On Distributed Network Resource Allocation

Andréa Werneck Richa

CMU-CS-98-146

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee

Bruce M. Maggs, *Chair*
Alan Frieze
R. Ravi
Greg C. Plaxton, *Univ. of Texas at Austin*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored in part by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute; an equipment grant from Sun Microsystems; DIMACS; and NEC Research Institute.

Keywords: Distributed network, resource allocation, approximation algorithm, randomized algorithm, shared object, wide-area network, hierarchical model, expected cost, packet routing, scheduling, Lovász Local Lemma, optimal schedule, network emulation, embedding, congestion, dilation, ordering, linear arrangement, linear array, combinatorial optimization, interval graph, storage–time product, spreading metric, planar graph.

Abstract

This thesis addresses several resource allocation problems that arise in the context of distributed networks. First, we present a scheme for accessing shared copies of objects in a network that has asymptotically optimal expected cost per access for a class of cost functions that captures the hierarchical structure of most wide-area networks. Second, we present an off-line polynomial-time algorithm that finds an asymptotically optimal schedule for the movement of packets whose paths through a network have already been determined. This is an improvement on a previous result by Leighton, Maggs, and Rao, who proved the existence of such schedules; their proof, however, was not constructive. Finally we present a polynomial-time $O(\log n)$-approximation algorithm for finding an embedding of a network with n processors into an n-node linear array so as to minimize the weighted sum of the edge dilations — i.e., for the minimum linear arrangement problem. This problem is NP-hard, and the previous best approximation bound known was $O(\log n \log \log n)$. In the case of planar networks, we bring the approximation factor down to $O(\log \log n)$. We also extend our approximation techniques to the minimum storage–time product and the minimum containing interval graph problem.

To Aykut,

Contents

List of Figures

Acknowledgments

First of all, I would like to thank Aykut, for his full and unconditional support during the course towards my PhD degree, and who in spite of enduring me through this thesis work, still married me. Second, I would like to thank Bruce, my great friend and advisor, who helped me gain the independence in research essential for earning a doctorate. I would like to thank my parents Thereza and José, and my brother Guilherme, whose support I could always feel so close, in spite of their being thousands of miles away. I would like to thank Greg Plaxton, Alan Frieze, and R. Ravi, whose feedback was so crucial for the conclusion of this dissertation. I would also like to thank Satish Rao, Rajmohan Rajaraman, and Tom Leighton, as well as Bruce and Greg, for their most valuable contribution to the work in this thesis. I would like to thank Danny Sleator, for helping me out in particular at the beginning of my PhD. I would like to thank Claudson Bornstein, for his "help-with-all" during the course of our PhD. For standing all my ups-and-downs in our Wean Hall office, I would like to thank Anja Feldmann, Chris Stone, Jeff Polakow, and Adrian Perrig. I would like to thank all the people in the Algorithms, Combinatorics and Optimization PhD Program, and all the people in the Computer Science Department at Carnegie Mellon University — faculty, staff, and graduate students altogether (in particular I would like to thank Sharon Burks, Dorothy Zabrowsky, Karen O'lack, Catherine Copetas, Gary Miller, Merrick Furst, and Ravi Kannan). I would like to thank Pınar Keskinocak and Bertrand Guenin, for their unconditional support, for being such very special friends and classmates from day one at Carnegie Mellon University. I cannot forget to also thank Bülent Başaran, Jürgen Dingel, and the Sinharoys, for their full support and unmatchable friendship. Last, but not least, I would like to thank all of my friends — in particular all of the wonderful friends I made in Pittsburgh, whose friendship is so closely related to my PhD course — and family, for their most valuable support.

Chapter 1

Introduction

The advent of high-speed distributed networks has made it feasible for a large number of geographically dispersed computers to cooperate and share information (e.g, messages, files, control data). Indeed, the last few years have seen the emergence of large distributed databases, such as the World Wide Web, and more generally, of a variety of distributed network applications that rely on communication for performing their basic tasks. The distributed nature of the databases and the rapidly growing demands of the users have in turn overloaded the underlying network resources (e.g., links, memory space at the processors, buffer space at the links and processors).

In an attempt to minimize communication delays and to satisfy as many users as possible, strategies for making efficient use of network resources have been devised. As, for example, in this thesis, where efficient resource allocation strategies are used to obtain efficient solutions to three problems that arise in the context of distributed networks.

The first problem we consider in this thesis is the one of efficiently supporting requests for shared objects (e.g., files, pages of memory) that have been distributed among the processors (nodes) of a wide-area network. Multiple copies of each object may exist in the network. In particular, we would like to devise a protocol that satisfies each request for an object with a "nearby" copy of the object, since this ensures fast response times and minimizes the cost incurred in accessing the object. Chapter 2 presents a protocol that achieves asymptotically optimal expected cost for satisfying a request for an object while making efficient use of the memory space at each node, for a class of cost functions that captures the hierarchical structure of most wide-area networks.

Next, we study the movement of packets in a network: More specifically, we consider the problem of scheduling the movement of packets whose edge-simple[†] paths through a network have already been determined. This problem arises in a scenario where nodes of the network exchange information via point-to-point communication paths. In Chapter 3 of this thesis, we present a polynomial-time algorithm that finds an asymptotically optimal schedule for routing the packets along the given paths. This is an improvement on a previous result by Leighton, Maggs, and Rao [26], who proved the existence of such schedules; their proof, however, was not constructive.

The problem of scheduling the movement of packets in the network relates to network emulations that are performed via network embeddings, as we will see. Network emulations and embeddings will also be addressed in Chapter 4, where we consider embeddings of networks into the linear array. Thus we briefly discuss emulations and embeddings in the paragraphs that follow. For a more complete discussion of emulations and embeddings, see [24]; see also [32].

We can model the topology of a network as a graph $G(V, E)$, where each node in V uniquely represents a processor of the network, and where each edge (x, y) in E uniquely represents a communication link between the processors corresponding to nodes x and y in the network. Throughout this thesis, we will implicitly use this model, interchangeably referring to a network as a graph, and to processors and links of the network as nodes and edges respectively.

A guest network G can be emulated by a host network H by embedding G into H. An *embedding* maps nodes of G to nodes of H, and edges of G to paths in H — an edge (x, y) of G is mapped to some path in H between the nodes of H that x and y were mapped to. There are three important measures of an embedding: the load, congestion, and dilation. The *load of an embedding* is the maximum number of nodes of G that are mapped to any one node of H. The *congestion of an embedding* is the maximum number of paths corresponding to edges of G that use any one edge of H. The *dilation of an embedding*

[†]An *edge-simple* path uses no edge (i.e., link of the network) more than once.

is the length of the longest path of H in the embedding. Let l, c, and d denote the load, congestion, and dilation of the embedding, respectively.

Once G has been embedded in H, H can emulate G in a step-by-step fashion as follows. Each node of H first emulates the local computations performed by the l (or fewer) nodes mapped to it. This takes $O(l)$ time. Then for each packet sent along an edge of G, H sends a packet along the corresponding path in the embedding. Using the algorithm presented in Chapter 3, H can emulate each step of G in $O(l + c + d)$ steps.

We address a problem that relates to embeddings of networks into the linear array in Chapter 4. Suppose a network G with n nodes is embedded one-to-one (with respect to the mapping of its nodes) into a network H. The *dilation of an edge* of G in the embedding is the length of the path of H that this edge is mapped to. We would like to be able to minimize the average edge dilation of the embedding, since high average dilation may incur high average cost of communication. Unfortunately, the problem of determining if there exists an embedding with average edge dilation d', for any $d' > 0$, is NP-hard even for the case when the host network is a linear array. A generalization of this problem is to assign nonnegative weights to each edge of G, which may represent the amount (or the cost) of communication through that edge; in this case, we would like to minimize the average weighted edge dilation of the embedding.

In Chapter 4, we present a polynomial-time $O(\log n)$-approximation algorithm for finding a one-to-one embedding of a graph with n nodes into the n-node linear array so as to minimize the weighted sum of the edge dilations. An embedding that has minimum weighted sum of edge dilations is called a *minimum linear arrangement*. If the network is a planar graph, we obtain an improved approximation factor of $O(\log\log n)$.

We conclude Chapter 4 by extending the ideas used for approximating the minimum linear arrangement problem to obtain $O(\log n)$-approximation algorithms for two other problems that involve finding a linear ordering of the nodes of a graph: the problems of finding a minimum storage–time product, and of finding a minimum cost containing interval graph of a given graph. For the latter problem, in case the input graph is planar, we also obtain an improved approximation bound of $O(\log\log n)$.

Each of the following chapters is self-contained. Concluding remarks and suggestions of future work regarding any of the problems considered will be presented at the end of the relevant chapter.

1.1 Notation

In this section we introduce some basic notation that will be used in this thesis. Throughout this thesis, for any positive integer x, we use $[x]$ to denote the set $\{0, \ldots, x-1\}$; and for any integers a and b, we let $[a, b]$ denote the set $\{k \in \mathbf{Z} : a \leq k \leq b\}$. Also, all logarithms are to base 2, unless otherwise specified.

In the context of randomized algorithms (Chapters 2 and 3), we use "with high probability" to mean "with probability at least $1 - n^{-c}$, where n is the number of nodes in the network and c is a constant that can be set arbitrarily large by appropriately adjusting other constants defined within the relevant context."

Chapter 2

Accessing Nearby Copies of Replicated Objects in a Distributed Environment

2.1 Introduction

As one might expect, the task of designing efficient algorithms for supporting access to shared objects (e.g., files, pages of memory) over wide-area networks is challenging, both from a practical as well as a theoretical perspective. With respect to any interesting measure of performance (e.g., latency, throughput), the optimal bound achievable by a given network is a complex function of many parameters, including edge delays, edge capacities, buffer space, communication overhead, and patterns of user communication. Ideally, we would like to take all of these factors into account when optimizing performance with respect to a given measure. However, such a task may not be feasible in general because the many network parameters interact in a complex manner. For this reason, we adopt a simplified *cost model* for a network, in which the combined effect of the detailed network parameter values is assumed to be captured by a single function that specifies the cost of communicating a fixed-length message between any given pair of nodes. We anticipate that analyzing algorithms under this model will significantly aid in the design of practical algorithms for modern distributed networks.

This is joint work with Greg Plaxton, University of Texas at Austin, and Rajmohan Rajaraman, DIMACS; a preliminary version of this work appears in [41].

Accessing shared objects. Consider a set $\mathcal{A}$ of m objects being shared by a network G, where several copies of each object may exist. In this paper, we consider the basic problem of *reading* objects in $\mathcal{A}$. Motivated by the need for efficient network utilization, we seek algorithms that minimize the cost of the read operation. We do not address the *write* operation, which involves the additional consideration of maintaining consistency among the various copies of each object. The problem of consistency, although an important one, is separate from our main concern, namely, that of studying *locality*. Our results for the read operation apply for the write operation in scenarios where consistency either is not required or is enforced by an independent mechanism.

We differentiate between *shared* and *unshared* copies of objects. A copy is shared if any node can read this copy; it is unshared if only the node that holds the copy may read it. We say that a node u *inserts* (resp., *deletes*) a copy of object A (that u holds) if u declares the copy shared (resp., unshared).

We refer to the set of algorithms for read, insert, and delete operations as an *access scheme*. Any access scheme that efficiently supports these operations incurs an overhead in memory. It is desirable that this overhead be small, not only because of space considerations, but also because low overhead usually implies fast adaptability to changes in the network topology or in the set of object copies.

The main source of difficulty in designing an access scheme that is efficient with respect to both time and space is the competing considerations of these measures. For example, consider an access scheme in which each node stores the location of the closest copy of each object in the network. This allows very fast read operations since a node knows the location of the closest copy of any desired object. However, such an access scheme is impractical because it incurs a prohibitively large memory overhead (proportional to the number of objects in the network), and every node of the network may have to be informed whenever a copy of an object is inserted or deleted. At the other extreme, one might consider an access scheme using no additional memory. In this case insert and delete operations are fast, but read operations are costly since it may be necessary to search the entire network in order to locate a copy of some desired object.

Our access scheme. We design a simple randomized access scheme that exploits locality and distributes control information to achieve low overhead in memory. The central part of our access scheme is a mechanism to maintain and locate the addresses of copies of objects. For a single object, say A, we can provide such a mechanism by the following approach. We embed an n-node "virtual" height-balanced tree T one-to-one into the network. Each node u of the network maintains information associated with the copies of A residing in the set of nodes that form the subtree of T rooted at u. Given the embedding of T, the read operation may be easily defined as follows. When a node u attempts to read A, u first checks its local memory for a copy of A or information about copies of A in the subtree of T rooted at u. If this local check is unsuccessful, u forwards the request for object A to its parent.

Naive extensions of the above approach to account for all objects require significant overhead in memory for control information at individual nodes. We overcome this problem by designing a novel method to embed the different trees associated with different objects. Our embedding enables us to define simple algorithms for the read, insert, and delete operations, and to prove their efficiency for a class of cost functions that is appropriate for modeling wide-area networks.

One important property of our access scheme is that it does not require location dependent naming of the copies of the objects, as we will see. Thus it avoids renaming a copy of an object every time this copy migrates (i.e., moves to another location in the network). Location dependent naming also poses a problem when keeping track of replicated objects, since copies of the same object located at different addresses in the network will have different names. A distributed shared object may be replicated in order to improve fault-tolerance or performance, for example. Other important properties of our scheme for the restricted class of cost functions considered are that (i) for its distribution of control information and of shared data, our scheme is expected to avoid "hot-spots" in the network (i.e., heavily accessed nodes); and (ii) for its distribution of data, combined with its support for object replication, and fast adaptability to changes in the network, our scheme is expected to scale well. Scalability is one of the most important problems to be solved in today's large-scale networks — for example, the World Wide Web, in spite of using scalable components (e.g., clients, servers, TCP/IP connections, DNS), has serious problems of scalability as a whole.

The cost model. As indicated above, we assume that a given function determines the cost of communication between each pair of nodes in the network. Our analysis is geared towards a restrictive class of cost functions which we believe to be of practical interest. The precise set of assumptions that we make with respect to the cost function is stated in Section 2.2. Our primary assumption is that for all nodes x and costs r, the ratio of the number of nodes within cost $2r$ of node x to the number of nodes within cost r of node x is bounded from above and below by constants greater than 1 (unless the entire network is within cost $2r$ of node x, in which case the ratio may be as low as 1).

There are several important observations we can make concerning this primary assumption on the cost function. First, a number of commonly studied fixed-connection network families lead naturally to cost functions satisfying this assumption. For example, fixed-dimension meshes satisfy this assumption if the cost of communication between two nodes is defined as the minimum number of hops between them; constant degree trees satisfy this assumption if the cost of communication between two nodes is given by the distance between these nodes in the physical layout (e.g., a wide-area layout, or a VLSI layout) of the tree.

Following the latter example, fat-tree topologies [30] satisfy our assumption if the cost of communication between two nodes is determined by the total cost of a shortest path between them, where the cost assigned to individual edges grows at an appropriate geometric rate as we move higher in the tree. Fat-trees are of particular interest here, because of all the most commonly studied fixed-connection network families, the fat-tree captures the hierarchical structure of most wide-area networks, and may provide the most plausible approximation to the structure of current wide-area networks.

Even so, it is probably inappropriate to attempt to model the Internet, say, with any kind of uniform topology, including the fat-tree. Note that our assumption on the cost function is purely "local" in nature, and allows for the possibility of a network with a highly irregular global structure. This may be the most important characteristic of our cost model.

Performance bounds. We show that our access scheme achieves optimality or near-optimality in terms of several important complexity measures for the restricted class of cost functions discussed above. In particular, our scheme achieves the following bounds:

- The expected cost for any read request is asymptotically optimal.
- If the number of objects that can be stored at each node is q, then the additional memory required is $O(q \log^2 n)$ words with high probability, where a word is an $O(\log n)$-bit string. Thus, if the objects are sufficiently large, i.e., $\Omega(\log^2 n)$ words, the memory for objects dominates the additional memory.
- The expected cost of an insert (resp., delete) operation at node u is $O(C)$ (resp., $O(C \log n)$), where C is the maximum cost of communicating a single word message between any two nodes.
- The number of nodes that need to be updated upon the addition or removal of a node is $O(\log n)$ expected and $O(\log^2 n)$ with high probability.

An obvious shortcoming of our analysis is that it only applies to the restricted class of cost functions discussed above. While we do not expect that all existing networks fall precisely within this restricted class, we stress that (i) our access scheme is well-defined, and functions correctly, for arbitrary networks, and (ii) we expect that our access scheme would have good practical performance on any existing network. (Although we have not attempted to formalize any results along these lines, it seems likely that our performance bounds would only degrade significantly in the presence of a large number of nontrivial violations of our cost function assumptions.)

Related work. The basic problem of sharing memory in distributed systems has been studied extensively in different forms. Most of the earlier work in this area — as in emulations of PRAM on completely-connected distributed-memory machines (e.g., [21, 54]) or bounded-degree networks (e.g., [45]), and algorithms for providing concurrent access to a set of shared objects [40] — assume that each of the nodes of the network has knowledge of a hash function that indicates the location of any copy of any object.

The basic problem of locating an object arises in every distributed system [37], and was formalized by Mullender and Vitányi [38] as an instance of the distributed matchmaking problem. Awerbuch and Peleg [5], and subsequently Bartal, Fiat, and Rabani [7] and Awerbuch, Bartal, and Fiat [3], give near-optimal solutions in terms of cost to a related problem by defining sparse-neighborhood covers of graphs. Their studies do not address the overhead due to control information and hence, natural extensions of their results to our problem may require an additional memory of m words at some node. However, we note that their schemes are designed for arbitrary cost functions, whereas we have focused on optimizing performance for a restricted class of cost functions.

In [6], Awerbuch and Peleg examine the problem of maintaining a distributed directory server, that enables keeping track of mobile users in a distributed network. This problem can be viewed as an object location problem, where objects migrate in the network.

In recent work, access schemes for certain Internet applications have been described in [18, 20, 55]. Some of the ideas in our scheme are similar to those in [55]; however, the two schemes differ considerably in the details. Moreover, the schemes of [18] and [55] have not been analyzed. As in our study, the results of [20] concerning locality assume a restricted cost model. However, their cost model, which is based on the ultrametric, is different from ours. Also, their algorithms are primarily designed for problems associated with "hot spots" (i.e., popular objects).

In [31], Maggs et al. investigate both the problem of determining the placement of copies of the objects in the network, and the problem of devising an efficient access scheme, with the main goal of keeping the edge congestion low. Their work considers cost models that arise in some restricted network topologies, such as trees, meshes, and clustered networks.

A closely related problem is that of designing a dynamic routing scheme for networks [4, 11]. Such a scheme involves maintaining routing tables at different nodes of the network in much the same way as our additional memory. However, in routing schemes the size of additional memory is a function of network size, i.e., n, while in our problem the overhead is primarily a function of m. Straightforward generalizations of routing schemes result in access schemes that require an additional memory of m words at each node.

The remainder of this paper is organized as follows. Section 2.2 defines the model of computation. Section 2.3 presents our access scheme. Section 2.4 contains a formal statement of the main results. Section 2.5 analyzes the algorithm and establishes the main results. Section 2.6 discusses directions for future research.

2.2 Model of computation

We consider a set V of n nodes, each with its own local memory, sharing a set $\mathcal{A}$ of $m = \text{poly}(n)$ objects. We define our model of computation by characterizing the following aspects of the problem: (i) objects, (ii) communication, (iii) local memory, (iv) local computation, and (v) complexity measures.

Objects. Each object A has a unique $(\log m)$-bit identification. For i in $[\log m]$, we denote the ith bit of the identification of A by A^i. Each object A consists of $\ell(A)$ words, where a word is an $O(\log n)$-bit string.

Communication. Nodes communicate with one another by means of messages; each message consists of at least one word. We assume that the underlying network supports reliable communication.

We define the cost of communication by a function $c{:}V^2 \mapsto \mathbf{R}$. For any two nodes u and v in V, $c(u, v)$ is the cost of transmitting a single-word message from u to v. We assume that c is symmetric and satisfies the triangle inequality. We also assume for simplicity that for u, v, and w in V, $c(u, v) = c(u, w)$ if and only if $v = w$. (We make the latter assumption for the sake of convenience only, and with essentially no loss in generality, since an arbitrarily small perturbation in the cost function can be used to break ties.)

The cost of transmitting a message of length ℓ from node u to node v is given by $f(\ell)c(u, v)$, where $f : \mathbf{N} \mapsto \mathbf{R}^+$ is any nondecreasing function such that $f(1) = 1$.

Given any u in V and any real r, let $M(u, r)$ denote the set $\{v \in V : c(u, v) \leq r\}$. We refer to $M(u, r)$ as the *ball* of *radius* r around u. We assume that there exist real constants $\delta > 8$ and Δ such that for any node u in V and any real $r \geq 1$, we have

$$\min\{\delta|M(u, r)|, n\} \quad\leq\quad |M(u, 2r)| \quad\leq\quad \Delta|M(u, r)|. \tag{2.1}$$

as illustrated in Figure 2.1.

Local Memory. We partition the local memory of each node u into two parts. The first part, the *main memory*, stores objects. The second part, the *auxiliary memory*, is for storing possible control information.

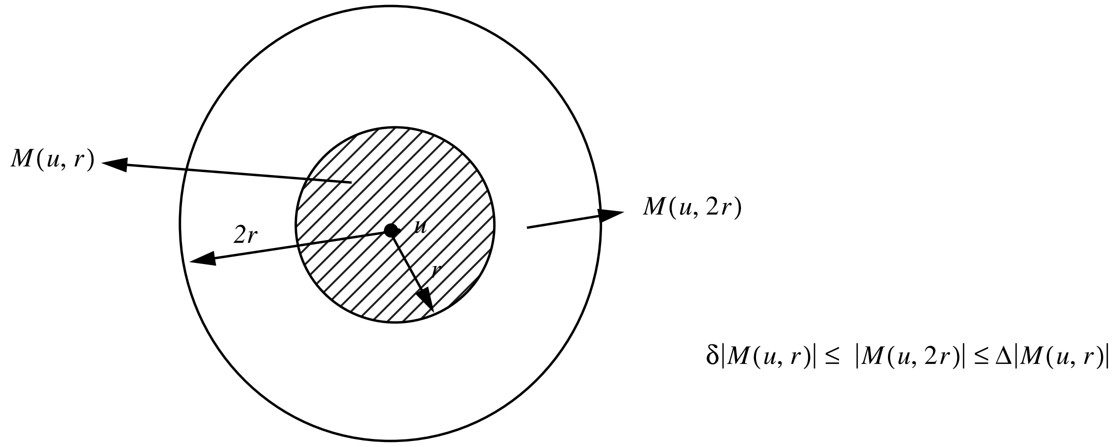


Figure 2.1: Properties of the cost function.

Local Computation. There is no cost associated with local computation. (Although the model allows an arbitrary amount of local computation at zero cost, our algorithm does not perform any particularly complex local operations.)

Complexity measures. We evaluate any solution on the basis of four different complexity measures. The first measure is the cost of reading an object. The second measure is the size of the auxiliary memory at any node. The remaining two measures concern the dynamic nature of the problem: We address the complexity of inserting or deleting a copy of an object, and of adding or removing a network node. The third measure is the cost of inserting or deleting a copy of an object. The fourth measure is *adaptability*, which is defined as the number of nodes whose auxiliary memory is updated upon the addition or removal of a node. (Our notion of adaptability is analogous to that of [11].)

2.3 The access scheme

In this section, we present our access scheme for shared objects. We assume that n is a power of 2^b, where b is a fixed positive integer to be specified later (see the beginning of Section 2.5). For each node x in V, we assign a label independently and uniformly at random from $[n]$. For i in $[\log n]$, let x^i denote the ith bit of the label of x. Note that the label of a node x is independent of the unique $(\log n)$-bit identification of the node. For all x in V (resp., A in $\mathcal{A}$), we define $x[i]$ as the nonnegative integer with binary representation $x^{(i+1)b-1} \cdots x^{ib}$ (resp., $A[i]$ denotes $A^{(i+1)b-1} \cdots A^{ib}$), for i in $[(\log n)/b]$. We also assign a total order to the nodes in V, given by the bijection $\beta : V \rightarrow [n]$.

We partition the auxiliary memory of each node in two parts, namely the *neighbor table* and the *pointer list* of the node.

- **Neighbor table.** For each node x, the neighbor table of x consists of $(\log n)/b$ levels. The ith level of the table, i in $[(\log n)/b]$, consists of *primary*, *secondary*, and *reverse* (i, j)-*neighbors*, for all j in $[2^b]$. The *primary* (i, j)-*neighbor* y of x is such that $y[k] = x[k]$ for all k in $[i]$, and either (i) $i < (\log n)/b - 1$ and y is the node of minimum $c(x, y)$ such that $y[i] = j$, if such a node exists, or (ii) y is the node with largest $\beta(y)$ among all nodes z such that $z[i]$ matches j in the largest number of rightmost bits. Note that the primary (i, j)-neighbor of a node x is guaranteed to exist, since x itself is a candidate node. Let d be a fixed positive integer, to be specified later (see the beginning of Section 2.5). Let y be the primary (i, j)-neighbor of x. If $y[i] = j$, then let $W_{i,j}$ denote the set of nodes w in $V \setminus \{y\}$ such that $w[k] = x[k]$ for k in $[i]$, $w[i] = j$, and $c(x, w)$ is at most $O(c(x, y))$. Otherwise, let $W_{i,j}$ be the empty set. The set of *secondary* (i, j)-*neighbors* of x is the subset U of $\min\{d, |W_{i,j}|\}$ nodes u with minimum $c(x, u)$ in $W_{i,j}$ — i.e., $c(x, u)$ is at most $c(x, w)$, for all w in $W_{i,j}$, and for all u in U, and $|U| \leq d$. A node w is a *reverse* (i, j)-*neighbor* of x if and only if x is a primary (i, j)-neighbor of w.

  In Figure 2.2, we illustrate the primary neighbors entries in the neighbor table of node x for $b = 1$; suppose the level i-neighbors of x in the table are given by (i) above.

- **Pointer list.** Each node x also maintains a pointer list *Ptr*(x) with pointers to copies of some objects in the network. Formally, *Ptr*(x) is a set of triples (A, y, k), where A is in $\mathcal{A}$, y is a node that holds a copy of A, and k is an upper bound on the cost $c(x, y)$. We maintain the invariant that there is at most one triple associated with any object in *Ptr*(x). The pointer list of x may only be updated as a result of insert and delete

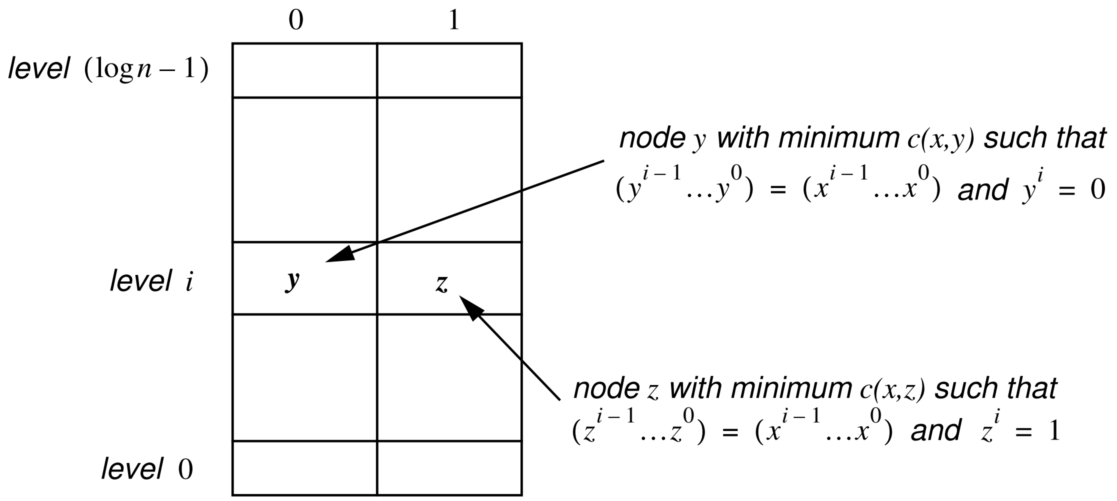


Figure 2.2: The primary neighbor table of node x, for $b = 1$.

operations. All the pointer lists can be initialized by inserting each shared copy in the network at the start of the computation. We do not address the cost of initializing the auxiliary memories of the nodes.

Let r be the node whose label matches (in terms of binary representation) the identification of A in the largest number of rightmost bits. (In case of a tie between several nodes $r_0, \ldots, r_k$, let r be the unique node r_i maximizing $\beta(r_i)$.) We call r the *root* node for object A. The uniqueness of the root node for each A in $\mathcal{A}$ is crucial to guarantee the success of every read operation.

In this section and throughout the paper, we use the notation $\langle \alpha \rangle_k$ to denote the sequence $\alpha_0, \ldots, \alpha_k$ (of length $k+1$). When clear from the context, k will be omitted. In particular, a *primary neighbor sequence* for A is a maximal sequence $\langle u \rangle_k$ such that u_0 is in V, u_k is the root node for A, and u_{i+1} is the primary $(i, A[i])$-neighbor of u_i, for all i. It is worth noting that the sequence $\langle u \rangle_k$ is such that the label of node u_i satisfies $(u_i[i-1], \ldots, u_i[0]) = (A[i-1], \ldots, A[0])$, for all i.

We now give an overview of the read, insert, and delete operations.

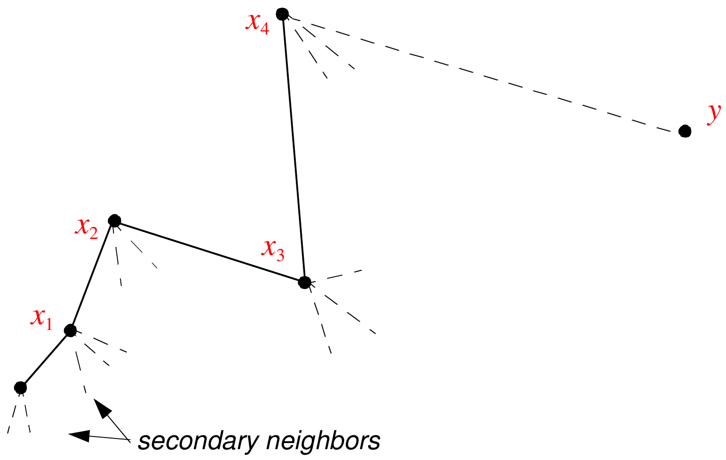


Figure 2.3: A read request for object A is forwarded along the primary neighbor sequence for A with $x = x_0$.

- **Read.** Consider a node x attempting to read an object A. The read operation proceeds by successively forwarding the *read request* for object A originating at node x along the primary neighbor sequence $\langle x \rangle$ for A with $x_0 = x$. When forwarding the read request, node x_{i-1} informs x_i of the current best upper bound k on the cost of sending a copy of A to x. On receiving the read request with associated upper bound k, node x_i proceeds as follows. If x_i is the root node for A, then x_i requests that the copy of A associated with the current best upper bound k be sent to x. Otherwise, x_i communicates with its primary and secondary $(i, A[i])$-neighbors to check whether the pointer list of any of these neighbors has an entry (A, z, k_1) such that k_1 is at most k. Then, x_i updates k to be the minimum of k and the smallest value of k_1 thus obtained (if any). If k is within a constant factor of the cost of following $\langle x \rangle$ up to x_i, that is, k is $O(\sum_{j=0}^{i-1} c(x_j, x_{j+1}))$, then x_i requests that the copy of A associated with the upper bound k be sent to x. Otherwise, x_i forwards the read request to x_{i+1}. Figure 2.3 illustrates an example of a read request for object A generated by node $x = x_0$, which is forwarded along $\langle x \rangle$ until a pointer to a copy of A is found at node y, one of the secondary neighbors of node x_4.

