International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:3, 2009

Coerced Delay and Multi Additive Constraints QoS Routing Schemes

P.S. Prakash, S. Selvan

Abstract**—**IP networks are evolving from data communication infrastructure into many real-time applications such as video conferencing, IP telephony and require stringent Quality of Service (QoS) requirements. A rudimentary issue in QoS routing is to find a path between a source-destination pair that satisfies two or more end-to-end constraints and termed to be NP hard or complete. In this context, we present an algorithm Multi Constraint Path Problem Version 3 (MCPv3), where all constraints are approximated and return a feasible path in much quicker time. We present another algorithm namely Delay Coerced Multi Constrained Routing (DCMCR) where coerce one constraint and approximate the remaining constraints. Our algorithm returns a feasible path, if exists, in polynomial time between a source-destination pair whose first weight satisfied by the first constraint and every other weight is bounded by remaining constraints by a predefined approximation factor (α). We present our experimental results with different topologies and network conditions.

Keywords**—**Routing, Quality-of-Service (QoS), additive constraints, shortest path, delay coercion.

I. Introduction

In Multi Constraint Quality-of-Service (QoS) routing, one finds a path from a source to destination that satisfies many QoS constraints such as cost, delay and probability of packet loss [2], [13], [15]. We model a computer network by a set of 'v' vertices and 'e' edges, where vertices and edges represent nodes and links respectively. Each edge has K weights assigned to it, representing delay, cost, etc. The corresponding path weight is obtained by adding the weight of edges in case of additive metrics (e.g. delay, cost) and multiplying in case of multiplicative metrics (e.g. packet loss), etc. Bandwidth is considered as concave or min/max (bottleneck) metric where the corresponding weights of the path is the smallest of the weights of the edges on that path. [5].We concentrate on additive metrics only as the concave metric issues would relatively be easily solved.

Problems involving two or more QoS constraints have been shown to be NP-Complete. [9]. Many researchers have studied this problem in the last few years and most of the existing works concentrate on Multi Constraint Path Problem (MCP) with two additive constraints. Delay Constrained Least Cost path problem (DCLC) where two edge weights are cost and delay, and one seeks a minimum cost path subject to a given delay constraint. Chen [2] studied DCLC problem where we want to find a path that satisfies both the delay and cost constraints.

In [3] Ergun *et al*, presented a polynomial approximation scheme for acyclic graphs. Lorenz and Raz [10] presented an enhanced polynomial algorithm. All these algorithms find a source-destination path whose delay is at most Δ_d (delay constraint) and cost is not more than (1+α) times the cost of the least- cost delay constrained path, provided that there is a source-destination path whose delay is at most Δ_d. If this condition is not satisfied then all these algorithms will terminate which means that the problem is infeasible.

Many researchers deal with multi constraint problem with two constraints. Goel *et al* [4] presented an approximation algorithm for the single source all destinations delay sensitive problem. Linear combination of two weights and presented some simple algorithms in [17].

Xiao [16] presented a primal simplex approach. Orda and Sprintson [12] presented a pre computation scheme for QoS routing with two additive parameters and efficient approximation algorithms [11] for computing a pair of disjoint QoS paths. Van Mieghem *et al* [14] proposed a self adaptive multiple constraints routing problem. Korkmaz and Krunz [7] proposed a general multi constrained randomized heuristic with two additive constraints. Yuan [18] presented a limited granularity algorithms and limited path heuristic.

The rest of this paper is organized as follows. In section II, we present our MCPv3 algorithm. In section III, we discuss about DCMCR and its pseudo code. In section IV, we present our results and finally a concise conclusion in section V.

II. MCPv3 Algorithm

MCPv1 (G,s,d,K,W,w): We represent our network for all algorithms by a graph G=(V,E,w), where 'V' is the set of vertices, 'E' is the set of edges and w =(w_1,…,w_k) is an edge weight vectors so that $w_k(e)\geq 0$ is kth weight of edge 'e', $\forall e \in E$, $1\leq k\leq K$. For a path 'p' in G, kth weight of path 'p' denoted by $w_k(p)$ is the sum of kth weights over the edges in p, $w_k(p)=\sum_{e\in E} w_k(e)$. Integer constant K is to denote the number

Manuscript received 18 June 2009.
P. S. Prakash, is with Computer Science and Engineering (PG) Department, Sri Ramakrishna Engineering College, Coimbatore 641 022, TamilNadu, India. (Phone: +91 422 2312021, 99945 25625; e-mail: prakashpsrajan@rediffmail.com).
Dr. S. Selvan is with St.Peter's Engineering College, Chennai, India.

of QoS metrics. A constant W = ($W_1,\ldots,W_k$), where each W_k is a positive constant for a source-destination pair s-d. In our model, we assume all variables are assumed to be real values. We assume $w_k(p) \le W_k$ is called K^{th} QoS constraint. We represent this as MCPv1.

An s-d path 'p' satisfying all K QoS metrics is called a feasible path of our MCPv1. We say that MCPv1 is feasible if it has a feasible path and infeasible otherwise and the problem of MCPv1 is found to be NP-hard.

MCPv2.1 (G,s,d,K,Δ_K,w): A Graph G=(V,E) with K edge positive integer valued edge weights $w_k(e), 1 \le k \le K$, associated with each edge $e \in E$;a positive constant Δ_K and a source-destination pair s-d. Our objective is to find a s-d path 'p' such that $w_k(p) \le \Delta_K, 1 \le k \le K$.

MCPv3 (G,s,d,K,W,w): A graph G= (V,E) with K edge weights $w_k(e), 1 \le k \le K$ associated with each edge $e \in E$; a positive constant W and a source-destination pair. Our objective is to find s-d path 'p_{opt}' such that $w_k(p_{opt}) \le x_{opt}.W$ for all $1 \le k \le K$ where x_{opt} is smallest real number and $x \ge 0$ such that there exists an s-d path 'p' satisfying $w_k(p) \le x.W$ for all $1 \le k \le K$.

We define x_{opt} is the optimal value of MCPv3 and call p_{opt} an optimal path of MCPv3. We choose $x_{opt} \le 1$ if and only if MCPv1 is feasible. When $x_{opt} \le 1$ any optimal solution of MCPv3 is a feasible path for MCPv1. But all MCPv1 feasible path solutions cannot be applied to MCPv3, i.e. reverse is not true, when a feasible of path for MCPv1 is not an optimal solution of MCPv3, we need to choose $x_{opt} < 1$.

The algorithm computes an auxiliary edge weight $w_{aux}(e)$ as the maximum of all K edge weights $w_1(e),\ldots,w_k(e)$ divided by W. It then computes a shortest s-d path p_{aux} using this auxiliary weight. The path p_{aux} is guaranteed to be K-approximation of MCPv3. The auxiliary edge weights can be computed locally at each node and shortest path can be computed using conventional algorithms. K-approximation algorithm can be implemented as either a centralized or distributed algorithm and can be used by existing link state or distance vector routing protocols [8].

