Order Number 9521293

File forwarding approximation algorithms

Lord, Kenneth John, Ph.D.

City University of New York, 1995

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

File Forwarding Approximation Algorithms

by

Kenneth J. Lord

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

1995

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

10/7/94
Date

Jennifer Whitehead
Dr. Jennifer Whitehead,
Chair of Examining Committee

10/7/94
Date

Roy Stanley Habib
Dr. Stanley Habib,
Executive Officer

Supervisory Committee

Dr. Michael Anshel

Dr. Theodore Brown

Dr. Joel Wein

The City University of New York

Abstract

File Forwarding Approximation Algorithms

by

Kenneth J. Lord

Adviser: Professor Jennifer Whitehead

The file transfer scheduling problem was introduced and studied by Coffman, Garey, Johnson and LaPaugh where it was shown that determining the optimal schedule where all file transfers are completed in minimal time is NP-Complete. This was extended by Whitehead to include the case of forwarded files. We examine polynomial-time approximation algorithms for the forwarding model where knowledge of the file transfer graph on the part of the individual nodes is local and global. We show that in the worst case the local algorithms of "List Scheduling" and "Farthest-First" yield polynomial approximations to the optimal length schedule. We also show randomized and deterministic algorithms that yield worst case polylogarithmic approximations to the optimal length schedule.

Acknowledgments

I am surrounded by friends without whom the production of thesis would not be possible. At those times when satisfying the requirements for the doctoral degree seemed more a test of endurance than intelligence, I have found support in their kindness, wisdom and understanding.

While there are so many people who I count daily as friends, I limit my acknowledgments here to those who have contributed most directly to assisting me with this dissertation (whether they might know it or not). Therefore, I risk the danger of leaving someone out, and ask for forgiveness in advance.

Among my professional colleagues and friends I am proud to thank my adviser, Jennifer Whitehead, for her brilliant technical assistance. Her guidance, clarifications and patience, especially on those occasions when I lost all sense of direction, were given constructively and without judgment.

Ted Brown, a good friend and good department chair, has stood by me both personally and professionally. He helped on may occasions to put things in focus and perspective, and gave me departmental support whenever possible.

I am also indebted to Carol Friedman for her "LaTeXpertise", without which I would still be suffering through the efforts of rewrites.

My close personal friends have contributed in an immeasurable way to this document. Not that any of them actually cares about File Forwarding, but they have given me an unconditional support for which I am sincerely grateful. I thank Bruce Sternemann for his humor and tolerance, Peter Klarman for his acumen and good ear, and Olga Salizkiy, for her honest sympathy and outlook.

Lastly, this thesis would have been possible without the love and support of my parents, who are more than just "mom" and "dad"; they are my friends. I'm sure they thought it would never get done. I'm very sure they wanted to nag me about getting it done more than they did. No one could have more loving and supportive parents, and I thank them for all they have done for me.

Contents

List of Figures

Chapter 1

Introduction

The interconnection of two or more computers to facilitate the sharing of data using any number of communication technologies is almost universal. Machines are linked together from room to room and from coast to coast, and the amount of information shared between them is staggering. The number of techniques that can be used to connect these machines is similarly large (for a more complete discussion of computer networks, see [17]). One thing remains common in these strategies, however, regardless of the technology over which information is transferred. The speed of the processors, relative to the speed of the communication lines, will undoubtedly cause a backup of the data to be transferred and necessitate decisions about how to handle the waiting data.

Naturally, it is desirable to automate the process of making these decisions. Therefore, a mathematical model (implemented as a data structure) which algorithms can use as an object for their calculations is necessary. Since many nodes can be connected via many communication lines, a graph, by virtue of its many-to-many data relationship, is commonly used.

A graph is applied to a computer network by letting the vertices represent the processors (or "processing nodes", or just "nodes"), and the edges the communication lines between them. The existence of an edge (v_1,v_2) in the set E, therefore, indicates that nodes v_1 and v_2 can transfer information between each

other. Throughout this discussion it will be presumed that the edge is "undirected" whereby the flow of information can go from either of the nodes to the other. It is also possible that two nodes may have more than one communication line between them, and thereby transfers may occur simultaneously between nodes. This can be represented using a multigraph, where edges in the set E may be repeated. (For a more complete discussion of graphs, see [8]).

A graph is said to be "completely connected" if every vertex in the graph is adjacent to every other vertex (consequently there need to be $\frac{n(n-1)}{2}$ edges). Within the context of a network, this means that every node may communicate with every other node. Figure 1.1 shows a completely connected graph representing a network of five nodes.

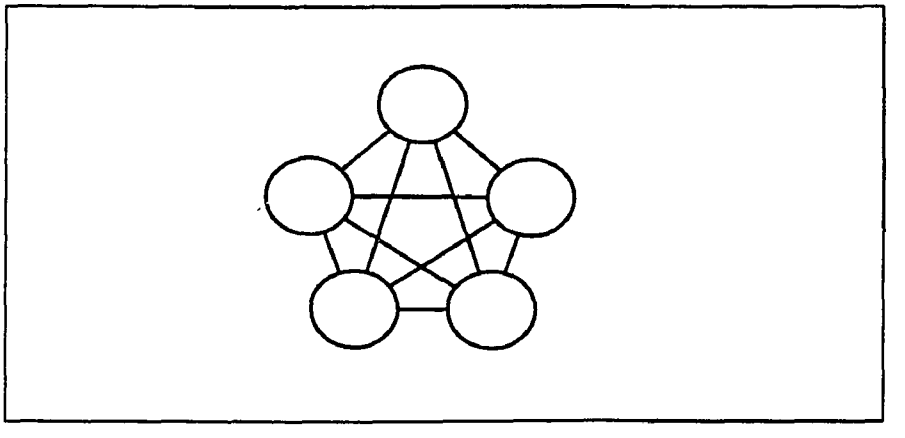

Figure 1.1: A Completely Connected Graph

The basic unit of information stored externally from the machine's central processing unit is the file, and we will presume that it is files that will be transferred from node to node. In Figure 1.1, a file may be transferred directly from any node to any other node, as the graph is completely connected. If each edge represents multiple communication lines between the nodes, then more than one file may be sent from node to node at the same time. However, if the number of files to be sent between nodes exceeds the number of communication lines, then a decision has to be made as to which one will be sent first. As the size of these

files may vary, and the communication lines, (whatever the technology), tend to transfer data at a fixed speed, this decision may greatly affect the time at which all the files have reached their destination. Clearly, it is desirable for this time to be as small as possible.

1.1 The File Transfer Problem

Coffman [2] *et al.* have studied the problem of transferring large files between various nodes of a network in order to minimize the total time for the overall transfer process. They use a labeled, undirected multigraph which they call a File Transfer Graph. The vertices are labeled with integers that represent the maximum number of file transfers that can occur simultaneously (the "port capacity"). A labelled edge represents a file to be transferred, with the label indicating the length of the file. Note that in this case edges do not represent connections between nodes, as the network is presumed to be completely connected.

Figure 1.2 shows a graph with four vertices (v_1, v_2,..., v_4) and six edges ($e_1, e_2, ..., e_6$) representing a network with four nodes and six files that are to be

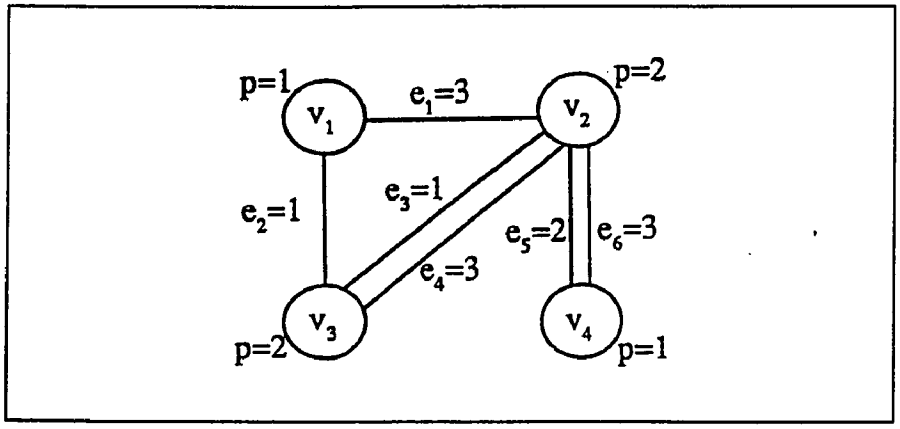

Figure 1.2: A File Transfer Graph

transferred. The labeling shows that the port capacities of nodes v_1 to v_4 are 1, 2, 2 and 1 respectively, and the lengths of the files e_1 to e_6 are 3, 1, 1, 3, 2 and

3, respectively.

Since the number of files to be transferred to each of the nodes exceeds the port capacity of the nodes, it is impossible to transmit all of the files starting at the same time. It is necessary to create a schedule which will put the files in some order for transmission. Coffman *et al.* have graphed this schedule, marking time along the horizontal axis and the possible simultaneous file transfers along the vertical axis. It is presumed that a length n file takes n time units to be transferred, and there is no other overhead in the file transfer process. Figure 1.3 shows two possible schedules ("a" and "b") that can be used for completing the file transfers for the graph in Figure 1.2. In schedule "a", all files are transmitted

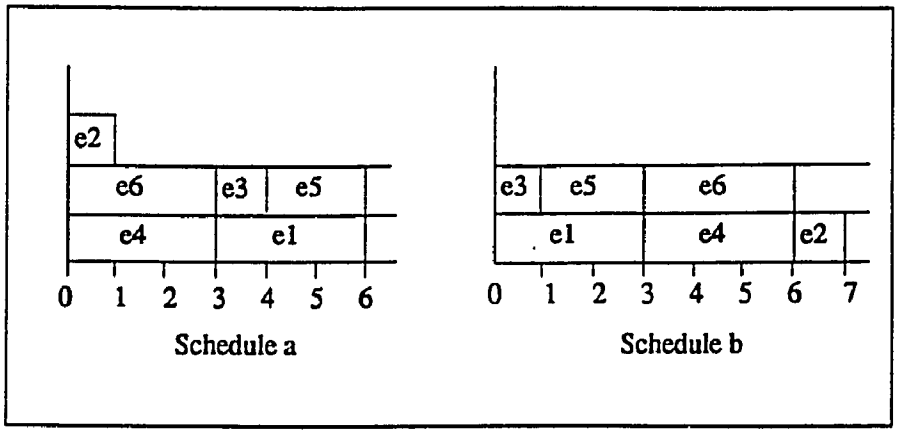

Figure 1.3: Two Possible Schedules

after six time units. In schedule "b" the files finish in 7 time units.

The *makespan* of a schedule is the time the last file transfer is finished. A natural goal, therefore, is to find a schedule which has the smallest makespan. Schedule "a" in Figure 1.3 is such a schedule (having a makespan of 6). An optimal schedule, OPT(G), is one which has the smallest possible makespan. Since node v_2 cannot finish before time 6 (the sum of the file sizes divided by the port capacity, which is $\frac{12}{2}$) it is clear that schedule "a" is an optimal schedule (i.e., OPT ≥ 6, since schedule "a" takes 6 time units, schedule "a" is optimal).

1.2 Summary of Optimal Solutions With No Forwarding

Coffman *et al.* go on to prove that the problem of finding an optimal schedule (the "file transfer scheduling problem") is NP-Complete. They show this is true even when restricted to file transfer graphs in which the port capacity of each vertex is 1, there is at most one edge between each pair of vertices, and all edges have the same length. This is done by likening the problem to the Chromatic Index problem as presented by Garey and Johnson [5]. It is also shown that the problem is NP-Complete when the graphs are restricted to two vertices.

If certain restrictions are put on the graph then polynomial time algorithms exist to find the optimal solution. Coffman *et al.* show this is true in the single port (i.e. port capacity is 1) and single edge cases when the graphs are bipartite, trees, paths, even cycles, and odd cycles.

This thesis will be restricted to graphs with unit edge length and single port capacity, but using general unrestricted graphs, for which the optimal solution is NP-Complete. We will examine several approximation algorithms as they may be used for file transfers with forwarding. We will first summarize what is known about approximation algorithms without forwarding.

1.3 Summary of Approximation Results With No Forwarding

The problem of scheduling activities which may be performed concurrently but which are partially ordered is very common. One example is known as "job shop scheduling," a problem in which there are many factory workers carrying out a sequence of operations which must be done in a certain order. As this problem is known to be NP-Complete, a great deal of attention has been focused on

algorithms which will produce an order for the activities which will be completed in an amount of time reasonably close to the optimal time. One well-known algorithm is List Scheduling.

List Scheduling

The list scheduling algorithm presumes that all the activities to be performed are ordered in a "priority" list. Each time an activity is finished, the algorithm scans the list for another activity that is ready to start. No ready activity is ever delayed when it has the opportunity to be done. That is, it may be delayed because another ready activity is chosen, but it cannot be delayed in favor of doing nothing at all. (It can be shown in some scheduling instances that including this inactive time may shorten the overall schedule.)

In the case of scheduling files in a file transfer graph, it is the edges (representing the file transfers) that are put in the priority list. An edge is "ready" when it has not been started and there is a port available at each end of the edge (i.e., at each node). The list is scanned, starting all possible file transfers. When a transfer is begun, the ports used by the transfer are marked as unavailable. When the transfer is complete, the ports are then marked as being available and any other ready transfers are started.

The time to perform the list scheduling is $O(\|E\|^2)$, as stated by Coffman *et al.*, as the list may have to be scanned once for each file to be transferred. However, Coffman *et al.* go on to show that by using a more clever implementation with careful attention to data structures, the time can be reduced to $O(\|V\|\|E\| + \|E\| \log \|E\|)$. The schedules produced by list scheduling can vary widely, depending on the order of the edges to be transferred. Figure 1.4 shows an example file transfer graph where all files are unit length and the port capacity of every node is one.

