

**CONCURRENCY IN SYSTEMS WITH NEIGHBORHOOD
CONSTRAINTS**

Valmir Carneiro Barbosa

**September 1986
CSD-860035**

CONCURRENCY IN SYSTEMS WITH NEIGHBORHOOD CONSTRAINTS

Valmir Carneiro Barbosa

**September 1986
CSD-860035**

UNIVERSITY OF CALIFORNIA

Los Angeles

Concurrency in Systems
with Neighborhood Constraints

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science


by

Valmir Carneiro Barbosa


1986

© Copyright by
Valmir Carneiro Barbosa
1986

The dissertation of Valmir Carneiro Barbosa is approved.



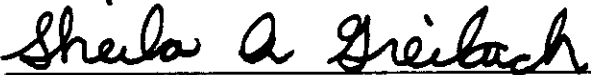
Bruce L. Rothschild



Kirby A. Baker



Judea Pearl



Sheila A. Greibach



Eliezer M. Gajni, Committee Chair

University of California, Los Angeles

1986

This dissertation is dedicated to Alzira, my wife.

TABLE OF CONTENTS

	page
1. INTRODUCTION	1
1.1. Neighborhood Constraints and Concurrency	1
1.2. Concurrent Processes and Resource Sharing	3
1.3. Concurrent Processes and Optimization	4
1.4. An Overview of the Dissertation	8
2. BACKGROUND AND PRELIMINARY RESULTS	12
2.1. Introduction	12
2.2. Simulated Annealing	12
2.2.1. The Method	12
2.2.2. Functions Generating Sparse Graphs	14
2.2.3. An Example: Satisfiability	15
2.2.4. More Examples: Graph Problems	17
2.2.5. A Generic Formulation	22
2.2.6. Amenability to Speedup in Graph Problems	24
2.3. Graph-Theoretic Background	26
2.3.1. Acyclic Orientations	26
2.3.2. Node Multicolorings	27
2.3.3. Sink Decompositions	29
2.4. Scheduling by Edge Reversal	30
3. EDGE REVERSAL IN A SYNCHRONOUS MODEL	34
3.1. Introduction	34
3.2. Orientation-Transition Diagrams	35
3.3. Synchronous Schedules	36
3.3.1. General and Greedy Schedules	36
3.3.2. A Stronger Form of Starvation-Freedom	37
3.3.3. A Comparison between General and Greedy Schedules	38
3.4. Structure of the Orientation-Transition Diagram	39
3.4.1. Reachability Equations	39
3.4.2. Strongly Connected Components	41
3.4.3. Properties under Greedy Scheduling	42
3.4.3.1. Periodicity	42
3.4.3.2. Evolution of Sink Decompositions	45
3.5. A Study of Periodic Behavior	45
3.5.1. Periodicity and Node Multicolorings	45
3.5.2. Periods in which Nodes Operate Exactly Once	48
3.5.3. Periodicity in Trees, Cycles, and Complete Graphs	50
3.5.3.1. Trees	50
3.5.3.2. Cycles	52
3.5.3.3. Complete Graphs	58
3.5.4. Periods in which Nodes Operate More than Once	58
4. CONCURRENCY MEASURES	60
4.1. Introduction	60

4.2. The Concurrency of a Schedule	61
4.3. The Concurrency of an Orientation	62
4.3.1. The Concurrency of Greedy Schedules	62
4.3.2. A Closed-Form Expression	64
4.3.3. The Length of a Period	72
4.4. The Concurrency of a Graph	73
4.4.1. The Optimality of Scheduling by Edge Reversal	73
4.4.2. Concurrency, Chromatic and Multichromatic Numbers	74
4.4.3. The Concurrency of Bipartite Graphs, Cycles, and Complete Graphs	75
4.4.4. Concurrency and Multichromatic Number May Be Distinct	78
4.4.5. Intractability of Maximizing Concurrency	79
5. EDGE REVERSAL IN AN ASYNCHRONOUS MODEL	84
5.1. Introduction	84
5.2. Events and Global States: the General Case	85
5.2.1. Events and Orders	85
5.2.2. Global States	86
5.2.3. Precedence Graphs	87
5.3. Events and Global States: the Case of Edge Reversal	89
5.3.1. Events, Orders, and Global States	89
5.3.2. A Modified Precedence Graph	91
5.3.3. Executions and the Timing of Events	92
5.3.4. Greedy Synchronous Operation	94
5.3.5. A Measure of Concurrency	95
6. THE PROBLEM OF IDENTIFYING OPTIMAL GLOBAL STATES	98
6.1. Introduction	98
6.2. Statement of the Problem	99
6.3. Min-Flow Formulation	100
6.4. The Problem under Scheduling by Edge Reversal	103
6.4.1. Min-Flow Formulation	103
6.4.2. Max-Flow Formulation	104
6.4.3. An Example: Distributed Simulated Annealing	105
6.4.4. Centralized Implementation	109
7. DIRECTIONS FOR FURTHER RESEARCH	113
REFERENCES	117

LIST OF FIGURES

	page
Figure 2.1. Graph G Corresponding to the Formula $A_1 \wedge \cdots \wedge A_5$	18
Figure 2.2. Graphs G' and G for the Minimum Dominating Set Problem	21
Figure 2.3. $G[K_2]$, G the 5-Node Cycle	28
Figure 2.4. Example of a Sink Decomposition	31
Figure 2.5. Examples of Edge Reversal	33
Figure 3.1. Generic Structure of a Connected Component in OTD_g	43
Figure 3.2. A Connected Component in OTD_g and in OTD	44
Figure 3.3. Periodic Orientations and Associated Multicolorings	49
Figure 3.4. Types of Oriented Arcs in a Cycle	53
Figure 3.5. A cw -Uniform Orientation	54
Figure 3.6. Arc Drifting under Greedy Scheduling	55
Figure 3.7. A 5-Periodic Orientation	57
Figure 4.1. Used in the Proof of Theorem 4.7	70
Figure 4.2. A 3-Cube and a 5-Cube	77
Figure 4.3. Optimal Orientations of a 6-Node and a 5-Node Cycle	78
Figure 4.4. An Example for which $[\chi(G)]^{-1} < \gamma^*(G) < [\chi^*(G)]^{-1}$	80
Figure 5.1. Precedence Graph for a Generic Asynchronous Computation	89
Figure 5.2. Precedence Graphs under Scheduling by Edge Reversal	93
Figure 6.1. Graph H for a Generic Asynchronous Computation	102
Figure 6.2. Graphs PG' , H' and $\overline{H'}$	106

ACKNOWLEDGEMENTS

I wish to thank the members of my Doctoral Committee, consisting of Professors Eliezer M. Gafni, Sheila A. Greibach, Judea Pearl, Kirby A. Baker, and Bruce L. Rothschild. In particular, I thank Eli Gafni, my dissertation adviser, for his help in devising the topic of this dissertation and throughout the work.

During my four years of graduate study at UCLA, I was supported by a scholarship from CAPES, Ministry of Education, Brazil, under grant 3638/81-4. I am thankful to them for the promptness and dependability of their support. Some additional support was provided by Xerox Corporation under grant W850813.

Many people have been of great help, in a way or another, during my stay here. Our families, mine and my wife's, back in Brazil, have been extremely supportive, and I thank them for that. I would also like to single out my officemate John M. Marberg, for his patient review of the manuscript, and my Brazilian friends José Nagib C. Arabe, José D. P. Rolim, and Frank A. Schaffa, for the companionship.

Most of all, I thank Alzira M. Barbosa, my wife, for all her love and encouragement. Things would have been much harder, were it not for her constant support for all this time. This dissertation is warmly dedicated to her. I must also remember Leonardo, who has been with us for over one year, for being such a great boy.

VITA

February 27, 1958	Born, Rio de Janeiro, Brazil
1980	Electronics Engineer, Universidade Federal do Rio de Janeiro, Brazil
1981-1982	Research Assistant, Centro de Pesquisas de Energia Elétrica, Brazil
1982	Master of Science in Computer Science, Universidade Federal do Rio de Janeiro, Brazil
1982	Research Engineer, Centro de Pesquisas de Energia Elétrica, Brazil
1984-present	Post-Graduate Research Engineer, University of California, Los Angeles

ABSTRACT OF THE DISSERTATION

Concurrency in Systems
with Neighborhood Constraints

by

Valmir Carneiro Barbosa

Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1986

Professor Eliezer M. Gafni, Chair

A collection of processes is considered, and each two of them are classified as either neighbors or nonneighbors, according to a symmetric neighborhood relation. This system can be represented by an undirected graph G whose set of nodes is determined by the set of processes and set of edges by the neighborhood relation. Neighborhood is meant to stand for resource sharing of some type, and therefore we restrict ourselves to considering computations in which neighboring processes are precluded from being concurrently active. This dissertation contains studies on this type of system when the processes are scheduled for operation according to a scheme that we refer to as Scheduling by Edge Reversal. This scheme is derived from similar techniques previously employed in other contexts, and is based on the manipulation of acyclic orientations of G .

We present an analysis of Scheduling by Edge Reversal under synchronous and asynchronous models of computation. This analysis includes studies of the graph structures involved and definitions of concurrency measures. The chief concurrency

measure that we analyze is a number depicting a property of G . This number is related to G 's chromatic and multichromatic numbers, being in some cases distinct from both of them. We show, in addition, that the problem of maximizing concurrency is *NP*-complete.

A second contribution is the definition and solution of a problem on the precedence graphs generated by general asynchronous computations. This is the problem of identifying an optimal global state, according to a criterion that we specify. We show that this problem can be solved through the use of Max-Flow techniques on variations of the precedence graph, despite the known *NP*-completeness of similar problems on precedence systems. The special case in which the precedence graph is generated by Scheduling by Edge Reversal admits efficient implementations.

A combination of the previous two contributions yields a proposal to implement the Simulated Annealing method distributedly for some types of functions. The approach is applicable to approximating the solution to a number of *NP*-complete problems, as for example the problem of finding a maximum independent set of a graph.

CHAPTER 1

INTRODUCTION

1.1. Neighborhood Constraints and Concurrency

Many modern computer systems can be regarded as collections of processes that cooperate, and compete for resources, with one another. Until some years ago, this view of a computer system was best exemplified by the so-called multiprogrammed systems, which consist basically of a computer being shared by more than one user, each user having associated with it one or more processes. In systems like this, processes compete for the resources of the single, centralized computer. Recent years, however, have witnessed a trend towards parallelizing and distributing computer systems, giving rise to multiprocessor systems. Among the reasons generally cited for such a tendency, one finds the speedup of computations, cost-effectiveness, and reliability. In parallel and distributed systems, resources for which the various processes compete may be scattered throughout the system. More than one processing unit is available, and these units communicate with one another either through the access to shared memory cells (parallel systems), or through messages sent over communication channels (distributed systems).

Resources are typically scarce in computer systems, having therefore to be shared by the various processes. The design of systems in which processes are allowed to operate concurrently requires that we establish rules whereby this sharing of resources is to take place. In this respect, one assumption commonly made is that,

given a resource, only a predefined set of operations can be applied to it, and, moreover, no two processes may have simultaneous access to it. One can, through this orderly use of shared resources, ensure that their integrity is preserved, and that the outcome of deterministic computations is reproducible.

In this dissertation, we consider a collection of processes, and together with it a neighborhood relation that classifies each two processes as either neighbors or non-neighbors. This neighborhood relation is taken to mean resource sharing, and therefore no two neighboring processes may be concurrently using a resource that they share. Clearly, the level of concurrency that such a system is capable of providing depends very intimately on how dense the neighborhood relation on its processes is. This dissertation contains a study of the interplay between these two concepts, namely concurrency and a process' being constrained by its neighborhood to use the resources that it needs in order to operate. Throughout this work, a system of the neighborhood-constrained type described above will be represented by a connected undirected graph $G=(N,E)$. In G , each node corresponds to a process and each edge to a resource shared by two processes. We let $|N|$ be denoted by n .

Though the neighborhood-constrained model of concurrent computation described above can represent a wide variety of systems, the details of an actual model may vary considerably from system to system. For the sake of making the exposition throughout the dissertation more uniform, we assume a distributed system in which each process resides in a distinct processor, and processors communicate through messages sent over point-to-point communication channels. One such channel connects two processors if the corresponding processes share a resource. We shall, in the sequel, refer to G 's nodes, processes, and processors without distinction.

The following is how the remaining sections of this chapter are organized. Section 1.2 contains a description of heavily loaded neighborhood-constrained systems in terms of the Dining Philosophers Problem of Dijkstra. In Section 1.3, we describe the instance of such systems which originally motivated this work. The problem we consider is that of implementing distributedly the Simulated Annealing method to approximate the optimum of functions with a special form. Section 1.4 contains an overview of the remainder of the dissertation.

