Abstract—Delay-constrained least cost (DCLC) path problem is a typical delay sensitive quality of service (QoS) routing problem. In this paper, we provide an explicit expression of the time complexity of a previously proposed heuristic \( \text{NR}_{\text{DCLC}} \) when delay and cost are integers. The expression indicates it compares favorably with other heuristics. We further describe a heuristic \( \text{NR}_{\text{DCLC}_{\text{ItoN}}} \) to compute the DCLC paths from one source to all destinations satisfying the delay constraint. Simulations demonstrate that \( \text{NR}_{\text{DCLC}_{\text{ItoN}}} \) can always find delay constrained paths with lower costs than the delay scaling algorithm (DSA), which is so far the best heuristic for such problem. Even though \( \text{NR}_{\text{DCLC}_{\text{ItoN}}} \) has higher execution time than DSA in some cases, it runs much faster than DSA in many other cases such as when the delay constraint has to be strictly guaranteed or when the delay and cost are distributed on large intervals.

I. INTRODUCTION

Delay sensitive routing has been attracting intensive research attention in recent years due to many emerging multimedia applications (teleconferencing, interactive multimedia games, etc.) that have stringent quality of service (QoS) requirements [1]. As one of the most typical delay sensitive routing problems, the NP-hard delay-constrained least cost (DCLC) routing problem [5], i.e., to find a path that has the minimal cost subject to a delay constraint, has been extensively studied and a variety of heuristics have been proposed in the past decade [3], [8], [7].

In one of the earliest works, Hassin [7] proposed an \( \epsilon \)-approximation heuristic which can find a feasible solution with the cost at most \( 1 + \epsilon \) times the optimal cost in time \( O(\log \log UB/LB/mn/\epsilon + \log \log UB/LB) \), where \( m \) and \( n \) are the numbers of links and nodes, respectively, \( UB \) and \( LB \) are the upper and lower bounds of the cost of any feasible path from the source to the destination. Hassin’s heuristic uses an exact algorithm, which can find the shortest path at certain cost from the source to any other node, as a subroutine, and achieves the \( \epsilon \)-approximation by applying some basic scaling and rounding techniques to the cost. Recently, Goel et al. [6] proposed a similar heuristic DSA (delay scaling algorithm), in which they apply the scaling and rounding to the delay instead of the cost. It has been proved that DSA can find a path with a cost no greater than the cost of the path that satisfies the delay upper bound, and the delay of the path is at most \( 1 + \epsilon \) times the delay upper bound. In comparison with Hassin’s heuristic, the time complexity of DSA, which can be expressed as \( O((m + n \log n)E/\epsilon) \), where \( E \) is the length of the longest path in hops, is much lower.

Direct Lagrange relaxation based techniques belong to another class of heuristics that are very effective for solving the DCLC problem. Jütten et al. [8] proposed a heuristic called LARAC based on linear Lagrange relaxation, in which the Dijkstra’s algorithm [2] is executed iteratively to find the shortest path with respect to (w.r.t.) an aggregate weight of a linear combination of delay and cost. The heuristic \( \text{NR}_{\text{DCLC}} \) proposed by Feng et al. [3] is based on a nonlinear Lagrange relaxation technique [9], where a nonlinear function of delay and cost plays a critical role.

Simulations demonstrate that DSA and \( \text{NR}_{\text{DCLC}} \) are probably the best two heuristics for solving the DCLC problem as far as the time complexity and the quality of solution are concerned. However, it is an open question that which of the two heuristics is better. Several differences between them make it hard to have a fair comparison. Firstly, unlike \( \text{NR}_{\text{DCLC}} \), which can only find a solution between the source and a single destination, DSA can find paths from the source to all destinations that satisfy the delay constraint. Secondly, DSA assumes both delay and cost are positive integers, while \( \text{NR}_{\text{DCLC}} \) allows them to be nonnegative real numbers. As such, we can determine the time complexity for DSA, but not for \( \text{NR}_{\text{DCLC}} \). However, it does not mean that we can not find the worst case time complexity of \( \text{NR}_{\text{DCLC}} \) in the case that delay and cost are positive integers. Thirdly, DSA guarantees that the solution of the cost is not greater than the optimal cost with the delay no greater than \( 1 + \epsilon \) times the delay upper bound, but \( \text{NR}_{\text{DCLC}} \) tries to find the best solution with the delay constraint not being violated.