The graph in Figure 1.4 shows 16 files to be transferred, labeled e_1 to e_{16}.

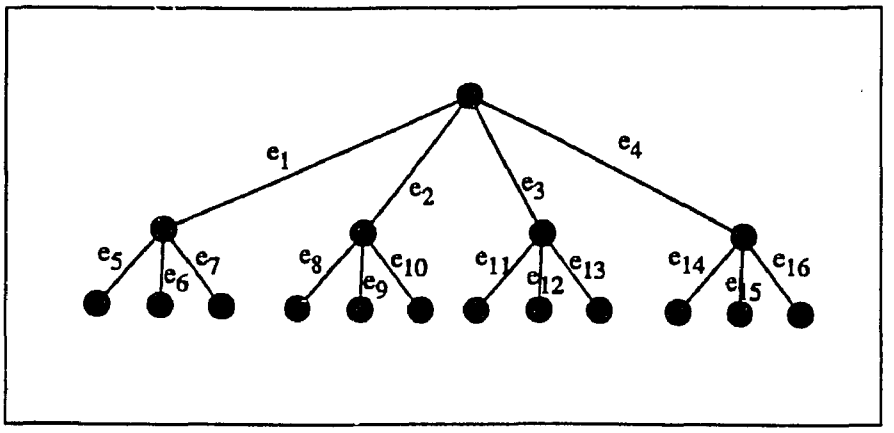

Figure 1.4: File Transfer Graph

If we let S_1 be the set of edges are ordered $e_1, e_2, e_3, ..., e_{16}$, and S_2 be the set of edges ordered $e_5, e_6, e_7, ..., e_{16}, e_1, e_2, e_3, e_4$, then the schedules produced by the list scheduling algorithm for S_1 and S_2 finish at times 4 and 7, respectively. Figure 1.5 shows the graphs for each of these sets. Of all of the possible schedules produced

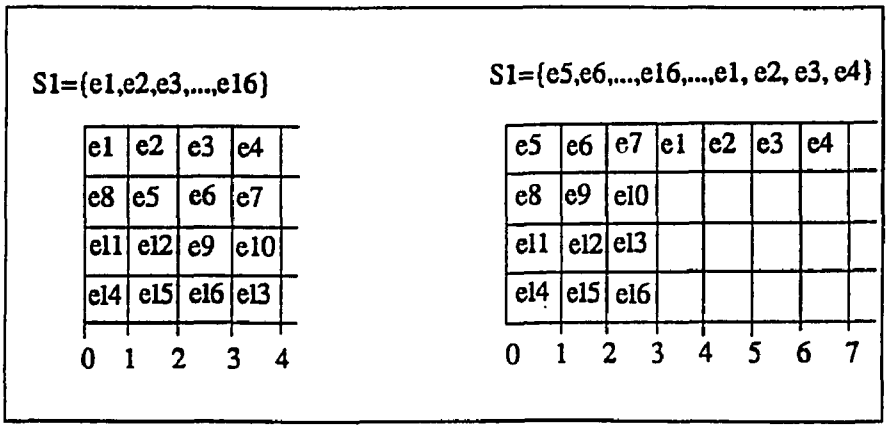

Figure 1.5: List Schedules for Sets S_1 and S_2

by the list scheduling for a given graph, there will be a minimum finishing time. Let that time be $OPT_{LS}(G)$. Since it is an approximation to the optimal schedule, how close can it possibly come to the optimal schedule, OPT(G)? Coffman *et al.* show that for any $d > 0$ there exists a file transfer graph G such that

$$OPT_{LS}(G) > \frac{4}{3} - d\,OPT(G)$$

Conversely, another important question is how bad will the worst schedule for a

graph be using list scheduling? Coffman *et al.* show that for any file transfer graph G and any list schedule S we have

$$LS(G,S) \leq 2OPT(G) + \max\left[0, L(1-\frac{2}{p})\right]$$

where L is the maximum edge length and p is the maximum port capacity.

If files are restricted to unit length, Coffman *et al.* show that

$$OPT_{LS}(G) \leq 2\,OPT(G) - 1$$

Figure 1.6 offers an example which shows that this bound can be achieved. On the left side is a labelling for the optimal schedule (OPT), and on the right is a similar labelling for the worst-case schedule under list scheduling (OPT_{LS}). The label of an edge represents the time that the file transfer is begun. Therefore, the time at which all file transfers are completed will be one more than the maximum of the edge labels (because the label represents the time the transfer *begins*, and all file transfers take unit time). The finish time for the optimal schedule in this case is 4, and the finish time for the list schedule is 2(4)-1, or 7 (as required). The delay is created by the fact that node "b" cannot begin to work with files until

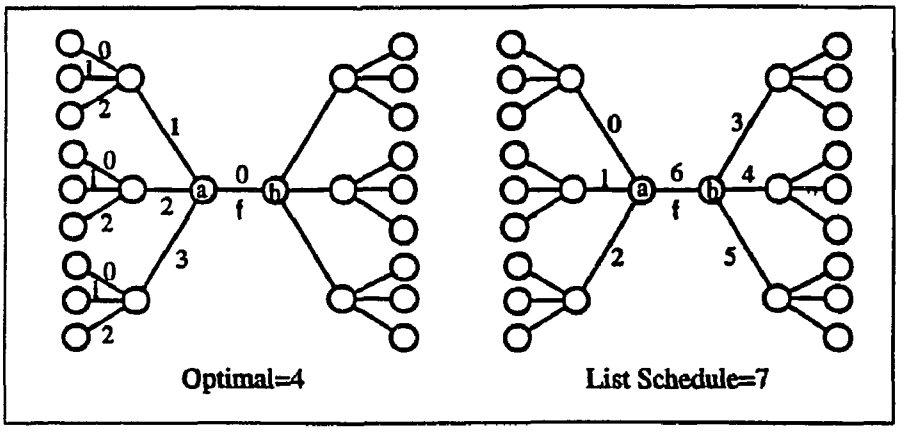

Figure 1.6: Optimal and Worst Case File Transfers

time 3 (which is also the time file f can be transferred). According to the list

schedule, node "b" then transfers other files at times 3, 4, and 5, and is finally ready to transfer file f at time 6.

One should be careful in constructing such examples when using the list scheduling algorithm, as ready files may not be unnecessarily delayed (i.e., if available ports are at each end). Figure 1.7 shows an erroneous labelling for the same graph as presented in Figure 1.6. The delays that are on the right side are now duplicated on the left side. Note, however, that at time 0 neither node "a" nor node "b" is busy, and consequently file f must be scheduled at time 0.

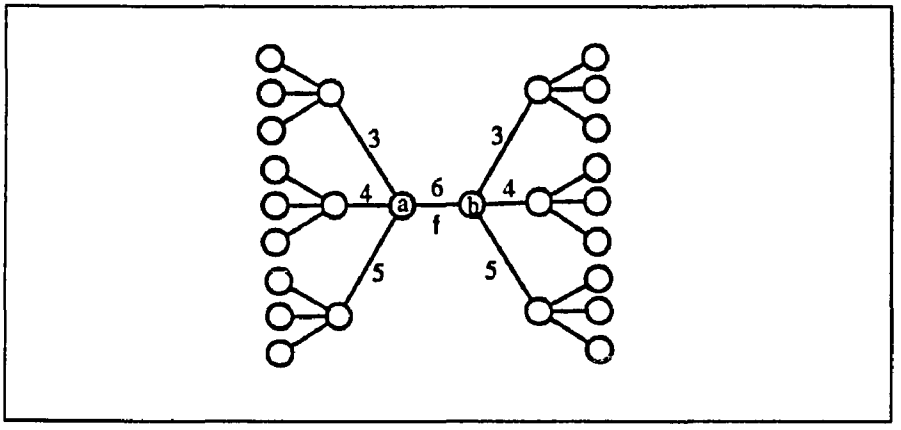

Figure 1.7: Incorrect List Scheduling

Edge Coloring

Coffman *et al.* observe that in the case of unit length, single port capacity graphs that the problem needs to be solved by assigning a unique time to every transfer done by a node. This problem can be solved using an edge coloring algorithm (equating file transfer times to colors). There are polynomial time solutions that can approximate the solution to within one color of the minimum number of colors necessary (the "chromatic index"). This means that the optimal solution (OPT) can be approximated to to within $OPT + 1$ using edge coloring.

For our purposes, however, where we want to consider forwarded files, edge coloring alone will not work, as forwarded files represent a partial order,

and cannot be colored arbitrarily. That is, the last step cannot be done before the previous step is finished. Optimal coloring might decide to schedule just that.

In Chapter 3 we will discuss the use of edge coloring for an approximation algorithm.

1.4 Summary of Optimal Solutions With Forwarding

So far, we have considered file transfer graphs where each transfer is represented by a single edge. That is, there is a direct connection between the node from which the file is being sent and the node which is receiving the file. In order to make this a general model of file transfers between nodes, where a file may be sent from any node to any other node, the graph would have to be completely connected. A completely connected graph is one in which there is an edge from every vertex to every other vertex (requiring $\frac{n(n-1)}{2}$ edges).

It is unlikely that a communications network would be practical if it required $O(n^2)$ connections to link every node to every other node. It is more common to employ a "store and forward" system where a file may be sent from its source to its destination through a sequence of intermediary nodes, each of which will receive the file and then forward it to the next node. A connected graph, requiring $O(n)$ edges, may be used to set up such a graph, albeit minimally connected if there are n-1 edges.

The same questions that applied to scheduling files of one edge apply to scheduling multi-edge files. As a port of a node may previously have been used only to send or to receive a file, it may now be called upon to do both. It is even more possible now that a backup of files to be transferred will be created.

Certainly there is an optimal schedule for the files in this graph. Since *non-forwarding* is a special case of the forwarding problem, and has been shown

to be NP-Complete, it follows that *forwarding* is also NP-Complete. Whitehead extends the Coffman *et al.* model to include forwarding, and shows that certain topologies which had polynomial-time algorithms (as shown by Coffman *et al.*), remain NP-Complete when forwarding is allowed. These include fixed-length files on the cases when the graphs are bipartite and the degree of each node is bounded, trees when the degree is unbounded, as well as variable-length files in the cases of paths and even cycles when multiple edges are allowed. Whitehead does show, however, that there are still polynomial time algorithms for trees with bounded degree and single edges between nodes, and for graphs that are paths and cycles.

Again, this thesis will be limited to unit-length files and graphs with single edges between nodes. The following table summarizes the results of Coffman *et al.* and Whitehead with respect to computation of the optimal schedule assuming unit-length, single edge file transfer graphs.

Computation Time for the Optimal Schedule Unit Edge Lengths, One Port, One Edge		
Restrictions	No Forwarding (*Coffman, et al.*)	Forwarding (*Whitehead*)
Bipartite	$O(\|E\|\|V\|)$	NP-Complete
Paths	$O(\|E\|)$	$O(\|E\|)$
Even Cycle	$O(\|E\|)$	$O(\|E\|)$
Odd Cycle	$O(\|E\|)$	$O(\|E\|)$
Tree *bounded degree*	$O(\|E\|)$	$O(n\|E\|^2 \log \|E\|)$
Tree *unbounded degree*	$O(\|E\|)$	NP-Complete
General	**NP-Complete**	**NP-Complete**

Rivera-Vega, *et al.* [14] have explored a related problem called the "File Redistribution Scheduling Problem." In this model the graph is completely connected, and there are only two edges involved in the file forwarding. They show that their problem is NP-hard, and introduce some polynomial time approximation algorithms.

1.5 File Transfers vs. Packet Routing

It should be noted that the problem of scheduling unit-sized files is related to that of packet routing which has been researched by Leighton, Maggs and Rao [11], and Valiant [18]. In our model, routes are determined statically, before the transfer begins, and are considered fixed with respect to the scheduling algorithm. This is similar to a connection of two computers via the telephone system.

In the packet model, given in [18], the route that the packets take is determined dynamically as the packets flow through the network from the source to the destination. Their path is not determined in advance. This model would be appropriate for an interconnection in a multi-processor parallel machine.

Chapter 2

Local Forwarding Algorithms

An important factor which can be used to categorize the algorithm used to create the schedule is the kind of knowledge the algorithm has about the network. Some algorithms require a view of the entire network and all file transfer requests (source, destination and route). They will then, based on this global knowledge, determine which file will be processed by each node so as to approximate the optimal schedule. These would be considered "global" forwarding algorithms, and will be discussed in chapter 3.

Other algorithms presume that each node has some local intelligence about what it is processing, can make some decisions based solely on this local knowledge, and therefore participates in creating the schedule locally. Whatever algorithmic guidelines the node needs to use can be performed in terms of this local information. This kind of algorithm would be considered a "local" forwarding algorithm.

2.1 List Scheduling

The list scheduling algorithm, as described in section 1.3, is a local algorithm as each node would independently consult a list of ready files and select one to process. The list of ready files can be considered a local queue of files waiting to

be processed by the node.

The effect of forwarding on the list scheduling algorithm is that all files are not ready to be transferred at time 0. A file will proceed through a series of nodes, and for any node, x, that node may be a source (the first forwarding step), a sink (the last forwarding step), or it may both receive and send the file (an "i-th") step. At time 0, only files proceeding from source nodes are ready to be scheduled under list scheduling. At time 1, only forwarded files which have been transferred at time 0 (which may not be all of those that were ready, due to port constraints) may be scheduled, or files which have not yet left their source node.

Given a file transfer graph which has an optimal schedule, what will be the worst case finishing time if the list scheduling algorithm is used? If the number of forwarding steps is 1 (essentially, there is no forwarding), then Coffman *et al.* have shown that this time is $2OPT - 1$ (as mentioned in the introduction to this thesis). We now consider file transfer graphs that have an arbitrary number of forwarding steps, n.

