

Heuristic Methods for Delay Constrained Least Cost Routing Using k -Shortest-Paths

Zhanfeng Jia and Pravin Varaiya

Department of Electrical Engineering and Computer Science

University of California, Berkeley, CA 94720

E-mail: {jia,varaiya}@eecs.berkeley.edu

Abstract— The Delay Constrained Least Cost (DCLC) problem is to find the least cost path in a graph while keeping the path delay below a specified value. First formulated in the context of routing in computer networks, the DCLC model also applies to mission planning and other decision problems. DCLC is NP-complete and therefore many heuristic algorithms have been proposed for it. This paper presents two new methods based on k -Shortest-Path (k SP) algorithms. These heuristic methods, one centralized and one distributed, are both polynomial. In numerical experiments the proposed algorithms almost always find the optimal paths.

I. INTRODUCTION

Conventional routing protocols characterize network links by a single metric, such as hop count; Dijkstra's algorithm and the Bellman-Ford algorithm find the shortest path based on the metric. For routing with explicit quality of service (QoS) requirements, the link model also includes parameters such as bandwidth, cost, delay, and loss probability. Routing is then more complicated because it must find paths that satisfy multiple constraints.

The *Delay Constrained Least Cost* DCLC problem characterizes network links by two metrics, cost and delay. The link cost may be a monetary cost or a measure of link utilization such as hop count. The link delay usually consists of the propagation and queuing delay. Different link metrics may represent different QoS requirements. Most of these metrics are additive: the total metric of a path is the sum of link metrics along the path. The problem of finding a path subject to two or more additive metrics is NP-complete [2].

The DCLC formulation also models some decision making and mission planning problems. For example, in planning a robot's path through a hazardous environment one may want to find the shortest path subject to a constraint on the total risk incurred [8]. Another example is

This research was supported by DARPA-N66001-00-C-8062 and DARPA-F33615-01-C-3150.

to find a path through a tolled road network that takes minimum time subject to a constraint on the total toll.

More abstractly, in a finite state system with states x , control u , two cost functions $c(x, u)$ and $d(x, u)$, initial state s and final state d , one may wish to find a control sequence u_1, \dots, u_n that steers the system from s to d and minimizes $\sum c(x_i, u_i)$, keeping $\sum d(x_i, u_i) \leq \Delta$. This, too, is a DCLC problem.

The *Pareto set* of non-dominated paths illustrates the difficulty in finding the optimal DCLC path. A path p is *dominated* if there exists another path p' that is better than p in one metric, and not worse with respect to the other metric. The optimal DCLC path must belong to the Pareto set. Heuristic methods for the DCLC problem generally find a promising subset of the Pareto set and then select the optimum within the subset. For example, the Lagrange relaxation method finds paths that minimize a convex combination of cost and delay, and so it can only find paths that are on the boundary of the convex hull of the Pareto set, which may not include the optimal path because of non-convexity.

A. DCLC Problem Formulation

The network is modeled as a directed, connected graph $G = (V, E)$, where V is the set of nodes and $E \subset V \times V$ is the set of directed links. Let $n = |V|$ and $m = |E|$ be the number of nodes and links. Each link is characterized by a cost $c(e)$ and a delay $d(e)$, both nonnegative.

Given a source node $s \in V$, a destination node $d \in V$, and a positive delay constraint Δ_{delay} , the DCLC problem is:

$$\min_{p \in P'(s,d)} \sum_{e \in p} c(e). \quad (1)$$

Here $P'(s, d)$ is the set of paths from s to d for which the end-to-end delay is bounded by Δ_{delay} . With $P(s, d)$ denoting the set of all paths from s to d , a path $p \in P(s, d)$

belongs to $P'(s, d)$ if

$$\sum_{e \in p} d(e) \leq \Delta_{delay}. \quad (2)$$

As noted, the DCLC problem is NP -complete [6].

The *Multiple Constraints Path* (MCP) problem is related: Given two or more additive link metrics, the MCP problem is to find a feasible path (or paths) that satisfies a constraint for each metric. Because no metric is optimized, some believe that MCP, although NP -complete, is simpler than DCLC.

B. Previous Work

The *Constrained Bellman-Ford* (CBF) routing algorithm proposed by Widyono in [13] solves the DCLC problem and gives the optimal path. CBF performs a breadth-first search and records all paths that are feasible but not dominated. Thus CBF searches the entire Pareto set. Unfortunately, CBF has exponential worst-case complexity because the size of the Pareto set grows exponentially.

To solve the DCLC problem in polynomial time, the proposed algorithms use heuristics and take approximations. As a simple and fast initial approach, Lee et al [7] propose a *Fallback* algorithm that tries each of the metrics and checks whether it is feasible. Pornavalai et al [9] improve upon this idea by combining paths calculated with the different metrics. The *Delay Constrained Unicast Routing* (DCUR) algorithm by Salama et al [10] lets each node choose between least cost and least delay paths independently of the choice of the previous node.

The other type of algorithms beginning with Jaffe [3] first construct a single metric and then use a shortest-path algorithm. Juttner et al in [5] propose a Lagrange relaxation based method (LARAC) that linearly combines cost and delay. Wang and Crowcroft [11] propose non-linear combinations. None of these algorithms guarantees the optimal solution.

This paper presents two new methods to DCLC. The underlying idea is to use k -Shortest-Path (k SP) algorithms so that the methods have more than one choice at each step of the heuristics, thereby exploring a larger subset of the Pareto set. The proposed algorithms are polynomial. Moreover, in the numerical experiments reported here they almost always achieve the optimal solution.

The remainder of the paper is organized as follows. In Section II, we describe the k SP algorithms and how we use them. In Section III and IV, we describe the two proposed methods, and analyze their complexity. Section V presents the numerical experiments. Section VI further discusses the relationship between optimality and k . Section VII concludes the paper.

