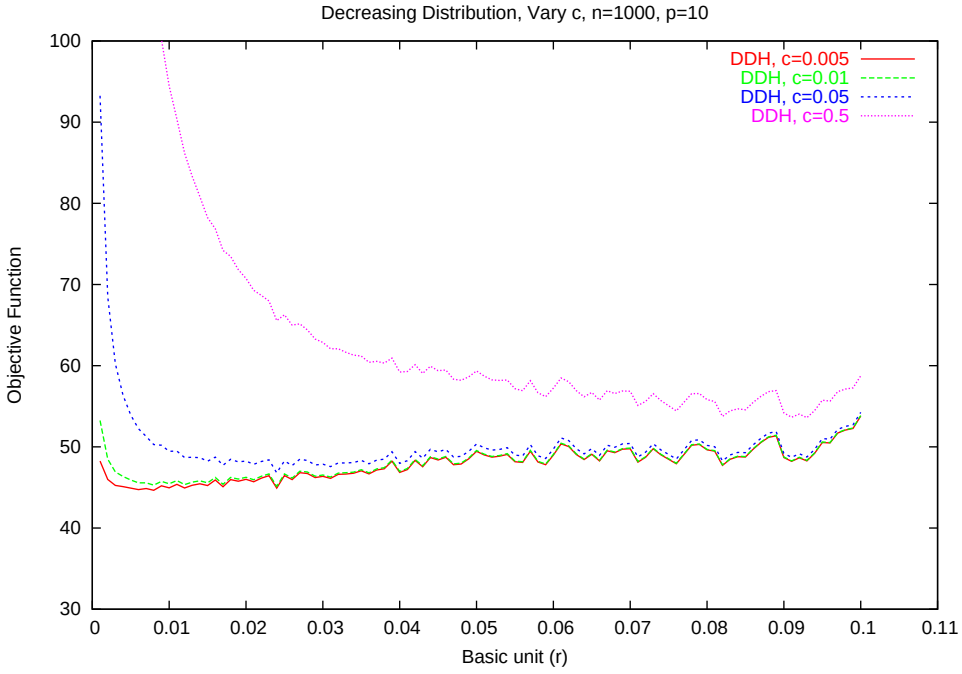

Figure 4.13: Nature of the Objective function by varying c

instances, each having demand points generated from a separate seed.

We infer from the graphs that DDH gives the best results most of the time, but at the cost of a slower running complexity. The BSDH performs almost as good as the DDH with some instances sometimes performing better.

4.2 Cost Comparisons after Normalization

The results in this section show the graphs after normalization of the cost function with respect to the continuous system. Cost function is defined as the total bandwidth spent for the system. Hence it can be calculated as the sum of the objective function and the sum of all demand points. For the continuous system, the cost function is the same as the sum of all demand point. In order to normalize the cost function of DDH, we do the following: We calculate the optimum objective value for a given instance of n demand points. We then remove the control/management cost factor from this term as this does not contribute to the bandwidth utilization of the network. We add this to the sum of all demand points to get the cost of the DDH $cost_{DDH}$ for that instance of demand points. We then divide this by the cost function of the continuous system $cost_{continuous}$ for the particular instance of demand points. In our simulation, we do this for 30 instances, each with demand points generated from a different seed and find the mean normalized cost function $norm_{DDH}$. In other words,

$$norm_{DDH} = \frac{\sum_{j=1}^{30} \frac{cost_{DDH_j}}{cost_{continuous_j}}}{30}$$

where

$$cost_{continuous} = \sum_{i=1}^{n} (d_i)$$

and $cost_{DDH}$ is as mentioned in Equation (3.3).

Similar calculations are done to obtain the normalized cost function of DPM1. All graphs are a plot of such normalized cost against the value of p. For each value of p, the normalized cost function is the average of 30 instances to obtain statistically significant results.