1.2. Concurrent Processes and Resource Sharing

In this section we use a generalization of Dijkstra's Dining Philosophers Problem as an abstraction of the neighborhood-constrained type of systems introduced in the previous section.

The Dining Philosophers Problem was introduced by Dijkstra in [Dijk71]. Solutions to this problem, and variations thereof, can be found in [Chan80, Lync80, Lync81, Rabi81, Chan84]. Here we consider the generalization addressed in [Chan84]. According to this generalization, a set of Philosophers is given, and each Philosopher may share a fork with one or more of the others. We consider the following particular form of the problem. Given that a Philosopher is always hungry, and that he needs all of his forks in order to eat, schedule Philosophers for eating such that deadlock- and starvation-freedom are ensured. We may look at each Philosopher as being represented by a process, and at each fork as being a shared resource.

This form of the Dining Philosophers Problem can be thought of as modeling resource sharing systems in which all resources are under heavy demand. Readily,

such situations of heavy demand provide an ideal framework to study a system's capability of supporting concurrent processing. In this sense, the problem of scheduling Philosophers for eating under the constraints and requirements outlined above is in fact the problem of scheduling processes to operate on shared resources in heavily loaded systems. The particular scheme we will employ, described in the next chapter, requires that a Philosopher who has just eaten will not eat again before all other Philosophers with whom he shares forks have eaten as well. It is deadlock- and starvation-free, thus meeting the requirements for a solution to the problem.

1.3. Concurrent Processes and Optimization

In this section we describe a distributed implementation of the Simulated Annealing method which can be viewed as an instance of neighborhood-constrained systems. Here we only present the method's characteristics which are essential to the understanding of the distributed implementation. In Chapter 2, we present more details, and elaborate on some examples.

Let $X = \{X_1, \dots, X_n\}$ be a set of n variables taking values from a common set D , and FS the set of feasible points in D^n . The problem we address in this section is that of finding a global minimum of a function f of the form

$$f : D^n \rightarrow R$$

on the feasible set FS .

The problem of finding a global minimum of f on FS is typically a hard one, especially if no simplifying characteristics are assumed, such as the convexity of f , etc. The reason for this is that most iterative methods to solve it will at some point get stuck in local minima, sometimes far from a global optimum. Also, many well-

known combinatorial problems, *NP*-complete in their decision-problem formulations [Gare79], can be expressed as optimization problems. With respect to such problems, there is a widespread feeling that no efficient solution method exists, which stands as an incentive to the search for alternative techniques.

The application of a technique known as *Simulated Annealing* to approximate the solution to the problem of finding a global minimum of f over FS was recently proposed by Kirkpatrick *et al.* in [Kirk83]. Simulated Annealing originates from a similar technique used by Metropolis *et al.* to simulate the behavior of many-body physical systems at a fixed temperature [Metr53]. Its name originates from the laboratory procedure known to physicists as *annealing*, whose purpose is to bring a material to its ground energy states through its slow cooling from a relatively high temperature. This is a delicate process, since there exists the risk of ending up at poor states of locally minimum energy, if the cooling process is not sufficiently slow. By analogy with this physical process, the proposal in [Kirk83] amounts essentially to performing a stochastic search on FS so to resemble the behavior of the microscopic constituents of the material being cooled. Their proposal is to jump probabilistically from point to point in FS , therefore allowing occasional uphill moves, i.e. moves in which the value of f increases. Such jumps are to be governed by the Boltzmann distribution parameterized by a slowly decreasing temperature-like parameter, still in analogy with the dynamics of the physical system.

Let the function f be written as

$$f(x) = \sum_{Y \subset X} g_Y(x),$$

for all $x \in D^n$, where $g_Y(x)$ depends only on those coordinates of x corresponding to

variables in Y . The case of interest to us is the one in which the subsets Y of X such that $g_Y(x)$ is nonconstant are not “too large.” The application of the Simulated Annealing method to functions like this is discussed in [Gema84], where the unconstrained minimization of f is considered, i.e. the problem of finding $x^* \in D^n$ such that $f(x^*) \leq f(x)$ for all $x \in D^n$. In order to guarantee that all global optima will be in FS , one can incorporate penalty functions into f [Luen73].

For functions like this, the stochastic search is such that the decision to change the value of a variable, say X_i , depends only on the values of those variables X_j such that there is a subset $Y \subseteq X$ of which both X_i and X_j are members and $g_Y(x)$ is nonconstant. We say that two such variables are neighbors of each other. Because of the special form of f , the change caused to it by a change in the value of a variable depends only on that variable and on its neighbors. This type of “local” dependence of one variable’s value upon another’s hints at a possible speedup of the Simulated Annealing method via a distributed implementation. In this implementation, n processors are available, each one being responsible for the updating of one of the variables in X . Processors communicate through messages sent over point-to-point communication channels laid down between processors whose variables are neighbors. When a variable is updated based on the values of its neighbors, its new value is sent over to them through these channels. Clearly, variables which are not neighbors can have their values updated concurrently, thus the speedup in the simulation.

Recall that this system can be represented by the graph G introduced earlier. We make the observation that this graph is such that it contains a clique, i.e. a complete subgraph, for each subset Y of X such that $g_Y(x)$ is nonconstant. Furthermore, the form of f implies that G is somewhat sparse, since no clique involves a

significantly large portion of G .

A convergence proof for this variation of the Simulated Annealing method is found in [Gema84]. It is shown that convergence can be ensured if all variables are updated infinitely often and the parameter T is reduced no faster than a function of time that the authors specify. The proof also requires that variables be updated based on the current values of their neighbors, thus the need to avoid the concurrent updating of neighboring variables. In order to see why the updating of variables based on obsolete information has to be precluded, consider the variable X_i and one of its neighbors X_j . Suppose that at a certain point in time they exchange information about their current values, and then operate concurrently based on that information. As we will see in Chapter 2, the value of X_i is changed based on a probability distribution which is conditional on the value of its neighbors. If at the time its value is changed the value of X_j , say, is no longer the one on which the decision was based, the system may be found in a state whose probability is not in accordance with the probabilities used to do the updating.

The problem we now face is to come up with a distributed scheme to schedule variables for updating which ensures that the conditions required for convergence are met. The scheduling scheme used to schedule Philosophers for eating in the previous section can be employed here, too. According to this scheme, once a variable has been updated, it can only get updated again after all of its neighbors have been updated as well. Since it is deadlock- and starvation-free, it satisfies all the requirements for convergence.

Another problem is that of identifying the point of optimal value found during the simulation. Because Simulated Annealing performs a stochastic search, it is not necessary that such a point will be the one at which the simulation ends. It may have occurred at an earlier stage in the computation, instead. Furthermore, each variable is now placed in a different processor, its value constituting the local state of that processor. Therefore, the problem of identifying the best point found by the simulation amounts to finding a global state of the system, in the sense described in [Lamp78, Chan85], at which the function attained the optimal value during the simulation.

These two problems constitute the main issues dealt with in this dissertation.

1.4. An Overview of the Dissertation

A brief summary of the contributions of this dissertation follows.

- (i) Analysis of a distributed scheme to schedule processes for operation under the neighborhood constraints described above. This analysis includes definitions of concurrency measures, and a proof that the decision problem corresponding to optimizing the amount of concurrency for a given system is *NP*-complete. Such an amount of concurrency has an interesting graph-theoretic interpretation, and is related to G 's chromatic and multichromatic numbers.
- (ii) Solution of the problem of identifying optimal global states described above. We propose a Max-Flow solution on precedence graphs for the case of a generic asynchronous computation, and specialize this solution to the case of the scheduling referred to in item (i).

- (iii) A combination of items (i) and (ii) above yields a proposal for implementing the Simulated Annealing optimization method distributedly for the particular class of functions mentioned previously. In this proposal, the scheduling scheme in (i) is used to schedule processors for updating their variables, and the Max-Flow solution to the problem in (ii) allows the optimum found to be retrieved.

We present below a description of Chapters 2 through 7 of this dissertation.

We start in Chapter 2 by presenting a more quantitative description of Simulated Annealing, including its specialization to the class of functions discussed above. In doing so, we elaborate on some examples, and establish some conditions for some *NP*-complete graph problems to be cast in the formulation amenable to speedup via concurrent execution. In Chapter 2, we also review the graph-theoretic concepts that we need in the sequel. Particularly, we review node multicolorings and sink decompositions of a graph's node set under an acyclic orientation of its edges. We end Chapter 2 by presenting the scheduling scheme we employ throughout, which operates on acyclic orientations of G . We refer to it as Scheduling by Edge Reversal, since it consists of having the orientation of all edges incident to a sink in G reversed, when that sink operates, starting at an initial acyclic orientation. This scheme is derived from similar techniques used by Gafni and Bertsekas to generate loop-free routes in networks with time-varying topology [Gafn81], and by Chandy and Misra to coordinate the eating of their generalized Dining Philosophers [Chan84].

Chapter 3 contains a study of Scheduling by Edge Reversal in a synchronous model of computation. Under the synchronous model, processes operate in lockstep, and our scheduling can be viewed as the evolution in time of acyclic orientations of G , such that from one orientation to the next at least one sink becomes a source. We refer to the set of acyclic orientations of G , together with the mapping function from orientation to orientation as dictated by the scheduling, as the Orientation-Transition Diagram. A detailed study of this diagram is contained in Chapter 3. Besides presenting general properties, which include the relation of Scheduling by Edge Reversal to node multicolorings, we discuss the special structure of the Orientation-Transition Diagram when G is a tree, a cycle, or a complete graph.

Chapter 4 contains concurrency measures for the synchronous model. We define the concurrency of a schedule to be the average number of times that nodes operate over the entire schedule. The concurrency of an orientation is then defined as the maximum concurrency attainable from that orientation. A closed-form expression is found for this measure in terms of the orientations of simple cycles in the graph G . The concurrency of a graph is then defined to be the concurrency of the orientation that provides the most concurrency. This concurrency allows us to establish that Scheduling by Edge Reversal is optimal among all schedules which operate based on interleaved multicolorings. The problem of finding this number is then shown to be *NP*-complete. Oddly enough, the hardest part of this *NP*-completeness proof is to show membership in *NP*, which arises as a consequence of the closed-form expression for the concurrency of an orientation.

Chapter 5 is devoted to analyzing our scheduling scheme in an asynchronous model of computation. In this model, each process is run by an independent clock,

and therefore no global timing basis exists. In describing the asynchronous case, we use the concepts of events and global states found in the works of Lamport and Chandy [Lamp78, Chan85]. We first review these concepts for the case of a general asynchronous system, and then specialize the definitions for the case of asynchronous systems in which processes are scheduled for operation by Edge Reversal. A measure is then presented for the concurrency attainable under the asynchronous model, and it is shown to equal the concurrency that the synchronous model provides from the same initial conditions.

In Chapter 6, we describe, in terms of precedence graphs, the problem of identifying a global state of optimal value, where the value of a global state is the sum over the participating local states of real numbers associated with them. That this problem admits a polynomial-time solution is not immediately apparent, since similar problems on precedence systems are known to be *NP*-complete [Abde76]. We show that a formulation exists to this problem which allows it to be solved by finding the minimum flow in a slight variation of the original precedence graph, properly labeled. In the case of general asynchronous computations, this Min-Flow computation can be performed by a centralized processor in a time which is cubic in the number of events in the computation. Further in Chapter 6, we specialize the problem to the case of the precedence graphs generated by Edge Reversal. We then show that a Max-Flow formulation is also possible, and that the solution can be found more efficiently, due to the particular structure that these precedence graphs then have.

We present directions for further research in Chapter 7.

CHAPTER 2

BACKGROUND AND PRELIMINARY RESULTS

2.1. Introduction

This chapter contains all the needed background to the dissertation, along with a few preliminary results. Section 2.2 contains the description of a distributed implementation of the Simulated Annealing method. We first discuss Simulated Annealing in more detail, and then present necessary and sufficient conditions for some *NP*-complete graph problems to be cast in the formulation amenable to speedup via distributed processing. Section 2.3 contains some graph-theoretic background. We review the concepts of node multicolorings and of sink decompositions of a graph's node set under a given acyclic orientation of its edges. Finally, we present in Section 2.4 the distributed node scheduling scheme that we employ throughout this work, which we call Scheduling by Edge Reversal.