$W_{aux}(p_{aux})/K$ is the lower bound for x_{opt} and $W_{aux}(p_{aux})/2$ is an upper bound for x_{opt}. If $W_{aux}(p_{aux})=0$ we can conclude that p_{aux} is also a feasible solution to MCPv3. If $W_{aux}(p_{aux})>0$, then we have a 'testing procedure' to have some pairs of lower-upper bounds so that ratio of upper bound to lower bound goes sufficiently close to 1. Then we need to solve MCPv2.1 to obtain an $(1+\alpha)$ approximation to MCPv3. As found in [7], we denote UB[i] is an approximate upper bound, and we initialize lower bound of x_{opt} to LB[i] = $W_{aux}(p_{aux})/K$ and initial upper bound of x_{opt} to UB[i] = $W_{aux}(p_{aux})/2$, where i=0 initially.

Our algorithm finds a $(1+\alpha)$ approximation to MCPv3. Let LB[i] and UB[i] denote the lower and upper bound as explained above.

$$LB[i] \le x_{opt} \le 2UB[i] \tag{1}$$

is true for i=0.

We construct an auxiliary graph G_{aux} having an auxiliary edge weight $w_{aux}(e)=[w_k(e).\gamma]+1$ for every $e \in E$. We define $\gamma = \frac{v-1}{LB.W.\alpha}$ and let p_{opt} is an optimal solution to MCPv3, which means that p_{opt} is an source-destination s-d path such that $w_k(p_{opt}) \le x_{opt}.W$ for k=1,2,...,K. Since, $w_{aux}(e)=\left\lceil w_k(e).\frac{v-1}{LB\cdot W\cdot\alpha}\right\rceil+1 \le w_k(e)\cdot\frac{v-1}{LB\cdot W\cdot\alpha}+1$ for all edges $e \in E$. We have,

$$w_{aux}(p_{opt}) \le \frac{w_k(p_{opt})\cdot(v-1)}{LB\cdot W.\alpha}+(v-1) \tag{2}$$

We know that, $x_{opt} = \frac{w_k(p_{opt})}{W}$, therefore,

$$w_{aux}(p_{opt}) = \frac{x_{opt}\cdot(v-1)}{LB\cdot\alpha}+(v-1)$$

We know that, 2UB = x_{opt}, hence,

$$w_{aux}(p_{opt}) = \frac{2UB\cdot(v-1)}{LB\cdot\alpha}+(v-1) \tag{3}$$

Since w_{aux} have integer values, from (3),

$$w_{aux}(p_{opt}) \le \left\lceil \frac{2UB\cdot(v-1)}{LB\cdot\alpha}\right\rceil+(v-1) \text{ for all } 1\le k\le K \tag{4}$$

It may be seen from (4) p_{opt} is a feasible solution to MCPv2.1. Therefore our algorithm is guaranteed to return a feasible path, if exists. Also from (3) we learn that,

$$\max_{1\le k\le K} w_{aux}(p_{opt}) \le x_{opt}\cdot\frac{v-1}{LB.\alpha}+(v-1) \tag{5}$$

Let p_{aux} be the s-d path in the final stage of algorithm and p_{aux} is feasible solution to $MCPv2.1(G_{aux},s,d,K,\Delta_K)$, where Δ_K is the smallest integer less than or equal to $\left\lceil\frac{2UB\cdot(v-1)}{LB\cdot\alpha}\right\rceil+(v-1)$ such that MCPv2.1 is feasible. Since p_{aux} is optimal while p_{opt} is only feasible, the maximum path weight of p_{aux} cannot exceed the maximum path weight of p_{opt}. Hence we can say,

$$\max_{1\le k\le K} w_{aux}(p_{aux}) \le \max_{1\le k\le K} w_{aux}(p_{opt}) \tag{6}$$

By combining (5) and (6), we get

$$\max_{1\le k\le K} w_{aux}(p_{aux}) \le x_{opt}\cdot\frac{v-1}{LB\cdot\alpha}+(v-1) \tag{7}$$

We also have,

$$\max_{1\le k\le K} w_{aux}(p_{aux}) = \sum_{e\in p_{aux}} w_{aux}(e) \ge \sum_{e\in p_{aux}} \frac{w_k(e)\cdot(v-1)}{LB\cdot W\cdot\alpha}$$
$$= w_k(p_{aux})\cdot\frac{(v-1)}{LB\cdot W\cdot\alpha} \text{ for all } 1\le k\le K \tag{8}$$

Now combining (7) and (8), we get

$$w_k(p_{aux})\frac{(v-1)}{LB\cdot W\cdot\alpha} \le x_{opt}\frac{(v-1)}{LB\cdot\alpha}+(v-1) \text{ for all } 1\le k\le K \tag{9}$$

$$
\begin{aligned}
w_k(p_{aux}) \le & \left\lceil x_{opt}\frac{(v-1)}{LB\cdot\alpha} + (v-1)\right\rceil \left\lceil \frac{LB\cdot W\cdot\alpha}{v-1} \right\rceil \\
\le & \left\lceil \frac{x_{opt}(v-1)+(v-1)\cdot LB\cdot\alpha}{LB\cdot\alpha} \right\rceil \cdot \left\lceil \frac{LB\cdot W\cdot\alpha}{(v-1)} \right\rceil \\
\le & \ (x_{opt} + LB\cdot\alpha)\, W \\
\le & \ x_{opt}\, W + LB\cdot\alpha\cdot W \\
\le & \ LB\cdot W\,(1+\alpha) \qquad [\because LB = x_{opt}]
\end{aligned}
$$

$$w_k(p_{aux}) \ \le \ x_{opt}\ W(1+\alpha) \ \text{ for all } 1 \le k \le K \qquad (10)$$

This proves that p_{aux} is an $(1+\alpha)$ approximation to MCPv3. We may reduce the time complexity by carefully choosing various parameters that decides γ.

If k=2, MCPv3 runs in much lesser time (i.e. O(v.e/α)) comparing to any other conventional algorithms. Here we approximate both delay and cost and there is another methodology where one constraint may be coerced and remaining constraints could be approximated when $k \ge 2$ which will discussed in next section.

A. Pseudo Code

UB upper bound	LB lower bound
p_{opt} feasible shortest path	p_{aux} auxiliary path
α approximation factor	c smallest constraint
T test value	v number of vertices
S set of vertices V	Q minimum priority queue
s-d source-destination pair	G_{aux} auxiliary graph
(u,v) edge	K number of QoS constraints
W constraint bound	Δ_K auxiliary constraint bound

(V,E) set of vertices and set of edges.

