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 coercion 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 remaining constraint is not 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 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 algorithms 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].


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=(w1,...,wk) is an edge weight vectors so that wk(e)>0 is kth weight of edge ‘e’, ∀e ∈ E, 1≤k≤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 p} w_k(e). Integer constant K is to denote the number of constraints and each constraint has k weights. The path weight is defined as sum of all edge weights over the path normalized by corresponding constraint.
of QoS metrics. A constant $W = (W_1, ..., W_K)$, 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 $w_k(p) \leq W_k$ 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, $\Delta_K$, 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 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 $w_k(p) \leq W$ for all $1 \leq k \leq K$ where $x_{opt}$ is smallest real number and $x \geq 0$ such that there exists an s-d path ‘p’ satisfying $w_k(p) \leq W$ for all $1 \leq k \leq K$.

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 $x_{opt} < 1$ 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 path of MCPv3. We choose $x_{opt} > 1$ if and only if MCPv1 is feasible. When $x_{opt} > 1$ 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 $w_1(e), ..., w_k(e)$ 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+\$) 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+\$) approximation to MCPv3. Let LB[i] and UB[i] denote the lower and upper bound as explained above.

\[ LB[i] \leq x_{opt} \leq 2UB[i] \] (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 $\gamma = \frac{v-1}{LB \cdot W \cdot \alpha}$ and let $p_{opt}$ is an optimal solution to MCPv3, which means that $p_{opt}$ is a source-destination s-d path such that $w_k(p_{opt}) \leq x_{opt} \cdot W$ for $k=1, 2, ..., K$. Since, $w_{aux}(e)=[w_k(e), \gamma]+1 \leq w_k(e) \cdot \frac{v-1}{LB \cdot W \cdot \alpha} + 1$ for all edges $e \in E$. We have,

\[ w_{aux}(p_{opt}) \leq w_k(p_{opt}) \cdot \left( \frac{v-1}{LB \cdot W \cdot \alpha} + (v-1) \right) \] (2)

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

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

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

\[ w_{aux}(p_{opt}) = \frac{2UB \cdot (v-1)}{LB \cdot \alpha} + (v-1) \] (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,

\[ \max_{1 \leq k \leq K} w_{aux}(p_{opt}) \leq x_{opt} \cdot \frac{v-1}{LB \cdot \alpha} + (v-1) \] (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 $\left\lfloor \frac{2UB \cdot (v-1)}{LB \cdot \alpha} + (v-1) \right\rfloor$ 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,

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

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

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

We also have,

\[ \max_{1 \leq k \leq K} w_{aux}(p_{aux}) = \sum_{e \in p_{aux}} w_{aux}(e) \geq \sum_{e \in p_{aux}} w_k(e) \cdot \frac{v-1}{LB \cdot W \cdot \alpha} = w_k(p_{aux}) \cdot \frac{(v-1)}{LB \cdot W \cdot \alpha} \] for all $1 \leq k \leq K$ (8)

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

\[ w_k(p_{aux}) \cdot \frac{(v-1)}{LB \cdot W \cdot \alpha} \leq x_{opt} \cdot \frac{(v-1)}{LB \cdot \alpha} + (v-1) \] for all $1 \leq k \leq K$ (9)
 compute_auxweight( )

 2.   int k, K, w aux[ ], w[ ], W, c, S, V, E, Q, T, ΔK;
 3.   int i=0, α, γ, LB[ ], len[ ], pred[ ];
 4.   string popt, paux, e, u, v, b, s;
   /* compute auxiliary edge weight */
 5.   compute_auxweight( )
 6.   K=3;
 7.   for(e=1;e<=E;e++) {  
 8.       for(k=1;k<=K,k++) {
 9.           w aux(e)=max(w(e)/W); } }  
/* compute shortest path using Dijkstra’s algorithm */
10.  compute_shortest path( )