II. k -SHORTEST-PATH ALGORITHMS

Shortest path problems are widely studied. Dijkstra's algorithm finds the shortest paths from a source node to every other node in $O(n \log n)$ time. In this paper, we are interested not only in the shortest path, but in the k shortest paths between nodes, to better explore the Pareto set. Algorithms for k shortest paths were first proposed in the 1950s.

The asymptotically fastest known k SP algorithm is due to Eppstein in [1]. After running Dijkstra's algorithm to find a shortest path tree rooted at the source node, Eppstein's algorithm (EA) builds a heap-ordered data structure representing all possible deviations from the shortest paths. Building this data structure takes $O(m + n \log n)$ time. The k th shortest path can then be derived by traversing the heap, in $O(\log k)$ time. So EA has almost the same time complexity as Dijkstra's algorithm.

Jimenez and Marzal later proposed the *Recursive Enumeration Algorithm* (REA) [4]. At step k , REA maintains a set of candidate paths, denoted $C_k(v)$, in which v is the destination node. The k th shortest path p_k from source to v is the shortest path in $C_k(v)$. To find the $(k + 1)$ st shortest path, REA removes p_k from $C_k(v)$ and inserts deviation paths of p_k to form $C_{k+1}(v)$. Although REA requires $O(m + k \log(m/n))$ time in the worst case, REA outperforms EA in practice, especially for small k . On the other hand, Eppstein's data structure makes it easy to find k shortest paths to every single node in the graph, whereas REA focuses on one specific destination node.

These k SP algorithms are fast. In our numerical experiments the runtime of finding up to 1,000 shortest paths between two nodes is no more than four times the runtime of Dijkstra's single shortest path algorithm.

Both EA and REA are centralized since they require information about the entire graph and build data structures based on it. To find the k shortest paths in a distributed way, we extend the Bellman-Ford algorithm and record up to k paths ordered by length at each node. This extended algorithm, called the *k -Bellman-Ford* (k BF) algorithm, takes $O(knm)$ steps in the worst case and $O(k \log k)$ time within each step.

The speed of k SP algorithms makes it attractive to integrate them into algorithms for the DCLC problem. The integration is straightforward, if not trivial, for the following reasons. First, the k SP algorithms are extensions of the shortest-path algorithms such as Dijkstra or Bellman-Ford, which form the core of many current heuristic methods for the DCLC problem. Therefore, the k SP algorithms can easily replace and play the role of Dijkstra's shortest path algorithm.

Second, all the heuristic methods first reduce the path

space, and then find the optimum within the reduced space. This procedure may happen recursively, until a satisfactory suboptimal path is found. Dijkstra's shortest path algorithm can serve in either part of each recursion. For example, in DCUR [10], the path space is narrowed by leaving only two choices at each node, the least cost link and the least delay link, both derived from Dijkstra's algorithm; in LARAC [5], Dijkstra's algorithm is used in each recursion of the heuristic to find the "shortest" path based on a single metric that linearly combines the cost and delay. k SP algorithms provide more candidate paths than Dijkstra's algorithm, and can find a larger subset of the Pareto set with paths not on the boundary of the convex hull. In the next two sections, we discuss the integration in more detail, and construct the k SP based algorithms.

III. CENTRALIZED k LAM ALGORITHM

The *linearly aggregated metric* (LAM) method is an efficient heuristic method because the aggregated single metric translates the DCLC problem into a shortest path problem. The heuristic recursively minimizes an aggregated metric function,

$$m_\lambda = c + \lambda \cdot d,$$

for given λ , and finds the minimizing path, denoted \hat{p}_λ . The parameter λ represents how much weight is put on the delay metric. If $d(\hat{p}_\lambda) > \Delta_{delay}$, we increase λ to increase the importance of the delay metric in m_λ .

As discussed, finding more than one shortest paths is helpful to the heuristic. Let P_λ be the set of k shortest path for given λ in terms of the λ -metric, i.e., $P_\lambda = \{p_1, p_2, \dots, p_k\}$, with $m_\lambda(p_1) \leq m_\lambda(p_2) \leq \dots \leq m_\lambda(p_k) \leq m_\lambda(p)$, $p \notin P_\lambda$. Let P'_λ be the (possibly empty) subset of paths in P_λ that satisfy the delay constraint, i.e., $P'_\lambda = \{p \in P_\lambda : d(p) \leq \Delta_{delay}\}$. We introduce a bound function $B(\lambda)$ based on P_λ that leads to the k LAM solution.

Definition: Let $L(\lambda, p)$ be the Lagrangian defined by

$$\begin{aligned} L(\lambda, p) &= m_\lambda(p) - \lambda \Delta_{delay} \\ &= c(p) + \lambda(d(p) - \Delta_{delay}). \end{aligned} \quad (3)$$

The bound function $B(\lambda)$ is defined as

$$B(\lambda) = \begin{cases} L(\lambda, p_{c_{min}}), & \text{if } P'_\lambda = \phi \\ L(\lambda, p'_{c_{min}}), & \text{otherwise} \end{cases} \quad (4)$$

where

$$\begin{aligned} p_{c_{min}} &= \arg \min_{p \in P_\lambda} c(p), \\ p'_{c_{min}} &= \arg \min_{p \in P'_\lambda} c(p). \end{aligned}$$

For given λ , the path that achieves $B(\lambda)$ is called the *chosen path* at λ , denoted by p_λ . According to the value of Δ_{delay} , the set P_λ can be classified into three types. The first type contains no feasible path that meets the delay constraint, i.e. P'_λ is empty. We call it the p_c -type. The second type consists of paths that are all feasible, called the p_d -type. In this case $P'_\lambda = P_\lambda$. Sets of the third type have both feasible and unfeasible paths and are called the mixed type.