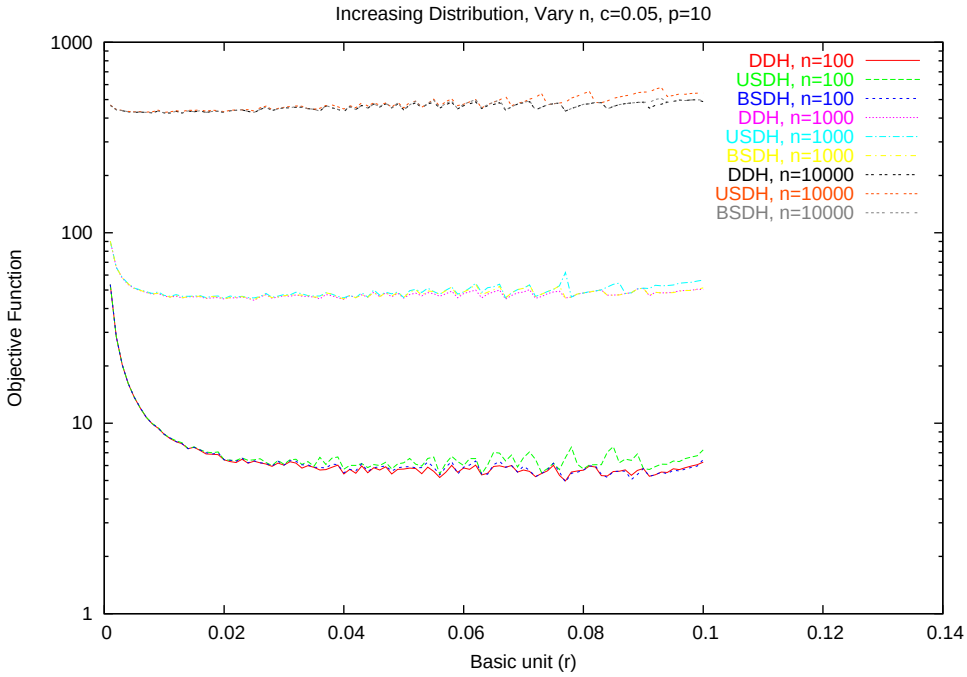

Figure 4.14: Nature of the Objective function by varying n

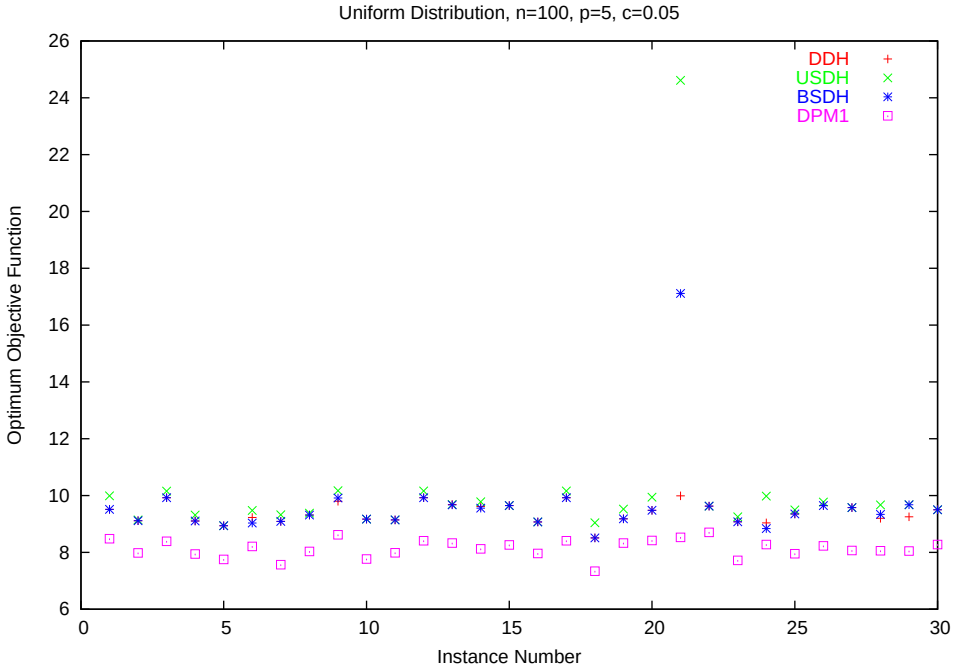

Figure 4.15: Optimum Objective function plot for Uniform distribution

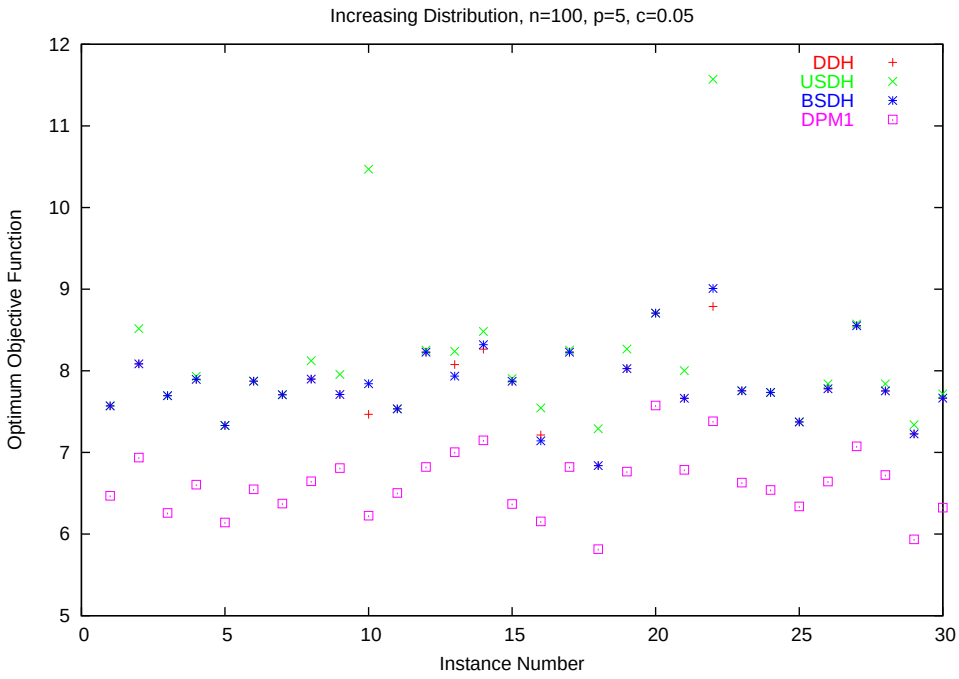

Figure 4.16: Optimum Objective function plot for Increasing distribution

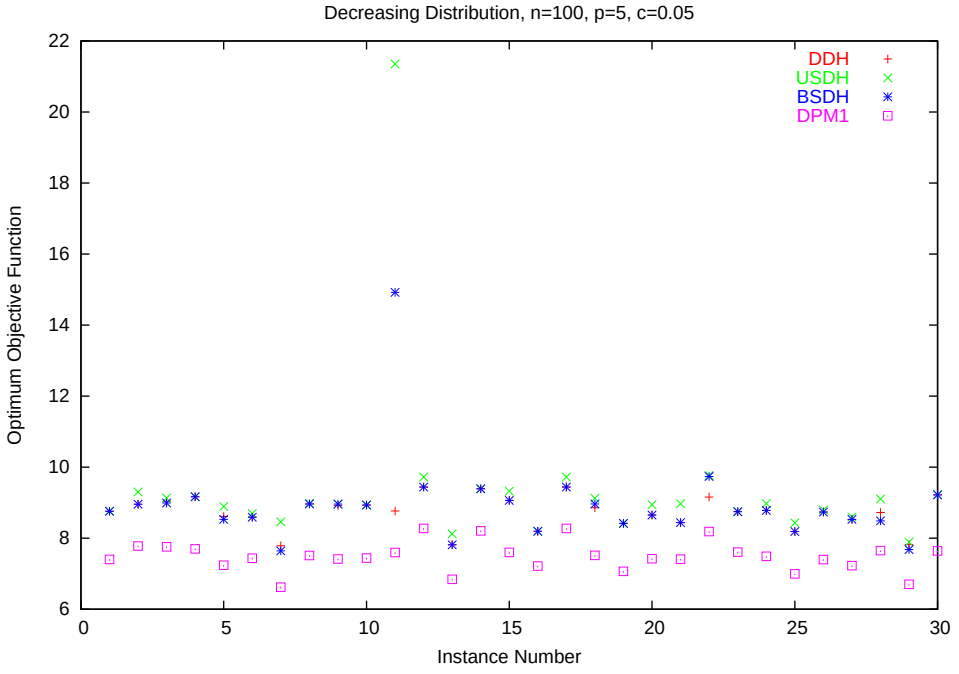

