model has versions allowing both limited and unlimited size messages, and many typical results are insensitive to this distinction.

Bibliographical notes

A number of common standard models for distributed systems appear in the literature. Our basic framework of a point-to-point communication network follows numerous references in the literature (cf. [LL90, Lyn95]). Among the first papers that paved the way to the systematic study of distributed network algorithms in this model are [Dij74, Gal76, GMS77, LeL77, Gal82, GHS83].

Exercises

1. Give an example for a (small) tree with a vertex whose level, max-level and min-level are *three different* numbers.

2. (a) State and prove an inequality relating $L(v)$, $\hat{L}(v)$ and $Depth(T)$ for a vertex v in a rooted tree T.

   (b) Characterize the vertices v for which equality holds.

3. Let PRUNE(T) be a procedure eliminating an arbitrary leaf from the tree T, and consider a tree T' obtained from T by a number of repeated applications of Procedure PRUNE(T). For a vertex $v \in T'$, let $L_T(v)$ and $L_{T'}(v)$ denote, respectively, v's layer number in T and T' and similarly for $\hat{L}$ and $\check{L}$. Disprove, prove or strengthen each of the following claims:

   (a) $L_{T'}(v) \leq L_T(v)$.

   (b) $\hat{L}_{T'}(v) \leq \hat{L}_T(v)$.

   (c) $\check{L}_{T'}(v) \leq \check{L}_T(v)$.

4. Consider an n-vertex network $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$. The *individual messages (IM)* task requires vertex v_1 to deliver a (distinct) $\log n$-bit message to every other vertex in the network along some prespecified shortest route. Prove or disprove each of the following claims regarding the message complexity of the problem.

   (a) **[Upper bound]:** Message$(IM) = O(nD)$ (or there exists a constant $c > 0$ such that for every network G as above, Message$(IM, G) \leq cnD$).

   (b) **[Lower bound]:** Message$(IM) = \Omega(nD)$ (or there exists a constant $c > 0$ such that for every $n \geq 1$, there exists an n-vertex network G as above for which Message$(IM, G) \geq cnD$).

   (c) **[Global lower bound]:** There exists a constant $c > 0$ such that for every network G as above, Message$(IM, G) \geq cnD$.

5. Consider the IM task of the previous question in the synchronous $\mathcal{CONGEST}$ model. Prove or disprove each of the following claims regarding the time complexity of the problem.

   (a) **[Upper bound]:** Time$(IM) = O(D)$ (or there exists a constant $c > 0$ such that for every network G as above, Time$(IM, G) \leq cD$).

   (b) **[Upper bound]:** Time$(IM) = O(n)$ (or there exists a constant $c > 0$ such that for every network G as above, Time$(IM, G) \leq cn$).

   (c) **[Lower bound]:** Time$(IM) = \Omega(D)$ (or there exists a constant $c > 0$ such that for every $n \geq 1$, there exists an n-vertex network G as above for which Time$(IM, G) \geq cD$).

   (d) **[Global lower bound]:** There exists a constant $c > 0$ such that for every network G as above, Time$(IM, G) \geq cD$.

6. Repeat the previous question in the $\mathcal{LOCAL}$ and the $\mathcal{ASYNC}$ models.

presented in Section 3.4.1, can be computed by performing Procedure CONVERGE($\bigwedge$, *Pred*) for

$$Pred(v) = \text{“}v \text{ has received the message.”}$$

(This is a rather trivial use of the logical and computation, as all inputs are 1 once defined, hence so is the output.)

The message and time complexities of this procedure on a tree T are $O(n)$ and $O(Depth(T))$, respectively.

Again, the same applies if we wish to apply our function to the inputs X_v of only a subset W of the vertices. In this case, we refer to the process corresponding to the operation *op* by Procedure CONVERGE(op, X, W). □

3.4.3 Pipelined broadcasts and convergecasts

We next consider a situation in which the vertices of the network store data of a number of different types, and it is necessary to perform separate convergecast computations on each of the different data collections. For instance, suppose that each vertex in the tree stores k separate variables $X_i(v)$, $1 \leq i \leq k$, and we would like to compute all k maximums

$$M_i = \max_{v \in V}\{X_i(v)\}$$

and inform all vertices of the results. Clearly, for each i, it is possible to compute M_i separately, through a convergecast process performed by Procedure CONVERGE(max), and then broadcast the result on the tree. Assuming the variables hold $\log n$-bit numbers, each of those k processes requires Message(CONVERGE(max)) $= O(n)$ and Time(CONVERGE(max)) $= O(Depth(T))$. But performing all k operations sequentially, waiting for the computation of M_{i-1} to end before beginning the computation of M_i, would multiply the time complexity by a factor of k, resulting in a total of $O(k \cdot Depth(T))$ time.