$B(\lambda)$ is related to the Lagrange function $L(\lambda)$ in [5],

$$L(\lambda) = \min_{p \in P(s,d)} L(\lambda, p). \quad (5)$$

Although $B(\lambda)$ loses some good properties of $L(\lambda)$ such as continuity and concavity, it lower bounds the objective function.

Proposition 1: $B(\lambda)$ is a lower bound to the DCLC problem for all $\lambda \geq 0$. Moreover, $B(\lambda)$ is a better lower bound than the Lagrange function $L(\lambda)$.

Proof: Let p^* be the optimal solution to the DCLC problem. If $p^* \in P_\lambda$, P_λ is either of p_d -type or mixed type. In both cases p^* minimizes the cost metric of feasible paths in P_λ , so $B(\lambda) = L(\lambda, p^*)$. On the other hand, if $p^* \notin P_\lambda$, for any $p \in P_\lambda$, $m_\lambda(p) \leq m_\lambda(p^*)$, so $B(\lambda) \leq L(\lambda, p^*)$. Thus for all $\lambda \geq 0$,

$$\begin{aligned} B(\lambda) &\leq L(\lambda, p^*) \\ &= m_\lambda(p^*) - \lambda \Delta_{delay} \\ &= c(p^*) + \lambda(d(p^*) - \Delta_{delay}) \leq c(p^*). \end{aligned}$$

Since the chosen path does not necessarily minimize m_λ ,

$$L(\lambda) \leq B(\lambda). \quad (6)$$

□

Maximizing the bound function $B(\lambda)$ will give the best lower bound to the optimal path. Finding the maximal value

$$B^* = \max_{\lambda \geq 0} B(\lambda) \quad (7)$$

and the maximizing λ^* form a well-defined optimization problem. Problem (7) is the k LAM approximation of the original DCLC problem.

Figure 1 shows an example of $B(\lambda)$ when $k=2$. Each path p yields a straight line in the figure with slope $(d(p) - \Delta_{delay})$ and intercept $c(p)$. Lines with positive slopes represent paths that are not feasible, while lines with negative or zero slopes are feasible paths. The λ space can be divided based on the type of P_λ . In the example, small λ values correspond to P_λ of p_c -type. As λ increases, P_λ becomes mixed, and ends up with p_d -type, representing the increasing dominance of the delay metric. Note that in this example the solution λ^* to problem

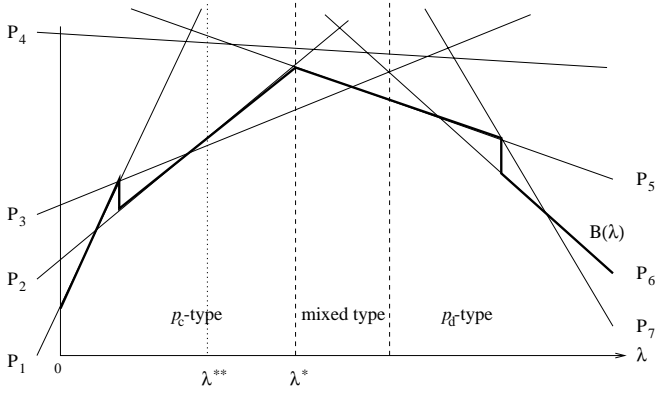


Fig. 1: This figure shows the $B(\lambda)$ curve ($k = 2$) in thick line. Each thin straight line represents the Lagrangian of a path. Observe that $B(\lambda)$ is always greater than or equal to $L(\lambda)$, which minimizes the Lagrangian at all λ .

(7) is located at the boundary of p_c -type and mixed type areas.

In the example of Figure 1, the optimal path, which is p_4 since it has the smallest intercept among the feasible paths, cannot be found no matter what λ is taken. However, if we increase k from 2 to 3, p_4 will become the chosen path at λ^{**} . This example illustrates that k LAM is able to find paths closer to the optimum by increasing k , and leads to the following proposition.

Proposition 2: Let $P_{\lambda,k}$ be the k shortest paths and $B_k(\lambda)$ be the bound. Then $B_k(\lambda) \leq B_{k+1}(\lambda)$.

Proof: Let p_{k+1} be the $(k+1)$ st shortest path at λ so that $P_{\lambda,k+1} = P_{\lambda,k} \cup \{p_{k+1}\}$. The chosen path of $P_{\lambda,k+1}$ is either the chosen path of $P_{\lambda,k}$ or p_{k+1} . In either case $B_k(\lambda)$, the Lagrangian for the chosen path, is less than or equal to $B_{k+1}(\lambda)$.

□

Note that the bound function $B_1(\lambda)$, with $k=1$, is the Lagrange function $L(\lambda)$. Propositions 1 and 2 together yield the theorem below, which provides theoretical support for the k LAM algorithm.

Theorem: $B_k(\lambda)$ gives a better lower bound to the optimal objective $c(p^*)$ as k increases, i.e.,

$$L(\lambda) = B_1(\lambda) \leq \dots \leq B_k(\lambda) \leq B_{k+1}(\lambda) \leq c(p^*). \quad (8)$$

The k LAM algorithm is described as follows. The algorithm recursively calls Dijkstra and k SP subroutines, with parameters s representing the source node, d the destination node, and λ the linear aggregation coefficient. Recall that \hat{p}_λ and p_λ are respectively the shortest path and the chosen path at λ .