The first observation is that $n \leq OPT$. A file that requires n forwarding steps of unit time cannot be transferred in less than n time units. In addition, we observe that the optimal schedule time cannot be less than the maximum degree of all the nodes ($\max_i[D_i] \leq OPT$). If there are i files to be transferred by node i, and the port capacity is 1, then it will take at minimum i time units to complete all file transfers.

Suppose a file, f, is to be forwarded with n steps, through a series of nodes $a_0, a_1, ...a_i, ...a_n$. The file, as previously stated, may optimally be sent in n time units if there is nothing to delay it at any of the nodes. Conversely, the worst case forwarding time may be achieved by applying the maximum possible delay to the file at each of the nodes. This delay may be caused by other files which are

ready at the same time as f, and which are selected in favor of file f by the list scheduling algorithm. These other files must have undergone at least i forwarding steps before reaching node a_i. Files which require fewer than i forwarding steps will already have reached node a_i and must already have been selected by list scheduling since we assume file f has the worst case arrival time of any file after i steps ($1 \leq i \leq n$). We will let e_i represent the number of edges into or out of node a_i which are of length i or more, excluding the edges of file f going into and out of the node.

The earliest ready time for any of these e_i files to be transferred into node a_i is $(i-1)$, as the first of these files must at least have gone through $i-1$ forwarding steps to get to a_i. The earliest time for file f to be forwarded out of a node (including as much delay as possible at that node) is the time it takes for the other files to be ready $(i-1)$, plus the time it takes to send them through (e_i), plus 2 more time units to forward file f in and out of the node. This must be less than or equal to OPT, as the delay of file f is unavoidable because of the bottleneck at node a_i.

The worst case time file f leaves node a_i may be calculated by taking the worst case time it is sent from node a_{i-1}, plus e_i (delaying it with all other ready files), plus 1 to send file f out. Let g(i) represent the worst case arrival and delay time the file can experience at node i. Thus, by time g(i) the file will have either left node a_i, or will have arrived at node a_i and will experience no more delays at node a_i. The worst case time for g(i) is $(i+1)OPT - \frac{i(i+1)}{2} - 1$, as will be shown in Theorem 2.1.

Theorem 2.1 *In a file transfer graph under list scheduling, where a file f is forwarded from node $a_0, a_1, ..., a_i$, the worst case delay f may undergo, g(i), is* $(i+1)OPT - \frac{i(i+1)}{2} - 1, \quad 0 \leq i \leq n-1.$

Proof. Let e_i represent the number of edges into or out of node a_i which are of

length i or more, excluding the edges of file f going into and out of the node. The earliest time node a_i will be done processing all files of length i or more is

$$(i-1)+(e_i)+2 \leq OPT$$

Thus

$$e_i \leq OPT-2-(i-1)$$

In general, the worst case time, g(i), will be the worst case time the file will have arrived at a_{i-1} and be free of delays at a_{i-1} (i.e., g$(i-1)$), plus the delay in serving other files, plus 1 to receive file f:

$$g(i) = g(i-1)+e_i+1,$$

and since $e_i \leq OPT-2-(i-1)$,

$$\begin{aligned} &\leq g(i-1)+OPT-2-(i-1)+1 \\ &\leq g(i-1)+OPT-i \end{aligned}$$

The worst delay g(0) from a_0 (the source node) will be achieved by sending all other files (there are D(a_0)-1 of them) first. Since D(a_0) $\leq$ OPT,

$$g(0) \leq OPT-1.$$

Using difference equations, with g(i) $\leq$ g$(i-1)$+OPT-1, we obtain

$$\begin{aligned} g(1)-g(0) &\leq OPT-1 \\ +g(2)-g(1) &\leq OPT-2 \\ +g(3)-g(2) &\leq OPT-3 \\ &\cdots \\ +g(i)-g(i-1) &\leq OPT-1 \\ g(i)-g(0) &\leq iOPT-(1+2+...+i) \\ &\cdots \\ g(i) &\leq iOPT-\frac{i(i+1)}{2}+g(0) \\ g(i) &\leq iOPT-\frac{i(i+1)}{2}+OPT-1 \end{aligned}$$

$$g(i) \leq (i+1)OPT - \frac{i(i+1)}{2} - 1$$

□

The worst case finishing time for the entire graph is the time at which the most delayed file arrives at its last forwarding step (i.e., file f is received by node a_n).

Corollary 2.1 *The worst case time for all file transfers to be completed in a file transfer graph under list scheduling with n forwarding steps is* $(n+1)OPT - \frac{n(n+1)}{2}$.

Proof. Let T be the worst case arrival time of file f at node a_n. Using the same argument as in Theorem 2.1:

$$T \leq g(n-1) + e_n + 1$$

At node a_n, file f is not forwarded from a_n, therefore

$$\begin{aligned} (n-1) + e_n + 1 &\leq OPT \\ e_n &\leq OPT - n \\ T &\leq g(n-1) + e_n + 1 \\ &\leq g(n-1) + OPT - n + 1 \end{aligned}$$

From Theorem 2.1, using i = n-1:

$$\begin{aligned} &= nOPT - \frac{n(n-1)}{2} - 1 + OPT - n + 1 \\ &= (n+1)OPT - \frac{n^2 - n + 2n}{2} \\ &= (n+1)OPT - \frac{n^2 + n}{2} \\ &= (n+1)OPT - \frac{n(n+1)}{2} \end{aligned}$$

□

Figure 2.1 shows an example of a graph which achieves this bound in the case of two forwarding steps and an optimal schedule, OPT, of 3. The labels of the edges indicate the time that the file was sent.

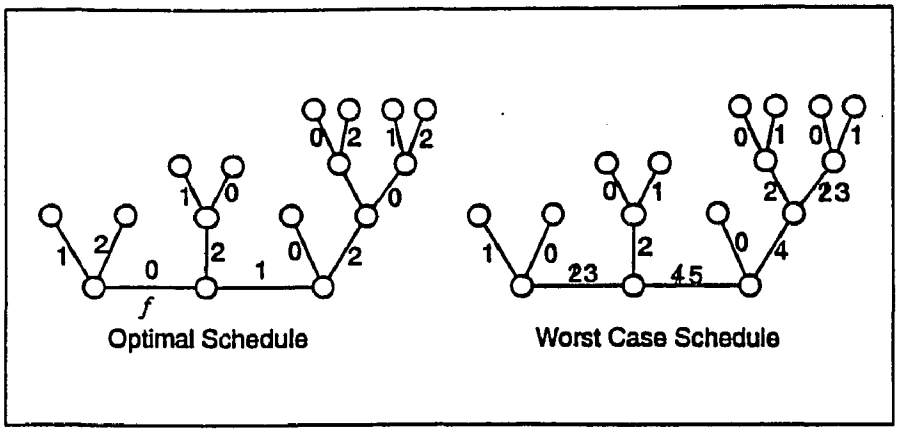

Figure 2.1: Optimal and Worst Case Schedules, n=2, OPT=3

2.2 Farthest First

The order of files chosen by the list scheduling algorithm has been arbitrary up to this point. It seems reasonable that putting the ready files in order of decreasing number of forwarding steps (i.e., the files that have farthest to go) would schedule the most time-consuming activities first, thereby creating a smaller worst case finishing time. Since all file routes are determined centrally in our model, an n-step file departing from its source node is clearly a length-n file, and the algorithm at the receiving node could base its processing decision on this value. After the file is passed on, it may be marked as a file whose length is one less (having undergone one forwarding step) and the next receiving node may process it based on this value. It is possible, therefore, for all nodes to be aware locally of the number of remaining forwarding steps such that it can select the largest. Following Leighton [10] we will call this algorithm "Farthest First".

The Farthest First algorithm is a special case of List Scheduling which of order $O(n)OPT$. This can be exemplified by presuming that all other files at each node can cause delay to some file f as it passes through, and summing the delay over the number of nodes. The number of files that can cause delay to a file, f, passing through can be no more than the degree of the node less

two (for f passing in and out). The degree of the node cannot exceed OPT, so we can consider the greatest delay to be $OPT - 2$. Since f is only processed by the terminal nodes *once*, an extra two files may be processed there, adding a potential delay of 2. There are $n + 1$ nodes through which the length-n file will pass, and it will take n time units for f to go from source to destination.

Will the Farthest First algorithm produce a smaller worst case time than list scheduling? If the number of forwarding steps, n, is 1 (i.e., there is no forwarding), the answer has to be no, as all files will be the same length, and there is no choice for the Farthest First algorithm to make; it will function the same as list scheduling. For cases of multi-step forwarding ($n > 1$), there is the possibility of a difference. We consider first a simple case of two-step forwarding.

2-Step Forwarded Files

Figure 2.2 shows file f being sent from a_0 to a_1 and then to a_2 (a two-step file).

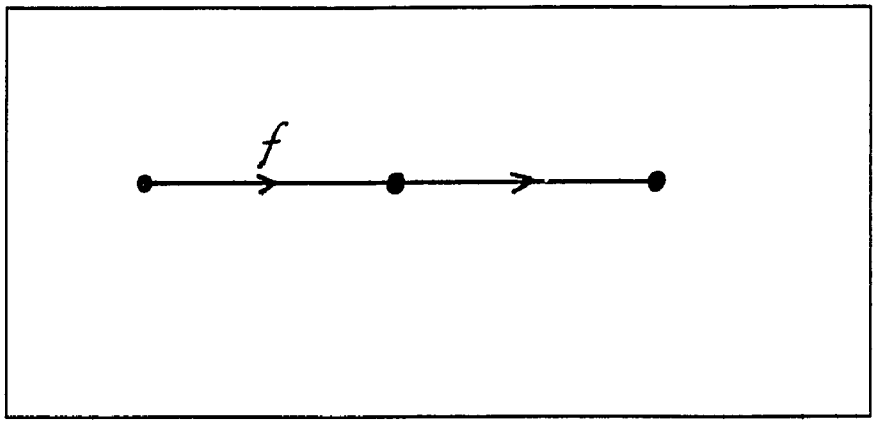

Figure 2.2: A Two-Step File

To determine the effect of the Farthest First algorithm on the worst case delay, we wish to calculate the largest possible value for the time t_d in Figure 2.2. This time represents when f finally reaches its destination, a_2, and may be calculated by applying as much delay as possible to every step of f. That is, if we delay f as much as possible from leaving a_0, passing through a_1 and arriving at a_2, we

will know the worst case delay for f. This delay may be calculated as:

$$\text{Maximum Delay} = \text{Delay at } a_0 + \text{Delay at } a_1 + \text{Delay at } a_2$$

Delay may only be caused by files of length greater than or equal to f's length when using the Farthest First algorithm, so it is important to know how many such files may occur at a node.

The optimal schedule, OPT, is the limiting factor on how many n-length files there may be at any node in the graph. When checking that all file transfers may finish within time OPT, we need to be concerned with *minimum* delays at all nodes. If a file cannot arrive at its destination within time OPT after being subjected to minimum delays, then the file cannot exist in the transfer graph (if it does, then the minimal amount of time to complete all file transfers is greater than OPT which contradicts the definition of OPT).

We will consider the end nodes of an n-step file first (i.e., a_0 and a_n), and then the intermediate nodes $(a_1, ..., a_{n-1})$. The end nodes need to be considered separately as the file does not pass through them and they need to allot only one time unit to the file in order to process it. Intermediate nodes need to allot two time units: one to receive the file and one to send it on. In Lemma 2.1 we show that at the end nodes, there can be no more than $OPT - n$ files of length n in addition to f.

Lemma 2.1 *Given a file transfer graph, G, with an optimal schedule OPT, an n-step file f, forwarded from node $a_0, a_1, ..., a_n$, there can be at most OPT-n files of length n in addition to f departing from the source node a_0 or arriving at node a_n under the Farthest-First forwarding algorithm.*

Proof. At node a_0, let x represent the number maximum number of files in addition to f which may depart from a_0 such that all transfers finish within time OPT. Let t_x represent the time the last of these files departs from a_0, and let

f_x be that file. The minimum time that f_x will arrive at (and be processed by) its destination, a_n, is the time it departs a_0 plus its length (n) time units. This minimal time is equal to OPT. Thus,

$$\begin{aligned} t_x + n &= OPT \\ t_x &= OPT - n \end{aligned}$$

As there are $x+1$ total files (including f) of length n (the highest priority), they will depart at times 0, 1, ..., t_x. Thus, $t_x = x$, and

$$x = OPT - n$$

At node a_n, let y represent the number maximum number of files which may arrive at a_n in addition to f such that all transfers finish within time OPT. The earliest time all of the y files can be ready to be sent into node a_n is $n-1$ (each has undergone $n-1$ steps as each is arriving at its destination). Let t_y represent the time the last of these files arrives at a_n, and let f_y be that file. The transfers of the y files will be finished at a_n no earlier than $(n-1)+y$ (the earliest time the first file can arrive, plus y time units for each of the y files). As this is a minimal time, and one more time unit is needed for f, it must be equal to $OPT-1$, thus

$$\begin{aligned} OPT - 1 &= (n-1) + y \\ y &= OPT - n \end{aligned}$$

□

We now consider the delay that can happen to f as it is forwarded through the intermediate nodes $a_1, ..., a_{n-1}$ under the Farthest First algorithm. In these cases, f is processed twice by the node (as it enters and exits). It may be delayed, however, only by files of equal or greater length (under Farthest First). In Lemma 2.2, we show that the maximum number of length-n files that may pass through a node is $\lceil \frac{OPT-n}{2} \rceil$.

