Upgrading Links For Performance *

Andrew Lim[†¶] Yeow Meng Chee[‡] Wynne Hsu[¶]

[†] Information Technology Institute, 71 Science Park Drive, S0511
[¶] DISCS, National University of Singapore
[‡] Dept of Comp Sci, University of Waterloo, Canada

Abstract

The performance of a computer network is commonly measured by the maximum minimum time required to move certain amount of data between any 2 nodes in the network. Due to the advances in technology, links in the network may now be upgraded, for instance to optical fibre links, so that better performance can be achieved. In this paper, we study the LINK UPGRADE problem for networks. We first show that the LINK UPGRADE problem is $\mathcal{NP}$-complete. We also show that, a closely related problem, the MINIMUM COST LINK UPGRADE problem is $\mathcal{NP}$-complete even if the underlying topology of the network is a linear array. However, for certain classes of networks, the LINK UPGRADE problem can be solved in polynomial time. For general networks, we provide effective heuristics for the above problems.

1 Introduction

In recent years, advances in very large scale integration (VLSI) technology have resulted in smaller and more powerful microprocessors that are relatively inexpensive. As a result there is a move from large centralized computers towards many smaller decentralized ones. This proliferation of small autonomous machines has increased the demand for data communications between computers. Computer networks provide the capability of interconnecting these small machines within a geographical area.

Current developments in integrated services digital network (ISDN) require computer networks to transfer large amounts of data with minimum delay in order to effectively and efficiently support various voice, digital data, text, and image applications. The performance of a network becomes an important issue. If an existing network does not meet the required performance, we can upgrade some or all of its links in order to improve the network's performance. Optical fibers are usually used to upgrade these links since they offer very large bandwidths.

In this paper, we consider the link upgrade problem. Given a network and a specified performance, the link upgrade problem is to determine which links of the network are to be upgraded so that the specified performance of the network is attained. In subsequent sections, we will formulate the LINK UPGRADE problem and the MINIMUM COST LINK UPGRADE problem, study their computational complexities, propose exact algorithms for special cases of the link upgrade problem and provide effective heuristics for the general link upgrade and minimum cost link upgrade problem.

2 Problem Formulation

We model the topology of a computer network as a *weighted graph*. A graph $G = (V, E; w)$ is a set V of n vertices which represent the communication centers or concentrators in the network, together with a set E of m undirected edges representing bidirectional communication links. If an edge e is incident with vertices u and v, we will sometimes write e as the unordered pair (u, v). This graph incorporates information about the network's topology, but does not include information about the characteristics of the links. A weighted graph has, in addition, a weighting function $w : E \rightarrow \mathbf{Z}_{\geq 0}$. The value $w(e)$ is called the *weight* of edge e.

Let $G = (V, E; w)$ be a weighted graph modeling the network $\mathcal{N}$ with $w(e) = P/R(e)$ for each $e \in E$ where $R(e)$ is the data rate for link e and P is the amount of data to be sent across link e. A source vertex s can transmit data with size P to a target vertex t along any s, t-path in G, and the shortest possible time in which this can be done is given by the length of the shortest s, t-path. The length of the shortest s, t-path in G is commonly called the *distance* between s and t in G, denoted $dist_G(s, t)$. The performance measure (to send data of size P) we use for a network $G = (V, E)$ is its *diameter*:

$$D(G) = \max_{s,t \in V} \{dist_G(s, t)\}.$$

Intuitively, $D(G)$ is a guarantee on the speed of the network G; any transfer of a packet of data with size P between any two vertices in G can be done in no more than $D(G)$ units of time.

*This research was supported in part by the National University of Singapore Research Grants RP940643 and RP940644

IEEE Catalogue No. 95TH8061
0-7803-2579-6/95/$4.00©1995IEEE

Suppose now we are given a network $G = (V, E; w)$ and optical fibers whose data rate is R. Let $c = P/R$. We assume that the data rate R is so large that P/R is negligible when compared to other links in the network G. Therefore, upgrading a subset S of the links in E to the available optical fibers simply means transforming G into a new network $G' = (V, E; w')$, where

$$w'(e) = \begin{cases} w(e), & \text{if } e \in E \setminus S; \\ 0, & \text{if } e \in S. \end{cases}$$

We may assume without loss of generality that all weights are integers, by normalizing, if necessary. We can now formally state the LINK UPGRADE problem.