*w initialize the set S of vertices and minimum priority queue Q*
11.  S=0;
12.  Q=V(G);
13.  while(Q!=0) {
14.      Q=V-S;
15.      u=Q;
16.      S=S+u;
17.      w aux(paux)=len[S];} }
18.  if(w aux(paux)<= max(w aux(p opt)/W)) {
19.      broken=paux;
20.      printf("feasible path is returned %s",p opt);
21.      break ( ); } }
22.  else {
23.   /* initialize lower LB and upper bound UB */
24.   LB[0] = w aux(paux)/K;
25.   UB[0] = w aux(paux)/2; }
26.  /* update the lower bound and upper bound values
    using testing procedure */
27.  while(UB[i] >= 2*LB[i]) {
28.      α = 1;
29.      γ = (v-1)/LB[i]*W;
30.      if(w aux(paux) = (v-1)/LB[i]*W) {  
31.          String Test(T,α) = "yes";  }
32.      else Test(T,α) = "no";  
33.      if(Test(T,α) = "yes")  
34.      LB[i] = T;
35.      else UB[i] = T;  
36.      i++;  
37.      UB[i] = UB[i];  
38.      LB = LB[i];
39.   /* construct an auxiliary graph G aux which is an instance
    of graph G */
40.   construct_auxgraph( )
41.   {γ = (v-1)/LB[i]*w aux(α)+1;  
42.    mcpv2.1( )  
43.    /*initialize (K-1) dimensional array to store length and
     predecessor of edges*/
44.    ΔK = <K-1> dimensional array to store length and
45.    predecessor of edges.
46.    for(k=1;k<=K;k++)  
47.    len[v,c]=∞;  
48.    pred[v,c]=null;  
49.    len[s,c]=0;  }
/* update array values */
50.   for(k=1;k<=K;k++)  
51.   for((u,v),<=E(u,v)+1) {  
52.       if(len[v,c] > len[u,b]+w(u,v) ) {  
53.           len[v,c] = len[u,b]+w(u,v);  
54.           pred[v,c]=u;  }  }
/* find the s-d path p aux such that w aux(p aux) <= ΔK
55.   if(len[d,c] <= ΔK)  
56.   p opt = p aux;  
57.   printf("Feasible path is returned %s",p opt);
58.   break ( ); }  

A. Pseudo Code
UB upper bound    LB lower bound
p opt feasible shortest path    paux auxiliary path
α approximation factor    c smallest constraint
T test value    v number of vertices
S set of vertices    Q minimum priority queue
s-d source-destination pair    G aux auxiliary graph
(u,v)edge    K number of QoS constraints
W constraint bound    ΔK auxiliary constraint bound
(V,E) set of vertices and set of edges.
w k(p opt) kth weight of feasible path p opt
w k(p aux) kth 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.
1. void main() {  
2.   int k, K, w aux[ ], w[ ], W, c, S, V, E, Q, T, ΔK;
3.   int i=0, α, γ, LB[ ], len[ ], pred[ ];
4.   string popt, paux, e, u, v, b, s;
   /* compute auxiliary edge weight */
5.   compute_auxweight( )
6.   K=3;
7.   for(e=1;e<=E;e++) {  
8.       for(k=1;k<=K,k++) {
9.           w aux(e)=max(w(e)/W); } }  
/* compute shortest path using Dijkstra’s algorithm */
10.  compute_shortest path( )
are positive integer constants. Our objective is to find \( D_\text{aux} \) which is same as graph \( G \) except that the edge requirements of QoS metrics. Then we construct an auxiliary graph \( G_{\text{aux}} \) having an auxiliary edge weight \( w_{\text{aux}}(p_{\text{aux}}) \) 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 \( G_{\text{aux}} \) which is same as graph \( G \) except that the edge weighting function \( w_k(e) \) is changed to \( w_{\text{aux}}(e) \), such that \( w_{\text{aux}}(e) = \gamma \cdot (w_k(e) + 1) \) 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 \( p_{\text{aux}} \) 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

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 \( w_k(e) \geq 0 \) is the \( k \)th weight of edge \( e \), for all \( e \in E \), \( 1 \leq k \leq K \). For a path \( p_{DMR} \) in \( G \), the \( K \) th weight of path \( p_{DMR} \) is denoted by \( w_k(p_{DMR}) \), is the sum of \( K \)th 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. \( \Delta_d \) is the delay constraint for the source-destination pair. 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 \( c \leq \chi_{opt} \leq c_v \), where \( c_v \) is the number of vertices and \('v'\) is the smallest constraint.