A simple technique for reducing this time complexity is to *pipeline* the computations. Each leaf v starts the k processes one after another, sending $X_1(v)$ followed by $X_2(v)$ and so on. Denote by $T(v)$ the subtree of T rooted at a vertex v. Each intermediate vertex v computes each of the k partial maximums

$$M_i(v) = \max_{w \in T_v}\{X_i(w)\}$$

immediately when receiving the corresponding partial maximums from all its children and sends the values $M_i(v)$ to its parent one by one. This process is illustrated in Figure 3.6.

In the synchronous model, the algorithm can be formalized by noting that each vertex v sends the values $M_i(v)$ to its parent consecutively, at rounds $\hat{L}(v) + i$ (for $1 \leq i \leq k$) (as defined in Section 2.1.2). This can be proved for every i by induction on $\hat{L}(v)$, from the leaves up. We conclude the following.

Lemma 3.4.6 *In the synchronous model, computing k global semigroup functions on a tree T can be performed in $Depth(T) + k$ time.*

In the asynchronous model, we rely on the fact that messages are passed up the tree in first-in first-out order, so an intermediate vertex can match together the values of the ith type, which it receives from each of its children, and interpret them as belonging to the ith maximum computation. It is easy to verify that the complexity bounds remain the same.

Lemma 3.4.7 *In the asynchronous model, computing k global semigroup functions on a tree T can be performed in $Depth(T) + k$ time as well.*

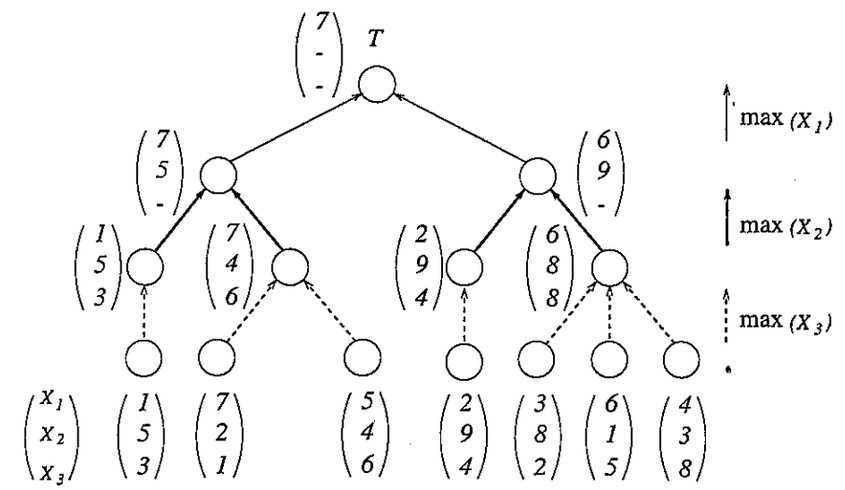

Figure 3.6: Three parallel processes of Procedure CONVERGE(max) carried over the spanning tree T. (For pictorial simplicity, this example assumes the inputs are stored only at the leaves.)

Bibliographical notes

Broadcast in computer networks was the subject of extensive studies. One early source for the study of broadcast as a basic primitive in distributed computing is [DM78].

Literature on broadcast has dealt extensively with a broad spectrum of aspects and models. Much of the research dealt with the so-called telephone model, in which communication is synchronous and a vertex can speak with only one neighbor in each round. The survey papers [HHL88, FL94] cover some of the work done in this area in recent years (also see Chapters 5.2.2 and 5.2.3 in [BG92]). A variant of the pipelined convergecast problem is studied in detail under an elaborate database setting in [T95]. Exercise 4 was suggested by [SS96].

Exercises

1. Formally prove Lemmas 3.2.1 and 3.2.3.
2. Consider a partially synchronous model in which the delay incurred by each message is between 1/4 and 1 time unit. What can be said about the behavior of Algorithm FLOOD&ECHO in this model? In particular, how long does it take until every vertex receives the message, what is the time complexity of the algorithm and what is the depth of the spanning tree constructed by it?
3. Describe the processes resulting from applying Procedure CONVERGE(min) to the tree of Figure 3.5 and Procedure CONVERGE(+) to the tree of Figure 3.6 (with the given inputs).
4. Consider the global addition operation described in the first example in Section 3.4.2. Explain how pipelining can be used to speed up the computation, and determine the resulting time complexity.
5. (a) Give a distributed algorithm for counting the number of vertices in a rooted tree T, initiated at the root.

(b) Extend your algorithm to an arbitrary graph G.

(c) Give a distributed algorithm for counting the number of vertices in each layer of the rooted tree T separately. Analyze the time and message complexities of your algorithm.