Given G , s , d , and Δ_{delay}

$\lambda_c = \text{cost-only}$

$\hat{p}_{\lambda_c} = \text{Dijkstra}(s, d, \lambda_c)$

if $d(\hat{p}_{\lambda_c}) \leq \Delta_{delay}$, then return \hat{p}_{λ_c}

$P_{\lambda_c} = k\text{SP}(s, d, \lambda_c)$

if P_{λ_c} is mixed type, then return p_{λ_c}

$\lambda_d = \text{delay-only}$

$\hat{p}_{\lambda_d} = \text{Dijkstra}(s, d, \lambda_d)$

if $d(\hat{p}_{\lambda_d}) > \Delta_{delay}$, then return *failed*

$P_{\lambda_d} = k\text{SP}(s, d, \lambda_d)$

if P_{λ_d} is mixed type, then return p_{λ_d}

repeat

$$\lambda = \frac{c(p_{\lambda_c}) - c(p_{\lambda_d})}{d(p_{\lambda_d}) - d(p_{\lambda_c})}$$

Dijkstra(s, d, λ)

$P_\lambda = k\text{SP}(s, d, \lambda)$

if P_λ is mixed type, then return p_λ

else if P_λ is p_c -type, then $\lambda_c = \lambda$

else if P_λ is p_d -type, then $\lambda_d = \lambda$

end repeat

The first two iterations of k LAM work on the cost and delay metric separately. They correspond to $\lambda = 0$ and $\lambda = \infty$ so that k LAM can deal with extreme cases where the delay constraint is too big or too small. Though similar to a Lagrange relaxation method, the k LAM algorithm differs in two respects. First, it replaces the m_λ -minimal path with the P_λ set containing up to k shortest paths at each step of the heuristic. Second, it relaxes the stop condition to be any λ that makes the corresponding P_λ a mixed type. The following proposition shows that although k LAM is a relaxed version of problem (7), it always outperforms the conventional Lagrange relaxation method.

Proposition 3: Let λ_L maximize $L(\lambda)$ function and p_L be the solution of the Lagrange relaxation method such that $L(\lambda_L) = L(\lambda_L, p_L)$. Let λ be any value that makes P_λ a mixed type and, let p_λ be the chosen path. Then $c(p_\lambda) \leq c(p_L)$.

Proof: Because P_λ is a mixed type, it can be divided into two non-empty sets, namely P'_λ and its complement P^c_λ . Then

$$p_\lambda = \arg \min_{p \in P'_\lambda} c(p).$$

Let p^c_λ be the m_λ -minimal path among P^c_λ ,

$$p^c_\lambda = \arg \min_{p \in P^c_\lambda} m_\lambda(p).$$

If $p_L \in P_\lambda$, since p_λ minimizes the cost metric of the feasible paths, it is straightforward that

$$c(p_\lambda) \leq c(p_L).$$

If $p_L \notin P_\lambda$, consider the crossing point of the two lines

representing p_λ and p_λ^c . Let $\lambda' = \frac{c(p_\lambda^c) - c(p_\lambda)}{d(p_\lambda) - d(p_\lambda^c)}$. Then

$$\begin{aligned} L(\lambda, p_L) &\geq \max(L(\lambda, p_\lambda), L(\lambda, p_\lambda^c)) \\ &\geq L(\lambda', p_\lambda) \\ &\geq L(\lambda_L) = L(\lambda_L, p_L). \end{aligned}$$

The last inequality is due to the definition of $L(\lambda_L)$, which must be smaller than or equal to any crossing point of a feasible path and a non-feasible path. Now we have the following inequalities:

$$\begin{cases} L(\lambda_L, p_L) \leq L(\lambda_L, p_\lambda) \\ L(\lambda, p_L) \geq L(\lambda, p_\lambda) \\ L(\lambda, p_L) \geq L(\lambda_L, p_L) \end{cases}, \quad (9)$$

or

$$\begin{cases} c(p_L) + \lambda_L \cdot d(p_L) \leq c(p_\lambda) + \lambda_L \cdot d(p_\lambda) \\ c(p_L) + \lambda \cdot d(p_L) \geq c(p_\lambda) + \lambda \cdot d(p_\lambda) \\ \lambda \leq \lambda_L \end{cases}. \quad (10)$$

The result of (10) is that $c(p_\lambda) \leq c(p_L)$, which proves the proposition.

□

The running time of the algorithm is affected by both the number of steps during the heuristic and the computation within each step. It is shown in [5] that LARAC, which applies the Lagrange Relaxation method, terminates after $O(m \log^3 m)$ steps. Since k LAM uses the same method to calculate λ and has a relaxed stop condition, this time complexity applies to it, too. On the other hand, the computation of each step consists of building a heap-ordered data structure, finding first k paths, and sorting these paths. The total work adds up to $O(m + n \log n + k \log k)$ time. Therefore, the worst case time complexity of k LAM algorithm is $O(m \log^3 m \cdot (m + n \log n + k \log k))$.

It is hard to analyze the optimality of the paths found by k LAM. One can always build a graph so that k LAM fails to find the optimum. However, by simply increasing k at the expense of longer running times, it seems most constructed graphs are solvable. Therefore it is more reasonable to evaluate the optimality of the algorithms on real graphs, as in Section V.

IV. DISTRIBUTED k DCBF ALGORITHM

The k LAM algorithm is inherently centralized, because it uses Eppstein's k SP algorithm and replaces Dijkstra's algorithm. To solve DCLC in a distributed way, we introduce the well-known Bellman-Ford message exchange mechanism and develop a hop-by-hop algorithm. The

idea, following the CBF algorithm, is to extend possible paths to promising downstream nodes. This hop-by-hop method applies a completely different search strategy on the Pareto set compared with the Lagrange relaxation method.

The DCUR algorithm [10] is a simple instance using this idea. In DCUR, each node of the graph maintains two outgoing links that represent least cost (LC) and least delay (LD) paths to the destination node d , respectively. These two choices are found in advance using any shortest path algorithm. The DCUR algorithm starts from the source node s , and repeats until it reaches the destination or fails. At each node u , DCUR tests the LC link first and checks if the delay is feasible, i.e., if

$$\text{delay}(s, u) + d(e_{LC}) + d_{LD}(v, d) \leq \Delta_{\text{delay}}, \quad (11)$$

where $\text{delay}(s, u)$ is the sum of the delay metric along the DCUR path from s to u , $e_{LC} = (u, v)$ is the LC link, and $d_{LD}(v, d)$ is the delay of the LD path from v to d . DCUR then selects the LC link if test (11) succeeds, otherwise the LD link is selected. The selected link extends the path and DCUR moves to the downstream node of that link.