2.2. Simulated Annealing

2.2.1. The Method

Recall from Chapter 1 that $X=\{X_1, \dots, X_n\}$ is a set of n variables taking values from a common set D , and that FS is the set of feasible points in D^n . The problem is to find a point $x^* \in FS$ such that $f(x^*) \leq f(x)$ for all $x \in FS$, where f is of the form

$$f : D^n \rightarrow R .$$

A description of the Simulated Annealing technique more detailed than the one we presented before is the following. Let $x \in FS$ be the point at which the simulation is currently found and let $x' \in FS$ be a point obtainable from x by changing one of the variables in X . A move to x' is accepted with probability 1 if $f(x') \leq f(x)$, and with probability

$$e^{-\frac{f(x') - f(x)}{T}} ,$$

otherwise. Here T is a parameter meant to act like absolute temperature, i.e. it is slowly decreased from a relatively high initial value. It is intuitive that, at high values of T , a coarse-grained scan of FS is performed. As T is decreased, the acceptance of uphill moves becomes less and less probable, and therefore a fine-grained scan takes place.

Convergence proofs, variations, and discussions of other aspects of Simulated Annealing can be found in many places, including [Gema84, Gree86, Rome84b, Whit84, Gida85, Haje85, Mitr]. The method has been applied successfully to a number of hard combinatorial problems, especially in the field of VLSI, as reported in [Vecc83, Hema84, Otte84, Rome84a, Felt85]. A more thorough experimental evaluation of the Simulated Annealing capabilities is found in [Arag84], where some typical *NP*-complete combinatorial problems are examined. The approach behaves surprisingly well for some of these problems, whereas it cannot outrun the best existing heuristics for some others.

2.2.2. Functions Generating Sparse Graphs

Let the function f to be optimized be written as

$$f(x) = \sum_{Y \subset X} g_Y(x),$$

for all $x \in D^n$, where $g_Y(x)$ is nonconstant only if Y is not “too large,” and depends only on the coordinates of x corresponding to variables in Y . The application of the Simulated Annealing method to functions like this is discussed in [Gema84].

Define two variables to be *neighbors* of each other if they both belong to a subset Y of X such that $g_Y(x)$ is nonconstant. The stochastic search takes the following form. For $1 \leq i \leq n$, variable X_i is assigned value x_i with probability

$$\pi(X_i=x_i | X_j=x_j, j \neq i),$$

where π is the Boltzmann distribution, here given by

$$\pi(x) = \frac{1}{Z} e^{-\frac{1}{T} f(x)},$$

for all $x \in D^n$, Z being a normalizing constant and T our temperature-like parameter. Because of the form of f , it can be seen that

$$\pi(X_i=x_i | X_j=x_j, j \neq i) = \pi(X_i=x_i | X_j=x_j, X_j \text{ neighbor of } X_i).$$

The decision to set the value of X_i to x_i is then seen to depend only on the values of those variables X_j which are neighbors of X_i .

As we have seen previously, this system is of the neighborhood-constrained type we have been considering, which allows a distributed implementation by assigning each variable to a processor. In order for variable X_i to be updated, the current values of its neighbors have to be used, and for this reason X_i cannot be

updated concurrently with any of them. As we remarked in the previous chapter, the scheme we will employ to schedule nodes to update their variables is such that all neighbors of X_i get updated between two consecutive updates of X_i .

An observation to make is that the set of variables X , when regarded as a set of random variables, is said to be a *Markov Random Field (MRF)* with respect to the graph G , as a consequence of the strict positivity of π over D^n , and of the special form of the conditional probabilities discussed above [Rota64, Spit71, Besa74, Grif76, Kind80, Isha81]. MRFs represent a way of generalizing the Markovian dependence beyond the usual one-dimensional setting. The distributed version of the Simulated Annealing method we have described can be used, as long as T is kept constant, to update MRFs. In fact, there has been a proposal of a scheme to update MRFs concurrently [Berg], but it is specific to the case in which G is a bipartite graph.

In the next section we discuss in detail an *NP*-complete problem which can be cast in the form of minimizing a function with the special form we introduced in this section.

2.2.3. An Example: Satisfiability

The Satisfiability Problem is the problem of deciding whether a finite set of disjunctive clauses can be satisfied. The version of this problem that we consider here is the one in which each clause is restricted to have at most three literals. This decision problem is *NP*-complete, as seen in [Cook71, Gare79].

Formulated more precisely, we are given the set of n variables X introduced earlier, in this case taking values from the common set $D = \{TRUE, FALSE\}$. On this set of variables, r disjunctive clauses A_1, \dots, A_r are defined. For $1 \leq j \leq r$, clause A_j is of the form

$$A_j = L_{j1} \vee L_{j2} \vee L_{j3},$$

i.e. it is the logical disjunction of the three literals L_{j1}, L_{j2}, L_{j3} . Each of these literals is either a variable X_i , for some $1 \leq i \leq n$, or its negation, or the constant value *FALSE*. The decision to be made is whether the formula

$$A_1 \wedge \dots \wedge A_r$$

can be satisfied, i.e. made *TRUE* by some assignment of values to the variables in X .

The following is a formulation of this problem that matches the template introduced in the previous section. Consider the function

$$f(x) = \sum_{Y \subseteq X} g_Y(x),$$

where $g_Y(x) = c$, $0 \leq c \leq r$, for all $Y \subseteq X$ such that c of the clauses A_1, \dots, A_r depend on exactly the variables in Y and those same c clauses are *TRUE* under the assignment x . One can see that u clauses can be simultaneously satisfied if and only if there exists a point $x \in \{TRUE, FALSE\}^n$ for which $f(x) = u$. Thus a decision can be reached by finding a point, say x^* , at which $-f$ is minimum, and then checking whether $f(x^*) = r$. The unconstrained formulation of the previous section causes no problem here, since in this case $FS = D^n$.

As an example, consider the formula $A_1 \wedge \dots \wedge A_5$, given by

$$(X_1 \vee X_3) \wedge (\overline{X_1} \vee X_3 \vee X_4) \wedge (\overline{X_3} \vee X_5) \wedge (X_2 \vee X_4) \wedge (\overline{X_2} \vee X_4).$$

This formula is satisfiable if and only if the function f given by the sum of the functions below is such that $-f$ has a global minimum of value $r=-5$:

$$g_{\{X_1, X_3\}}(x) = \begin{cases} 1, & \text{if } A_1 \text{ is } TRUE \text{ under assignment } x, \\ 0, & \text{otherwise;} \end{cases}$$

$$g_{\{X_1, X_3, X_4\}}(x) = \begin{cases} 1, & \text{if } A_2 \text{ is } TRUE \text{ under assignment } x, \\ 0, & \text{otherwise;} \end{cases}$$

$$g_{\{X_3, X_5\}}(x) = \begin{cases} 1, & \text{if } A_3 \text{ is } TRUE \text{ under assignment } x, \\ 0, & \text{otherwise;} \end{cases}$$

$$g_{\{X_2, X_4\}}(x) = \begin{cases} 2, & \text{if both } A_4 \text{ and } A_5 \text{ are } TRUE \text{ under assignment } x, \\ 1, & \text{if only one of } A_4 \text{ and } A_5 \text{ is } TRUE \text{ under assignment } x. \end{cases}$$

Figure 2.1 shows the graph G corresponding to this problem. It should be recalled that, in G , node i corresponds to variable X_i , $1 \leq i \leq 5$. Notice that G contains a clique for each of the five clauses in the formula.

In the next section we turn to graph problems which can be cast in our special formulation. Unlike the Satisfiability Problem we just discussed, all the graph problems we consider require the incorporation of penalty functions into the objective function, in order to guarantee that all global optima are found inside $FS \neq D^n$.

2.2.4. More Examples: Graph Problems

In this section we consider an undirected graph $G'=(N',E')$, and we assume that $|N'|=n$. With each node in N' we associate one of the variables in X , which, in the cases considered here, take values from $D=\{0,1\}$. If we recall that N is G 's node

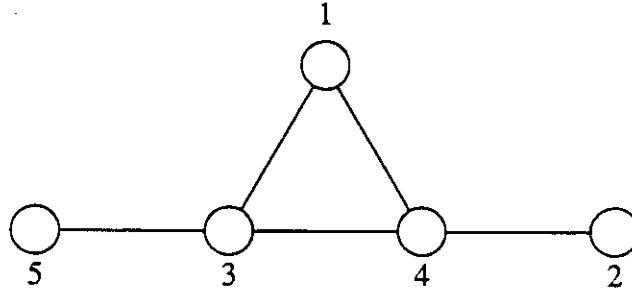


Figure 2.1. Graph G Corresponding to the Formula $A_1 \wedge \cdots \wedge A_5$

set, then N' and N are the same set. Below we present three optimization problems on G' . All of them are *NP*-complete in their decision-problem form [Karp72, Gare79]. For each problem, we present its statement, the set *FS* of feasible solutions, its formulation as an unconstrained minimization problem of the form described previously, and how the graph G corresponding to this formulation is obtained from G' . In presenting the formulation, we specify the functions $g_Y(x)$ which are nonconstant in the expression $f(x) = \sum_{Y \subseteq X} g_Y(x)$. In each of the formulations, penalty functions have been included to ensure that all global optima on D^n are in *FS*.

Minimum Dominating Set Problem:

- (i) **Statement:** Find a subset $I \subseteq N'$ with the properties that (a) each node $i \in N'$ is such that either $i \in I$ or there exists a node $j \in I$ such that $(i, j) \in E'$, and (b) no other subset of N' for which property (a) holds

has cardinality smaller than $|I|$.

(ii) $FS = \{x \in D^n \mid \text{if } X_i=0, \text{ then } X_j=1 \text{ for some } (i,j) \in E', 1 \leq i, j \leq n\}$.

(iii) Formulation:

For all $i \in N'$:

$$g_{\{i\}}(x) = X_i ;$$

$$g_{\{i\} \cup \{j \mid (i,j) \in E'\}}(x) = \begin{cases} 2, & \text{if } X_i=X_j=0 \text{ for all } (i,j) \in E' , \\ 0, & \text{otherwise .} \end{cases}$$

(iv) $E = E' \cup \{(i,j) \mid \text{there is a node } k \in N' \text{ such that } (i,k), (j,k) \in E'\}$.

Minimum Vertex Cover Problem:

(i) Statement: Find a subset $I \subseteq N'$ with the properties that (a) no edge $(i,j) \in E'$ is such that $i, j \notin I$, and (b) no other subset of N' for which property (a) holds has cardinality smaller than $|I|$.

(ii) $FS = \{x \in D^n \mid \text{if } X_i=X_j=0, \text{ then } (i,j) \notin E', 1 \leq i, j \leq n\}$.

(iii) Formulation:

For all $i \in N'$:

$$g_{\{i\}}(x) = X_i ;$$

For all $(i,j) \in E'$:

$$g_{\{i,j\}}(x) = \begin{cases} 2, & \text{if } X_i=X_j=0 , \\ 0, & \text{otherwise .} \end{cases}$$

(iv) $G = G'$.

In these two problems, one sees that a minimum of f is achieved if and only if, at this minimum, there is a one-to-one correspondence between variables with value 1 and nodes in an optimal subset I of N' (a minimum dominating set or a minimum vertex cover). The value of f at the optimum is then $|I|$.

Maximum Independent Set Problem:

(i) Statement: Find a subset $I \subseteq N'$ with the properties that (a) no two nodes $i, j \in I$ are such that $(i, j) \in E'$, and (b) no other subset of N' for which property (a) holds has cardinality larger than $|I|$.

(ii) $FS = \{x \in D^n \mid \text{if } X_i = X_j = 0, \text{ then } (i, j) \notin E', 1 \leq i, j \leq n\}$.

(iii) Formulation:

For all $i \in N'$:

$$g_{\{i\}}(x) = X_i ;$$

For all $(i, j) \in E'$:

$$g_{\{i, j\}}(x) = \begin{cases} 2, & \text{if } X_i = X_j = 0, \\ 0, & \text{otherwise.} \end{cases}$$

(iv) $G = G'$.

In this problem, a minimum of f is achieved if and only if, at this minimum, there is a one-to-one correspondence between variables with value 0 and nodes in a maximum independent set I of N' . The value of f at the optimum is then $n - |I|$.

An interesting point to make is that not all of the examples given are such that $G = G'$. This means, in practice, that if G' is the graph corresponding to an actual

point-to-point network, and if we wish the optimization to be carried out on that same network, then its processors may require information which is more than one hop away in G' in order to operate. For the Minimum Dominating Set Problem, for instance, the updating of variable X_i requires the value of all variables X_j such that the shortest path in G' between nodes i and j is either 1 or 2 hops long. Figure 2.2 shows a graph in which solid edges belong to G' and to G , whereas dashed edges belong to G alone. To solve the Minimum Dominating Set Problem on G' through our distributed approach, a network with the interconnections shown in G would be needed, if information were to be exchanged only between neighbors. The processing on this network can, of course, be simulated on a network with the interconnections in G' . For such, information may have to be routed between nodes through paths which are more than one hop long.