6. (a) Describe a distributed algorithm initiated by the root of a tree T for calculating the max-level $\hat{L}(v)$ of each vertex v in the tree, and analyze its complexities.

(b) Repeat the question for the min-level $\check{L}(v)$ of the vertices.

Chapter 4

Downcasts and upcasts

Another pair of tasks that can be carried out via the process similar to (but different from) broadcast and convergecast is *downcast* and *upcast*. These tasks refer to the case where there is a possibly different item communicated between the root and each of the vertices in the tree, hence each such item needs to be treated individually and may not be combined. In this chapter we discuss the downcast and upcast operations under different settings. For simplicity, we will assume the synchronous model, although a number of the principal results described hold also for the asynchronous model. As our focus here is on message complexity, we will disallow large messages, i.e., concentrate on the synchronous $\mathcal{CONGEST}$ model.

4.1 Downcasts

We first examine the case where the root has m distinct items $\mathcal{A} = \{\mu_1, \ldots, \mu_m\}$, each destined to one specific vertex in the tree. (Each vertex in the tree may get zero or more such messages.) Clearly, both the depth of the tree and the number of distinct messages that need to be sent are potential bottlenecks, hence we have the following lower bounds.

Lemma 4.1.1

1. *Downcasting m distinct messages on T requires $\Omega(Depth(T))$ time in the worst case for every tree T.*

2. *Downcasting m distinct messages on arbitrary trees requires $\Omega(m)$ time in the worst case.*

These lower bounds can be met by the straightforward algorithm DOWNCAST. For each of its children w, the root r_0 simply sends the messages destined to the subtree rooted at w one by one on the edge (r_0, w), in an arbitrary order. Each intermediate vertex v in the tree receives at most one message at each step and passes it on towards its destination (unless the destination of the message is itself).

Lemma 4.1.2 *Algorithm* DOWNCAST *performs downcasting of m distinct messages on the tree T in time $O(m + Depth(T))$.*

Proof: The root releases at least one message per step, so it terminates by time m. The proof is completed upon noting that messages are never delayed, so a message leaving the root at time t will reach its destination at time at most $t + Depth(T)$. □

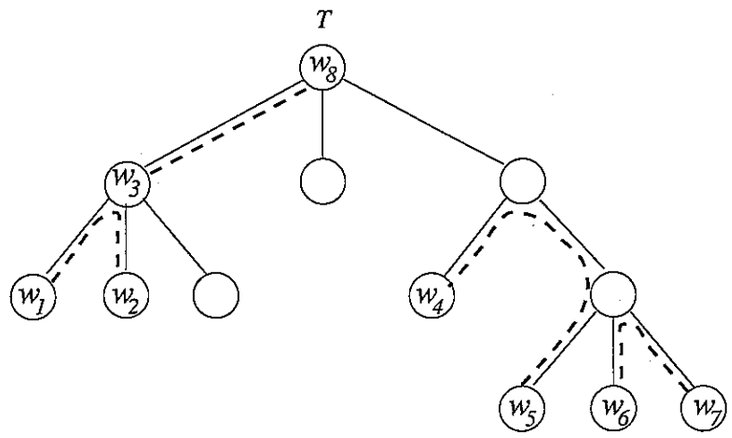

Figure 4.4: An instance of the route-disjoint matching problem and a possible solution for it.

vertex $v \neq w_{i_1}$ along Υ_i should know the edge leading from it towards w_{i_1} on Υ_i.) We have the following lemma, whose proof is left to Exercise 6.

Lemma 4.3.2

1. *For every tree T and for every set W as above, there exists an edge-disjoint matching as required.*
2. *Furthermore, this matching can be found by a distributed algorithm on T in time $O(Depth(T))$.* □

4.3.4 Token distribution

The *token distribution* problem is stated as follows: n tokens (of $O(\log n)$ bits each) are initially distributed among the n vertices of the tree with no more than K at each site. Redistribute the tokens so that each processor will have exactly one token.

The limit on the token size implies that each token can be sent in a single message. Hence the cost of the entire redistribution process equals the sum of the distances traversed by the tokens in their way to their destinations. An optimal solution can be derived by using a convergecast process in order to determine, for every vertex v in the tree, the following three parameters:

1. s_u, the number of tokens in the subtree T_u,
2. n_u, the number of vertices in the subtree T_u,
   and consequently,
3. $p_u = s_u - n_u$, the (positive or negative) number of tokens that need to be transferred out of T_u.

The total number of messages required for achieving an even distribution of the tokens is at least $P = \sum_{u \neq r_0} |p_u|$. This bound can be met by an appropriate distributed algorithm, as stated in the following lemma.