Test (11) plays a key role in DCUR algorithm, as well as in the proposed k Delay-Constrained Bellman-Ford (k DCBF) algorithm. We call (11) the *delay test*. To achieve the optimal path, k DCBF keeps a set of promising, feasible paths at each node. When these paths are extended to downstream nodes, the delay test will help to determine feasibility. Nodes need to know the delay to the destination for the delay test. Obviously, a set of k shortest delay paths is more helpful than only the shortest one. These sets can be prepared in advance by the distributed k -Bellman-Ford (k BF) algorithm.

The rest of this section describes the k DCBF algorithm. In k DCBF, each node u maintains two sets of paths for an (s, d) pair. The first set, P_u^d , contains k_d shortest delay paths from u to d . The second set, P_u^c , contains up to k_c paths from s to u in the shortest *aggregate cost* metric, defined by

$$\text{agg_cost}(p_{s,u}) = c(p_{s,u}) + \min_{\substack{p_{u,d} \in P_u^d \\ d(p_{s,u}) + d(p_{u,d}) \leq \Delta_{\text{delay}}}} c(p_{u,d}). \quad (12)$$

It can be seen that all paths in P_u^c pass the delay test. During the time that k DCBF is running, the P_u^c set is periodically updated, while the P_u^d set remains constant.

As the names suggest, k DCBF is similar to CBF in that both record more than one path at every intermediate node. The difference is that the number of paths held by k DCBF is bounded, but can grow exponentially in CBF. This bound restricts the total number of update

messages needed before the path between (s, d) is constructed and makes the algorithm polynomial. To estimate the worst case time complexity, suppose the k DCBF algorithm needs $O(k_c mn)$ update messages like in the k BF algorithm. With each received message, the node does the delay test, calculates the aggregate cost for each path, merges new paths to P_u^c , sorts them, and truncates the set to form an updated P_u^c . All this computation takes $O(k_c k_d + k_c \log k_c)$ time. Moreover, it takes $O(k_d^2 \log k_d mn)$ time to build the P_u^d sets at each node u . Therefore, the time complexity of the k DCBF algorithm is $O((k_c^2 k_d + k_c^2 \log k_c + k_d^2 \log k_d) mn)$.

V. NUMERICAL EXPERIMENTS

Two measures evaluate the performance of the DCLC algorithms: optimality and complexity. Algorithms trade-off between them. For example, CBF guarantees optimality but runs in exponential time. Since it is difficult to evaluate these measures analytically, we use numerical experiments instead.

A. Experiment Setup

The algorithms are implemented in C++. The experiments are run on a Linux PC. We compare the behavior of the implemented algorithms with the following measures.

- *Average Excess Cost*: Since CBF provides the optimal paths, we can calculate the excess cost each heuristic method introduces, defined as

$$excess_cost = \frac{c(p_{alg}) - c(p_{CBF})}{c(p_{CBF})} \times 100\%. \quad (13)$$

- *Average Number of Steps (NOS)*: Number of steps is defined differently for different heuristic methods. For linearly aggregated metric methods, NOS is the number of different λ 's that are used. For algorithms of Bellman-Ford style, NOS measures how many update messages are sent between neighbors.
- *Average Runtime*: The runtime in seconds is the most straightforward complexity measure. However, this is only a rough measure of the time complexity, because it depends on the programming language, code efficiency, and running environment.

For the experiments we randomly generate graphs with an average node degree of four. Two models are used to generate the graphs. The mesh graphs are generated by a rectangular mesh model, with nodes on the grids. The Waxman graphs are generated by the Waxman method [12], in which nodes are generated randomly on a two-dimensional plane of size 40×40 , and there is a link between two nodes u and v with probability

$$p(u, v) = \alpha \cdot e^{-d(u, v)/(\beta L)}. \quad (14)$$

here $0 < \alpha, \beta \leq 1$, $d(u, v)$ is the Euclidean distance between u and v , and L is the maximum distance between any two nodes. The cost metrics of the links are uniformly distributed between $(0, 15)$. The delay metrics, however, are differently generated in the two models. The delay metrics of mesh graphs are uniformly distributed between $(0, 30)$, independent of link costs. The delay metrics of Waxman graphs are the link length plus a random number between $(0, 5)$. The second model is also called the flat topology model of Internet, representing computer networks without hierarchical structure. For both models, the graphs are link-symmetrical, meaning that if there is a link from node u to v , there will also be a link from node v to u with the same cost and delay metrics.

During the experiments we calculate the delay metrics of the least cost path and the least delay path for a graph. The delay constraint is then chosen to be

$$\Delta_{delay} = 0.75 \cdot d(p_{LD}) + 0.25 \cdot d(p_{LC}), \quad (15)$$

preventing the DCLC problem from being trivial.

B. Performance Evaluation

Besides the CBF algorithm, which gives the optimal DCLC path, the experiments also include LARAC and DCBF algorithms. The LARAC algorithm follows the Lagrange Relaxation method exactly. Both LARAC and DCBF can be viewed as versions of k LAM and k DCBF with $k=1$. We now study the performance of the proposed algorithms. The statistics are based on 100 randomly generated graphs of 400 nodes with the mesh and Waxman models. The results are listed in Table 1. As expected, the k SP based algorithms outperform their $k=1$ counterparts. For mesh graphs, the average excess cost is less than 0.1%, while for Waxman graphs, this measure is literally zero, showing that both k LAM and k DCBF almost always find the optimal DCLC paths.