  Relating to the more general description of the read operation of our scheme described in Section 2.1, the tree T associated with object A is given by the following

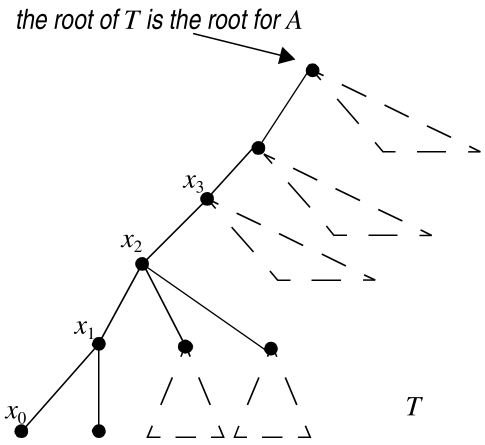


Figure 2.4: The tree T associated with object A and the primary neighbor sequence for $x = x_0$.

rule: The parent of node x in T is the primary $(i, A[i])$-neighbor of x, where i is the maximum index such that $(x[i-1], \ldots x[0]) = (A[i-1], \ldots, A[0])$; or in other words, the parent of node x is the node x_i in the primary neighbor sequence $\langle x \rangle$ for A with $x = x_0$. Figure 2.4 illustrates the tree T and the sequence $\langle x \rangle$ (without loss of generality, suppose all the x_i's in this sequence are distinct).

- **Insert.** An *insert request* for object A generated by node y updates the pointer lists of the nodes in some prefix of the primary neighbor sequence $\langle y \rangle$ for A with $y_0 = y$. When such an update arrives at a node y_i by means of an insert message, y_i updates its pointer list if the upper bound $\sum_{j=0}^{i-1} c(y_j, y_{j+1})$ on the cost of getting object A from y is smaller than the current upper bound associated with A in this list. In other words, y_i updates *Ptr*(y_i) if $(A, \cdot, \cdot)$ is not in this list, or if $(A, \cdot, k)$ is in *Ptr*(y_i) and k is greater than $\sum_{j=0}^{i-1} c(y_j, y_{j+1})$. Node y_i forwards the insert request to node y_{i+1} only if *Ptr*(y_i) is updated.

- **Delete.** A *delete request* for object A generated by node y eventually removes all triples of the form $(A, y, \cdot)$ from the pointer lists *Ptr*(y_i), where $\langle y \rangle$ is the primary neighbor sequence for A with $y_0 = y$, making the copy of A at y unavailable to other nodes in the network. Upon receiving such a request by means of a delete message, node y_i checks whether the entry associated with A in its pointer list is of the form

$(A, y, \cdot)$. If it is not, the delete procedure is completed and we need not proceed further in updating the pointer lists in $\langle y \rangle$. Otherwise, y_i deletes this entry from its pointer list, and checks for entries associated with A in the pointer lists of its reverse $(i-1, A[i-1])$-neighbors. If an entry is found, y_i updates *Ptr*(y_i) by adding the entry $(A, w, k + c(w, y_i))$, where w is the reverse $(i-1, A[i-1])$-neighbor of y_i with minimum upper bound k associated with A in its pointer list. A delete message is then forwarded to y_{i+1}.

The read, insert, and delete operations are summarized in Figures 2.5 and 2.6. The messages and requests in the figure are all with respect to object A. A *read request* is generated by node x when x $(= x_0)$ sends a message *Read*$(x, \infty, \cdot)$ to itself, if x does not hold a copy of A. A read message *Read*(x, k, y) indicates that (i) a read request for object A was generated at node x, (ii) the current best upper bound on the cost of reading a copy of A is k, and (iii) such a copy resides at y. An *insert* (resp., *delete*) *request* is generated when node y $(= y_0)$ sends a message *Insert*$(y, 0)$ (resp., *Delete*(y)) to itself. An insert message *Insert*(y, k) indicates to its recipient node z that the best known upper bound on the cost incurred by bringing the copy of A located at y to the node z is k. We assume that y holds a copy of A and that this copy is unshared (resp., shared) when an insert (resp., delete) request for A is generated at y.

The correctness of our access scheme follows from the two points below:

1. The insert and delete procedures maintain the following invariants. For any A in $\mathcal{A}$ and any y in V, there is at most one entry associated with A in the pointer list of y. If y holds a shared copy of A and $\langle y \rangle$ is the primary neighbor sequence for A with $y_0 = y$, then (i) there is an entry associated with A in the pointer list of every node in $\langle y \rangle$, and (ii) the nodes that have a pointer list entry associated with the copy of A at y form a prefix subsequence of $\langle y \rangle$. The preceding claims follow directly from the insert and delete procedures as described.

2. Every read request for any object A by any node x is successful; that is, it locates and brings to x a shared copy of A, if such a copy is currently available. The read operation proceeds by following the primary neighbor sequence $\langle x \rangle$ for A with $x_0 =$

x, until either a copy of A is located or the root for A is reached. By point 1 above, there exists a shared copy of A in the network if and only if the root for A has a pointer to it.

Action of x_i on receiving a message Read(x, k, y):

If $i > 0$ and $x_i[i-1] \neq A[i-1]$, or $i = (\log n)/b - 1$ (that is, x_i is the root for A) then

- Node x_i sends a message *Satisfy*(x) to node v such that $(A, v, \cdot)$ is in *Ptr*(x_i), requesting it to send a copy of A to x. If *Ptr*(x_i) has no such entry, then there are no shared copies of A.

Otherwise

- Let U be the set of secondary $(i, A[i])$-neighbors of x_i. Node x_i requests a copy of A with associated upper bound at most k from each node in $U \cup \{x_{i+1}\}$.
- Each node u in $U \cup \{x_{i+1}\}$ responds to the request message received from x_i as follows: if there exists an entry (A, v, q_v) in *Ptr*(u) and if $q'_v = q_v + c(x_i, u) + \sum_{j=0}^{i-1} c(x_j, x_{j+1})$ is at most k, then u sends a success message *Success*(v, q'_v) to x_i.
- Let U' be the set of nodes u from which x_i receives a response message *Success*(u, k_u). If U' is not empty, then x_i updates (k, y) to be (k_z, z), where z is a node with minimum k_u over all u in U'.
- If $k = O(\sum_{j=0}^{i-1} c(x_j, x_{j+1}))$ then x_i sends a message *Satisfy*(x) to node y, requesting y to send a copy of A to x. Otherwise, x_i forwards a message *Read*(x, k, y) to x_{i+1}.

Figure 2.5: Action on receiving a message *Read* for object A.

2.4 Performance bounds

In this section, we state our main claims regarding the performance of our access scheme. In Theorems 1 through 4 below, we state bounds on the cost of a read, the cost of an insert or delete, the size of auxiliary memory, and the adaptability of our access scheme.

Theorem 1 *Let x be any node in V and let A be any object in $\mathcal{A}$. If y is the node with minimum $c(x, y)$ that holds a shared copy of A, then the expected cost of satisfying a read request for A by x is $O(f(\ell(A))c(x, y))$.*

Let C denote $\max\{c(u,v){:}u, v \in V\}$. If a node x tries to read an object A for which there is currently no shared copy in the network, then the expected cost of the read operation is $O(C)$.

Theorem 2 *The expected cost of an insert operation is $O(C)$, and that of a delete operation is $O(C \log n)$.*

Theorem 3 *Let q be the number of objects that can be stored in the main memory of each node. The size of the auxiliary memory at each node is $O(q \log^2 n)$ words with high probability.*

Theorem 4 *The adaptability of our scheme is $O(\log n)$ expected and $O(\log^2 n)$ with high probability.*

Action of y_i on receiving a message Insert(y, k):
If $(A, \cdot, \cdot)$ is not in *Ptr*(y_i), or $(A, \cdot, k')$ is in *Ptr*(y_i) and $k' > k$, then

- Node y_i accordingly creates or replaces the entry associated with A in *Ptr*(y_i) by inserting (A, y, k) into this list.
- If $y_i[i-1] = A[i-1]$ then y_i sends a message *Insert*$(y, k + c(y_i, y_{i+1}))$ to y_{i+1}.

Action of y_i on receiving a message Delete(y):
If $(A, y, \cdot)$ is in *Ptr*(y_i) then

- Let U be the set of reverse $(i-1, A[i-1])$-neighbors of y_i. Node y_i removes $(A, y, \cdot)$ from *Ptr*(y_i), and requests a copy of A from each u in U.
- Each u in U responds to the request message from y_i by sending a message *Success*$(v, q_v + c(y_i, u))$ to y_i if and only if (A, v, q_v) is in *Ptr*(u).
- Let U' be the set of nodes u such that y_i receives a message *Success*(u, k_u) in response to the request message it sent. If $|U'| > 0$ then y_i inserts (A, w, k_w) into *Ptr*(y_i), where w is the node in U' such that $k_w \leq k_u$, for all u in U'.
- If $y_i[i-1] = A[i-1]$ then y_i sends a message *Delete*(y) to y_{i+1}.

Figure 2.6: Actions on receiving messages *Insert* and *Delete* for object A.

2.5 Analysis

In this section, we analyze the access scheme described in Section 2.3, and establish the main results described in Section 2.4. Section 2.5.1 presents some useful properties of balls. Section 2.5.2 presents properties of primary and secondary neighbors. Section 2.5.3 presents the proofs of Theorems 1 and 2. Sections 2.5.4 and 2.5.5 present the proofs of Theorems 3 and 4, respectively.

Several constants appear in the model, the algorithms, and the analysis: δ and Δ appear in the model, b and d appear in the algorithms, γ and ε appear in the analysis. We choose b, d, γ, and ε such that the following inequalities hold.

$$2^b \geq \Delta^2\gamma^3 \tag{2.2}$$

$$d \geq \gamma^2 \tag{2.3}$$

$$\gamma \geq \Delta^2 \tag{2.4}$$

$$\varepsilon \geq \max\{6\Delta/\gamma, 4e^{-\gamma/4\Delta}, 6(d+1)/2^b, 6(2e/2^b)^{d/2}, 6(e\Delta\gamma^2/d)^{d/2}\} \tag{2.5}$$

$$\varepsilon < (10 \cdot 2^{\lceil b \log_\delta 2 \rceil})^{-1} \tag{2.6}$$

An assignment of values to the constants γ, b, d, and ε that satisfies the above inequalities may be obtained as follows: Set γ equal to $2^{b/3}/\Delta^{2/3}$, d equal to $e2^{2b/3+1}/\Delta^{1/3}$, and ε equal to $6e\Delta^{5/3}/2^{b/3}$. The preceding assignment satisfies Equations 2.5 if b is set sufficiently large. Equations 2.4 and 2.6 can be satisfied by setting b large enough so that $2^b \geq \Delta^8$ and $2^{b/3-\lceil b \log_\delta 2 \rceil} > 60e\Delta^{5/3}$.

2.5.1 Properties of balls

In this section, we prove several properties of the "local neighborhoods" of the nodes in V with respect to the cost function c. We view these "neighborhoods" as *balls* centered at the nodes of V. In Section 2.2, we defined the ball of radius r around a node u, $M(u,r)$. Now we define the *ball of size* k around node u, $N(u,k)$, for any u in V and any integer k in $[1,n]$: Let $N(u,k)$ denote the unique set of k nodes such that for any v in $N(u,k)$ and w not in $N(u,k)$, $c(u,v)$ is less than $c(u,w)$. For convenience, if k is greater than n,

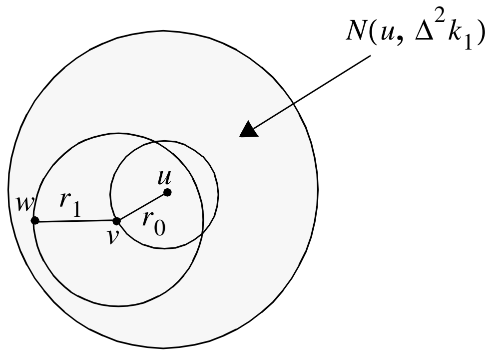


Figure 2.7: Illustrating the proof of Lemma 2.5.1.

we let $N(u, k)$ be V. As for the balls $M(u, r)$, we define the *radius* of $N(u, k)$ to be the maximum value of $c(u, v)$ over all v in $N(u, k)$.

In the proofs of the lemmas in this section, we extensively use Equation 2.1 as well as the fact that the cost metric is symmetric and satisfies the triangle inequality.

Lemma 2.5.1 *Let u, v, and w be in V and let k_0 and k_1 be positive integers. If v is in $N(u, k_0)$ and w is in $N(v, k_1)$, then w is in $N(u, \Delta k_0 + \Delta^2 k_1)$.*

Proof: Let r_0 and r_1 denote $c(u, v)$ and $c(v, w)$, respectively. The node w is contained in the ball $M(u, c(u, w))$. If $r_0 \geq r_1$, then $|M(u, c(u, w))|$ is at most $|M(u, r_0 + r_1)|$, which is at most Δk_0 by Equation 2.1. Otherwise, $|M(u, r_0 + r_1)|$ is at most $|M(v, 3r_1)|$, which is at most $\Delta^2 k_1$ by Equation 2.1. Therefore, w belongs to $N(u, \Delta k_0 + \Delta^2 k_1)$. Figure 2.7 illustrates the case $r_0 < r_1$. Q.E.D.

We now consider the smallest (resp., the largest) ball centered at a node v that contains (resp., is contained in) some given subset of nodes. Given any subset S of V and some node u in S, let $n_{\subseteq}(u, S)$ (resp., $n_{\supseteq}(u, S)$) denote the largest (resp., smallest) integer k such that $N(u, k)$ is a subset (resp., superset) of S. Let $N_{\subseteq}(u, S)$ and $N_{\supseteq}(u, S)$ denote $N(u, n_{\subseteq}(u, S))$ and $N(u, n_{\supseteq}(u, S))$, respectively.

Lemma 2.5.2 *Let u be in V, let S be a subset of V, and let k be in $[1, n]$. Then $N(u, k)$ is a subset (resp., superset) of S if and only if $N(u, k)$ is a subset of $N_{\subseteq}(u, S)$ (resp., superset of $N_{\supseteq}(u, S)$).*

Proof: If $N(u,k)$ is a subset of S then $n_{\subseteq}(u,S)$ is at least k; hence, $N(u,k)$ is a subset of $N_{\subseteq}(u,S)$. If $N(u,k)$ is a subset of $N_{\subseteq}(u,S)$ then $N(u,k)$ is a subset of S because $N_{\subseteq}(u,S)$ is a subset of S. If $N(u,k)$ is a superset of S then $n_{\supseteq}(u,S)$ is at most k; hence, $N(u,k)$ is a superset of $N_{\supseteq}(u,S)$. If $N(u,k)$ is a superset of $N_{\supseteq}(u,S)$ then $N(u,k)$ is a superset of S because $N_{\supseteq}(u,S)$ is a superset of S. Q.E.D.

Lemma 2.5.3 *Let u belong to V, and let k_0 and k_1 denote positive integers such that $k_1 \geq \Delta^2 k_0$. For any v in $N(u,k_0)$, $n_{\subseteq}(v,N(u,k_1))$ is at least k_1/Δ and $N_{\supseteq}(v,N(u,k_1))$ is a subset of $N(u,\Delta k_1)$.*

Proof: We first obtain a lower bound on $n_{\subseteq}(v,N(u,k_1))$. Let r_0 and r_1 denote the radii of $N(u,k_0)$ and $N(u,k_1)$, respectively. Since $k_1 \geq \Delta^2 k_0$, Equation 2.1 implies that $r_1 - r_0 \geq (r_1+r_0)/2$. Let w be the node in $N_{\subseteq}(v,N(u,k_1))$ such that $c(v,w)$ is maximum. A ball of radius $(r_1 - r_0)$ around v is contained in $N(u,k_1)$ (since v is contained in $N(u,k_0)$). Thus $r_1 - r_0 \leq c(v,w)$. It follows that $2c(v,w)$ is at least $2(r_1-r_0) \geq r_1+r_0$ and $M(v,2c(v,w))$ is a superset of $N(u,k_1)$. We now obtain a lower bound on $n_{\subseteq}(v,N(u,k_1))$ as follows:

$$\begin{aligned}
n_{\subseteq}(v,N(u,k_1)) &= |M(v,c(v,w))| \\
&\geq |M(v,2c(v,w))|/\Delta \\
&\geq k_1/\Delta.
\end{aligned}$$

We now place an upper bound on $n_{\supseteq}(v,N(u,k_1))$. Let w be the node in $N_{\supseteq}(v,N(u,k_1))$ such that $c(v,w)$ is maximum. We have $r_1 - r_0 \leq c(v,w) \leq r_1 + r_0$. (We showed that $r_1 - r_0 \leq c(v,w)$ in the preceding paragraph, and the other inequality follows from the triangle inequality.) It follows that $2(r_1 - r_0)$ is at least $c(v,w)$ and $M(v,c(v,w))$ is a subset of $M(u,2r_1)$. Therefore,

$$\begin{aligned}
n_{\supseteq}(v,N(u,k_1)) &= |M(v,c(v,w))| \\
&\leq |M(u,2r_1)| \\
&\leq \Delta k_1.
\end{aligned}$$

Q.E.D.

We use Lemmas 2.5.1 to 2.5.3 to prove Lemma 2.5.4 below. Lemma 2.5.4 and Corollary 2.5.4.1 are used in Section 2.5.3. We refer to any predicate on V that depends only on the label of v as a *label predicate*. Given any node u in V and a label predicate $\mathcal{P}$ on V, let $p(u, \mathcal{P})$ denote the node v such that (i) $\mathcal{P}(v)$ holds, and (ii) for any node w such that $\mathcal{P}(w)$ holds, $c(u, v)$ is at most $c(u, w)$. (We let $p(u, \mathcal{P})$ be null if such a v is not defined.) Let $P(u, \mathcal{P})$ be $M(u, c(u, p(u, \mathcal{P})))$, if $p(u, \mathcal{P})$ is not null, and V otherwise.

Lemma 2.5.4 examines the effect of the relationship between the set $P(u, \mathcal{P})$ and the probability distribution of the labels of the nodes, for any given node u and label predicate $\mathcal{P}$. For u in V, and i in $[(\log n)/b]$, let $\lambda_{\geq i}(u)$ denote the string of $(\log n - ib)$ bits given by $u[(\log n)/b - 1] \cdots u[i+1]u[i]$. For convenience, we let $\lambda_{>i}(u)$ denote $\lambda_{\geq i+1}(u)$. For all i and all u in V, let $\mathcal{P}_i(u)$ hold if and only if $u[i] = A[i]$. For all i and all u in V, let $\mathcal{P}_{<i}(u)$ denote $\wedge_{j \in [i]} \mathcal{P}_j(u)$. Let $\mathcal{P}_{\leq i}(u)$, $\mathcal{P}_{>i}(u)$, and $\mathcal{P}_{\geq i}(u)$ be defined similarly. We note that for u and v in V, and nonnegative integers i and j, if $(u \neq v) \vee ((u = v) \wedge (i \neq j))$, then $\mathcal{P}_i(u)$ and $\mathcal{P}_j(v)$ are independent random variables. Also, each of the predicates defined above is a label predicate.

Lemma 2.5.4 *Let S and S' be subsets of V, and let u belong to S. Let $\mathcal{P}$ be a label predicate on V and for each v in S', let $\lambda_{\geq 0}(v)$ be chosen independently and uniformly at random.*

1. *Given that $P(u, \mathcal{P}) \subseteq S$, we have that (i) the variables $\lambda_{\geq 0}(v)$, for all v in $S' \setminus P(u, \mathcal{P})$, are independent and uniformly random, and (ii) for each node v in $P(u, \mathcal{P}) \setminus \{p(u, \mathcal{P})\}$, $\mathcal{P}(v)$ is false.*

2. *Given that $P(u, \mathcal{P}) \not\subseteq S$, we have that (i) the variables $\lambda_{\geq 0}(v)$ for all v in $S' \setminus N_{\subseteq}(u, S)$ are independent and uniformly random, and (ii) for each node v in $N_{\subseteq}(u, S)$, $\mathcal{P}(v)$ is false.*

3. *Given that $P(u, \mathcal{P}) \supseteq S$, we have that (i) the variables $\lambda_{\geq 0}(v)$ for all v in $S' \setminus N_{\supseteq}(u, S)$ are independent and uniformly random, and (ii) for each node v in $N_{\supseteq}(u, S) \setminus \{p(u, \mathcal{P})\}$, $\mathcal{P}(v)$ is false.*

Proof: We first consider Part 1 of the lemma. Part 1(i) follows from the independence of $\mathcal{P}(v)$ and $\mathcal{P}(w)$, for any two distinct nodes v and w. By the definition of P, $\mathcal{P}(p(u,\mathcal{P}))$ holds and for each node v in $P(u,\mathcal{P})$, $\mathcal{P}(v)$ is false. This proves Part 1(ii). Parts 2 and 3 follow similarly. For Part 2, we note that the event $P(u,\mathcal{P}) \not\subseteq S$ is equivalent to the event that for each node v in $N_{\subseteq}(u,S)$, $\mathcal{P}(v)$ is false. For Part 3, we note that the event $P(u,\mathcal{P}) \supseteq S$ is equivalent to the event that for each node in $N_{\supseteq}(u,S) \setminus \{n_{\supseteq}(u,S)\}$, $\mathcal{P}(v)$ is false. Q.E.D.

The following claim follows from repeated application of Part 1 of Lemma 2.5.4.

Corollary 2.5.4.1 *Let S be an arbitrary subset of V, let i be in $[(\log n)/b - 1]$, and let S' be a subset of V such that $\lambda_{\geq 0}(u)$ is independently and uniformly random for each u in S'. Given a sequence of nodes $u_0, \ldots, u_i$ such that for all j in $[i]$, $u_{j+1} = p(u_j, \mathcal{P}_{\leq j})$ and $P(u_j, \mathcal{P}_{\leq j}) \subseteq S$, we have*

1. *The variables $\lambda_{\geq 0}(u)$ for all u in $S' \setminus \cup_{j \in [i]} P(u, \mathcal{P}_{\leq j})$ are independent and uniformly random.*
2. *The variable $\lambda_{>i}(u_i)$ is uniformly random and for each node u in $\cup_{j\in[i]} P(u_j, \mathcal{P}_{\leq j}) \setminus \{u_i\}$, $\mathcal{P}_{\leq i}(u)$ is false.* *Q.E.D.*

2.5.2 Properties of neighbors

In this section, we establish certain claims concerning the different types of neighbors that are defined in Section 2.3. We differentiate between root and nonroot primary (i,j)-neighbors. A *root primary* (i,j)-neighbor w of v is a primary (i,j)-neighbor w of v such that $w[i] \neq j$ or $i = (\log n)/b - 1$. A primary neighbor that is not a root primary neighbor is a *nonroot primary* neighbor. Note that, for $i < (\log n)/b - 1$, if u is a root primary (i,j)-neighbor of v, then $u[\ell]$ equals $v[\ell]$, for each ℓ in $[i]$, and there is no node w in V such that $w[i]$ equals j and $w[\ell]$ equals $v[\ell]$, for each ℓ in $[i]$.

Lemma 2.5.5 *Let u and v be in V, and let k denote $|M(u, c(u,v))|$. For any j in $[2^b]$, we have that (i) for any i in $[(\log n)/b - 1]$, the probability that u is a primary (i,j)-neighbor* *of v is at most $e^{-((k/\Delta)-2)/2^{(i+1)b}}$, and (ii) for any i in $[(\log n)/b]$, the probability that u is a root primary (i,j)-neighbor of v is at most $e^{-n/2^{(i+1)b}}$.*

Proof: Consider the ball $M(v, c(v,u))$. By Equation 2.1, $|M(v, c(v,u))| \geq |M(v, 2c(v,u))|/\Delta$. Since $M(v, 2c(v,u))$ is a superset of $M(u, c(u,v))$, we have $|M(v, c(v,u))| \geq k/\Delta$. The probability that a node w in $M(v, c(u,v)) \setminus \{u, v\}$ does not match the label of v in its $(i+1)b$ rightmost bits is at most $1 - 1/2^{(i+1)b}$. Since i is less than $(\log n)/b - 1$, the probability that u is a primary (i,j)-neighbor of v is at most

$$\begin{aligned} & (1 - 1/2^{(i+1)b})^{(k/\Delta)-2} \\ \leq\; & e^{-((k/\Delta)-2)/2^{(i+1)b}}. \end{aligned}$$

If u is a root primary (i,j)-neighbor of v, then $u[\ell]$ equals $v[\ell]$ for each ℓ in $[i]$ and there is no node w in V such that $w[i]$ equals j and $w[\ell]$ equals $v[\ell]$ for each ℓ in $[i]$. Therefore, the probability that u is a root primary (i,j)-neighbor of v is at most

$$\begin{aligned} & (1/2^{ib})(1 - 1/2^{(i+1)b})^{n-1}(1 - 1/2^{b}) \\ \leq\; & (1/2^{ib})(1 - 1/2^{(i+1)b})^{n} \\ \leq\; & (1/2^{ib})e^{-n/2^{(i+1)b}}. \end{aligned}$$

Q.E.D.

Corollary 2.5.5.1 *Let u and v be in V, let i be in $[(\log n)/b]$, and let j be in $[2^b]$. If u is a primary (i,j)-neighbor of v, then v is in $N(u, O(2^{ib} \log n))$ with high probability.*

Q.E.D.

Lemma 2.5.6 and Corollary 2.5.6.1 below establish bounds on the number of nodes v such that u is a primary or secondary neighbor of v, and on the number of nodes v such that v is a reverse neighbor of u, respectively. For any u in V, let a_u denote the total number of triples (i,j,v) such that i belongs to $[(\log n)/b]$, j belongs to $[2^b]$, v belongs to V, and u is a primary or secondary (i,j)-neighbor of v. Lemma 2.5.6 is used in the proof of Theorem 4, while Corollary 2.5.6.1 is used in the proofs of Theorems 2 and 3.

Lemma 2.5.6 *Let u be in V and let i be in $[(\log n)/b]$. Then the number of nodes for which u is an ith level primary neighbor is $O(\log n)$ with high probability. Also, $E[a_u] = O(\log n)$ and a_u is $O(\log^2 n)$ with high probability.*

Proof: Given a node v in V, i in $[(\log n)/b-1]$, and j in $[2^b]$, it follows from Lemma 2.5.5 that the probability that u is a root primary (i,j)-neighbor of v is at most $(1/2^{ib})e^{-n/2^{(i+1)b}}$. Given a node v in V and j in $[2^b]$, the probability that u is a root $((\log n)/b, j)$-primary neighbor of v is at most $1/n$.

Fix j in $[2^b]$. Let ℓ equal $(\log n - \log\log n)/b - \Omega(1)$, where the constant in the $\Omega(1)$ term is chosen sufficiently large. We consider two cases: $i < \ell$ and $i \geq \ell$. If $i < \ell$, then the probability that there exists v in V such that u is a root primary (i,j)-neighbor of v is at most

$$\begin{aligned} & n(1/2^{ib})e^{-n/2^{(i+1)b}} \\ \leq\; & ne^{-\Omega(\log n)} \\ =\; & O(1/\text{poly}(n)). \end{aligned}$$

If $i \geq \ell$, then given v in V, the probability that u is a root primary (i,j)-neighbor of v is at most $1/2^{\ell b} = O((\log n)/n)$. It follows from Chernoff bounds [10] that the number of nodes for which u is a root primary (i,j)-neighbor is $O(\log n)$ with high probability.

We now consider secondary and nonroot primary neighbors. For any i in $[(\log n)/b]$, u is a secondary or nonroot primary (i,j)-neighbor of v only if j is $u[i]$ and u is one of the $d+1$ nodes w in V with minimum $c(v,w)$ whose lowest ib bits match those of v. We now fix u and i, set j to $u[i]$, and obtain an upper bound on the probability that u is one of the at most $d+1$ nodes w with minimum $c(v,w)$ and whose first ib bits match those of v.

Consider a node v in $N(u, \mu^{k+1}2^{(i+1)b}) \setminus N(u, \mu^k 2^{(i+1)b})$, where μ is a real constant to be specified later. If k equals zero, then the probability that u is a primary or secondary (i,j)-neighbor of v is at most $1/2^{ib}$. Otherwise, consider the ball $M(v, c(v,u))$. By the low-expansion condition (i.e., the right inequality of Equation 2.1), $|M(v, c(v,u))|$ is at least $|M(v, 2c(v,u))|/\Delta$. We are given that $M(u, c(u,v))$ is a superset of $N(u, \mu^k 2^{(i+1)b})$. Since $M(v, 2c(v,u))$ is a superset of $M(u, c(u,v))$, we obtain that $|M(v, c(v,u))|$ is at least $\mu^k 2^{(i+1)b}/\Delta$. The probability that u is a primary or secondary (i,j)-neighbor of v is at most

$$\begin{aligned}
& d\binom{\mu^k 2^{(i+1)b}/\Delta}{d}(1-1/2^{(i+1)b})^{(\mu^k 2^{(i+1)b}/\Delta)-d}/(2^{ib}2^{(i+1)bd}) \\
& \leq d(e\mu^k 2^{(i+1)b}/(\Delta d))^d e^{-\mu^k/\Delta}(1-1/2^{(i+1)b})^{-d}/(2^{ib}2^{(i+1)bd}) \\
& \leq 4d(e\mu^k/(\Delta d))^d(e^{-\mu^k/\Delta}/2^{ib}) \\
& \leq 1/((2\mu)^k 2^{ib}).
\end{aligned}$$

(The second step holds since $d \leq 2^b \leq 2^{ib}$ and $(1-1/2^{ib})^{-2^{ib}}$ is at most 4. The third step follows by choosing μ large enough with respect to Δ and d such that $e^{\mu^k/\Delta} \geq (2^k\Delta^k/d^{d-1})(\mu^k/\Delta)^{d+1}$ for all $k \geq 1$.)

Thus, the expected number of nodes for which u is a secondary or nonroot primary neighbor is at most

$$\begin{aligned}
& \sum_{i\in[(\log n)/b],j=u[i]} \sum_{k\geq 0} \sum_{v\in N(u,\mu^{k+1}2^{(i+1)b})\setminus N(u,\mu^k 2^{(i+1)b})} 1/((2\mu)^k 2^{ib}) \\
\leq & \sum_{i\in[(\log n)/b],j=u[i]} 2^b\mu \\
= & \; O(\log n).
\end{aligned}$$

To obtain a high probability bound on the number of nodes for which u is a secondary or nonroot primary neighbor, we proceed as follows. For any v not in $N(u,\Theta(2^{(i+1)b}\log n))$, it follows from Lemma 2.5.5 that the probability that u is a secondary or nonroot primary (i,j)-neighbor of v is $O(1/\text{poly}(n))$. For any v in $N(u,\Theta(2^{(i+1)b}\log n))$, the probability that u is a secondary or nonroot primary (i,j)-neighbor of v is at most $1/2^{(i+1)b}$. Therefore, the number of nodes for which u is a secondary or nonroot primary neighbor is $O(\log^2 n)$ with high probability.

The bounds on expectation and the high probability bounds together establish that $E[a_u]$ is $O(\log n)$ and a_u is $O(\log^2 n)$ with high probability. Q.E.D.

Corollary 2.5.6.1 *For any u in V, the total number of reverse neighbors of u is $O(\log^2 n)$ with high probability, and $O(\log n)$ expected.*

Proof: The desired claim follows directly from Lemma 2.5.6, since v is a reverse (i, j)-neighbor of u only if u is a primary (i, j)-neighbor of v. Q.E.D.

Lemma 2.5.7 is used in the proof of Theorem 3. For a given a node u, it provides a bound on the number of primary neighbor sequences that have u in the ith position. For any u and v in V and i in $[(\log n)/b]$, v is said to be an i*-leaf* of u if there exists a sequence $v = v_0, v_1, \ldots, v_{i-1}, v_i = u$, such that for all j in $[i]$, v_{j+1} is a primary $(j, v_{j+1}[j])$-neighbor of v_j.

Lemma 2.5.7 *Let u belong to V, and let i be in $[(\log n)/b]$. Then the number of i-leaves of u is $O(2^{ib} \log n)$ with high probability.*

Proof: We establish the lemma by showing that if v is an i-leaf of u, then v is in $N(u, c_0 2^{ib} \log n)$ with high probability, where c_0 is a real constant to be specified shortly (see the next paragraph). By Corollary 2.5.5.1, we have that for all j in $[i]$, v_j is in $N(v_{j+1}, c_1 2^{(j+1)b} \log n)$ with high probability for some sufficiently large real constant c_1. We will prove by induction on j in $[i + 1]$ that $v = v_0$ is in $N(v_j, c_0 2^{jb} \log n)$ with high probability.