Lemma 2.2 *Given a file transfer graph, G, with n forwarding steps and nodes* $a_0, a_1, ..., a_n$, *an optimal schedule OPT, and a node* a_i *which has an n-step file forwarded through it using the farthest-first algorithm, there can be at most* $\lceil \frac{OPT-n}{2} \rceil$ *other n-step files equal in length to f which also pass through node* a_i.

Proof. Let x be the maximum number of other length-n files which pass through node a_i. Since the maximum number of files passes through node a_i, they must arrive as quickly as possible at a_i and, as they are all the same length, arrive there at the same time. If the files travel unimpeded to a_i, this time can be calculated as $i-1$. If the x files are processed first (requiring $2x$ time units) and then f is processed, it is ready to leave a_i at time $(i-1)+2x+1$. File f now has $n-1$ forwarding steps remaining, requiring $n-1$ time units, and must arrive within OPT. Thus,

$$\begin{aligned} OPT &= (i-1)+2x+1+(n-1) \\ x &= \left\lceil \frac{OPT-n}{2} \right\rceil \end{aligned}$$

We use the integer ceiling as the number of files must be an integer. □

Figure 2.3 shows a labelling for a node a_i graph where OPT=10 and n=6. File f arrives at node a_n in time OPT.

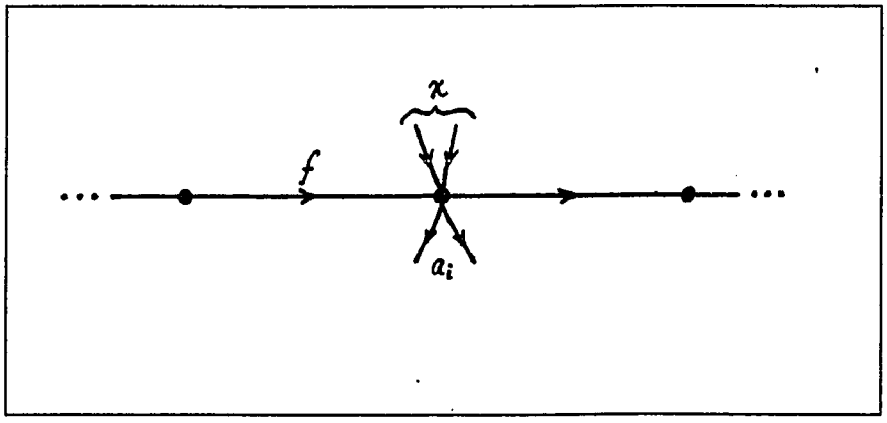

Figure 2.3: Files passing through an intermediate node

To calculate the maximum delay to be experienced by a 2-step file under the farthest-first forwarding algorithm, we simply need to sum the delays at each node (a_0, a_1 and a_2). The combination of these delays leads to a maximum possible delay in the graph of $3OPT - 4$, as stated in Theorem 2.2.

Theorem 2.2 *Using the "Farthest First" algorithm, the maximum delay of a file in a file transfer graph with at most n forwarding steps ($2 = n < OPT$) with optimal schedule OPT is $\leq 3OPT - 4$.*

Proof. Let f be a file to be forwarded from node a_0 through a_1 to node a_2. Let $\delta(a_i)$ be the delay at node a_i. The worst case finish time for file f will be the sum of the delays at each node, plus 2 for the actual transfer of the length-2 file f. The delay at each node is given as a separate case:

Case 1: $\delta(a_0) = OPT - 2$: Since a_0 is a terminal node, by Lemma 2.1, there can be no more than $OPT - 2$ other files of length n at a_0 giving a delay of $OPT - 2$. Any other files must be shorter (i.e., length-1) and cannot cause delay under Farthest First.

Case 2: $\delta(a_1) = OPT - 2$:

Since a_1 is an intermediate node, by Lemma 2.2, there can be no more than $\lceil \frac{OPT-n}{2} \rceil$ other length-n files. These files, being equal in length to f when entering a_1, may cause a delay of $\lceil \frac{OPT-n}{2} \rceil$. File f may no longer be delayed and may enter node a_1. Now, both f and each of the $\lceil \frac{OPT-n}{2} \rceil$ other files wish to leave node a_1, and are again all the same length, so the other files cause another delay of $\lceil \frac{OPT-n}{2} \rceil$. File f may then depart, having undergone a total delay of $\lceil \frac{OPT-n}{2} \rceil + \lceil \frac{OPT-n}{2} \rceil$.

It is important to note in this case, however, that because of the ceiling function, this delay is different when OPT is odd and when it is even:

OPT is **odd**:

$$2 \times \left\lceil \frac{OPT - n}{2} \right\rceil = OPT - n - 1$$

OPT is **even:**

$$2 \times \left\lceil \frac{OPT - n}{2} \right\rceil = OPT - n$$

When OPT is even, the case is straighforward, as the delay is $OPT - n$, as expected. There can be no other files at node a_1 because the total number of files (i.e. edges) at a node may not exceed OPT (i.e. $OPT \leq MaxDegree$). The number of edges at node a_1 equals the number of other files going in and out $(OPT-n)$ plus 2 edges for f itself. Thus, the degree of the node is $OPT-n+2$, and, as n=2, is equal to OPT. There can be no other files.

When OPT is odd, then the number of edges at a_1 due to the other length-n files is $OPT-n-1$, plus two more for f, giving a total of $OPT-n-1+2$. Since n=2 there are $OPT-1$ edges used by length-n files. There is room at the node for one extra file, f_e. This file must be a length-1 file, as Lemma 2.2 says it cannot be a length-2 file. File f_e would have adequate priority to delay f when f is leaving node a_1, because at that time, f is also length-1. This delay will happen only if f_e is processed at a time equal to or later than the time f is ready to be delayed. That is, if the maximum delay which can be applied to $f_e \geq$ that applied to f, then f_e causes delay to f. As stated in case 1, the maximum delay for f coming into a_1 is OPT-2. As f_e is length-1, its maximum delay may be caused by OPT-1 files at its other node, and therefore have a delay of OPT-1. Thus,

$$\text{Delay of } f_e \geq \text{Delay of } f \Leftrightarrow OPT - 1 \geq OPT - 2$$

The total delay at node a_1, therefore, is $OPT - n - 1$ from the other length-n files, plus one more due to the extra length-1 file, or

$$OPT - n - 1 + 1 = OPT - n$$

Case 3: $\delta(a_2) = OPT - 2$:

The delay at node a_2 will be caused by other length-n files arriving at the same time as f. By Lemma 2.1, there may be $OPT - n$ of them. There can also be one length-1 file (there can be no more than OPT files, and counting the OPT-n other length-n files and f itself, that leaves one.) This file, however, will be processed before f is ready to enter node a_2, as the time f is ready to enter a_2 is greater than the largest possible delay of the extra length-1 file at node a_2:

$$\begin{aligned} \text{Delay of } f \text{ entering } a_2 &\geq \text{Largest length-1 delay} \\ OPT - 2 + 1 + OPT - 2 &\geq OPT - 1 + 1 \end{aligned}$$

Since $2 = n < OPT$,

$$2OPT - 3 \geq OPT$$

The maximum delay for file f is the sum of the delays at each node, plus two more for the actual transfer of f itself.

$$\begin{aligned} \text{Maximum delay} &\leq \delta(a_0) + \delta(a_1) + \delta(a_2) + 2 \\ &\leq OPT - 2 + OPT - 2 + OPT - 2 + 2 \\ &\leq 3OPT - 4 \end{aligned}$$

□

Figure 2.5 shows examples of two-step file transfer graphs for OPT=3, 4 and 5. To minimize both drawing and reading the graphs, the examples are constructed by replicating subgraphs which have a known delay. They are labeled SG(d), for a subgraph that will produce a file which is ready to be sent with a maximum delay of d. For example, Figure 2.4 depicts a "black box" graph, SG(3),that will produce a file which is sent at a maximum delayed time of 3. This subgraph may be composed of smaller subgraphs and can be used to create larger subgraphs. This is how the examples are produced.

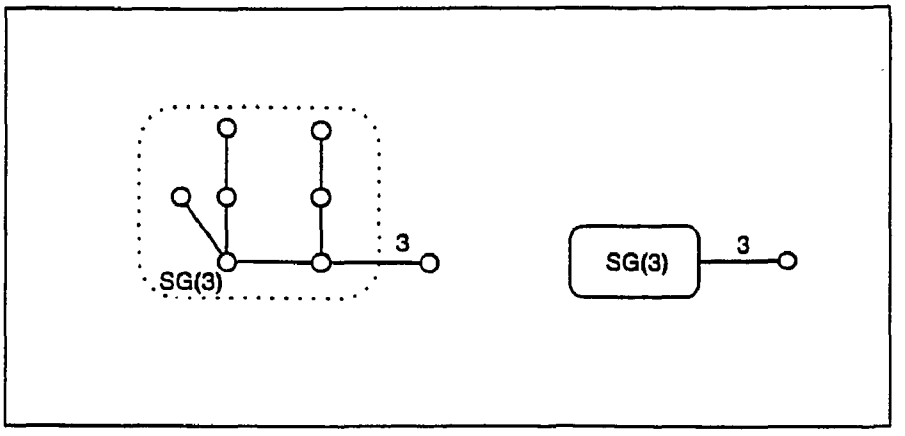

Figure 2.4: Example black box subgraph with delay 3

An Anomalous Case: $n = 2 = OPT$

As was seen in Therorem 2.2, in case 3 it was necessary that $OPT > 2$ to ensure that a length-1 file entering the final node (a_3) could not cause any delay to file f arriving at that node. When $OPT = 2$ and $n = 2$, however, the extra length-1 file entering node a_3 arrives sufficiently late and with adequate priority under Farthest First to cause delay to f. Corollary 2.2 demonstrates this simple case.

Corollary 2.2 *Using the "Farthest First" algorithm, the maximum delay of a file in a file transfer graph with at most n forwarding steps* ($2 = n = OPT$) *with optimal schedule OPT is* $\leq 3OPT - 3$.

Proof. As stated in case 3 of Theorem 2.2, file f may be delayed in entering node a_3 if the length-1 file's delay is greater than or equal to f's delay. The maximum delay of f entering a_2 is $2OPT - 3$. The maximum delay of the length-1 file, f_1, entering a_3 is 1 as another length-1 file may use one time unit when it is processed by f_1's other node. As $OPT = n = 2$,

$$\begin{aligned} \text{Delay of } f \text{ entering } a_2 &= \text{Largest length-1 delay} \\ 2OPT - 3 &= 1 \end{aligned}$$

$$\text{Maximum delay} \leq \delta(a_0) + \delta(a_1) + \delta(a_2) + 1 + 2$$

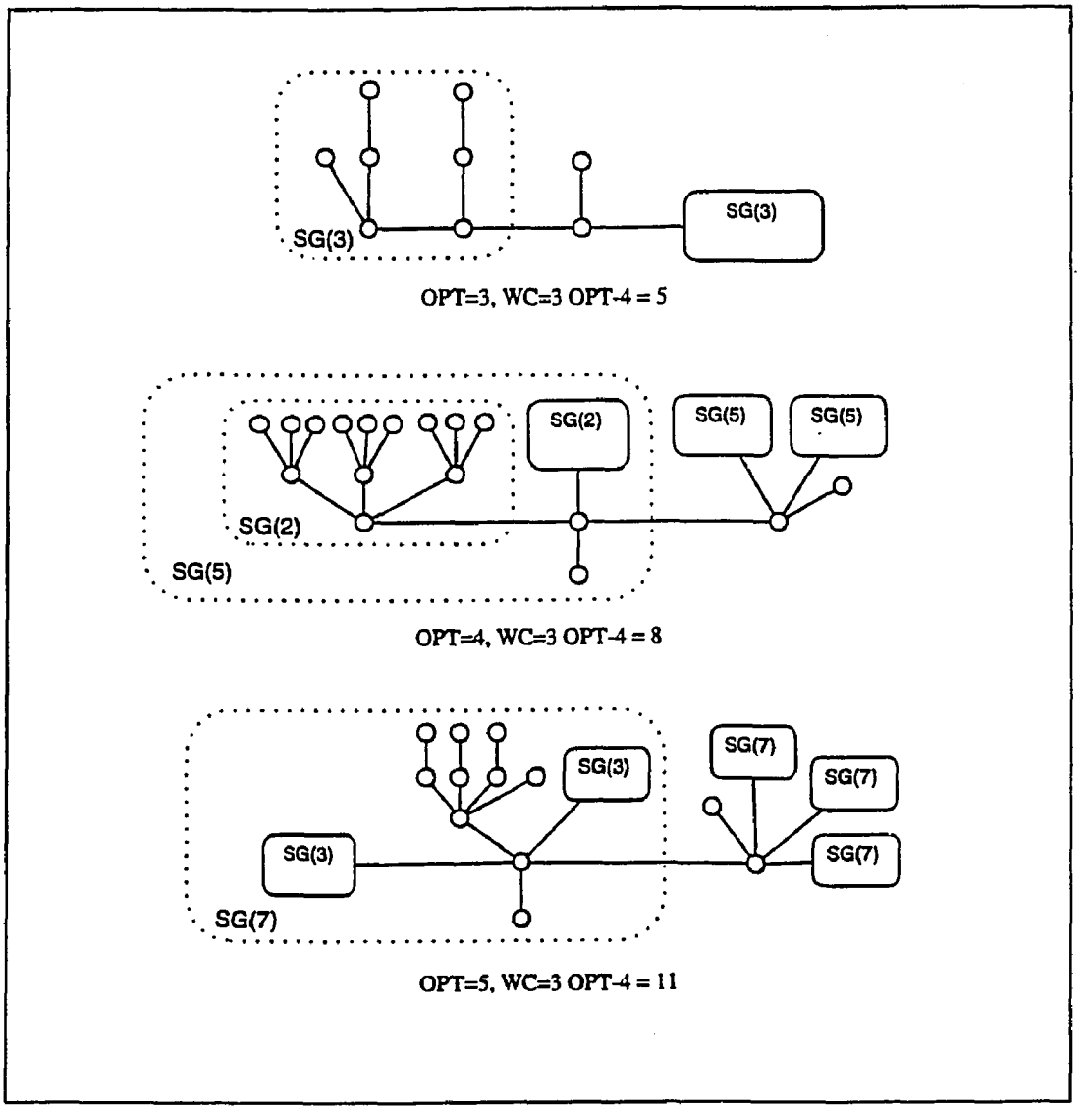

Figure 2.5: Examples of worst case delays

$$\leq OPT - 2 + OPT - 2 + OPT - 2 + 3$$
$$\leq 3OPT - 3$$

□

This anomalous case ($OPT = n = 2$) gives the same worst case delay as List Scheduling ($3OPT - 3$), and demonstrates a case where Farthest First is no better than List Scheduling as an approximation algorithm. Figure 2.6 shows a file transfer graph with $OPT = n = 2$, and labelling of the worst case delays.