$w_k(p_{opt})$ k^{th} weight of feasible path p_{opt}

$w_{aux}(p_{aux})$ k^{th} auxiliary weight of auxiliary path p_{aux}

γ real number to construct auxiliary graph

len[ ][ ] (K-1) dimensional array to store the length of the edge.

pred[ ][ ] (K-1) dimensional array to store the predecessor.

```
void main( ) {
    int k, K, $w_{aux}$[ ], $w_k$[ ], W, c, S, V, E, Q, T, $\Delta_K$ ;
    int i=0, α, γ, LB[ ], UB[ ], len[ ][ ],pred[][ ];
    string $p_{opt}$, $p_{aux}$, e, u, v, b, s;
    /* compute auxiliary edge weight */
compute_auxweight( ) {
        K=3;
            for(e=1;e<=E;e++) {
                for(k=1;k<=K,K++) {
                    $w_{aux}$(e)=max($w_k$(e)/W); } } }
    /* compute shortest path using Dijkstra's algorithm */
compute_shortest path( ) {
    /* initialize the set S of vertices and minimum priority queue Q */
        S=0;
        Q=V(G);
            while(Q!=0) {
                Q=V-S;
                u=Q;
                S=S+u;
                $w_{aux}$($p_{aux}$)=len[S];} }
                    if($w_{aux}$($p_{aux}$)<= max($w_k$($p_{opt}$)/W)) {
                    $p_{opt}$ = $p_{aux}$;
            printf("feasible path is returned %s",$p_{opt}$);
            break ( ); }
        else {
    /* initialize lower LB and upper bound UB */
            LB[0] = $w_{aux}$($p_{aux}$)/K;
            UB[0] = $w_{aux}$($p_{aux}$)/2; }
    /* update the lower bound and upper bound values using testing procedure */
            while(UB[i] >= 2*LB[i]) {
                T =sqrt(LB*UB);
                α = 1;
                if($w_{aux}$($p_{aux}$) < = ((v-1)/α)+(v-1) ) {
                String Test(T,α) = "yes"; }
                else {Test(T,α)= "no"; }
                if(Test(T,α) = = "yes") {
                    LB[i] = T;}
                else { UB[i] = T; }}
                    i++;
                    UB= UB[i];
                    LB = LB[i];
    /* construct an auxiliary graph $G_{aux}$ which is an instance of graph G */
    construct_auxgraph( ) {
            γ = (v-1)/LB[i]*α*W;
            $w_{aux}$(e) = (γ* $w_k$(e)+1); }
    mcpv2.1( ) {
    /*initialize (K-1) dimensional array to store length and predecessor of edges*/
        K=3;
            for(k=1;k<=K;k++) {
                len[v,$c_k$] = ∞;
                pred[v,$c_k$] = null;
                len[s,$c_k$] = 0; }
    /* update array values */
            for(k=1;k<=K;k++) {
              for((u,v)=1;(u,v)<=E;(u,v)++) {
                if(len[v,$c_k$] > len[u,$b_k$] +$w_k$(u,v) ) {
                    len[v,$c_k$] = len[u,$b_k$] +$w_k$(u,v);
                        pred[v,$c_k$] = u; }}}
    /* find the s-d path $p_{aux}$ such that $w_{aux}$($p_{aux}$) < = $\Delta_K$
                if(len[d,$c_k$] < = $\Delta_K$ ) {
                  $p_{opt}$ = $p_{aux}$;
            printf("Feasible path is returned %s",$p_{opt}$);
            break ( ); }
55.                 else {
56.                     printf("No feasible path found");
57.                     break ( ) ;}}}
```

Fig.1. MCPv3 Algorithm

Initially we compute an auxiliary edge weight w_{aux} for each edge 'e' of graph G by considering the *max* ($w_k(e)/W$) a shortest path algorithm is applied. This operation is explained between line 5 and 17.

Then we compare the auxiliary weight $w_{aux}(p_{aux})$ with achieved shortest path ratio. Our algorithm returns the path, if auxiliary weight of the path is less than achieved shortest path ratio, else upper and lower bounds are to be initialized as shown in lines from 18 to 24.

Upper and lower bounds are refined based on the requirements of QoS metrics. Then we construct an auxiliary graph G_{aux} which is same as graph G except that the edge weighting function $w_k(e)$ is changed to $w_{aux}(e)$, such that $w_{aux}(e) = (\gamma \cdot (w_k(e)+1))$ as shown between lines 25 and 39.

Then MCPv2.1 is applied to find feasible path. Initially (K-1) dimensional array is initialized to store the length and predecessor of each edge. Then values of these arrays are updated based on the values returned by the algorithm as shown in lines from 40 to 50.

Finally, we compare the auxiliary weight of the auxiliary path with the auxiliary constraint bound. If the auxiliary weight is less than or equal to auxiliary constraint bound, then we returned p_{aux} is a feasible path or otherwise as in lines between 51 and 57. We depicted MCPv3 Algorithm in figure 1.

III. Delay Coerced Multi Constrained Routing (DCMCR)

We define a network by an edge weighted directed graph G= (V,E,w), where V is the set of 'v' vertices and E is the set of 'e' edges and w =($w_1,\ldots,w_k$) is an edge weight vector so that $w_k(e)\geq 0$ is the Kth weight of edge 'e', for all $e \in E$, $1\leq k\leq K$.

For a path 'p_{DMR}' in G, the Kth weight of path p_{DMR} is denoted by $w_k(p_{DMR})$, is the sum of Kth weights over the edges on p_{DMR}. i.e. $w_k(p_{DMR}) = \sum_{e\in p_{DMR}} w_k(e)$.

In our algorithm an edge-weighted directed graph G= (V,E,D,C) where each edge $e \in E$ is associated with a delay D(e) and a cost C(e). We assume both delay and cost are non-negative real values. Δd is the delay constraint for the source-destination pair.

Our objective is to find a path 'p_{DMR}' for a given s-d in G such that $C(p_{DMR}) = \sum_{e\in p_{DMR}} C(e)$ is minimized subject to the constraint $D(p_{DMR}) = \sum_{e\in p_{DMR}} D(e) \leq \Delta_d$. An source destination path 'p_{DMR}' is called delay constrained path if $D(p_{DMR}) \leq \Delta_d$. Our algorithm searches for a least-cost delay constrained path and is denoted by p_{DMR}. We also use x_{opt} to denote $C(p_{DMR})$ and call it the optimal value of $DCMCR(G,s,d,\Delta_d,D,C)$. In this algorithm we coerce the first constraint $w_1(p_{DMR})\leq \Delta d$ in the s-d pair.

MCPv2.2 $(G,s,d,K,\Delta_d,\Delta_c,w)$: An edge-weighted directed graph G=(V,E,w) with K non negative edge weights $w_k(e), 1\leq k\leq K$, associated with each edge $e\in E$ such that $w_k(e)$ is a positive integer for $e \in G$ and $k = 2,\ldots..K, \Delta_d$ and Δ_c are positive integer constants. Our objective is to find a source-destination path 'p' such that $w_1(p) \leq \Delta_d$ and $w_k(p) \leq \Delta_c$, $2 \leq k \leq K$. In MCPv2.2, the edge weights $w_{2,\ldots,}w_k$ are positive integer-values.

A. Algorithm

DCMCR (G, s, d, K, $\Delta_{d,}$ D, C)**:** Our algorithm, for any given value $\alpha > 0$ returns a path 'p_{DMR}' for a source-destination pair s-d that is an $(1+\alpha)$ approximation of DCMCR (G, s, d ,K, $\Delta_{d,}$ D, C)**.**

We find bottleneck edge cost 'c' such that an s-d path p_{DMR}, with $D(p_{DMR})\leq \Delta_d$ and $C(e)\leq c$ for all $e\in p_{DMR}$. Secondly any s-d path 'p', $D(p)\leq \Delta_d$ must contain at least one edge 'e' $C(e)\geq c$. Here D(p) is any path in s-d, this can be accomplished by any conventional algorithm, such as Dijkstra's shortest path algorithm.