In this paper, we attempt to make a fair comparison of the performance of DSA and \( \text{NR}_{\text{DCLC}} \). We will first briefly review the related work in Section II, and in Section III show the worst case time complexity of \( \text{NR}_{\text{DCLC}} \) when delay and cost are positive integers. Section IV provide a \( \text{NR}_{\text{DCLC}} \) based heuristic to find DCLC paths from one source to all destinations. Performance comparison of the heuristics will be presented in Section V, and Section VI concludes the paper.

II. PRIOR WORK

A. Notations

Suppose that a network is represented by a digraph \( G(V, E) \), where \( V \) is the set of nodes and \( E \) is the set of links. Associated with each link \( e \), there are two non-negative weights, a delay \( d(e) \) and a cost \( c(e) \). We denote the delay and cost of a path \( p \) by \( d(p) \) and \( c(p) \), respectively, which are the sum of the delay and cost of each link along path \( p \).
The DCLC problem is to find a path \( p \) from a source \( s \) to a destination \( t \) such that \( d(p) \leq D \) and \( c(p) \leq C \), where \( D \) is a delay upper bound, and \( q \) is any path from \( s \) to \( t \) that satisfies \( d(q) \leq D \).

As a closely related problem, the delay-cost constrained (DCC) problem is to find a path \( p \) such that \( d(p) \leq D \) and \( c(p) \leq C \), where \( C \) is a cost upper bound.

B. Heuristic H\(_{\text{DCC}}\) based on nonlinear Lagrange relaxation

Proposed in [3] for solving the DCC problem, heuristic H\(_{\text{DCC}}\) uses the following nonlinear cost function to find a feasible path:

\[
g_\lambda(p) = \left( \frac{d(p)}{D} \right)^\lambda + \left( \frac{c(p)}{C} \right)^\lambda
\]

Where \( \lambda \geq 1 \) is a constant. Notice that when \( \lambda = \infty \), \( g_\lambda(p) \) is the maximum of \( d(p)/D \) and \( c(p)/C \). If there exists an algorithm that can find a path \( p \) which exactly minimizes \( g_\lambda(p) \), we must be able to find a feasible path if such a path exists (by setting \( \lambda \) to a large value). However, as the DCC problem is NP-hard [5], one cannot find such an exact algorithm that has polynomial time complexity.

H\(_{\text{DCC}}\) is designed in a way to approximate the process of finding a feasible solution using the nonlinear cost function \( g_\lambda(\cdot) \). It includes two basic procedures, Reverse_Dijkstra and Look_Ahead_Dijkstra. Reverse_Dijkstra runs a modified Dijkstra’s algorithm in reverse direction to find a post-path from any node to the destination \( t \) by using a linear cost function \( g_1(\cdot) \) \((g_\lambda(\cdot) \text{ with } \lambda = 1)\). Correspondingly, Look_Ahead_Dijkstra runs the Dijkstra’s algorithm in forward direction to find a pre-path from the source \( s \) to any other node by using a nonlinear cost function \( g_\lambda(\cdot) \) \((\lambda > 1)\). Since both procedures calculate the accumulated delay and cost of each post-path or pre-path, H\(_{\text{DCC}}\) may find a feasible solution if at any intermediate node the concatenation of the pre-path and the post-path is feasible.

C. Heuristic H\(_{\text{DCC}}\) Improved

Our recent research results [4] indicate that we can significantly improve H\(_{\text{DCC}}\)’s probability of finding feasible solutions by making some changes on the relaxation procedures.