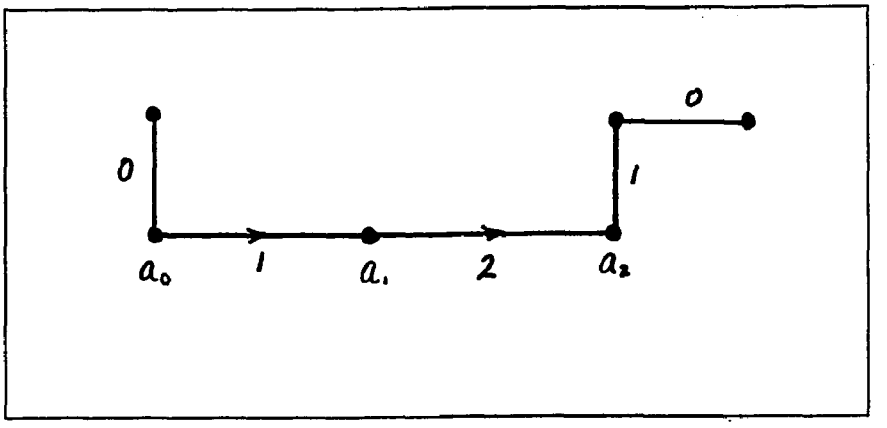

Figure 2.6: Worst Case Graph for $OPT = n = 2$

The General Case: $2 < n \leq OPT$

Obtaining a closed form for the general case is complex as there are many possibilities and conditions that can cause a delay to a long file as it passes through the network under the Farthest-First algorithm. As mentioned earlier, if a file is delayed by all other files at all nodes, the total delay is $OPT - 2$ (the delay by all other files at a node), times $n + 1$, (for all nodes), plus n time units (for f to be transmitted), less 2 because the terminal nodes only process f once:

$$WC_{FF} \leq (n+1)(OPT-2) + n + 2$$

In Theorem 2.2 we have seen that a delay of at least $OPT - n$ can occur on a file at every node in the graph. If we were to presume that this is only delay, then the total delay would be a delay of $OPT - n$ for each of the $n + 1$ nodes, plus n time units for f to be transmitted. The delay would be no less than:

$$(n+1)(OPT-n) + n \leq WC_{FF}$$

It is possible, then, to at least state the limits of the Farthest First worst case delay:

$$(n+1)(OPT-n) + n \leq WC_{FF} \leq (n+1)(OPT-2) + n + 2$$

We now offer some insights into the intricacy of the calculations and a successful algorithmic approach for the case of $n = 2$.

Calculation of the delay at a node, a_i, seems a simple idea, but is surprisingly complicated. To determine what the delay of a file exiting node a_i will be, we need to take the maximum delay of the file as it enters node a_i, and add the delay caused by other files at a_i. Let $\tau(n, i, OPT)$ be the value of the delay for a file of length n exiting node i in a transfer graph with optimal schedule OPT. Let $\delta(n, i, OPT)$ represent the delay caused to a length n file by files at node i in the transfer graph with optimal schedule OPT, exclusive of the delay caused by length-n files passing through (already calculated as $2 \times \left\lceil \frac{OPT-n}{2} \right\rceil$). Then $\tau(n, i, OPT)$ is equal to the delay of the file entering the previous node $(i-1)$, plus the delay by other files at node i, plus the delay caused by other length-n files, plus one for f itself to be transferred:

$$\tau(n, i, OPT) = \tau(n, i-1, OPT) + \delta(n, i, OPT) + 2 \times \left\lceil \frac{OPT - n}{2} \right\rceil + 1$$

The recursion on τ easily halts when $i = 0$, as Lemma 2.1 has shown that this delay is equal to $OPT - n$. It is function $\delta()$ which is difficult to calculate because of the constraints and conditions necessary for delay to be possible. For example, when calculating the delay of file f caused by other files at node a_i, all of the following must hold:

1. It must be possible for the transfer graph to complete within time OPT. Files added at node a_i, especially longer ones, may be so congested at node a_i that they cannot all pass through and arrive at their destination within OPT.

2. If node a_i is to be saturated with extra files (i.e. $OPT = MaxDegree$), then at least one file must be ready at time 0, or it will not be possible to finish within time OPT.

3. The extra files must be of sufficient length such that they have adequate priority to delay f under the farthest first algorithm.

4. Delay may be caused by departing files of adequate priority under some conditions.

5. Delay may be caused by arriving files of adequate priority under some conditions.

The nodes that are best understood are the first (a_0), the last (a_n), and the penultimate node (a_{n-1}). The penultimate node is easier to understand than the other intermediate nodes because at this node, f's length is 1 and therefore all files have equal priority to cause delay (as *incoming* files only). For node i, $2 \leq i < n-1$, delay may be caused by files which are (1) entering, (2) exiting and/or (3) passing through which greatly complicates the computation of the δ function. If we let $n = 2$, however, then these troublesome nodes do not exist (we have only the terminal and penultimate nodes in the graph), and we can complete the formulation of an algorithm based on the τ and δ functions previously described.

An Algorithm for n=2

If $n = 2$, then the worst case finishing time will be the worst delay of a file exiting node a_1, plus delay caused at node a_2 ($OPT - n$ by Lemma 2.1), plus 1 for the transfer of f, or

$$\tau(2, 1, OPT) + OPT - n + 1$$

As stated previously,

$$\tau(n, i, OPT) = \tau(n, i-1, OPT) + \delta(n, i, OPT) + 2 \times \left\lceil \frac{OPT - n}{2} \right\rceil + 1$$

thus for $n = 2$,

$$\tau(2, 1, OPT) = \tau(2, 0, OPT) + \delta(2, 1, OPT) + 2 \times \left\lceil \frac{OPT - n}{2} \right\rceil + 1$$

The δ function needs to count the number of length-1 files that may cause delay, with the conditions that

- The graph may complete within time OPT
- The file has a worst case delay time late enough to delay f.

File f and the other length-n files, being the longest files, will arrive last under the optimal schedule. We can calculate that the number of length-n files passing through node a_i will be $2 \times \left\lceil \frac{OPT-n}{2} \right\rceil$ (for the other length-n files) -2 (for file f). Let this number be x. The extra files, may then use time units from 0 to $x-1$, and therefore their lengths can be no more than 1 to x. The question is, which of these files can cause delay? In general, they can cause delay, as previously stated, if their delay time is greater than or equal to the delay time of f as it entered node a_i from node a_{i-1}, which is $\tau(n, i-1, OPT)$. Therefore,

$$\delta(n, n-1, OPT) = \sum_{j=1}^{x} \begin{cases} 1, & \text{if } \tau(j, j-1, OPT) \geq \tau(n, i-1, OPT); \\ 0, & \text{otherwise.} \end{cases}$$

The calculation of $\tau(n, i-1, OPT)$ is difficult in cases where $n > 2$, but when $n = 2$ we reach the base case $\tau(n, 0, OPT)$ which is equal to $OPT - n$.

2.3 Farthest First with Static Priority

The Farthest First algorithm as explored in the previous section presumed that the file's priority was dynamic as it moved through the network. That is, the priority was based on its current length left to go, which decreased as it moved from node to node. An alternative approach would be to assign a static priority to the file based on its inital length, and have the file keep that priority throughout the network. It can be easilty shown, based on the results obtained in the dynamic algorithm, that this is worse than the dynamic case.

Theorem 2.3 *Using the "Static Farthest First" algorithm, the maximum delay of a file in a file transfer graph with at most n forwarding steps with optimal schedule OPT is* $2(OPT - n) + (n-2)(2\left\lceil \frac{OPT-n}{2} \right\rceil)$.

Proof. Let f be a file of length n. Since f may only be delayed by files which have lengths equal to itself, the maximum delay of f is the sum of the delays caused by files of length n at each of the i nodes, $0 < i < n$. At the terminal nodes, a_0 and a_n, there can only be $OPT - n$ files of length n causing delay, by Lemma 2.1. At the intermediate nodes, $1 \leq i \leq n-1$, there may only be $2\left\lceil \frac{OPT-n}{2} \right\rceil$ length n files, by Lemma 2.2. The total delay, therefore, is

$$2(OPT - n) + (n-2)(2\left\lceil \frac{OPT-n}{2} \right\rceil)$$

□

Chapter 3

Global Forwarding Algorithms

In this chapter we will focus on algorithms that have knowledge of the entire file transfer graph. Unlike the algorithms in Chapter 2 where the nodes could make decisions based on local information, Global Algorithms will produce a schedule for the entire transfer graph before any file transmissions begin. Each step of each file transfer will be assigned a time at which the transfer will take place. The question, therefore, is how close can an approximated schedule come to the optimal schedule using these algorithms. The two algorithms discussed are adaptations of algorithms used by Shmoys, Stein and Wein [15] for job shop scheduling.

3.1 A Randomized Algorithm

Given global knowledge of the files and their routes through the File Transfer Graph, a schedule can be computed that is no worse than $O(\frac{\log n}{\log \log n})OPT$ with high probability. Leighton *et al.* [11] introduce a three-step algorithm for assigning random delays to jobs of unit length. Shmoys, *et. al* [15] extend the algorithm for the general case of Job Shop Scheduling Problem. Here we will liken the File Transfer Problem to that of Job Shop Scheduling and show that the techniques used can be adapted to produce an efficient file transfer schedule. We propose

the following adapted algorithm:

1. Define the *oblivious* schedule, where each file is scheduled at time 0 and proceeds to its destination in n steps (i.e., it is oblivious to other files). This schedule is of length n, but there may be times when more than one file is scheduled to be processed by a node at any given time.

2. *Perturb* this schedule by delaying the start of each file by a random integral amount chosen uniformly in $[0, MaxDegree]$. The resulting schedule, with high probability, has no more than $O(\frac{\log n}{\log \log n})$ files assigned to any node at the same time.

3. *Flatten* this into a schedule that has at most one file for each time unit. This is done by giving files which, in the perturbed schedule, had conflicting times each a unique time by using edge coloring.

The Oblivious Schedule

The oblivious schedule treats every file as an independent unit, and thus each is scheduled to begin at time 0. Figure 3.1 shows a file transfer graph with an oblivious labeling. Every file's first step is labelled with time 0. Each file would be presumed to have its next forwarding step processed at the next time unit, ignoring any possible conflicts for ports at the nodes. For example, file a's first step is obliviously scheduled at time 0, and its next forwarding step is therefore scheduled at time 1. For comparison's sake, Figure 3.2 shows an optimal schedule for this file transfer graph. It is optimal as all transfers are finished at time 5 which is equal to the $MaxDegree$. The optimal schedule cannot be less than the maximum degree of the file transfer graph.

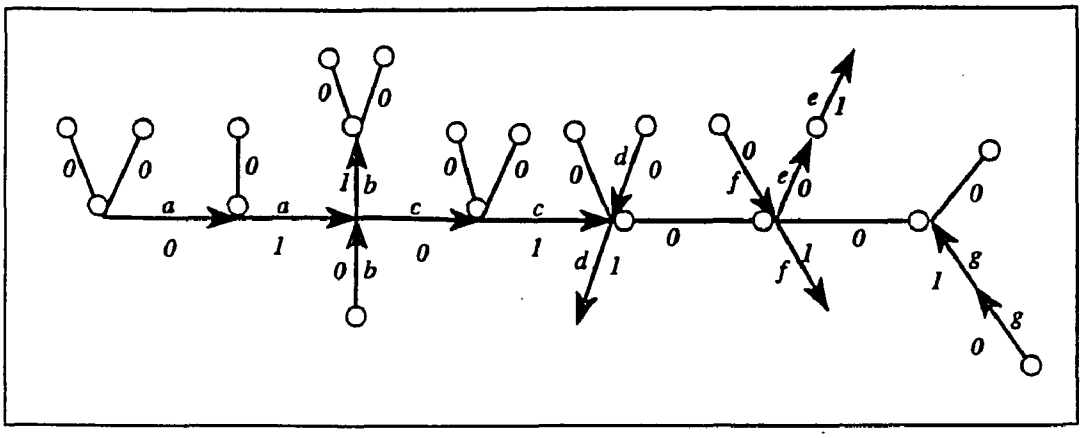

Figure 3.1: Oblivious Schedule

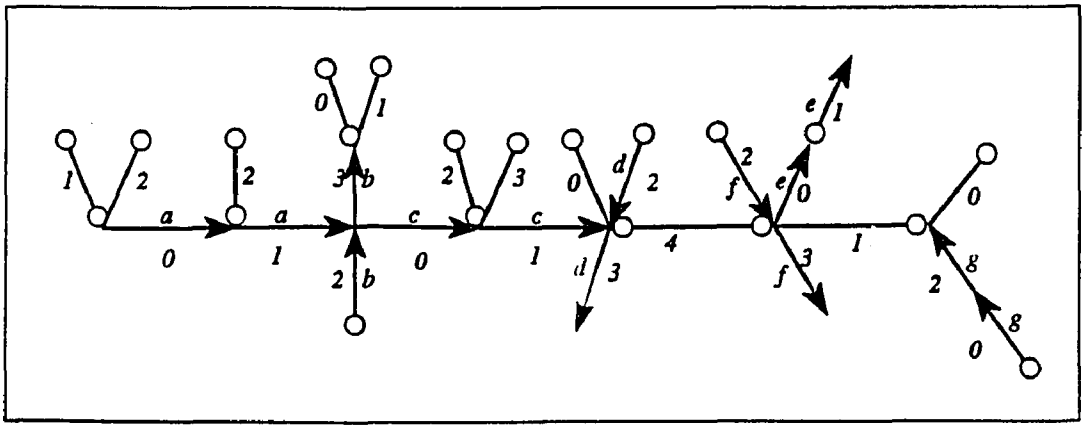

Figure 3.2: Optimal Schedule

The Perturbation Algorithm

The initial steps of all files (those ready at time 0), are given uniformly distributed random starting times in the interval [0,$MaxDegree$] ($MaxDegree$ is the maximum degree of all the nodes in the graph.) As is shown in Lemma 3.1, the number of files at any node which are assigned the same (and therefore, conflicting) time is no more than $O(\frac{\log n}{\log \log n})$, with high probability. The schedule will be invalid because of the conflicts, but the number of conflicts is greatly reduced.