Lemma 4.3.3 *There exists a distributed algorithm for performing token distribution on a tree using an optimal number of messages P and $O(n)$ time, after a preprocessing stage requiring $O(Depth(T))$ time and $O(n)$ messages.*

Bibliographical notes

Results similar to Lemma 4.1.2 and Corollary 4.2.7 hold also in much more general settings, without the tree structure. In particular, the bounds hold even when the m messages are sent from different senders to different (possibly overlapping) recipients along arbitrary shortest paths and under a wide class of conflict resolution policies (for resolving collisions in intermediate vertices between messages competing over the use of an outgoing edge), so long as these policies are consistent (namely, if μ_i is preferred over μ_j at some vertex v along their paths, then the same preference will be made whenever their paths intersect again in the future). This was first shown in [CKMP90, RVVN90] for two specific policies and was later extended to any consistent greedy policy in [MPS91].

Part 1 of Lemma 4.3.2 is due to [KR93].

Exercises

1. (a) Explain why the first lower bound of Lemma 4.1.1 is global while the second is only existential.

   (b) Characterize a subclass of trees for which the second lower bound of the lemma applies globally.

2. Prove Lemma 4.2.2 in the setting of Section 4.2.2, namely, gathering ordered items with unmarked rank. (Hint: the proof requires a double induction: on $\hat{L}(v)$ from the leaves up and on i. Also, it may help to strengthen the lemma and prove that if the ith item is in M_v, then (1) at the end of round $\hat{L}(v)+i-1$, it is stored at v, and (2) at the end of round $\hat{L}(v)+i$, v has already upcast this item to its parent.)

3. Devise downcast and upcast algorithms analogous to those in Section 4.1 and the unordered case of Section 4.2.3 for the asynchronous model, and analyze their time complexities.

4. The following policy is proposed for solving the "smallest k-of-m" problem of Section 4.3.1:

   - At any given moment along the execution, every vertex keeps only the set of k smallest elements that it knows.
   - In each step, each vertex sends to its parent an *arbitrary* element from this set that hasn't been sent yet.

   (a) Prove that using this policy, the k smallest elements must still arrive at the root by time $O(kDepth(T))$.

   (b) For every integer $m \geq 1$, give an example for a tree T and an initial distribution of m elements, where $k = Depth(T) = \lceil \sqrt{m} \rceil$ and the above process takes $\Omega(m)$ steps.

5. Consider a tree-shaped distributed network connecting a number of sites of a scientific organization. Suppose that the organization is interested in measuring k parameters and towards that end it employs measuring equipment at various sites in the network. Suppose that the measurements approximate the true parameters by bounding them from above and that we are interested in finding the most accurate approximation for each of the k parameters and collecting these approximations at the root.
Explain how this problem can be solved in $O(Depth(T)+k)$ time by fitting it into the framework of Section 3.4.3.

6. Prove Lemma 4.3.2.

7. Describe and analyze the token distribution algorithm of Section 4.3.4 (including both the preprocessing stage and the actual distribution stage), and prove Lemma 4.3.3.

Chapter 5

Tree constructions

In this chapter we discuss some basic constructions of various types of spanning trees, including *Breadth-First Search (BFS)* trees, *Depth-First Search (DFS)* trees and *Minimum-weight Spanning Trees (MST)*, and their distributed implementation.

5.1 BFS tree construction

The BFS tree of a given network with respect to a given root was defined in Section 3.2. Recall that in Section 3.3 we saw how broadcast through flooding can be used to define a spanning tree for the network. Moreover, by Lemma 3.3.2, in a synchronous network Algorithm FLOOD generates a BFS tree, and this is done with complexities Message(FLOOD) $= \Theta(|E|)$ and Time(FLOOD) $= \Theta(Diam(G))$. These complexities are asymptotically optimal for a clean network (in which vertices know nothing on the topology). This is implied by the following lower bounds on distributed BFS tree construction, which we state without proof. In Chapter 23 (Theorem 23.2.2), we will see a proof for a result similar to part 1 of the lemma concerning broadcast.

Lemma 5.1.1

1. *For every n-vertex graph $G = (V, E)$, distributed BFS tree construction in a clean network requires $\Omega(|E|)$ messages in the worst case.*

2. *For arbitrary n-vertex graphs, distributed BFS tree construction requires $\Omega(Diam)$ time in the worst case.*

In the asynchronous model, however, the tree generated by Algorithm FLOOD need not necessarily be a BFS tree. Hence in our discussion of BFS tree constructions, we shall concentrate on the asynchronous setting. As our main focus here is on the trade-offs between the time and message complexities, we will disallow large messages, i.e., concentrate on the asynchronous $\mathcal{CONGEST}$ model.