We know that $c \leq x_{opt} \leq c.v$, where 'v' is the number of vertices and 'c' is the smallest constraint.

$$LB[0] \leq x_{opt} \leq UB[0] \leq v.LB[0] \tag{11}$$

(since LB=c and UB=c.v)

Let p_{DMR} denote optimal solution of DCMCR(G,s,d,K,Δ_d,D,C) that is p_{DMR} is an s-d path such that,

$$D(p_{DMR}) \leq \Delta_{d,}\ C(p_{DMR}) \leq x_{opt} \tag{12}$$

We construct an auxiliary graph G_{aux}, having an auxiliary edge weight, $C_{aux}(e)=[C(e).\gamma]$. We define $\gamma = \frac{(v-1)}{LB\cdot\alpha}$.

Here we construct an auxiliary graph G_{aux} which is an instance of graph G to find the optimal value x_{opt} of our algorithm DCMCR. To find a x_{opt}, we perform a testing procedure defined in our algorithm. Hence,

$$C_{aux}(p_{DMR}) = \sum_{e\in p_{DMR}} [(C(e)\cdot\gamma)+1]$$

$$\leq \sum_{e\in p_{DMR}} [(C(e)\cdot\gamma)+(v-1)]\ [\because p_{DMR} \text{ has at most}$$

(v-1) edges].

$$\leq (v-1)+\gamma \sum_{e\in p_{DMR}} C(e)$$

$$\leq (v-1)+\gamma C(p_{DMR}) \tag{13}$$

$$\leq (v-1)+C(p_{DMR})\frac{(v-1)}{LB\cdot\alpha}$$

Substitute the value of $C(p_{DMR})$ from (12) ,

$$C_{aux}(p_{DMR}) = (v-1) + x_{opt} \cdot \frac{(v-1)}{LB \cdot \alpha}$$

Substitute the value of x_{opt} from (11),

$$C_{aux}(p_{DMR}) = (v-1) + \frac{UB \cdot (v-1)}{LB \cdot \alpha} \qquad (14)$$

The value of $C_{aux}(p_{DMR})$ is scaled down to nearest lower integer. From (14), the path p_{DMR} is feasible solution of $MCPv2.2\left(G_{aux}, s, d, \Delta_d, \left\lceil \frac{UB \cdot (v-1)}{LB \cdot \alpha} \right\rceil + (v-1), D, C_{aux}\right)$

Therefore, we find a path p_{aux} (from auxiliary graph) and this path p_{aux} may be different from p_{DMR}. If p_{aux} is feasible, we are guaranteed to return a feasible path in DCMCR.

Now, we prove that path p_{aux} found by algorithm MCPv2.2 is guaranteed to be an (1+α)-approximation of DCMCR. Since p_{aux} is computed by the algorithm, we have,

$$D(p_{aux}) \le \Delta_d \text{ and } C_{aux}(p_{aux}) \le \left\lceil \frac{UB \cdot (v-1)}{LB \cdot \alpha} \right\rceil + (v-1) \qquad (15)$$

And, $C_{aux}(p_{aux}) \le C_{aux}(p_{DMR})$ (16)

We know that, $C_{aux}(e) = \lceil C(e) \cdot \gamma \rceil$

$$\therefore C_{aux}(p_{aux}) = \lceil C(p_{aux}) \cdot \gamma \rceil,$$

$$\therefore C(p_{aux}) = \frac{1}{\gamma} \cdot C_{aux}(p_{aux})$$

From (15), we have,

$$C(p_{aux}) = \frac{1}{\gamma} \cdot C_{aux}(p_{DMR})$$

From (13), we get,

$$C(p_{aux}) = \frac{1}{\gamma} \cdot [\gamma \cdot C(p_{DMR}) + (v-1)]$$

$[\because C_{aux}(p_{DMR}) \le (v-1) + \gamma \cdot C(p_{DMR})]$

We know that C $(p_{DMR}) \le x_{opt}$, therefore,

$$C(p_{aux}) = \frac{1}{\gamma} \cdot [\gamma \cdot x_{opt} + (v-1)]$$

$$\le x_{opt} + \frac{(v-1)}{\gamma}$$

$$\le x_{opt} + \frac{(v-1)}{\frac{(v-1)}{LB \cdot \alpha}}, \qquad [\because \gamma = \frac{(v-1)}{LB \cdot \alpha}].$$

$$\le x_{opt} + LB \cdot \alpha$$

$$\le x_{opt} + x_{opt} \cdot \alpha, \qquad [\because LB \le x_{opt}]$$

$$C(p_{aux}) \le x_{opt}(1+\alpha) \qquad (17)$$

Equation (17) show that p_{aux} is an (1+α) approximation of DCMCR.

B. Pseudo Code

p_{DMR}	feasible shortest path	Δ_d	delay constraint
Δ_C	cost constraint	C	real value cost
C_{aux}	auxiliary real value cost	D	real value

```
1.  void main( ) {
2.      int c, C[], D[], $C_{aux}$[], V, S,E,Q, k, K,i=0;
3.      int LB[ ], UB[], len[][ ], pred[][];
4.      int α, a, T, $\Delta_d$, $\Delta_C$, γ;
5.      string $p_{DMR}$, $p_{aux}$, e, u, v, b, s,;
6.      find_smallest_constraint( ) {
/* initialize the set S of vertices and minimum
priority queue Q */
7.          S=0;
8.          Q=V(G);
9.          while(Q!=0) {
10.             Q=V-S;
11.             c=Q;}
/* initialize a lower bound and upper bound such that
any s-d path '$p_{DMR}$' with D($p_{DMR}$)<= $\Delta_d$ must
contain at least one edge 'e' with C($p_{DMR}$)>= c */
12.         D($p_{DMR}$) =null; C($p_{DMR}$)=null;
13.         for($e_k$=1;$e_k$<=$p_{DMR}$;$e_k$++){
14.             D($p_{DMR}$) = D($p_{DMR}$) + D($e_k$);
15.             C($p_{DMR}$) = C($p_{DMR}$) + C($e_k$); }
16.         if (D($p_{DMR}$)<= $\Delta_d$ && C($p_{DMR}$)>= c) {
17.             LB[0]=c;
18.             UB[0]=c*v; } }
/*construct an auxiliary graph $G_{aux}$ which is an
instance of graph G to find the optimal value $x_{opt}$ of
DCMCR */
19.     construct_auxgraph( ) {
20.         γ = (v-1)/LB*α;
21.         $C_{aux}$(e) = γ* C(e);}
22.             a = log(v);
23.             α = pow(a,2);
/* perform testing procedure to refine the value of
lower bound LB and upper bound UB */
24.     while (UB[i] >=2(1+ α)LB[i]) {
25.         T=sqrt ((LB[i]*UB[i])/(1+α));
26.         if(D(p)<= $\Delta_d$ && $C_{aux}$(p)<= (v-1)/α) {
27.             String Test(T,α)= "yes"; }
28.         else { Test(T,α)= "no"; }
29.         if(Test(T,α) = = "yes") {
30.             UB[i+1] = T( 1+α);
31.             LB[i+1] = LB[i]; }
32.         else { UB[i+1]= UB[i];
33.             LB[i+1] = T; } }
34.         i++;
35.         UB = UB[i];
36.         LB = LB[i];
37.     mcpv2.2( ) {
/*initialize (K-1) dimensional array to store length
and predecessor of edges*/
38.         K=3;
39.         for(k=2;k<=K;k++) {
40.             len[v,$c_k$] = ∞;
41.             pred[v,$c_k$] = null;
42.             len[s,$c_k$] = 0; }
```

```
        /* update array values */
43.             for(k=2;k<=K;k++) {
44.                 for( (u,v)=1;(u,v)<=E;(u,v)++) {
45.                     if(len[v,c_k] > len[u,b_k] +w_1(u,v) ) {
46.                         len[v,c_k] = len[u,b_k] +w_1(u,v);
47.                         pred[v,c_k] = u;  }}}
        /* find the s-d path p_aux such that D(p_aux) < = Δ_d and
        C_aux(p_aux) <=c */
48.         if(len[d,c] < = Δ_d ) {
49.             p_DMR =p_aux;
50.             printf("Feasible path is returned %s",p_DMR};
51.             break ( ); }
52.         else {
53.             printf("No feasible path found"};
54.             break ( ); }}}
```