The induction base follows trivially. For the induction step, assume that v is in $N(v_j, c_0 2^{jb} \log n)$. By Corollary 2.5.5.1, v_j belongs to $N(v_{j+1}, c_1 2^{jb} \log n)$ with high probability. Applying Lemma 2.5.1 with the substitution (v_{j+1}, v_j, v) for (u, v, w), we obtain that v is in $N(v_{j+1}, (\Delta c_1 + \Delta^2 c_0) 2^{jb} \log n)$, with high probability. Since $\Delta^2 < 2^b$ (by Equation 2.2), we can choose c_0 large enough so that $c_0(2^b - \Delta^2)$ is at least Δc_1. It thus follows that v is in $N(v_{j+1}, c_0 2^{(j+1)b})$.

Applying the above inductive claim with $j = i$, we obtain that v is in $N(u, O(2^{ib} \log n))$ with high probability, completing the proof. Q.E.D.

2.5.3 Cost of Operations

In this section, we place upper bounds on the cost of the read, insert, and delete operations by establishing Theorems 1 and 2. We first introduce some notation and prove a few elementary lemmas in Section 2.5.3.1. The bulk of the analysis is in Sections 2.5.3.2

and 2.5.3.3. Using the tools developed in these two sections, we finally prove Theorems 1 and 2 in Section 2.5.3.4. Before beginning the analysis, we remark that most of the notation and tools developed in Sections 2.5.3.1, 2.5.3.2, and 2.5.3.3 are only used in the analysis of the read operation.

2.5.3.1 Preliminaries

Consider a read request originating at node x for an object A. Let y denote a node that has a copy of A. In the following, we show that the expected cost of a read operation is $O(f(\ell(A))c(x,y))$. Letting y denote the node with minimum $c(x,y)$ among the set of nodes that have a copy of A, this bound implies that the expected cost is asymptotically optimal.

Let $\langle x \rangle$ and $\langle y \rangle$ be the primary neighbor sequences for A with $x_0 = x$ and $y_0 = y$, respectively. For any nonnegative integer i, let A_i (resp., D_i) denote the ball of smallest radius around x_i (resp., y_i) that contains x_{i+1} (resp., y_{i+1}). Let B_i (resp., E_i) denote the set $\cup_{0 \le j \le i} A_j$ (resp., $\cup_{0 \le j \le i} D_j$). Let C_i denote the ball of smallest radius around x_i that contains all of the secondary $(i, A[i])$-neighbors of x_i. For convenience, we define $B_{-1} = E_{-1} = \emptyset$.

It is useful to consider an alternative view of x_i, y_i, A_i, and D_i. For any nonnegative i, if x_{i+1} (resp., y_{i+1}) is not the root node for A, then x_{i+1} (resp., y_{i+1}) is $p(x_i, \mathcal{P}_{\le i})$ (resp., $p(y_i, \mathcal{P}_{\le i})$) and A_i (resp., D_i) is $P(x_i, \mathcal{P}_{\le i})$ (resp., $P(y_i, \mathcal{P}_{\le i})$).

Let γ be an integer constant satisfying Equations 2.2 through 2.6. For any nonnegative integer i and any integer j, let X_i^j (resp., Y_i^j) denote the ball $N(x, \gamma^j 2^{(i+1)b})$ (resp., $N(y, \gamma^j 2^{(i+1)b})$). Let i^* denote the least integer such that the radius of $X_{i^*}^1$ is at least $c(x,y)$. Let a_i (resp., b_i) denote the radius of X_i^1 (resp., Y_i^1).

Lemma 2.5.8 *For all i such that $i \ge i^*$, X_i^2 is a superset of Y_i^1.*

Proof: By the definition of i^*, a_i is at least $c(x,y)$. Therefore, $M(y, 2a_i)$ is a superset of X_i^1. Hence, $M(y, 2a_i)$ contains at least $\gamma 2^{(i+1)b}$ nodes and is a superset of Y_i^1.

By Equation 2.1, $|M(x, 3a_i)|$ is at most $\Delta^2|M(x, a_i)| \leq \Delta^2|X_i^1|$. By Equation 2.4, $\Delta^2|X_i^1| \leq \Delta^2\gamma 2^{(i+1)b} \leq \gamma^2 2^{(i+1)b}$. Thus, $M(x, 3a_i)$ is a subset of X_i^2. Since $M(x, 3a_i)$ is a superset of $M(y, 2a_i)$, which is a superset of Y_i^1, the claim holds. Q.E.D.

Lemma 2.5.9 *For all i in $[(\log n)/b - 2]$ we have $2^{\lfloor b \log_\Delta 2 \rfloor} a_i \leq a_{i+1} \leq 2^{\lceil b \log_\delta 2 \rceil} a_i$ and $2^{\lfloor b \log_\Delta 2 \rfloor} b_i \leq b_{i+1} \leq 2^{\lceil b \log_\delta 2 \rceil} b_i$. For $i = (\log n)/b - 2$, we have $a_{i+1} \leq 2^{\lceil b \log_\delta 2 \rceil} a_i$ and $b_{i+1} \leq 2^{\lceil b \log_\delta 2 \rceil} b_i$. Also, a_{i^*} and b_{i^*} are both $O(c(x, y))$.*

Proof: We prove the bounds for a_{i+1} and a_{i^*}. The bounds for b_{i+1} and b_{i^*} follow the same lines. Since $\gamma \leq 2^b$ by Equation 2.4, for all i in $[(\log n)/b - 2]$, we have $|X_{i+1}^1| = 2^b|X_i^1|$. Therefore, for all i in $[(\log n)/b - 2]$, it follows from Equation 2.1 that $2^{\lfloor b \log_\Delta 2 \rfloor} a_i \leq a_{i+1} \leq 2^{\lceil b \log_\delta 2 \rceil} a_i$. For $i = (\log n)/b - 2$, $|X_{i+1}^1| \leq 2^b|X_i^1|$, and hence, $a_{i+1} \leq 2^{\lceil b \log_\delta 2 \rceil} a_i$.

If $i^* > 0$, then a_{i^*} (resp., b_{i^*}) is at most $2^{\lceil b \log_\delta 2 \rceil} c(x, y) = O(c(x, y))$. Otherwise, a_{i^*} is $O(2^{\lceil \log_\delta \gamma \rceil}) = O(c(x, y))$, since δ and γ are constants. Q.E.D.

We define two sequences $\langle s_i \rangle$ and $\langle t_i \rangle$ of nonnegative integers as follows:

$$s_i = \begin{cases} 0 & \text{if } B_i \subseteq X_i^1,\ A_i \supseteq X_i^{-1},\ C_i \supseteq X_i^2, \\ 1 & \text{if } B_i \subseteq X_i^1,\ A_i \supseteq X_i^{-1},\ C_i \not\supseteq X_i^2, \\ 2 & \text{if } B_i \subseteq X_i^1,\ A_i \not\supseteq X_i^{-1}, \text{ and} \\ 3+j & \text{if } 0 \leq j \leq i,\ B_{i-j} \not\subseteq X_i^1,\ B_{i-j-1} \subseteq X_i^1. \end{cases}$$

$$t_i = \begin{cases} 0 & \text{if } E_i \subseteq Y_i^1, \text{ and} \\ 1+j & \text{if } 0 \leq j \leq i,\ E_{i-j} \not\subseteq Y_i^1,\ E_{i-j-1} \subseteq Y_i^1. \end{cases}$$

The following intuition underlies the above definitions of s_i and t_i. For any j, the expected sizes of the balls A_i and D_i are both $2^{(i+1)b}$. Thus, the expected sizes of the balls B_i and E_i are both at most $2^{(i+1)b+1}$. Moreover, the expected size of C_i is at least $c_1 2^{(i+1)b}$, where c_1 is a constant that depends on d. The constant γ is chosen sufficiently large and d is chosen sufficiently larger than γ such that the "expected behavior" of the balls A_i, B_i, C_i, and E_i is as follows: $B_i \subseteq X_i^1$, $A_i \supseteq X_i^{-1}$, $C_i \supseteq X_i^2$, and $E_i \subseteq Y_i^1$. The value of s_i (resp., t_i) indicates the degree to which the sizes of the balls A_i, B_i, and C_i (resp., D_i and E_i) deviate from this expected behavior. The larger the value of s_i, the greater the deviation from the expected behavior.

Lemma 2.5.10 *If s_i is in $\{0,1,2\}$, then $c(x_i, x_{i+1})$ is $O(a_i)$. If t_i is 0, then $c(y_i, y_{i+1})$ is $O(b_i)$.*

Proof: The proof of the first claim follows from the observation that if s_i is in $\{0,1,2\}$ then $A_i \subseteq B_i \subseteq X_i^1$. The proof of the second claim follows from the observation that if t_i is 0 then $D_i \subseteq E_i \subseteq Y_i^1$. Q.E.D.

2.5.3.2 Properties of $\langle s_i \rangle$ and $\langle t_i \rangle$

Our plan for determining an upper bound on the cost of the given read operation for object A is as follows. Let τ be the smallest integer $i \geq i^*$ such that $(s_i, t_i) = (0,0)$. By the definitions of s_τ and t_τ, $C_\tau \supseteq X_\tau^2$ and $Y_\tau^1 \supseteq E_\tau \supseteq D_\tau$. By Lemma 2.5.8, $X_\tau^2 \supseteq Y_\tau^1$, thus implying that C_τ is a superset of D_τ. Thus, a copy of A is located within τ forwarding steps along $\langle x \rangle$. By the definition of the primary and secondary neighbors, the cost of any request (resp., forward) message sent by node x_i is at most $O(c(x_i, x_{i+1}))$ (resp., $c(x_i, x_{i+1})$). Since a copy of A is located within τ forwarding steps, the cost of all messages needed in locating the particular copy of A that is read is at most $O(\sum_{0 \leq j < \tau}(dc(x_j, x_{j+1}) + c(y_j, y_{j+1})))$. Figure 2.8 illustrates a read request for object A generated by node $x = x_0$, which is forwarded along $\langle x \rangle$ until a pointer to the copy of A residing at node y is found at node y_3 (a secondary neighbor of x_4). The cost of reading the copy is at most $f(\ell(A))$ times the preceding cost. Since d is a constant, the cost of reading A is at most

$$\sum_{0 \leq j < \tau} O(f(\ell(A))(c(x_j, x_{j+1}) + c(y_j, y_{j+1}))). \tag{2.7}$$

The remainder of the proof concerns the task of showing that $E[\sum_{0 \leq j < \tau}(c(x_j, x_{j+1}) + c(y_j, y_{j+1}))]$ is $O(c(x,y))$. A key idea is to establish that the sequence $\langle s_i, t_i \rangle$ corresponds to a two-dimensional random walk that is biased towards $(0,0)$. Lemmas 2.5.11 and 2.5.12 below provide a first step towards formalizing this notion.

Lemma 2.5.11 *Let i be in $[(\log n)/b - 1]$. Given s_j and t_j for all j in $[i]$ such that s_{i-1} is at least 3, the probability that s_i is less than s_{i-1} is at least $1 - \varepsilon^2$. Given s_j and t_j for all j in $[i]$ such that t_{i-1} is at least 1, the probability that t_i is less than t_{i-1} is at least $1 - \varepsilon^2$.*

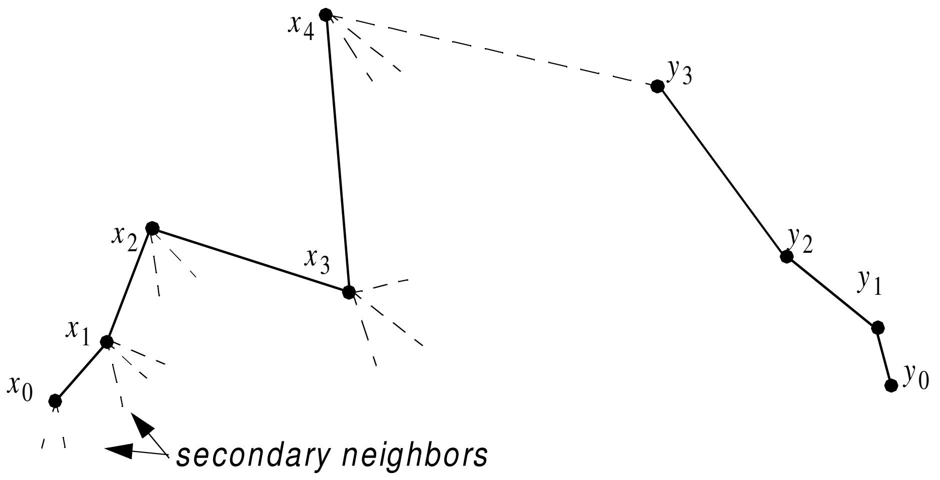


Figure 2.8: A read request for object A is forwarded until a pointer to a copy of A is found.

Lemma 2.5.12 *Let i be in $[(\log n)/b - 1]$. Given s_j and t_j for all j in $[i]$ such that s_{i-1} is at most 3, the probability that s_i is 0 is at least $1 - \varepsilon$. Given s_j and t_j for all j in $[i]$ such that t_{i-1} is at most 1, the probability that t_i is 0 is at least $1 - \varepsilon$.*

In order to establish the above lemmas, we first introduce some additional notation. For each $i \geq -1$, we define S_i and T_i as follows. Let $S_{-1} = T_{-1} = \emptyset$ and for $i \geq 0$ let

$$S_i = \begin{cases} S_{i-1} \cup B_i \cup (C_i \cap N_{\supseteq}(x_i, X_i^2)) & \text{if } s_i \in \{0,1\}, \\ S_{i-1} \cup B_i & \text{if } s_i = 2, \\ S_{i-1} \cup B_{i-s_i+2} \cup N_{\subseteq}(x_{i-s_i+3}, X_i^1) & \text{otherwise.} \end{cases}$$

$$T_i = \begin{cases} T_{i-1} \cup E_i & \text{if } t_i = 0, \\ T_{i-1} \cup E_{i-t_i} \cup N_{\subseteq}(y_{i-t_i+1}, Y_i^1) & \text{otherwise.} \end{cases}$$

The set S_i (resp., T_i) contains all of the nodes whose labels need to be examined to determine the values of s_0 through s_i (resp., t_0 through t_i). Moreover, as we show in Lemma 2.5.13, the particular values of s_0 through s_i and t_0 through t_i bias the distribution of only a suffix of the labels of the nodes in $S_i \cup T_i$.

Lemma 2.5.13 *Let i be in $[(\log n)/b - 1]$. Given s_j and t_j for all j in $[i]$, we have*

1. *The variables $\lambda_{\geq 0}(u)$, for all u not in $S_i \cup T_i$, are independent and uniformly random.*

2. *There exists a subset S'_i of S_i of size at most $d+1$ such that (i) the variables $\lambda_{>i}(u)$, for all u in S'_i, are independent and uniformly random, and (ii) for each node u in $S_i \setminus S'_i$, $\mathcal{P}_{\leq i}(u)$ is false.*

3. *There exists at most one node v in T_i such that (i) the variable $\lambda_{>i}(v)$ is uniformly random, and (ii) for each node u in $T_i \setminus \{v\}$, $\mathcal{P}_{<i}(u)$ is false.*

Proof: We prove Parts 1, 2, and 3 for all $i \geq -1$. The proof is by induction. For the induction base we set $i = -1$. Part 1 follows directly from the random assignment of labels. For Part 2, we set S'_{-1} to $\emptyset$, and the desired claim holds since S_{-1} is $\emptyset$. The claim of Part 3 holds vacuously since T_{-1} is $\emptyset$.

For the induction hypothesis, we assume that Parts 1, 2, and 3 of the lemma hold for $i-1$. We first consider different cases depending on the value of s_i.

(a) $s_i = 3 + j$, $j \in [i]$: The event $s_i = 3 + j$ is equivalent to the event $(B_{i-j-1} \subseteq X_i^1) \wedge (A_{i-j} \not\subseteq X_i^1)$. We first condition on the event $B_{i-j-1} \subseteq X_i^1$ by invoking Corollary 2.5.4.1 with the substitution $(X_i^1, V \setminus (S_{i-1} \cup T_{i-1}), i-j)$ for (S, S', i). We next condition on the event $A_{i-j} \not\subseteq X_i^1$ by invoking Part 2 of Lemma 2.5.4 with the substitution $(x_{i-j}, X_i^1, V \setminus (S_{i-1} \cup T_{i-1} \cup B_{i-j-1}), \mathcal{P}_{\leq i})$ for $(u, S, S', \mathcal{P})$. By combining Part (i) of both invocations, we have (a.i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_{i-1} \cup T_{i-1} \cup B_{i-j-1} \cup N_{\subseteq}(x_{i-j}, X_i^1)$, are independent and uniformly random. By combining Part (ii) of both invocations, we have (a.ii) for each node v in $B_{i-j-1} \cup N_{\subseteq}(x_{i-j}, X_i^1)$, $\mathcal{P}_{\leq i}(v)$ is false.

We set S'_i to $S'_{i-1} \setminus (B_{i-j-1} \cup N_{\subseteq}(x_{i-j}, X_i^1))$.

(b) $s_i = 2$: The event $s_i = 2$ is equivalent to the event $(B_i \subseteq X_i^1) \wedge (A_i \not\supseteq X_i^{-1})$. We first condition on the event $B_i \subseteq X_i^1$ by invoking Corollary 2.5.4.1 with the substitution $(X_i^1, V \setminus (S_{i-1} \cup T_{i-1}), i)$ for (S, S', i). It follows from the preceding invocation and the definition of B_i that (b.i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_{i-1} \cup T_{i-1} \cup B_i$, are independent and uniformly random, and (b.ii) for each node v in $B_i \setminus \{x_{i+1}\}$, $\mathcal{P}_{\leq i}(v)$ is false.

We set S'_i to $S'_{i-1} \setminus (B_i \setminus \{x_{i+1}\})$.

(c) $s_i \in \{0,1\}$: The event $s_i \in \{0,1\}$ is equivalent to the event $(B_i \subseteq X_i^1) \wedge (A_i \supseteq X_i^{-1})$. We condition on the event $B_i \subseteq X_i^1$ by invoking Corollary 2.5.4 with the substitution $(X_i^1, V \setminus (S_{i-1} \cup T_{i-1}), i)$ for (S, S', i). It follows from the preceding invocation and the definition of B_i that (i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_{i-1} \cup T_{i-1} \cup B_i$, are independent and uniformly random, and (ii) for each node v in $B_i \setminus \{x_{i+1}\}$, $\mathcal{P}_{\leq i}(v)$ is false.

Let S_i' equal the set $\{v \in C_i \cap N_{\supseteq}(x_i, X_i^2) : \mathcal{P}_{\leq i}(v)\}$. By the definition of C_i, $|S_i'|$ is at most $d+1$. If $C_i \not\supseteq X_i^2$, then $C_i \subseteq N_{\supseteq}(x_i, X_i^2)$ and it follows from the definition of C_i that (c.i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_{i-1} \cup T_{i-1} \cup B_i \cup C_i$, are independent and uniformly random, and (c.ii) the variables $\lambda_{>i}(v)$, for all v in S_i', are independent and uniformly random, and for each node v in $(B_i \cup C_i) \setminus S_i'$, $\mathcal{P}_{\leq i}(v)$ is false. If $C_i \supseteq X_i^2$ then $C_i \supseteq N_{\supseteq}(x_i, X_i^2)$ and it follows from Part 3 of Lemma 2.5.4 that (c.i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_{i-1} \cup T_{i-1} \cup B_i \cup N_{\supseteq}(x_i, X_i^2)$, are independent and uniformly random, and (c.ii) the variables $\lambda_{>i}(v)$, for all v in S_i', are independent and uniformly random, and for each node v in $(B_i \cup N_{\supseteq}(x_i, X_i^2)) \setminus S_i'$, $\mathcal{P}_{\leq i}(v)$ is false.

We thus obtain from (a.i), (b.i), and (c.i) and the definition of S_i that (i) the variables $\lambda_{\geq 0}(u)$, for all u not in $S_i \cup T_{i-1}$, are independent and uniformly random. By the definitions of s_i and t_i, the particular values of s_i and t_i are independent of the suffix $\lambda_{>i}(u)$ of any node u. In particular, the variables $\lambda_{>i}(u)$, for all u in S_i', are independent and uniformly random. It follows from the preceding observation and claims (a.ii), (b.ii), and (c.ii) that (ii) the bits of $\lambda_{>i}(u)$, for all u in S_i', are independent and uniformly random, and for each node in $S_i \setminus S_i'$, $\mathcal{P}_{\leq i}(u)$ is false. We next consider two cases depending on the value of t_i.

(d) $t_i = 1 + j$, $j \in [i]$: This case is similar to Case (a). The event $t_i = 1 + j$ is equivalent to the event $(E_{i-j-1} \subseteq Y_i^1) \wedge (D_{i-j} \not\subseteq Y_i^1)$. We first condition on the event $E_{i-j-1} \subseteq Y_i^1$ by invoking Corollary 2.5.4.1 with the substitution $(Y_i^1, V \setminus (S_i \cup T_{i-1}), i-j)$ for (S, S', i). We next condition on the event $D_{i-j} \not\subseteq Y_i^1$ by invoking Part 2 of Lemma 2.5.4 with the substitution $(y_{i-j}, Y_i^1, V \setminus (S_i \cup T_{i-1} \cup E_{i-j-1}), \mathcal{P}_{\leq i})$ for $(u, S, S', \mathcal{P})$.

By combining Part (i) of both invocations, we have (d.i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_i \cup T_{i-1} \cup E_{i-j-1} \cup N_{\subseteq}(y_{i-j}, Y_i^1)$, are independent and uniformly random.

By combining Part (ii) of both invocations, we have (d.ii) for each node v in $E_{i-j-1} \cup N_{\subseteq}(y_{i-j}, Y_i^1)$, $\mathcal{P}_{\leq i}(v)$ is false.

(e) $t_i = 0$: This case is similar to Case (b). The event $t_i = 2$ is equivalent to the event $E_i \subseteq Y_i^1$. We invoke Corollary 2.5.4.1 with the substitution $(Y_i^1, V \setminus (S_i \cup T_{i-1}), i)$ for (S, S', i) to obtain that (e.i) the variables $\lambda_{\geq 0}(v)$, for all v not in $S_i \cup T_{i-1} \cup E_i$, are independent and uniformly random, and (e.ii) for each node v in $E_i \setminus \{y_{i+1}\}$, $\mathcal{P}_{\leq i}(v)$ is false.

To complete the induction step, we consider each part of the statement of the lemma separately:

1. By (i), (d.i), (e.i), and the definition of T_i, it follows that given s_j and t_j, $j \in [i]$, the variables $\lambda_{\geq 0}(u)$, for all u not in $S_i \cup T_i$, are independent and uniformly random.
2. This part follows directly from (ii) above.
3. By (d.ii) and (e.ii), it follows that given arbitrary values for s_j and t_j, $j \in [i]$ (i) the variable $\lambda_{>i}(y_{i+1})$ is uniformly random, and (ii) for each node u in $T_i \setminus \{y_{i+1}\}$, $\mathcal{P}_{\leq i}(u)$ is false.

Q.E.D.

Lemma 2.5.14 places upper bounds on the sizes of S_i and T_i.

Lemma 2.5.14 *Let i be a nonnegative integer. If s_i is in $\{0, 1\}$, S_i is a subset of X_i^3; otherwise, S_i is a subset of X_i^1. The set T_i is a subset of Y_i^1.*

Proof: The proof is by induction on i. For convenience, we set $i = -1$ for the induction base. Since $S_{-1} = T_{-1} = \emptyset$, the claims follow trivially. Let the claims of the lemma hold for S_{i-1} and T_{i-1}. We will show that $S_i \subseteq X_i^1$. The proof for T_i is along the same lines.

By the induction hypothesis, $S_{i-1} \subseteq X_{i-1}^3$. Since $\gamma^2 \leq 2^b$ by Equation 2.2, $X_{i-1}^3 \subseteq X_i^1$, hence implying that $S_{i-1} \subseteq X_i^1$. We now consider three cases depending on the value of s_i.

If s_i is in $\{0,1\}$, then $B_i \subseteq X_i^1$. Moreover, by Lemma 2.5.3, $N_{\supseteq}(x_i, X_i^2) \subseteq N(x, \Delta\gamma^2 2^{(i+1)b})$. Since $\Delta \le \gamma$ by Equation 2.4, $N(x, \Delta\gamma^2 2^{(i+1)b}) \subseteq X_i^3$. It thus follows that S_{i-1}, B_i, and $N_{\supseteq}(x_i, X_i^2)$ are all subsets of X_i^3. Thus, S_i is a subset of X_i^3.

If s_i is 2, then $B_i \subseteq X_i^1$. It thus follows that S_{i-1} and B_i are both subsets of X_i^3. Thus, S_i is a subset of X_i^3.

If s_i is greater than 2, then $B_{i-s_i+2} \subseteq X_i^1$. Moreover, $N_{\subseteq}(x_{i-s_i+3}, X_i^1)$ is a subset of X_i^1. Thus, S_i is a subset of X_i^1. Q.E.D.

The following lemma is used in the proof of Lemma 2.5.11.

Lemma 2.5.15 *Let i be in $[(\log n/b) - 1]$. Given s_k and t_k for all k in $[i]$ such that s_{i-1} is $3+j$ for some j in $[i+1]$, the probability that B_{i-j-1} is a subset of X_{i-1}^2 is at least $1-\varepsilon^2/2$. Given s_k and t_k for all k in $[i]$ such that t_{i-1} is $1+j$ for some j in $[i+1]$, the probability that E_{i-j-1} is a subset of Y_{i-1}^2 is at least $1-\varepsilon^2/2$.*

Proof: Let $\mathcal{E}$ denote the event that the $2i$ random variables s_k and t_k, $k \in [i]$, take the given values. Let us assume that $\mathcal{E}$ holds. We begin with the proof of the first claim. Since s_{i-1} is $3+j$, B_{i-j-1} is not a subset of X_{i-1}^1, and B_{i-j-2} is a subset of X_{i-1}^1.

By Part 1 of Lemma 2.5.13, it follows that given $\mathcal{E}$, the variables $\lambda_{\ge 0}(u)$, for all u not in $S_{i-1} \cup T_{i-1}$, are independent and uniformly random. By Lemma 2.5.14, $|S_{i-1} \cup T_{i-1}|$ is at most $\gamma 2^{ib+1}$. By Lemma 2.5.3, since $\gamma \ge \Delta^2$ by Equation 2.4, $n_{\subseteq}(x_{i-j-1}, X_{i-1}^2)$ is at least $\gamma^2 2^{ib}/\Delta$. Therefore, the probability that A_{i-j-1} is not a subset of $N_{\subseteq}(x_{i-j-1}, X_{i-1}^2)$ is at most

$$\begin{aligned}(1 - 1/2^{(i-j)b})^{(\gamma^2/\Delta - 2\gamma)2^{ib}} &\le e^{-(\gamma^2/\Delta-2\gamma)2^{jb}} \\ &\le \varepsilon^2/2.\end{aligned}$$

(The second step makes use of the following inequalities: (i) $\gamma \ge 4\Delta$, which is obtained from Equation 2.4, and (ii) $e^{-\gamma/\Delta} \le (4e^{-\gamma/4\Delta})^2/2 \le \varepsilon^2/2$, which is obtained from Equation 2.5.) Since $N_{\subseteq}(x_{i-j-1}, X_{i-1}^2)$ is a subset of X_{i-1}^2, the probability that A_{i-j-1} is not a subset of X_{i-1}^2 is at least $1-\varepsilon^2/2$. Since B_{i-j-2} is a subset of $X_{i-1}^1 \subseteq X_{i-1}^2$ and

$B_{i-j-1} = B_{i-j-2} \cup A_{i-j-1}$, we obtain that B_{i-j-1} is a subset of X^2_{i-1} with probability at least $1 - \varepsilon^2/2$.

The proof of the second claim is analogous to the above proof and is obtained by substituting (t, D, E, y, Y) for (s, A, B, x, X). Q.E.D.

We are now ready to prove Lemmas 2.5.11 and 2.5.12.

Proof of Lemma 2.5.11: Let $\mathcal{E}$ denote the event that the $2i$ random variables s_j and t_j, $j \in [i]$, take the given values. Let us assume that $\mathcal{E}$ holds. We begin with the proof of the first claim. Let s_{i-1} be $3 + j$ for some j in $[i]$. Thus, B_{i-j-1} is not a subset of X^1_{i-1}, and B_{i-j-2} is a subset of X^1_{i-1}.

We show that B_{i-j} is a subset of X^1_i with probability at least $1 - \varepsilon^2$. We first invoke Lemma 2.5.15 to obtain that (a) B_{i-j-1} is a subset of X^2_{i-1} with probability at least $1-\varepsilon^2/2$. Let us now assume that $\mathcal{E}$ and the event that B_{i-j-1} is a subset of X^2_{i-1} hold.

We now show (b) the probability that B_{i-j} is a subset of X^1_i is at least $1 - \varepsilon^2/2$. It follows from Part 1 of Lemma 2.5.13 that given $\mathcal{E}$ the variables $\lambda_{\geq 0}(u)$, for all u not in $S_{i-1} \cup T_{i-1}$, are independent and uniformly random. Thus, given $\mathcal{E}$ and the event that B_{i-j-1} is a subset of X^2_{i-1}, the variables $\lambda_{\geq 0}(u)$, for all u not in $S_{i-1} \cup T_{i-1} \cup X^2_{i-1}$, are independent and uniformly random. By Lemma 2.5.14, $S_{i-1} \subseteq X^1_{i-1}$ and $T_{i-1} \subseteq Y^1_{i-1}$. Therefore, the size of the set $S_{i-1} \cup T_{i-1} \cup X^2_{i-1}$ is at most $\gamma(\gamma+1)2^{ib}$. By Lemma 2.5.3, since $\gamma \geq \Delta^2$ by Equation 2.4, $n_{\subseteq}(x_{i-j}, X^1_i)$ is at least $\gamma 2^{(i+1)b}/\Delta$. Therefore, the probability that A_{i-j} is not a subset of $N_{\subseteq}(x_{i-j}, X^2_i)$ is at most

$$\begin{aligned}(1 - 1/2^{(i-j+1)b})^{(\gamma/\Delta - \gamma^2/2^b - \gamma/2^b)2^{(i+1)b}} &\leq e^{-(\gamma/\Delta - \gamma^2/2^b - \gamma/2^b)2^{jb}} \\ &\leq \varepsilon^2/2.\end{aligned}$$

(The second step makes use of the following inequalities: (i) $2\Delta(\gamma+1) \leq \Delta^2\gamma^3 \leq 2^b$, which is obtained from Equation 2.2, and (ii) $e^{-\gamma/2\Delta} \leq (4e^{-\gamma/4\Delta})^2/2 \leq \varepsilon^2/2$, which is obtained from Equation 2.5.) Thus, the probability that B_{i-j} is not a subset of X^1_i is at most $\varepsilon^2/2$.

It follows from (a) and (b) above that with probability at least $(1 - \varepsilon^2)$, s_i is less than s_{i-1}, thus establishing the first claim of the lemma. The proof of the second claim is anal-

ogous to the above proof and is obtained by substituting (t, D, E, y, Y) for (s, A, B, x, X).
Q.E.D.

Proof of Lemma 2.5.12: Let $\mathcal{E}$ denote the event that the random variables s_j, t_j, $j \in [i]$, take the given values. Let us assume that $\mathcal{E}$ holds. We begin with the proof of the first claim. If s_{i-1} is in $\{0, 1, 2\}$, B_{i-1} is a subset of X_{i-1}^1. If s_{i-1} is 3, then by Lemma 2.5.15, B_{i-1} is a subset of X_{i-1}^2 with probability at least $1 - \varepsilon^2/2$. We now assume that B_{i-1} is a subset of X_{i-1}^2.

We first show that (a) the probability that B_i is a subset of X_i^1 is at least $1 - \varepsilon/3 + \varepsilon^2/2$. By Part 1 of Lemma 2.5.13, it follows that given $\mathcal{E}$ the variables $\lambda_{\geq 0}(u)$, for all nodes u not in $S_{i-1} \cup T_{i-1}$, are independent and uniformly random. By Lemma 2.5.14, $|S_{i-1} \cup T_{i-1}|$ is at most $\gamma^3 2^{ib+1}$. By Lemma 2.5.3, since x_i is in X_{i-1}^2 and $2^b \geq \Delta^2\gamma$ (by Equation 2.2), $n_{\subseteq}(x_i, X_i^1)$ is at least $\gamma 2^{(i+1)b}/\Delta$. Therefore, the probability that A_i is not a subset of $N_{\subseteq}(x_i, X_i^1)$ is at most

$$\begin{aligned}(1 - 1/2^{(i+1)b})^{2^{(i+1)b}(\gamma/\Delta - 2\gamma^3/2^b)} &\leq e^{-(\gamma/\Delta - 2\gamma^3/2^b)} \\ &\leq e^{-\gamma/2\Delta} \\ &\leq \varepsilon/3 - \varepsilon^2/2.\end{aligned}$$

(The second inequality follows from the inequality $4\gamma^2\Delta \leq 2^b$, which is obtained from Equation 2.2. The last inequality follows from the inequalities (i) $e^{-\gamma/2\Delta} \leq \varepsilon/4$, which is obtained from Equation 2.5, and (ii) $\varepsilon/4 \leq \varepsilon/3 - \varepsilon^2/2$, which holds since $\varepsilon \leq 1/10$ by Equation 2.6.) This implies that the probability that A_i is not a subset of X_i^1 is at most $\varepsilon/3 - \varepsilon^2/2$.