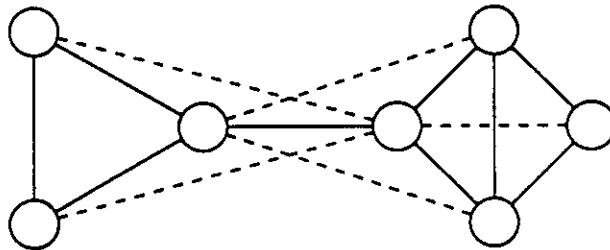


Figure 2.2. Graphs G' and G for the Minimum Dominating Set Problem

2.2.5. A Generic Formulation

In this section we specialize once again to the case $D = \{0, 1\}$, and assume that a point in FS will still be a member of FS if some variables with value 0 are assigned value 1. Consequently, the point in which all variables have value 1 is in FS . Our goal is to provide a general formulation to the problem of finding a point in $FS \subseteq \{0, 1\}^n$ in which the number of variables with value 1 is minimum. We first provide some definitions.

Let F be a function of the form

$$F : 2^X \rightarrow 2^{D^n},$$

i.e. a function that maps each subset Y of variables into a region in D^n . We define F as follows. For each $Y \subseteq X$, a point x' is a member of $F(Y)$ if and only if there exists a point $x \in FS$ such that x' and x have the same entries for the variables in Y . Clearly, $F(X) = FS$. The intuition behind this definition is that $F(Y)$ represents the set of points which would be feasible if only variables in Y were looked at. Put differently, we can make a member of $F(Y)$ feasible (i.e. a member of FS) by just changing the values of variables outside Y . A property of F is that

$$F(Y_1) \subseteq F(Y_2),$$

for all $Y_1, Y_2 \subseteq X$ such that $Y_2 \subseteq Y_1$.

Let a subset Y of X be called *minimal infeasible at x* if and only if $x \notin F(Y)$ and $x \in F(Y')$ for all $Y' \subset Y$.

Theorem 2.1 presents a formulation to the problem of finding a point in FS in which the number of variables with value 1 is minimum.

Theorem 2.1: Consider the function f given by the sum of the components below for all $x \in D^n$:

For all $X_i \in X$:

$$g_{(X_i)}(x) = X_i ;$$

For all subsets Y of X :

$$g_Y(x) = \begin{cases} 2, & \text{if } Y \text{ is minimal infeasible at } x, \\ 0, & \text{otherwise.} \end{cases}$$

Let x^* be a global minimum of f over D^n . This point x^* is such that no other point in FS contains a smaller number of variables with value 1 than it does.

Proof: It suffices to show that $x^* \in FS$, since for points in FS the value of f is the number of variables with value 1 in them. For suppose $x^* \notin FS$, we then show that x^* is not a global minimum of f over D^n .

If $x^* \notin FS$, then there must exist a subset Y of X which is minimal infeasible at x^* . Because the point in which all variables have value 1 is in FS , at least one variable in Y must have value 0 in x^* . By the definition of minimal infeasibility, if this variable is turned from 0 to 1, a point $x \in F(Y)$ is obtained. By doing this, f will not increase by more than 1. On the other hand, because x is a member of $F(Y)$, f will have decreased by 2. Since no other infeasibility may have been caused by this operation, f suffered from x^* to x a total decrease of 1, and therefore $f(x) < f(x^*)$. The process may be repeated as long as any minimal infeasible set at the point obtained remains, and in the end a point x^{**} will be obtained such that $f(x^{**}) < f(x^*)$, the required contradiction. ■

The three graph problems presented in the previous section are instances of the more general problem we have just considered. In the case of the Minimum Dominating Set Problem, a point $x \in D^n$ is infeasible if and only if the nodes corresponding to variables with value 1 in x do not constitute a dominating set. The same applies to the Minimum Vertex Cover Problem. The Maximum Independent Set Problem, on the other hand, is such that $x \in D^n$ is infeasible if and only if the nodes corresponding to variables with value 0 in x are not an independent set.

Notice that, in practice, if the Simulated Annealing method is used to minimize f , it will provide an approximation to the minimum of f , in which case Theorem 2.1 offers no assurance that the point thus found will be a member of FS . If x^* is such a point and $x^* \notin FS$, we can use the techniques described in the proof of Theorem 2.1 to obtain a point $x^{**} \in FS$ such that $f(x^{**}) < f(x^*)$. In other words, we will be improving on the outcome of the Simulated Annealing method by searching for a feasible point x^{**} obtainable from x^* .

In order that the formulation presented in the theorem be amenable to speedup by distributed processing, there must exist a constant $d \geq 0$ such that the diameter of the graph induced by a minimal infeasible set is no greater than d . In the case of the graph problems we discussed previously, this value is equal to 1 for the Minimum Vertex Cover Problem and the Maximum Independent Set Problem, whereas it equals 2 for the Minimum Dominating Set Problem.

2.2.6. Amenability to Speedup in Graph Problems

Our goal here is to investigate a necessary condition under which a graph problem can be solved on the graph G' introduced previously through a network

whose graph $G=(N,E)$ is such that $N=N'$ and $E=E' \cup \{(i,j) \mid d_{G'}(i,j) \leq d\}$, for some $d \geq 0$, where $d_{G'}(i,j)$ denotes the shortest distance, in terms of number of hops, between nodes i and j in G' . The practical implication of these conditions is that the optimization can be simulated on the very network to which G' corresponds, information having to be routed through paths which are no longer than d hops.

Theorem 2.2: Consider a problem on G' whose solution is given by the number $\eta(G')$. Suppose that $\eta(G')$ can be found by minimizing a function like f such that $g_Y(x)$ is constant if any two variables $X_i, X_j \in Y$ are such that $d_{G'}(i,j) > d$ for some $d \geq 0$. Let $G'=(N'_1 \cup N'_2, E'_1 \cup E'_2)$ be such that $G'_1=(N'_1, E'_1)$ and $G'_2=(N'_2, E'_2)$ are connected undirected graphs. Then it must be that $\eta(G') = \eta(G'_1) + \eta(G'_2)$.

Proof: Let X_1 and X_2 be the partitions of X that contain the variables corresponding to nodes in N'_1 and N'_2 , respectively. Since all subsets Y of X such that $Y \cap X_1 \neq \emptyset$ and $Y \cap X_2 \neq \emptyset$ yield $g_Y(x) \equiv 0$, we have

$$f(x) = \sum_{Y \subset X_1} g_Y(x) + \sum_{Y \subset X_2} g_Y(x),$$

for all $x \in D^n$. If we denote $\sum_{Y \subset X_k} g_Y(x)$ by $f_k(x)$, we see that the minimization of $f_k(x)$

solves the problem on G'_k , $k \in \{1, 2\}$. In other words, we must have

$$\eta(G') = \eta(G'_1) + \eta(G'_2).$$

■

One consequence of this theorem is that we cannot hope to use our technique to find a graph's chromatic number, for instance. The chromatic number of G' , usually denoted by $\chi(G')$, is the minimum number of colors needed to color the nodes of

CHAPTER 3

EDGE REVERSAL IN A SYNCHRONOUS MODEL

3.1. Introduction

We present in this chapter a study of Scheduling by Edge Reversal in a synchronous environment. In our synchronous model, all nodes operate in lockstep according to a global clock. Local computation takes no time, and messages sent in the beginning of a clock cycle reach their destinations before the beginning of the next cycle. Under a synchronous model of distributed computation, the following is how Scheduling by Edge Reversal operates. At the beginning of the first clock cycle, G is oriented by an acyclic orientation. One or more of the sinks according to this initial orientation operate at this time, and send messages on all incident edges. These messages correspond to reversing the orientation of those edges, and may carry additional information, depending on the particular application at hand. At the beginning of the next clock cycle, all such messages have been delivered to their destinations, and G is oriented by a new acyclic orientation in which the sinks which operated in the previous clock cycle are now sources. Edge Reversal in a synchronous model can then be regarded as the evolution in time of acyclic orientations of G .

In Section 3.2, we define Orientation-Transition Diagrams. In Section 3.3, we define schedules, greedy schedules, and elaborate on a tighter form of starvation-freedom that holds for Scheduling by Edge Reversal. The structural

properties of G 's Orientation-Transition Diagram are discussed in Section 3.4. The discussion includes its strong connectivity and its main characteristics as implied by greedy schedules, such as periodicity in the Orientation-Transition Diagram. Section 3.5 contains a detailed study of greedy schedules, including their relation to node multicolorings and the description of what happens in the cases in which G is a tree, a cycle, or a complete graph.

3.2. Orientation-Transition Diagrams

Here we introduce a graph description of the evolution of acyclic orientations of G . The *Orientation-Transition Diagram* of G , OTD for short, is the directed graph $OTD=(\Omega,\Gamma)$. A vertex in OTD is an acyclic orientation of G . The directed edges in OTD are defined by the function Γ , which is of the form

$$\Gamma : \Omega \rightarrow 2^\Omega ,$$

where $\omega' \in \Gamma(\omega)$ if and only if there is a nonempty subset of *sinks*(ω) whose reversal yields ω' . In OTD , a directed edge exists from ω to ω' if and only if $\omega' \in \Gamma(\omega)$.

It will be sometimes useful to have the function Γ^{-1} , which is of the form

$$\Gamma^{-1} : \Omega \rightarrow 2^\Omega ,$$

and such that $\Gamma^{-1}(\omega) = \{\omega' \mid \omega \in \Gamma(\omega')\}$. In other words, $\Gamma(\omega)$ is the set of acyclic orientations that can be reached from ω in exactly one synchronous step, while $\Gamma^{-1}(\omega)$ is the set of acyclic orientations from which ω can be reached in exactly one synchronous step. One sees that, if $\omega' \in \Gamma^{-1}(\omega)$, then ω' can be obtained from ω by turning some of ω' 's sources into sinks.

It is easy to see that the following holds for all $\omega \in \Omega$:

$$|\Gamma(\omega)| = 2^{|\text{sinks}(\omega)|} - 1,$$

$$|\Gamma^{-1}(\omega)| = 2^{|\text{sources}(\omega)|} - 1.$$

These two quantities represent what we call the out-degree and the in-degree, respectively, of a vertex ω in OTD .

3.3. Synchronous Schedules

3.3.1. General and Greedy Schedules

Let $(\omega_k, k \geq 1)$ be an infinite sequence of acyclic orientations such that $\omega_{k+1} \in \Gamma(\omega_k)$ for all $k \geq 1$. We call such a sequence a *schedule*, and denote a generic schedule by σ . The set of all possible schedules is denoted by Σ . In terms of the graph OTD , a schedule is an infinite directed path.

A schedule $\sigma \in \Sigma$ is said to be *greedy* if, for any two of its orientations ω and ω' such that $\omega' \in \Gamma(\omega)$, ω' is obtained from ω by reversing all sinks in ω . The set of all greedy schedules is denoted by $\Sigma_g \subseteq \Sigma$. We specialize the functions Γ and Γ^{-1} to the case of greedy schedules, by defining the two functions

$$\Gamma_g : \Omega \rightarrow \Omega$$

and

$$\Gamma_g^{-1} : \Omega \rightarrow 2^\Omega.$$

Let $OTD_g = (\Omega, \Gamma_g)$ be the graph depicting the evolution of acyclic orientations under greedy scheduling.

The out-degree of a vertex ω in OTD_g is clearly equal to 1, by the definition of Γ_g . In order to evaluate its in-degree, we state the following simple lemma.

Lemma 3.1: Let ω and ω' be acyclic orientations of G such that $\omega' = \Gamma_g(\omega)$. A sink in ω' has at least one neighbor which is a sink in ω .

Proof: Since no sink in ω' is a sink in ω , then every sink in ω' has at least one neighbor which is a source in ω' and is not a source in ω . Thence the lemma. ■

It is an immediate consequence of Lemma 3.1 that $|\Gamma_g^{-1}(\omega)|$ is given by the number of nonempty subsets of *sources*(ω) in which each member of *sinks*(ω) has at least one neighbor. In other words, a member ω' of $|\Gamma_g^{-1}(\omega)|$ is obtained from ω by turning into sinks a subset of sources in such a way that all sinks in ω become nonsinks.

3.3.2. A Stronger Form of Starvation-Freedom

Consider node $i \in N$, and a schedule $\sigma \in \Sigma$. Define $m_i(\sigma, q)$ to be the number of times that node i operates in the first q orientations of σ , $q \geq 1$. Clearly, $m_i(\sigma, 1) = 0$ for all $i \in N$ and all $\sigma \in \Sigma$.

In this section we state a property of Scheduling by Edge Reversal which is a stronger version of its starvation-freedom property. As we shall see in the proof of Theorem 3.2 below, the fact that we assume G to be connected implies that a node cannot operate much faster than any other.