Lemma 3.1 *Given a File transfer graph with number of nodes and maximum degree bounded by a polynomial in n, the strategy of delaying each file by an integral amount chosen randomly and uniformly from* $[0, MaxDegree]$ *and then forwarding those files with no further delay will yield an (invalid) schedule that is of length at most* $MaxDegree + n$ *and, with high probability, has no more than* $O(\frac{\log n}{\log\log n})$ *files scheduled at conflicting times at any node.*

Proof: Consider some time t and some node a_i. Let p be the probability that at least τ files are scheduled at the same time at node a_i. There are at most $\binom{MaxDegree}{\tau}$ ways to choose τ files from all those which node a_i must process. Considering one of these ways at some time t, then the probability that this unit is scheduled at time t is $\frac{1}{MaxDegree}$, since the delay is chosen uniformly at random from among $MaxDegree$ possibilities. Since all τ files are from different files (they must all be different as each time unit in τ represents the starting time of individual files), then the probability they are all scheduled at time t is at most $\left(\frac{1}{MaxDegree}\right)^{\tau}$ since the delays are chosen independently. Therefore,

$$\begin{aligned} p &\leq \binom{MaxDegree}{\tau}\left(\frac{1}{MaxDegree}\right)^{\tau} && (3.1)\\ &\leq \left(\frac{e\,MaxDegree}{\tau}\right)^{\tau}\left(\frac{1}{MaxDegree}\right)^{\tau} && (3.2)\\ &\leq \left(\frac{e}{\tau}\right)^{\tau} \end{aligned}$$

(Justification for the transition from equation (3.1) to equation (3.2) is given in Lemma 3.2.)

If we let $\tau = k\frac{\log n}{\log\log n}$, then $p < \frac{1}{n^{k-1}}$:

$$\begin{aligned} p &\leq \left(\frac{e}{\tau}\right)^{\tau}\\ &\leq \left(\frac{e}{k\frac{\log n}{\log\log n}}\right)^{k\frac{\log n}{\log\log n}} \end{aligned}$$

$$\leq \frac{1}{n^k - 1}$$

For *any* node at *any* time, multiply p by $(n + MaxDegree)$, and by the number of nodes. Choosing k large enough yields that, with high probability, no more than $O(\frac{\log n}{\log \log n})$ files are scheduled at conflicting times at any node. □

For the transition from equation 3.1 to equation 3.2 in Lemma 3.1, we make use of the following Lemma as given by Leighton [10].

Lemma 3.2 (Leighton) *For any* $0 < y < x$,

$$\binom{x}{y} < \left(\frac{xe}{y}\right)^y$$

Proof: By definition,

$$\begin{aligned} \binom{x}{y} &= \frac{x!}{(x-y)!y!} \\ &\leq \frac{x^y}{y!} \end{aligned}$$

From Stirling's formula, we know that for any z,

$$z! = \frac{\sqrt{2\Pi z} z^z}{e^z}(1 + \Theta(1/z))$$

and that

$$\sqrt{2\Pi z}\left(\frac{z}{e}\right)^z \leq z! \leq \sqrt{2\Pi z}\left(\frac{z}{e}\right)^{z+\frac{1}{12z}}$$

Hence,

$$z! > \left(\frac{z}{e}\right)^z$$

for all $z \geq 1$. Plugging in and simplifying, we find that

$$\binom{x}{y} < \left(\frac{xe}{y}\right)^y$$

as desired. □

An Example of the Perturbation Step

We now show an example of applying the perturbation step to the oblivious schedule as given in Figure 3.1. Random start times are given to each starting step of a file (*i.e.*, those with a starting time of 0). While the method of assigning the uniform random start times is not important, the reader may observe that this was done by assigning start times cyclically from 0 to 5 starting at the left of the graph and moving around clockwise. Figure 3.3 shows the perturbed starting times.

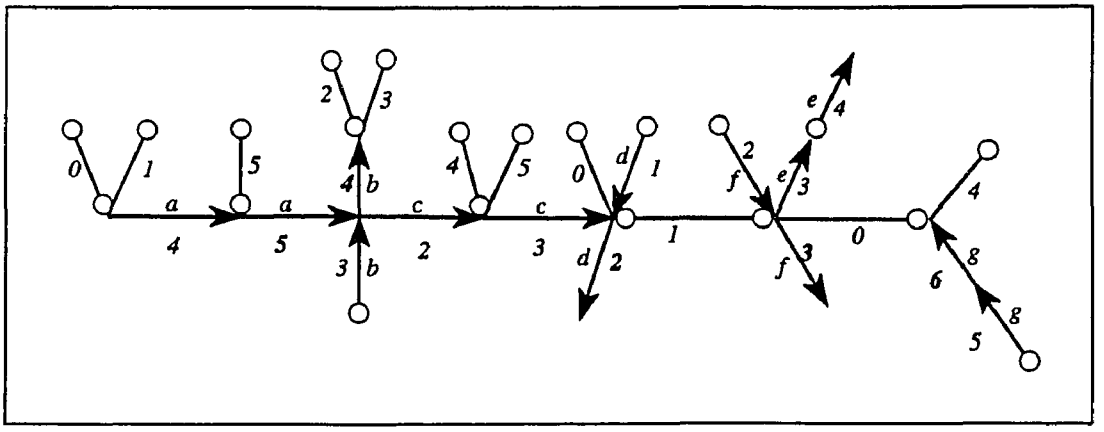

Figure 3.3: Perturbed Schedule

The times in bold face are those of file steps after the inital starting step. For example, file a was assigned a random start time of 4. Its next forwarded step, therefore, is assigned the next time unit, 5. Note that conflicts have indeed arisen in the perturbed schedule, as shown in Figure 3.4. This is due not only to conflicting random times assigned (e.g., the first step of file d at time 1), but but also because of conflicts between the random time assigned to one file and the time assigned to a following step of another file (e.g., the second step of file a). These conflicts will now be resolved by the flattening algorithm.

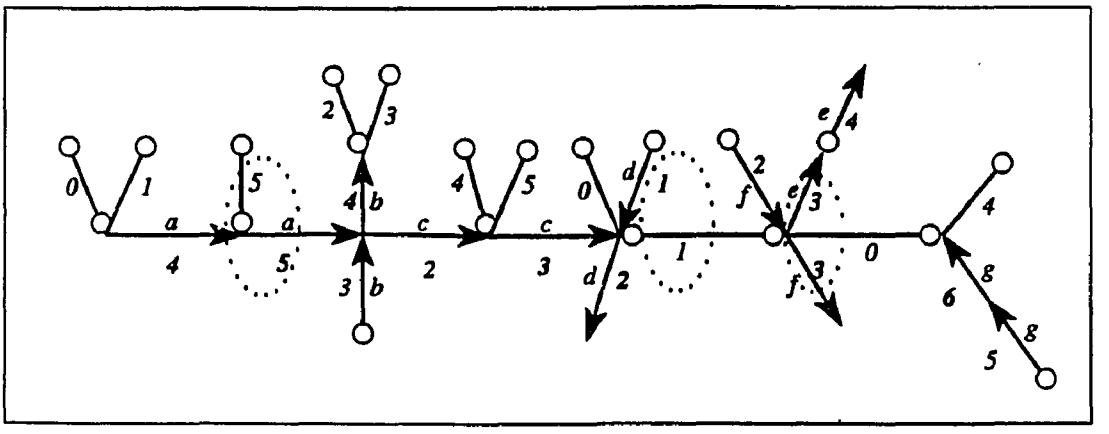

Figure 3.4: Conflicts After Perturbation

The Flattening Algorithm

After the schedule has been perturbed, there are at most $O(\frac{\log n}{\log \log n})$ files scheduled at conflicting times at any node. The next step is to assign each of those files a unique, non-conflicting time to legitimize the schedule. This will be accomplished by treating the edges in the graph scheduled at duplicate times as edges which require unique values, such as colors, and using a graph coloring algorithm to resolve the conflicts.

Suppose two files, f_1 and f_2, are scheduled at the same time t after the schedule is perturbed. After coloring, one of the files, say f_1, would retain its scheduled time t, and f_2 would be scheduled at $t+1$. It is then possible that f_2 would now conflict with other files which were scheduled at time $t+1$. To avoid all such conflicts, we can increase the scheduled time of all files which were scheduled at time $t+1$ or later by 1, causing a global increase in the length of the entire schedule by one time unit. This would avoid all potential conflicts with f_2.

If several files at a node, $f_1, ...f_i, ...f_x$ are scheduled at the same time, t, then each would need to be assigned a unique color, requiring at least x colors (i.e., x time units). This would increase the length of the overall schedule by x time units.

What is of such importance here is that the first two steps of the global scheduling algorithm (creating oblivious and perturbed schedules) can be done in polynomial time, and will not extend the approximated schedule by more than $O(\frac{\log n}{\log \log n})$ of the optimal schedule. Using edge coloring of graphs as a vehicle to flatten out all the conflicting times in the perturbed schedule is an efficient solution for several reasons. First, graph coloring algorithms are well-known and have been studied in detail. Indeed, an entire book has been written by Fiorini and Wilson on edge coloring alone [3]. Second, *edge* coloring (opposed to *vertex* coloring) is appropriate because it is the edges of file transfer graphs which are being labelled (either with *time* or with a *color*). Thirdly, edge coloring algorithms work on the entire graph at once, making them suitable for a global approximation algorithm. Finally, and perhaps most importantly and most fundamental to their application to the problem of file forwarding, is that the number of colors required to edge-color a graph (called the *chromatic index*) is related to the maximum degree of the graph.

In his famous theorem, Vizing [1, 3, 16, 6] shows that the chromatic index, ψ, of a graph is either equal to the maximum degree or to the maximum degree plus one:

$$MaxDegree \leq \psi \leq MaxDegree + 1$$

It should be pointed out that to *decide* if the graph's chromatic number is equal to the maximum degree or equal to the maximum degree plus one is NP-Complete. With no loss of generality (but with, perhaps, the loss of one unit of time) we will presume that the chromatic number is the maximum degree plus one. That is, to color a graph with maximum degree x, we will presume that $x + 1$ colors are necessary, even though x colors may have been sufficient.