We next show that (b) the probability that A_i is a superset of X_i^{-1} is at least $1 - \varepsilon/3$. Since $\Delta^2\gamma^3 \leq 2^b$ by Equation 2.2, Lemma 2.5.3 implies that $n_{\supseteq}(x_i, X_i^{-1}) \leq \Delta 2^{(i+1)b}/\gamma$. By Lemma 2.5.13, we have (i) the variables $\lambda_{\geq 0}(u)$, for all u not in $S_{i-1} \cup T_{i-1}$, are independent and uniformly random, and (ii) there are at most $d+1$ nodes in $S_{i-1} \cup T_{i-1}$ for which the predicate $\mathcal{P}_{<i}$ holds. Therefore, the probability that A_i is a subset of $N_{\supseteq}(x_i, X_i^{-1})$ is at most $(d+1)/2^b + \Delta/\gamma$, which is at most $\varepsilon/3$. It follows that the probability that A_i is not a superset of X_i^{-1} is at most $\varepsilon/3$.

We finally show that (c) given that B_i is a subset of X_i^1 and A_i is a superset of X_i^{-1}, the probability that C_i is a superset of X_i^2 is at least $1-\varepsilon/3$. We note that given $\mathcal{E}$, and the two events B_i is a subset of X_i^1 and A_i is a superset of X_i^{-1}, (i) the variables $\lambda_{\geq 0}(u)$, for all u not in $S_{i-1} \cup T_{i-1} \cup X_i^1$, are independent and uniformly random, and (ii) there exist at most $d+1$ nodes in $S_{i-1} \cup T_{i-1}$ for which the predicate $\mathcal{P}_{<i}$ holds.

We will place an upper bound on the probability that C_i is not a superset of X_i^2 by placing a lower bound on the probability that C_i is not a superset of $N_{\supseteq}(x_i, X_i^2)$, which is a superset of X_i^2. Let r_0 (resp., r_1) denote $n_{\supseteq}(x_i, X_i^{-1})$ (resp., $n_{\supseteq}(x_i, X_i^2)$). By definition, $n_{\supseteq}(x_i, X_i^{-1})$ is at least $2^{(i+1)b}/\gamma$. By Lemma 2.5.3, $n_{\supseteq}(x_i, X_i^2)$ is at most $\Delta\gamma^2 2^{(i+1)b}$.

We first show that the nodes in $N_{\supseteq}(x_i, X_i^2)$ are within a cost of $d \cdot c(x_i, x_{i+1})$. We note that $c(x_i, x_{i+1})$ is at least the difference of the radii of X_i^{-1} and X_{i-1}^2. Moreover, since $N_{\supseteq}(x_i, X_i^2)$ is a subset of X_i^3, $n_{\supseteq}(x_i, X_i^2)$ is at most the sum of the radii of X_i^3 and X_{i-1}^2. Since $(4\gamma^4)^{\log_\delta 2} \leq \gamma^2 \leq d$ by Equation 2.3, all of the nodes in $N_{\supseteq}(x_i, X_i^2)$ are within a cost of $d \cdot c(x_i, x_{i+1})$ from x_i.

It now follows that the probability that C_i is not a superset of X_i^2 is at most the probability that there exist d nodes in $N_{\supseteq}(x_i, X_i^2)$ whose $(i+1)b$ rightmost bits match a certain bit-sequence. This probability is at most

$$\begin{aligned}\binom{d+1}{d/2}(1/2^b)^{d/2} + \binom{\Delta\gamma^2 2^{(i+1)b}}{d/2}(1/2^{(i+1)b})^{d/2} &\leq (2e/2^b)^{d/2} + (e\Delta\gamma^2/d)^{d/2} \\ &\leq \varepsilon/3.\end{aligned}$$

(The second step follows from the inequalities $(2e/2^b)^{d/2} \leq \varepsilon/6$ and $(e\Delta\gamma^2/d)^d \leq \varepsilon/6$, both of which are derived from Equation 2.5.)

It follows from (a), (b), and (c) above that with probability at least $1-\varepsilon$, s_i is 0, thus establishing the first claim of the lemma. The proof of the second claim is analogous to the proof of (a) and is obtained by substituting (t, D, E, y, Y) for (s, A, B, x, X).

Q.E.D.

By the definitions of s_i and t_i, it follows that $0 \leq s_{i+1} \leq 3$ if $s_i \leq 2$, and $0 \leq s_{i+1} \leq s_i + 1$ otherwise. In addition, $0 \leq t_{i+1} \leq t_i + 1$, for all i. Let s_i' equal 0 if $s_i = 0$, 1 if $s_i \in \{1, 2, 3\}$, and $s_i - 2$ otherwise. Hence $0 \leq \max\{s_{i+1}', t_{i+1}\} \leq \max\{s_i', t_i\} + 1$, for all i. In Section 2.5.3.3 below, we analyze the random walk corresponding to the sequence $\langle \max\{s', t\} \rangle$.

2.5.3.3 Random walks

We begin the analysis of the random walk corresponding to the sequence $\langle \max\{s', t\} \rangle$ by proving several useful properties of certain random walks on a line. These properties are stated in Lemmas 2.5.16 through 2.5.21. The main technical claim of this section is Lemma 2.5.23.

Let $W(U, F)$ be a directed graph in which U is the set of nodes and F is the set of edges. For all u in U, let $\mathcal{D}_u$ be a probability distribution over the set $\{(u, v) \in F\}$. We define $\Pr_{\mathcal{D}_u}[(u, v) : (u, v) \notin F] = 0$, for convenience. A *random walk* on W starting at v_0 and according to $\{\mathcal{D}_u : u \in U\}$ is a random sequence $\langle v \rangle$ such that (i) v_i is in U and (v_i, v_{i+1}) is in F, for all i, and (ii) given any fixed (not necessarily simple) path $u_0, \ldots, u_i$ in W and any fixed u_{i+1} in U, $\Pr[v_{i+1} = u_{i+1} \mid (v_0, \ldots, v_i) = (u_0, \ldots, u_i)] = \Pr[v_{i+1} = u_{i+1} \mid v_i = u_i] = \Pr_{\mathcal{D}_{u_i}}[(u_i, u_{i+1})]$.

Let H be the directed graph with node set **N** (the set of nonnegative integers) and edge set $\{(i, j) : i \in \mathbf{N}, 0 \leq j \leq i + 1\}$. Let H' be the subgraph of H induced by the edges $\{(i + 1, i), (i, i + 1) : i \in \mathbf{N}\} \cup \{(0, 0), (1, 1)\}$.

Let p and q be reals in $(0, 1]$ such that $p \geq q$. We now define two random walks, $\omega_{p,q}$ and $\omega'_{p,q}$, on graphs H and H', respectively. The walk $\omega_{p,q} = \langle w \rangle$ is characterized by (i) $\Pr[w_{i+1} \leq j - 1 \mid w_i = j] \geq p$, for any integer $j > 1$, (ii) $\Pr[w_{i+1} = 0 \mid w_i = j] \geq q$, for j equal 0 or 1, and (iii) $\Pr[w_{i+1} = 2 \mid w_i = 1] \leq 1 - p$. The walk $\omega'_{p,q} = \langle w' \rangle$ is characterized by (i) $\Pr[w'_{i+1} = j - 1 \mid w'_i = j] = p$, for all integer $j > 1$, (ii) $\Pr[w'_{i+1} = 0 \mid w'_i = j] = q$, for j equal 0 or 1, and (iii) $\Pr[w'_{i+1} = 2 \mid w'_i = 1] = 1 - p$. We note that Lemmas 2.5.11 and 2.5.12 imply that the sequence $\langle \max\{s', t\} \rangle$ can be represented by the random walk $\omega_{p,q}$ with $p = 1 - 2\varepsilon^2$ and $q = 1 - 2\varepsilon$.

We analyze random walk $\omega_{p,q}$ by first showing that $\omega_{p,q}$ is more "favorable" than $\omega'_{p,q}$ with respect to the properties of interest. The random walk $\omega'_{p,q}$ is easier to analyze as it is exactly characterized by p and q. Lemmas 2.5.16 and 2.5.18 show that the bias of $\omega_{p,q}$

towards 0 is more than that of $\omega'_{p,q}$. Since the values of p and q are fixed throughout the following discussion, we omit the subscript p, q in the terms $\omega_{p,q}$ and $\omega'_{p,q}$ for convenience.

Lemma 2.5.16 *For all i and k in* **N**, *for random walks ω and ω', we have* $\Pr[w_i \leq k] \geq \Pr[w'_i \leq k]$.

Proof: We prove the claim by induction on i. The base case $i = 0$ is trivial. Assume the claim holds for i and any k. If $k \geq 1$, then we have

$$\begin{aligned}
\Pr[w'_{i+1} \leq k] &= \Pr[w'_i \leq k-1] + p\Pr[k \leq w'_i \leq k+1] \\
&= (1-p)\Pr[w'_i \leq k-1] + p\Pr[w'_i \leq k+1], \text{ and} \\
\Pr[w_{i+1} \leq k] &\geq \Pr[w_i \leq k-1] + p\Pr[k \leq w_i \leq k+1] \\
&= (1-p)\Pr[w_i \leq k-1] + p\Pr[w_i \leq k+1].
\end{aligned}$$

If $k = 1$, then we have

$$\begin{aligned}
\Pr[w'_{i+1} \leq 0] &= q\Pr[w'_i \leq 0] + q\Pr[w'_i = 1] \\
&= q\Pr[w'_i \leq 1], \text{ and} \\
\Pr[w_{i+1} \leq 0] &\geq q\Pr[w_i \leq 0] + q\Pr[w_i = 1] \\
&= q\Pr[w_i \leq 1].
\end{aligned}$$

The lemma now follows by induction. Q.E.D.

We now establish a probabilistic relationship between the number of steps it takes for the random walks ω and ω' to reach node 0 starting from a given node i. Let $z_i(\sigma)$ be the random variable denoting the number of steps taken to reach node 0 starting from node i, for a random walk σ.

Lemma 2.5.17 *For all ℓ and all $i > 0$, we have* $\Pr[z_i(\omega') \leq \ell] \leq \Pr[z_{i-1}(\omega') \leq \ell]$,

Proof: We use induction on ℓ. The base case $\ell = 0$ is trivial. Let $\ell \geq 1$. If $i > 2$ then

$$\begin{aligned}
\Pr[z_{i-1}(\omega') \leq \ell] &= p\Pr[z_{i-2}(\omega') \leq \ell - 1] + (1-p)\Pr[z_i(\omega') \leq \ell - 1] \\
&\geq p\Pr[z_{i-1}(\omega') \leq \ell - 1] + (1-p)\Pr[z_{i+1}(\omega') \leq \ell - 1] \\
&= \Pr[z_i(\omega') \leq \ell],
\end{aligned}$$

where the second step follows from the induction hypothesis. If $i = 2$, then we have

$$\begin{aligned}
\Pr[z_1(\omega') \leq \ell] &= q + (1 - q - (1 - p)) \Pr[z_1(\omega') \leq \ell - 1] + (1 - p) \Pr[z_2(\omega') \leq \ell - 1] \\
&\geq p \Pr[z_1(\omega') \leq \ell - 1] + (1 - p) \Pr[z_2(\omega') \leq \ell - 1] \\
&\geq p \Pr[z_1(\omega') \leq \ell - 1] + (1 - p) \Pr[z_3(\omega') \leq \ell - 1] \\
&= \Pr[z_2(\omega') \leq \ell],
\end{aligned}$$

where the second step follows from the induction hypothesis. If $i = 1$ then we have

$$\begin{aligned}
\Pr[z_0(\omega') \leq \ell] &= q + (1 - q) \Pr[z_1(\omega') \leq \ell - 1] \\
&\geq q \Pr[z_0(\omega') \leq \ell - 1] + (p - q) \Pr[z_1(\omega') \leq \ell - 1] + \\
&\quad +(1 - p) \Pr[z_2(\omega') \leq \ell - 1] \\
&= \Pr[z_1(\omega') \leq \ell],
\end{aligned}$$

where the second step follows from the induction hypothesis. Q.E.D.

We now use Lemma 2.5.17 to argue that the random variable $z_i(\omega)$ is stochastically dominated by the random variable $z_i(\omega')$.

Lemma 2.5.18 *For all i and ℓ in* **N**, *we have* $\Pr[z_i(\omega) \leq \ell] \geq \Pr[z_i(\omega') \leq \ell]$.

Proof: The proof is by induction on ℓ. The base case $\ell = 0$ is trivial. Let $p_j = \Pr[w_{i+1} \leq j - 1 \mid w_i = j]$, for $j > 1$, and $q_j = \Pr[w_{i+1} = j \mid w_i = j]$, for all j in **N**. Note that the following inequalities hold: (i) $p \leq p_j$, for all $j > 1$, (ii) $q \leq \min\{p_1, q_0\}$, and (iii) $p \geq q$.

If $i \geq 2$, then we have

$$\begin{aligned}
\Pr[z_i(\omega') \leq \ell] &= p \Pr[z_{i-1}(\omega') \leq \ell - 1] + (1 - p) \Pr[z_{i+1}(\omega') \leq \ell - 1] \\
&\leq p_i \Pr[z_{i-1}(\omega') \leq \ell - 1] + (1 - p_i) \Pr[z_{i+1}(\omega') \leq \ell - 1] \\
&\leq p_i \Pr[z_{i-1}(\omega') \leq \ell - 1] + q_i \Pr[z_i(\omega') \leq \ell - 1] + \\
&\quad +(1 - p_i - q_i) \Pr[z_{i+1}(\omega') \leq \ell - 1] \\
&\leq p_i \Pr[z_{i-1}(\omega) \leq \ell - 1] + q_i \Pr[z_i(\omega) \leq \ell - 1] + \\
&\quad +(1 - p_i - q_i) \Pr[z_{i+1}(\omega) \leq \ell - 1] \\
&= \Pr[z_i(\omega) \leq \ell].
\end{aligned}$$

(The second step holds because (i) $p \le p_i$, and (ii) $\Pr[z_{i-1}(\omega') \le \ell - 1] \ge \Pr[z_{i+1}(\omega') \le \ell - 1]$, which follows from Lemma 2.5.17. The third step holds since $\Pr[z_i(\omega') \le \ell - 1] \ge \Pr[z_{i+1}(\omega') \le \ell - 1]$ by Lemma 2.5.17. The fourth step follows from the induction hypothesis.) For $i = 1$, we have

$$\begin{aligned}
\Pr[z_1(\omega') \le \ell] &= q + (p - q)\Pr[z_1(\omega') \le \ell - 1] + (1 - p)\Pr[z_2(\omega') \le \ell - 1] \\
&\le p_1 + q_1 \Pr[z_1(\omega') \le \ell - 1] + (1 - p_1 - q_1)\Pr[z_2(\omega') \le \ell - 1] \\
&\le p_1 + q_1 \Pr[z_1(\omega) \le \ell - 1] + (1 - p_1 - q_1)\Pr[z_2(\omega) \le \ell - 1] \\
&= \Pr[z_1(\omega) \le \ell].
\end{aligned}$$

(The second step holds because (i) $q \le p_1$, (ii) $1 - p \ge 1 - p_1 - q_1$, and (iii) $\Pr[z_1(\omega') \le \ell - 1] \ge \Pr[z_2(\omega') \le \ell - 1]$, which follows from Lemma 2.5.17. The third step follows from the induction hypothesis.)

For $i = 0$, we have

$$\begin{aligned}
\Pr[z_0(\omega') \le \ell] &= q + (1 - q)\Pr[z_1(\omega') \le \ell - 1] \\
&\le q_0 + (1 - q_0)\Pr[z_1(\omega') \le \ell - 1] \\
&\le q_0 + (1 - q_0)\Pr[z_1(\omega) \le \ell - 1] \\
&= \Pr[z_0(\omega) \le \ell].
\end{aligned}$$

(The second step holds because $q \le q_0$. The third step follows from the induction hypothesis.) We have thus established the desired claim. Q.E.D.

We now show that, in a probabilistic sense, the time to return to 0 is smaller for ω than for ω'. For any i, let τ_i (resp., τ_i') denote the smallest $j \ge 0$ such that $w_{i+j} = 0$ (resp., $w'_{i+j} = 0$). We note that by letting $\langle w \rangle$ represent $\langle \max\{s', t\} \rangle$, the terminating step τ is given by $i^* + \tau_{i^*}$. Lemma 2.5.20 shows that, for any i, the random variable τ_i is stochastically dominated by the random variable τ_i'. We first prove the following technical lemma:

Lemma 2.5.19 *Let m be a nonnegative integer and let $\langle n \rangle$ be a sequence of m nonincreasing reals. Let $\langle p \rangle$ and $\langle q \rangle$ be two sequences of m reals each such that (i) for all j in $[m]$,*

$\sum_{0 \le i \le j} p_i \ge \sum_{0 \le i \le j} q_i$ *and (ii)* $\sum_{0 \le i \le m} p_i = \sum_{0 \le i \le m} q_i$. *Then, we have:*

$$\sum_{0 \le i \le m} p_i n_i \ge \sum_{0 \le i \le m} q_i n_i.$$

Proof: The proof is by induction on m. The induction basis is trivial. For the induction hypothesis, we assume that the statement of the lemma holds for m. We now establish the claim for $m + 1$.

$$\begin{aligned}
\sum_{0 \le i \le m+1} p_i n_i &= q_0 n_0 + (p_0 - q_0) n_0 + \sum_{1 \le i \le m+1} p_i n_i \\
&\ge q_0 n_0 + (p_0 - q_0 + p_1) n_1 + \sum_{2 \le i \le m+1} p_i n_i \\
&\ge q_0 n_0 + \sum_{1 \le i \le m+1} q_i n_i \\
&= \sum_{0 \le i \le m} q_i n_i.
\end{aligned}$$

(The third step follows from the induction hypothesis and the inequalities $n_0 \ge n_1$ and $p_0 \ge q_0$. We note that the induction hypothesis can be invoked since $p_0 - q_0 + p_1 + \sum_{2 \le i \le j} p_i \le \sum_{1 \le i \le j} q_i$ and $p_0 - q_0 + p_1 + \sum_{2 \le i \le m+1} = \sum_{1 \le i \le m+1} q_i$.) Q.E.D.

Lemma 2.5.20 *For any* i *and* $j \ge i$, *we have* $\Pr[\tau_i \le j] \ge \Pr[\tau_i' \le j]$.

Proof: The desired claim follows from the following argument:

$$\begin{aligned}
\Pr[\tau_i \le j] &= \sum_{0 \le k \le i} \Pr[w_i = k] \Pr[z_k(\omega) \le j] \\
&\ge \sum_{0 \le k \le i} \Pr[w_i = k] \Pr[z_k(\omega') \le j] \\
&\ge \sum_{0 \le k \le i} \Pr[w_i' = k] \Pr[z_k(\omega') \le j] \\
&= \Pr[\tau_i' \le j].
\end{aligned}$$

(The first step follows from the definitions of τ_i and $z_i(\omega)$. For the second step, we use Lemma 2.5.18. For the third step we first invoke Lemma 2.5.16 and then invoke Lemma 2.5.19 with the substitution $(i, k, \Pr[w_i = k], \Pr[w_i' = k], \Pr[z_k(\omega') \le j])$ for (m, i, p_i, q_i, n_i). We note that one of the conditions for the latter invocation, namely, $\Pr[z_k(\omega') \le j]$ is nonincreasing with k, follows from Lemma 2.5.17. The fourth step follows from the definitions of τ_i' and $z_i(\omega')$.) Q.E.D.

Lemma 2.5.20 indicates that we can obtain an upper bound on the time taken for the random walk ω to return to 0 by deriving a corresponding bound for the random walk ω'. Indeed, we will use Lemma 2.5.20 to obtain an upper bound on the length of any "excursion" in ω. An *excursion* of length ℓ in a graph W with node set **N** is a walk that starts at node 0 and first returns to the start node at time ℓ, for all ℓ in **N**. For all i such that $w_i = 0$, let $\ell_i(\omega)$ be the random variable that gives the length of the excursion in ω starting at time i. We note that for all i, $\ell_i(\omega)$ equals $z_0(\omega)$.

The following lemma, which describes a probabilistic recurrence relation for the length of an excursion in ω', is proved using a classical combinatorial result known as Raney's lemma [17, 46].

Lemma 2.5.21 *Let p and q satisfy the inequality $1 - p \leq (p - q)^2$. For all i and ℓ in* **N**, *we have* $\Pr[\ell_i(\omega') = \ell + 1 \mid w'_i = 0] \leq \max\{1 - q, 5(p - q)\} \Pr[\ell_i(\omega') = \ell \mid w'_i = 0]$.

Proof: Since ω' is a random walk,

$$\Pr[\ell_i(\omega') = \ell \mid (w'_0, \ldots, w'_{i-1}, w'_i) = (u_0, \ldots, u_{i-1}, 0)] = \Pr[\ell_0(\omega') = \ell \mid w'_0 = 0]$$

for any $u_0, \ldots, u_{i-1}$ in **N**. For the remainder of the proof, we assume without loss of generality that i is 0.

For $\ell = 1$, the desired claim holds since $\Pr[\ell_0(\omega') = 2] / \Pr[\ell_0(\omega') = 1] = (1 - q)$. We now consider $\ell \geq 2$. Let $\mathcal{E}_j$ denote the event that the random walk does not reach node 0 in the first j steps. That is, $\mathcal{E}_j$ is the event that w'_k is nonzero for all k in $[1, j]$. For all j, let α_j denote the probability that w'_{j+1} is 1 and $\mathcal{E}_{j+1}$ holds, given that w'_1 is 1. For convenience, we assume that α_{-1} equals $1/(p - q)$. We obtain that:

$$\Pr[\ell_0(\omega') = \ell] = (1 - q) \cdot \alpha_{\ell - 2} \cdot q. \tag{2.8}$$

It thus follows that the ratio of $\Pr[\ell_0(\omega') = \ell + 1]$ and $\Pr[\ell_0(\omega') = \ell]$ equals $\alpha_{\ell-1}/\alpha_{\ell-2}$. The remainder of the proof is devoted to obtaining an upper bound on α_{j+1}/α_j for all $j \geq 0$.

Let β_m denote the probability that the following conditions hold given that $w'_1 = 1$: (i) $\mathcal{E}_{2m+1}$ holds, (ii) $w'_{2m+1} = 1$, and (iii) the edge $(1, 1)$ is not traversed in any of the first

$2m+1$ steps. Using Raney's lemma [17, 46], we have $\beta_m = \frac{1}{2m+1}\binom{2m+1}{m}(p(1-p))^m$. By the definitions of α_j and β_m, it follows that:

$$\begin{aligned}
\alpha_j &= \sum_{0 \le m \le \lfloor j/2 \rfloor} \beta_m \cdot (p-q) \cdot \alpha_{j-2m-1} \\
&= \sum_{0 \le m \le \lfloor j/2 \rfloor} \frac{1}{2m+1}\binom{2m+1}{m}(p(1-p))^m \cdot (p-q) \cdot \alpha_{j-2m-1}.
\end{aligned}$$

We now prove by induction on $j \ge 2$ that α_{j+1}/α_j is at most $5(p-q)$. The induction base holds since α_0 is 1 and α_1 is $5(p-q)$. For the induction hypothesis, we assume that α_{j+1}/α_j is at most $5(p-q)$ for all $j \le k-1$. If k is even, then we have:

$$\begin{aligned}
\alpha_{k+1}/\alpha_k &\le \max_{0 \le m \le k/2} \alpha_{k-2m}/\alpha_{k-2m-1} \\
&\le 5(p-q),
\end{aligned}$$

where the last step follows from the induction hypothesis. If k is odd, then α_{k+1}/α_k is at most

$$\begin{aligned}
&\max\left\{\left(\frac{1}{k}\binom{k}{(k-1)/2}(p-q)^2 + \frac{1}{k+2}\binom{k+2}{(k+1)/2}p\right) \Big/ \left(\frac{1}{k}\binom{k}{(k-1)/2}(p-q)\right),\right. \\
&\left.\max_{0 \le m \le (k-3)/2\rfloor} \alpha_{k-2m}/\alpha_{k-2m-1}\right\} \le \max\{5(p-q), 5(p-q)\} = 5(p-q),
\end{aligned}$$

where the second step follows from the induction hypothesis along with the inequalities $1-p \le (p-q)^2$ and

$$\binom{k+2}{(k+1)/2} \le 4\binom{k}{(k-1)/2}.$$

The claim of the lemma follows from the upper bound on α_{k+1}/α_k and Equation 2.8.

Q.E.D.

We are now ready to use the properties of the random walks ω and ω' that are stated in Lemmas 2.5.20 and 2.5.21 to analyze the random walk obtained by the sequence $\langle \max\{s', t\}\rangle$. We set $p = 1 - 2\varepsilon^2$ and $q = 1 - 2\varepsilon$. Lemmas 2.5.11 and 2.5.12 imply that ω characterizes the random walk corresponding to the sequence $\langle \max\{s', t\}\rangle$. Consider the random walk ω'. We define a sequence $\langle v\rangle$ associated with $\langle w'\rangle$ as follows: If $w'_j = 0$ then $v_j = G$; otherwise, $v_j = B$.

Lemma 2.5.22 *Let i be in $[(\log n/b) - 1]$. Given any fixed sequence $\langle v \rangle_{i-1}$ of B, G values, the probability that w'_i is 0 is at least $1 - 10\varepsilon$.* *Q.E.D.*

Proof: Assume that $v_j = G$. What is the probability that $v_i = G$, $i > j$, if we know that $v_k = B$, for all integers k in the interval $[j+1, i)$? From Lemma 2.5.21, it follows that this probability is at least $1 - 10\varepsilon$. This is because Lemma 2.5.21 states that $1 - \max\{2\varepsilon, 10(\varepsilon - \varepsilon^2)\}$ is a lower bound on the probability that there is an excursion of length $i - j$ starting at j in H', given that there is an excursion of length at least $i - j$ starting at j in H'. (Note that $1 - p \leq (p - q)^2$ since $\varepsilon < 1/10$ by Equation 2.6.)

Given any fixed B, G sequence $\langle u \rangle_{j-1}$, we have that $\Pr[v_i = G \mid (v_0, \ldots, v_{i-1}) = = (u_0, \ldots, u_{j-1}, G, B, \ldots, B)]$ is equal to $\Pr[v_i = G \mid (v_j, \ldots, v_{i-1}) = (G, B, \ldots, B)]$. Since this holds for any $j > 0$ and since $w'_i = 0$ if and only if $v_i = G$, we have $\Pr[w'_i \mid (v_0, \ldots, v_{i-1}) = (u_0, \ldots, u_{i-1})] \geq 1 - 10\varepsilon$. Q.E.D.

Our main technical claim concerning the random walk ω now follows from Lemmas 2.5.20 and 2.5.22.

Lemma 2.5.23 *For any i in $[(\log n)/b - 1]$ and any nonnegative integer j, the probability that $\tau_i \geq j$ is at most $(10\varepsilon)^j$.*

Proof: By Lemma 2.5.22, the probability that τ'_i is at least j is at most $(10\varepsilon)^j$. The desired claim then follows from Lemma 2.5.20. Q.E.D.

2.5.3.4 Proofs of Theorems 1 and 2

We first derive upper bounds on $E[c(x_i, x_{i+1})]$ and $E[c(y_i, y_{i+1})]$, for all i, using Lemma 2.5.23. Recall that a_i and b_i denote the radii of X_i^1 and Y_i^1, respectively, and i^* is the smallest integer i such that a_i is at least $c(x, y)$.

Lemma 2.5.24 *For any i in $[(\log n)/b - 1]$, $E[c(x_i, x_{i+1})] = O(a_i)$ and $E[c(y_i, y_{i+1})] = O(b_i)$.*

Proof: We first observe that $c(x_i, x_{i+1})$ (resp., $c(y_i, y_{i+1})$) is at most a_k (resp., b_k), where k is the least $j \geq i$ such that s_j (resp., t_j) belongs to $\{0, 1, 2\}$ (resp., $\{0\}$); if such a j does not exist, then k is $(\log n)/b - 1$. Thus, k is at most $i + \tau_i$. By Lemma 2.5.23, it follows that for any $j \geq i$, the probability that $k \geq j$ is at most $(10\varepsilon)^{j-i}$. By Lemma 2.5.9, we thus have

$$E[c(x_i, x_{i+1})] \quad \leq \quad \sum_{j \geq i} a_i (10\varepsilon)^{j-i} 2^{\lceil b \log_\delta 2 \rceil (j-i)} \quad = \quad O(a_i)$$

and

$$E[c(y_i . y_{i+1})] \quad \leq \quad \sum_{j \geq i} b_i (10\varepsilon)^{j-i} 2^{\lceil b \log_\delta 2 \rceil (j-i)} \quad = \quad O(b_i),$$

since $10\varepsilon 2^{\lceil b \log_\delta 2 \rceil} < 1$ by Equation 2.6. Q.E.D.

We now use Lemmas 2.5.9, 2.5.23, and 2.5.24 to establish Theorem 1.

Proof of Theorem 1: By Equation 2.7, the expected cost of the read operation is bounded by the expected value of $f(\ell(A)) \sum_{0 \leq i < \tau} O(c(x_i, x_{i+1}) + c(y_i, y_{i+1}))$. (Recall that τ is the smallest integer $i \geq i^*$ such that $(s_i, t_i) = (0, 0)$.) We upper bound the two terms $E[\sum_{0 \leq i < i^*} (c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$ and $E[\sum_{i^* \leq i < \tau} (c(x_i, x_{i+1}) + c(y_i, y_{i+1}))]$ separately. By Lemmas 2.5.9 and 2.5.24, the first term is $O(a_{i^*} + b_{i^*})$. We upper bound the second term as follows. Since τ is $i^* + \tau_{i^*}$, we obtain from Lemma 2.5.23 that for any $j \geq 0$, the probability that $\tau \geq i^* + j$ is at most $(10\varepsilon)^j$. Therefore,

$$\begin{aligned} E[\sum_{i^* \leq i < \tau} (c(x_i, x_{i+1}) + c(y_i, y_{i+1}))] &\leq \sum_{j \geq 0} j (10\varepsilon)^j (a_{i^*+j} + b_{i^*+j}) \\ &\leq \sum_{j \geq 0} j (10\varepsilon)^j 2^{j \lceil b \log_\delta 2 \rceil} (a_{i^*} + b_{i^*}) \\ &= O(a_{i^*} + b_{i^*}) \\ &= O(c(x, y)). \end{aligned}$$

(The second step follows from Lemma 2.5.9. The third step holds since $10\varepsilon 2^{\lceil b \log_\delta 2 \rceil} < 1$ by Equation 2.6. The fourth step follows from Lemma 2.5.9.) Q.E.D.

Theorem 2 follows from Lemmas 2.5.6, 2.5.9, and 2.5.24.

Proof of Theorem 2: Consider an insert operation performed by x for any object. The expected cost of the operation is bounded by $E[\sum_{0\le i<\log n/b} c(x_i, x_{i+1})]$, which by Lemmas 2.5.9 and 2.5.24 is $O(a_{(\log n)/b-1}) = O(C)$.

We now consider the cost of the delete operation. By Lemma 2.5.6, for each i, the number of reverse (i, j)-neighbors of x_i for any j is $O(\log n)$ with high probability, where x_i is the ith node in the primary neighbor sequence of x. Therefore, the expected cost of the delete operation executed by x is bounded by the product of $E[\sum_{0\le i<\log n/b} c(x_i, x_{i+1})]$ and $O(\log n)$. By Lemma 2.5.24, it follows that the expected cost of a delete operation is $O(C \log n)$. Q.E.D.