Theorem 3.2: Consider nodes $i, j \in N$, and let them be connected in G by the r -node undirected path whose nodes are $i = i_1, \dots, i_r = j$. Then $|m_i(\sigma, q) - m_j(\sigma, q)| \leq r - 1$, for all $\sigma \in \Sigma$ and all $q \geq 1$.

Proof: We use induction on r . For $r=2$, i and j are neighbors, and therefore alternate in their turns to operate, thus proving the assertion. As the induction hypothesis, assume that the assertion is true for any two nodes connected to each other by an undirected path with no more than $r-1$ nodes, for some $r \geq 3$. Now consider nodes i and j in the statement of the theorem. The induction hypothesis applied to the path $i=i_1, \dots, i_{r-1}$ yields

$$|m_i(\sigma, q) - m_{i_{r-1}}(\sigma, q)| \leq r-2,$$

and applied to the 2-node path $i_{r-1}, i_r=j$ yields

$$|m_{i_{r-1}}(\sigma, q) - m_j(\sigma, q)| \leq 1.$$

It is therefore clear that

$$|m_i(\sigma, q) - m_j(\sigma, q)| \leq r-1.$$

■

Corollary 3.3: Consider nodes $i, j \in N$, and let the shortest path connecting them in G have $r-1$ edges. Then $|m_i(\sigma, q) - m_j(\sigma, q)| \leq r-1$, for all $\sigma \in \Sigma$ and all $q \geq 1$.

■

3.3.3. A Comparison between General and Greedy Schedules

The following theorem establishes a comparison between greedy and nongreedy schedules starting at a same orientation. It originates from a similar result in [Camp86].

Theorem 3.4: Consider orientation ω_1 , and let $\sigma_g = (\omega_1, \omega_2, \dots) \in \Sigma_g$ be the greedy schedule that starts at ω_1 . Let $\sigma = (\omega'_1, \omega'_2, \dots) \in \Sigma$ be any other schedule starting at the same orientation (i.e. $\omega'_1 = \omega_1$). Then $m_i(\sigma, q) \leq m_i(\sigma_g, q)$ for all $i \in N$ and

all $q \geq 1$.

Proof: We use induction on q . For $q=1$, the assertion clearly holds with equality. As the induction hypothesis, assume that $m_i(\sigma, q) \leq m_i(\sigma_g, q)$ for all $i \in N$ and some $q \geq 1$. We then show that $m_i(\sigma, q+1) \leq m_i(\sigma_g, q+1)$. For such, it suffices to show that, if a node $i \in N$ exists for which $m_i(\sigma, q) = m_i(\sigma_g, q)$, and i operates in going from ω'_q to ω'_{q+1} , then it must also operate in going from ω_q to ω_{q+1} , i.e. it must be a sink in ω_q . Let $i \in N$ be such a node. Because i is a sink in ω'_q , it must be, by Corollary 3.3, that either $m_i(\sigma, q) = m_j(\sigma, q)$ or $m_i(\sigma, q) + 1 = m_j(\sigma, q)$, for all neighbors j of i . By the induction hypothesis applied to i 's neighbors, and using the fact that $m_i(\sigma, q) = m_i(\sigma_g, q)$ together with Corollary 3.3, it must also be that either $m_i(\sigma_g, q) + 1 = m_j(\sigma_g, q)$, or $m_i(\sigma_g, q) = m_j(\sigma_g, q)$. The former happens for some of those neighbors j of i for which $m_i(\sigma, q) = m_j(\sigma, q)$. Since both schedules start at the same initial orientation ω_1 , i is a sink in ω_q . ■

Theorem 3.4 shows that a node operates no less frequently under a greedy schedule than it does under any other schedule starting at the same orientation as the greedy one.

3.4. Structure of the Orientation-Transition Diagram

3.4.1. Reachability Equations

Let ω and ω' be any two acyclic orientations. We say that ω' is *reachable* from ω if and only if $\omega' \in \Gamma^*(\omega)$, where Γ^* is the transitive closure of Γ . Let $m_i(\omega, \omega')$ denote the number of times that node i has to operate in order for ω' to be reached from ω through some schedule. Say that an edge (i, j) is *marked* if $\omega((i, j)) \neq \omega'((i, j))$, and *unmarked*, if $\omega((i, j)) = \omega'((i, j))$. Consider the following

equations, which we call the *reachability equations from ω to ω'* .

$$m_j(\omega, \omega') = \begin{cases} m_i(\omega, \omega') + 1, & \text{if } (i, j) \text{ is marked,} \\ m_i(\omega, \omega'), & \text{otherwise,} \end{cases}$$

for all edges (i, j) such that $\omega((i, j)) = j$.

Theorem 3.5: The reachability equations from ω to ω' admit a nonnegative integer solution if and only if $\omega' \in \Gamma^*(\omega)$.

Proof: If $\omega' \in \Gamma^*(\omega)$, then let y_i denote the number of times that node i operates from ω to ω' in a schedule σ . Consider edge (i, j) such that $\omega((i, j)) = j$. If (i, j) is marked, then in σ node j has to operate exactly once more than node i does from ω to ω' , i.e. $y_j = y_i + 1$. If it is unmarked, then they both have to operate the same number of times from ω to ω' , i.e. $y_j = y_i$. Thus $(y_i, i \in N)$ is a nonnegative integer solution to the reachability equations from ω to ω' .

Conversely, let $(y_i, i \in N)$ be a nonnegative integer solution to the reachability equations from ω to ω' . If edge (i, j) is such that $\omega((i, j)) = j$, then it must be that $y_j = y_i + 1$ if (i, j) is marked, and $y_j = y_i$ otherwise. If we build a schedule in which each node i operates exactly y_i times up to a certain point, then at that point each marked edge will have been reversed an odd number of times, whereas an unmarked edge will have been reversed an even number of times. In other words, ω' will have been reached. It remains to argue that such a schedule can always be built. For such, notice that the nonnegative integers $(y_i, i \in N)$ are nondecreasing along any directed path in ω . If a sink i such that $y_i \geq 1$ operates in ω yielding orientation ω'' , then the same n -tuple, with y_i replaced with $y_i - 1$, is a nonnegative integer solution to the reachability equations from ω'' to ω' . This solution is also nondecreasing along any directed path, and we see as a consequence that a sink can always be found to

operate. ■

One consequence of Theorem 3.5 is that any orientation is reachable from itself, and that the corresponding solutions to the reachability equations are any n -tuple with the same nonnegative integer in all entries.

Theorem 3.6: Let ω' be reachable from ω , and $S_0, \dots, S_{\lambda-1}$ be the sink decomposition according to ω . There is a schedule through which ω' can be reached from ω such that at least one node in $S_{\lambda-1}$ does not operate between ω and ω' .

Proof: We show that the reachability equations from ω to ω' admit a nonnegative integer solution in which $m_i(\omega, \omega')=0$ for some $i \in S_{\lambda-1}$. For let $(y_i, i \in N)$ be a nonnegative integer solution to those equations. Clearly, $(y_i - c, i \in N)$ is also an integer solution, for any integer c . In particular, take

$$c = \min_{i \in S_{\lambda-1}} y_i,$$

and we get a nonnegative integer solution, since the numbers y_i are nondecreasing along any directed path. In this solution, at least one of the nodes in $S_{\lambda-1}$, say i , is such that $y_i - c = 0$, by the definition of c . Thence the theorem. ■

3.4.2. Strongly Connected Components

OTD is a directed graph whose underlying undirected graph may have several connected components. In this section, we want to show that each of these connected components is strongly connected.

Theorem 3.7: Each of the connected components of *OTD* is strongly connected.

Proof: We show that, given the acyclic orientations ω and ω' , if $\omega' \in \Gamma^*(\omega)$, then $\omega \in \Gamma^*(\omega')$. By Theorem 3.5, the reachability equations from ω to ω' admit a nonnegative integer solution $(y_i, i \in N)$. Let

$$c = \max_{i \in N} y_i,$$

and notice that the n -tuple $(c, i \in N)$ is a solution to the reachability equations from ω to itself. Then it must be that $(c - y_i, i \in N)$ is a nonnegative integer solution to the reachability equations from ω' to ω , and by Theorem 3.5 $\omega \in \Gamma^*(\omega')$. ■

3.4.3. Properties under Greedy Scheduling

3.4.3.1. Periodicity

If only greedy schedules are considered, we see from the definition of Γ_g that, due to the finiteness of Ω , some orientations must repeat themselves periodically. In other words, there must be at least one integer $p > 1$ for which p orientations $\alpha_0, \dots, \alpha_{p-1}$ can be found, such that $\alpha_1 = \Gamma_g(\alpha_0)$, $\alpha_2 = \Gamma_g(\alpha_1), \dots, \alpha_0 = \Gamma_g(\alpha_{p-1})$. We let such orientations be called *periodic*, and refer to the sequence of orientations $\alpha_0, \dots, \alpha_{p-1}$ as a *period of length p* . Also, the value of $|\Gamma_g^{-1}(\omega)|$ can be equal to zero for some orientations, and therefore these orientations are such that they can only participate in a greedy schedule as the first orientation in that schedule. So OTD_g , like OTD , also has connected components, which, by the fact we just mentioned, are not in this case strongly connected. A greedy schedule can then be viewed as an initial sequence of orientations, followed by an infinite repetition of a period.

In Figure 3.1 we illustrate the general structure of a connected component of OTD_g . In Figure 3.2, we present an example for the case in which G is the 5-node cycle. The example shows a connected component of OTD_g , which includes only the full edges in the drawing, and its enlargement to the corresponding connected component in OTD , which includes the full edges and the dashed edges.

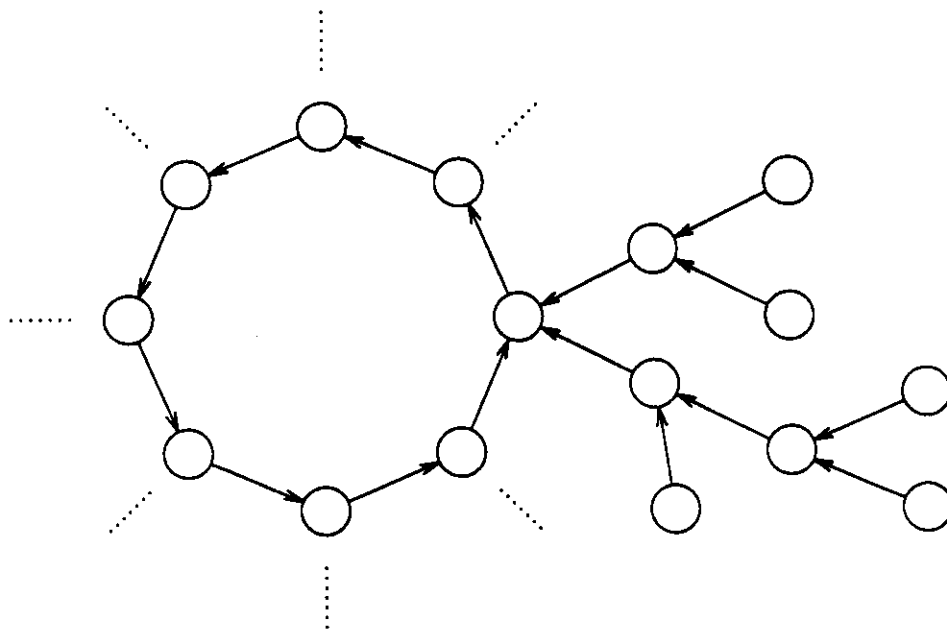


Figure 3.1. Generic Structure of a Connected Component in OTD_g

We now establish a simple property of periods.

Corollary 3.8: All nodes operate the same number of times in a period.