LINK UPGRADE
INSTANCE: A weighted graph $G = (V, E; w)$ and a nonnegative integer D.
QUESTION: Find a smallest (in cardinality) subset $S \subseteq E$ such that the weighted graph $G' = (V, E; w')$, where

$$w'(e) = \begin{cases} w(e), & \text{if } e \in E \setminus S; \\ 0, & \text{if } e \in S, \end{cases}$$

has diameter $D(G') \leq D$.

If the cost of upgrading each link differs and is given by $C(e)$, and the objective is to find a subset of edges with the minimum cost to be upgraded such that the $D(G') \leq D$, we name this problem, MINIMUM COST LINK UPGRADE problem:

MINIMUM COST LINK UPGRADE
INSTANCE: A weighted graph $G = (V, E; w, C)$, a nonnegative integer D, and positive integer K.
QUESTION: Is there a subset $S \subseteq E$, $\sum_{e \in S} C(e) \leq K$, such that the weighted graph $G' = (V, E; w', C)$, where

$$w'(e) = \begin{cases} w(e), & \text{if } e \in E \setminus S; \\ 0, & \text{if } e \in S, \end{cases}$$

has diameter $D(G') \leq D$?

3 Computational Complexity

Theorem 1 *The decision version of the LINK UPGRADE problem is $\mathcal{NP}$-complete.*

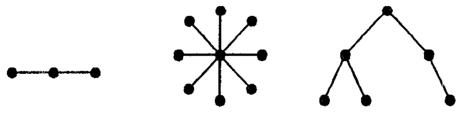
Proof: See [7]. □

Theorem 2
The MINIMUM COST LINK UPGRADE problem is $\mathcal{NP}$-complete even for linear array.

Proof: See [7]. □

4 Exact Algorithms for Restricted Classes

The following are some common network topologies [8].


(i)Linear Array (ii)Star (iii)Tree

- The *linear array*, in which nodes are connected linearly.
- The *star*, in which every node or communication site is connected to a single central communication node, the *hub* of the star.
- The *tree*, in which the path between any 2 nodes in the network is unique. The linear array and star topologies are restricted forms of trees.

4.1 The Linear Array and Star

Let $G = (V, E; w)$ and D, be an instance of LINK UPGRADE problem, where G is a linear array of n vertices, and let S be a solution to this instance. A linear array has the property that its diameter is the sum of all its edge weights. So, obviously, the greedy method which works iteratively by selecting at each step an edge of highest weight among those not in S and putting it in S until $D(G' = (V, E; w')) \leq D$, solves the problem. Since the sorting step to reorder the edge weight takes $O(n \log n)$ time and all the edge selections and diameter computations and recomputations takes $O(n)$ time, the overall time complexity of the greedy algorithm is $O(n \log n)$.

The diameter of the star is the sum of its 2 highest edges. The same approach used for the linear array can be applied to the star. The complexity remains the same, which is $O(n \log n)$.

4.2 Trees

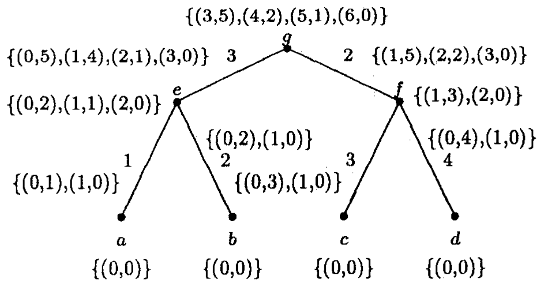
If the graph given $G(V, E; w)$ is a tree (i.e. the path between any 2 nodes in the graph is unique), the dynamic programming approach may be used to obtain the smallest number of upgrades given the required diameter D. The details of the algorithm are given in Figure 1.

Our method begins with the leaves of the tree. At a leaf node, the maximum distance from its subtrees of the leaf is 0, i.e. there is actually no subtree. For each node, we keep an ordered list of pairs (α, β), where α is the number of upgrades that has been carried out and β is the farthest node in the current node's subtrees from the node. For each leaf, the list of ordered pairs is initialized to $\{(0, 0)\}$.