Fig.2. DCMCR Algorithm

In lines 6 to 23, we are finding the smallest constraints 'c' i.e. cost of each edge C(e) by applying shortest path algorithm. We compare the delay of the path $D(p_{DMR})$ with delay constraint Δ_d and cost of the path $C(p_{DMR})$ with smallest constraint 'c'. If any of the edge cost is greater than 'c' then it invokes further process of our algorithm by constructing auxiliary graph.

Both upper and lower bounds are defined and tested whether or not UB is greater than $2(1+\alpha)$LB. Then we apply MCPv2.2 to compute the feasible path p_{aux} for auxiliary graph G_{aux}. Then the (K-1) dimensional array is initialized to store length and predecessor of each edge 'e' as shown in lines between 24 and 47.

Finally, we compare the computed length of auxiliary path with delay constraint Δ_d , if the length of the path p_{aux} is less than or equal to delay constraint, the algorithm returns the feasible path, or otherwise as explained in lines from 48 to 54. We represent DCMCR Algorithm in figure 2.

IV. Results

We applied few Internet topologies [1] to verify the suitability of algorithms and arbitrarily generated topologies. We verified our analytical model with simulation for feasibility with 60 to 120 nodes. Edge weights are uniformly selected between the range 1 and 10 as in [4] [18]. We selected the same source-destination pair in all topologies for comparison and from our analysis it can be seen that our algorithms to perform similarly on various edge weights. We selected a wide range of W, that is, a small value of W to large value of W and we performed the experiments for three constraints. We selected W such that MCPv1 is infeasible and large value of weights so that MCPv1 is feasible.

For each topology, we selected 8-10 test cases and for each test case, we arbitrarily generated a source-destination pair and used this pair for all tested algorithms. Our test case consists of topology, approximation factor and the corresponding source-destination pair. We used a small value of W and a large value of W to test the algorithms for this node pair. These values of W may change where the node pair changes. The example topology is shown in figure 3.

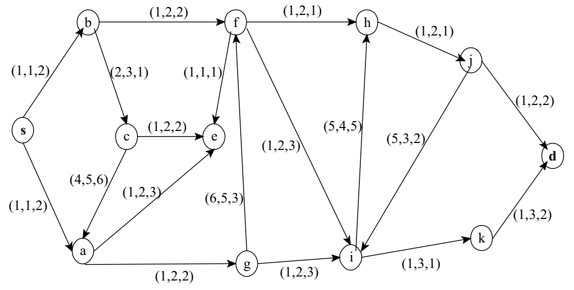

Fig.3. Example Topology

Our algorithm MCPv3 is to find an source-destination path p_{opt} such that K^{th} weight of path p_{opt}, $w_k(p_{opt}) \le x_{opt}.W$, where x_{opt} is an optimal value of MCPv3 and W is constraint bound. We chose our constraint bound values W = (10, 22, 20) and K = 3. We explain how feasible paths are chosen in our example topology. Consider the paths $p(s \to b \to f \to i \to k \to d)$ and $p(s \to b \to c \to a \to g \to i \to h \to j \to d)$ in figure 3.

$$w_{aux}(e) = \max_{1 \le k \le K}\left(\frac{w_k(e)}{W}\right)$$

We calculate K^{th} weights of path p (s $\to$ b $\to$ f $\to$ I $\to$ k $\to$d),

$$w_k(p(s \to b \to f \to i \to k \to d)) = \sum w_k(e)$$

$$= w_k(s,b)+w_k(b,f)+w_k(f,i)+w_k(i,k)+w_k(k,d)$$

$$= (1,1,2)+(1,2,2)+(1,2,3)+(1,3,1)+(1,3,2)$$

$$w_k(p(s \to b \to f \to i \to k \to d)) =(5,11,10)$$

From the above K^{th} weight of path, we compute the auxiliary path weight $w_{aux}(p)$,

$$\therefore w_{aux}(p(s \to b \to f \to i \to k \to d))$$

$$= \max_{1 \le k \le K}\left(\frac{w_k(s \to b \to f \to i \to k \to d)}{W}\right)$$

$$= \max_{1 \le k \le K}\left(\frac{w_1(\text{sbfikd})}{W_1}, \frac{w_2(\text{sbfikd})}{W_2}, \frac{w_3(\text{sbfikd})}{W_3}\right)$$

$$w_{aux}(p(s \to b \to f \to i \to k \to d)) = \max_{1 \le k \le K}\left(\frac{5}{10}, \frac{11}{22}, \frac{10}{20}\right) = 0.5.$$

Then, we find the auxiliary path weight for path $p(s \to b \to c \to a \to g \to i \to h \to j \to d)$ using the same method,

$$w_{aux}(p(\text{sbcagihjd})) = \max_{1 \le k \le K}\left(\frac{w_k(\text{sbcagihjd})}{W}\right)$$

$w_k(p(\text{sbcagihjd})) = w_k(s,b) + w_k(b,c) + w_k(c,a) + w_k(a,g) + w_k(g,i) + w_k(i,h) + w_k(h,j) + w_k(j,d)$
=(1,1,2) + (2,3,1)+ (4,5,6)+ (1,2,2)+ (1,2,3) + (5,4,5)+ (1,2,1)+ (1,2,2) = (16,21,22)

$$\therefore \quad w_{aux}(p(\text{sbcagihjd})) = \max_{1 \le k \le K}\left(\frac{16}{10}, \frac{21}{22}, \frac{22}{20}\right) = 1.6$$

We apply shortest path algorithm to compute the shortest path after calculating the auxiliary path weight.The path $p(s \to b \to f \to i \to k \to d)$ is found to be shortest, because its path weight is 0.5 which is less than the another one path $p(s \to b \to c \to a \to g \to i \to h \to j \to d)$. Then we compare auxiliary path weight whether $w_{aux}(p(s \to b \to f \to i \to k \to d)) = 0$ or not. In our example $w_{aux}(p(s \to b \to f \to i \to k \to d)) = 0.5$, so we initialize the lower and upper bounds according to our MCPv3 algorithm.