Figure 4.17: Optimum Objective function plot for Decreasing distribution

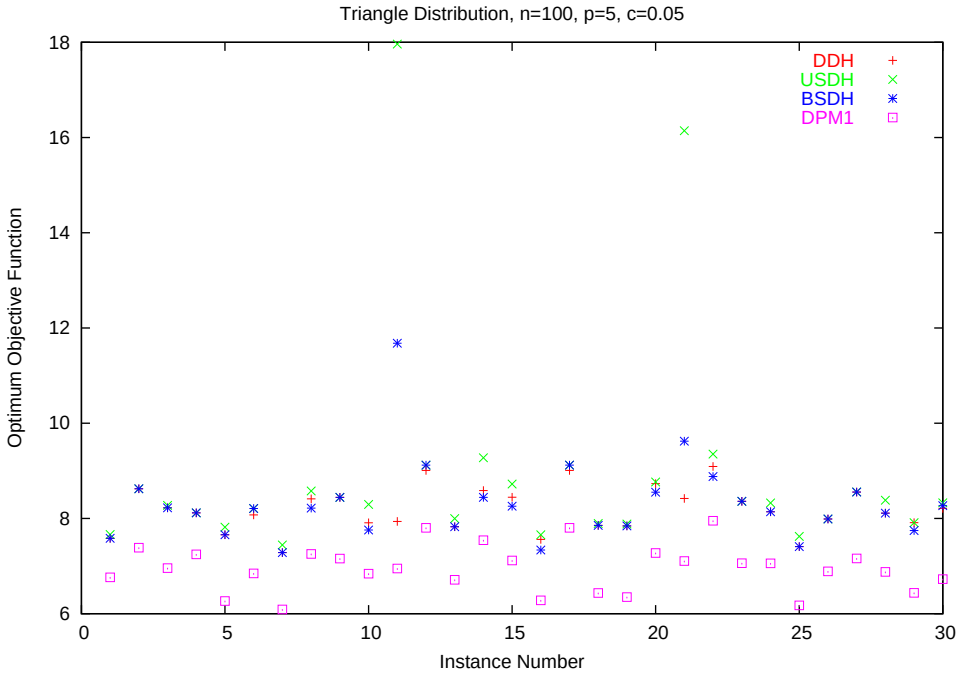

Figure 4.18: Optimum Objective function plot for Triangle distribution

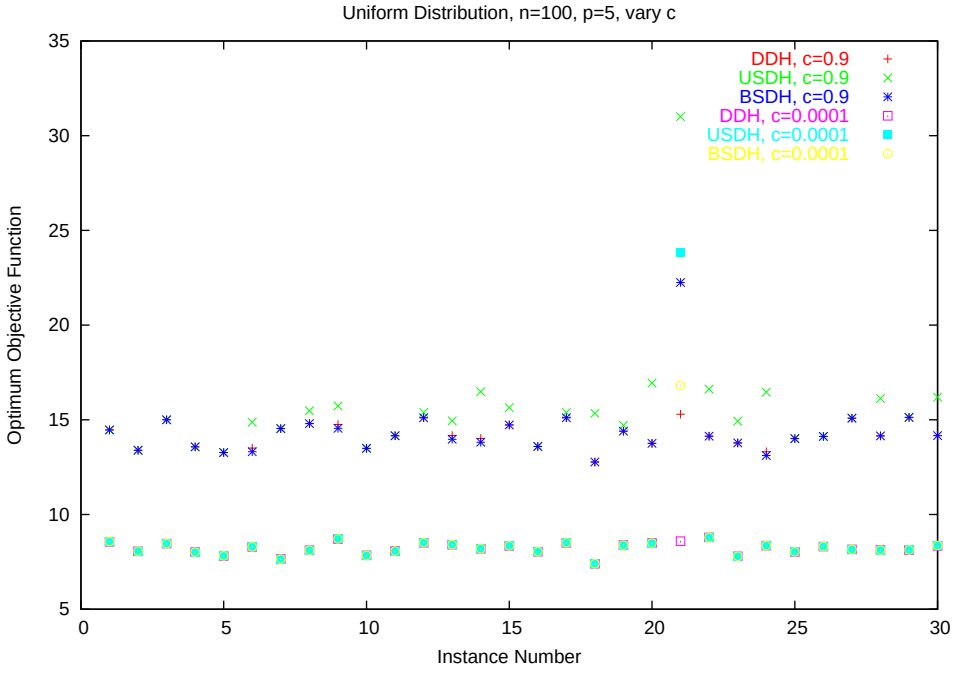

Figure 4.19: Optimum Objective function plot for Uniform distribution varying c

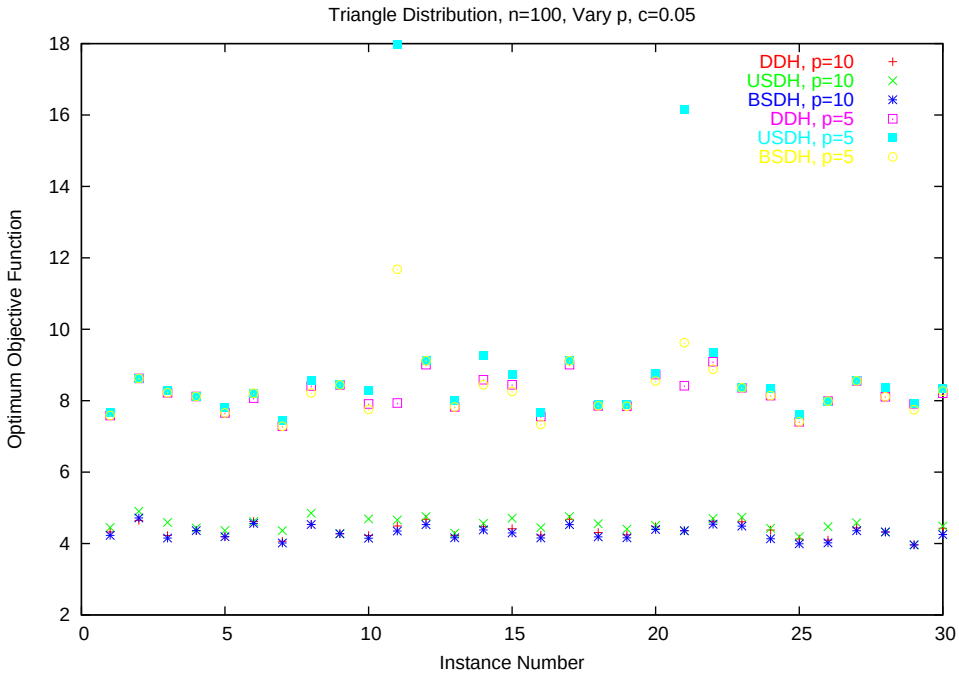

Figure 4.20: Optimum Objective function plot for Triangle distribution varying p

