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 pre-defined 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 \( \Delta_d \) (delay constraint) and cost is not more than \((1+\alpha)\) times the cost of the least-cost delay constrained path, provided that there is a source-destination path whose delay is at most \( \Delta_d \). If this condition is not satisfied then all these algorithms will terminate which means that the problem is infeasible.


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 \(w=(w_1,\ldots,w_k)\) is an edge weight vectors so that \(w_i(e)\geq0\) is \(k^{th}\) weight of edge ‘e’, \(\forall e\in E, 1\leq k\leq K\). For a path ‘p’ in G, \(k^{th}\) weight of path ‘p’ denoted by \(w_k(p)\) is the sum of \(k^{th}\) weights over the edges in p, \(w_k(p)=\sum_{e\in p} w_k(e)\). 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, Tamil Nadu, 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 \( w_d(p) \) of \( W_k \) 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,W,w): A graph \( G=(V,E) \) with \( K \) edge positive integer valued edge weights \( w_k(e),1 \leq k \leq K, \) associated with each edge \( e \in E; \) a positive constant \( \Delta_k \) and a source-destination pair s-d. Our objective is to find a s-d path ‘p’ such that \( w_k(p) \leq \Delta_k,1 \leq k \leq K. \)

MCPv3 (G,s,d,K,W,w): A graph \( G=(V,E) \) with \( K \) edge positive integer valued edge weights \( w_k(e),1 \leq k \leq K, \) 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’ such that \( w_k(p) \leq \Delta_k,1 \leq k \leq K. \)

We define \( x_{opt} \) is the optimal value of MCPv3 and call \( p_{opt} \) an optimal path of MCPv3. We choose \( x_{opt} \leq 1 \) if and only if MCPv1 is feasible. When \( x_{opt} \leq 1 \) 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 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 distributed routing protocols [8].

\[
W_{aux}(p_{aux})/K \text{ is the lower bound for } x_{opt} \text{ and } W_{aux}(p_{aux})/2 \text{ is an upper bound for } x_{opt}. \]

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 \( (1+\gamma) \) 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 \( (1+\gamma) \) approximation to MCPv3. Let \( LB[i] \) and \( UB[i] \) denote the lower and upper bound as explained above.

\[
LB[i] \leq x_{opt} \leq UB[i] \quad (1)
\]

is true for \( i=0. \)

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

\[
w_{aux}(p_{opt}) = \frac{w_k(p_{opt}) \cdot (v-1)}{L.B.W.\alpha} + (v-1) \quad (2)
\]

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

\[
w_{aux}(p_{opt}) = \frac{x_{opt} \cdot (v-1)}{L.B.\alpha} + (v-1) \quad (3)
\]

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

\[
w_{aux}(p_{opt}) \leq \frac{2UB \cdot (v-1)}{L.B.\alpha} + (v-1) \quad (4)
\]

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,

\[
\max_{1 \leq k \leq K} w_{aux}(p_{opt}) \leq \frac{x_{opt} \cdot (v-1)}{L.B.\alpha} + (v-1) \quad (5)
\]

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 \( \Delta_K \) is the smallest integer less than or equal to \( \frac{2UB \cdot (v-1)}{L.B.\alpha} + (v-1) \) 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,

\[
\max_{1 \leq k \leq K} w_{aux}(p_{aux}) \leq \max_{1 \leq k \leq K} w_{aux}(p_{opt}) \quad (6)
\]

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

\[
\max_{1 \leq k \leq K} w_{aux}(p_{aux}) \leq x_{opt} \cdot \frac{v-1}{L.B.\alpha} + (v-1) \quad (7)
\]

We also have,

\[
\max_{1 \leq k \leq K} w_{aux}(p_{aux}) = \sum_{e \in p_{aux}} w_{aux}(e) \geq \sum_{e \in p_{opt}} w_k(e) \cdot \frac{(v-1)}{L.B.W.\alpha} = \frac{w_k(p_{aux}) \cdot (v-1)}{L.B.W.\alpha} \quad \text{for all } 1 \leq k \leq K \quad (8)
\]

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

\[
\frac{w_k(p_{opt}) \cdot (v-1)}{L.B.W.\alpha} \leq x_{opt} \cdot \frac{(v-1)}{L.B.\alpha} + (v-1) \quad \text{for all } 1 \leq k \leq K \quad (9)
\]
compute_auxweight( )