There are two basic sequential algorithms for computing a BFS tree for a given graph, known as the *Dijkstra* and *Bellman–Ford* algorithms. The two distributed algorithms we present next are in fact the respective distributed implementations of those two algorithms.

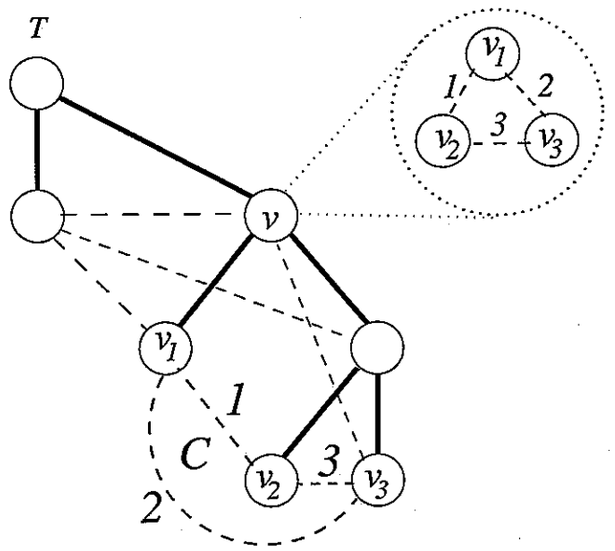

Figure 5.12: A cycle C identified locally by the vertex v. Bold edges belong to the tree T, and dashed lines represent graph edges outside T. The edge (v_2, v_3) is the heaviest on the cycle C, so v will avoid upcasting it to its parent in the tree T.

If the tree T is not available to us initially, it is possible to first construct such a tree efficiently, say, using Algorithm FLOOD of Section 3.3. This takes at most $O(Diam(G))$ additional time. Hence the resulting procedure PIPELINEMST satisfies the following.

Lemma 5.6.9 *Procedure* PIPELINEMST *constructs an MST, and its time complexity is* Time(PIPELINE(MST)) $= O(n)$. □

Bibliographical notes

The problem of designing an asynchronous distributed algorithm for breadth first search, introduced by Gallager in [Gal82], has been studied intensively since then [Gal82, Fre85, Awe85a, Awe85c, AG85b, AG87, Awe89]. The layer-synchronized BFS construction algorithm and the update-based BFS construction algorithm are distributed implementations of the sequential algorithms of Dijkstra and Bellman and Ford, respectively (cf. [Eve79, CLR90, Gal82]). A distributed implementation of the Bellman–Ford algorithm for the weighted case and a discussion of its execution time are presented in [BG92]. The best known distributed BFS algorithm is due to [AP90b, AR92]. The time-efficient distributed DFS algorithm of Section 5.4 was introduced in [Awe85b]. The label assignment problem of Exercises 8 and 9 is treated in [FPPP00].

The MST problem, the "blue rule" and sequential algorithms for MST construction are described in most textbooks on algorithms, e.g., [CLR90, Tar83]. Kruskal's algorithm for constructing an MST is originated in [Kru56], cf. [CLR90]. The distributed MST construction algorithm GHS presented in Section 5.5 is a simplified version of the renowned algorithm of [GHS83]. Lemma 5.5.4 is also due to [GHS83]. The original GHS algorithm of [GHS83] applies also to asynchronous networks and has optimal message complexity (unlike the variant described here). The time complexity of the algorithm of [GHS83] is $O(n \log n)$, which was later improved to the existentially optimal $O(n)$ in [Awe87]. We refer the reader

to [GHS83, Lyn95, AW98] for a more detailed description and discussion of the complete GHS algorithm.

The distributed algorithm of Section 5.6 for solving matroid problems, due to [Pel98], is an extension of a procedure appearing in [GKP98, KP98b] as a component in a fast distributed algorithm for computing an MST. For more on the subject of matroids see, e.g., [PS82]. Greedoids were thoroughly treated in [KL81, KL83, KL84b, KL84a]. The "red rule" and Lemma 5.6.2 appear, e.g., in [Tar83].

Exercises

1. Prove that in phase p of Dijkstra's algorithm, at most two (respectively, four) messages are sent over each edge of $E_{p,p+1}$ (resp., E_p). Show how the algorithm can be modified so that at most two messages are sent over each edge of E_p as well.

2. Prove the tightness of the message complexity analysis of Dijkstra's algorithm by establishing the following (existential) lower bound: For arbitrary integers n and D, $2 \leq D \leq n-1$, there exists an n-vertex, D-diameter graph $G = (V, E)$ on which the execution of Dijkstra's algorithm requires $\Omega(nD + |E|)$ messages.