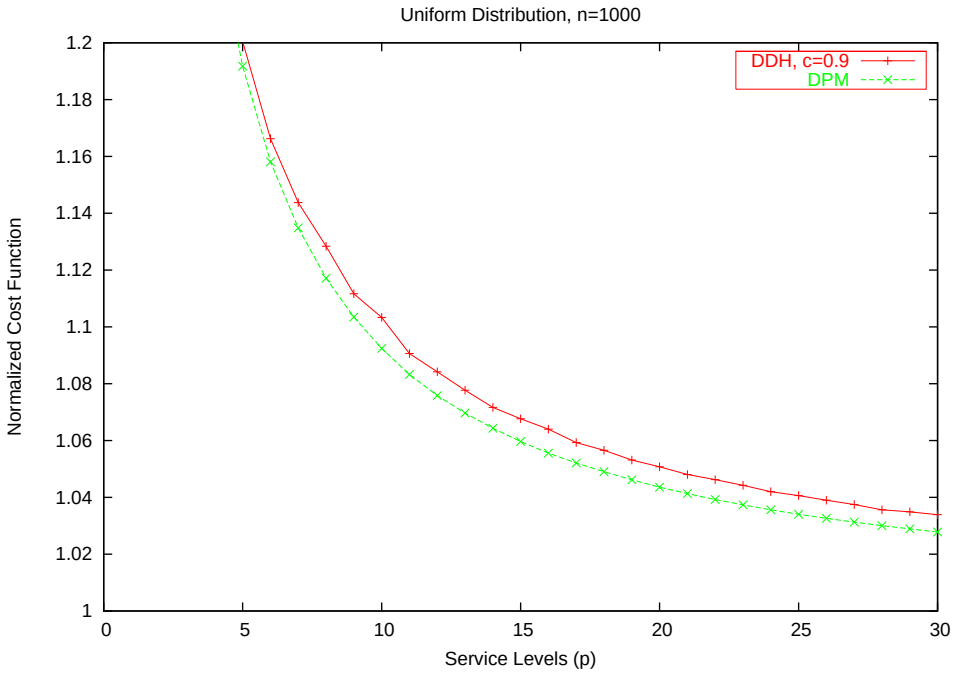

Figure 4.21: Bandwidth cost comparison of DPM1 algorithm to the DDH algorithm after normalization

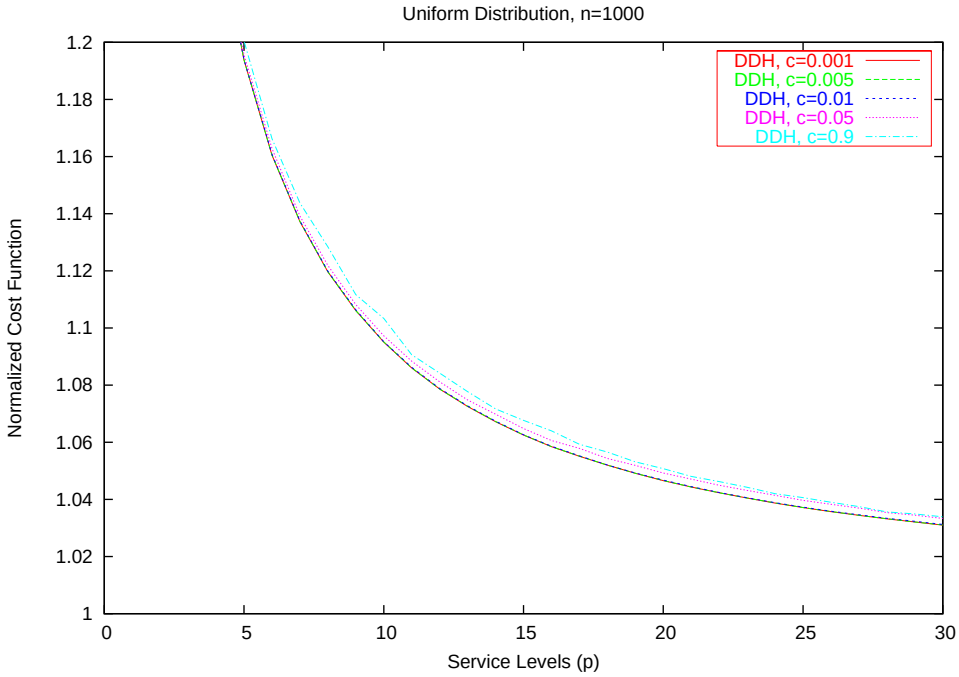

Figure 4.22: Normalized cost comparison of the DDH algorithm with varying value of c

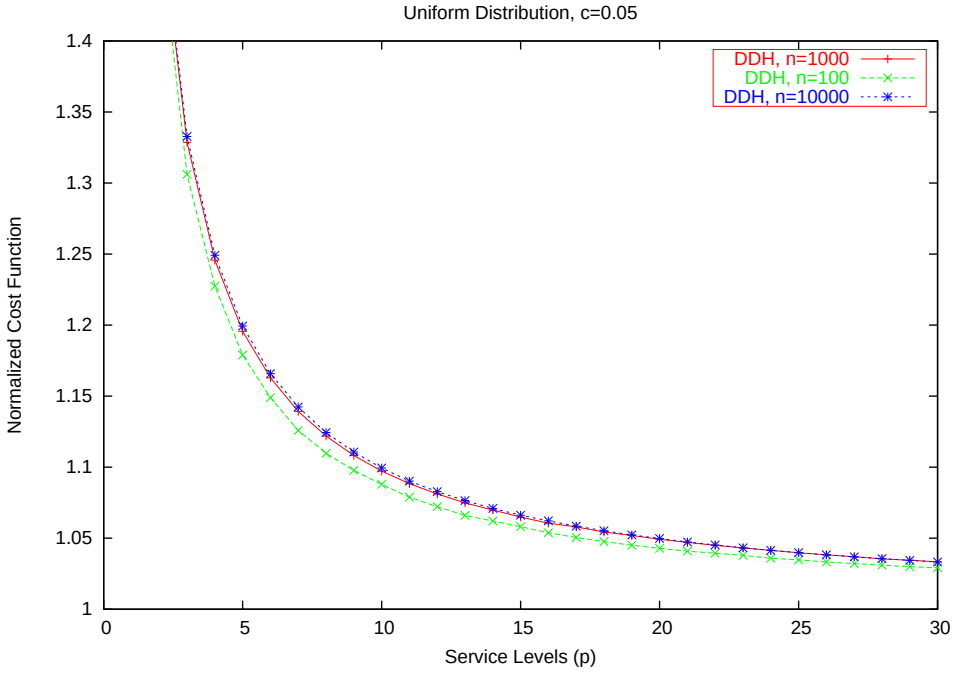

Figure 4.23: Normalized cost comparison of the DDH algorithm with varying value of n