/* compute auxiliary edge weight */

2.   int k, K, w aux[], wk[], W, c, S, V, E, Q, T, \( \Delta_K \);
3.   int i=0, \( \alpha \), LB[], len[], pred[];  
4.   string popt, paux, e, u, v, b, s;
5.   /* compute auxiliary edge weight */
6.   compute_auxweight( ){
7.     K=3;
8.     for(e=1;e<=E;e++) {
9.       for(k=1;k<=K;k++) {
10.      \( w_{aux}(e) = \max(w_k(e)/W) \); } }  
11. /* initialize the set S of vertices and minimum priority queue Q */
12.     S=0;
13.     Q=V(G);
14.     while(Q!=0) {
15.       Q=V-S;
16.       u=Q;
17.       S=S+u;
18.       \( \text{if}(w_{aux}(p_{aux}) \leq \max(w_{k}(p_{opt})/W)) \) {
19.         p_{aux} = p_{aux};
20.         printf("feasible path is returned \%s\",p_{aux});
21.         break(); }
22.   }
23.   /* update lower bound and upper bound values using testing procedure */
24.   while(UB[i] >= 2*LB[i]) {
25.     T=sqrt(LB*UB);
26.     \( \alpha = 1; \)
27.     \( \text{if}(w_{aux}(p_{aux}) = ((v-1)/\alpha)+(v-1)) \) {
28.       String Test(T,\( \alpha \))="yes"
29.     } else 
30.       else 
31.         \( \text{Test}(T,\alpha) = \text{"no"} \)
32.     \( \text{if}(Test(T,\alpha) = \text{"yes"}) \) {
33.       LB[i]=T;
34.     } else  
35.       UB[i]=T;  }
36.     i++;
37.     LB=UB[i];
38.     UB=LB;
39.   /* construct an auxiliary graph \( G_{aux} \) which is an instance of graph \( G \) */
40.   construct_auxgraph() {  
41.     \( \gamma = (v-1)/LB[i]*\alpha*W; \)
42.     w_{aux}(e) = \( \gamma * w_{aux}(e)+1; \)
43.   }
44.   mcpv2.1() 
45.   /*initialize \( K-1 \) dimensional array to store length and predecessor of edges*/
46.   for(k=1;k<=K;k++) {
47.     len[v,ck] = \infty;
48.     pred[v,ck] = null;
49.     len[s,ck] = 0;  
50.   /* update array values */
51.   for(k=1;k<=K;k++) {
52.     for(u,v) E(u,v) \leq E(u,v)++) {
53.       if(len[v,ck] > len[u,ck] +w_{aux}(e,v)) {
54.         len[v,ck] = len[u,ck] +w_{aux}(e,v);
55.         pred[v,ck] = w_{aux};  } }  
56.   /* find the s-d path \( P_{aux} \) such that \( w_{aux}(p_{aux}) \leq \Delta_K \)
57.   if(len[d,ck] \leq \Delta_K) {
58.     p_{opt} = p_{aux};
59.     printf("Feasible path is returned \%s\",p_{opt});
60.     break(); }
61.
62.
55. else {
  printf("No feasible path found");
  break ();}

Fig.1. MCPv3 Algorithm

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

Then we compare the auxiliary weight of the path is less than 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.

To show the 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 \( w_{aux}(e) \) is changed to \( w_{aux}(e) \), such that \( w_{aux}(e) = \left( w(e) + 1 \right) \) 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 \( w=(w_1,…,w_k) \) is an edge weight vector so that \( w_{aux}(e) > 0 \) 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 weight over the edges on \( p_{DMR} \), i.e. \( w_k(p_{DMR}) = \sum_{c \in p_{DMR}} w_k(c) \).

In our algorithm 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 \( k = 2, ..., K \). Then MCPv2.2 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 51 to 57.

We construct an auxiliary edge-weighted directed graph \( G_{aux} \) having an auxiliary edge 'e' and \( w_{aux}(e) \) is an edge weight vector so that \( w_{aux}(e) > 0 \) is the Kth weight of edge 'e', for all \( e \in E \), \( 1 \leq k \leq K \).

Our objective is to find a path 'p_{DMR}' for a given s-d in G such that \( C(p_{DMR}) = \sum_{c \in p_{DMR}} C(c) \) is minimized subject to the constraint \( D(p_{DMR}) = \sum_{c \in p_{DMR}} D(c) \leq \Delta_d \). 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 DCMC(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 \( k = 2, ..., K \). Then MCPv2.2 is applied to find a s-d path 'p' such that \( w_1(p) \leq \Delta_d \) and \( w_k(p) \leq \Delta_c \). In MCPv2.2, the edge weights \( w_2,...,w_k \) 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 \( C(e) \leq v \) for all \( e \in p_{DMR} \). Secondly any s-d path 'p', \( D(p) \leq \Delta_d \) must contain at least one edge '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.