It is possible to use coloring to solve the flattening problem in the following way. The perturbed schedule has left us with some number of conflicting files, c, files that are all to be processed at the same time, t. Using no more than

$c + 1$ time units, we can start at with the conflicting time, t, and assign non-conflicting times to all files in the file transfer graph that conflicted at time t. Of course, any files (in the perturbed schedule) that previously were scheduled to be processed at that next time unit $(t + 1)$ will need to have their processing times adjusted by adding $c + 1$ so that they don't get processed until the last of the c conflicting files is finished. This is necessary because the file processed at time $t + 1$ may be a forwarding step to occur after a step of one of the c files. All files originally in the perturbed schedule at time $t + 1$ will now be scheduled at $t + 1 + (c + 1)$. Indeed, all files in the transfer graph need to have their perturbed times increased by $c + 1$ for the same reason. Consider the perturbed file transfer graph as was seen in Figure 3.4. There are 2 files which conflict at time 1. No more than 2 time units would be required to resolve this conflict. Figure 3.5 shows a possible labelling for the flattened time units. Note that the times for the other steps are increased by 2 (for example, file a's time was 4 in

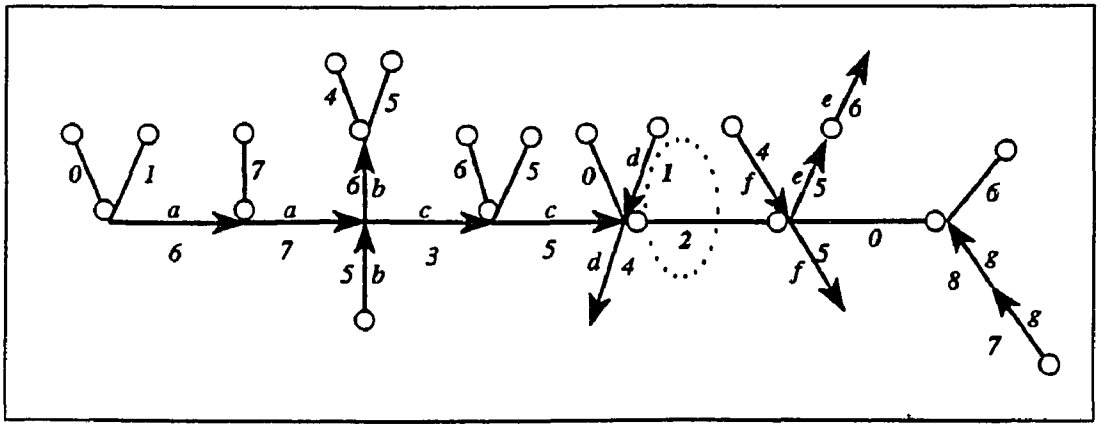

Figure 3.5: Conflicts at Time 1 Flattened

Figure 3.4, and is now 6 in Figure 3.5. Originally, there were conflicts at time 3 in Figure 3.4. Those conflicts still exist, but since we added 2 to each time (above 1), these conflicts are now at time 5. It is therefore necessary to color all the conflicts at time 5 (formerly conflicts at time 3). This is shown in Figure 3.6. The first step of file f is "colored" 5, and the the second step of file f is now

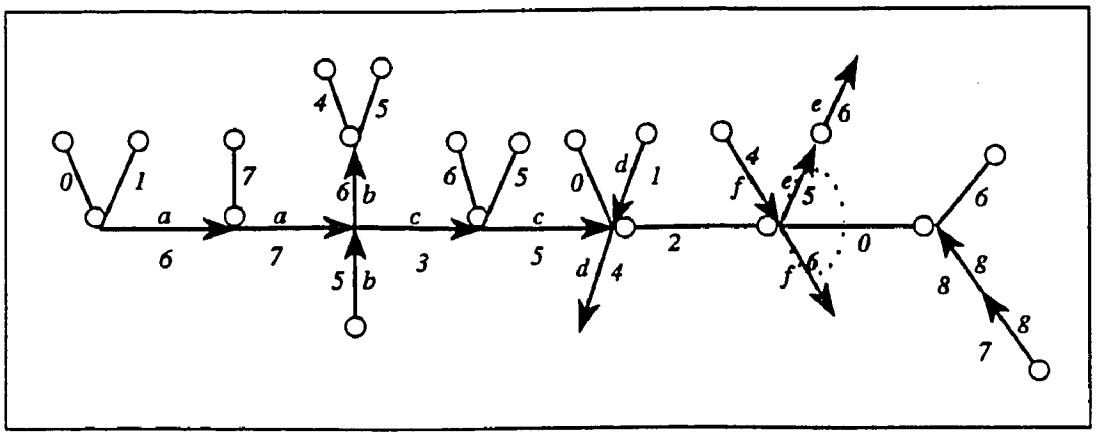

Figure 3.6: Conflicts at Time 5 Flattened

"colored" 6. This will add two (since two colors were used) to all times greater than 5. Finally, the remaining conflict (now at time 9) will be colored, using two colors and adding two to all times greater than 9 (Figure 3.7). As Vizing's

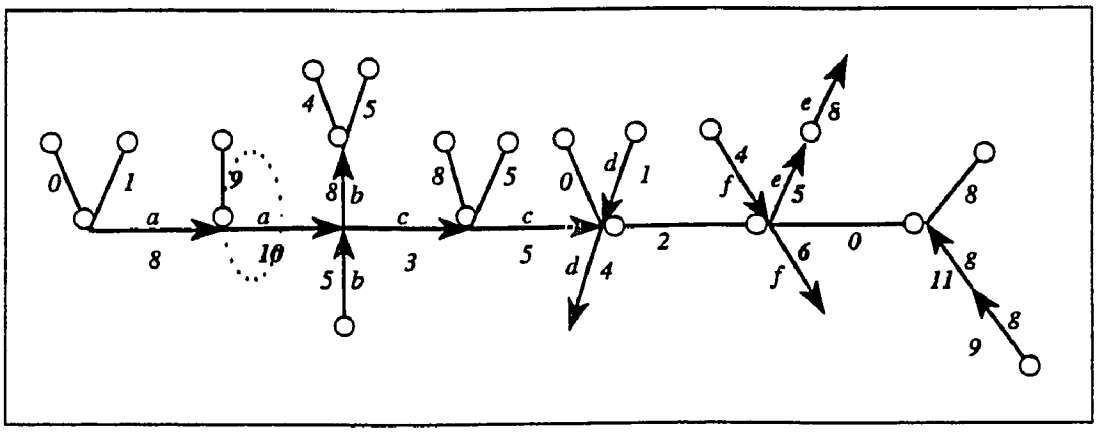

Figure 3.7: Conflicts at Time 9 Flattened

proof is constructive, there is an algorithm for coloring with order $O(n^3)$. Much research has been done to improve on coloring algorithms. For example, Garey and Johnson have explored the complexity [4], and Leighton [9] has devised an algorithm which works in $O(n^2)$ on most sparse graphs. It would be reasonable to presume the "sparseness" of file transfer graphs undergoing the perturbation and flattening algorithm because (1) a natural application in computer networks would find that computer installations serving as nodes would be connected via

```
While there are time conflicts at any node
     for each time t=0,1,...
        Assign non-conflicting sequential times to
           each file conflicting at time t using
           edge coloring. Let c represent the number
           of colors (time units) needed to do so.
        Add c to the scheduled time for all files
           which were scheduled at times greater
           than t.
```

Figure 3.8: The Flattening Algorithm

communication lines to a few other computers and, (2) the coloring would be applied to a subgraph consisting only of edges involved in the coloring, which is $O(\frac{\log n}{\log \log n})$

As files are flattened, what will this do to the schedule? Clearly, every time the coloring algorithm eliminates c conflicts the schedule may be lengthened by $c+1$ time units. The number of times that the coloring algorithm needs to be run is equal to the $MaxDegree$ plus the length of the longest file, n, as shown in Lemma 3.3.

Lemma 3.3 *Considering a File Transfer Graph with maximum vertex degree M and length of longest file transferred n, the maximum number of times that the coloring algorithm needs to be applied is M+n.*

Proof. Consider a file f departing from node a_i. This file is subject to being delayed by all the other files at a_i. There will be $M-1$ of these files as the total number of files can be no more than M. If these other files can delay f, then they must be scheduled at conflicting times. Once these conflicts are resolved by the assigning of non-conflicting times by the coloring algorithm, f will depart node a_i at some later time, say t_f. File f can be delayed at no further nodes because

the entire graph has been colored and all conflicting times have been removed. Only files which have not had their conflicts resolved could cause a delay for f. There can be no such files left as M conflicts have already been resolved, with t_f begin the latest scheduled time. Since M is the maximum degree, there can be no more files anywhere in the graph. File f may proceed unimpeded to its destination, which will require n steps at most (as n is the longest file length). Therefore, the total delay is $M + n$. □

If the number of times the coloring algorithm will be applied is $M + n$, and at each application the length of the schedule is increased by the number of conflicting files, c, plus one, then the total length of the schedule after coloring will be $(M+n)(c+1)$. The order increase to the schedule, therefore, is $O(c)OPT$ as shown in Lemma 3.4.

Lemma 3.4 *Given a schedule of length L with at most c files processed by a node at any time, there exists a polynomial time algorithm that produces a valid schedule of length O(c)OPT.*

Proof. As shown in Lemma 3.3, the length of the schedule, "L", will be $(M + n)(c+1)$ after applying the coloring algorithm (which can be done in polynomial time). Therefore,

$$L \leq (M+n)(c+1)$$

Given that $n \leq OPT, M \leq OPT$

$$\leq 2OPT(c+1)$$

$$= O(c)OPT$$

□ Taking advantage of the fact that the number of conflicts, c, as shown in Lemma 3.1, is $O(\frac{\log n}{\log \log n})$, and that $MaxDegree$ and n are bounded by the optimal schedule, OPT, the maximum schedule length is $O(\frac{\log n}{\log \log n})OPT$ as shown in Theorem 3.1.

Theorem 3.1 *Using the perturbation and flattening algorithm on a File Transfer Graph with number of nodes and maximum degree bounded by a polynomial in n and with maximum file length n, the maximum schedule length will be $O(\frac{\log n}{\log \log n})OPT$.*

Proof: By Lemma 3.4, we know that the maximum schedule length, L is of order $O(c)OPT$. By Lemma 3.1, we know that, with high probability, for c (the number of conflicts after perturbation) $c = O(\frac{\log n}{\log \log n})$, Thus,

$$\begin{aligned} \mathsf{L} &= O(c)OPT \\ &= O(\frac{\log n}{\log \log n})OPT \end{aligned}$$

□

3.2 A Deterministic Algorithm

We now show that the delays assigned to the files can be done deterministically in polynomial time such that no node processes more than $O(\log n)$ files at any time. This will be done by using a case of an integer program used to solve the vector selection problem.

Integer Programming

Integer Programming Problems often model a set of linear equations which are to be solved within some minimum or maximum. For example, in the Knapsack problem it is desired to fill a knapsack with items that each have a weight and a value. Food, for example, may have some weight and be of very high value. The problem is to fill the knapsack with so that the weight is beneath some threshhold and the items are the most valuable. There are many other problems which can be solved using integer programming because their solutions involve choosing from among many values and achieving some kind of minimum or maximum. A good discussion is given by Moret and Shapiro [12].

Raghavan [13] describes the integer packing program as it applies to vector selection as follows. Let Λ be a collection of sets of vectors, $\Lambda = \{\lambda_1, ..., \lambda_r\}$. Each set λ_j consists of n-vectors $\{V_1, ..., V_k\}$, where $k_j = |\lambda_j|$. For $1 \leq i \leq n$, the ith component $v_j^i(i)$ of vector V_k^j is either 0 or 1. We are to choose exactly one vector from each set λ_j; we denote by V^j the vector chosen from λ_j. We wish to minimize $\left\| \sum_{j=1}^{r} V^j \right\|_\infty$.

This can be formulated as an integer program by using an *indicator* (0-1) variable x_i^j to indicate whether or not the vector V_k^j is selected to represent λ_j ($1 \leq j \leq r$ and $1 \leq k \leq k_j$). The integer program is to choose one vector from each set such that the sum of the columns,W, is minimized, subject to

$$x_k^j \in 0, 1$$

and that there is only one V^j from each set λ_j:

$$\sum_{k=1}^{k_j} x_k^j = 1, \quad 1 \leq j \leq r$$

and that the sum of each of the columns of the selected vectors is $\leq W$:

$$\sum_{j=1}^{r} \sum_{k=1}^{k_j} x_k^j \cdot v_k^j(i) \leq W, \quad 1 \leq i \leq n$$

This integer program can be shown to be NP-hard (by a reduction as shown by Kramer and van Leeuwen [7]). It can, however, be shown that there is a deterministic polynomial time approximation algorithm which obtains a solution within $O(\log \mu)$ (where μ is the length of the vector) of the optimal solution.

Letting W' represent the optimal solution, the integer packing algorithm yields an approximate solution, W^I, with two cases, depending on W':

$W' < \log \mu$:

$$W^I \leq \left\lceil W' + (e-1)[W' \log \mu]^{\frac{1}{2}} \right\rceil$$

$W' \geq \log \mu$:

$$W^I \leq \left\lceil \frac{W' + e \log \mu}{\log(\frac{e \log \mu}{W'})} \right\rceil$$

Raghavan [13] proves that the integer solution is bounded above by the optimal solution plus a linear addition of order $O(\log \mu)$, where μ represents the length of the vectors:

$$W^I \leq \left\lceil W' \cdot \left[1 + D(W', \frac{1}{\mu}) \right] \right\rceil$$

where

$$D(m,x) \leq (e-1)\left\lceil \frac{\ln \frac{1}{x}}{m} \right\rceil^{\frac{1}{2}} \quad , m > \ln \frac{1}{x}$$

$$D(m,x) \leq \frac{e \ln \frac{1}{x}}{m \ln[(e \ln \frac{1}{x})/m]} \quad , m \leq \ln \frac{1}{x}$$

Integer Packing and File Scheduling

Integer packing can be modified for the application of file forwarding. For all files ready at time 0 (from the *oblivious* schedule), we wish to assign a delay deterministically in the range [0, $\frac{MaxDegree}{\log n}$]. The maximum length of a file's schedule, l is therefore the length of the file plus the maximum delay, thus l = n+$\frac{MaxDegree}{\log n}$. The complete "history" of the schedule of a file can be represented by recording which node (of all the N nodes in the graph) processes it and at what time, t, it is processed ($0 \leq t \leq l$). This representation can be considered as a vector of N groups of l ones and zeros (i.e., an ($N \times l$)-length (0,1) vector).

Figure 3.9 shows one vector that represents the history of the pictured file, f. For each of the three nodes, a_0, a_1, a_2, there are l ones and zeros. At node

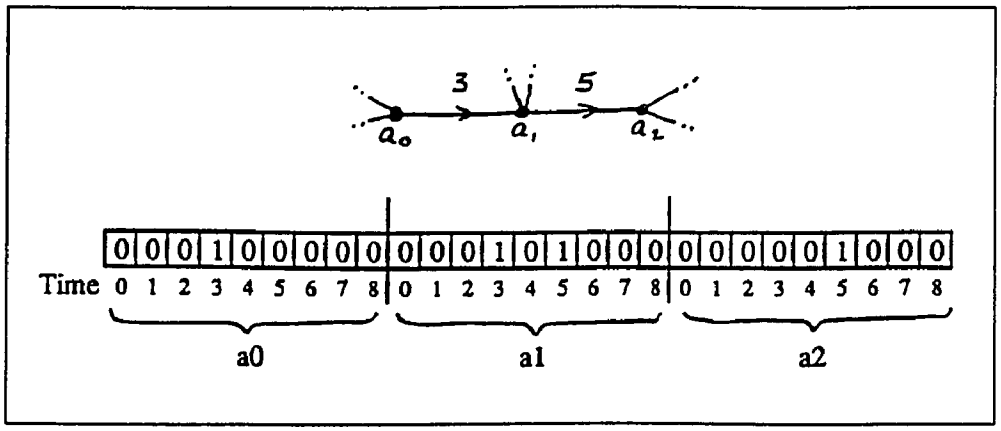

Figure 3.9: History of One File

a_0, the 1 at time unit 3 indicates that a_0 is processing a file at time 3. Node a_1 also has a 1 at time 3, as it is receiving the same file. Similarly, the 1 at time 5 in a_1 and a_2 indicated the transmission of file f from a_1 to a_2. Thus, the entire history of file f as it is forwarded from a_0 to a_2 is recorded in an ($N \times l$)-length (0,1) vector.