We see from the above graphs that the normalized cost function of the variants of DDH and the DPM1 are close to 1 which is a desired result. Also the comparison of DPM1 to that of DDH is interesting as the difference in the normalized costs of these two algorithms is negligible. Hence we infer that we do not loose much in bandwidth when compared to DPM1 in spite of having the additional constraint to quantize traffic. Additionally, although not quantified, we definitely gain a lot on control/management costs when compared to DPM1 at the cost of a negligible utilization of excess bandwidth.

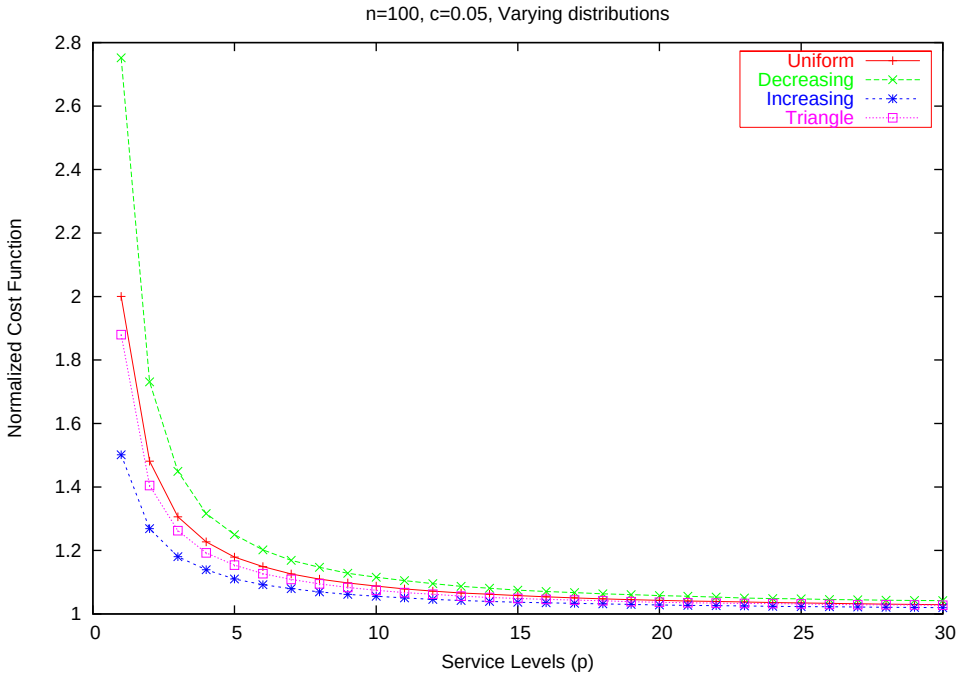

Figure 4.24: Normalized cost comparison of the DDH algorithm with varying the distribution

Chapter 5

Blocking Probability - A Simulation Study

The goal of this chapter is to demonstrate through event based simulations that the blocking probability by applying our optimization heuristics is comparable to that of the DPM1 algorithm presented in [4] and the continuous network. We define the blocking probability as the ratio of number of flows blocked due to unavailability of the residual bandwidth in the network to the total number of flows coming into the network.

5.1 The Simulation Setup and Implementation

We take a network of 16 nodes on which we run our simulation. The network structure is as shown in Figure 5.1. We randomly generate the user demands from a particular distribution between 0 and 1. We have decided to take 100,000 user demand requests from any of the six distributions presented in Table 4.1. All the links in the network have an initial bandwidth capacity of 2 units. We do this because we know that in reality, there would not be any requests that are close to the initial capacity of links in the network. There are other parameters that go into defining the values of the user requests when compared to the capacities of the network which we ignore here for simplicity. Hence