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

\[
\Delta_d \leq x_{opt}\leq c.v
\]

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

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,

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

\[
\leq \sum_{c \in p_{DMR}} [C(c) + (v-1)] \quad \because p_{DMR} \text{has at most (v-1) edges}.
\]

\[
\leq (v-1) + \gamma \sum_{c \in p_{DMR}} C(c)
\]

\[
\leq (v-1) + \gamma -D(p_{DMR}) \quad (13)
\]

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

\[
\Delta_d \leq (v-1) + \gamma -D(p_{DMR})
\]
Substitute the value of $x_{opt}$ from (11),

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

(14)

The value of $C_{aux}(p_{DMR})$ is scaled down to the nearest lower integer. From (14), the path $p_{DMR}$ is feasible solution of $\text{MCPv2.2}$. 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 $(1+\gamma)$-approximation of DCMCR. Since $p_{aux}$ is computed by the algorithm, we have,

$$D(p_{aux}) \leq \Delta_d$$

(15)

And,

$$C_{aux}(p_{aux}) \leq C_{aux}(p_{DMR})$$

(16)

We know that,

$$C_{aux}(e) = C(e) \cdot \gamma$$

Therefore, we have,

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

(17)

From (15), we have,

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

From (13), we get,

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

(18)

From (13), we get,

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

(19)

From (11), we get,

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

(20)

And,

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

(21)

From (20), we get,

$$C_{aux}(p_{DMR}) \leq C_{aux}(p_{aux}) + \frac{UB \cdot (v - 1)}{LB}$$

(22)

From (11), we get,

$$C_{aux}(p_{DMR}) \leq C_{aux}(p_{aux}) + \frac{UB \cdot (v - 1)}{LB}$$

(23)

We know that $C_{DMR} = C_{aux}(p_{DMR})$, therefore,

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

(24)

Equation (17) shows that $p_{aux}$ is an $(1+\alpha)$-approximation of DCMCR.

B. **Pseudo Code**

```plaintext
p_{DMR} \quad \text{feasible shortest path} \quad \Delta_d \quad \text{delay constraint}
\Delta_C \quad \text{constraint cost} \quad C \quad \text{real value cost}
C_{aux} \quad \text{auxiliary real value cost} \quad D \quad \text{real value}

1. void main() {
   2.   int c, C[], D[], Caux[], V, S, E, Q, k, i = 0;
   3.   int LB[], UB[], len[], pred[];
   4.   int a, a', T, \Delta_d, \Delta_C, \gamma;
   5.   string p_{DMR}, p_{aux}, e, u, v, b, s;
   6.   find_smallest_constraint() {
      7.      /* initialize the set S of vertices and minimum priority queue Q */
      8.      S = 0;
      9.      Q = V(G);
     10.     while(Q != 0) {
     11.         Q = V - S;
     12.         c = Q;
     13.      }
     14.      /* construct an auxiliary graph G_{aux} which is an instance of graph G to find the optimal value x_{opt} of DCMCR */
     15.      construct_auxgraph() {
     16.         \gamma = (v-1)/LB \cdot \alpha;
     17.         C_{aux}(e) = \gamma \cdot C(e);
     18.         a = log(v);
     19.         \alpha = pow(a,2);
     20.      }
     21.      /* perform testing procedure to refine the value of lower bound LB and upper bound UB */
     22.      while(UB[i] \geq 2(1+\gamma)LB[i]) {
     23.         T = sqrt((LB[i]*UB[i])/(1+\gamma));
     24.         if(D(p) \leq \Delta_d \&\& C_{aux}(p) \leq (v-1)/LB) {
     25.             String Test(T, \alpha) = "yes";
     26.         } else {
     27.             Test(T, \alpha) = "no";
     28.         }
     29.     }
     30.      }
     31.      /* construct an auxiliary graph G_{aux} which is an instance of graph G to find the optimal value x_{opt} of DCMCR */
     32.      void mcpv2.2() {
     33.         /* initialize the set S of vertices and minimum priority queue Q */
     34.         S = 0;
     35.         Q = V(G);
     36.         while(Q != 0) {
     37.             Q = V - S;
     38.             c = Q;
     39.         }
     40.      }
     41.      /* construct an auxiliary graph G_{aux} which is an instance of graph G to find the optimal value x_{opt} of DCMCR */
     42.      }*/
```
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(pDMR) with delay constraint \( \Delta_d \) and cost of the path C(pDMR) 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+o)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 path length and predecessor of each edge ‘c’ as shown in lines between 24 and 47.