3. Repeat the previous question with the additional requirement that the graph G has $|E| = O(n)$ edges.

4. Prove or disprove the following global lower bounds:

   For every n-vertex, D-diameter graph $G = (V, E)$, there exists an execution of Dijkstra's algorithm requiring

   (a) $\Omega(nD)$ messages,

   (b) $\Omega(|E|)$ messages,

   (c) $\Omega(D^2)$ time.

5. Give an example for an execution of the Bellman–Ford algorithm requiring $\Omega(n^3)$ messages.

6. Modify the Bellman–Ford algorithm so that it detects termination.

7. (a) Describe a distributed algorithm based on DFS for counting the number of vertices in the network $G = (V, E)$ and informing the outcome to all the vertices of G. The algorithm should function correctly even when invoked by a number of initiators.

   (b) What are the time and message complexities of your algorithm assuming it was invoked by K initiators?

   (c) Does the algorithm work in the asynchronous model?

8. Describe a distributed algorithm based on DFS that when invoked by a single initiator on an n-vertex anonymous network $G = (V, E)$, assigns unique labels from the range $[1, n]$ to the vertices of G.

9. In the synchronous model, give a faster ($O(Diam(G))$ time) algorithm for the label assignment problem of the previous problem.

10. Suppose that the network $G = (V, E, \omega)$ has unique vertex identifiers but the weights of different edges might be identical. Describe an algorithm allowing each vertex in the network to locally assign a new edge weight $\omega'(e)$ to each of its adjacent edges, based on its original weight and the identifiers of its endpoints, so that the resulting edge weights are distinct and consistent (i.e., the weights assigned to the edge $e = (u, w)$ by u and by w are the same).

11. Give an $O(n)$ time distributed algorithm for MST construction on an n-vertex weighted ring with a single initiator.

12. Prove Lemma 5.5.7.

13. Explain how Exercise 5 of Chapter 4 can be solved in $O(Depth(T)+k)$ time by fitting it into the framework of Section 5.6.

14. (a) Analyze the message complexity of Procedure PIPELINE of Section 5.6.
    (b) What is the resulting bound on the message complexity of the MST construction Procedure PIPELINEMST of Section 5.6.4? Take into account also the need to first construct a spanning tree for the network.

Chapter 6

Synchronizers

Algorithms for synchronous networks are easier to design, debug and test than similar algorithms for asynchronous networks. The behavior of asynchronous systems is typically harder to grasp and analyze. Consequently, it is desirable to have a uniform methodology for transforming an algorithm for synchronous networks into an algorithm for asynchronous networks. This tool will enable one to design an algorithm for a synchronous network, test it and analyze it in that simpler environment and then use the standard methodology to transform the algorithm into an asynchronous one and use it in the asynchronous network.

This general approach for handling asynchrony is known as a *synchronizer*. We will discuss several specific methods for implementing a simulation of this type and look at their complexities. Naturally, throughout this chapter we concentrate on the $\mathcal{ASYNC}$ model.

One might argue that in order to achieve a fast asynchronous algorithm,[1] it is necessary to program it directly in the environment in which it is to be run (much the same as programming in low-level languages usually yields better performance than compiling a high-level program). However, the surprising fact that in spite of the inevitable overheads involved in such a simulation, asynchronous algorithms designed in this way are sometimes more efficient than any previously known. This is mainly due to the inherent difficulty in reasoning about an asynchronous network, which sometimes makes it hard to reach an optimal solution for a problem directly in such an environment.

6.1 The synchronizer methodology

6.1.1 Underlying simulation principles

The synchronizer is intended to enable any synchronous algorithm to run on any asynchronous network. More specifically, given an algorithm Π_S written for a synchronous network and a synchronizer ν, it is possible to *combine* Π_S on top of ν to yield a protocol $\Pi_A = \nu(\Pi_S)$ that can be executed on an asynchronous network. The simulation should be *correct*, in the sense that Π_A's execution in a synchronous network should be "similar" to Π_S's execution in a synchronous network. (A more precise definition is given later on.)

The combined protocol Π_A is composed of two main components, which we hereafter refer to as the "original component" and the "synchronization component." Each of these components has its own local variables and message types at every vertex. The "original

[1] Henceforth we will use the term "(a)synchronous algorithm" to indicate an algorithm written for operation in an (a)synchronous network.

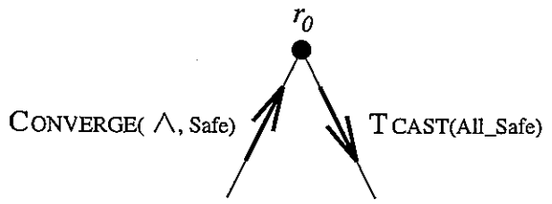

Figure 6.1: The two phases of synchronizer β.