D. Heuristic NR\(_{\text{DCLC}}\)

NR\(_{\text{DCLC}}\) solves the DCLC problem by using H\(_{\text{DCC}}\) (or H\(_{\text{DCC}}\) Improved) as a basic step. As illustrated by the pseudocode in Fig. 5, it starts by checking the least cost (LC) path \( q \) using a regular Dijkstra’s algorithm. If \( q \) is feasible, it must be an optimal solution. Otherwise, the least delay (LD) path \( p \) is checked, and if it is infeasible, no feasible path can be found.

The resulting improved version, called H\(_{\text{DCC}}\) Improved, has the same time complexity as H\(_{\text{DCC}}\).

The top-level description of the heuristic H\(_{\text{DCC}}\) Improved is shown in Fig. 1. Similar to H\(_{\text{DCC}}\), the improved version also runs a modified Dijkstra algorithm at most twice, one in reverse direction and the other, if necessary, in forward direction.

The relaxation procedures of Reverse_Dijkstra and Look_Ahead_Dijkstra are shown in Figs. 2 and 3, respectively. Fig. 4 describes the preference rule when choosing the predecessor of a node in Look_Ahead_Dijkstra.

The notations are defined as follows: \( R_d[u] \) and \( R_c[u] \) are the accumulated delay and cost of the post-path from \( v \) to \( t \), respectively; \( \pi_r[v] \) is the predecessor of \( v \); and \( r[v] \) is the value of the cost function \( g_\lambda(\cdot) \) \((\lambda = \infty)\) of the post-path. Correspondingly, \( G_d[v] \) and \( G_c[v] \) are the accumulated delay and cost of the pre-path from \( s \) to \( v \), respectively; \( \pi_g[v] \) is the predecessor of \( v \); and \( g[v] \) is the value of the cost function \( g_\lambda(\cdot) \) \((\lambda = \infty)\) of the pre-path. Notations \( d(u, v) \) and \( c(u, v) \) are respectively the delay and cost of the link from \( u \) to \( v \).

The major differences between H\(_{\text{DCC}}\) Improved and H\(_{\text{DCC}}\) lie in the following two aspects. First, in Look_Ahead_Dijkstra of H\(_{\text{DCC}}\) Improved, the predecessor of a node is determined by evaluating the cost function \( c(p) - \delta \) to be the cost upper bound \( C \). \( \delta \) is a small positive number such that no path between \( s \) and \( t \) has a cost on interval \([c(p) - \delta, c(p)]\). Heuristic H\(_{\text{DCC}}\) is then employed to solve the DCC problem.

---

**Fig. 1.** The heuristic algorithm H\(_{\text{DCC}}\) Improved for the DCC problem.

**Fig. 2.** The relaxation procedure of subroutine Reverse_Dijkstra.

**Fig. 3.** The relaxation procedure of subroutine Look_Ahead_Dijkstra.

**Fig. 4.** The preference rule.
DCLC by keeping in mind that we need to find the DCLC, we need to know if the constant $c(p) = D$ works as follows: Improved)

Obviously if a feasible solution $r$ is found by $H_{DCC}$, $r$ must have a lower cost than $p$. Thus, we can replace the current solution $p$ by $r$, and repeat this procedure until no feasible solution can be found by $H_{DCC}$.

III. THE TIME COMPLEXITY OF NR_DCLC WITH DELAY
AND COST BEING INTEGERS

Before we discuss the time complexity of NR_DCLC, we have to understand how the $H_{DCC}$ (or $H_{DCC,Improved}$) finds a feasible solution using the nonlinear Lagrange relaxation. Fig. 6 illustrates the distribution of paths in a $x - y$ plane, in which the horizontal and vertical coordinates represent delay and cost normalized w.r.t. $D$ and $C$, respectively. Each small circle represents a path. $p_d$ and $p_c$ are the LD and LC paths, respectively. Apparently the box with bold edges represents the feasibility region. Each dash-dotted contour line represents a set of paths that have the same value for the nonlinear cost function $g_X(\cdot)$ with $\lambda = \infty$. The search process of $H_{DCC}$ is like incrementally pushing the contour line along the direction of the arrow and returning the first path hit by the contour line. In this case, path $X$ would be returned.