2.5.4 Auxiliary memory

Proof of Theorem 3: We first place an upper bound on the size of the neighbor table of any u in V. By definition, the number of primary and secondary neighbors of u is at most $(d + 1)2^b(\log n)/b$, which is $O(\log n)$. By Corollary 2.5.6.1, the number of reverse neighbors of u is $O(\log^2 n)$ with high probability.

We next place an upper bound on the size of the pointer list of any u in V. The size of *Ptr*(u) is at most the number of triples of the form $(A, v, \cdot)$, where A is in $\mathcal{A}$ and v is in V such that (i) there exists i in $[(\log n)/b]$ such that v is an i-leaf of u, (ii) $A[j] = u[j]$ for all j in $[i]$, and (iii) A is in the main memory of v.

By Lemma 2.5.7, the number of i-leaves of u is $O(2^{ib} \log n)$ with high probability. The probability that $A[j] = u[j]$, for all j in $[i]$, is at most $1/2^{ib}$. Since the number of objects in the main memory of any node is at most ℓ, it follows that with high probability, $|$*Ptr*$(u)|$ is at most $\sum_{i\in[\log n/b]} O(\ell \log n)$ which is $O(\ell \log^2 n)$.

Combining the bounds on the sizes of the neighbor table and pointer list, we obtain that the size of the auxiliary memory of u is $O(\ell \log^2 n)$ with high probability. Q.E.D.

2.5.5 Adaptability

Proof of Theorem 4: By Lemma 2.5.6, for any node u, the number of nodes for which u is a primary or secondary neighbor is $O(\log n)$ expected and $O(\log^2 n)$ with high probability. Moreover, u is a reverse neighbor of $O(\log n)$ nodes since u has $O(\log n)$ primary neighbors. Therefore, the adaptability of our scheme is $O(\log n)$ expected and $O(\log^2 n)$ with high probability. Q.E.D.

2.6 Future work

We would like to extend our study to more general classes of cost functions and determine tradeoffs among the various complexity measures. It would also be interesting to consider models that allow faults in the network. We believe that our access scheme can be extended to perform well in the presence of faults, as the distribution of control information in our scheme is balanced among the nodes of the network.

Chapter 3

Fast Algorithms for Finding O(Congestion+Dilation) Packet Routing Schedules

3.1 Introduction

In this chapter, we consider the problem of scheduling the movements of packets whose paths through a network have already been determined. The problem is formalized as follows. We are given a network with n nodes (switches) and m edges (communication channels). Each node can serve as the source or destination of an arbitrary number of *packets* (or *cells* or *flits*, as they are sometimes referred to). Let N denote the total number of packets to be routed. The goal is to route the N packets from their origins to their destinations via a series of synchronized time steps, where at each step at most one packet can traverse each edge, and each packet can traverse at most one edge at each step. Without loss of generality, we assume that all edges in the network are used in the path of some packet, and thus that m gives the number of such edges (all the other edges are irrelevant to our problem).

Figure 3.1 shows a 5-node network in which one packet is to be routed to each node. The shaded nodes in the figure represent switches, and the edges between the nodes represent channels. A packet is depicted as a square box containing the label of its destination.

This is joint work with Tom Leighton, MIT, and Bruce Maggs, CMU. This work appears in [28].

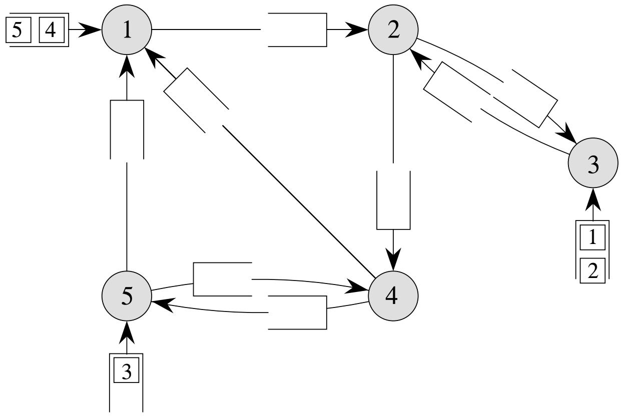


Figure 3.1: A graph model for packet routing.

During the routing, packets wait in three different kinds of queues. Before the routing begins, packets are stored at their origins in special *initial queues*. When a packet traverses an edge, it enters the *edge queue* at the end of that edge. A packet can traverse an edge only if at the beginning of the step, the edge queue at the end of that edge is not full. Upon traversing the last edge on its path, a packet is removed from the edge queue and placed in a special *final queue* at its destination. In Figure 3.1, all of the packets reside in initial queues. For example, packets 4 and 5 are stored in the initial queue at node 1. In this example, each edge queue is empty, but has the capacity to hold two packets. Final queues are not shown in the figure. Independent of the routing algorithm used, the sizes of the initial and final queues are determined by the particular packet routing problem to be solved. Thus, any bound on the maximum queue size required by a routing algorithm refers only to the edge queues.

We focus on the problem of timing the movements of the packets along their paths. A *schedule* for a set of packets specifies which move and which wait at each time step. The *length* of a schedule is the number of time steps required to route all the packets to their destinations according to the schedule. Given any underlying network, and any selection of paths for the packets, our goal is to produce a schedule for the packets that minimizes the length of the schedule and the maximum queue size needed to route all of the packets to their destinations.

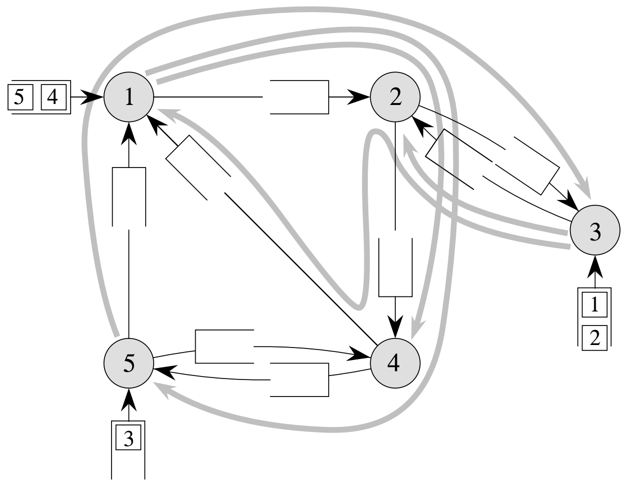


Figure 3.2: A set of paths for the packets with dilation $d = 3$ and congestion $c = 3$.

Of course, there is a strong correlation between the time required to route the packets and the selection of the paths. In particular, the maximum distance, d, traveled by any packet is always a lower bound on the time. We call this distance the *dilation* of the paths. Similarly, the largest number of packets that must traverse a single edge during the entire course of the routing is a lower bound. We call this number the *congestion*, c, of the paths. Figure 3.2 shows a set of paths for the packets of Figure 3.1 with dilation 3 (since the path followed by the packet going from node 5 to node 3 has length 3) and congestion 3 (since three paths use the edge between nodes 1 and 2).

3.1.1 Related work

Given any set of paths with congestion c and dilation d, in any network, it is straightforward to route all of the packets to their destinations in cd steps using queues of size c at each edge. In this case the queues are big enough that a packet can never be delayed by a full queue in front, so each packet can be delayed at most $c - 1$ steps at each of at most d edges on the way to its destination.

In [27], Leighton, Maggs, and Rao showed that there are much better schedules. In particular, they established the existence of a schedule using $O(c + d)$ steps and constant-size queues at every edge, thereby achieving the naive lower bounds (up to constant factors) for any routing problem. The result is highly robust in the sense that it works for any set of edge-simple paths and any underlying network. (Ā priori, it would be easy to imagine that there might be some set of paths on some network that requires more than $\Omega(c+d)$ steps or larger than constant-size queues to route all the packets.) The method that they use to show the existence of optimal schedules, however, is not constructive. In other words, prior to this work, the fastest known algorithms for producing schedules of length $O(c+d)$ with constant-size edge queues required time that is exponential in the number of packets.

Scheideler recently presented in [49] an alternative simpler proof for the existence of $O(c+d)$-step schedules that only require edge queues of size 2. The main idea in his proof is to decompose the problem in a different way by using so-called "secure edges".

In [35], Meyer auf der Heide and Scheideler showed the existence of an off-line protocol that only requires edge queues of size 1. However, the schedule produced by this protocol has length $O([d + c(\log(c+d))(\log\log(c+d))]\log\log\log^{(1+\epsilon)}(c+d))$, for any constant $\epsilon > 0$.

For the class of *leveled* networks, Leighton, Maggs, Ranade, and Rao [25] showed that there is a simple on-line randomized algorithm for routing the packets to their destinations within $O(c+L+\log N)$ steps, with high probability, where L is the number of levels in the network, and N is the total number of packets. (In a leveled network with L levels, each node is labeled with a level number between 0 and $L-1$, and every edge that has its tail on level i has its head on level $i+1$, for $0 \le i < L-1$.)

Mansour and Patt-Shamir [33] showed that if packets are routed greedily on shortest paths, then all of the packets reach their destinations within $d+N$ steps. These schedules may be much longer than optimal, however, because N may be much larger than c. Meyer auf der Heide and Vöcking [36] devised a simple on-line randomized algorithm that routes all packets to their destinations in $O(c+d+\log N)$ steps, with high probability, provided that the paths taken by the packets are short-cut free (e.g., shortest paths).

Recently, Rabani and Tardos [42], and Ostrovsky and Rabani [39] extended the main ideas used in [27], and in the centralized algorithm presented in this chapter, to obtain on-line local control algorithms for the general packet routing problem that produce near-optimal schedules. More specifically, Tardos and Rabani [42] show a randomized on-line algorithm that with high probability delivers all packets in $O(c + d((\log^* N)^{O(\log^* N)}) + \log^6 N)$ steps; Ostrovsky and Rabani [39] improved on this result by presenting a randomized on-line algorithm that delivers all the packets to their destinations in $O(c + d + \log^{(1+\epsilon)} N)$ steps with high probability, for arbitrary $\epsilon > 0$.

It was also in recent work that Srinivasan and Teo [53] answered a long-standing question: Given source and destination nodes for each packet, can we select the routing paths for the N packets, with congestion c and dilation d, in order to approximate the minimum value of $c + d$ (over all possible choices of paths) to within a constant factor ? (Finding the minimum value of $c + d$ is NP-hard.) They provided an algorithm that selects such paths in polynomial time; by applying our algorithm on the selected paths, Srinivasan and Teo described the first off-line constant factor approximation algorithm for routing N packets (if we are only given the source and destination nodes of each packet) using constant-size queues. It is interesting to note that there is still no polynomial-time algorithm known for which the congestion c alone is asymptotically optimal: It was clever (and crucial) that Srinivasan and Teo minimized the sum $c + d$ rather than just c.

The problem of scheduling packets through given paths strongly relates to network emulations via embeddings, as we will see. Koch et al. in [24], and Maggs and Schwabe in [32] address the problem of performing network emulations via embeddings.

Shmoys, Stein, and Wein [51] give randomized and deterministic algorithms that produce schedules of length within a polylogarithmic factor of that of an optimal schedule, for job-shop scheduling when jobs are not assumed to have unit length and a machine may have to work on the same job more than once. Our results can be used to find schedules of length within a constant factor of the optimal schedule for the less general job-shop problem when jobs have unit length and a machine can work at most once on any job, as discussed below.

3.1.2 Our results

In this chapter, we show how to produce schedules of length $O(c + d)$ in $O(m(c + d)(\log \mathcal{P})^4(\log\log \mathcal{P}))$ time steps, with probability at least $1 - 1/\mathcal{P}^\delta$, for any constant $\delta > 0$, where m is the number of distinct edges traversed by some packet in the network. The schedules can also be found in polylogarithmic time on a parallel computer using $O(m(c + d)(\log \mathcal{P})^4(\log\log \mathcal{P}))$ work, with probability at least $1 - 1/\mathcal{P}^\delta$.

The algorithm for producing the schedules is based on an algorithmic form of the Lovász Local Lemma (see [12] or [52, pp. 57–58]) discovered by Beck [8]. Showing how to modify Beck's arguments so that they can be applied to scheduling problems is the main contribution of our work. Once this is done, the construction of asymptotically optimal routing schedules is accomplished using the methods of [27].

The result has several applications. For example, if a particular routing problem is to be performed many times over, then it may be feasible to compute an asymptotically optimal schedule once using global control. This situation arises in network emulation problems (see Chapter 1). Suppose a network G is being emulated by a host network H by embedding G into H. The algorithm described in this chapter can be used to produce a schedule in which the packets are routed to their destinations in $O(c + d)$ steps. Thus, H can emulate each step of G in $O(l + c + d)$ steps, where l is the load of this embedding (i.e., the maximum number of nodes of G that are mapped to a node of H).

The result also has applications to job-shop scheduling. In particular, consider a scheduling problem with jobs $j_1, \ldots, j_r$, and machines $m_1, \ldots, m_s$, for which each job must be performed on a specified sequence of machines. In this application, we assume that each job occupies each machine that works on it for a unit of time, and that no machine has to work on any job more than once. Of course, the jobs correspond to packets, and the machines correspond to edges in the packet routing problem. Hence, we can define the *dilation* of the scheduling problem to be the maximum number of machines that must work on any job, and the *congestion* to be the maximum number of jobs that have to be run on any machine. As a consequence of the packet routing result, we know that any scheduling problem can be solved in $O(c + d)$ steps. In addition, we know that there is a schedule for which each job waits at most $O(c+d)$ steps before it starts running, and that each job waits at most a constant number of steps in between consecutive machines. The queue of jobs waiting for any machine will also always be at most a constant.

3.1.3 Outline

The remainder of the chapter is divided into sections as follows. In Section 3.2, we give a very brief overview of the non-constructive proof in [27]. We also introduce some definitions, and present two important propositions and a new lemma that will be of later use. In Section 3.3, we describe how to make the non-constructive method in [27] constructive. In Section 3.4, we analyze the running time of the algorithm. The propositions presented in Sections 3.2 and 3.3 are meant to replace (and are numbered according to) some of the lemmas in [27].

In Section 3.5, we show how to parallelize the scheduling algorithm. We conclude with some remarks in Section 3.6.

3.2 Preliminaries

In [27], Leighton, Maggs, and Rao proved that for any set of packets whose paths are edge-simple and have congestion c and dilation d, there is a schedule of length $O(c+d)$ in which at most one packet traverses each edge of the network at each step, and at most a constant number of packets wait in each queue at each step. (An *edge-simple* path uses no edge more than once.) Note that there are no restrictions on the size, topology, or degree of the network or on the number of packets.

The strategy for constructing an efficient schedule is to make a succession of refinements to the "greedy" schedule, S_0, in which each packet moves at every step until it reaches its final destination. This initial schedule is as short as possible: Its length is only d. Unfortunately, as many as c packets may traverse an edge at a single time step in S_0, whereas in the final schedule at most one packet is allowed to traverse an edge at each step. Each refinement will bring us closer to meeting this requirement.

The proof uses the Lovász Local Lemma ([12] or [52, pp. 57–58]) at each refinement step. Given a set of "bad" events in a probability space, the lemma provides a simple inequality that, when satisfied, guarantees that with probability greater than zero, no bad event occurs. The inequality relates the probability that each bad event occurs with the dependence among them. A set of events $A_1, A_2, \ldots, A_q$ in a probability space has *dependence* at most b if every event is mutually independent of some set of $q-b-1$ other events. The lemma is non-constructive; for a discrete probability space, it shows only that there exists some elementary outcome that is not in any bad event.

Lemma [Lovász] *Let $A_1, A_2, \ldots, A_q$ be a set of "bad" events, each occurring with probability p, and with dependence at most b. If $4pb < 1$, then with probability greater than zero, no bad event occurs.* Q.E.D.

Before proceeding, we need to introduce some notation. A *T-frame* — or a *frame of size T* — is a sequence of T consecutive time steps. The *congestion* of an edge g in a T-frame is the number of packets that traverse g in this frame; the *relative congestion* of edge g in a T-frame is given by the congestion of g in the frame divided by the frame size T. The *frame congestion* in a T-frame is the largest congestion of an edge in the frame; the *relative congestion* in a T-frame is the largest relative congestion of an edge in the frame.

3.2.1 A pair of tools for later use

In this section we re-state Lemma 3.5 of [27] and we prove Proposition 3.6, which replaces Lemma 3.6 of [27]. Both will be used in the proofs in Section 3.3.

Lemma 3.5 *[27] In any schedule, if the number of packets that traverse a particular edge g in any y-frame is at most Ry, for all y between T and $2T-1$, then the number of packets that traverse g in any y-frame is at most Ry, for all $y \geq T$.*

Proof: Consider a frame τ of size T', where $T' > 2T-1$. The first $(\lfloor T'/T \rfloor - 1)T$ steps of the frame can be broken into T-frames. In each of these T-frames, at most RT packets traverse g. The remainder of the T'-frame τ consists of a single y-frame, where $T \leq y \leq 2T-1$, in which at most Ry packets traverse g. Q.E.D.

The following proposition is basically a re-statement of Lemma 3.6 of [27] and will be used here in place of this lemma. Proposition 3.6 applies when the number of distinct edges traversed by the packets in the schedule considered, m', is at most a polynomial in I (I as defined below).

Proposition 3.6 *Suppose that (i) there are positive constants $\alpha_1, \alpha_2, \alpha_3$, where $\alpha_1 \geq \alpha_2$, $\alpha_1 \geq 2\alpha_3$, and $\alpha_3 \geq \alpha_2$; (ii) I is larger than some sufficiently large constant; and (iii) for all edge g, in a schedule of length I^{α_1} (or smaller) the relative congestion of edge g in frames of size I^{α_2} or larger is at most ρ_g, where ρ_g is a constant. Let m' be the number of distinct edges traversed by the packets in this schedule. Furthermore, suppose that each packet is assigned a delay chosen randomly, independently, and uniformly from the range $[0, I^{\alpha_2}]$, and that if a packet is assigned a delay of x, then x delays are inserted in the first I^{α_3} steps and $I^{\alpha_2} - x$ delays are inserted in the last I^{α_3} steps of the schedule.*

1. *Then for any constant $\xi > 0$, there exists a constant $k_1 > 0$ such that with probability at least $1 - m'/I^{\xi}$ the relative congestion of any edge g in any frame of size $\log^2 I$ or larger, in between the first and last I^{α_3} steps of the new schedule is at most $\rho_g(1+\sigma)$, for $\sigma = k_1/\sqrt{\log I}$.*

2. *We can find such a schedule and verify whether it satisfies the conditions in 1. in $O(m' I^{\alpha_1}(\log^2 I))$ time.*

Proof: To bound the relative congestions of each edge in frames of size $\log^2 I$ or larger, we need to consider all m' edges and, by Lemma 3.5, all frames of size between $\log^2 I$ and $2\log^2 I - 1$.

As we shall see, the number of packets that traverse an edge g during a particular T-frame τ has a binomial distribution. In the new schedule, a packet can traverse g during τ only if in the original schedule it traversed g during τ or during one of the I^{α_2} steps before the start of τ. Since the relative congestion of edge g in any frame of size I^{α_2} or greater in the original schedule is at most ρ_g, there are at most $\rho_g(I^{\alpha_2} + T)$ such packets. The probability that an individual packet that could traverse g during τ actually does so is at most T/I^{α_2}. Thus, the probability p that ρ'_g or more packets traverse an edge g during a particular T-frame τ is at most

$$p \leq \sum_{k=\rho'_g}^{\rho_g(I^{\alpha_2}+T)} \binom{\rho_g(I^{\alpha_2}+T)}{k} \left(\frac{T}{I^{\alpha_2}}\right)^k \left(1-\frac{T}{I^{\alpha_2}}\right)^{\rho_g(I^{\alpha_2}+T)-k}.$$

To estimate the area under the tails of this binomial distribution, we use the following Chernoff-type bound [10]. Suppose that there are x independent Bernoulli trials, each of which is successful with probability p'. Let S denote the number of successes in the x trials, and let $\mu = E[S] = xp'$. Following Angluin and Valiant [2], we have

$$\Pr[S \geq (1+\gamma)\mu] \leq e^{-\gamma^2\mu/3}$$

for $0 \leq \gamma \leq 1$. (Note that e denotes the base of the natural logarithm.)

In our application, $x = \rho_g(I^{\alpha_2}+T)$, $p' = T/I^{\alpha_2}$, and $\mu = \rho_g(I^{\alpha_2}+T)T/I^{\alpha_2}$. For $\gamma = \sqrt{3k_0/\log I}$ (where k_0 is a positive constant to be specified later), $\rho_g \geq 1$, and $T \geq \log^2 I$, we have $\Pr[S \geq (1+\gamma)\mu] \leq e^{-k_0\rho_g(I^{\alpha_2}+T)T/(I^{\alpha_2}\log I)} \leq e^{-k_0 \log I} \leq e^{-k_0 \ln I} = 1/I^{k_0}$. Set $\rho'_g T$ to be $(1+\gamma)\mu = (1+\sqrt{3k_0/\log I})\rho_g(I^{\alpha_2}+T)T/I^{\alpha_2}$. For I large enough, $2\log^2 I/I^{\alpha_2} \leq 1/\sqrt{\log I}$, and thus $\rho'_g \leq \rho_g(1+k_1/\sqrt{\log I})$, for some constant $k_1 \geq 3k_0 + \sqrt{3k_0} + 1$. Let $\sigma = k_1/\sqrt{\log I}$. Then $\rho'_g \leq \rho_g(1+\sigma)$. Thus $p = \Pr[S \geq \rho'_g T] \leq \Pr[S \geq (1+\gamma)\mu] \leq 1/I^{k_0}$.

Since there are at most $(I^{\alpha_1}+I^{\alpha_2}) \leq 2I^{\alpha_1}$ starting points for a frame, and $\log^2 I$ different size frames starting at each point, and there are at most m' distinct edges per frame, the probability that the relative congestion of any edge g is more than ρ'_g in any frame is at most $m'I^{\alpha_1}\log^2 I/I^{k_0} \leq m'/I^{k_0-\alpha_1-2}$ (since $\log^2 I \leq I^2$). For any $\xi > 0$, we set $k_0 = \xi + \alpha_1 + 2$, which in turn sets k_1 and σ, completes the proof of part 1.

We assign a random delay to each packet, and verify whether the conditions in 1. apply as follows. We construct the schedule by routing all the packets one step at a time. At time t, for $1 \leq t \leq (I^{\alpha_1}+I^{\alpha_2})$, we compute the congestion of edge g in a T-frame ending at t, for all edges g that are traversed by some packet in the schedule, for all $T \in [\log^2 I, 2\log^2 I - 1]$ using the following rules: (i) if $T = \log^2 I$ then the congestion of g in a T-frame ending at time t can be computed by taking the congestion of g in the T-frame ending at $t-1$, subtracting the number of packets that traverse edge g at time $t-T$ and adding the number of packets that traverse g at time t; otherwise (ii) if $t \geq T$, then the congestion of edge g in a T-frame that ends at t is given by the congestion of g in a $(T-1)$-frame that ends at $t-1$ plus the number of packets that traverse edge g at time t. The relative congestion of an edge in a frame is given by the congestion of the edge in the frame divided by the size of the frame. This can be done in time $O(m'(I^{\alpha_1} + I^{\alpha_2}) \log^2 I) = O(m' I^{\alpha_1} \log^2 I)$, m' being the number of distinct edges traversed in this schedule. Q.E.D.

In the remainder of this chapter, for a schedule of size polynomial in I, we assume we check for the congestions of all T-frames, $\log^2 I \leq T < 2\log^2 I$, of the schedule as described in the proof of Proposition 3.6.

3.3 An algorithm for constructing optimal schedules

In this section, we describe the key ideas required to make the non-constructive proof of [27] constructive. There are many details in that proof, but changes are required only where the Lovász Local Lemma is used, in Lemmas 3.2, 3.7, and 3.9 of [27]. The non-constructive proof showed that a schedule can be modified by assigning delays to the packets in such a way that in the new schedule the relative congestion can be bounded in much smaller frames than in the old schedule. In this chapter, we show how to find the assignment of delays quickly. We will not regurgitate the entire proof of [27], but only reprove those lemmas, trying to state the replacement propositions in a way as close as possible to the original lemmas.

In Section 3.3.1, we provide a proposition, Proposition 3.2, that is a constructive version of Lemma 3.2 of [27]. In Sections 3.3.2 and 3.3.3, we provide three propositions that are meant to replace Lemma 3.7 of [27]. Lemma 3.7 is applied $O(\log^*(c+d))$ times in [27]. We will use Propositions 3.7.1 and 3.7.2 to replace the first two applications of Lemma 3.7. The remaining applications will be replaced by Proposition 3.7.3. In Section 3.3.4, we present the three replacement propositions for Lemma 3.9 of [27]. Our belief is that a reader who understands the structure of the proof in [27] and the propositions in this chapter can easily see how to make the original proof constructive. We analyze the running time of our algorithm in Section 3.4.

3.3.1 The first reduction in frame size

For a given set of N packets, let c and d denote the congestion and dilation of the paths taken by these packets, and let $\mathcal{P}$ denote the sum of the lengths of these paths. Note that $m \leq \mathcal{P} \leq mc$, and that $c, d < \mathcal{P}$, where m is the number of edges traversed by some packet in the network. The following proposition is meant to replace Lemma 3.2 of [27]. It is used just once in the proof, to reduce the frame size from d to $\log \mathcal{P}$.

Proposition 3.2 *For any constant $\beta > 0$, there is a constant $\alpha > 0$, such that there exists an algorithm that constructs a schedule of length $d + \alpha c$ in which packets never wait in edge queues and in which the relative congestion in any frame of size $\log \mathcal{P}$ or larger is at most 1. The algorithm runs in $O(m(c + d)(\log \mathcal{P}))$ time steps, and succeeds with probability at least $1 - 1/\mathcal{P}^{\beta}$.*

Proof: The algorithm is simple: Assign each packet an initial delay that is chosen randomly, independently, and uniformly from the range $[0, \alpha c]$, where α is a constant that will be specified later; the packet will wait out its initial delay and then travel to its destination without stopping. The length of the new schedule is at most $\alpha c + d$.

To bound the relative congestion in frames of size $\log \mathcal{P}$ or larger, we need to consider all m edges and, by Lemma 3.5, all frames of size between $\log \mathcal{P}$ and $2 \log \mathcal{P} - 1$. We assume without loss of generality that $c > 2 \log \mathcal{P}$, since any frame of length c or larger has relative congestion at most 1. For any particular edge g, and T-frame τ, where $\log \mathcal{P} \leq T \leq 2 \log \mathcal{P} - 1$, the probability p that more than T packets use g in τ is at most

$$\begin{aligned} p &\leq \sum_{i=T+1}^{c} \binom{c}{i} \left(\frac{T}{\alpha c}\right)^{i} \left(1 - \frac{T}{\alpha c}\right)^{c-i} \\ &\leq \binom{c}{T+1} \left(\frac{T}{\alpha c}\right)^{(T+1)} \\ &\leq \left(\frac{e}{\alpha}\right)^{(T+1)} \end{aligned}$$

since each of the at most c packets that pass through g has probability at most $T/\alpha c$ of using g in τ, and since $\binom{a}{b} \leq (ae/b)^b$, for any $0 < b \leq a$. The total number of frames to consider is at most $(\alpha c + d) \log \mathcal{P}$, since there are at most $\alpha c + d$ places for a frame to start, and $\log \mathcal{P}$ frame sizes. Thus the probability that the relative congestion of any edge is too large in any frame of size $\log \mathcal{P}$ or larger is at most

$$m(\log \mathcal{P})(\alpha c + d)\left(\frac{e}{\alpha}\right)^{\log \mathcal{P}}.$$

We bound the probability above by summing the probabilities that the relative congestion is too large for all $m \log \mathcal{P}$ edge-frame pairs. Using the inequalities $\mathcal{P} \geq c$, $\mathcal{P} \geq m$, and $\mathcal{P} \geq d$, we have that for any constant $\beta > 0$, there exists a constant $\alpha > 0$, such that this probability is at most $1/\mathcal{P}^{\beta}$.

Assigning the delays to the packets takes $O(N)$ time steps. Verifying whether the relative congestion is at most 1 in any T-frame of size $\log \mathcal{P} \leq T \leq 2\log \mathcal{P} - 1$ can be done in $O(m(c+d)(\log \mathcal{P}))$ time steps (see the last paragraph of the proof of Proposition 3.6).

Q.E.D.

Before applying Proposition 3.7.1, we first apply Proposition 3.2 to produce a schedule S_1 of length $O(c+d)$ in which the relative congestion in any frame of size $\log \mathcal{P}$ or larger is at most 1. For any positive constant β, this step succeeds with probability at least $1 - 1/\mathcal{P}^{\beta}$. If it fails, we simply try again.

3.3.2 A randomized algorithm to reduce the frame size

In this section, we prove two very similar propositions, Propositions 3.7.1 and 3.7.2, that are meant to replace the first two applications of Lemma 3.7 of [27], which we state below. In proving all these propositions, we use a constructive version of the Lovász Local Lemma that applies to scheduling problems. Let $I \geq 0$. We break a schedule S into *blocks* of $2I^3 + 2I^2 - I$ consecutive time steps. The *size* of a block is the number of time steps it spans.

Lemma 3.7 *[27] In a block of size $2I^3 + 2I^2 - I$, let the relative congestion in any frame of size I or greater be at most r, where $1 \leq r \leq I$. Then there is a way of assigning delays to the packets so that in between the first and the last I^2 steps of this block, the relative congestion in any frame of size $I_1 = \log^2 I$ or greater is at most $r_1 = r(1+\epsilon_1)$, where $\epsilon_1 = O(1)/\sqrt{\log I}$.*

After applying Proposition 3.2 to reduce the frame size from d to $\log \mathcal{P}$, Propositions 3.7.1 and 3.7.2 are used once on each block (for $I = \log \mathcal{P}$ and $I = (\log\log \mathcal{P})^2$ respectively) to further reduce the frame size. Unlike Lemma 3.7 of [27], Propositions 3.7.1 and 3.7.2 may increase the relative congestion by a constant factor. In general, we cannot afford to pay a constant factor at each of the $O(\log^*(c + d))$ applications of Lemma 3.7 of [27], but we can afford to pay it twice.

These two propositions avoid the use of exhaustive search, since they relate to problem sizes that are still large: In these propositions we designed separate algorithms that use the fact that the problem sizes are sufficiently large that we can guarantee a "good" solution with high probability. In the remainder of this chapter, we assume without loss of generality that $\mathcal{P}$ is not a constant. If $\mathcal{P} = O(1)$, then we can find an optimal schedule in a constant number time steps by using exhaustive search. For the application of Proposition 3.7.1, $I = \log \mathcal{P}$ and $r = 1$. With probability at least $1 - 1/\mathcal{P}^\beta$, for any constant $\beta > 0$, we succeed in producing a schedule S_2 in which the relative congestion is $O(1)$ in frames of size $\log^2 I = (\log\log \mathcal{P})^2$ or greater (if we should fail, we simply try again). In the application of Proposition 3.7.2, $I = (\log\log \mathcal{P})^2$, and $r = O(1)$. In the resulting schedule, S_3, the relative congestion is $O(1)$ in frames of size $\log^2((\log\log \mathcal{P})^2) = (\log\log\log \mathcal{P})^{O(1)}$ or greater, with probability at least $1 - 1/\mathcal{P}^\beta$, for any constant $\beta > 0$. At this point, the problem sizes are small enough for using exhaustive search, and we start using Proposition 3.7.3.

Proposition 3.7.1 *Let the relative congestion in any frame of size I or greater be at most r in a block of size $2I^3 + 2I^2 - I$, where $1 \leq r \leq I$ and $I = \log \mathcal{P}$. Let $\mathcal{Q}$ be the sum of the lengths of the paths taken by the packets in this block. Then, for any constant $\beta > 0$*

1. *there is an algorithm for assigning initial delays in the range $[0, I]$ to the packets so that in between the first and last I^2 steps of the block, the relative congestion in any frame of size $\log^2 I$ or greater is at most r', where $r' = 2r(1 + \sigma)$ and $\sigma = O(1)/\sqrt{\log I}$;*

2. *this algorithm runs in $O(\mathcal{Q}(\log \mathcal{P})^4(\log\log \mathcal{P}))$ time steps, with probability at least $1 - 1/\mathcal{P}^\beta$.*

Proof: We define the *bad event* for each edge g in the network and each T-frame τ, for $\log^2 I \leq T \leq 2\log^2 I - 1$, as the event that more than $r'T$ packets use g in τ. A particular bad event may or may not *occur* — i.e., may or may not be true — in a given schedule. If no bad event occurs, then by Lemma 3.5, the relative congestion in all frames of size $\log^2 I$ or greater will be at most r'. Since there are $\log^2 I$ different frame sizes and there are at most $(2I^3 + 2I^2 - I) + I = 2I^3 + 2I^2$ different frames of any particular size, the total number of bad events involving any one edge is at most $(2I^3 + 2I^2)\log^2 I < I^4$, for I greater than some large enough constant.