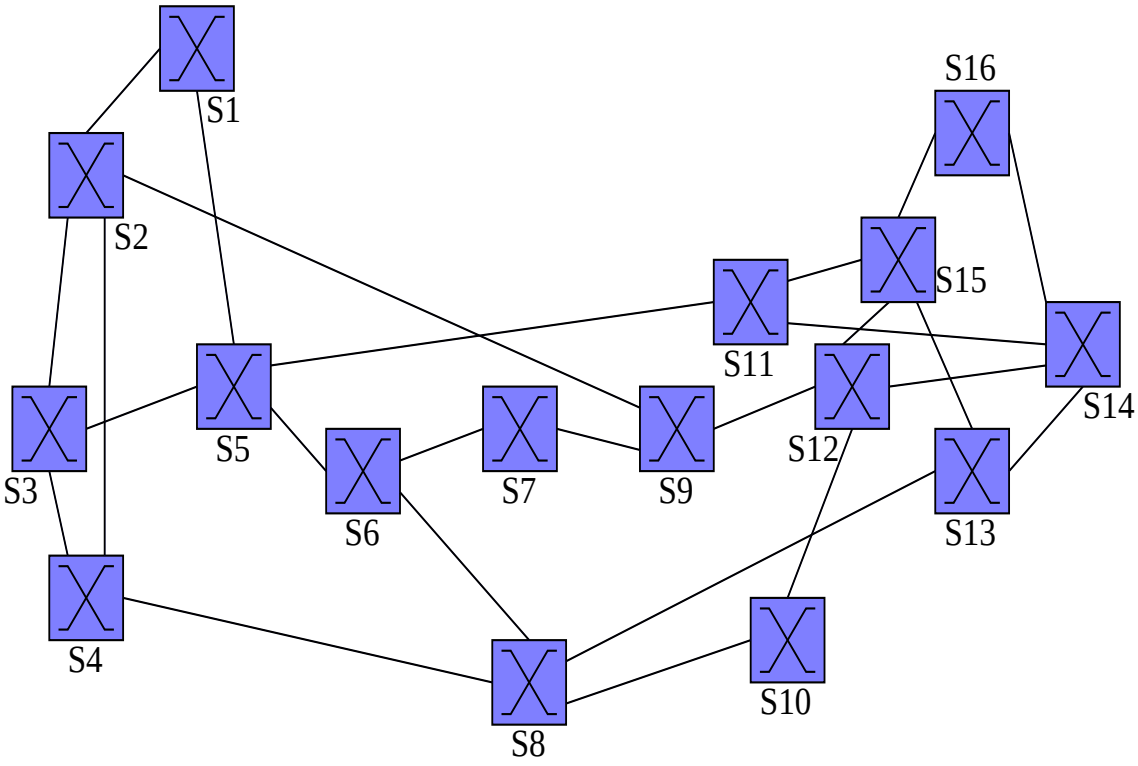

Figure 5.1: NSF Network consisting of 16 nodes

in our case, no user request can be more than half the network's initial bandwidth capacity.

The simulation is event based where each event describes the characteristics of each user demand. Each demand d_i is characterized based on the following tuple:

$$\{source_i, destination_i, path_i, bw_i, time_i, duration_i, id_i\}$$

$source_i$ is the source node and $destination_i$ is the destination node of the user demand d_i. The source and destination nodes are calculated by randomly selecting numbers between 1 and 16 such that the source and destination are not the same nodes for a given user demand. bw_i is the bandwidth specification of the corresponding demand request. $time_i$ is the time at the which the request occurs and $duration_i$ is the duration it takes for the data to reach the destination or in other words, it is the transmission delay. Both the $time_i$ and the $duration_i$ is generated from an exponential distribution with parameters λ and μ respectively. In our simulation, the value of μ is always set to 1 and the value of λ is varied accordingly. id_i is the identity of the demand d_i.

Finally $path_i$ is the set of vertices describing the path taken from the source to destination. This path is calculated as and when the request arrives depending on the current residual bandwidth in the links of the network. We use the Constrained Shortest Path First (CSPF) algorithm for determining the path taken. We have assumed the number of hops as the parameter for determining the shortest path between the source and the destination. Hence we implement the breadth first search (BFS) algorithm instead of the Dijkstra's shortest path algorithm for better efficiency with respect to time. To implement the CSPF algorithm, we first remove all the links from the network whose residual bandwidth is less than the bandwidth bw_i requested by the user demand d_i. We then apply the BFS algorithm on the resulting network to find the shortest path. This path is saved in the set $path_i$.

If the event arises where there could be more than one shortest path from source to destination, we define three different policies for selecting the path from the source to destination. Among the n different shortest paths, we select the link with the minimum residual bandwidth in each of the paths, say $min_1, min_2, \ldots, min_n$. Each of the min_j defines the weakest link in the corresponding paths thus making it the appropriate attribute to decide on, for selecting a path. Let us define min_j as the strength of the path $path_j$. We then define the best fit policy as choosing the path whose strength is maximum and worst fit policy as choosing the path whose strength is minimum. The third policy is to choose

the first path selected by the algorithm, appropriately naming it the first fit policy. Due to the nature of the BFS algorithm and for the simplicity of the simulation, we choose the first fit policy, consequently ignoring the strength of the paths.

There is also some amount of pre computation involved in the simulation. Since the simulation is basically to compare the blocking probabilities of the DDH algorithm to the continuous network and the algorithm presented in [4], we need to find the set of service levels given the value of p, where p is the number of service levels. This set also depends on the distribution used to generate the 100,000 user demand requests. In order to generate the p service points of a particular distribution using both DDH algorithm and the DPM1 algorithm presented in Chapter 2, we generate 100,000 demand points from the corresponding distribution and run both the algorithms. We then store the resulting p service points in two separate arrays corresponding to the two algorithms. The 100,000 demand points generated for this purpose would correspond to the empirical data that the network provider should collect in order to run the algorithm and get the required service points. In the continuous network, the service point will be the same as the requested demand point.

The simulation process is initialized by creating an event list containing the events corresponding to each demand, sorted with the time in which the events occurs. Each event could either be an arrival event where the demand request has arrived or a departure event in which case the user demand have been serviced. If the event E_i is an arrival event, we first calculate the amount of bandwidth $serve_i$ that has to be provided for the requested bandwidth bw_i depending on the algorithm that we use. For the continuous network, $serve_i = bw_i$. We then find out if there is a path from $source_i$ to $destination_i$ in the current state of the network using the CSPF algorithm discussed earlier. If there is a path $path_i$, we allocate the resource by reducing the residual bandwidth in each link of $path_i$ by an amount of $serve_i$. We also create a corresponding departure event to be inserted in the sorted event list, containing the time of departure calculated as $time_i + duration_i$, the provided bandwidth $serve_i$ and the path $path_i$ so that the bandwidth provided for this request is restored in that path. If on the other hand, there is no path possible in the current network for the requested bandwidth, the event is considered to be blocked in which case a global counter is incremented. If the event E_i is a departure event, we restore the bandwidth on all the links of path $path_i$ by $serve_i$. Finally, the blocking probability is calculated as the number of events blocked by the total number of events arriving into the

network which is 100000.

5.2 Simulation Results and Discussion

We now consider the blocking probabilities of the system of the 16 node network for 100000 input requests. All the graphs in this section have been plotted using logarithmic scale and are plots of the blocking probabilities with λ, the rate of arrival into the system. The entire simulation is conducted for 30 instances, each with a different seed for generating the 100000 demand requests from a particular distribution. The mean blocking probability is calculated for the 30 instances and plotted. This is done to obtain statistically significant results. Additionally, confidence intervals are also calculated and plotted for the 30 instances with 95% confidence. As mentioned earlier, μ is always set to 1 for all simulations.