To find the time complexity of NR_DCLC, we need to know the maximum number of times to run $H_{DCC}$, or the worst case of the distribution of the feasible paths. Fig. 7 shows three typical cases of the distribution of feasible paths. Supposing all feasible paths are distributed along the dotted line $AB$, the path returned in the first iteration would be $X$. In the next iteration, $c(X) - \delta$ would be set to the upper bound of the cost. This procedure is repeated until the optimal solution is found. In the case that all feasible paths are distributed along $AC$ or $AD$, the search process is quite similar. However, in the case that in each iteration the feasible paths are always distributed along the diagonal line $AC$, $H_{DCC}$ would take the maximal number of iterations before locating the optimal solution. This is because in such case the remaining feasibility region that has not been checked (the shaded area) is the maximal. Therefore, in the worst case the width and length of the feasibility region will each be reduced by half after each run of $H_{DCC}$. When both delay and cost are integers, this would take $\min \{ \log(D - d(p_d)), \log(c(p_d) - c(p_c))\}$ times before finding the optimal solution.

As $H_{DCC}$ runs a modified Dijkstra’s algorithm at most twice, its time complexity is $O(m + n \log n)$. Therefore, the worst case running time of NR_DCLC is $O(\min \{ \log(D - d(p_d)), \log(c(p_d) - c(p_c))\}) (m + n \log n)$. Obviously, the time complexity of DSA or Hassin’s $\epsilon$-approximation heuristic (see Section I) would be much higher than that of NR_DCLC if the constant $\epsilon$ is given a small value.

IV. HEURISTIC NR_DCLC_1TO1N FOR COMPUTING DCLC
PATHS FROM ONE SOURCE TO ALL DESTINATIONS

In this Section we describe a heuristic called NR_DCLC_1toN to find the delay constrained paths from the source to all the destinations that satisfy the delay constraint. NR_DCLC_1toN is essentially modified from NR_DCLC by keeping in mind that we need to find the DCLC paths for multiple destinations. However, it is more than a simple application of NR_DCLC to multiple source-destination pairs, which would unnecessarily waste a lot of time.

Heuristic NR_DCLC_1toN works as follows:

1. Find the LD tree rooted at $s$. Let $S$ be the set of nodes satisfying the delay constraint. $\forall v \in S$, let the LD path from $s$ to $v$ be the current solution for node $v$, and let $D(v)$ and $C(v)$ be the delay and cost of its current solution.
2. Find the LC tree rooted at $s$. Let $I(v)$ be an indicator if the final solution for node $v \in S$ has been determined. Set the initial value of $I(v)$ to $true$ and let the LC path be the current solution of $v$ if the delay of the LC path from $s$ to $v$ is not greater than $D$; otherwise, set $I(v)$ to $false$. 
3. Repeat
   (1) Find node \( v \in S \) whose \( I(v) = false \) and \( D - D(v) \) is maximal.
   (2) Repeat
      (i) \( r \leftarrow H_{DCLC}^{improved}(G, s, v, d, c, D, C(v) - \delta) \).
      If \( r \neq NULL \), continue with steps (ii) and (iii).
      (ii) Let \( r \) be the current solution for node \( v \). Let \( C(v) = c(r) \).
      (iii) Update the solution of \( u \in S \) if the path from \( s \) to \( u \) found by \( Look\_Ahead\_Dijkstra \) is feasible with lower cost than the current solution of \( u \); Update \( D(u) \) and \( C(u) \) as well.
   Until \( r = NULL \).
   (3) Let \( I(v) = true \).
   Until all nodes in \( S \) have been processed.