An edge has similar ordered pairs. Given the ordered pairs list L of the child node of the edge and the edge cost c, the list of ordered pairs associated with each edge can be computed using the procedure *Edge_Comb(c, L)*. The root of the subtrees obtains its list of ordered pairs by merging the lists of its

```
Procedure $UP\_TREES(T)$
  /*Return a list of possible optimal upgrades
  and their resultant diameters. */

begin
  if $leaf(T)$ return $\{(0,0)\}$
  else
  begin
    // $left(T)$ returns the left subtree
    $L := UP\_Tree(left(T))$;
    // $right(T)$ returns the right subtree
    $R := UP\_Tree(right(T))$;
    $L_e := Edge\_Comb(left\_edge\_cost(T), L)$;
    $R_e := Edge\_Comb(right\_edge\_cost(T), R)$;
    return $Node\_Comb(L_e, R_e)$;
  end;
end;

Procedure $Edge\_Comb(c, L)$
  /*Procedure takes in a integer value $c$ (the edge cost)
    and a list $L$ and returns a new list $L'$ that
    considers if the edge is up-graded. */

begin
  $L' := \emptyset$;
  for each element $e = (\alpha, \beta) \in L$
  begin
    $e_1 := (\alpha, \beta + c)$;
    $e_2 := (\alpha + 1, \beta)$;
    add $e_1$ and $e_2$ to $L'$ and
    eliminate all infeasible and suboptimal entries;
  end
  return $L'$;
end

Procedure $Node\_Comb(L, R)$
  /*Procedure combines 2 lists $L$ and $R$. */

begin
  $L' := \emptyset$;
  for each element $e_L = (\alpha_L, \beta_L) \in L$ & $e_R = (\alpha_R, \beta_R) \in R$
  begin
    $e := (\alpha_L + \alpha_R, \max\{\beta_L, \beta_R\})$;
    if $\beta_L + \beta_R \leq D$ then
      add $e$ to $L'$ and eliminate suboptimal entries;
  end
  return $L'$;
end
```

Figure 1: Algorithms for link-upgrades for binary trees



A Tree, the required diameter is $D \leq 5$

Figure 2: An example of our approach

branches (edges) to its subtrees using the procedure $Node\_Comb(L, R)$ ($UP\_TREE(T)$ algorithm assumes T to be a binary tree. In general, it works for all trees, but binary trees will give an $O(n^2)$ time complexity).

An entry (α, β) is *infeasible* if, as a result of not upgrading the edge, the maximum distance to the leaves is greater than D (in *Edge_Comb*) or when merging 2 entries, the sum of distance from the furthest leaf in the left subtree and right subtree is greater than D (in *Node_Comb*).

A merged pair (α_j, β_j) is *sub-optimal* if there exists a merged pair (α_k, β_k) such that ($\alpha_k \leq \alpha_j$ and $\beta_k < \beta_j$) or ($\alpha_k < \alpha_j$ and $\beta_k \leq \beta_j$). Sub-optimal pairs are not needed for the optimal solution.

Using the example in Figure 2, the 4 merged pairs in f are given by :

$(0,3)$ and $(0,4)$	$\rightarrow$	not feasible as $3+4 > D = 5$
$(0,3)$ and $(1,0)$	$\rightarrow$	$(1,3)$
$(1,0)$ and $(0,4)$	$\rightarrow$	$(1,4)$ eliminated (suboptimal)
$(1,0)$ and $(1,0)$	$\rightarrow$	$(2,0)$

So, the f's list is $\{(1,3),(2,0)\}$. For the edge (g,f), the decision is whether to upgrade the link (g,f) :

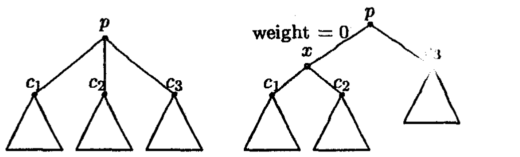
$(1,3)$ and (g,f) not upgraded	$\rightarrow$	$(1,5)$
$(1,3)$ and (g,f) upgraded	$\rightarrow$	$(2,3)$ subopt
$(2,0)$ and (g,f) not upgraded	$\rightarrow$	$(2,2)$
$(2,0)$ and (g,f) upgraded	$\rightarrow$	$(3,0)$

The final result is $\{(1,5),(2,2),(3,0)\}$.

The process of deriving the ordered pair list for each node and edge continues until the root is reached. The pair with the smallest α gives the solution. During the course of creating (α, β) pairs, the edges that are upgraded are stored with the ordered pairs so that these edges can be retrieved and given as part of the solution when the algorithm terminates.