At the same time, k LAM and k DCBF are both efficient. The k LAM algorithm experiences 40-60% fewer steps than LARAC. This result follows the analysis in Section III that k LAM relaxes the stop condition of the heuristic. The runtime of k LAM is therefore only slightly greater than LARAC. The k DCBF algorithm pays relatively more for optimality. The average NOS is almost doubled as k increases from 1 to 10. Combined with the extra work of path ordering in each step, k DCBF is about 4-6 times slower than DCBF. Nevertheless, the complexity of k DCBF is acceptable compared with CBF that needs up to tens of thousands of steps to find the optimal paths in graphs of the same size.

When the graph size increases, the k SP based algorithms scale well. Figure 2 shows how the graph size

Mesh graphs	avg excess cost (%)	max excess cost (%)	avg number of steps (NOS)	avg runtime (sec)
LARAC	2.118	23.087	6.76	0.0761
k LAM	0.058	2.812	3.14	0.097
DCBF	12.486	46.036	602.37	0.0317
k DCBF	0.033	2.767	1025.4	0.2422
Waxman graphs	avg excess cost (%)	max excess cost (%)	avg number of steps (NOS)	avg runtime (sec)
LARAC	2.291	38.89	3.9	0.119
k LAM	0	0	2.18	0.126
DCBF	0.068	6.791	431.52	0.048
k DCBF	0	0	891.47	0.166

TABLE 1: Performance of the DCLC algorithms on different graph models. The statistics are based on 100 randomly generated graphs of 400 nodes. The k parameters of k LAM is $k = 100$, and for k DCBF, $k_c = k_d = 10$.

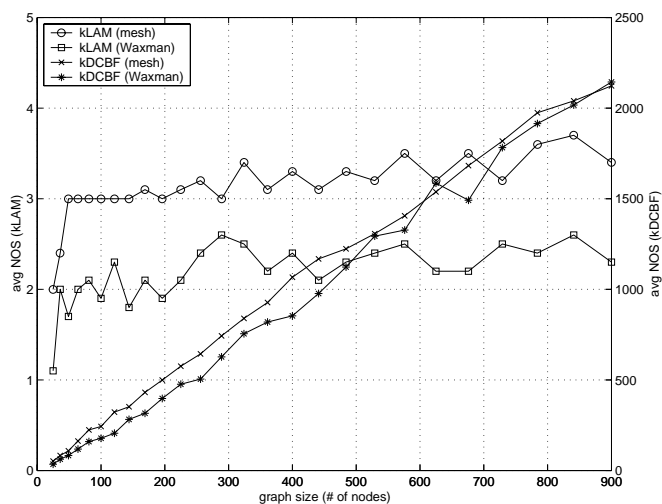


Fig. 2: Average NOS as the graph size increases. The statistics are based on 10 randomly generated graphs by two different models. The k parameters are same as they were in Table 1

affects the NOS of the heuristics. Each point in Figure 2 represents the average NOS of a group of ten experiments on graphs of certain size. The NOS increments are actually much smaller than the polynomial bounds given in Section III and IV. For the k LAM algorithm, the NOS remains small throughout the experiments. For the k DCBF algorithm, this quantity increases in proportion to the graph size. The observation is further strengthened by an extra group of experiments on 10,000 nodes mesh graphs (100-by-100 grid), generating average NOS values 4.6 for k LAM, and 18,427 for k DCBF. The average runtimes are 6.21 and 6.29 seconds, respectively. Note that the different graph models have little effect on k DCBF, but for k LAM, this is not the case – the NOS values are steadily larger on mesh graphs than on Waxman graphs.

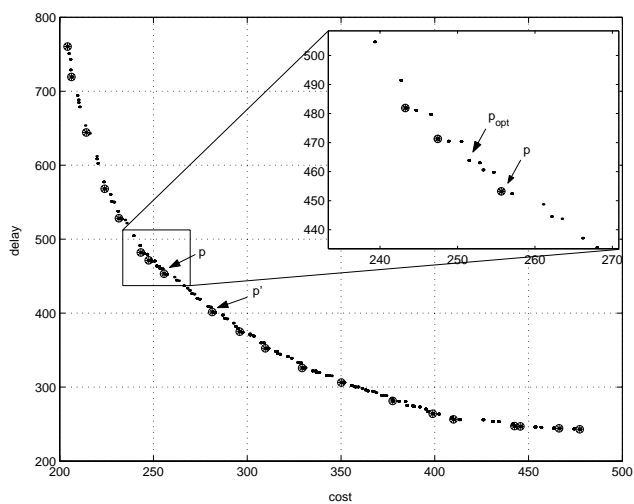


Fig. 3: Pareto set of the DCLC problem on a 900 nodes mesh graph (30-by-30 grid), with an enlarged portion on the top right. Each point represents the cost and delay metrics of a path. The points with circled star are those paths on the convex hull of the Pareto set.

C. Nonconvexity of the Pareto Set

In this subsection we show some experiments that explore the Pareto sets, and illustrate why it is hard to find the optimal DCLC paths using Lagrange relaxation. The difficulty comes from the nonconvexity of the Pareto set.

Figure 3 shows the Pareto set of the DCLC problem on a 900 nodes mesh graph, where the small figure in the top right corner shows the enlarged portion in the box. The Pareto set is not convex but the degree of nonconvexity is small. To find the optimal DCLC path on this graph with the delay constraint of 470, the Lagrange relaxation method, which can only search the convex hull, would find a path $p = (255.6, 453.2)$, where the two numbers in