LB[0] \leq \chi_{opt} \leq UB[0] \leq v.LB[0] \quad \text{(11)}

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, \( D(p_{DMR}) \leq \Delta_d \)

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

Here we construct an auxiliary graph \( G_{\text{aux}} \) which is an instance of graph \( G \) to find the optimal value \( \chi_{opt} \) of our algorithm DCMCR. To find a \( \chi_{opt} \), we perform a testing procedure defined in our algorithm. Hence, \( C_{\text{aux}}(p_{DMR}) = \sum_{e \in p_{DMR}} [C(e) \cdot \gamma] + 1 \)

\( \leq \sum_{e \in p_{DMR}} [C(e) \cdot \gamma] + (v-1) \quad \text{[
\begin{align*}
&\leq (v-1) + \gamma \sum_{e \in p_{DMR}} C(e) \\
&\leq (v-1) + \gamma C(p_{DMR}) \\
&\leq (v-1) + C(p_{DMR}) \frac{(v-1)}{\text{LB} - c}
\end{align*}
\]

Substitute the value of \( C(p_{DMR}) \) from (12),
\[ C_{aux} (p_{DMR}) = (v-1) + \frac{x_{opt} \cdot (v-1)}{LB \cdot \alpha} \]

Substitute the value of \(x_{opt}\) from (11),
\[ C_{aux} (p_{DMR}) = (v-1) + \frac{UB \cdot (v-1)}{LB \cdot \alpha} \]

(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 \(MCPv2.2\).

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 \(1+\alpha\) approximation of DCMCR. Since \(p_{aux}\) is computed by the algorithm, we have,
\[ D(p_{aux}) \leq \Delta_d \text{ and } C_{aux}(p_{aux}) \leq \left[ \frac{UB \cdot (v-1)}{LB \cdot \alpha} \right] + (v-1) \]

(15)

And,
\[ C_{aux}(p_{aux}) \leq C_{aux}(p_{DMR}) \]

(16)

We know that,
\[ C_{aux}(e) = \gamma \cdot C(e) \]
\[ \therefore C_{aux}(p_{aux}) = \gamma \cdot C_{aux}(p_{DMR}) \]

(17)

From (15), we have,
\[ C(p_{aux}) = \frac{1}{\gamma} \cdot C_{aux}(p_{DMR}) \]

(18)

From (13), we get,
\[ C(p_{aux}) = \frac{1}{\gamma} \left[ \gamma \cdot C(p_{DMR}) + (v-1) \right] \]

(19)

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

\[ \leq x_{opt} + \frac{(v-1)}{\gamma} \]

(20)

\[ \leq x_{opt} + \frac{(v-1)}{\gamma} \cdot \frac{1}{LB \cdot \alpha} \]

\[ \leq x_{opt} + x_{opt} \cdot \alpha, \quad [\therefore LB \leq x_{opt}] \]