$LB = w_{aux}(p(s \to b \to f \to i \to k \to d))/K$ and $UB = w_{aux}(p(s \to b \to f \to i \to k \to d))/2$.

$\therefore LB = 0.5/3 = 0.17$

$UB = 0.5/2 = 0.25$

We know that the optimal value x_{opt} of MCPv3 should be between LB and twice that of upper bound.

$LB \leq x_{opt} \leq 2UB$

$0.17 \leq x_{opt} \leq 0.5$

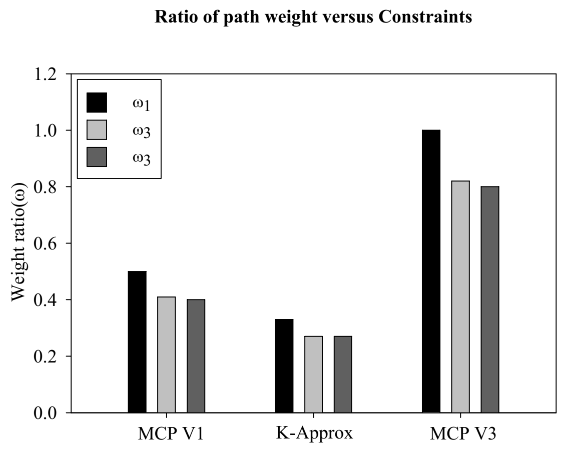

Fig.4. Ratio of path weight versus Constraints for Arbitrary Topology

We chose in our example $x_{opt} = 0.5$. We know that in MCPv3 algorithm a s-d path p_{opt} should satisfy the condition $w_k(p_{opt}) \leq x_{opt}.W$.

$\therefore w_k(p_{opt}) \leq 0.5(10,22,20)$

$w_k(p_{opt}) \leq (5,11,10)$

Edge weights of path $p(s \to b \to f \to i \to k \to d)$ and $w_k(p(s \to b \to f \to i \to k \to d)) = (5,11,10)$ found to be feasible, since $w_k(p_{opt}) \leq (5,11,10)$. Similarly, $w_k(sbcagihjd) = (16, 21, 22)$ and is not satisfying the above condition and found infeasible.

MCPv3 algorithm returns 3 feasible paths out of 16 paths from source 's' to destination 'd' in our example topology. For example paths $p(s \to b \to f \to h \to j \to d = 5, 9, 8)$, $p(s \to b \to f \to i \to k \to d = 5, 10, 9)$ and $p(s \to a \to g \to i \to k \to d = 5, 11, 10)$ are satisfying our constraint bound (5, 11, 10) in figure 3. Our algorithm returns the shortest feasible path $p(s \to b \to f \to h \to j \to d = 5, 9, 8)$ among the three feasible paths and remaining paths are not selected since they have not satisfied the constraint bound. For example, path $p(s \to a \to g \to f \to h \to j \to d = 11, 14, 11)$ and $p(s \to b \to c \to a \to g \to i \to k \to d = 11, 19, 17)$. Our results are shown in figure 4 to figure 6 for different topologies.

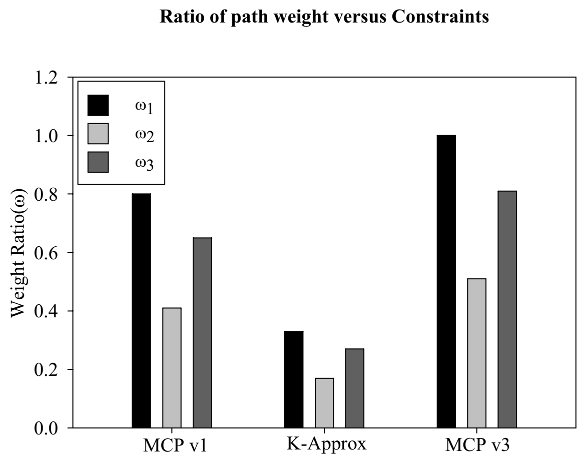

Fig.5. Ratio of path weight versus Constraints for ARPANET Topology

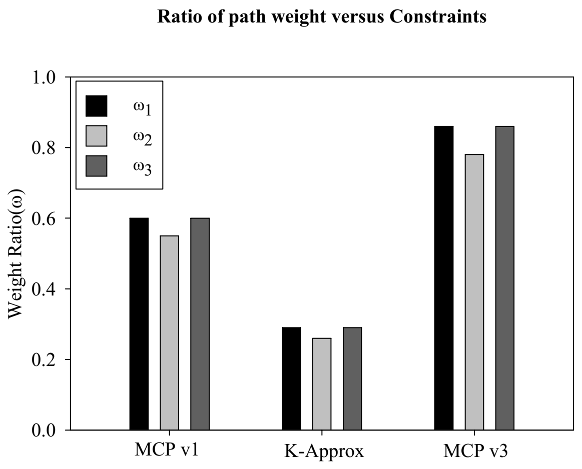

Fig.6. Ratio of path weight versus Constraints for ANSNET Topology

DCMCR algorithm is applied to the same arbitrary topology to find the shortest feasible path, p_{DMR} such that $C(p_{DMR}) = \sum_{e \in p_{DMR}} C(e)$ is minimized subject to (K-1) constraints and the delay of the path $D(p_{DMR}) \leq \Delta_d$, where $\Delta_d = W_1$ is end-to-end delay constraint. We chose $\Delta_d = 10$, W= (10, 22, 20) and K = 3. DCMCR identifies 5 paths are feasible with the following weights {(5,9,8), (5,10,9), (10,13,15), (5,11,10), (10,13,15)}.

To find a feasible path we need to calculate the delay D(e) and cost C(e) for each edge along the paths. $D(e) = w_1(e)$ and $C(e) = \max_{2 \leq k \leq K} \left(\frac{w_k(e)}{W} \right)$.

For a path p(sbfhjd), delay of path p(sbfhjd) is calculated by adding the delay of each edges along the path.

D(p(sbfhjd)) = D(s,b)+D(b,f)+D(f,h)+D(h,j)+D(j,d)

$= w_1(s,b)+ w_1(b,f)+ w_1(f,h)+ w_1(h,j)+ w_1(j,d)$
$= 1+1+1+1+1$

D(p(sbfhjd)) = 5.

Also the cost of the path p(sbfhjd) is calculated by adding the cost of each edges along the path.