Theorem 3 *The number of ordered pairs at each node or edge is at most $min(n, D+1)$ where n is the number of nodes in the tree and D the diameter required.*

Proof: If a tree has n nodes, it has $n-1$ links. Let α be the number of edges that have been upgraded so



(i)Non binary tree (ii)Transformed to a binary tree

Figure 3: An Example to illustrate the transformation of trees into binary trees

far. α can take values from 0 to $n-1$. Hence, there can be only n distinct αs, if all suboptimal pairs are eliminated. Similarly, with the required diameter of no more than D, β, the maximum distance to the leaves can take values from 0 to D. Hence, there are at most $D+1$ unique values of β, as $\beta \in \mathbf{Z}_{\geq 0}$. Combining both constraints, the number of ordered pairs at each node is at most $min(n, D+1)$. □

Theorem 4 *The number of steps required to produce the list of ordered pairs for any edge e is 2n, where n is the number of nodes in the subtree connected to e.*

Proof: Let r be the root of the subtree connected to e. e's list may be derived from r's list by simply considering if the link e is to be upgraded or not. For each ordered pair in r, 2 pairs are produced of which some may be infeasible or suboptimal. Since r has n nodes, from Theorem 3, there can be at most n number of ordered pairs. This implies that e will produce at most $2n$ ordered pairs, which at most $n+1$ will remain after eliminating suboptimal and infeasible solutions. □

Our method works only for binary trees. For k-nary trees, the number of ordered pairs produced at node p prior to removal of suboptimal or infeasible solutions is the product of the sizes of the lists of ordered pairs of the edges linking the subtrees to p. In the worst case, the time complexity is $O((\frac{n}{k})^k)$. This happens when all the subtrees have equal size; n is the total numbers of nodes in the tree. If $k = \frac{n}{2}$, then the time complexity for producing the ordered pairs for p with n nodes in its subtree becomes $O(2^n)$.

We can transform all non-binary trees to binary trees with the addition of dummy nodes and edges with weight 0. This transformation can be seen in Figure 3. The size of the new tree is at most twice the size of the original tree and the transformation can be done easily in $O(n^2)$ time.

Theorem 5 *The time complexity of deriving the ordered pairs for all nodes in a binary tree of size n is* $O(n^2)$.

Proof: The time required at each node in the binary tree to produce the its ordered pairs is at most $(\frac{m}{2})^2$, where m is the total number of nodes in the 2 subtrees of the node. The total number of steps is maximize if the tree is balanced; i.e. subtrees of every node has almost equal number of leaves. The number of steps required is :

$$(\frac{n}{2})^2 + 2(\frac{n}{4})^2 + 4(\frac{n}{8})^2 + \ldots + (\frac{n}{2})(\frac{n}{n})^2 \leq n^2$$

Therefore, the time complexity is $O(n^2)$. □

Theorem 6 *The time complexity for the LINK UPGRADE problem for trees is* $O(n^2)$.

Proof: Follows from the above discussions. □

Heuristic H_1
// For H'_1 we use $\frac{\mathcal{F}(G(V,E))-\mathcal{F}(G(V,E-e))}{C(e)}$ in Step 3

Step 1: Find the shortest path between all pairs of vertices.

Step 2: Remove all redundant edges.

Step 3: Select the edge, e, in the graph $G(V,E)$ with the largest $\mathcal{F}(G(V,E))-\mathcal{F}(G(V,E-e))$ such that:

$$\mathcal{F}(G(V,E)) = \sum_{\forall e \in E \ \& \ dist(e)>D} dist(e)$$

Step 4: If $\mathcal{F}(G(V,E-e)) = 0$ then goto Step 9.

Step 5: Upgrade the edge $e = (u,v)$ by contraction of e.

Step 6: Update the graph G due the Step 5.

Step 7: Update the shortest path matrix of the vertices.

Step 8: Goto Step 2.

Step 9: End.

Figure 4: Heuristics H_1 and H'_1

5 Heuristics for General Networks

For general networks, the link upgrade problem is $\mathcal{NP}$-complete. The situation is even worse for the minimum cost link upgrade problem, which is $\mathcal{NP}$-complete even for linear list/array. As a result, an effective heuristic H_1 is proposed for the link upgrade problem. A variant of the heuristic H_1, which is H'_1 is proposed for the minimum cost link upgrade problem. Both H_1 and H'_1 can be found in Figure 4.