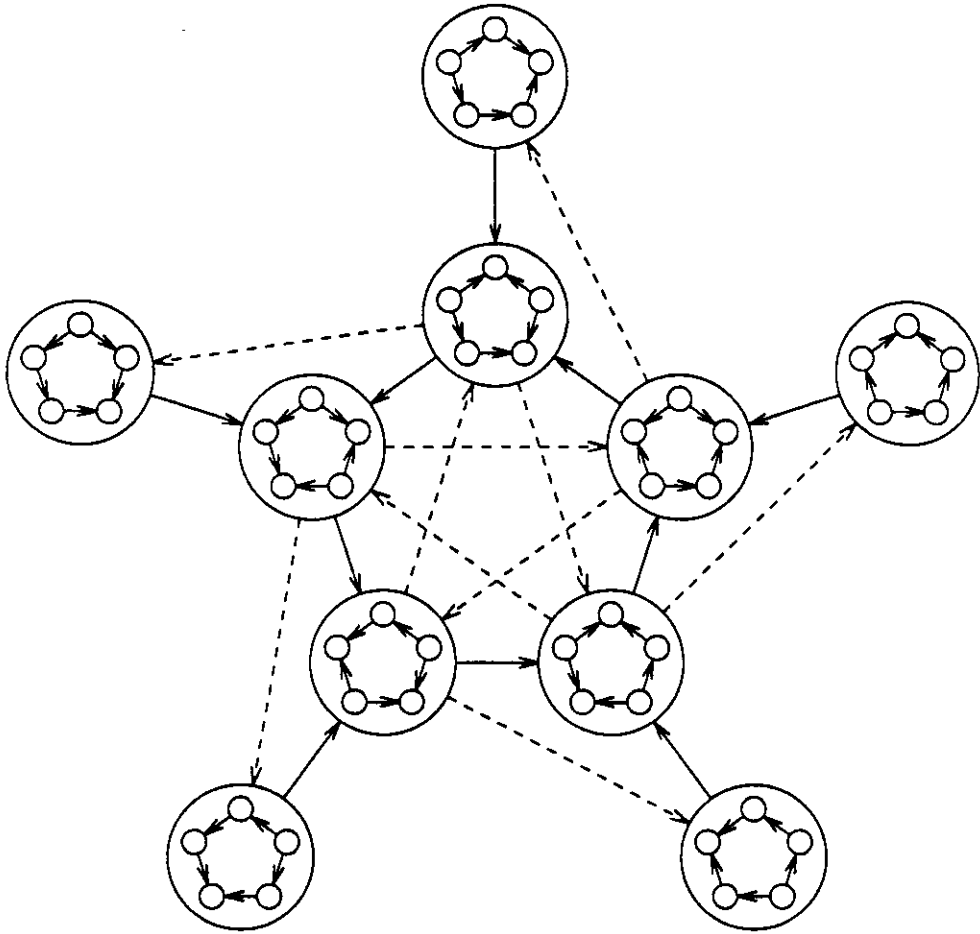


Figure 3.2. A Connected Component in OTD_g and in OTD

Proof: Immediate from Theorem 3.2. Also follows from the reachability equations from an orientation to itself. ■

In the sequel, we shall denote by m the number of times that a node operates in the period under consideration.

3.4.3.2. Evolution of Sink Decompositions

Here we look at greedy schedules by considering the sink decompositions of the orientations involved. We state the following theorem.

Theorem 3.9: The length of sink decompositions is monotonically nonincreasing under greedy schedules, and if it decreases it does so by exactly one.

Proof: Recall that the length of a sink decomposition is the length of a longest path in the corresponding orientation. If all sinks are reversed, the length of such a path can either stay the same, if it starts at one of the new sources, or decrease by one, otherwise. ■

Now define $\Omega(\alpha_0, \dots, \alpha_{p-1}) \subseteq \Omega$ as the set of acyclic orientations from which the period $\alpha_0, \dots, \alpha_{p-1}$ is reachable in some greedy schedule.

Corollary 3.10: Consider the period $\alpha_0, \dots, \alpha_{p-1}$. Orientations $\alpha_0, \dots, \alpha_{p-1}$ have sink decompositions of same length, and this length is no greater than that of any of the orientations in $\Omega(\alpha_0, \dots, \alpha_{p-1})$. ■

3.5. A Study of Periodic Behavior

3.5.1. Periodicity and Node Multicolorings

As we remarked earlier, Scheduling by Edge Reversal in a synchronous environment is such that at each cycle the nodes which are allowed to operate simultaneously constitute an independent set. As a consequence, a node coloring of some

sort may assign the same color to all these nodes which operate at the same time. In this section we relate our synchronous scheduling method to node multicolorings.

For the next theorem, we recall that $\xi: N \rightarrow \{c_0, \dots, c_{q-1}\}^k$ is a k -tuple coloring of G 's nodes, and $C_l \subset N$ denotes the subset of nodes with color c_l , for $0 \leq l \leq q-1$.

Theorem 3.11: Consider the period $\alpha_0, \dots, \alpha_{p-1}$, in which each node operates m times. G admits the p -color, m -tuple coloring by colors c_0, \dots, c_{p-1} , such that $C_k = \text{sinks}(\alpha_k)$, $0 \leq k \leq p-1$. Moreover, the following two properties hold for this coloring:

- (i) Let c_{i_1}, \dots, c_{i_m} and c_{j_1}, \dots, c_{j_m} be the m colors assigned to nodes i and j , respectively, and assume without loss of generality that $i_1 < \dots < i_m$ and $j_1 < \dots < j_m$. If i and j are neighbors such that $\alpha_0((i, j)) = i$, then $i_1 < j_1 < i_2 < j_2 < \dots < i_m < j_m$;
- (ii) Each node in C_k has at least one neighbor in $C_{(k-1) \bmod p}$, $0 \leq k \leq p-1$.

Conversely, if G oriented by α_0 admits a p -color, m -tuple coloring by colors c_0, \dots, c_{p-1} obeying properties (i) and (ii) above, then α_0 repeats itself according to the sequence $\alpha_0, \dots, \alpha_{p-1}, \alpha_0$, in which each node operates m times in going from α_0 back to itself, and such that $\text{sinks}(\alpha_k) = C_k$, $0 \leq k \leq p-1$.

Proof: In order to show the first part, color the nodes in $\text{sinks}(\alpha_k)$ with color c_k , $0 \leq k \leq p-1$. Each node is a sink in exactly m of the orientations in the period, so it receives exactly m colors, and therefore the coloring described above is a p -color, m -tuple coloring such that $C_k = \text{sinks}(\alpha_k)$ for all applicable values of k . If $\alpha_0((i, j)) = i$ for an edge (i, j) , it must be that $i_1 < j_1 < i_2 < j_2 < \dots < i_m < j_m$, since node i 's operations

must alternate with those of node j , and i must be the first one to operate. Also, it must be by Lemma 3.1 that each node in C_k has at least one neighbor in $C_{(k-1)\bmod p}$, $0 \leq k \leq p-1$.

Now we show the converse statement. By property (i), all nodes in C_0 are sinks in α_0 , i.e. $C_0 \subseteq \text{sinks}(\alpha_0)$. Also, if a node not in C_0 is to be a sink in α_0 , then it must be, by property (i), that either it is a member of C_1 and has no neighbor in C_0 , or it is a member of C_2 and has no neighbor in $C_0 \cup C_1$, etc., all the way to its being a member of C_{p-1} with no neighbor in $C_0 \cup \dots \cup C_{p-2}$. In any of these cases property (ii) is contradicted, and consequently $\text{sinks}(\alpha_0) \subseteq C_0$, that is, $\text{sinks}(\alpha_0) = C_0$. The same argument can be applied inductively to showing that $\text{sinks}(\alpha_2) = C_2, \dots, \text{sinks}(\alpha_p) = C_0$, since property (i) guarantees that all nodes in C_k are sinks in α_k , and property (ii) implies that such is the case for nodes in C_k only, where α_k denotes the orientation obtained after all nodes in C_0, \dots, C_{k-1} have operated, in this order, for $2 \leq k \leq p$. Because each node belongs to exactly m of the color classes C_0, \dots, C_{p-1} , it must be that each node operates m times in going from α_0 to α_p . Consequently, it must be that $\alpha_p = \alpha_0$. ■

Corollary 3.12: Let G oriented by α_0 admit a p -color, m -tuple coloring by colors c_0, \dots, c_{p-1} obeying properties (i) and (ii) in the statement of Theorem 3.11. Orientation α_0 is in a period of length p in which each node operates m times if and only if no integer $q \geq 2$ exists such that G oriented by α_0 admits a $\frac{p}{q}$ -color, $\frac{m}{q}$ -tuple coloring for which properties (i) and (ii) hold with $\frac{p}{q}$ and $\frac{m}{q}$ in place of p and m , respectively.

Proof: The sufficiency condition is a consequence of the first part of Theorem 3.11. Necessity follows from the second part. ■

By Corollary 3.12, if orientation α_0 is such that G oriented by it admits a p -color, m -tuple coloring for which properties (i) and (ii) in the statement of Theorem 3.11 hold, then α_0 is in a period of length $\frac{p}{q}$, in which each node operates $\frac{m}{q}$ times, where q is the largest integer for which G , oriented by α_0 , admits a $\frac{p}{q}$ -color, $\frac{m}{q}$ -tuple coloring of which properties (i) and (ii) are true. Conversely, if α_0 is in the period $\alpha_0, \dots, \alpha_{p-1}$, and each node operates m times in it, then there is no integer $q \geq 2$ for which G admits a $\frac{p}{q}$ -color, $\frac{m}{q}$ -tuple coloring obeying properties (i) and (ii) under α_0 .

In Figure 3.3 we present periodic orientations of two different graphs to illustrate Theorem 3.11. Next to each node in each of the two orientations one sees the appropriate colors. The orientation on the left is in a period of length 5 in which each node operates twice (hence the 5-color, 2-tuple coloring of its nodes), while the one on the right is in a period of length 8 in which each node operates 3 times (hence its nodes' 8-color, 3-tuple coloring).

3.5.2. Periods in which Nodes Operate Exactly Once

Periods in which each node operates only once ($m=1$) are especially simple to analyze. In this section we investigate a necessary and sufficient condition for an orientation to be in such a period.

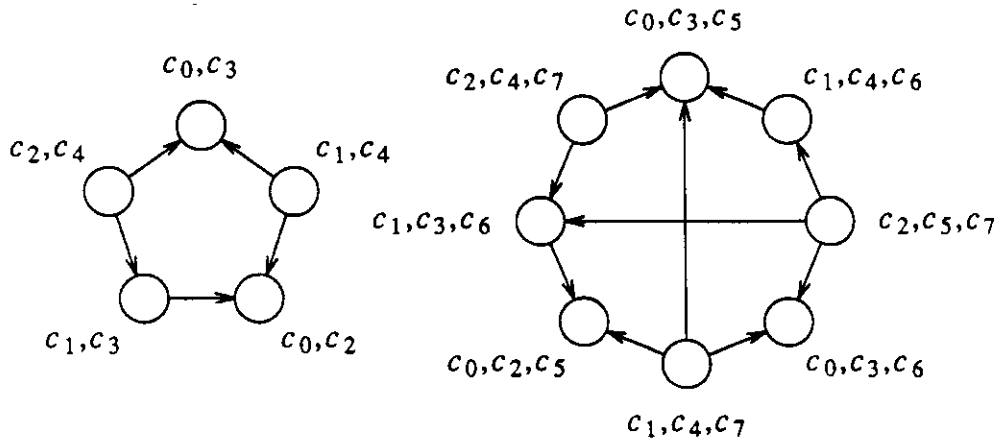


Figure 3.3. Periodic Orientations and Associated Multicolorings

Corollary 3.13: Consider orientation α_0 , and let $S_0, \dots, S_{\lambda-1}$ be its sink decomposition. This orientation is in a period $\alpha_0, \dots, \alpha_{p-1}$ in which each node operates exactly once (i.e. $m=1$) if and only if each node in S_0 has at least one neighbor in $S_{\lambda-1}$. In this case, $p=\lambda$ and $\text{sinks}(\alpha_k)=S_k, 0 \leq k \leq \lambda-1$.

Proof: Immediate from Theorem 3.11. ■

Corollary 3.14: Let α_0 be a periodic orientation such that $m=1$, and denote by $S_0, \dots, S_{\lambda-1}$ its sink decomposition. If G is not a tree, then it contains, for some $q \geq 1$, a simple cycle with $q\lambda$ nodes, which, shown in order, are

$$i_0^1, \dots, i_{\lambda-1}^1, \dots, i_0^q, \dots, i_{\lambda-1}^q,$$

where $i_k^l \in S_k$ for all $1 \leq l \leq q$, $0 \leq k \leq \lambda - 1$.

Proof: If G is not a tree, then the assertion follows immediately for $\lambda = 2$. If $\lambda > 2$, then build the simple cycle as follows. Choose a node from those in $S_{\lambda-1}$. By the properties of a sink decomposition, this node must have a neighbor in $S_{\lambda-2}$, this neighbor a neighbor in $S_{\lambda-3}$, and so on. A chain with λ nodes is then obtained, each one belonging to a different layer in the sink decomposition. The node in S_0 , in particular, has a neighbor in $S_{\lambda-1}$, by Corollary 3.13, and we pick this node to be part of the chain, if it still is not. We proceed likewise until a node is picked which is already in the chain, which must happen if $\lambda > 2$, since G cannot possibly be a tree in this case (see Theorem 3.16 below). Then a simple cycle is formed with $q\lambda$ nodes, for some $q \geq 1$, q of them from each of the layers of the sink decomposition. ■

The next section contains a study of the structure of OTD_g when G is assumed to be a tree, a cycle, and a complete graph.