Several techniques have been used to reduce the time complexity. First of all, as all destinations have the same upper bound, it is very likely that for some destinations the constraint is so loose that even the LC paths may satisfy the constraint. In such case, the LC paths would be the optimal solutions for those destinations. Secondly, in our implementation, \( Reverse\_Dijkstra \) is stopped and returned to \( H_{DCLC}^{improved} \) once the source node is permanently checked (i.e., without necessarily checking every node). Similarly, \( Look\_Ahead\_Dijkstra \) is stopped once the destination node is permanently checked. Thirdly, each time after the execution of \( Look\_Ahead\_Dijkstra \), it may find a path from \( s \) to a node \( u \in S \) which is better than the current solution associated with \( u \). In that case, the solution of \( u \) will be updated.

Notice that the \( LD \) tree and the \( LC \) tree each can be found by a single invocation of \( Dijkstra \)’s algorithm. The worst case time complexity of \( NR\_DCLC\_1toN \) should be \( |D| \) times the time complexity of \( NR\_DCLC \), where \( |D| \) is the number of destinations satisfying the delay constraint.

V. PERFORMANCE EVALUATION

In this section we compare the performance of \( NR\_DCLC\_1toN \) and DSA. As mentioned earlier, DSA guarantees that the solution has a delay not greater than \((1 + \epsilon)/D\), while \( NR\_DCLC\_1toN \) guarantees the delay constraint is not violated. To achieve a fair comparison, we reset the delay upper bound to be \( D/(1 + \epsilon) \) each time when \( DSA \) is executed. Three following performance measures will be investigated in our experiments: the cost deviation in percentage of the solution of a heuristic from the solution of an exact algorithm (e.g., Algorithm A described in [7]), the execution time of \( NR\_DCLC\_1toN \) or DSA for finding the delay constrained paths from one source to all destinations, and the execution time of \( NR\_DCLC \) for finding a solution between a single source-destination pair.

The execution time is collected when all the heuristic is run on a Pentium IV based PC. The cost deviation is calculated by

\[
\frac{1}{|S|} \sum_{v \in S} \frac{C_{heuristic}(v) - C_{exact}(v)}{C_{exact}(v)},
\]

where \( S \) is the set of nodes satisfying the delay constraint, \( C_{heuristic}(v) \) is the cost of the solution found by a heuristic for node \( v \), and \( C_{exact}(v) \) is the cost of the corresponding exact solution.

An important factor in DSA is \( \tau_0 \), an initial value for scaling the delay. The author of [6] only mentioned that \( \tau_0 \) should be far less than \( D \). In order to see the impact of \( \tau_0 \) on the performance, we test two cases with \( \tau_0 = 0.05D \) and \( \tau_0 = 0.15D \), respectively. The results corresponding to the two cases are denoted in each figure by “DSA (0.05)” and “DSA (0.15)”, respectively.

Three sizes of network topologies, 50-, 100-, and 200-node, are used in our experiments, and they are generated by the BRITE topology generator [10], which is being extensively used to simulate real communication networks. The delay and cost are uniformly distributed either on \([1, 1000]\) or \([1, 10000]\). Each point in a figure presented in this Section is obtained based on the statistical results of processing 5,000 ~ 20,000 (depending on the network size) routing requests, which are generated under various combinations of network topologies and link- weight distributions. The delay upper bound \( D \) is set to be the maximal delay from the source to any other node multiplied by a coefficient called \( UB\_factor \in (0, 1) \). The error bars shown in the figures represent the standard error of the mean. Notice that even though the error bars for the cost deviation in some figures span to negative values, it does not mean that the solution of a heuristic can have a lower cost than the exact solution.

Fig. 8 shows the performance measures when \( UB\_factor = 0.5 \), \( \epsilon = 0.05 \), and both delay and cost are uniformly distributed on \([1, 1000]\). We can see that in this case \( NR\_DCLC\_1toN \) can achieve lower cost deviation than DSA in less execution time regardless of the network size and the choice of \( \tau_0 \) in DSA. As expected, the execution time of \( NR\_DCLC \) is extremely low when compared with that of DSA.