We now describe the algorithm for finding the assignment. In a first pass of assigning delays to the packets, we process the packets one at a time. To each packet, we assign a delay chosen randomly, independently, and uniformly from 0 to I. We then examine every event in which the packet participates.

A packet can use an edge g in a T-frame τ only if it traversed edge g in τ or in one of the I steps preceding τ in the original schedule. At most $r(T + I)$ packets use edge g in a frame of size $(I + T)$, since the relative congestion in this frame is at most r in the original schedule. Thus at most $r(T + I)$ packets can traverse edge g in the new schedule (after delays are assigned to the packets). We call these $r(I + T)$ packets the *candidate* packets to use edge g in τ. Let C_g be the number of candidate packets to use g in τ that have been assigned delays so far. We say that the event for an edge g and a T-frame τ is *critical* if more than $C_gT/I + kr(I+T)T/(I\sqrt{\log I})$ packets actually traverse edge g in τ, where k is a positive constant to be specified later. Intuitively, an event becomes critical if the number of packets assigned delays so far that traverse edge g in τ exceeds the expected number of such packets (C_gT/I) by an excess term $kr(I+T)T/(I\sqrt{\log I})$. Since $C_g \leq r(T+I)$, we allow an excess of at most $k/\sqrt{\log I}$ times the expected number of packets that would use edge g in the frame if *all* of the packets were assigned delays. Hence, the maximum final excess allowed does not depend on C_g. If a packet causes an event to become critical, then we set aside all of the other packets that could also use g during τ, but whose delays have not yet been assigned.

The main difference between our algorithm and Beck's [8] constructive version of the Lovász Local Lemma is that we never allow the number of packets passing through an edge in a T-frame to deviate from the expectation by more than a low order term. In particular, we do not allow this number to deviate by a constant factor times the expectation. In Beck's algorithm, the random variable associated with a bad event (in our case, the number of packets that traverse an edge in a T-frame) may deviate from the expectation by a constant factor times the expectation.

We will deal with the packets that have been set aside later. Let P denote the set of packets that have been assigned delays. As we shall see, after one pass of assigning random delays to the packets, the problem of scheduling the packets that have been set aside is broken into a collection of much smaller subproblems, with probability at least $1 - 1/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$. Once the size of a subproblem (given by the number of edges involved in the subproblem) gets small compared to the frame length, we can try assigning random delays to the packets that were set aside.

In this initial pass, we assign a random delay to each packet, and check whether the event for an edge g traversed by the packet in this block and T-frame τ becomes critical, for all edges g traversed by the packet in this block and for all frames of size T in $[\log^2 I, 2\log^2 I - 1]$, by following the same procedure as described in the last paragraph of the proof of Proposition 3.6. Here the schedule length after we insert the delays is $2(I^3 + I^2) = O(\log^3 \mathcal{P})$ (and so there are $O(\log^3 \mathcal{P})$ starting points for a T-frame τ) and there are $\log^2 I = (\log\log \mathcal{P})^2$ different frame sizes to consider. The sum of the lengths of the paths traversed by the packets in this block is $\mathcal{Q}$. Thus, a pass takes $O(\mathcal{Q}(\log \mathcal{P})^3(\log\log \mathcal{P})^2)$ time steps. If a pass fails to reduce the component size, we try again.

In order to proceed, we must introduce some notation. The *dependence graph*, G, is the graph in which there is a node for each bad event, and an edge between two nodes if the corresponding events share a packet. Let q denote the number of distinct edges traversed by the packets in this block. Let b denote the degree of G. Whether or not a bad event for an edge g and a time frame τ occurs depends solely on the assignment of delays to the packets that pass through g. Thus, the bad event for an edge g and a time frame τ, and the bad event for an edge g' and a time frame τ' are dependent only if g and g' share a packet. Since at most $r(2I^3 + 2I^2 - I) \leq rI^4$ (for I large enough) packets pass through g, each of these packets passes through at most $2I^3 + 2I^2 - I \leq I^4$ other edges g', and there are at most $(2I^3 + 2I^2)(\log^2 I) \leq I^4$ time frames τ' the dependence b is at most rI^{12}. For $r \leq I$, we have $b \leq I^{13}$. Since the packets use at most q different edges in the network, and for each edge there are at most I^4 bad events, the total number of nodes in G is at most qI^4. We say that a node in G is critical if the corresponding event is critical. We say that a node is *endangered* if its event shares a packet with an event that is critical. After each packet has been either assigned a delay or set aside, let G_1 denote the subgraph of G consisting of the critical and endangered nodes and the edges between them. If a node is not in G_1, then all of the packets that use the corresponding edge have already been assigned a delay, and the bad event represented by that node cannot occur, no matter how we assign delays to the packets not in P. Hence, from here on we need only consider the nodes in G_1.

Since different components are not connected by edges in G_1, no two components share a packet. Also, any two events that involve edges traversed by the same packet share an edge in G_1, and so are in the same connected component. Thus there exists a one-to-one correspondence between components of G_1 and disjoint sets in a partition of the packets not in P. Hence, we can assign the delays to the packets in each component separately.

In the following claim, we show that, with high probability, the size of the largest connected component U of G_1 is at most $I^{52} \log \mathcal{P}$, with high probability. Hence we have reduced the maximum possible size of a component from qI^4 in G to $I^{52} \log \mathcal{P}$ in G_1. Recall that the constant k is used to determine whether and event becomes critical.

Claim 1 *For any constant $\beta' > 0$, there exists a constant $k > 0$ such that the size of the largest connected component of G_1 is at most $I^{52} \log \mathcal{P}$ with probability at least $1 - 1/\mathcal{P}^{\beta'}$.*

Proof: The trick to bounding the size of a largest connected component U is to observe that the subgraph of critical nodes in U is connected in the cube, G_1^3, of the graph G_1, i.e., the graph in which there is an edge between two distinct nodes u and v if in G_1 there is a path of length at most 3 between u and v. In G_1^3, the critical nodes of U form a connected subgraph because any path u, e_1, e_2, e_3, v in G_1 that connects two critical or endangered nodes u and v by passing through three consecutive endangered nodes e_1, e_2, e_3 can be replaced by two paths u, e_1, e_2, w and w, e_2, e_3, v of length 3 that each pass through e_2's critical neighbor w. Let G_2 denote the subgraph of G_1^3 consisting only of the critical nodes and the edges between them. Note that the degree of G_2 is at most b^3, and if two critical nodes lie in the same connected component in G_2, then they also lie in the same connected component in G_1^3, and hence in G_1.

By a similar argument, any maximal independent set of nodes in a connected component of G_2 is connected in G_2^3. Note that if a set of nodes is independent in G_2, then it is also independent in G_1^3 and in G_1. The nodes in an independent set in G_1 do not share any packets, therefore the probabilities that each of these nodes becomes critical are independent. Let G_3 be the subgraph of G_2^3 induced by the nodes in a maximal independent set in G_2 (any such maximal independent set in G_2 will do). The nodes in G_3 form an independent set of critical nodes in G_1. The degree of G_3 is at most b^9.

Our goal now is to show that, for any constant $\beta' > 0$, there exists a constant $k > 0$ such that the number of nodes in any connected component W of G_3 is at most $\log \mathcal{P}$, with probability $1 - 1/\mathcal{P}^{\beta'}$. To begin, with every connected component W of G_3, we associate a spanning tree of W (any such tree will do). Note that, if W and W' are two distinct connected components of G_3, then the spanning trees associated with W and W' are disjoint.

Now let us enumerate the different trees of size ℓ in G_3. To begin, a node is chosen as the root. Since there are at most qI^4 nodes in G_3, there are at most qI^4 possible roots. Next, we construct the tree as we perform a depth-first traversal of it. Nodes of the tree are visited one at a time. At each node u in the tree, either a previously unvisited neighbor of u is chosen as the next node to be visited, or the parent of u is chosen to be visited (at the root, the only option is to visit a previously unvisited neighbor). Thus, at each node there are at most b^9 ways to choose the next node. Since each edge in the tree is traversed once in each direction, and there are $\ell - 1$ edges, the total number of different trees with any one root is at most $(b^9)^{2(\ell-1)} < b^{18\ell}$.

Any tree of size ℓ in G_3 corresponds to an independent set of size ℓ in G_1; moreover, it corresponds to an independent set of ℓ critical nodes in G_1. We can bound the probability that all of the nodes in any particular independent subset U of size ℓ in G_1 are critical as follows. Let p_C be the probability that more than $M = CT/I + kr(I + T)T/(I\sqrt{\log I})$

packets use edge g in τ after C candidate packets to use g in τ have been assigned delays. Then

$$p_C \leq \sum_{j=M+1}^{C} \binom{C}{j} \left(\frac{T}{I}\right)^j \left(1 - \frac{T}{I}\right)^{C-j}.$$

For a fixed deviation, $kr(I+T)T/(I\sqrt{\log I})$, that does not depend on C, the probability p_C of exceeding this deviation is maximized when C is maximized — i.e., when $C = r(I+T)$, implying $M = r(T+I)T(1+k/\sqrt{\log I})/I$. Thus, $p_C \leq p_{r(I+T)}$. Using the Chernoff-type bound as in the proof of Proposition 3.6, with $\mu = r(I+T)T/I$ and $\gamma = k/\sqrt{\log I}$, and $k_0 = k^2/3$, we have $p_C \leq p_{r(I+T)} = Pr[S \geq (1+\gamma)\mu] \leq e^{-\gamma^2\mu/3} = e^{-(k^2 r(I+T)T)/(3I\log I)} \leq e^{-(k_0 \log I)} \leq e^{-(k_0 \ln I)} = 1/I^{k_0}$, since $T \geq \log^2 I$ and $r > 1$. Thus the probability that the event for g and τ becomes critical after C candidate packets to use g in τ have been assigned delays is at most $1/I^{k_0}$. Since the nodes in U are independent in G_1, the corresponding events are also independent. Hence the probability that all of the nodes in the independent set are critical after all packets are assigned delays or put aside is at most $1/I^{k_0\ell}$. Thus the probability that there exists a tree of size ℓ in G_3 is at most $qI^4b^{18\ell}/I^{k_0\ell} \leq qI^{4-(k_0-234)\ell}$ (since there exists at most $qI^4b^{18\ell}$ different trees of size ℓ in G_3 and $b \leq I^{13}$). Since $q \leq \mathcal{P}$, we can make this probability less than $1/\mathcal{P}^{\beta'}$, for $\ell = \log \mathcal{P}$, and any fixed constant $\beta' > 0$, by choosing k to be a sufficiently large constant. Hence, with probability at least $1 - 1/\mathcal{P}^{\beta'}$, the size of the largest spanning tree in G_3 will be $\log \mathcal{P}$.

We can now bound the size of the largest connected component in G_1. Since (i) the largest connected component in G_3 has at most ℓ nodes, with probability at least $1 - 1/\mathcal{P}^{\beta'}$; (ii) each of these ℓ nodes may have b^3 neighbors in G_2; and (iii) each node in G_2 is either in G_3 or is a neighbor of a node in G_3, the largest connected component in G_2 contains at most $b^3\ell$ nodes, with probability at least $1 - 1/\mathcal{P}^{\beta'}$. As we argued before, the critical nodes in any connected component of G_1 are connected in G_2. Thus, the maximum number of critical nodes in any connected component of G_1 is at most $b^3\ell$. Since each of these nodes may have as many as b endangered neighbors (and each endangered neighbor is adjacent to a critical node), and since $\ell = \log \mathcal{P}$, the size of the largest connected component in G_1 is at most $b^4\ell \leq I^{52} \log \mathcal{P}$, with high probability. Q.E.D.

Since $I = \log \mathcal{P}$ in the scope of this lemma, the size of the largest connected component in G_1 is at most $(\log \mathcal{P})^{53}$, for k large enough, with probability at least $1 - 1/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$. We still have to find a schedule for the packets not in P. We now have a collection of independent subproblems to solve, one for each component in the dependence graph. We can use Proposition 3.6 to find the initial delays for these packets. Since each node in the dependence graph corresponds to an edge in the routing network, a component with x nodes in the dependence graph corresponds to at most x, and possibly fewer, edges in the routing network.

We apply Proposition 3.6 to each of the independent subproblems. In the original block, let r_g be the relative congestion of edge g with respect to the packets not in P in frames of size I or larger, and let r_g^H be the relative congestion of edge g with respect to the packets in subproblem H in frames of size I or larger ($r_g = \sum_H r_g^H$). Let q^H be the number of distinct edges associated with subproblem H, for all H. Note that $q^H \leq I^{52}\ell \leq I^{52}\log\mathcal{P} = I^{53}$, since $I = \log\mathcal{P}$. After applying Proposition 3.6 to a subproblem H, the relative congestion of any edge g with respect to the packets in H in frames of size $\log^2 I$ or larger, in between the first and last I^2 steps in the final schedule is at most $r_g^H(1 + k_1/\sqrt{\log I})$, for some constant $k_1 > 0$, with probability at least $1 - q^H/I^\xi \geq 1 - 1/(\log\mathcal{P})^{\xi-53}$, for any constant $\xi > 0$.

Since the routing subproblems are mutually independent and disjoint, if we apply Proposition 3.6 $\log\mathcal{P}/(\log\log\mathcal{P})$ times to each of the at most $N \leq \mathcal{P}$ subproblems (note that each packet appears in at most one subproblem), then for any constant $\xi > 53$, and $\mathcal{P}$ large enough, with probability at least $1 - N/(\log\mathcal{P})^{(\xi-53)\log\mathcal{P}/(\log\log\mathcal{P})} \geq 1 - 1/\mathcal{P}^{\xi-54}$, the relative congestion of edge g with respect to the packets not in P, in any frame of size $\log^2 I$ or greater is at most $\sum_{H\in G_1} r_g^H(1 + k_1/\sqrt{\log I}) = r_g(1 + k_1/\sqrt{\log I})$.

Applying Proposition 3.6 $\log\mathcal{P}/(\log\log\mathcal{P})$ times for each subproblem takes time $O(\sum_H q^H(I^3 + I^2)(\log^2 I)\log\mathcal{P}/\log\log\mathcal{P}) = O(\mathcal{Q}(\log^4\mathcal{P})(\log\log\mathcal{P}))$, since $I = \log\mathcal{P}$ and $\mathcal{Q} \geq \sum_H q^H$.

We now have schedules for the packets in P and for the packets not in P. Fix any edge g traversed by some packet in the block and a T-frame τ, where $T \in [\log^2 I, 2\log^2 I - 1]$. The total number of candidate packets in P to use edge g in τ after the delays have been assigned is given by C_g. The number of packets in P that traverse edge g in τ in the resulting schedule is at most $C_g T/I + kr(I+T)T/(I\sqrt{\log I}) \leq r(I+T)T(1 + k/\sqrt{\log I})/I \leq$

$rT(1 + (2k + 1)/\sqrt{\log I})$, since $(I + T)/I \leq (1 + 1/(2\sqrt{\log I}))$, for I large enough, and $C_g \leq r(I + T)$. Hence the relative congestion of edge g in τ with respect to the packets in P is at most $r(1 + (2k + 1)/\sqrt{\log I})$.

Now we consider the relative congestion of the packets not in P. With probability at least $1 - 1/\mathcal{P}^{\beta'} - 1/\mathcal{P}^{(\xi - 54)}$, for any positive constants β' and ξ, there exists a constant k_1 such that the number of packets not in P that traverse g in the new schedule is at most $r_g T(1 + k_1/\sqrt{\log I}) \leq rT(1 + k_1/\sqrt{\log I})$.

Combining the relative congestions for packets in P and not in P, we get that the relative congestion of edge G in τ is at most $2r(1 + \max(2k + 1, k_1)/\sqrt{\log I})$. Choose $\sigma = \max(2k + 1, k_1)/\sqrt{\log I}$. Choose β such that $1/\mathcal{P}^{\beta} \geq 1/\mathcal{P}^{\beta'} + 1/\mathcal{P}^{\xi - 54}$. Hence, for any constant $\beta > 0$, there exist constants k and $k_1 > 0$ such that the relative congestion of edge g in τ is at most $2r(1 + \sigma)$, for any edge g, for any T-frame τ, for any $T \in [\log^2 I, 2\log^2 I - 1]$, with probability at least $1 - 1/\mathcal{P}^{\beta}$.

The number of time steps taken by the algorithm just described is $O(\mathcal{Q}(\log\log\mathcal{P} + \log\mathcal{P})(\log^3\mathcal{P})(\log\log\mathcal{P})) = O(\mathcal{Q}(\log^4\mathcal{P})(\log\log\mathcal{P}))$. Q.E.D.

Proposition 3.7.2 *Let the relative congestion in any frame of size I or greater be at most r in a block of size $2I^3 + 2I^2 - I$, where $1 \leq r \leq I$ and $I = (\log\log\mathcal{P})^2$. Let $\mathcal{Q}$ be the sum of the lengths of the paths traversed by the packets in this block. Then, for any constant $\beta > 0$*

1. *there is an algorithm for assigning initial delays in the range $[0, I]$ to the packets so that in between the first and last I^2 steps of the block, the relative congestion in any frame of size $\log^2 I$ or greater is at most r', where $r' = 2r(1 + \sigma)$ and $\sigma = O(1)/\sqrt{\log I}$;*

2. *this algorithm runs in $\mathcal{Q}(\log\mathcal{P})(\log\log\mathcal{P})^6(\log\log\log\mathcal{P})^{O(1)}$ time steps, with probability at least $1 - 1/\mathcal{P}^{\beta}$.*

Proof: The first part of the proof of this proposition is identical to the part where we assign delays to the packets in P in the proof of the Proposition 3.7.1 (we let $I = (\log\log\mathcal{P})^2$ in that proof).

However, since $I = (\log\log\mathcal{P})^2$, we need to make an additional pass assigning delays to the packets in this proof, in order to reduce the component size in the dependence graph to a polynomial in I. From there, we proceed by applying Proposition 3.6 to each component separately, as we did in the proof of Proposition 3.7.1. In the first pass, we reduce the maximum component size in G_1 from qI^4 to $I^{52}\log\mathcal{P}$ (by taking $\ell = \log\mathcal{P}$, as in Proposition 3.7.1), with probability at least $1 - 1/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$. In the second pass, we reduce the component size from $I^{52}\log\mathcal{P}$ down to $I^{52}\log\log\mathcal{P} \leq I^{53}$, by taking $\ell = \log\log\mathcal{P}$, and noting that the number of edges in any connected component is at most $I^{52}\log\mathcal{P}$. For any component, this step will succeed with probability at least $1 - 1/(I^{52}\log\mathcal{P})^{\beta'}$, for any constant $\beta' > 0$. To make this probability as high as it was in the case $I = \log\mathcal{P}$, if a pass fails for any component, we simply try to reduce the component size again, up to $\log\mathcal{P}/(\log\log\mathcal{P})$ times. Then with probability at least $1 - 1/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$, we have reduced the component size to at most I^{53}. Since (i) for each packet assigned a delay in these two passes, we have to check whether the event for an edge g and a T-frame τ becomes critical, for all edges g traversed by the packet, for all T-frames τ, for $T \in [\log^2 I, 2\log^2 I - 1]$ (using a similar procedure to that in the last paragraph of Proposition 3.6), and since (ii) we repeat the second pass $O(\log\mathcal{P}/\log\log\mathcal{P})$ times, the two passes take $O(\mathcal{Q}(I^3 + I^2)(\log^2 I)(\log\mathcal{P})/(\log\log\mathcal{P})) \leq O(\mathcal{Q}(\log\log\mathcal{P})^6\log^2((\log\log\mathcal{P})^2)\log\mathcal{P}/(\log\log\mathcal{P})) \leq \mathcal{Q}(\log\mathcal{P})(\log\log\mathcal{P})^5(\log\log\log\mathcal{P})^{O(1)}$ time steps.

The second pass adds some packets to the set P. Let P_1 and P_2 denote the number of packets assigned delays in the first and second pass, respectively. Then the relative congestion due to these packets will be at most $[(P_1 + P_2)T/I + 2kr(I+T)T/(I\sqrt{\log I})]/T \leq r(I+T)/I + 2kr(I+T)/(I\sqrt{\log I}) \leq r[1 + T/I + 2k(I+T)/(I\sqrt{\log I})] \leq r(1 + (4k+1)/\sqrt{\log I})$, since $T < 2\log^2 I$ and $2\log^2 I/I \leq 1/\sqrt{\log I}$.

If the two passes fail to achieve the desired relative congestion, we try again.

Now we apply Proposition 3.6 up to $\log\mathcal{P}/(\log\log\log\mathcal{P})$ times, assigning delays to the packets not in P, verifying at the end of each application whether the schedule obtained has relative congestion $r(1 + k_1/\sqrt{\log I})$, for some constant k_1 to be specified later. Here we need to apply Proposition 3.6 up to $\log\mathcal{P}/(\log\log\log\mathcal{P})$ times to each resulting component (rather than $\log \mathcal{P}/(\log\log \mathcal{P})$ as in the proof of Proposition 3.7.1) since the component size now is $O(I^{53}) = (\log\log\mathcal{P})^{O(1)}$, and so our bound on the failure probability for each component is only $1/(\log\log\mathcal{P})^{O(1)}$ (since the bound given by Proposition 3.6 is at best polynomially small in I and $I = (\log\log\mathcal{P})^2$), rather than $1/(\log\mathcal{P})^{O(1)}$. The assignment of delays to the packets not in P takes at most $\mathcal{Q}(\log\log\mathcal{P})^6(\log\log\log\mathcal{P})^{O(1)}(\log\mathcal{P})$ time steps, since $\mathcal{Q}$ is an upper bound on the sum of the number of edges in each component. For any constant $\beta' > 0$, there exists a constant $k_1 > 0$ such that we obtain a feasible schedule for these packets with relative congestion $r(1 + k_1/\sqrt{\log I})$ with probability at least $1 - 1/\mathcal{P}^{\beta'}$.

We have schedules for the packets in P and for the packets not in P, with relative congestions $r(1 + (4k+1)/\sqrt{\log I})$ and $r(1 + k_1/\sqrt{\log I})$, respectively, with probability at least $1 - 2/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$. The two schedules can be found in at most $\mathcal{Q}(\log\mathcal{P})(\log\log\mathcal{P})^6(\log\log\log\mathcal{P})^{O(1)}$ time steps. When we merge the two schedules, the resulting relative congestion may be as large as the sum of the two relative congestions — that is, the resulting relative congestion may be as large as $2r(1+\max(4k+1, k_1)/\sqrt{\log I})$, with probability at least $1 - 1/\mathcal{P}^{\beta}$, for large enough constants k and $k_1 > 0$, for any fixed $\beta > 0$ (choose β' such that $1/\mathcal{P}^{\beta} > 2/\mathcal{P}^{\beta'}$). Let $\sigma = \max(4k+1, k_1)/\sqrt{\log I}$.

Q.E.D.

3.3.3 Applying exhaustive search

The remaining $O(\log^*(c+d))$ applications of Lemma 3.7 in [27] are replaced by applications of the following proposition, which uses the same technique as Propositions 3.7.1 and 3.7.2, except that instead of using Proposition 3.6 for each component of the subgraph induced by critical and endangered nodes in the dependence graph, it uses the Lovász Local Lemma and exhaustive search to find the settings of the delays for the packets. Proposition 3.7.3 does not allow a constant factor increase in the relative congestion of the refined schedule, which prevents a $2^{O(\log^*(c+d))}$ blowup in the final relative congestion.

Proposition 3.7.3 *Let the relative congestion in any frame of size I or greater be at most r in a block of size $2I^3 + 2I^2 - I$, where $1 \le r \le I$ and $I \le (\log\log\log\mathcal{P})^{O(1)}$. Let $\mathcal{Q}$ be* *the sum of the lengths of paths taken by the packets in this block. Then, for any constant* $\beta > 0$*,*

1. *there is an algorithm for assigning initial delays in the range* $[0, I]$ *to the packets so that the relative congestion in any frame of size* $\log^2 I$ *or greater in between the first and last* I^2 *steps in the resulting schedule is at most* r'*, where* $r' = r(1+\sigma)$ *and* $\sigma = O(1)/\sqrt{\log I}$*;*

2. *this algorithm runs in* $\mathcal{Q}(\log \mathcal{P})(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}$ *time steps, with probability at least* $1 - 1/\mathcal{P}^{\beta}$*.*

Proof: The proof uses the Lovász Local Lemma to show that an assignment of initial delays satisfying the conditions of the proposition exists.

We first assign delays to some packets by making three passes through the packets using the algorithm of Proposition 3.7.1 (for making the initial pass assigning delays to the packets in P) in each pass. Let $C_g^{(i)}$, $1 \le i \le 3$, be the number of candidate packets to use edge g in τ that were assigned delays in the ith pass. After the first pass, we have that (i) the number of packets assigned delays in this pass that use edge g in the new schedule is at most $C_g^{(1)}T/I + kr(I+T)/(I\sqrt{\log I})$, and (ii) with probability at least $1 - 1/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$, the size of the largest component in the dependency graph is $I^{52}\log \mathcal{P}$.

We need to make two more passes assigning delays to the packets, reducing the size of the largest connected component first to $I^{52}(\log\log \mathcal{P})$, and then to $I^{52}(\log\log\log \mathcal{P}) = (\log\log\log \mathcal{P})^{O(1)}$ (since $I \le (\log\log\log \mathcal{P})^{O(1)}$), by taking $\ell = \log\log \mathcal{P}$ in the second pass and $\ell = \log\log\log \mathcal{P}$ in the third pass. If we fail to reduce the component size as desired, the second pass is repeated up to $\log \mathcal{P}/(\log\log \mathcal{P})$ times and the third pass is repeated up to $\log \mathcal{P}/(\log\log\log \mathcal{P})$ times. The number of packets assigned delays in the second (resp., third) pass that traverse edge g in the new schedule is at most $C_g^{(2)}T/I + kr(I+T)/(I\sqrt{\log I})$ (resp., $C_g^{(3)}T/I + kr(I+T)/(I\sqrt{\log I})$). As before, k is chosen large enough so that the failure probability in each pass is at most $1/\mathcal{P}^{\beta'}$, for any constant $\beta' > 0$.

In each pass, we assign a random delay to each packet and check whether the event for any edge g traversed by this packet and any T-frame τ, where $T \in [\log^2 I, 2\log^2 I - 1]$, becomes critical, as we did in Propositions 3.7.1-2. Thus each pass takes time $O(\mathcal{Q}(I^3 + I^2)(\log^2 I)) = \mathcal{Q}(\log\log\log\mathcal{P})^{O(1)}(\log\log\log\log\mathcal{P})^{O(1)}$. For any constant $\beta > 0$, choose β' such that $1/\mathcal{P}^{\beta} > 3/\mathcal{P}^{\beta'}$. Hence, since we repeat the second and third passes up to $\log\mathcal{P}/(\log\log\mathcal{P})$ and $\log\mathcal{P}/(\log\log\log\mathcal{P})$, respectively, we succeed in reducing the component size to $(\log\log\log\mathcal{P})^{O(1)}$ in $\mathcal{Q}(\log\mathcal{P})(\log\log\log\mathcal{P})^{O(1)}(\log\log\log\log\mathcal{P})^{O(1)}$ time steps, with probability at least $1 - 1/\mathcal{P}^{\beta}$. Let P be the number of packets assigned delays in the three passes.

We now use the Lovász Local Lemma to show that there exists a way of completing the assignment of delays (i.e., to assign delays to the packets not in P) so that the relative congestion in frames of size $\log^2 I$ or greater in this block is at most $r(1 + O(1)/\sqrt{\log I})$. We associate a bad event with each edge and each time frame of size $\log^2 I$ through $2\log^2 I - 1$. The bad event for an edge g and a particular T-frame τ occurs when more than $M_g = (r(I+T) - P_g)T/I + kr(I+T)T/(I\sqrt{\log I})$ packets not in P use edge g in τ, where P_g is the number of packets in P that traverse edge g during τ after the delays have been assigned to the packets in P (note that there are at most $r(T+I) - P_g$ candidate packets not in P to use edge g in τ). As we argued in the proof of Proposition 3.7.1, the total number of bad events involving any one edge is at most I^4. We show that if each packet not in P is assigned a delay chosen randomly, independently, and uniformly from the range $[0, I]$, then with nonzero probability no bad event occurs. In order to apply the lemma, we must bound both the dependence of the bad events, and the probability that any bad event occurs. The dependence b is at most I^{13}, as argued before. For any edge g and T-frame τ that contains g, where $\log^2 I \le T \le (2\log^2 I) - 1$, the probability p_g that more than M_g packets not in P use g in τ, can be shown to be at most $1/I^{14}$, for sufficiently large k, using exactly the same Chernoff-bound argument that was used in Proposition 3.7.1. Thus, $4\max_{g\in G}\{p_g\}b \le 4/I < 1$ (for $I > 4$). Hence, since $\max_{g\in G}\{p_g\}$ is an upper bound on the probability of any bad event occurring, by the Lovász Local Lemma, there is some way of assigning delays to the packets not in P so that no bad event occurs.

Since at most $r(T+I)$ packets pass through the edge associated with any critical node, and there are at most $(I+1)$ choices for the delay assigned to each packet, the number of different possible assignments for any subproblem containing $(\log\log\log\mathcal{P})^{O(1)}$ critical nodes is at most $(I+1)^{r(I+T)(\log\log\log\mathcal{P})^{O(1)}} \le I^{4I^2(\log\log\log\mathcal{P})^{O(1)}}$ (since $r < I$ and $T <$

$2\log^2 I$). For $I < (\log\log\log \mathcal{P})^{O(1)}$ and $\mathcal{P}$ larger that some constant, this quantity is smaller than $(\log \mathcal{P})^{\gamma}$, for any fixed constant $\gamma > 0$. Hence, we need to try out at most $\log^{\gamma} \mathcal{P}$ possible delay assignments.

After assigning delays to all of the packets, the number of packets that use an edge g in any T-frame τ is at most

$$\sum_{i=1}^{3}\left(\frac{C_g^{(i)}T}{I}+\frac{kr(I+T)T}{I\sqrt{\log I}}\right)+\frac{(r(I+T)-P_g)T}{I}+\frac{kr(I+T)T}{I\sqrt{\log I}} \\ \leq \frac{r(I+T)T}{I}+\frac{4kr(I+T)T}{I\sqrt{\log I}}$$

with probability at least $1-1/\mathcal{P}^{\beta}$, since each packet is assigned a delay exactly once, and thus $r(I+T)-P_g+C_g^{(1)}+C_g^{(2)}+C_g^{(3)} \leq r(I+T)$. Thus the relative congestion in any T-frame, for $\log^2 I \leq T < 2\log^2 I$, is at most

$$\left(\frac{r(I+T)}{I}\right)(1+\frac{4k}{\sqrt{\log I}}) = r\left(1+\frac{T}{I}\right)(1+\frac{4k}{\sqrt{\log I}}) \\ = r\left(1+\frac{(8k+1)}{\sqrt{\log I}}\right) = r(1+\sigma),$$

by taking $\sigma = (8k+1)/(I\sqrt{\log I})$, since $2\log^2 I/I \leq 1/\sqrt{\log I}$, for I large enough.

We can bound the total number of time steps taken by the algorithm as follows. The first three passes take time $\mathcal{Q}(\log \mathcal{P})(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}$, with probability at least $1-1/\mathcal{P}^{\beta}$. After the third pass, we solve subproblems containing $(\log\log\log \mathcal{P})^{O(1)}$ critical nodes exhaustively. For each subproblem, for each of the at most $\log^{\gamma} \mathcal{P}$ possible assignment of delays to the packets in the subproblem, for each of the at most $(I^3+I^2)\log^2 I$ T-frames τ in the subproblem, $\log^2 I \leq T < 2\log^2 I$, and for every edge g in τ, we check whether more than M_g packets traverse g during τ (using the procedure described in the proof of Proposition 3.6). This takes time $O(\mathcal{Q}(I^3+I^2)(\log^2 I)(\log^{\gamma} \mathcal{P}))$, which is at most $\mathcal{Q}(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}(\log^{\gamma} \mathcal{P})$, for $\mathcal{P}$ large enough, for any fixed $\gamma > 0$ (since the sum of the number of distinct edges in each subproblem is at most $\mathcal{Q}$, and since $I = (\log\log\log \mathcal{P})^{O(1)}$). In particular, for $\gamma = 1$, this quantity is bounded by $\mathcal{Q}(\log \mathcal{P})(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}$. Hence the algorithm runs in $\mathcal{Q}(\log \mathcal{P})(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}$ time steps, with probability at least $1 - 1/\mathcal{P}^{\beta}$, for any constant $\beta > 0$.