3.5.3. Periodicity in Trees, Cycles, and Complete Graphs

3.5.3.1. Trees

Here we consider the case in which G is a tree. In what follows, we recall that $d_\omega(i)$ stands for the depth of node i in orientation ω , i.e. $d_\omega(i) = k$ if and only if $i \in S_k$, where $S_0, \dots, S_{\lambda-1}$ is G 's sink decomposition according to ω and $0 \leq k \leq \lambda - 1$.

Lemma 3.15: Let G be a tree, and let it be oriented by ω . There is an orientation ω' reachable from ω through some greedy schedule, such that, for all $i, j \in N$, if $(i \rightarrow j)$ is an edge oriented according to ω' , then $d_{\omega'}(i) - d_{\omega'}(j) = 1$.

Proof: The proof goes by induction on the number of nodes in the tree, which we denote by r . In the basis case we have $r=2$, and the lemma holds trivially. Assume that the lemma holds for all trees with $r \geq 2$ nodes. Now let G be a tree such that $|N|=r+1$, and let it be oriented by ω . If in ω all leaves are sinks, then apply the argument that follows to the orientation immediately reachable from ω . Let l be one of the leaves which are sources in ω , and let $G^{(l)}$ be the tree induced by the nodes in $N-\{l\}$. Since $|N-\{l\}|=r$, the induction hypothesis applies to $G^{(l)}$ particular, if we orient $G^{(l)}$'s edges by the appropriate truncation ω_t of orientation ω , then there must be an orientation ω'_t of $G^{(l)}$'s edges reachable from ω_t such that for all $i, j \in N-\{l\}$, if the edge $(i \rightarrow j)$ exists in ω'_t , then $d_{\omega'_t}(i) - d_{\omega'_t}(j) = 1$. Let ω' be an extension of ω'_t to include the edge adjacent to leaf l . Because l is a source in ω , then it is clear that ω' is reachable from ω through a greedy schedule. According to ω' , l is either a source or a sink. If it is a source, then $d_{\omega'}(l) - d_{\omega'}(l') = 1$, l' being the only neighbor of l . If it is a sink, it may be that $d_{\omega'}(l) - d_{\omega'}(l') > 1$, in which case we consider the orientation immediately reachable from ω' , call it ω'' . According to ω'' , l is a source, and therefore $d_{\omega''}(l) - d_{\omega''}(l') = 1$. Since this is still true of all other edges (because it was so in ω'), the lemma follows, in the latter case with ω'' , in the former case with ω' itself. ■

Theorem 3.16: If G is a tree, then a period in OTD_g has length 2.

Proof: Orient G by orientation ω . By Lemma 3.15, there is an orientation ω' reachable from ω such that if an edge $(i \rightarrow j)$ is oriented by ω' , then $d_{\omega'}(i) - d_{\omega'}(j) = 1$. This means that in the sink decomposition of ω' there are no edges between non-successive layers. As a result, an orientation whose sink decomposition has length 2 must be reachable from ω' . The theorem then follows from the observation that any

orientation whose sink decomposition has length 2 must be in a period of length 2, by Corollary 3.13. ■

Corollary 3.17: If G is a tree, then OTD_g has exactly one connected component.

Proof: Theorem 3.16 implies that any period in OTD_g must correspond to a partition of G 's nodes into two independent subsets. Since a tree admits only one such partition, the period must be unique, and therefore so must the corresponding connected component. ■

3.5.3.2. Cycles

Define an *arc* to be a path in G falling into one of the following categories:

- (i) *a-arc*: alternating sinks and sources (Figure 3.4(a));
- (ii) *cw-arc*: comprises at least three nodes, starts at a source, ends at a sink, and all edges in between the endpoints are oriented clockwise (Figure 3.4(b));
- (iii) *ccw-arc*: comprises at least three nodes, starts at a source, ends at a sink, and all edges in between the endpoints are oriented counter-clockwise (Figure 3.4(c)).

An oriented cycle is therefore the juxtaposition of arcs of the types above with endpoints in common. Notice that no oriented cycle can be made of a single *cw*- or *ccw*-arc.

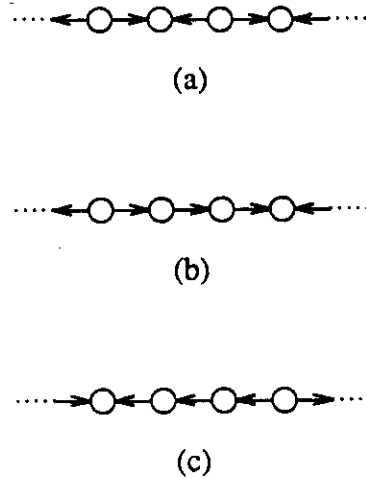


Figure 3.4. Types of Oriented Arcs in a Cycle

A further definition that we make is that of a *cw-uniform* orientation of a cycle, and likewise that of a *ccw-uniform* orientation. A cycle is said to be oriented by a *cw-uniform* orientation if its orientation is the result of the juxtaposition, endpoints shared, of arcs, none of which is a *ccw-arc*. It is said to be oriented by a *ccw-uniform* orientation if no *cw-arcs* can be found in the juxtaposition of arcs that constitutes its orientation. An orientation consisting of a single *a-arc* may be thought of as belonging to either class; such orientation is only possible in cycles with even length. In fact, any *a-arc* in a *cw-* or *ccw-uniform* orientation must have an even number of nodes. Figure 3.5 exemplifies a *cw-uniform* orientation. Boxed segments correspond to *a-arcs*.

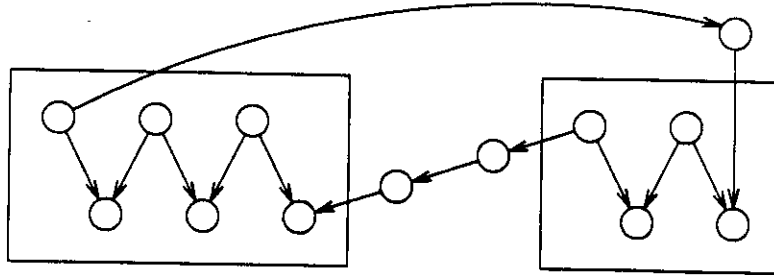


Figure 3.5. A *cw*-Uniform Orientation

Lemma 3.18: An orientation α_0 of a cycle is periodic if and only if it is either *cw*- or *ccw*-uniform.

Proof: The argument we use here is based on the fact that when all sinks operate synchronously in a cycle, the resulting orientation has all of its *cw*-arcs (*ccw*-arcs) shifted one node counter-clockwise (clockwise) with respect to the previous orientation. As the operation goes on, this counter-clockwise (clockwise) shifting continues, leaving an *a*-arc behind. This is illustrated in Figure 3.6, where we show a segment of a large cycle whose only non-*a*-arcs are those shown by squares in the figure. The nodes seen in the figure are naturally the same in all of its stages; what happens is that the *cw*- and *ccw*-arcs are formed by different nodes at each stage.

Sufficiency is shown by simply noting that in a *cw*-uniform (*ccw*-uniform) orientation all non-*a*-arcs are *cw*-arcs (*ccw*-arcs), and therefore the shifting described

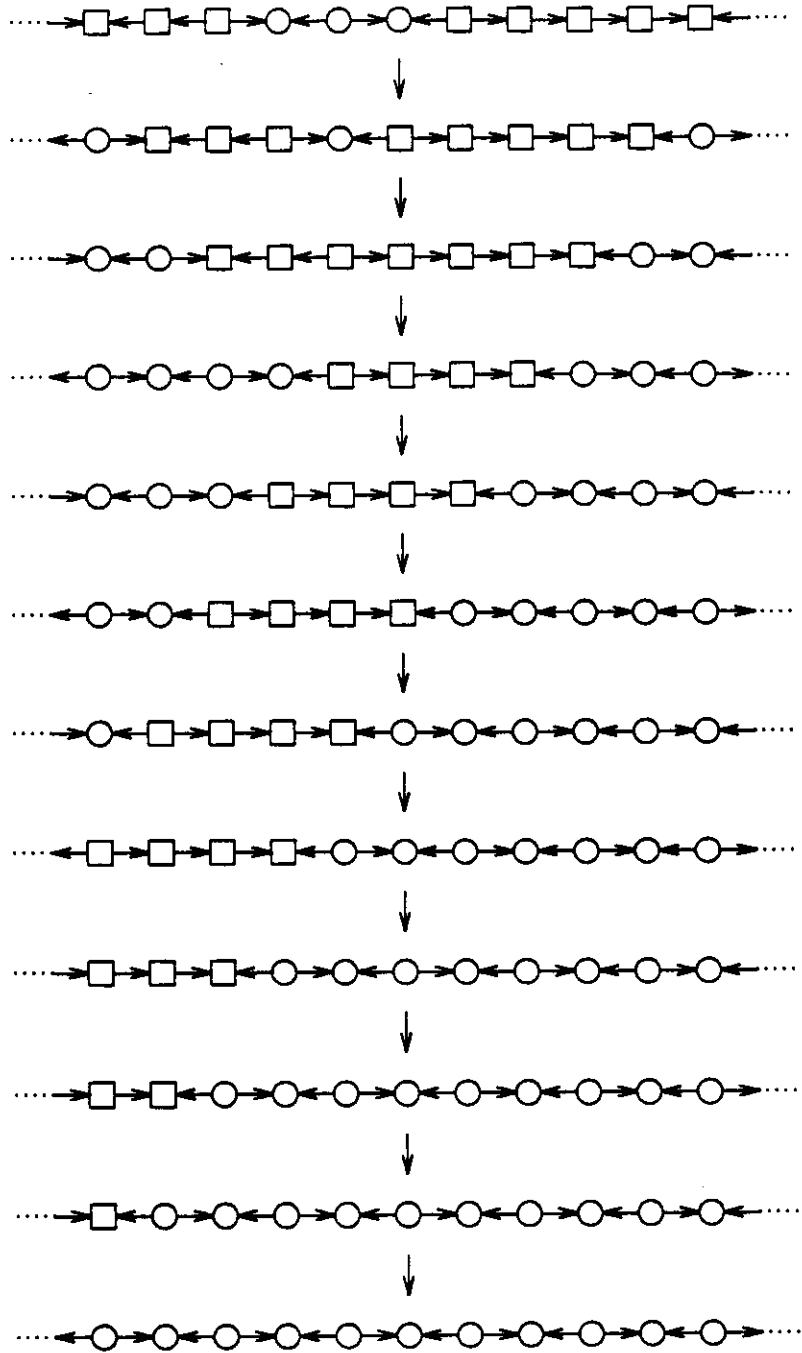


Figure 3.6. Arc Drifting under Greedy Scheduling

above never causes arcs to collide, since they all drift in the same direction. As a result, the initial orientation α_0 is necessarily repeated.

To prove necessity we merely observe that in the process described above through which a *cw*- or *ccw*-uniform orientation repeats itself, all intermediate orientations are also *cw*- or *ccw*-uniform, respectively, and for this reason all orientations in that period must be of one of these types. We see that this is the case for orientations in all periods by pointing out the process by which a *cw*- or *ccw*-uniform orientation is reached from an orientation of neither of these types as the synchronous operation of sinks progresses. For assume that an orientation ω has at least one *cw*- and one *ccw*-arc. As the synchronous operation proceeds, *cw*-arcs drift counterclockwise, and *ccw*-arcs drift clockwise, leaving α -arcs behind. As a result, *cw*-arcs meet *ccw*-arcs at some point, annihilating themselves, the shorter one completely. Eventually, only *cw*- or *ccw*-arcs remain, but not both, meaning that a *cw*- or *ccw*-uniform orientation has been reached, respectively. ■

An orientation is said to be *p*-periodic, for some $2 \leq p \leq n$, if it is either *cw*- or *ccw*-uniform, and $q = \frac{n}{p}$ nodes i_0, \dots, i_{q-1} can be identified such that the following conditions hold:

- (i) There are exactly $p - 1$ nodes between i_k and $i_{(k+1) \bmod q}$, $0 \leq k \leq q - 1$;
- (ii) The orientation of the edges in the sequence of nodes $i_k, \dots, i_{(k+1) \bmod q}$ is exactly the same for all $0 \leq k \leq q - 1$.

Note that by Lemma 3.18 an orientation is in a period if and only if it is *n*-periodic. Figure 3.7 illustrates a 5-periodic orientation of a 10-node cycle; nodes i_0

and i_1 have been identified.