Fig. 9 shows the results when \( \epsilon = 0.1 \). Comparing Figs. 8a and 9a, it is not surprising to see the cost deviation of DSA becomes higher with a larger value for \( \epsilon \). This is due to the lower requirement for the cost of the solution. Accompanying with this is the lower computation time, as we can see by comparing Figs. 8b and 9b. Fig. 10 shows the result for an extreme case where we let \( \epsilon = 0.02 \). With such a strong requirement on the cost, the running time of DSA increases dramatically. The corresponding cost deviation is quite low, but still worse than that of \( NR\_DCLC\_1toN \).

By comparing Figs. 8b, 9b, and 10b, we have reached another interesting conclusion: \( \tau_0 \) should be increased with the decrease of \( \epsilon \) in order to achieve a lower running time for heuristic DSA. One should also notice that the performance measures of \( NR\_DCLC\_1toN \) maintains consistently similar for all these cases.

Figs. 11 and 12 are the results when \( UB\_factor \) equals to 0.8 and 0.2, respectively, while other parameters take the same values as in Fig. 8. Comparing the execution times demonstrates that \( UB\_factor \) has nearly no impact on the execution time of DSA, but does affect \( NR\_DCLC\_1toN \) (and \( NR\_DCLC \)). This complies with the expressions of the time complexities of DSA and \( NR\_DCLC\_1toN \); The upper bound has nothing to do with the time complexity of DSA, but it determines how many times the Dijkstra’s algorithm should be run in \( NR\_DCLC \), and the number of destinations that need to be processed in \( NR\_DCLC\_1toN \).
Fig. 8. Performance measures when $UB\_factor = 0.5$, $\epsilon = 0.05$, and $d \sim [1, 1000]$, $c \sim [1, 1000]$.

Fig. 9. Performance measures when $UB\_factor = 0.5$, $\epsilon = 0.1$, and $d \sim [1, 1000]$, $c \sim [1, 1000]$.

Fig. 10. Performance measures when $UB\_factor = 0.5$, $\epsilon = 0.02$, and $d \sim [1, 1000]$, $c \sim [1, 1000]$.

Fig. 13 shows the results when both delay and cost are distributed on $[1, 10000]$, while other parameters take the same values as in Fig. 8. Comparing Figs. 8b and 13b indicates that the distribution of delay and cost has no impact on the execution time of NR\_DCLC\_1toN, but considerably changes that of DSA regardless of the choice of $\tau_0$.

VI. CONCLUSIONS

In this paper, we provide an explicit expression of the worst case time complexity of heuristic NR\_DCLC when both delay and cost are integers. This expression indicates it is comparable to other heuristics for solving the DCLC problem. We further propose a heuristic NR\_DCLC\_1toN to find the DCLC paths from one source to all destinations satisfying the delay constraint. Simulations demonstrate that NR\_DCLC\_1toN can always achieve lower cost deviation than DSA, which is probably the best heuristic for such problem in the literature. While the running time of NR\_DCLC\_1toN is slightly higher than DSA in some cases, it is much lower in many other cases such as when the delay constraint must be strictly guaranteed or when the delay and cost are distributed on large intervals to achieve a fine distinction of the performance measures.
Fig. 11. Performance measures when $U_{factor} = 0.8$, $\epsilon = 0.05$, and $d \sim [1, 1000]$, $c \sim [1, 1000]$.

Fig. 12. Performance measures when $U_{factor} = 0.2$, $\epsilon = 0.05$, and $d \sim [1, 1000]$, $c \sim [1, 1000]$.

Fig. 13. Performance measures when $U_{factor} = 0.5$, $\epsilon = 0.05$, and $d \sim [1, 10000]$, $c \sim [1, 10000]$.

REFERENCES