In the integer packing problem, a file's schedule, considered as an ($N \times l$)-

length (0,1) vector, is one of the V_k^j vectors. The entire set of vectors, λ_j for one file consists of one V_k^j vector for the file as it would appear with each possible delay d applied, $d = 0, 1, ...\frac{MaxDegree}{\log n}$. The set Λ, therefore, contains a λ set of vectors for each file in the transfer graph.

Figure 3.10 shows a small transfer graph where file f is going from a_0 to a_2. The λ set for this file contains four vectors representing the histories of f as

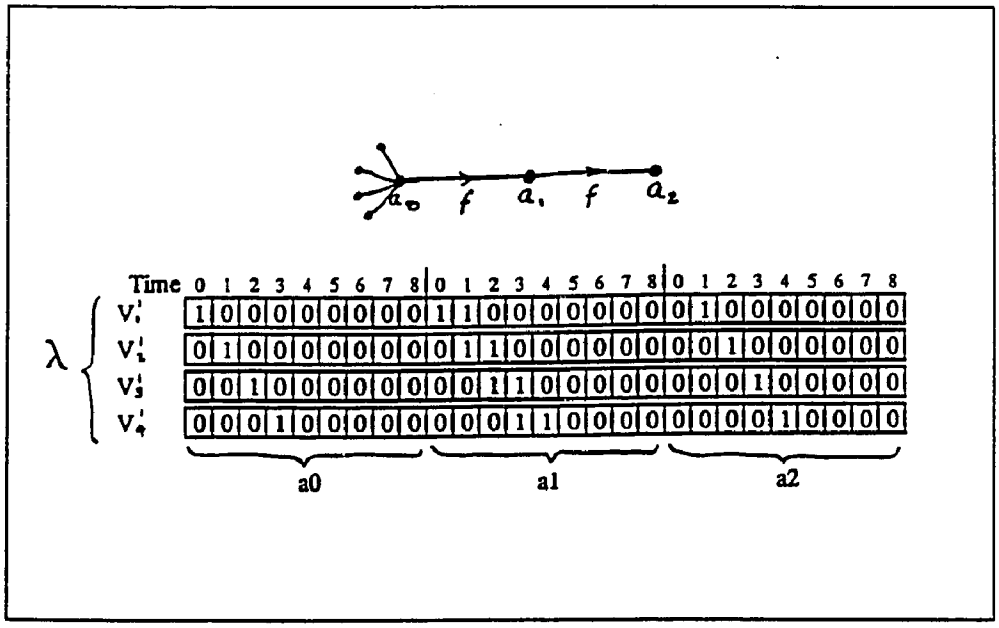

Figure 3.10: All Histories of One File

f is subjected to each delay in $[0, \frac{MaxDegree}{\log n}]$ (*i.e.*, delays of *0,1,2 and 3*). The Λ set for the file transfer graph would contain a λ set for each file in the graph. Figure 3.11 shows a possible *Lambda* set.

The integer packing problem will select one vector from each of the λ sets such that the sum of the columns of the selected vectors is minimal. When the λ vectors represent file histories, one column of all the selected vectors represents *one* node at *one* time, where a value of 1 indicates that the node is busy, and 0 that it is not. The sum of the ones in the column, therefore, is the number of items that node is processing at that time. In the optimal schedule, this minimum should be 1 as each node can process only one file at a time. The production

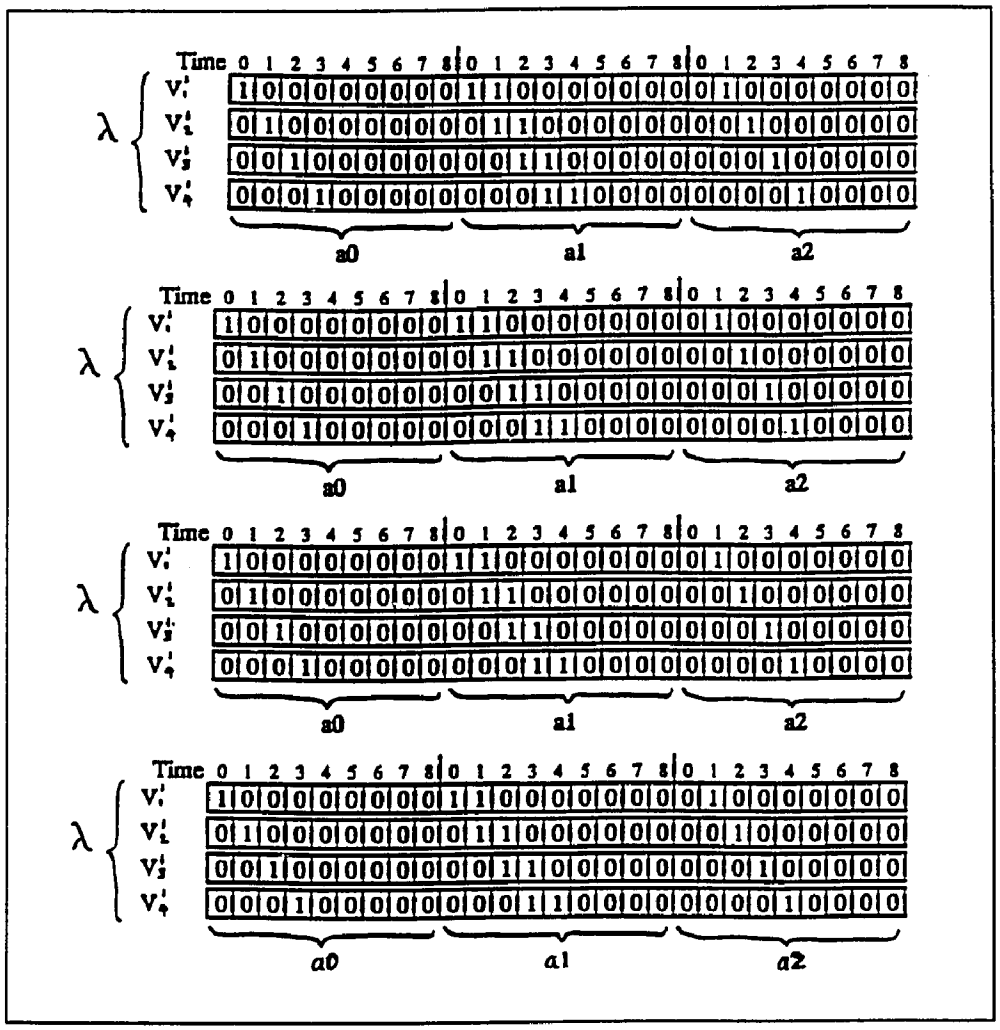

Figure 3.11: All Histories of All Files

of this optimal schedule, as previously stated, is intractable. It can be shown, however, that it can be approximated within $O(\log n)OPT$.

The deterministic integer packing program can produce a schedule within $\log n$ of OPT. Let the length of our vectors $(N \cdot l)$ be represented by μ. For either case of Ravaghan's approximation algorithm, the approximate minimum using the integer program will be no more than an O(log n) increase over the optimal minimum. Thus, the minimum number of files in contention for a node in a file transfer graph upon perturbation of the oblivious schedule will be no more than the optimal minumum plus $O(\log n)$. As has been shown in the previous section

using a randomized technique to minimize contention, the optimal minimization is also O(log n), and thus the minimum approximated by the integer program will be OPTIMAL + $O(\log n) = O(\log n) + O(\log n) = O(\log n)$

This argument also requires that $O(\log \mu) = O(\log n)$. This is true, as we are presuming that N, the number of nodes in the file transfer graph, is bounded by a polynomial in n (i.e., $N \leq an^k$ for some constant, a). We also presume $MaxDegree$ is bounded by a polynomial in n, and thus the length of a file's schedule, l, (which is the number of forwarding steps for the file plus the maximum initial delay (i.e., $n + \frac{Maxdegree}{\log n}$)) is obviously bounded by n. Thus,

$$O(\log \mu) = O(\log n)$$

For each case of Ravaghan's approximation which determines the bound for the integer programming solution, we obtain:

Case 1: $W' < \ln \mu$

$$\begin{aligned}
W^I &\leq \left\lceil W' \cdot \left[1 + D(W', \frac{1}{\mu})\right]\right\rceil \\
&\leq \left\lceil W' \cdot \left[1 + (e-1)\left[\frac{\ln \mu}{W'}\right]^{\frac{1}{2}}\right]\right\rceil \\
&\leq \left\lceil W' + W' \left[(e-1)\left[\frac{\ln \mu}{W'}\right]^{\frac{1}{2}}\right]\right\rceil \\
&\leq \left\lceil W' + (e-1)[W' \ln \mu]^{\frac{1}{2}}\right\rceil \\
&= W' + O([O \log n) \cdot O(\log n)]^{\frac{1}{2}} \\
&= W' + O(\log n) \\
&= O(\log n)
\end{aligned}$$

Case 2: $W' \geq \ln \mu$

$$W^I \leq \left\lceil W' \left[1 + \frac{e \ln \mu}{W' \ln[(e \ln \mu)/W']}\right]\right\rceil$$

$$\begin{aligned} &\leq W' + e \ln \mu \\ &= W' + O(\log n) \\ &= O(\log n) \end{aligned}$$

The integer solution, W^I, can be obtained in polynomial time. This solution represents the maximum number of contentions to be allowed at a node under a perturbed schedule.

It is possible to calculate this schedule in deterministic polynomial time, as shown in theorem 3.2:

Theorem 3.2 *A delay in the range* $\left[0, \frac{MaxDegree}{\log n}\right]$ *can be deterministically assigned in polynomial time to each file at each node in a File Transfer Graph with number of nodes and maximum degree bounded by a polynomial in n so as to produce a schedule where no node processes more than* $O(\log n)$ *files at any time.*

Proof. This can be formulated as an integer programming problem with approximated solution (W^I), calculated in polynomial time, where $W^I \leq O(\log n)$. □

Corollary 3.1 *A maximum schedule length of* $O(\log n)OPT$ *can be calculated in deterministic polynomial time.*

Proof. By Theorem 3.1, a maximum schedule length of $O(\log n)OPT$ may be obatined using the randomized algorithm. Replacing the randomized perturbation step (which produces an $O(\log n)$ length schedule) with the deterministic perturbation step in Theorem 3.2 (which also produces an $O(\log n)$ length schedule) does not change the maximum schedule length. □

Theorem 3.3 *There exists a deterministic algorithm that will produce a schedule of length no more than $O(\log n)OPT$ when the number of forwarding steps, n, the maximum degree of a node, $MaxDegree$, and the number of nodes in the file transfer graph, N, are bounded above by a polynomial in n.*

Proof. This can be accomplished by the perturbation/flattening algorithm. Perturbation is done by the deterministic integer packing program in polynomial time. It will extend the schedule by no more than $O(\log n)OPT$. Flattening can be accomplished by the coloring algorithm in polynomial time and extending the schedule by no more than $O(\log n)$OPT time. Thus, the schedule may be produced in polynomial time by deterministic algorithms, lengthening the schedule by no more than $O(\log n)$OPT □

Bibliography

[1] B. Bollobás. *Graph Theory.* Springer-Verlag, New York, N.Y., 1979.

[2] E. G. Coffman, M. R. Garey, and D. S. Johnson. Scheduling file transfers. *SIAM Journal of Computing*, 14(3):744–780, August 1985.

[3] S. Fiorini and R. J. Wilson. *Edge-colourings of Graphs.* Pitman Publishing Ltd., London, England, 1977.

[4] M. R. Garey and D. S. Johnson. The complexity of near optimal graph coloring. *J.A.C.M.*, 23:43–49, 1976.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, California, 1979.

[6] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, Cambridge, Mass., 1985.

[7] M. R. Kramer and J. van Leeuwen. Wire-routing is NP-Complete. Technical Report RUU-CS-82-4, Rijksuniversiteit Utrecht, 1982.

[8] R. Kruse. *Data Structures and Program Design.* Prentice-Hall, Englewood Cliffs, N.J., third edition, 1994.

[9] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[10] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, California, 1992.

[11] F. T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[12] B. E. Moret and H. Shapiro. *Algorithms from P to NP.* Benjamin/ Cummings, Redwood City, N.J., 1991.

[13] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[14] P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe. Scheduling data transfers in fully connected networks. *Networks*, 22:563–588, 1992.

[15] D. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal of Computing*, 23(3):617–632, June 1994.

[16] M. N. S. Swamy and K. Thulasiraman. *Graphs, Networks, and Algorithms.* John Wiley and Sons, New York, N.Y., 1900.

[17] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall, Englewood Cliffs, N.J., second edition, 1988.

[18] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal of Computing*, 11(2):350–361, May 1982.

[19] J. Whitehead. The complexity of file transfer scheduling with forwarding. *SIAM Journal of Computing*, 19(2):222–245, April 1990.