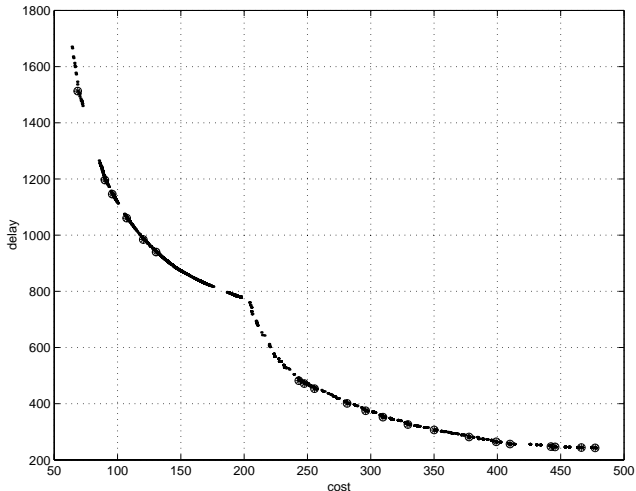


Fig. 4: Pareto set of the DCLC problem on a two-layer mesh graph. The two layers, both consisting of a 30-by-30 grid, are connected only at the source and destination nodes on two diagonal corners. Point representation in this figure is same as in Figure 3.

the parenthesis represent $(cost, delay)$ respectively. This is the lowest dark-circled point in the enlarged portion. However, the k SP based k LAM algorithm may be able to find the optimal path $p_{opt} = (251.5, 463.8)$, the fourth point above the previous one. The error introduced by the nonconvexity is 4.1 out of 251.5, or 1.63%, which is small. However, this is not always the case. Notice that the convex hull points are sometimes separated widely from each other. In Figure 3, the next convex hull point to the right of p is $p' = (281.3, 401.7)$. The difference in cost is about 25 and the error is up to 9.9%.

The degree of nonconvexity can be larger. In Figure 4 we show the Pareto set on a specialized two-layer mesh graph. Both layers of the graph consist of a 30-by-30 grid. The two layers are connected only at two diagonal corner nodes, which are also the source and destination nodes. Since the two layers are isolated, the Pareto set is a subset of the union of the Pareto sets with respect to each layer. Illustrated in Figure 4, the Pareto set divides at about $(200, 800)$ into two branches, each coming from the Pareto set of one layer. Thus, the two nearby convex hull points, $(130.5, 939.9)$ on the left and $(243.3, 481.9)$ on the right, generate a gap of 112.8 in cost, corresponding to an error up to 86.4%.

Obviously, because of the strong nonconvexity of the Pareto set, the Lagrange relaxation method performs poorly. Even k LAM, an enhanced version of Lagrange relaxation method, can hardly improve the result. However, the hop-by-hop k DCBF algorithm is not affected. This is easy to understand because k DCBF stores promis-

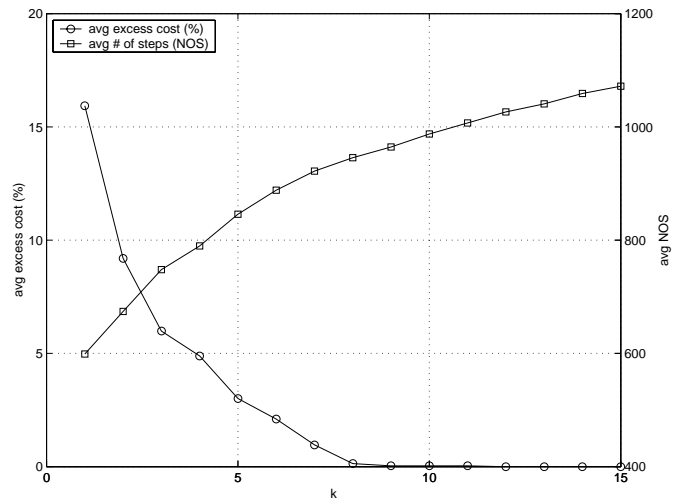


Fig. 5: This figure shows how k parameters affect the performance of the k DCBF algorithm. The results are average of 10 randomly generated mesh graphs of 400 nodes. The x -axis is the k parameters with $k_c = k_d = k$.

ing paths at each intermediate node, and the existence of the other layer doesn't destroy this information. Take the delay constraint to be 800. The LARAC algorithm finds a $p_{LARAC} = (243.3, 481.9)$, the first convex hull point to the right. k LAM returns an improved $p_{kLAM} = (227.6, 551.0)$. The k DCBF algorithm easily reaches the optimal $p_{kDCBF} = (185.1, 800.0)$.

VI. CHOICE OF k

Intuitively, when the graph size increases, the number of paths between nodes increases. We should increase the k parameter correspondingly so that the algorithms keep tracking the paths and are able to find the optimum. Of course, the runtime will also increase. Figure 5 justifies this guess. By increasing k from 1 to 12, the k DCBF algorithm is able to find the optimal paths of 400-node mesh graphs. The increments of NOS with respect to k are much less than the given polynomial bounds.

The results of Figure 5 introduce new concerns about the optimality and complexity of the proposed algorithms. The analysis in previous sections are based on fixed k parameters. For example, $k = 100$ for k LAM and $k_c = k_d = 10$ for k DCBF are used throughout the experiments in last section. These k values are carefully chosen so that they are good for graphs of hundreds of nodes. For graphs with tens of nodes, even $k = 1$ is very likely to find the optimum, But for large graphs with hundreds of thousands of nodes, although running CBF is not practical, there is evidence that $k=100$ is not adequate to find solutions close to the optimum. This leads to the question:

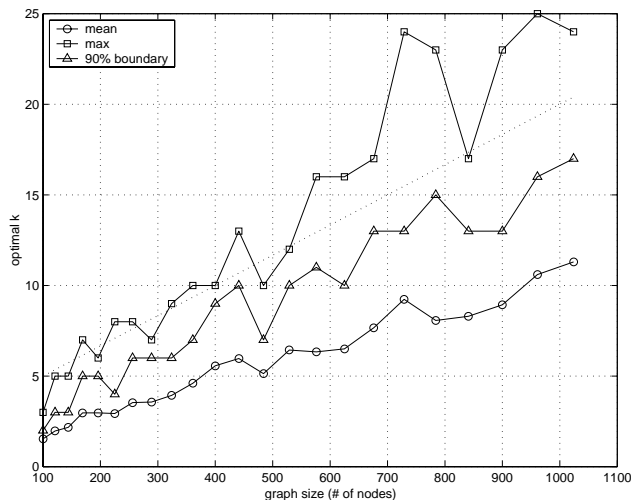


Fig. 6: This figure shows how the graph size affects optimal k parameters of the k DCBF algorithm. The results come from statistics of 30 randomly generated mesh graphs. The y -axis is the optimal k such that $k_c = k_d = k$.

how to choose the k parameter according to the model and size of graphs?