Lemma 6.3.4

1. $\mathsf{Message}_{init}(\beta) = O(n|E|)$,
2. $\mathsf{Time}_{init}(\beta) = O(Diam(G))$,
3. $\mathsf{Message}_{pulse}(\beta) = O(n)$,
4. $\mathsf{Time}_{pulse}(\beta) = O(Diam(G))$.

Proof: To set up synchronizer β, it is necessary to construct a BFS tree in the network. This can be done using a number of distributed algorithms, for example, a distributed implementation of Dijkstra's algorithm (whose costs are Time $= O(Diam^2(G))$ and Message $= O(|E| + Diam(G)n)$) or the Bellman–Ford algorithm (whose costs are Time $= O(Diam(G))$ and Message $= O(n|E|)$).

The overhead parameters are simply the costs involved in a single "convergecast and broadcast" cycle on a tree of depth $O(Diam(G))$. □

Note that synchronizer β is optimal for bounded-diameter networks.

Bibliographical notes

The synchronization ideas behind synchronizers α and β were used implicitly in a number of distributed asynchronous algorithms, e.g., [Cha79, Jaf80, Gal82, CL85]. The general approach of using synchronizers as a unified methodology for handling asynchrony was first introduced in [Awe85a]. This paper also proposed synchronizers α and β. Alternative constructions were presented in [PU89b, AP90b]. Applications of synchronizers were studied in [Awe85c, SM86, LTC89, PU89b, AP90b].

The difficulties in the correctness of the simulation based on the readiness rule were pointed out in [LT87], and various corrections and correctness proofs were proposed in [LT87, FLS88, SS91]. The former two are based on the idea of message delaying.

Various complexity issues concerning synchronizers, including their time slowdown and memory overheads, were studied further in [CGZ86, AS88, ER88, ER90, ER95, RS94, SS93b, SS93a]. Synchronization in the presence of faults was discussed in [APSPS92, HS94].

Exercises

1. Prove Lemmas 6.1.4 and 6.1.7.
2. (a) Design a scenario realizing $\mathsf{Time}_{gap}(\alpha) = \Omega(Diam(G))$, i.e., in which some processor is forced to wait for time $\Omega(Diam(G))$ until it can increase its pulse number.
   (b) Prove that this is the worst possible case.

3. Establish the maximum possible value for $\mathsf{Time}_{gap}(\beta)$ and prove it.
4. Consider a 15-processor asynchronous network with processors $0, \ldots, 14$. The processors constantly run a synchronizer. Let v and v' be two processors in the network, and suppose that at a certain moment, the pulse counter at v shows $p = 27$. What is the range of possible pulse numbers at v' in each of the following cases:
   (a) The network is a ring (with the processors arranged according to their numbers), v is processor number 11, v' is processor number 2 and the synchronizer used is α.
   (b) The network is a full balanced binary tree (4 levels), v is the root, v' is one of the leaves and the synchronizer used is β.
   (c) The same as in (b), except both v and v' are leaves.
5. What are the message and time complexities of the asynchronous broadcast algorithms α(FLOOD) and β(FLOOD) resulting from combining the synchronous Algorithm FLOOD on top of synchronizers α and β, respectively?
6. Consider a model combining the $\mathcal{ASYNC}$ and $\mathcal{LOCAL}$ models, that is, with asynchronous communication but allowing arbitrarily large messages. Which synchronizer type is preferable in this model? Justify your answer.

Lemma 7.5.7

1. *$\tilde{B}_{1,n}$ is the complete directed graph on n vertices (with every two vertices connected by one arc in each direction).*
2. *$\tilde{B}_{s+1,n} = \mathcal{DL}(\tilde{B}_{s,n})$.*

Proof: The first claim is immediate from the definition. The second claim is proved in a straightforward manner by establishing an appropriate isomorphism between $\tilde{B}_{s+1,n}$ and $\mathcal{DL}(\tilde{B}_{s,n})$. Consider the vertex e of $\mathcal{DL}(\tilde{B}_{s,n})$. Recall that e is an arc of $\tilde{B}_{s,n}$, say, connecting $(x_1, \ldots, x_s)$ to $(x_2, \ldots, x_{s+1})$. Then map e to the vertex $(x_1, \ldots, x_s, x_{s+1})$ of $\tilde{B}_{s+1,n}$. It is now straightforward to verify that this mapping preserves the adjacency relation (see Figure 7.13). □