Step 1, finds the all pairs shortest path. This can be done in O(n^3). In Step 2, the algorithm compares the edge weight $w(e)$ and $dist(e)$. If $dist(e) < w(e)$ then this edge is redundant and can be removed (note that remove is *not* the same as upgrade). This takes at most O($|E|$) time. Step 3, finds the edge e, where its upgrade results in the largest $\mathcal{F}(G(V,E))-\mathcal{F}(G(V,E-e))$. This implies that all pairs shortest path must be recomputed or updated. To update the all pairs shortest path matrix, let us take a look at the shortest path between any

$\|V\|$	$\|E\|$	D	$D' = 0.75D$			$D' = 0.5D$		
			D'	H_0	H_1	D'	H_0	H_1
10	23	42	31	2	2	21	5	3
25	156	38	29	3	1	19	11	4
50	606	20	15	12	4	10	36	11
75	1384	20	15	22	7	10	50	12
100	2487	15	11	57	8	8	101	15

Table 1: Test set 1 (density=0.5, max cost=50)

$\|V\|$	$\|E\|$	D	$D' = 0.75D$			$D' = 0.5D$		
			D'	H_0	H_1	D'	H_0	H_1
25	63	98	74	9	3	49	16	5
50	223	59	45	7	2	30	18	8
75	517	41	31	22	4	21	55	13
100	946	34	26	40	6	17	44	12

Table 2: Test set 2 (density=0.2, maxcost= 50)

2 vertices x and y. With the upgrade of $e = (u, v)$, the minimum distance between them becomes,

$$dist(x,y) = \min\{ \quad dist(x,y), dist(x,u) + dist(v,y), \\ dist(x,v) + dist(u,y) \quad \}$$

This is done for every edge and there can be at most $O(|E|)$ edges. The time required for Step 3 is at most $O(|E||V|^2)$. $\mathcal{F}(G(V, E - e)) = 0$ implies that the diameter $\leq D$, hence the algorithm terminates (see Step 4). Steps 5-9 are quite self explanatory. The entire process will be repeated at most $|V| - 1$ times, i.e. every upgrade decrease the number of vertices in the graph by 1, the entire algorithm takes at most $O(|E||V|^3)$ time. In practice the algorithm's running time is proportional to the cube of its input size.

6 Experimental Results

In order to test the effectiveness of our heuristic, H_1, we implemented a simple greedy algorithm, H_0, that selects the first kth largest edges to be upgraded such that the diameter of the graph is no more than D. k is made as small as possible. This can be done by first sorting the edges according to their weights and using binary search to determine k. A simple implementation takes at most $O(n^3 \log n)$ time.

We tested H_0 and H_1 on graphs that are randomly generated. The density, d, of a graph is defined as the probability that an edge between any 2 vertices exists. The results are summarized in Table 1 and Table 2. Column 1 is the number of vertices in the graph and column 2, the number of edges. Column 3, D, is the current diameter. The first row of column 4 indicates the desired diameter, D', of the graph as a percentage with respect to the original diameter. For instance, take the graph with 10 nodes in Table 1, the original diameter is 42, if the desired diameter is 75% of the original, we would like to use the smallest number of upgrades such that the diameter of the graph after the upgrade is 75% of 42 which is approximately 31. Column 5 has similar interpretation as column 4. The first sub-column of column 4 and column 5 are the desired diameter D's and the second and third sub-columns of column 4 and column 5 are the number of upgrades needed by the greedy approach H_0 and H_1 respectively.

As you can see from the experimental results, only small number of edges need to be upgraded to improve the performance of the network by 100% in many instances.

7 Conclusion

We have defined and studied the LINK UPGRADE problem and the MIN COST LINK UPGRADE problem. We have proposed fast algorithms for restricted cases of the link upgrade problem and have shown that the problem is $\mathcal{NP}$-complete for general graphs. As for the minimum cost link upgrade problem, we have shown that it is $\mathcal{NP}$-complete even for linear lists. As a result, we have developed effective heuristics for both problems. Our experiment results show that our heuristic greatly outperformed the simple greedy method for the LINK UPGRADE problem. The link upgrade problem and its variations find applications not only in computer networks but also in road and transportation networks [6].

References

[1] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, **1** (1959) 269–271.

[2] R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, **5** (1962) 345.

[3] 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.

[4] R.M. Karp. Reducibility among combinatorial problems. In: R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85–103, 1972.

[5] T. Li. Advances in optical fiber communications: An historical perspective. *IEEE Journal on Selected Areas in Communications*, **SAC-1(3)** (1983) 356–372.

[6] A. Lim. Minimum Cost Upgrades to Improve the Public Road/Transportation Networks. *In preparation.*

[7] A. Lim, Y. Chee and W. Hsu. Upgrading Links for Performance. Manuscript, *submitted for publication.*

[8] C.D. Tsao. A local area network architecture overview. *IEEE Communications Magazine*, **22** (1984) 7–11.