Finally, we compare the computed length of auxiliary path with delay constraint \( \Delta_d \), 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 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 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 \( p_{opt} \), \( W_k(p_{opt}) \), \( x_{opt} \), \( W \), 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 \( p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d) \) and \( p(s \rightarrow b \rightarrow c \rightarrow a \rightarrow g \rightarrow i \rightarrow h \rightarrow j \rightarrow d) \) in figure 3.

\[
\begin{align*}
W_{aux}(e) &= \max_{1 \leq k < K} \frac{w_k(e)}{W} \\
W_{aux}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) &= \max_{1 \leq k < K} \left( \frac{w_1(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)}{W_1} \right) \\
W_{aux}(p(s \rightarrow b \rightarrow c \rightarrow a \rightarrow g \rightarrow i \rightarrow h \rightarrow j \rightarrow d)) &= \max_{1 \leq k < K} \left( \frac{5}{10}, \frac{11}{22}, \frac{10}{20} \right) = 0.5.
\end{align*}
\]

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

\[
\begin{align*}
W_{aux}(p(sbcagijh)) &= \max_{1 \leq k < K} \frac{w_k(sbcagijh)}{W} \\
W_{aux}(p(sbcagijh)) &= \max_{1 \leq k < K} \left( \frac{16}{21} \frac{22}{22} \frac{22}{20} \right) = 1.6
\end{align*}
\]
We apply shortest path algorithm to compute the shortest path after calculating the auxiliary path weight. The path \( p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d) \) is found to be shortest, because its path weight is 0.5 which is less than the another one path \( p(s \rightarrow b \rightarrow c \rightarrow a \rightarrow g \rightarrow i \rightarrow h \rightarrow j \rightarrow d) \). Then we compare auxiliary path weight whether \( w_{aux}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) = 0 \) or not. In our example \( w_{aux}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) = 0.5 \), so we initialize the lower and upper bounds according to our MCPv3 algorithm.

\[
LB = w_{aux}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d))/K \quad \text{and} \quad UB = w_{aux}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d))/2.
\]

\[
LB = 0.5/3 = 0.17 \quad \text{and} \quad UB = 0.5/2 = 0.25
\]

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

\[
0.17 \leq x_{opt} \leq 0.25
\]

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

\[
w_k(p_{opt}) \leq x_{opt} \cdot W.
\]

Edge weights of path \( p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d) \) and \( w_k(p_{opt})(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) = 5, 10, 9 \) and \( w_k(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) = 5, 11, 10 \) are found to be feasible, since \( w_k(p_{opt}) \leq (5, 10, 9) \) and \( w_k(p_{opt}) \leq (5, 11, 10) \). 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 \( p(s \rightarrow b \rightarrow f \rightarrow h \rightarrow j \rightarrow d = 5, 9, 8) \), \( p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d = 5, 10, 9) \) and \( p(s \rightarrow a \rightarrow g \rightarrow f \rightarrow i \rightarrow k \rightarrow d = 5, 11, 10) \) are satisfying our constraint bound (5, 11, 10) in figure 3. Our algorithm returns the shortest feasible path \( p(s \rightarrow b \rightarrow f \rightarrow h \rightarrow j \rightarrow d = 5, 9, 8) \) among the three feasible paths and remaining paths are not selected since they have not satisfied the constraint bound. For example, path \( p(s \rightarrow a \rightarrow g \rightarrow f \rightarrow h \rightarrow j \rightarrow d = 11, 14, 11) \) and \( p(s \rightarrow b \rightarrow c \rightarrow a \rightarrow g \rightarrow i \rightarrow k \rightarrow d = 11, 19, 17) \). Our results are shown in figure 4 to figure 6 for different topologies.

![Fig.4. Ratio of path weight versus Constraints for Arbitrary Topology](image1)

![Fig.5. Ratio of path weight versus Constraints for ARPANET Topology](image2)

![Fig.6. Ratio of path weight versus Constraints for ANSNET Topology](image3)
For a path \( p(sbfhjd) \), delay of path is, 
\[
D(p(sbfhjd)) = D(s,b)+D(b,f)+D(f,h)+D(h,j)+D(j,d) = 1+1+1+1+1 = 5.
\]

Also the cost of the path \( p(sbfhjd) \) is calculated by adding the cost of each edges along the path. 
\[
C(p(sbfhjd)) = C(s,b)+C(b,f)+C(f,h)+C(h,j)+C(j,d)
\]
\[
= 1+1+1+1+1 = 5.
\]

Among these 5 feasible paths DCMCR returns path \( p(sbfhjd) \) as shortest feasible path, since the cost of this path \( =5 \) is minimum when compared to the cost of each other feasible path. We noticed that DCMCR algorithm returns the other two paths \( p(sbfhjd) \) and \( p(sbcagfikd) \) as feasible paths, because delay of these paths are \( =11 \) which is not returned as \( \Delta_d (=11) \) was away from the bound. Our results are presented in fig 8.

Optimal Weights for DCMCR

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 \( (\alpha) \) 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 \( \alpha = 0.5 \). 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

Discussion

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 $\alpha$ chosen becomes smaller, which implies better accuracy of results.

**Execution time versus Approximation factor ($\alpha$)**

![Graph showing execution time versus approximation factor](image)

**Execution time versus Number of nodes ($v$)**

![Graph showing execution time versus number of nodes](image)

**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 $(1-\alpha)$ 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**


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.