C(p(sbfhjd)) = C(s,b)+C(b,f)+C(f,h)+C(h,j)+C(j,d)

$$C(s,b)= \max_{2\le k\le K}\left(\frac{w_2(s,b)}{W_2},\frac{w_3(s,b)}{W_3}\right)$$

$$= \max_{2\le k\le K}\left(\frac{1}{22},\frac{2}{20}\right) = 0.1$$

$$C(b,f)= \max_{2\le k\le K}\left(\frac{w_2(b,f)}{W_2},\frac{w_3(b,f)}{W_3}\right)$$

$$= \max_{2\le k\le K}\left(\frac{2}{22},\frac{2}{20}\right) = 0.1$$

$$C(f,h)= \max_{2\le k\le K}\left(\frac{w_2(f,h)}{W_2},\frac{w_3(f,h)}{W_3}\right)$$

$$= \max_{2\le k\le K}\left(\frac{2}{22},\frac{1}{20}\right) = 0.09$$

$$C(h,j)= \max_{2\le k\le K}\left(\frac{w_2(h,j)}{W_2},\frac{w_3(h,j)}{W_3}\right) = \max_{2\le k\le K}\left(\frac{2}{22},\frac{1}{20}\right) = 0.09$$

$$C(j,d)= \max_{2\le k\le K}\left(\frac{w_2(j,d)}{W_2},\frac{w_3(j,d)}{W_3}\right) = \max_{2\le k\le K}\left(\frac{2}{22},\frac{2}{20}\right) = 0.1$$

$\therefore$ C(p(sbfhjd)) = 0.1+0.1+0.09+0.09+0.1 = 0.48.

For a path p(sbfihjd) , delay of path is,

D(p(sbfihjd)) = D(s,b)+D(b,f)+D(f,i)+D(i,h)+D(h,j)+D(j,d)
= 1+1+1+5+5 =10

And cost of path p(sbfihjd) is,

C(p(sbfihjd)) = C(s,b)+C(b,f)+C(f,i)+C(I,h)+C(h,j)+C(j,d)
= 0.1+0.1+0.15+0.25+0.09+0.1 = 0.79

For a path p(sbcagfikd) , delay of path is, D(p(sbcagfikd)) = 17 and cost of path p(sbfihjd) is C(p(sbfihjd)) = 1.3 .

Among these paths { p(sbfhjd), p(sbfihjd) ,p(sbcagfikd) }, the delay and cost of these paths are {5,10,17} and {0.48,0.79,1.3} respectively. Our DCMCR algorithm returns the path p(sbcagfikd) as infeasible path, because the delay of this path (=17) is greater than the given delay bound Δ_d (=10). DCMCR algorithm returns the other two paths p(sbfhjd) and p(sbfihjd) as feasible paths, because delay of these paths are satisfy the delay bound Δ_d. Finally DCMCR algorithm returns the path p(sbfhjd) as shortest feasible path, since the cost of this path p(sbfhjd) is (=0.48) minimum when compared to the cost of other feasible path p(sbfihjd) (=0.79). We represented these five feasible paths on a 3D plane. The weights w_1, w_2, w_3 lying on the 'plane' denotes feasible path region for a given source-destination pair as illustrated in figure 7.

We found five paths which satisfies this condition namely, p($s\rightarrow b\rightarrow f\rightarrow h\rightarrow j\rightarrow d$), p($s\rightarrow b\rightarrow f\rightarrow i\rightarrow k\rightarrow d$), p ($s\rightarrow b\rightarrow f\rightarrow i\rightarrow h\rightarrow j\rightarrow d$), p($s\rightarrow a\rightarrow g\rightarrow i\rightarrow k\rightarrow d$) and p ($s\rightarrow a\rightarrow g\rightarrow i\rightarrow h\rightarrow j\rightarrow d$) whose delay and cost are (5,5,10,5,10) and (0.48,0.63,0.79,0.63,0.79) respectively. Among these 5 feasible paths DCMCR returns path p($s\rightarrow b\rightarrow f\rightarrow h\rightarrow j\rightarrow d$) as shortest feasible path since it satisfies both delay and cost constraint. Other path such as p ($s\rightarrow a\rightarrow g\rightarrow f\rightarrow h\rightarrow j\rightarrow d$) = 11 not returned as Δ_d (=11) was away from the bound. Our results are presented in fig 8.

Optimal Weights for DCMCR

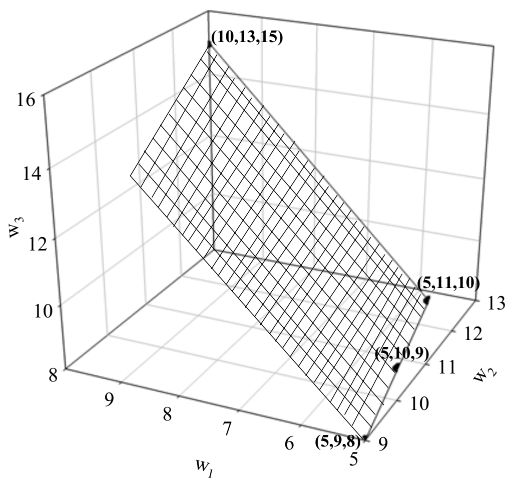

Fig.7 optimal weights on 3D plane

We present our simulation results conducted in a discrete event self written C++ simulator [6] on execution time of our algorithm and influences of W in figure 9 and figure 10. MCPv3 is faster than DCMCR for a particular approximation factor and W. It is observed that the running time and approximation factor (α) are negatively correlated. We noticed that MCPv3 is almost independent of W. Execution time is increasing with size of the network in all the algorithms as shown in figure 9. We applied number of constraints are 3 and α = 0.5. We applied the range of constraint bounds between 10 and 22.

Ratio of path versus constraints

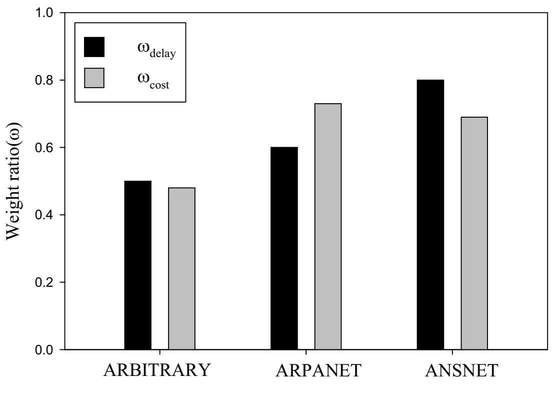

Fig.8. Ratio of path weight versus Constraints for DCMCR Algorithm

Most algorithms are fast in terms of execution time, but do not guarantee quality of solution. Among all algorithms

MCPv1 is fastest as it requires no bound conditions and followed by K-approximation, where it needs only one shortest path computation. Running time of MCPv1 is $O(v(H/\alpha)^2)$ time, where H (i.e. hop count) is small. MCPv3 and DCMCR take more time as the value of α chosen becomes smaller, which implies better accuracy of results.