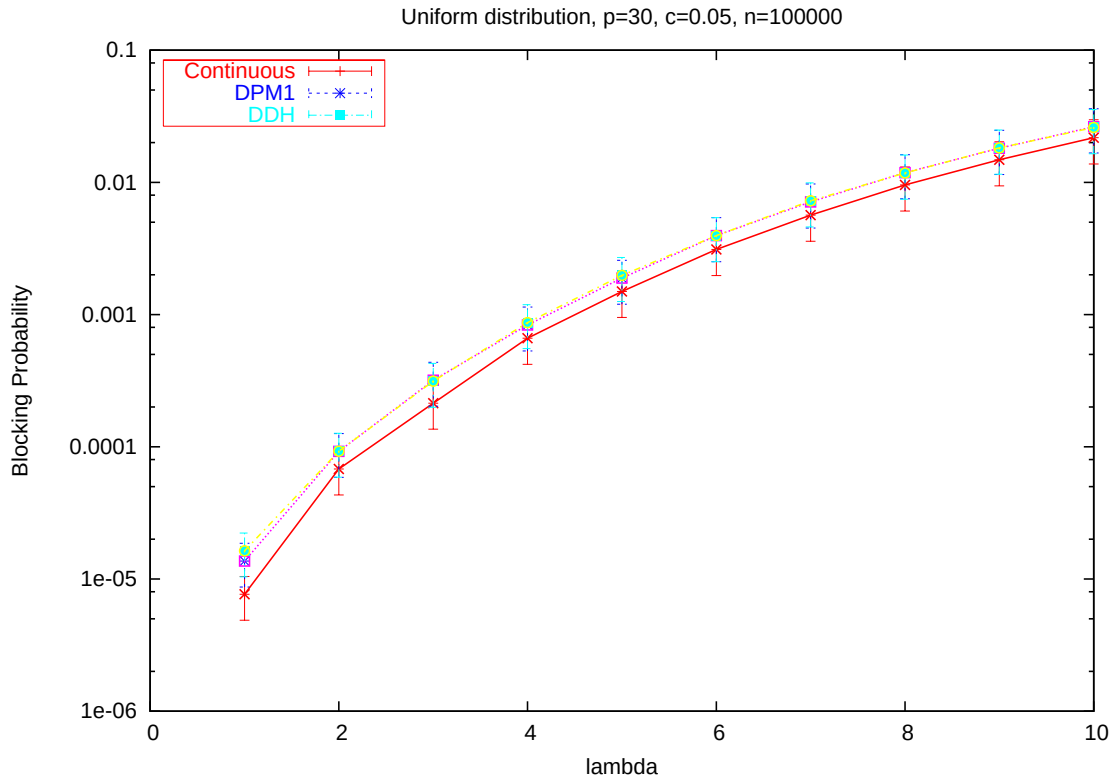

Figure 5.2: Blocking Probability comparisons for DPM1, continuous and DDH

For the result shown in Figure 5.2, three types of bandwidth allocation schemes

were chosen namely DPM1, continuous system and DDH. Hence the entire simulation was performed for each of these schemes. We generated random instances for the 100000 requests using the uniform distribution for all the three simulations. Pre computation of the service points was done for the DDH with $c = 0.05$, $p = 30$ and DPM1 with $p = 30$. Blocking probabilities approach close to 5% as λ reaches 10.

It can be inferred from Figure 5.2 that although the continuous system has the lowest blocking probability, the blocking probabilities of both DDH and DPM1 are very close to that of the continuous system. This is a positive result considering the other advantages of DDH when compared to DPM1 and the continuous system with respect to control/management cost reduction. It also implies that the excess bandwidth spent using DDH results in a negligible difference in blocking request when compared to the continuous system and DPM1. As seen in the graph, there is only a difference of 0.5% in the blocking probability of DDH and continuous system for λ as large as 10.

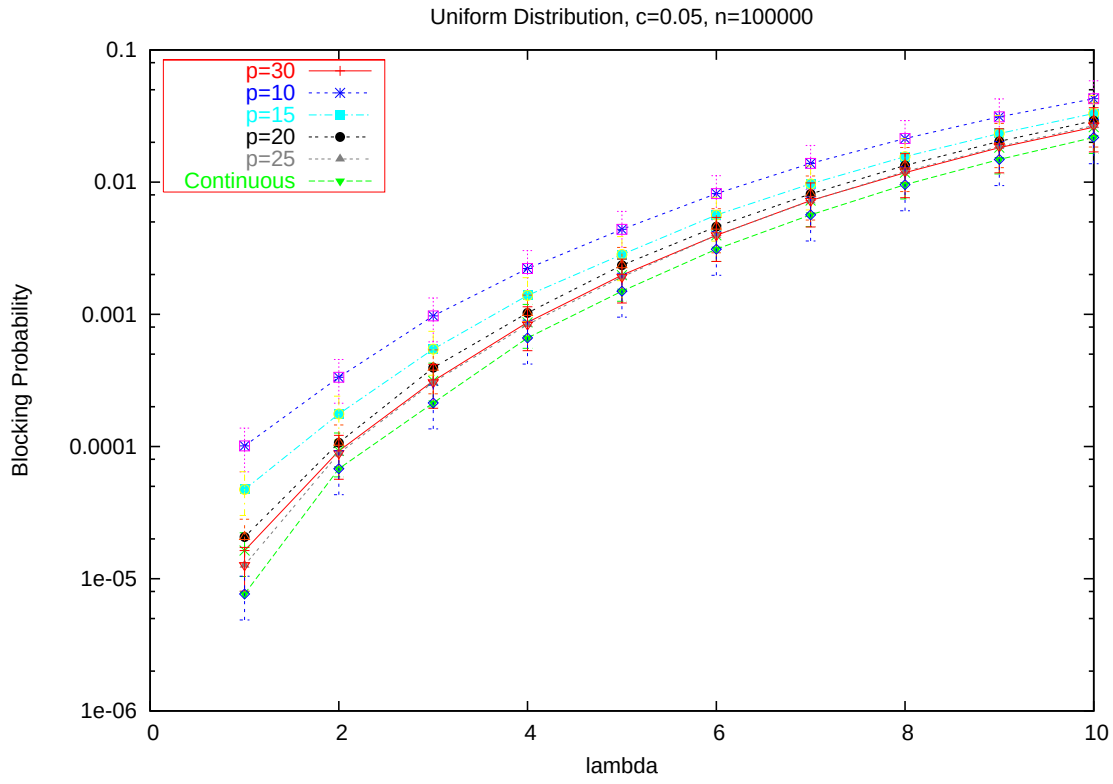

Figure 5.3: Blocking Probability for DDH varying p

Figure 5.3 is a plot of blocking probabilities using the DDH scheme but varying the value of p. Pre computation of the service points is done with different values of p and the simulation is run for each of these pre computed results. These are also plotted with the continuous system. We notice from the graph that as the value of p increases, the blocking probability decreases for a given value of λ. This is because, as the number of service points increases, the objective function of the DDH decreases thereby decreasing the excess bandwidth that is to be spent on the system. Since we save up on the bandwidth as p increases, the number of requests blocked decreases hence decreasing the blocking probability overall.