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

**B. Pseudo Code**

\(p_{DMR}\) feasible shortest path \(\Delta_d\) delay constraint \(\Delta C\) cost constraint \(C\) real value cost \(C_{aux}\) auxiliary real value cost \(D\) 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}, 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. T = s - S;
12. c = Q;}
13. /* construct an auxiliary graph \(G_{aux}\) which is an instance of graph \(G\) to find the optimal value \(x_{opt}\) of DCMCR */
14. **construct_auxgraph()** {
15. \(\gamma = (v-1)/LB\alpha\);\n16. \(C_{aux}(e) = \gamma \cdot C(e)\);\n17. \(a = \log(v)\);\n18. \(b = \pow(a,2)\);\n19. /* perform testing procedure to refine the value of lower bound LB and upper bound UB */
20. while (UB[i] >= 2 \((1+\alpha)\) LB[i]) {
21. T = sqrt ((LB[i] * UB[i]) / (1+\alpha));
22. if (D(p) <= \Delta_d \&\& C_{aux}(p) <= \((v-1)/\gamma\)) {
23. String Test(T, \gamma) = “yes”;\n24. \}
25. else { Test(T, \gamma) = “no”; \}
26. if (Test(T, \alpha) = “yes”) {
27. UB[i+1] = T; \alpha = pow(a,2);\n28. \}
29. else { UB[i+1] = LB[i];
30. LB[i+1] = T; \}
31. \}
32. \}
33. \}
34. \}
35. \}
36. \}
37. \}
38. \}
39. \}
40. \}
41. \}
42. \}
43. \}
44. \}
45. **mcpv2.2()** {
46. /* initialize \((K-1)\) dimensional array to store length and predecessor of edges */
47. K = 3;
48. for (k = 2; k <= K; k++) {
49. len[v, ek] = \infty;\n50. pred[v, ek] = null;\n51. len[s, ek] = 0;\n52. \}
/* update array values */
for(k=2;k<=K;k++) {
    for ((u,v)=1;(u,v)<E;(u,v)++) {
        if(len[v,c_k] > len[u,b_k] +w(u,v) ) {
            len[v,c_k] = len[u,b_k] +w(u,v); 
            pred[v,c_k] = u;  }}}
/* find the s-d path paux such that D(paux) <= D_d and Caux(paux) <= C_d */
if(len[d,c] <= D_d ) {
    pDMR =paux;
    fprintf("Feasible path is returned %s",pDMR;}
    break ( );}
else {
    fprintf("No feasible path found");
    break ( );}  }}

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. Upper bounds 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 sizes 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 topologies, 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 Kth weight of path p_opt w_k(p_opt) <= D_d, 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→b→f→i→k→d) and p(s→b→c→a→g→i→h→j→d) in figure 3.

w_{aux}(e) = \max_{k \leq K} \frac{w_k(e)}{W}

We calculate Kth weights of path p ( s → b → f → i → k → d),

\begin{align*}
    w_k(p(s→b→f→i→k→d)) &= \sum_{k} 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→b→f→i→k→d)) &= (5,11,10)
\end{align*}

From the above Kth weight of path, we compute the auxiliary path weight w_{aux}(p),

\begin{align*}
    w_{aux}(p(s→b→f→i→k→d)) &= \max_{k \leq K} \left[ \frac{w_k(s→b→f→i→k→d)}{W} \right] \\
    &= \max_{k \leq K} \left[ \frac{w_1(sfbikd)}{W}, \frac{w_2(sfbikd)}{W}, \frac{w_3(sfbikd)}{W} \right]
\end{align*}

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

w_{aux}(p(sbcagihjd))= \max_{k \leq K} \frac{w_k(sbcagihjd)}{W}

w_k(p(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)

\begin{align*}
    w_{aux}(p(sbcagihjd)) &= \max_{k \leq K} \left[ \frac{16}{10}, \frac{21}{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 \( w_{\text{aux}}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) = 0 \) or not. In our example \( w_{\text{aux}}(p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d)) = 0.5 \), so we initialize the lower and upper bounds according to our MCPv3 algorithm.

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

\[
\begin{align*}
\text{LB} &= \frac{0.5}{3} = 0.17 \\
\text{UB} &= \frac{0.5}{2} = 0.25
\end{align*}
\]

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

\[
\text{LB} \leq x_{\text{opt}} \leq 2 \text{UB}
\]

\[
0.17 \leq x_{\text{opt}} \leq 0.5
\]

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

\[
\begin{align*}
\frac{w_k(p_{\text{opt}})}{x_{\text{opt}}} \leq W_k
\end{align*}
\]

where \( w_k(p_{\text{opt}}) \leq 0.5(10,22,20) \)

\[
\text{and} \quad \frac{w_k(p_{\text{opt}})}{x_{\text{opt}}} \leq (5,11,10)
\]

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

\[
\text{LB} \leq x_{\text{opt}} \leq 2 \text{UB}
\]

\[
0.17 \leq x_{\text{opt}} \leq 0.5
\]