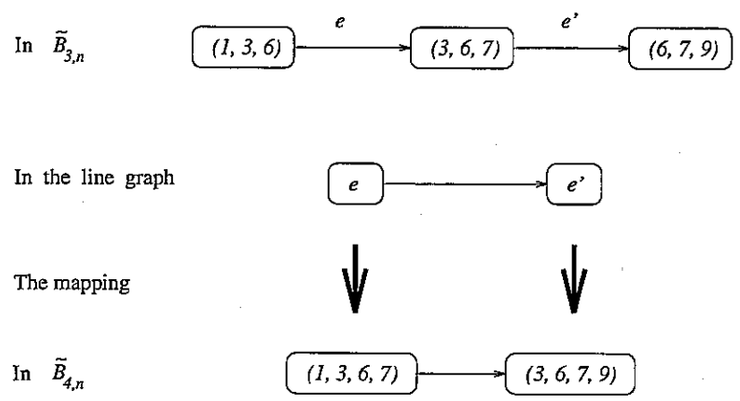

Figure 7.13: An example for the mapping from $\mathcal{DL}(\tilde{B}_{3,n})$ into $\tilde{B}_{4,n}$.

This relationship allows us to use the following lemma relating the chromatic number of a graph to that of its line graph.

Lemma 7.5.8 *For every directed graph H, $\chi(\mathcal{DL}(H)) \geq \log(\chi(H))$.*

Proof: Let $k = \chi(\mathcal{DL}(H))$, and consider a k-coloring $\hat{\varphi}$ of $\mathcal{DL}(H)$. The function $\hat{\varphi}$ can be thought of as an edge coloring for H, with the property that if e' starts at the vertex in which e ends, then $\hat{\varphi}(e') \neq \hat{\varphi}(e)$. Such a coloring can be used to create a 2^k-coloring $\tilde{\varphi}$ for H by setting the color of a vertex v to be the set $\varphi_v = \{\hat{\varphi}(e) \mid e \text{ ends in } v\}$.

Clearly, there are at most 2^k colors used by $\tilde{\varphi}$. To see that the coloring is legal, note that if v and w are neighbors in H, then there exists an edge e leading (without loss of generality) from v to w and then $\hat{\varphi}(e)$ occurs in φ_w but not in $\hat{\varphi}(v)$ (since its being in $\hat{\varphi}(v)$ as well would imply the existence of an edge e' entering v such that $\hat{\varphi}(e') = \hat{\varphi}(e)$, violating the legality of $\hat{\varphi}$ as an edge coloring of H).

It follows that $\chi(H) \leq 2^k$, proving the claim. □

Lemma 7.5.9 $\chi(\tilde{B}_{s,n}) \geq \log^{(s-1)} n$.

Proof: By straightforward induction on s, relying on Lemma 7.5.7 and Lemma 7.5.8. □

We can now complete the proof as follows.

Proof of Lemma 7.5.4: The proof is immediate from Lemmas 7.5.5 and 7.5.9. □

Theorem 7.5.10 *Any deterministic distributed algorithm for coloring the n-vertex ring with three colors requires at least $\frac{1}{2}(\log^* n - 1)$ rounds.*

Proof: By Lemma 7.5.4 and Corollary 7.5.2, if Π is an algorithm as stated in the theorem and requires t rounds, then $\log^{(2t)} n \leq \chi(B_{2t+1,n}) \leq 3$, hence $2t \geq \log^* n - 1$. □

Bibliographical notes

The 3-coloring algorithm of Section 7.3 is a generalization of the solution of [CV86] for simple chains, taken from [GP87] (see also [Plo88]). The $\Omega(\log^* n)$ lower bound for 3-coloring the ring (or computing an MIS on it) is due to [Lin92]. This lower bound was also extended to randomized algorithms in [Nao91, Lin92].

An algorithm for producing a Δ^2-coloring of an arbitrary graph of maximum degree Δ, yielding also a $(\Delta+1)$-coloring in time $O(\log^* n + \Delta^2)$, is presented in [Lin92]. The algorithm of Section 7.4 is an adaptation of the algorithm of [GPS88] presented in [AGLP89]. Results on distributed edge coloring can be found in [PS92a, PS95, DP95].

Exercises

1. Devise a coloring algorithm for trees which operates in one time unit and uses as few colors as possible.
2. Modify the 3-coloring algorithm of Section 7.3 into a 3-coloring algorithm for the ring.
3. How many vertices and edges does the graph $B_{s,n}$ of Section 7.5 contain?
4. Is the graph $\tilde{B}_{s,n}$ acyclic?
5. Let Π be a correct coloring algorithm for the n-vertex ring, which operates in one time unit. What lower bound can be deduced (from the results of Section 7.5) on the number of colors it must use?
6. What is the smallest n for which the lower bound of Theorem 7.5.10 on 3-coloring is greater than 1?