Theoretical analysis can partially answer this question. Graph theory defines a *number of biconnected components* for graphs. A biconnected component is a maximal subgraph that cannot be disconnected by deleting any node. Thus, the number of biconnected components measures the degree of connectivity of a graph. For mesh graphs, this quantity is always one, while for Waxman graphs, it is much larger. According to [14], for 100-node, 175-link Waxman graphs, the number of biconnected components varies from 5 to 30. Generally speaking, the larger the number of biconnected components a graph bears, the fewer the choice of different paths, therefore a small k would suffice. Looking back at the results listed in Table 1, we find that the DCBF algorithm ($k = 1$) works well for Waxman graphs of 400 nodes. k LAM, however, doesn't have a similar behavior. Thus although the number of biconnected components of graphs has implications for the k parameters, there is no explicit relationship between them.

The number of biconnected components says little about the effect of graph size on k , for which we need new measures. Define the *optimal k* to be the smallest k for which the k SP based algorithms find the optimal DCLC paths. Numerical experiments on mesh graphs indicate how the optimal k changes as graph size increases. Figure 6 shows the statistics of the optimal k 's on groups of 30 experiments of different graph sizes. The three plots represent the maximum, 90% boundary and average of the 30 optimal k 's. The result provides guidelines on how to

choose k parameters when using the k DCBF algorithm on mesh graphs. For example, if k follows the triangle-marked line, we expect to find the optimal DCLC paths with a probability of 0.9.

It seems that the optimal k doesn't increase very rapidly with the graph size, at least for graphs of hundreds of nodes. We can easily find a polynomial function that bounds the the optimal k from above. For the example in Figure 6, define a linear function $k = n/60 + 10/3$ (shown as the dotted line above the 90% curve). We can assert that, with a probability of 0.9, the k DCBF algorithm is able to find the optimal DCLC paths on mesh graphs of no more than 1,000 nodes. The time complexity of this specific k DCBF algorithm is then $O(mn^4)$. The same analysis can be applied to k LAM as well as graphs of different models and sizes.

VII. CONCLUSION

In this paper we proposed two new algorithms to solve the DCLC problem. These algorithms, centralized k LAM and distributed k DCBF, combine the k SP algorithms and efficient heuristic methods. The algorithms are polynomial. Although the DCLC problem is NP -complete, numerical experiments suggest that the algorithms almost always find the optimal DCLC paths with suitably chosen k parameters.

We studied the effect of changing k parameters, especially when the graphs have different models and increasing sizes. We introduce an experimental method that suggests how to choose k .

Future work includes extending these algorithms to solve *Multiple Constrained Optimization Routing* (MCOR) problem, where the paths have to satisfy more than one constraints. The extension is not trivial because of the following reason. In DCLC, finding a feasible path only requires solving a shortest delay path problem. In MCOR, however, finding a feasible path requires solving an MCP problem, which is NP -complete by itself. Of course, the k SP algorithms will continue to play important roles in these extensions.

REFERENCES

- [1] D. Eppstein, "Finding the k shortest paths", *SIAM Journal on Computing*, vol. 28, no. 2, pp.652-673, 1998.
- [2] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *W.H. Freeman and Company*, New York, 1979.
- [3] J. Jaffe, "Algorithms for finding path with multiple constraints", *Networks*, vol. 14, pp.95-116, 1984.
- [4] V.M. Jimenez and A. Marzal "Computing the K Shortest Paths: A New Algorithm and an Experimental Comparison", *Lecture Notes in Computer Science*, vol. 1668, pp.15-29, 1999

- [5] A. Juttner, B. Szviatovszki, I. Mecs and Z. Rajko, "Lagrange Relaxation Based Method for the QoS Routing Problem," *Proceedings IEEE INFOCOM*, April 2001.
- [6] F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem "An Overview of Constraint-Based Path Selection Algorithms for QoS Routing", *IEEE Communications Magazine*, vol. 40, no. 12, pp.50-5, Dec 2002.
- [7] W.C. Lee, M.G. Hluchyj and P.A. Humblet, "Routing Subject to Quality of Service Constraints in Integrated Communication Networks", *IEEE Network*, 9(4):14–16, July/August, 1995.
- [8] I. M. Mitchell and S. Sastry, "Continuous path planning with multiple constraints", *IEEE Conference on Decision and Control*, December 2003, to appear.
- [9] C. Pornavalai, G. Chakraborty and N. Shiratori, "Routing with multiple QoS requirements for supporting multimedia applications", *Journal of High Speed Networks*, 1998.
- [10] H.F. Salama, D.S. Reeves and Y. Viniotis, "A Distributed Algorithm for Delay– Constrained Unicast Routing", *IEEE INFOCOM'97*, April 1997.
- [11] Z. Wang and J. Crowcroft, "Quality of service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1228-1234, Sept. 1996.
- [12] B. Waxman "Routing of multipoint connections", *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617-1622, Dec 1988
- [13] R. Widyono, "The Design and Evaluation of Routing Algorithms for Real-Time Channels", *Technical Report TR-94-024*, Tenet Group, Department of EECS, University of California at Berkeley, 1994.
- [14] E.W. Zegura, K.L. Calvert and M.J. Donahoo "A quantitative comparison of graph-based models for Internet topology", *IEEE/ACM Transactions on Networking*, VOL. 5, NO. 6, pp. 770-783 Dec 1997