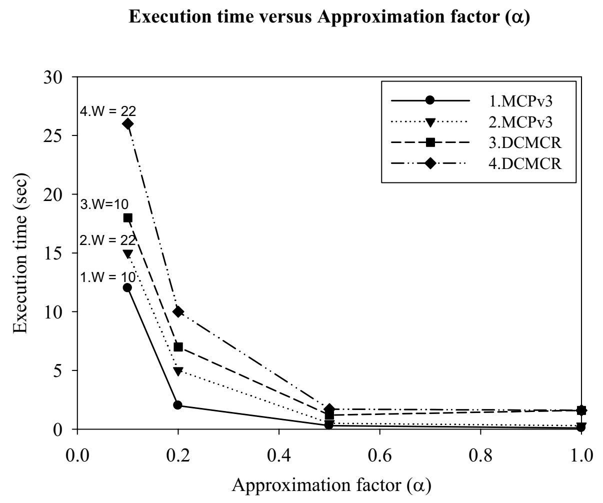

Fig.9. Execution time versus Approximation factor (α)

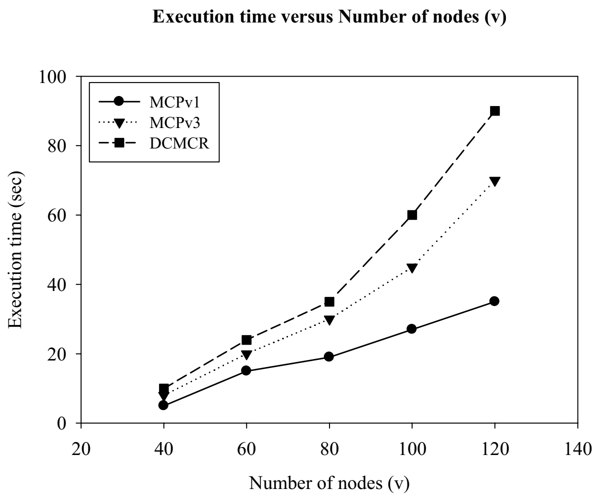

Fig.10. Execution time versus Number of nodes (v)

V. Conclusion

Optimal path selection subject to multiple constraints can only be addressed through heuristics and approximation algorithm. In this paper we have presented Multi Additive Constraint path problem with two or more constraints. We presented an approximation algorithm which uses all constraints are unified to form a single auxiliary edge weight to compute the feasible shortest path. This algorithm is implemented in current network scenario where we have dynamic network topology and requirement of QoS constraints are getting changed for variety of applications. Our simulation results have shown that the MCPv3 is competent with other algorithm in bringing the feasible path in polynomial time.

DCMCR is other algorithm where we coerce one of the QoS constraints and remaining constraints are approximated. If there exists a feasible shortest source-destination path whose first weight is bounded by the first constraint and every other path weight is bounded by (1-α) times the corresponding weight, our algorithm returns a feasible path. We executed on well known directed network graphs and found that our algorithms were competent and return a solution in linear time, if exists.

References

[1] R. Anderson, F. Chung, A. Sen and G. Xue, "On disjoint path pairs with wavelength Continuity in WDM networks", in Proc. 23rd Annual Joint Conference of the IEEE Computer and Communication Societies, IEEE INFOCOM'04, vol. 1, Hong Kong, PR China, March 2004, pp. 524 – 535.

[2] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in Proc. of IEEE International Conference on Communication. IEEE ICC'98, Atlanta, Georgia, USA, vol. 2, Atlanta, Georgia, USA, June 1998, pp. 874–879.

[3] F. Ergun, R. Sinha, and L. Zhang, "An improved FPTAS for restricted shortest path," Information Process Letters, vol. 83, no. 5, September 2002, pp. 287–291.

[4] A. Goel, K. G. Ramakrishnan, D. Kataria, and D. Logothetis, "Efficient computation of delay-sensitive routes from one source to all destinations," In Proc. 20th Annual Joint Conference of the IEEE Computer and Communication Societies, IEEE INFOCOM'01, vol. 1, Anchorage, Alaska, USA, no. x, April 2001, pp. 854–858.

[5] R. Guerin and A. Orda, "QoS routing in networks with inaccurate information: Theory and algorithms," IEEE/ACM Transaction on Networks, vol. 7, no. 3, June 1999, pp. 350–364.

[6] Kevin Fall, Kannan Varathan, "The ns Manuals, The Vint Project", University of California, Berkeley, USA, March 2007, pp. 28-130.

[7] T. Korkmaz and M. Krunz, "A randomized algorithm for finding a path subject to multiple QoS requirements," International Journal of Computer and Telecommunications Networking, vol. 36, no. 2/3, July, 2005, pp. 251-268.

[8] Larry L. Peterson, Bruce S. Davie Book, "Computer Networks: A System Approach, 4/e" Morgan Kaufmann Publications, San Francisco, CA, USA, 2007.

[9] W. Liu, W. Lou and Y.Fang, "An efficient quality of service routing algorithm for delay-sensitive application", Computer Networks, vol. 47, no. 1, January 2005, pp, 87-104.

[10] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," Operation Research Letters, vol. 28, no. 5, June 2001, pp. 213–219.

[11] A. Orda and A. Sprintson, "Efficient algorithms for computing disjoint QoS paths," in Proc. 23rd Annual Joint Conference of the IEEE Computer and Communication Societies, IEEE INFOCOM'04, vol.1, no.x, Hong Kong, PR China, March 2004, pp. 727–738.

[12] A. Orda and A. Sprintson, "Pre computation schemes for QoS routing," IEEE/ACM Transaction on Networks, vol. 11, no. 4, August 2003, pp. 578–591.

[13] A. Orda, "Routing with end-to-end QoS guarantees in broadband networks," IEEE/ACM Transactions on Networks, vol. 7, no. 3, June 1999, pp. 365–374.

[14] P. Van Mieghem and F. A. Kuipers, "Concepts of exact QoS routing algorithms," IEEE/ACM Transaction on Networks, vol. 12, no. 5, October 2004, pp. 851–864.

[15] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," IEEE Journal on Selected Areas in Communication, vol. 14, no. 7, Sep 1996, pp. 1228–1234.

[16] Y. Xiao "QoS routing in communication networks: Approximation algorithms based on the primal simplex method of linear programming," IEEE Transaction on Computers, vol. 55, no. 7, July 2006, pp. 815–829.

[17] G. Xue, "Minimum cost QoS multicast and unicast routing in communication networks," IEEE Transactions on Communications, vol. 51, no. 5, May 2003, pp. 817–824.

[18] X. Yuan, “Heuristic algorithms for multi constrained quality-of-service routing,” IEEE/ACM Transaction on Networks, vol. 10, no. 2, April 2002, pp. 244–256.

P.S.Prakash received B.E (EEE) and M.E (Electrical Machines) degrees from P.S.G College of Technology, Coimbatore, India and also received M.E degree in Computer Science and Engineering from Bharathiar University, India. His research interests include QoS Scheduling, Routing, Network Security and Management. At present, serving as Associate .Professor at Sri Ramakrishna Engg. College, Coimbatore, India.

S.Selvan received the B.E. degree in electronics and communication engineering and the M.E. degree in communication systems from the University of Madras, Chennai, India, in 1977 and 1979, respectively, and the Ph.D. degree in computer science and engineering from the Madurai Kamaraj University, Madurai, India, in 2001. He has 30 years of teaching and research experience. He is currently working as Principal and Professor of computer science and engineering at St. Peter’s engineering college, Chennai, India. He is a senior member of IEEE and fellow of IE(I) and IETE. He has published more than 140 papers in international and national journals and conference proceedings. His areas of research include computer networks, data mining, soft computing, signal processing, image processing and network security.