For the result shown in Figure 5.4, we use the DDH scheme and vary the distribution from which the requests are generated. All the pre computation is done with $p = 30$, $c = 0.05$ and $n = 100000$. We use the increasing, decreasing, uniform and the triangle distributions for this graph.

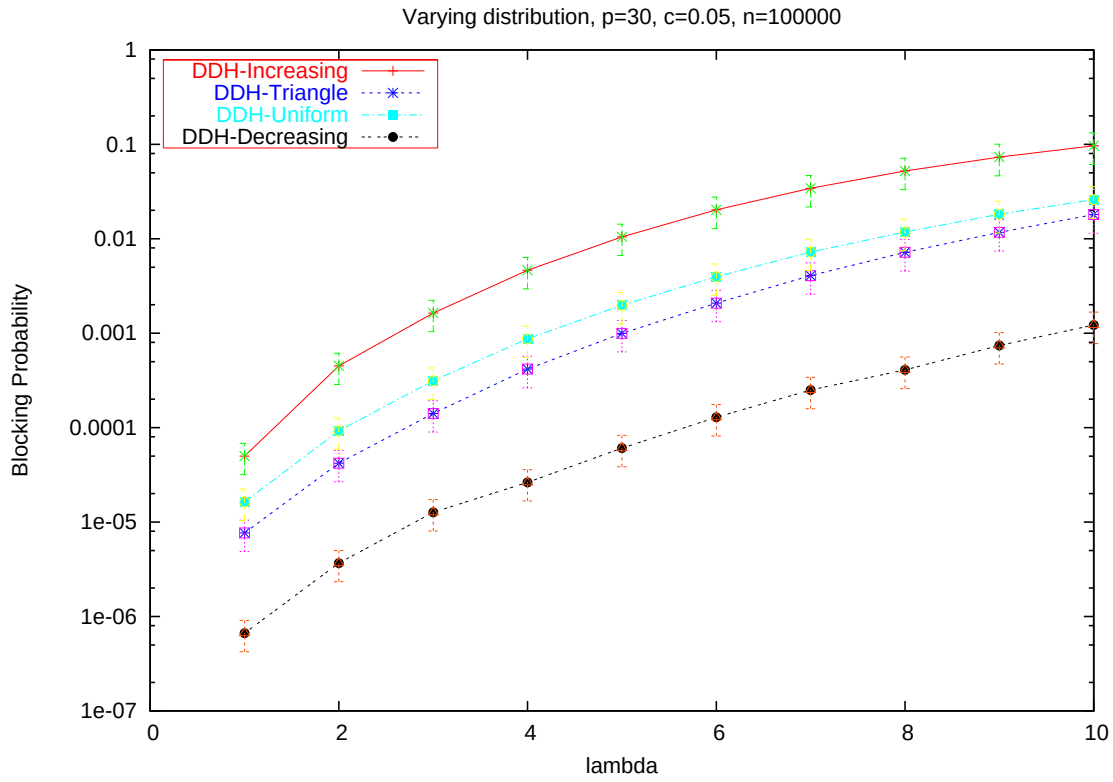

Figure 5.4: Blocking Probability for DDH varying the user demand distribution

As we see, the decreasing distribution turns out to have the lowest blocking probability among all distributions. This is because majority of the requests are very small and only a few of them are large thereby accommodating more requests into the network. The reverse is true for the increasing distribution thus making it suffer with higher blocking probabilities.

Based on the results presented here, we conclude that the amount of traffic calls blocked using the DDH scheme when compared to the continuous system and DPM1 is very minuscule and is well compensated with the advantages that we gain using this scheme as discussed in previous chapters. Hence, the blocking of requests due to excess bandwidth giveaway is not a problem to consider and is traded off very well with the pros of using such a scheme for traffic quantization.

Chapter 6

Summary and Future Work

6.1 Summary

We have considered the problem of quantizing traffic demands such that the network control/management costs as well as the excess bandwidth spent due to quantization is minimized. We established the need for an additional constraint in order to achieve our goal. We presented several algorithms and heuristics, influenced by the location theory and solved them using dynamic programming and the properties of totally monotone matrices. We evaluated the performance issues of such algorithms and the effect of such quantization on blocking probability of input traffic. Based on our results, we conclude that adopting our approach to quantizing traffic suffers little on additional bandwidth consumption when compared to other traffic quantization techniques studied in the past. The blocking probability of the proposed algorithms also has small difference when compared to the continuous network. Additionally we gain on the control/management cost that is not otherwise possible using other quantization approaches and the continuous network.

6.2 Future Work

Although we have outlined several efficient algorithms for quantizing the input traffic demands, we believe that there is ample scope for future research in developing

algorithms that will implement these ideas. Our quantization approaches give an insight to deriving heuristics that can solve other potential NP-hard problems related to traffic parameters such as bandwidth, delay, etc. in polynomial time. It would be interesting to find optimization techniques for problems of similar kind. It would also be interesting to study quantization techniques for multiple QoS parameters for each traffic request such as the combination of bandwidth and delay.

Bibliography

[1] Alok Aggarwal and James Park. Notes on searching in multidimensional monotone arrays, 1988.

[2] C.P.M Van Hoesel A.P.M.Wagelmans and A.W.J Kolen. Economic lot sizing: An algorithm that runs in linear time in the wagner-whitin case, 1989.

[3] R. Hassin and A. Tamir. Improved complexity bounds for location problems on the real line, 1990.

[4] Laura E. Jackson and George N. Rouskas. Optimal traffic quantization in packet-switched networks, 2003.

[5] N.Megiddo and K.J Supowit. On the complexity of some common geometric location problems, 1984.

[6] Alok Aggarwal Maria M.Klawe Shlomo Moran Peter Shor and Robert Wilber. Geometric applications of a matrix-searching algorithm. Algorithmica, pages 195–208. Springer Verlag New York Inc, 1987.

[7] Robert Wilber. The concave least-weight subsequence problem revisited, 1987.