Q.E.D.

3.3.4 Moving the block boundaries

Now we present the three replacement propositions for Lemma 3.9 of [27], which bounds the relative congestion after we move the block boundaries (as in [27]). The three propositions that follow are analogous to the three replacement propositions, Propositions 3.7.1-3, for Lemma 3.7 of [27]. The necessary changes in the proof of Lemma 3.9 of [27], in places where the Lovász Local Lemma is used, are analogous to the changes made in the proof of Lemma 3.7 of [27], for the cases $I = \log \mathcal{P}$, $I = (\log\log \mathcal{P})^2$, and $I = (\log\log\log \mathcal{P})^{O(1)}$. Therefore, we omit the proofs of Propositions 3.9.1-3.

Suppose we have a block of size $2I^3 + 3I^2$, obtained after the insertion of delays into the schedule as described in Propositions 3.7.1, 3.7.2, or 3.7.3, according to the current value of I. Then suppose we move the block boundaries as described in [27]. Each Proposition 3.9.1-3 also refers to a specific size of I. Note that in [27], the steps between steps I^3 and $I^3 + 2I^2$ in the block are called the "fuzzy region" of the block. We assume that the relative congestion in any frame of size I or greater in the block is at most r, where $1 \le r \le I$. Let $\mathcal{Q}$ be the sum of the lengths of the paths taken by the packets in the block.

Proposition 3.9.1 *For $I = \log \mathcal{P}$, for any constant $\beta > 0$*

1. *there is an algorithm for assigning delays in the range $[0, I^2]$ to the packets such that in between steps $I \log^3 I$ and I^3 and in between steps I^3+3I^2 and $2I^3+3I^2-I\log^3 I$, the relative congestion in any frame of size $\log^2 I$ or greater is at most $2r(1+\sigma_1)$, where $\sigma_1 = O(1)/\sqrt{\log I}$, and such that in between steps I^3 and I^3+3I^2, the relative congestion in any frame of size $\log^2 I$ or greater is at most $2r(1+\sigma_2)$, where $\sigma_2 = O(1)/\sqrt{\log I}$;*

2. *this algorithm runs in $O(\mathcal{Q}(\log \mathcal{P})^4(\log\log \mathcal{P}))$ time steps, with probability at least $1 - 1/\mathcal{P}^{\beta}$.*

Proposition 3.9.2 *For $I = (\log\log\mathcal{P})^2$, for any constant $\beta > 0$,*

1. *there is an algorithm for assigning delays in the range $[0, I^2]$ to the packets such that in between steps $I\log^3 I$ and I^3 and in between steps I^3+3I^2 and $2I^3+3I^2-I\log^3 I$, the relative congestion in any frame of size $\log^2 I$ or greater is at most $2r(1+\sigma_1)$, where $\sigma_1 = O(1)/\sqrt{\log I}$, and such that in between steps I^3 and I^3+3I^2, the relative congestion in any frame of size $\log^2 I$ or greater is at most $2r(1+\sigma_2)$, where $\sigma_2 = O(1)/\sqrt{\log I}$;*
2. *this algorithm runs in $\mathcal{Q}(\log\mathcal{P})(\log\log\mathcal{P})^6(\log\log\log\mathcal{P})^{O(1)}$ time steps, with probability at least $1-1/\mathcal{P}^\beta$.*

Proposition 3.9.3 *For $I = (\log\log\log\mathcal{P})^{O(1)}$, for any constant $\beta > 0$,*

1. *there is an algorithm for assigning delays in the range $[0, I^2]$ to the packets such that in between steps $I\log^3 I$ and I^3 and in between steps I^3+3I^2 and $2I^3+3I^2-I\log^3 I$, the relative congestion in any frame of size $\log^2 I$ or greater is at most $r(1+\sigma_1)$, where $\sigma_1 = O(1)/\sqrt{\log I}$, and such that in between steps I^3 and I^3+3I^2, the relative congestion in any frame of size $\log^2 I$ or greater is at most $r(1+\sigma_2)$, where $\sigma_2 = O(1)/\sqrt{\log I}$;*
2. *this algorithm runs in $\mathcal{Q}(\log\mathcal{P})(\log\log\log\mathcal{P})^{O(1)}(\log\log\log\log\mathcal{P})^{O(1)}$ time steps, with probability at least $1-1/\mathcal{P}^\beta$.*

3.4 Running time

Theorem 1 *For any constant $\delta > 0$, the algorithm for finding an $O(c+d)$-steps schedule of the packets takes $O(m(c+d)(log\mathcal{P})^4(\log\log\mathcal{P}))$ time steps overall, with probability at least $1-1/\mathcal{P}^\delta$.*

Proof: For any constant $\beta > 0$, we place an upper bound on the number of time steps taken by the application of Proposition 3.2, followed by the applications of Propositions 3.7.1, 3.9.1, 3.7.2, and 3.9.2, then followed by the applications of Propositions 3.7.3 and 3.9.3.

The application of Proposition 3.2 takes $O(m(c+d)\log \mathcal{P})$ time steps, with probability at least $1-1/\mathcal{P}^{\beta}$. Each of the Propositions 3.7.1-3, and each of the Propositions 3.9.1-3 dealt with a single block. For any I, partitioning the schedule into *disjoint* blocks and moving the block boundaries as described in [27] take $O(\mathcal{P})$ time. Let n_I be the number of blocks in the schedule for any given I.

We place an upper bound on the number of time steps taken by the applications of Propositions 3.7.1-3 and 3.9.1-3 as follows. Assume the blocks are numbered from 1 to n_I. Note that $\sum_{i=1}^{n_I} \mathcal{Q}_i = \mathcal{P}$, where $\mathcal{Q}_i$ is the sum of the lengths of the paths traversed by the packets in block i. Thus the applications of Proposition 3.7.1 and 3.9.1 take $O(\mathcal{P}(\log \mathcal{P})^4(\log\log \mathcal{P}))$ steps; and the applications of Propositions 3.7.2 and 3.9.2 take $\mathcal{P}(\log \mathcal{P})(\log\log \mathcal{P})^6(\log\log\log \mathcal{P})^{O(1)}$ steps. For each partition of the schedule for a given $I \leq (\log\log\log \mathcal{P})^{O(1)}$, we apply Propositions 3.7.3 and 3.9.3 to every block i in this partition, $1 \leq i \leq n_I$, taking overall time $\mathcal{P}(\log \mathcal{P})(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}$. Since we will repartition the schedule $O(\log^*(c+d))$ times after we bring I down to $(\log\log\log \mathcal{P})^{O(1)}$, the overall running time due to applications of Propositions 3.7.3 and 3.9.3 is $\mathcal{P}(\log \mathcal{P})(\log\log\log \mathcal{P})^{O(1)}(\log\log\log\log \mathcal{P})^{O(1)}\log^*(c+d)$.

Choose $\delta > 0$ such that $1/\mathcal{P}^{\delta} \geq O(\log^*(c+d))/\mathcal{P}^{\beta}$. Thus the total number of time steps taken by the algorithm is $O(m(c+d)(\log \mathcal{P})^4(\log\log \mathcal{P}))$, for $\mathcal{P}$ large enough, with probability at least $1-1/\mathcal{P}^{\delta}$, for any constant $\delta > 0$. Note that we used the inequalities $\mathcal{P} \geq c$, $\mathcal{P} \geq d$, and $\mathcal{P} \leq m(c+d)$. Q.E.D.

3.5 A parallel scheduling algorithm

At first glance, it seems as though the algorithm that was described in Section 3.3 is inherently sequential. This is because the decision concerning whether or not to assign a delay to a packet is made sequentially. In particular, a packet is deferred (i.e., not assigned a delay) if and only if the packet might be involved in an event — i.e., the packet traverses an edge that corresponds to an event — that became critical because of the delays assigned to prior packets.

In [1], Alon describes a parallel version of Beck's algorithm which proceeds by assigning values to all random variables (in this case delays to all packets) in parallel, and then unassigning values to those variables that are involved in bad events. The Alon approach does not work in this application because we cannot afford the constant factor blow-up in relative congestion that would result from this process.

Rather, we develop an alternative method for parallelizing the algorithm. The key idea is to process the packets in a *random* order. At each step, all packets that do not share an edge with an as-yet-unprocessed packet of higher priority are processed in parallel.

To analyze the parallel running time of this algorithm, we first make a dependency graph G' with a node for every packet and an edge between two nodes if the corresponding packets can be involved in the same event. Each edge is directed towards the node corresponding to the packet of lesser priority. By Brent's Theorem [9], the parallel running time of the algorithm is then at most twice the length of the longest directed path in G'.

Let D denote the maximum degree of G'. There are at most ND^L paths of length L in G'. The probability that any particular path of length L has all of its edges directed in the same way is at most $2/L!$ (the factor of 2 appears because there are two possible orientations for the edges). Hence, with probability near 1, the longest directed path length in G' is $O(D + \log N)$. This is because if $L \geq k(D + \log N)$, for some large constant k, then $ND^L \cdot \frac{2}{L!} \ll 1$.

Each packet can be involved in at most $(2I^3 + 2I^2)(2I^3 + I^2)\log^2 I$ events, and at most $r(I + T) \leq O(I)$ packets can be involved in the same event. Hence, the degree D of G' is at most $O(I^7 \log^2 I)$. By using the method of Proposition 3.2 as a preprocessing phase, we can assume that c, d, and thus I, are all polylogarithmic in $\mathcal{P}$. Hence, the parallel algorithm runs in NC, as claimed.

3.6 Concluding remarks

Our algorithm for packet scheduling can also be used to route messages that are composed of sequences of packets. This is possible since our algorithm can easily maintain the property that any two packets traveling along the same path to the same destination always proceed in order.

The algorithms described in this chapter are randomized, but they can be derandomized using the method of conditional probabilities [43, 52].

Chapter 4

New Approximation Techniques for Some Ordering Problems

4.1 Introduction

In this chapter, we address the *minimum linear arrangement problem*, an optimization problem that arises in embeddings of networks into a linear array. Let G be a network with associated nonnegative edge weights. The weight of an edge can represent the capacity, or the cost of communication through the edge. Informally, a minimum linear arrangement of G is an embedding of G into the linear array such that (i) we have a one-to-one mapping from the nodes of G to the nodes of the linear array, and (ii) the weighted sum of the edge dilations — that is, the *cost* of the linear arrangement — is minimum. In Figure 4.1, we show a linear arrangement σ for the network G with cost 28 (in fact this linear arrangement is a minimum linear arrangement of G).

As we saw in Chapter 1, a guest network G can be emulated by a host network H by embedding G into H. The *slowdown* of an emulation is given by the ratio between the number t' of steps on H needed to emulate any t steps of computation on G. We would like the slowdown to be as small as possible. The slowdown of an emulation is closely related to the dilations of the edges in the associated embedding: The dilation of an edge

This is joint work with Satish Rao, NEC Research Institute; a preliminary version of this work appears in [47].

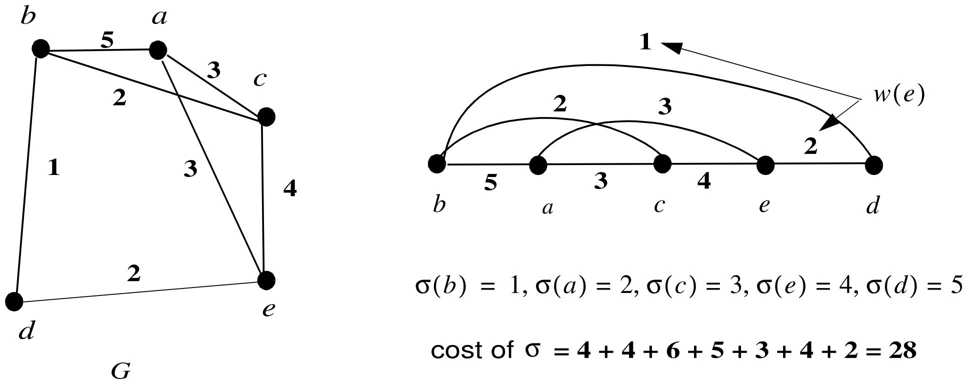


Figure 4.1: A graph G and a minimum linear arrangement σ of G.

introduces an extra factor in the cost of communication between the endpoint nodes of this edge in the emulation.

Note that, as in the problem addressed in Chapter 2, the idea of preserving locality in order to minimize the use of shared resources in the network also arises in the minimum linear arrangement problem — i.e., we would like that nodes that are "close" in the network G also be "close" in the linear array, in order to minimize the average edge dilation.

Finding a minimum linear arrangement is NP-hard, even for the case when all the edges have unit weight. An *α-approximation algorithm* is an algorithm that finds a solution to the problem whose cost is at most α times the cost of an optimal solution to the problem. In this chapter, we present a polynomial-time $O(\log n)$-approximation algorithm for finding a minimum linear arrangement of an n-node network, improving on the best previous approximation bound of Even, Naor, Rao, and Schieber [13] for this problem by a $\Theta(\log \log n)$ factor.

If the network is planar (or, more generally, if it excludes $K_{r,r}$ as a minor, for fixed r, where $K_{r,r}$ is the $r \times r$ complete bipartite graph), we achieve an $O(\log \log n)$-approximation factor for the minimum linear arrangement problem, using a variation of the algorithm presented for the general case. We obtain this improvement by combining the techniques used for the general case with the algorithm presented by Klein, Plotkin, and Rao [23] for finding separators in graphs that exclude fixed $K_{r,r}$-minors, as presented in Section 4.5.

We extend our approximation techniques (and bounds) to two other problems that involve finding a linear ordering of the nodes of a graph: the minimum containing interval graph, and the minimum storage–time product problems. Using techniques from [48], we can view the minimum containing interval graph problem (which we define formally in Section 4.7) as a "vertex version" of the minimum linear arrangement problem (see [13]). Thus, we also obtain an $O(\log n)$-approximation algorithm for general graphs, and an $O(\log \log n)$-approximation algorithm for graphs that exclude fixed $K_{r,r}$-minors for this problem. This improves on the best known previous approximation bounds for this problem of $O(\log n \log \log n)$ for general graphs [13], and of $O(\log n)$ for graphs that exclude fixed $K_{r,r}$-minors.

We can also use techniques from [48] to extend our ideas to produce an $O(\log T)$-approximation algorithm for the minimum storage–time product problem (defined in Section 4.6), improving on a previous approximation bound of $O(\log T \log \log T)$ [13], where T is the sum of the processing times of all tasks. The minimum storage–time product problem also generalizes the minimum linear arrangement problem. The techniques in [23] do not apply to directed graphs; therefore, the approach used in the two former problems that led to better approximation bounds for graphs that exclude $K_{r,r}$-minors does not apply to the minimum storage–time product problem.

Our approximation techniques rely on a lower bound W on the cost of an optimal solution provided by a *spreading metric* (to be defined soon), for each of the problems considered: We find a solution to the problem that has cost $O(W \log n)$ ($O(W \log T)$ for the minimum storage–time product problem). Alon and Seymour [50] showed that there exists a logarithmic gap between the lower bound provided by any spreading metric, and the true optimal values for certain instances of the problems of minimum linear arrangement, minimum containing interval graph, and minimum storage–time product. Thus we provide an existentially tight bound on the relationship between the lower bound provided by spreading metrics and the true optimal values for these problems.

4.1.1 Previous work

Leighton and Rao [29] presented an $O(\log n)$-approximation algorithm for balanced partitions of graphs. Among other applications, this provided $O(\log^2 n)$-approximation algorithms for the minimum feedback arc set, and for the minimum-cut linear arrangement problem. Hansen [19] used the ideas in [29] to present $O(\log^2 n)$-approximation algorithms for the minimum linear arrangement problem, and for the more general problem of graph embeddings in d-dimensional meshes. Ravi, Agrawal, and Klein [48] presented polynomial-time approximation algorithms that deliver a solution with cost within an $O(\log n \log T)$ factor from optimal for the minimum storage–time product problem, where T is the sum of the processing times of all tasks, and within an $O(\log^2 n)$ factor from optimal for the minimum containing interval graph.

Seymour [50] was the first to present a directed graph decomposition divide-and-conquer approach that does not rely on balanced cuts. He presented a polynomial-time $O(\log n \log\log n)$-approximation algorithm for the minimum feedback arc set problem. Even, Naor, Rao, and Schieber [13] extended the spreading metric approach used by Seymour to obtain polynomial-time $O(\log n \log\log n)$-approximation algorithms for the minimum linear arrangement, and the minimum containing interval graph problems, and an $O(\log T \log\log T)$-approximation algorithm minimum storage–time product problem. Even et al. actually showed similar approximation results for a broader class of graph optimization problems, namely for the problems that satisfy their "approximation paradigm": A graph optimization problem satisfies this paradigm if (i) the divide-and-conquer approach presented by Even et al. is applicable to the problem; and (ii) a spreading metric that provides a lower bound on the cost of an optimal solution to the problem can be computed in polynomial time. They defined spreading metrics that led to polynomial-time algorithms for these problems with an $O(\min\{\log W \log\log W, \log k \log\log k\})$ approximation bound, where k denotes the number of "interesting" nodes in the problem instance (clearly $k \leq n$), and W is the lower bound on the cost of a solution to the optimization problem provided by a spreading metric. Examples of such problems, besides the ones already mentioned, are graph embeddings in d-dimensional meshes, symmetric multicuts in directed networks, k-multiway separators and ρ-separators (for small values of ρ) in directed graphs. For a detailed description of each of those problems, see [13].

Even, Naor, Rao, and Schieber [14] also extended the spreading metric techniques to graph partitioning problems. They used simpler recursions that yield a logarithmic approximation factor for balanced cuts and multiway separators. However, they were not able to extend this simpler technique to obtain a logarithmic approximation bound for the other problems considered in [13].

4.1.2 Spreading metrics and our recursion

Our algorithms use an approach that relies on *spreading metrics*. Spreading metrics have been used in recent divide-and-conquer techniques to obtain improved approximation algorithms for several graph optimization problems that are NP-hard [13, 50]. These techniques perform the divide step according to the *cost* of a solution to the subproblems generated, rather than according to the *size* of such subproblems.

A spreading metric on a graph is an assignment of lengths to the edges or nodes of the graph that has the property of "spreading apart" (with respect to the metric lengths) all the nontrivial connected subgraphs. The *volume* of the spreading metric is the sum, taken over all edges (resp., nodes), of the length of each edge (resp., node) multiplied by its weight.

For each of the optimization problems we consider here, Even, Naor, Rao, and Schieber [13] defined a spreading metric of volume W such that W is a lower bound on the cost of a solution to the problem. Our techniques are based on showing that a spreading metric of volume W can be used to find a solution with cost $O(W \log n)$ ($O(W \log T)$, for the minimum storage–time product problem).

We develop a recursion where at each level we identify cost which, if incurred, yields subproblems with reduced spreading metric volume. Specifically, we present a divide-and-conquer strategy where the cost of a solution to a problem at a recursive level is C plus the cost of a solution to the subproblems, *and* where the spreading metric volume on the subproblems is less than the original volume by $\Omega(C/\log n)$ (resp., $\Omega(C/\log T)$ for the minimum storage–time product problem). We will show that this ensures that the resulting solution has cost $O(\log n)$ (resp., $O(\log T)$) times the original spreading metric volume.

The recursion is based on divide-and-conquer — that is, we find an edge set whose removal divides the graphs into subgraphs, and then recursively order the subgraphs. The cost of a recursive level is the cost associated with the edges in the cut selected at this level. Previous recursive methods and analyses proceeded by finding a small cutset of edges where the maximum spreading metric volumes of the subproblems were quickly reduced. We proceed by finding a *sequence* of cutsets whose total cost can be upper bounded, say by a quantity C, and whose total spreading metric volume is $\Omega(C/\log n)$ ($\Omega(C/\log T)$ for the minimum storage–time product problem), as stated above. The crux of the argument is that the cost associated with an edge in a cutset can be bounded by the number of nodes between the previous and the next cutset in the sequence.

We point out that the methods in [13] applied to more problems, including the d-dimensional graph embedding and the minimum feedback arc set problems [50]. We could not extend our methods to these other problems, since we were unable to find a suitable bound on the cost of a sequence of cutsets associated with any of these problems.

Finally, for planar graphs and other graphs that exclude some fixed minors, we combine a structural theorem of Klein, Plotkin, and Rao [23] with our new recursion techniques, to show that the spreading metric cost volumes are within an $O(\log \log n)$ factor of the cost of the optimal solution for the minimum linear arrangement and the minimum containing interval graph problems.

4.1.3 Overview

We present a formal definition of the minimum linear arrangement problem in Section 4.2. In Section 4.3, we define the spreading metric used for this problem; in Section 4.4, we present a polynomial-time $O(\log n)$-approximation algorithm for the minimum linear arrangement problem on an arbitrary graph with n nodes and nonnegative edge weights. In Section 4.5, we show how to improve this approximation factor to $O(\log \log n)$, in case the graph has no fixed $K_{r,r}$-minors — e.g., the graph is planar. In Sections 4.6 and 4.7, we define and briefly discuss the algorithms for approximating the minimum storage–time product problem and minimum containing interval graph problem respectively.

4.2 The problem

The *minimum linear arrangement (MLA) problem* is defined as follows: Given an undirected graph $G(V, E)$, with n nodes, and nonnegative edge weights $w(e)$, for all e in E, we would like to find a linear arrangement of the nodes $\sigma{:}V \to \{1, \ldots, n\}$ that minimizes the sum, over all (i, j) in E, of the weighted edge lengths $|\sigma(i) - \sigma(j)|$. In other words, we would like to minimize the *cost*

$$\sum_{(i,j)\in E} w(i, j)\, |\sigma(i) - \sigma(j)|.$$

of a linear arrangement σ. In the context of VLSI layout, $|\sigma(i) - \sigma(j)|$ represents the length of the interconnection between i and j.

4.3 Spreading metric

In this section, we define the spreading metric used in the algorithms for the MLA problem presented in Sections 4.4 and 4.5. Analogous functions are used when approximating the minimum storage–time product problem (as presented in Section 4.6), and the minimum containing interval graph problem (see Section 4.7).

Here we present spreading metrics in the context of the MLA problem (see [13] for a more general definition). A *spreading metric for the MLA problem* is a function $\ell : E \to \mathbf{Q}$ that assigns rational lengths to every edge in E, and that can be computed in polynomial time. It also satisfies the two properties below. The *volume* of a spreading metric ℓ is given by $\sum_{e\in E} w(e)\ell(e)$.

1. *Diameter guarantee:* Let the distances be measured with respect to the lengths $\ell(e)$. The distances induced by the spreading metric "spread" the graph and all its nontrivial subgraphs. In this application, this translates to "The diameter of every nontrivial connected subgraph U of V is $\Omega(|U|)$".

2. *Lower bound:* The minimum volume of a spreading metric is a lower bound on the cost of a MLA of G.

A solution ℓ to (4.1– 4.2) is a spreading metric for the MLA (see [13]). Let $\mathcal{V}$ denote the set of all nontrivial connected subgraphs of V.

$$\frac{(\sum_{u \in U} \mathsf{dist}(u,v))}{|U|} \geq \frac{|U|}{4},\ \forall U \in \mathcal{V}, \forall v \in U \tag{4.1}$$

$$\ell(e) \geq 0,\ \forall e \in E \tag{4.2}$$

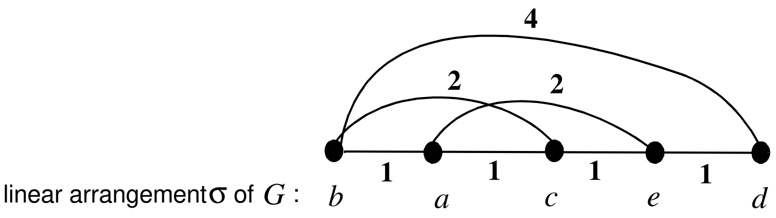
where $\mathsf{dist}(u,v)$ is the length of a shortest path from u to v according to the lengths $\ell(e)$. The metric ℓ can be computed in polynomial time (see [13]) using, e.g., the ellipsoid method (There may be an exponential number of constraints in (4.1)). Note that (4.1) actually implies that $\ell(e) \geq 1$, for all e in E (simply consider the subsets U that consist of a single edge and its endpoints).

A solution ℓ^* to (4.1– 4.2) that minimizes $\sum_{e \in E} w(e)\ell(e)$ is a *lower bound* on the cost of a MLA, since for any linear ordering σ of the nodes of G, the assignment of lengths to the edges of G given by $\ell(i,j) = |\sigma(i) - \sigma(j)|$ satisfies (4.1– 4.2). The volume of such an assignment is exactly the cost of σ. In particular this is true for a MLA σ. Hence $W^* = \sum_{e \in E} w(e)\ell^*(e)$ is less than or equal to the cost of a MLA (Note that this lower bound is existentially tight, since there exist instances of this problem such that $\ell^*(i,j) = |\sigma(i) - \sigma(j)|$, where σ is a MLA of G, as for example, when G is a linear array.). We will use this fact later, when proving Theorems 5 and 6. Figure 4.2 illustrates such an assignment of lengths for the linear arrangement σ given by the ordering of the nodes of G from left to right in this figure (the lengths $\ell(i,j)$ are the numbers associated with the edges in that picture; without loss of generality, assume that all the edge weights are 1).

Let ℓ be a spreading metric of volume $W = \sum_{e \in E} w(e)\ell(e)$ that satisfies (4.1– 4.2). In the remainder of this chapter, all the distances in G are measured with respect to ℓ.

4.4 The algorithm

We now present our $O(\log n)$-approximation algorithm for the MLA problem on general graphs. Let $G(V,E)$ be a graph with nonnegative edge weights $w(e)$. Assume without



$$W = \sum l(i,j) = 12$$

Figure 4.2: An assignment of lengths to the edges of G.

loss of generality that G is connected (otherwise consider each connected component of G separately), and that all the edge weights $w(e)$ are at least 1.

In this paragraph, we introduce the notion of a level according to ℓ. Fix a node v in V. An edge (x, y) is at *level* i with respect to v if and only if $\mathsf{dist}(v, x) \leq i$ and $\mathsf{dist}(v, y) > i$, for any $i \in \mathbf{N}$. Note that an edge may be at more than one level, and that there may be edges that are not at any level. Let the *weight of level* i, denoted by ρ_i, be the sum of the weights of the edges at level i. Without loss of generality, we will assume that $\log W$ is an integer. Let $\alpha_k = 2^k$, for all k in $[(\log W) + 1]$. Level i has *index* k, k in $[\log W]$, if and only if ρ_i belongs to the interval $I_k = (\alpha_k, \alpha_{k+1}]$.

It follows from (4.1) that there exists a node u such that $\mathsf{dist}(v, u) \geq n/4$. Hence we have at least $n/4$ distinct levels with nonzero weight. Note that since $w(e) \geq 1$ and $\ell(e) \geq 1$, for all e, any level with nonzero weight must have weight at least 1. Since there are $\log W$ distinct level indices, there are at least $n/(4 \log W)$ levels with same index k, for some k. Let κ be the exact number of levels of index k. Figure 4.3 illustrates the algorithm and charging scheme described below.

In a recursive step of the algorithm, we cut along the sequence of κ levels of index k — i.e., we remove all the edges that are at at least one of those levels, even if they also are at some other level of index different from k. For all i, let level a_i be the ith level of index k, in increasing order of distances to v. Let H_i be the subgraph induced by the nodes that are at distance greater than a_i and less than or equal to a_{i+1} from v; let H_0 (resp., H_κ) be the subgraph induced by the nodes that are at distance less than or equal to a_1 (resp.,

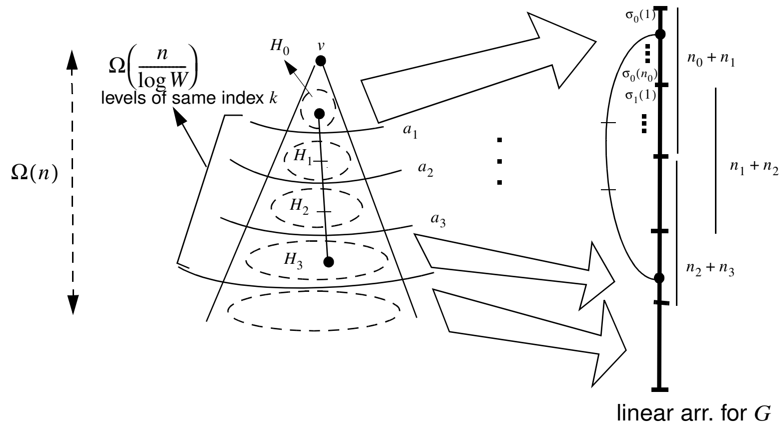


Figure 4.3: The algorithm and charging scheme.

greater than a_κ) from v. Let n_i denote the number of nodes in H_i. We recurse on each H_i, obtaining a linear arrangement σ_i for the n_i nodes in this subgraph. We combine the linear arrangements obtained for the H_i's, obtaining a linear arrangement σ for G, as follows:

$$(\sigma(1), \ldots, \sigma(n)) = (\sigma_0(1), \ldots, \sigma_0(n_0), \ldots, \sigma_\kappa(1), \ldots, \sigma_\kappa(n_\kappa))$$

Each recursive step runs in polynomial time; at each recursive step, we decompose a connected component into at least two connected components. Hence the algorithm runs in polynomial time.

We use a *charging scheme* to account for the length of an edge e in the linear arrangement for G obtained by our algorithm (note that we account for the *length of the edge in the linear arrangement*, rather than for the spreading metric length of the edge). If some edge e in level a_i belongs to some other level of index k, say level a_j, then this edge also belongs to every level of index k between a_i and a_j. Without loss of generality, assume that $i < j$. Edge e will be "stretched over" all the nodes in $H_i \cup \ldots \cup H_{j-1}$, and possibly over some of the nodes in H_{i-1} and H_j, in the linear arrangement produced by our algorithm. Hence the length of such an edge in the final linear arrangement will be at most $n_{i-1} + \ldots + n_j$. Suppose we charge $n_{p-1} + n_p$ for the portion of the edge that is stretched over the nodes in $H_{p-1} \cup H_p$, when considering level a_p, for all p in $[i, j+1]$. Then the total charge associated with edge e is equal to $n_{i-1} + 2(n_i + \ldots n_{j-1}) + n_j \geq n_{i-1} + \ldots + n_j$ — that is, edge e will be charged at least its length in the final linear arrangement.

We will now compute an upper bound on the cost of the linear arrangement obtained by our algorithm. Let $C(Z)$ be the maximum cost of a linear arrangement obtained by our algorithm for a subgraph of G whose volume of the spreading metric ℓ is at most Z. Since the sum of the weights of all edges in level a_i is ρ_{a_i}, and since we charge for the length of an edge as described in the preceding paragraph, we derive the following recurrence relation for $C(W)$:

$$\begin{aligned} C(W) &\leq C(W - \sum_{i=1}^{\kappa} \rho_{a_i}) + \sum_{i=1}^{\kappa} [\rho_{a_i}(n_{i-1} + n_i)] \\ &\leq C(W - \frac{\alpha_k n}{4 \log W}) + \alpha_{k+1} \sum_i (n_{i-1} + n_i) \end{aligned}$$

We now show that $C(W) = O(W \log n)$. We first prove the following lemma:

Lemma 4.4.1 $C(W) \leq cW \log W$, *for some constant c.*

Proof: We will use induction on W. Our base case for the induction will be the case $W \leq 0$. We can use induction on W here since, for any subgraph of G on x nodes whose volume of the spreading metric ℓ is at most Z ($Z \leq W$),

$$\frac{\alpha_k x}{4 \log Z} \geq \frac{\alpha_k x}{4 \log W} \geq \frac{1}{4 \log W}$$

That is, the recursive relation above will converge to the base case in at most $4W \log W$ steps.