Fig. 4. Ratio of path weight versus Constraints

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

\[
\begin{align*}
\frac{w_k(p_{\text{opt}})}{x_{\text{opt}}} \leq W_k
\end{align*}
\]

where \( w_k(p_{\text{opt}}) \leq 0.5(10,22,20) \)

\[
\text{and} \quad \frac{w_k(p_{\text{opt}})}{x_{\text{opt}}} \leq (5,11,10)
\]

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

\[
\text{LB} \leq x_{\text{opt}} \leq 2 \text{UB}
\]

\[
0.17 \leq x_{\text{opt}} \leq 0.5
\]

For example paths \( p(s \rightarrow b \rightarrow f \rightarrow h \rightarrow j \rightarrow d) \) are found to be feasible, since \( \frac{w_k(p_{\text{opt}})}{x_{\text{opt}}} = (5, 11, 10) \) 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 \( p(s \rightarrow b \rightarrow f \rightarrow h \rightarrow j \rightarrow d) \) and \( p(s \rightarrow b \rightarrow f \rightarrow i \rightarrow k \rightarrow d) \) are found to be feasible, since \( w_k(p_{\text{opt}}) \leq (5,11,10) \). Similarly, \( w_k(p_{\text{opt}}) \leq (5,11,10) \) and is not satisfying the above condition and found infeasible.

Fig. 5. Ratio of path weight versus Constraints

DCMCR algorithm is applied to the same arbitrary topology to find the shortest feasible path, \( p_{\text{DMR}} \) such that \( C(p_{\text{DMR}}) = \sum C(e) \) is minimized subject to (K-1) constraints and the delay of the path \( D(p_{\text{DMR}}) \leq \Delta_d \), 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.

\[
\begin{align*}
D(e) &= w_1(e) + D(f,h) + D(h,j) + D(j,d)
\end{align*}
\]

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

\[
D(p(sbfhjd)) = D(s,b) + D(b,f) + D(f,h) + D(h,j) + D(j,d)
\]
w3 lying on the ‘plane’ denotes feasible path region for a given source-destination pair as illustrated in figure 7.

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

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

\[ C(p(sbfihjd)) = C(s,b)+C(b,f)+C(f,i)+C(I,h)+C(h,j)+C(j,d) \]

\[ = 0.1+0.1+0.15+0.25+0.09+0.1 = 0.79 \]

And cost of path p(sbcagfikd) as infeasible path, because delay of these paths are (=17) is greater than the given delay bound. Our DCMCR algorithm returns the other two paths p(sbfhjd) and p(sbfhjd) as feasible paths, because delay of these paths are minimum when compared to the cost of other feasible path p(sbfhjd) (=0.48). We represented these five feasible paths on a 3D plane. The weights w1, w2, w3 lying on the ‘plane’ denotes feasible path region for a given source-destination pair as illustrated in figure 7.

Among these paths { p(sbfhjd), p(sbfhjd), 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 \( \Delta_d = 10 \). DCMCR algorithm returns the other two paths p(sbfhjd) and p(sbfhjd) as feasible paths, because delay of these paths satisfy the delay bound \( \Delta_d \). 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(sbfhjd) (=0.79). We represented these five feasible paths on a 3D plane. The weights w1, w2, w3 lying on the ‘plane’ denotes feasible path region for a given source-destination pair as illustrated in figure 7.

Among these 5 feasible paths DCMCR returns path p(s→b→f→h→j→d) as shortest feasible path since it satisfies both delay and cost constraint. Other path such as p (s→a→g→f→h→j→d) = 11 not returned as \( \Delta_d = 11 \) was away from the bound. Our results are presented in fig 8.

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 (\( \alpha \)) 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.

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 \( O(v(H/w)) \) time, where \( H \) (i.e. hop count) is small. MCPv3 and DCMCR take more time as the value of \( w \) chosen becomes smaller, which implies better accuracy of results.

### Execution time versus Approximation factor \( (\alpha) \)

![Execution time versus Approximation factor (\(\alpha\))](image1)

### Execution time versus Number of nodes \( (v) \)

![Execution time versus Number of nodes (v)](image2)

### 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.