The base case $W \leq 0$ corresponds to a totally disconnected graph (a graph with no edges); therefore $C(W) = 0$. If $W > 0$ then

$$\begin{aligned} C(W) &\leq C(W - \frac{\alpha_k n}{4 \log W}) + \alpha_{k+1} \sum_i (n_{i-1} + n_i) \\ &\leq c[W - \frac{\alpha_k n}{4 \log W}] \log[W - \frac{\alpha_k n}{4 \log W}] + 2\alpha_{k+1} n \\ &\leq c[W - \frac{\alpha_k n}{4 \log W}] \log W + 2\alpha_{k+1} n \\ &\leq cW \log W + \alpha_{k+1} n[2 - \frac{c}{8}] \\ &\leq cW \log W \end{aligned}$$

for a sufficiently large constant c ($c \geq 16$). The second step follows from the induction hypothesis, and the fourth step follows since $\alpha_{k+1} = 2\alpha_k$.

Q.E.D.

We still need to show how to bring the approximation factor down from $O(\log W)$ to $O(\log n)$. We will do this by using standard techniques of rescaling and rounding down the edge weights (as in [15]).

Our goal will be to reduce, by rescaling and rounding down weights, our original input graph G to an "equivalent" input graph G' whose spreading metric volume is a polynomial in n. Consider the set E' of edges e such that $w(e) \leq W/(mn)$. Since an edge has length at most n in any linear arrangement for G, the contribution of the edges in E' to a MLA of G is at most W. Suppose we delete all those edges, and apply a ρ-approximation algorithm to the resulting graph. We thus obtain a linear arrangement of G — by simply adding those edges back into the linear arrangement found — with cost that is within a $(\rho + 1)$ factor of the cost of a MLA of G.

We now round down each weight $w(e)$, for all e in $E \setminus E'$, to its nearest multiple of $W/(mn)$. The error incurred by this rounding procedure is again at most W. Furthermore, we scale the rounded weights by $W/(mn)$, obtaining new weights for the edges that are all integers in the interval $[0, mn]$. Note that we have only changed the units in which the weights are expressed. Hence we obtain a polynomial time $(\rho+2)$-approximation algorithm for the MLA problem on G with weights $w(e)$, by solving the MLA problem on $G' = G \setminus E'$ with integral weights that belong to $[0, mn]$. The volume W' of the spreading metric for G' is at most a polynomial in n. By Lemma 4.4.1, we have $C(W') \leq cW' \log(W') = c'W' \log n$, for some constant c'. Rescaling the edge weights back by multiplying $C(W')$ by $W/(mn)$, we obtain a MLA for the original weights on G with cost at most $c'W \log n$.

Finally, we can choose ℓ^* such that ℓ^* satisfies (4.1– 4.2), and whose volume W^* minimizes $\sum_{e \in E} w(e)\ell(e)$, over all spreading metrics ℓ that satisfy (4.1– 4.2). As we have seen, W^* is a lower bound on the cost of a MLA. Hence, by Lemma 4.4.1 and the considerations that follow this lemma, we have proved the following theorem:

Theorem 5 *The cost of a solution to the MLA problem, obtained by our algorithm for the spreading metric ℓ^* on G is within an $O(\log n)$ factor times the cost of a MLA of G.*

4.5 Graphs with excluded minors

In this section we show how to obtain, in polynomial time, an $O(\log\log n)$-approximation bound for the MLA problem on a graph G with no fixed $K_{r,r}$-minors — e.g., on a planar graph G. We denote the $r \times r$ complete bipartite graph by $K_{r,r}$.

Definition 4.5.1 *Let H and G be graphs. Suppose that (i) G contains disjoint connected subgraphs A_v, for each node v of H; and that (ii) for every edge (u,v) in H, there is a path $P_{(u,v)}$ in G with endpoints in A_u and A_v, such that any node in $P_{(u,v)}$ other than its endpoints does not belong to any A_w, w in H, nor to any $P_{(i,j)}$, (i,j) in $H \setminus (u,v)$. Then $\cup_v A_v$ is said to be an H*-minor *of G.*

Klein, Plotkin, and Rao [23], showed how to decompose (in polynomial time) a graph with no $K_{r,r}$-minors into connected components of small diameter. In our application, this implies that each connected component has at most a constant fraction of the nodes in G, as shown in the next section.

4.5.1 The algorithm

We recursively solve the problem, as we do in the general case. We combine the partial solutions returned by each recursive step, and charge for each edge removed at a cut step in the same way as in the algorithm of Section 4.4. It is in the way we decompose the graph before a recursive step that the algorithm of Section 4.4 differs considerably from the one presented in this section. Before each recursive step, we may perform a series of *shortest path levelings*, to be defined soon, on each induced connected subgraph, until we can guarantee that the original graph has been decomposed into subgraphs that contain at most a fixed fraction (strictly less than one) of the nodes each. In the algorithm of Section 4.4, we always perform only one shortest path leveling before each recursive step.

The algorithm proceeds in rounds. In each round we have a *cut step*, which corresponds to the series of cuts performed during the round, and a *recursive step*, which consists of recursing on the connected components that result from the cut step. Let $G(V,E)$ be a graph on n nodes that excludes $K_{r,r}$ as a minor, for some fixed $r > 0$. Let ℓ be a spreading metric for G of volume W that satisfies (4.1 – 4.2).

A *cut step* in G will produce a series of subgraphs of G, $G = G_0, \ldots, G_t$, $t < r$, where each G_{i+1} results from a shortest path leveling of G_i. Fix a node v in G_i. A *shortest path leveling (SPL)* of G_i rooted at v consists of an assignment of levels to the edges of G_i as follows: An edge (x, y) is at *level* j of this SPL if and only if $\mathsf{dist}(v, x)$ (in G_i) is at most j and $\mathsf{dist}(v, y)$ (in G_i) is greater than j, for all $j \in \mathbf{N}$. (An edge may be at more than one level.)

We will cut along a sequence of levels of the SPL; one of the connected components resulting from this cut procedure will be G_{i+1}. Let $n(G_i)$ denote the number of nodes in G_i. Let $s = n/b$, where b is a constant to be specified later. The spreading metric diameter guarantee implies that this SPL has at least $n(G_i)/4$ levels. We will see later that $n(G_i) = \Theta(n)$, and that we can choose b such that $n(G_i)/4 \geq 2s$ (we need $b \geq 8$). We group the levels of this SPL into *bands* of $2s$ consecutive levels as follows: Alternate coloring the bands "blue" and "red", in increasing order of the levels. Without loss of generality, assume that the subgraph induced by the blue bands has at least $n(G_i)/2$ nodes. We have $2s$ cuts of the following type: For $0 \leq j \leq 2s - 1$, a *leveled cut* j consists of all the edges in the jth level (with respect to distance from v) of every red band. That is, if the band consisting of the first $2s$ levels is colored blue, then the leveled cut j consists of the levels $2s + j, 6s + j, \ldots$, for all j.

Now we group the leveled cuts according to their indices. Let $\beta_k = W2^k/(s \log n)$, for all integer k in $[1, 2 \log \log n]$. Let $\beta_0 = 0$ and $\beta_{(2 \log \log n)} = W$. The *weight* of leveled cut j is the sum of the weights of the levels in the cut (the weight of a level being the sum of the weights of the edges at that level). Leveled cut j has *index* k, for all integer k in $[2 \log \log n]$, if and only if the weight of cut j belongs to the interval $I_k = (\beta_k, \beta_{k+1}]$. There are at least $2s/(2 \log \log n)$ leveled cuts with same index k_i (since there exist at least $2s$ distinct leveled cuts).

If $k_i > 0$, then we cut along these at least $s/(\log \log n)$ leveled cuts of index k_i, and recurse on the resulting connected components. In this case, we let $t = i$, and the cut step of this round is complete. Otherwise, we first cut along only one of the leveled cuts of index $k_i = 0$ (chosen arbitrarily). Then we check whether there exists a connected component G_{i+1} of G_i with more than $n(G_i)/2$ nodes. In case no such component exists, we let $t = i$ (the cut step of this round is complete), and we recurse. If $i = r-1$, we also let $t = i = r-1$ and recurse. Otherwise, we proceed by performing a SPL on G_{i+1}, following the procedure just described, with $i = i+1$.

The number of nodes in G_i, $n(G_i)$, is proportional to n, for all i in $[r]$. This follows since $n(G_{i+1}) \geq n(G_i)/2$, for all i, by the choice of G_{i+1} and since r is a constant.

Suppose we just performed a series of r SPL's and corresponding cut procedures. The last cut performed, on G_{r-1}, generated a collection of connected components of G_{r-1}. Klein, Plotkin, and Rao [23] proved that the distance in G between any pair of nodes in any such component is $O(s)$ (where the constant in the $O(\cdot)$ notation depends only on r). Thus, for a suitably chosen constant b, we can ensure that the distance between any pair of nodes is at most $n/6$, in any such component.

It follows from the result by Klein, Plotkin, and Rao that any connected component that results from this cut step has at most $2n/3$ nodes, as we now show. Fix any node u in G. It follows from (4.1), that any subgraph of G on $(n - x)$ nodes that contains u has a node at distance at least $(n - x)/4$ from u. Suppose we start with the graph G, and proceed by removing one node at a time, choosing always a node that has maximum distance to u among the remaining nodes. Thus, we need to remove at least one-third of the nodes before we are left only with nodes that are within distance $n/6$ from u in G. This implies that any resulting connected component of G_{r-1} has at most $2n/3$ nodes. Any other resulting connected component (of $G \setminus G_{r-1}$) has at most $n/2$ nodes, by the choice of the G_i's.

We distinguish between two types of cut steps: if $k_t = 0$, then we call the cut step in this round a *cut step with reduction in size*; otherwise $k_t > 0$, and we call the cut step in this round *a cut step with reduction in volume*. Note that $k_t = 0$ implies $k_j = 0$, for all j in $[t]$.

Let $C(Z, x)$ denote the maximum cost of a linear arrangement obtained by our algorithm for a subgraph of G with x nodes, whose volume of the spreading metric ℓ is at most Z.

Lemma 4.5.1 *$C(W, n) \leq cW \log\log n$, for some constant c.*

Proof: We use induction on n and W: We apply induction on n whenever we have a cut step with reduction in size, and we use induction on W whenever we have a cut step with reduction in volume. Our base cases will be the cases when $n \leq 1$ or $W \leq 0$. When we have a cut step with reduction in volume, for a subgraph of G on x nodes whose volume of the spreading metric ℓ is at most Z ($Z \leq W$), the reduction in volume in that cut step is at least

$$\frac{\beta_k x}{b \log\log x} \geq \frac{\beta_k x}{b \log\log n} \geq \frac{1}{b \log\log n}$$

Thus, at every recursive step, we either reduce the volume of the spreading metric in the remaining subgraph by at least $1/(b \log\log n)$ or we decompose the graph into more than one connected components, all of which have at most a 2/3-fraction of the nodes. Hence, the inductive process will converge to one of the base cases in at most $\max(bW \log\log n, O(\log n))$ steps (since we can have at most $bW \log\log n$ cut steps with reduction in volume and at most $O(\log n)$ cut steps with reduction in size).

The base cases for $W \leq 0$ or $n = 1$ are trivial. Suppose we perform a cut step with reduction in size. Let the connected components resulting from this step be $H_0, \ldots, H_p$. Then, since we cut along r leveled cuts of weight at least $W/(s \log n)$ (we over-charge n for the cost of each occurrence of an edge in each of these leveled cuts)

$$\begin{aligned}
C(W, n) &\leq \sum_{i=0}^{p} C(W_i, n_i) + r\frac{2W}{s \log n} n \\
&\leq \sum_{i=0}^{p} c(W_i) \log\log(2n/3) + \frac{2brW}{\log n} \\
&\leq cW \log\log n - \frac{cW}{3 \log n} + \frac{2brW}{\log n} \\
&\leq cW \log\log n - \frac{W(c/3 - 2br)}{\log n} \\
&\leq cW \log\log n
\end{aligned}$$

where W_i and n_i are the volume and number of nodes, respectively, associated with component H_i. We have shown that every n_i is at most $2n/3$. The second step follows by induction and since $\sum_{i=0}^{p} W_i \leq W$. Note that $\log\log(2n/3) \leq \log\log n - 1/(3 \log n)$, and thus the third step follows. The last step follows for a sufficiently large constant c (e.g., $c \geq 6br$).

If the cut step performed was with reduction in volume, then we performed a series of $t \leq r$ SPL's and respective cut procedures. The last term on the right-hand side of the first inequality below accounts for the first $(t-1)$th leveled cuts used. The second term on the right-hand side of that inequality accounts for the tth leveled cut used. The charging scheme for the edges removed in the tth leveled cut of this cut step is analogous to the scheme presented in Section 4.4.

$$
\begin{aligned}
C(W,n) &\leq C(W - \frac{\beta_k s}{\log\log n}, n) + 2\beta_{k+1}n + (r-1)\frac{2W}{s\log n}n \\
&\leq C(W - \frac{\beta_k s}{\log\log n}, n) + (r+3)\beta_k n \\
&\leq c(W - \frac{\beta_k s}{\log\log n})\log\log n + (r+3)\beta_k n \\
&\leq cW\log\log n + \beta_k n(r + 3 - \frac{c\log\log n}{b\log\log n}) \\
&\leq cW\log\log n
\end{aligned}
$$

when $c \geq b(r+3)$. The second step above follows from $\beta_k \geq 2W/(s\log n)$, and from $\beta_{k+1} = 2\beta_k$, $0 < k < 2\log\log n - 1$; the third step follows by induction.

Q.E.D.

As in Section 4.4, we choose a spreading metric ℓ^* that satisfies (4.1 – 4.2), and whose volume W^* is a lower bound on the cost of a MLA of G. By Lemma 4.5.1, we obtain the following theorem:

Theorem 6 *Given a graph G on n nodes that excludes fixed $K_{r,r}$-minors, the cost of a solution to the MLA problem, obtained by the algorithm presented in this section for the spreading metric ℓ^*, is within an $O(\log\log n)$ factor times the cost of a MLA of G.*

4.6 Minimum storage–time product

In this section, we sketch our approach to approximating the storage–time product for a directed acyclic graph $G(V,E)$. The minimum storage–time product problem arises in a

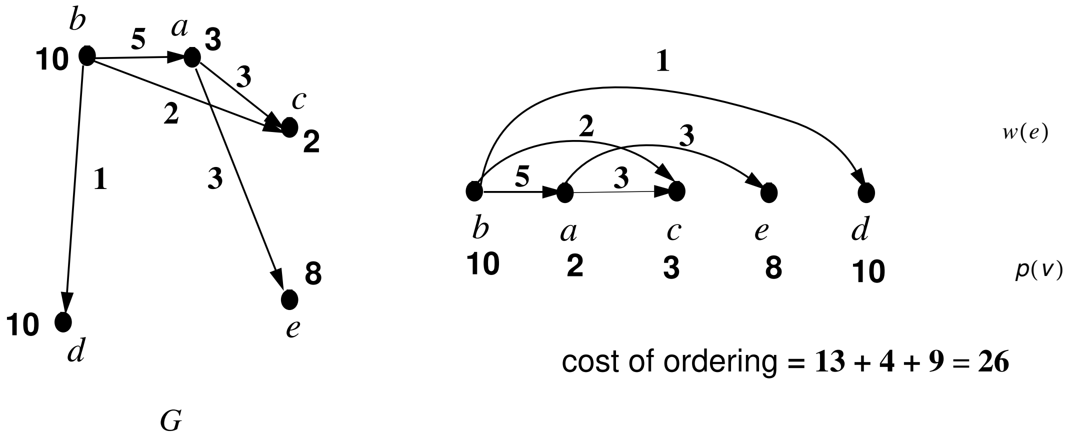


Figure 4.4: A minimum storage–time product of G.

manufacturing or computational process, in which the goal is to minimize the storage–time product of the process: We want to minimize the use of storage over time, assuming storage is an expensive resource. Let $G(V, E)$ be an acyclic directed graph on n nodes with edge weights $w(e)$, for all e in E, and node weights $\tau(v)$, for all v in V. The nodes of G represent tasks to be scheduled on a single processor. The time required to process task v is given by $\tau(v)$. The weight on edge (u, v), $w(u, v)$, represents the number of units of storage required to save the intermediate results generated by task u until they are consumed at task v. The *minimum storage–time product problem* consists of finding a topological ordering[1] of the nodes $\sigma : V \to \{1, \ldots, n\}$ that minimizes

$$\sum_{(i,j)\in E, \sigma(i)<\sigma(j)} \left\{ w(i,j) \left[\sum_{k\,:\,\sigma(i)\leq\sigma(k)\leq\sigma(j)} \tau(\sigma(k)) \right] \right\}.$$

Figure 4.4 illustrates a topological ordering of the nodes of G (given from left to right on the rightmost representation of the graph) with minimum storage–time product of 26.

This problem generalizes the MLA problem: When all tasks have unit execution time, it becomes a directed version of the MLA problem. It is also a generalization of the single-

[1] An ordering σ of the nodes of G (where G is an acyclic graph) is said to be *topological* if and only if for every $(i, j) \in E$, $\sigma(i) < \sigma(j)$.

processor scheduling problem, if we are minimizing the weighted sum of completion times (this problem is NP-complete [16, problem SS13, page 240]).

We use a spreading metric defined as follows (see [13]). Let $E^R = \{(u,v)|(v,u) \in E\}$. We define $G' = (V, E \cup E^R)$. Let $\mathcal{V}$ denote the set of all nontrivial strongly connected subgraphs of G'. Find $\ell^* : E \to \mathbf{Q}$ that minimizes $\sum_{e \in E} w(e)\ell(e)$, while satisfying the constraints

$$\frac{\sum_{v \in U} \delta(v,u)}{|U|} \geq \frac{\sum_{v \in U} \tau(v)}{4}, \ \forall u \in U, \forall U \in \mathcal{V}$$
$$\ell(i,j) \geq \tau(i) + \tau(j), \ \ \forall (i,j) \in E$$

where $\delta(u,v) = \mathsf{dist}(u,v) + \mathsf{dist}(v,u)$. Here we define $\mathsf{dist}(u,v)$ to be the length of a shortest path from u to v in G' according to the lengths $\ell(e)$ for e in E, and where each e in E^R has length 0.

For any linear ordering σ of V, the assignment of lengths to the edges given by $\ell(i,j) = \sum_{k\,:\,\sigma(i) \leq \sigma(k) \leq \sigma(j)} \tau(\sigma(k))$, for all (i,j) in E, satisfies the constraints above. Thus the volume W^* of the spreading metric ℓ^* is a lower bound on the optimal cost of a solution to the storage–time product problem.

Given the spreading metric constraints above we can apply the algorithm of Section 4.4 to this problem as follows. Let $T = \sum_{v \in V} \tau(v)$. There is a node v such that either the out-tree or the in-tree rooted at v has depth $\Omega(T)$. Thus, we can find a sequence $a_1, \ldots, a_\kappa$ of $\kappa = \Omega(T/\log W)$ levels whose weights $\rho_{a_1}, \ldots \rho_{a_\kappa}$ are within a factor of two of each other (as in Section 4.4.

Laying out the resulting pieces successively, we obtain a solution where the cost is bounded by

$$C(W) \leq C(W - \sum_{i=1}^{\kappa} \rho_{a_i}) + \sum_{i=1}^{\kappa} [\rho_{a_i}(\tau_{i-1} + \tau_i)].$$

where τ_i is the sum of $\tau(v)$ over all nodes v that lie between levels a_{i-1} and a_i (τ_0 and τ_κ are defined accordingly).

This recursion can be upper bounded by $O(W \log W)$, as in Section 4.4. This cost can be reduced to $O(W \log T)$ using the standard techniques that were used in Section 4.4 to reduce $O(\log W)$ to $O(\log n)$.

4.7 Minimum containing interval graph

In this section, we sketch our approach to approximating the cost of a minimum containing interval graph of a graph $G(V, E)$. We first introduce interval graphs. An *interval graph* is a graph whose vertices can be mapped to distinct intervals in the real line such that two vertices in the graph have an edge between them if and only if their corresponding intervals overlap. A completion of a graph G into an interval graph results in an interval graph with same node set as G that contains G as a subgraph.

We use the following characterization of interval graphs, due to [44]. An undirected graph $G(V, E)$ on n nodes is an interval graph if and only if there exists a linear ordering $\sigma : V \to \{1, \ldots, n\}$ of the nodes in V such that if an edge (u, v) is in E, where $\sigma(u) < \sigma(v)$, then every edge (u, w), for w such that $\sigma(u) < \sigma(w) < \sigma(v)$, also belongs to E. This characterization implies that, for any given ordering σ, there exists a ***unique*** way of completing G into an interval graph by adding as few edges to G as possible.

The cost of a completed graph is given by the *total number of edges* in the (completed) graph. This cost can be viewed as the sum over vertices of the maximum backward stretch of the vertex — i.e., of the distance to the farthest lower-numbered node to which the vertex is connected. This is very similar to the MLA problem, except that the nodes are stretched along the order rather than the edges (see [13]). Thus, our techniques also apply to this problem.

This problem arises in several areas, from computer science, to biology (see [34]), to archaeology (e.g., when finding a consistent chronological model for tool use while making as few assumptions as possible [22]).

The spreading metric ℓ^* that we use (due to [13]) assigns lengths to the nodes of the graph, rather than to its edges, as in the minimum linear arrangement and in the minimum storage–time product problems. Let $\mathcal{V}$ denote the set of all nontrivial connected subgraphs of G. The metric ℓ^* is a function $\ell : V \to \mathbf{Q}$ that minimizes $(\sum_{v \in V} \ell(v))/2$, while satisfying the constraints

$$\sum_{v \in U} \mathsf{dist}(u, v) \geq \frac{1}{4}(|U|^2 - 1), \quad \forall u \in U, \ \forall U \in \mathcal{V}$$

$$\ell(v) \geq 0, \quad \forall v \in V$$

where $\mathsf{dist}(u, v)$ is the shortest length — given by $[\ell(u) + \sum_{i=0}^{p} \ell(x_i) + \ell(v)]$ — of a path $u, x_0, \ldots, x_p, v$, $x_i \in V$, from u to v in G.

Let $G'(V, E')$ be a completion of G into an interval graph. If we let $\ell(v)$ be the degree of node v in G', the cost $(\sum_{v \in V} \ell(v))/2$ clearly gives the number of edges in E'. Also this assignment of lengths to the nodes satisfies the constraints above. Hence the volume W^* of the metric ℓ^* is a lower bound on the number of edges in a minimum containing interval graph of G.

The recurrence relations that bound the cost of a solution obtained for the minimum containing interval graph problem are analogous to the ones for the MLA problem, both for the general case and for the excluded $K_{r,r}$-minors case.

4.8 Conclusion

We provided an existentially tight bound on the relationship between the spreading metric cost volumes and the true optimal values for the problems of minimum linear arrangement, minimum containing interval graph, and minimum storage–time product.

It would be interesting to extend our techniques to obtain $O(\log n)$-approximation algorithms for other problems. In particular, it seems natural to extend our techniques to improve the best known approximation factors for other problems that satisfy the “approximation paradigm” of [13]. We would then provide an existentially tight bound — on the ratio between the value of an optimal solution and the spreading metric volume — for any such problem.

However, since the approach used here depends on the structure of graph ordering problems, new ideas may be required.

Bibliography

[1] N. Alon.
A parallel algorithmic version of the Local Lemma.
Random Structures and Algorithms, 2(4):367–378, 1991.

[2] D. Angluin and L. G. Valiant.
Fast probabilistic algorithms for hamiltonian circuits and matchings.
Journal of Computer and System Sciences, 18(2):155–193, April 1979.

[3] B. Awerbuch, Y. Bartal, and A. Fiat.
Distributed paging for general networks.
In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 574–583, January 1996.

[4] B. Awerbuch and D. Peleg.
Routing with polynomial communication space tradeoff.
SIAM Journal on Discrete Mathematics, 5:151–162, 1990.

[5] B. Awerbuch and D. Peleg.
Sparse partitions.
In *Proceedings of the Thirty-First Annual IEEE Symposium on Foundations of Computer Science*, pages 503–513, October 1990.

[6] B. Awerbuch and D. Peleg.
Online tracking of mobile users.
Journal of the ACM, 42(5):1021–1058, September 1995.

[7] Y. Bartal, A. Fiat, and Y. Rabani.
Competitive algorithms for distributed data management.

In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 39–47, May 1992.

[8] J. Beck.
An algorithmic approach to the Lovász Local Lemma I.
Random Structures and Algorithms, 2(4):343–365, 1991.

[9] R. P. Brent.
The parallel evaluation of general arithmetic expressions.
Journal of the ACM, 21(2):201–208, April 1974.

[10] H. Chernoff.
A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations.
Annals of Mathematical Statistics, 23:493–507, 1952.

[11] S. Dolev, E. Kranakis, D. Krizanc, and D. Peleg.
Bubbles: Adaptative routing scheme for high-speed dynamic networks.
In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 528–537, May 1995.

[12] P. Erdös and L. Lovás.
Problems and results on 3-chromatic hypergraphs and some related questions.
In A. Hajnal et al., editor, *Infinite and Finite Sets.*
Volume 11 of *Colloq. Math. Soc. J. Bolyai*, pages 609–627. North Holland, Amsterdam, The Netherlands, 1975.

[13] G. Even, J. Naor, S. Rao, and B. Schieber.
Divide-and-conquer approximation algorithms via spreading metrics.
In *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 62–71, October 1995.

[14] G. Even, J. Naor, S. Rao, and B. Schieber.
Spreading metric based approximate graph partitioning algorithms.
In *Proceedings of the Eigth Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 639–648, January 1997.

[15] G. Even, J. Naor, B. Schieber, and M. Sudan.
Approximating minimum feedback sets and multicuts in directed graphs.
In E. Balas and J. Clausen, editors, *Integer programming and combinatorial optimization.*
Volume 920 of *Lecture Notes in Computer Science*, pages 14–28. Springer–Verlag, New York, 1995.
Full version appears in IBM Research Report RC 20074 (88796).

[16] M. R. Garey and D. S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman, NY, 1979.

[17] R. L. Graham, D. E. Knuth, and O. Patashnik.
Concrete Mathematics.
Addison-Wesley, Reading, MA, 1989.

[18] J. D. Guyton and M. F. Schwartz.
Locating nearby copies of replicated Internet servers.
In *Proceedings of ACM SIGCOMM*, pages 288–298, 1995.

[19] M. Hansen.
Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems.
In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 604–609, October 1989.

[20] D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, and R. Panigrahy.
Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web.
In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[21] R. Karp, M. Luby, and F. Meyer auf der Heide.
Efficient PRAM simulation on a distributed memory machine.
In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 318–326, May 1992.

[22] D. G. Kendall.
Incidence matrices, interval graphs, and seriation in archeology.
Pacific J. Math., 28:565–570, 1969.

[23] P. Klein, S. Plotkin, and S. Rao.
Excluded minors, network decomposition and multicommodity flow.
In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 682–690, October 1993.

[24] R. R. Koch, F. T. Leighton, B. M. Maggs, S. B. Rao, A. L. Rosenberg, and E. J. Schwabe.
Work-preserving emulations of fixed-connection networks.
Journal of the ACM, 44(1):104–147, January 1997.

[25] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao.
Randomized routing and sorting on fixed-connection networks.
Journal of Algorithms, 17(1):157–205, July 1994.

[26] F. T. Leighton, B. M. Maggs, and S. Rao.
Universal packet routing algorithms.
In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 256–271, October 1988.

[27] F. T. Leighton, B. M. Maggs, and S. B. Rao.
Packet routing and job-shop scheduling in O(congestion + dilation) steps.
Combinatorica, 14(2):167–180, 1994.

[28] F. T. Leighton, B. M. Maggs, and A. W. Richa.
Fast algorithms for finding O(congestion + dilation) packet routing schedules.
Technical Report CMU–CS–96–152, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 1996.
To appear in *Combinatorica*.

[29] F. T. Leighton and S. Rao.
An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms.

In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 422–431. IEEE Computer Society Press, October 1988.

[30] C. E. Leiserson.
Fat-trees: Universal networks for hardware-efficient supercomputing.
IEEE Transactions on Computers, C–34:892–900, 1985.

[31] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann.
Exploiting locality for data management in systems of limited bandwidth.
In *Proceedings of the Thirty-Eighth Annual Symposium on Foundations of Computer Science*, pages 284–293, October 1997.

[32] B. M. Maggs and E. J. Schwabe.
Real-time emulations of bounded-degree networks.
Information Processing Letters, 1998.
To appear.

[33] Y. Mansour and B. Patt-Shamir.
Greedy packet scheduling on shortest paths.
Journal of Algorithms, 14:449–65, 1993.

[34] J. Meidanis and J. C. Setubal.
Introduction to Computational Molecular Biology.
PWS Publishing Co., Boston, MA, 1997.

[35] F. Meyer auf der Heide and C. Scheideler.
Routing with bounded buffers and hot-potato routing in vertex-symmetric networks.
In *Proceedings of the Third European Symposium on Algorithms*, pages 341–354, 1995.

[36] F. Meyer auf der Heide and B. Vöcking.
A packet routing protocol for arbitrary networks.
In *Proceedings of the Twelfth Symposium on Theoretical Aspects of Computer Science.*
Volume 349 of *Lecture Notes in Computer Science*, pages 291–302. Springer–Verlag, Heidelberg, Germany, March 1995.

[37] S. J. Mullender, editor.
Distributed Systems.
Addison-Wesley, 1993.

[38] S. J. Mullender and P. M. B. Vitányi.
Distributed match-making.
Algorithmica, 3:367–391, 1988.

[39] R. Ostrovsky and Y. Rabani.
Universal $O(\text{congestion} + \text{dilation} + \log^{1+\epsilon} N)$ local control packet switching algorithms.
In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 644–653, May 1997.

[40] C. G. Plaxton and R. Rajaraman.
Fast fault-tolerant concurrent access to shared objects.
In *Proceedings of the Thirty-Seventh Annual IEEE Symposium on Foundations of Computer Science*, pages 570–579, October 1996.

[41] C. G. Plaxton, R. Rajaraman, and A. W. Richa.
Accessing nearby copies of replicated objects in a distributed environment.
In *Proceedings of the Ninth ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.
To appear in special issue of *Theory of Computing Systems*.

[42] Y. Rabani and É. Tardos.
Distributed packet switching in arbitrary networks.
In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 366–375, May 1996.

[43] P. Raghavan.
Probabilistic construction of deterministic algorithms: Approximate packing integer programs.
Journal of Computer and System Sciences, 37(4):130–143, October 1988.

[44] G. Ramalingam and C. Pandu Rangan.
A unified approach to domination problems in interval graphs.

Information Processing Letters, 27:271–174, 1988.

[45] A. G. Ranade.
How to emulate shared memory.
Journal of Computer and System Sciences, 42:307–326, 1991.

[46] G. N. Raney.
Functional composition patterns and power series reversion.
Transactions American Mathematical Society, 94:441–451, 1960.

[47] S. Rao and A. W. Richa.
New aprroximation techiniques for some ordering problems.
In *Proceedings of the Ninth Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 211–218, January 1998.

[48] R. Ravi, A. Agrawal, and P. Klein.
Ordering problems approximated: Single processor scheduling and interval graph completion.
In *Proceedings of the Eighteenth International Colloquium on Automata, Languages and Programming*, pages 751–762, July 1991.

[49] C. Scheideler.
Universal Routing Strategies for Interconnection Networks, Vol. 1390 of *Lecture Notes in Computer Science*.
Springer–Verlag, Berlin, Germany, 1998.

[50] P. D. Seymour.
Packing directed circuits fractionally.
Combinatorica, 15(2):281–288, 1995.

[51] D. B. Shmoys, C. Stein, and J. Wein.
Improved approximation algorithms for shop scheduling problems.
In *Proceedings of the Second Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 148–157, January 1991.

[52] J. Spencer.
Ten Lectures on the Probabilistic Method.

SIAM, Philadelphia, PA, 1987.

[53] A. Srinivasan and C.-P. Teo.
A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria.
In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 636–643, May 1997.

[54] E. Upfal and A. Wigderson.
How to share memory in a distributed system.
Journal of the ACM, 34:116–127, 1987.

[55] M. van Steen, F. J. Hauck, and A. S. Tanenbaum.
A model for worldwide tracking of distributed objects.
In *Proceedings of Telecommunications Information Networking Architecture*, pages 203–212, September 1996.