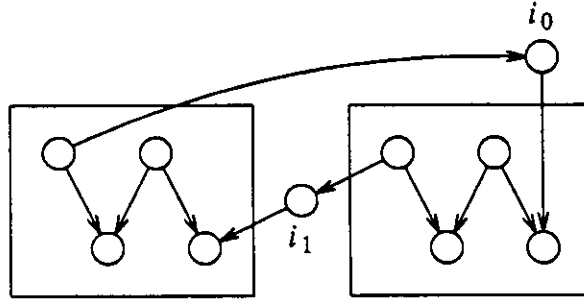


Figure 3.7. A 5-Periodic Orientation

Theorem 3.19: Let G be a cycle, and consider a period in OTD_g . If $2 \leq p \leq n$ is the least integer for which the orientations in this period are p -periodic, then p is the period's length.

Proof: Let α_0 be an orientation in the period of OTD_g considered here. By Lemma 3.18, α_0 is either cw - or ccw -uniform. In this proof we assume that α_0 is cw -uniform, since the case in which it is ccw -uniform is entirely symmetric. Because α_0 is cw -uniform, there is an integer $2 \leq p' \leq n$ for which α_0 is p' -periodic; we let p be the least such integer. By the definition of p -periodicity, we can find $q = \frac{n}{p}$ nodes i_0, \dots, i_{q-1} such that, for $0 \leq k \leq q-1$, i_k and $i_{(k+1) \bmod q}$ are $p-1$ nodes apart. Moreover, the edges in the sequences of nodes $i_k, \dots, i_{(k+1) \bmod q}$ are identically oriented for all applicable values of k . A step of synchronous operation can be viewed as each edge taking on the orientation of its neighboring edge on the clockwise side. By the

p -periodicity of α_0 , the orientations of all edges will repeat themselves every p steps. Also, because p is the least integer with the property mentioned above, no smaller number of steps will suffice for the orientations to be repeated, and therefore p is indeed the period's length. ■

3.5.3.3. Complete Graphs

Theorem 3.20: Let G be a complete graph. OTD_g is constituted by $(n-1)!$ periods such that $p=n$ and $m=1$.

Proof: Observe first that each of G 's acyclic orientations is such that each layer of its sink decomposition contains exactly one node. In particular, if we denote by $S_0, \dots, S_{\lambda-1}$ the sink decomposition of one particular orientation, it must be that $\lambda=n$ and that the node in S_0 has as neighbor the node in $S_{\lambda-1}$. By Corollary 3.13, such an orientation must be in a period $\alpha_0, \dots, \alpha_{p-1}$ with $p=n$ and $m=1$. Also, we see from the form of the sink decompositions that G admits exactly $n!$ acyclic orientations, and since they are all periodic with $p=n$, OTD_g must be constituted by $(n-1)!$ such periods. ■

3.5.4. Periods in which Nodes Operate More than Once

In this section we investigate the question of how small and how large the value of p can be for a generic period $\alpha_0, \dots, \alpha_{p-1}$, if the graph is not a tree. A lower bound and an upper bound on this value will be presented in Chapter 4, and here we provide a preview of those bounds. The upper bound is a consequence of the fact that a simple cycle whose length is a multiple of p exists in G if a period of such a length is present in OTD_g . In Chapter 4, we show that such is always the case, and that in fact something stronger holds. If in the period $\alpha_0, \dots, \alpha_{p-1}$ each

node operates m times, then G , oriented by α_k , has as a subgraph a simple cycle which, when oriented by the appropriate truncation of α_k , is itself in a period of length p in which each of its nodes operates m times, $0 \leq k \leq p-1$. Notice that this can be easily seen to be true if $m=1$ in the period $\alpha_0, \dots, \alpha_{p-1}$, as a consequence of Corollary 3.14.

The following are the bounds that we show in Chapter 4. Assume that G contains at least one cycle (i.e. it is not a tree), and let $K=(N_K, E_K)$ and $C=(N_C, E_C)$ denote G 's largest completely connected subgraph and largest simple cycle, respectively. Consider a period in OTD_g , and let p be its length. It must be that $|N_K| \leq p \leq |N_C|$.

This upper bound on the size of a period is useful in providing us with the cost of edge reversal simulation to detect periodicity. More specifically, we can centralizedly simulate the synchronous process of greedily reversing edges incident to sinks. If we do so from an arbitrary orientation ω , we know that if ω is periodic, then n steps of the simulation suffice for it to repeat itself. If this repetition does not happen in that number of steps, then certainly ω is not periodic. It is straightforward to see that such a simulation has a complexity of $O(n^3)$.

CHAPTER 4

CONCURRENCY MEASURES

4.1. Introduction

In this chapter, we discuss how much concurrency can be attained under Scheduling by Edge Reversal in a synchronous model. We define in Section 4.2 a measure for the concurrency of a schedule, and in Section 4.3 a measure for the concurrency of an orientation. The latter expresses the amount of concurrency attainable when the computation is started at a certain acyclic orientation. In this respect, we show that the best is to follow the greedy schedule from that initial orientation. We obtain a closed-form expression for this concurrency involving characteristics of G only. In Section 4.4, we come to the question of how much concurrency the graph G can provide. This question amounts to choosing an initial acyclic orientation which will provide a maximum amount of concurrency. Such a measure is a number related to G 's chromatic and multichromatic numbers, and allows us to establish the optimality of Scheduling by Edge Reversal among all scheduling schemes based on what we call interleaved multicolorings. We show the value of this measure for some particular graphs, which include those discussed in Chapter 3. Finally, we prove that in general the decision problem corresponding to finding it is *NP*-complete.

4.2. The Concurrency of a Schedule

We recall from Chapter 3 that operation in the synchronous model under Scheduling by Edge Reversal can be regarded as the evolution in time of acyclic orientations of G . In particular, we have defined a schedule $\sigma \in \Sigma$ to be an infinite sequence of orientations of the form $\sigma = (\omega_1, \omega_2, \dots)$, where a nonempty subset of $\text{sinks}(\omega_k)$ is reversed in going from ω_k to ω_{k+1} , $k \geq 1$. If all of the members of $\text{sinks}(\omega_k)$ are reversed, we say that σ is a greedy schedule, the set of all greedy schedules being denoted by Σ_g .

In this section, we wish to define a measure for the concurrency attainable with a schedule. For such, we start by defining the function

$$\gamma: \Sigma \times \{1, 2, \dots\} \rightarrow \mathbb{R}.$$

For a schedule $\sigma \in \Sigma$ and a positive integer q , $\gamma(\sigma, q)$ is the concurrency attainable in the first q orientations of schedule σ . We define $\gamma(\sigma, q)$ to be given by

$$\gamma(\sigma, q) = \frac{1}{qn} \sum_{i \in N} m_i(\sigma, q),$$

for all $\sigma \in \Sigma$ and all $q \geq 1$. We recall from Chapter 3 that $m_i(\sigma, q)$ stands for the number of times that node i operates in the first q orientations of σ . One sees intuitively that $\gamma(\sigma, q)$ represents the average number of times that a node operates in the first q orientations of schedule σ .

The amount of concurrency achievable with a schedule is then defined to be given by the function

$$\gamma_s: \Sigma \rightarrow \mathbb{R}$$

such that

$$\gamma_s(\sigma) = \overline{\lim}_{q \rightarrow \infty} \gamma(\sigma, q) ,$$

for all $\sigma \in \Sigma$. The reason for the notation $\overline{\lim}$ instead of simply \lim in this definition is that limit alone might not exist for some schedules (however, it does exist for greedy schedules, as we will see in the next section).

4.3. The Concurrency of an Orientation

4.3.1. The Concurrency of Greedy Schedules

Having defined a measure for the concurrency attainable with a schedule, the question of which schedule to follow from a given initial orientation arises. In this section, we define the concurrency of orientation $\omega \in \Omega$ to be given by the function

$$\gamma_o : \Omega \rightarrow R ,$$

such that

$$\gamma_o(\omega) = \max_{\sigma \in \Sigma(\omega)} \gamma_s(\sigma) ,$$

where $\Sigma(\omega) \subseteq \Sigma$ denotes the set of schedules whose first orientation is ω . Clearly, only one such schedule is a member of Σ_g as well, i.e. $|\Sigma(\omega) \cap \Sigma_g| = 1$.

The following is a corollary of Theorem 3.4, and states that the concurrency attainable from orientation ω is the concurrency attainable with the greedy schedule that starts at ω .

Corollary 4.1: For all $\omega \in \Omega$, let $\sigma \in \Sigma(\omega)$ be a greedy schedule. Then $\gamma_o(\omega) = \gamma_s(\sigma)$.

Proof: Immediate from the fact established by Theorem 3.4 that $m_i(\sigma', q) \leq m_i(\sigma, q)$, for all $i \in N$, all $\sigma' \in \Sigma$, and all $q \geq 1$. ■

Next we find an expression for the value of $\gamma_o(\omega)$ in terms of the period reachable from ω in OTD_g . Recall that $\Omega(\alpha_0, \dots, \alpha_{p-1})$ is the set of acyclic orientations from which the period $\alpha_0, \dots, \alpha_{p-1}$ is reachable in OTD_g .

Theorem 4.2: Let $\alpha_0, \dots, \alpha_{p-1}$ be a period in OTD_g in which each node operates m times. Then

$$\gamma_o(\omega) = \frac{m}{p},$$

for all $\omega \in \Omega(\alpha_0, \dots, \alpha_{p-1})$.

Proof: Let σ denote the greedy schedule from ω . By Corollary 4.1,

$$\gamma_o(\omega) = \gamma_s(\sigma) = \lim_{q \rightarrow \infty} \frac{1}{qn} \sum_{i \in N} m_i(\sigma, q),$$

where we have dropped the $\overline{\lim}$ notation from the definition of γ_s , since the limit always exists for greedy schedules. Let α_0 be the first periodic orientation reachable from ω in σ , and assume that it is the k th orientation in σ , $k \geq 1$. For sufficiently large

q , i.e. $q \geq k$, the first q orientations of σ include $\left\lfloor \frac{q-k}{p} \right\rfloor$ repetitions of the period, so

the sum above can be written as $n \left\lfloor \frac{q-k}{p} \right\rfloor m + o(q)$. Similarly, q itself can be written

as $\left\lfloor \frac{q-k}{p} \right\rfloor p + o(q)$. In the limit as $q \rightarrow \infty$, therefore, we have $\gamma_o(\omega) = \frac{m}{p}$. ■

Corollary 4.3: For all $\omega \in \Omega$, $\gamma_o(\omega) \leq \frac{1}{2}$.

Proof: Immediate from Theorem 4.2, if we recall that $m \leq \frac{p}{2}$, i.e. the most frequently that a node can operate in a period is in every other orientation. ■

If G is a tree, then we have, as a consequence of Corollary 3.17, that $\Omega(\alpha_0, \alpha_1) = \Omega$, where α_0, α_1 denotes the single, 2-orientation period in OTD_g . We then have the following corollary of Theorem 4.2.

Corollary 4.4: If G is a tree, then $\gamma_o(\omega) = \frac{1}{2}$, for all $\omega \in \Omega$. ■

4.3.2. A Closed-Form Expression

If G is not a tree, it is possible to obtain an expression for $\gamma_o(\omega)$ that depends only on the graph-theoretic properties of G oriented by ω . We develop such an expression in this section.

We start by recalling that, if K_k is the complete graph on k nodes, $k \geq 1$, then $G[K_k]$ stands for the lexicographic product of G and K_k . We saw previously that this product may be visualized as the graph obtained by replacing each of G 's nodes with an instance of K_k , and then connecting a node in the instance of K_k replacing node $i \in N$ to a node in the instance of K_k replacing node $j \in N$ if and only if i and j are neighbors in G . Here we propose to visualize $G[K_k]$ a little differently. We suggest that it be viewed as the graph obtained by replacing each node in K_k with an instance of G . A node in an instance of G is connected to a node in another instance of G if and only if either they correspond to the same node in G , or they correspond to neighboring nodes in G . This alternative way of looking at $G[K_k]$ is totally

equivalent to the previous one, and can be illustrated by Figure 2.3 of Chapter 2, where we show $G[K_2]$ in the case in which G is the 5-node cycle. In that figure, the outermost and the innermost pentagons correspond to the two instances of G used to build $G[K_2]$. In the sequel, we will look at $G[K_k]$'s node set as composed of the k layers L_0, \dots, L_{k-1} , each of which is an instance of N , G 's node set.

Let α_0 be a periodic orientation of G 's edges, and assume that it is in a period of length p in which each node operates m times. Now we investigate some of the periodicity characteristics of the graph $G[K_m]$. Build an orientation α'_0 of $G[K_m]$'s edges as follows. Edges in the graph induced by one of the layers L_0, \dots, L_{m-1} are oriented by α'_0 as those of G are by α_0 . Edges between nodes in different layers are oriented by α'_0 from the layer with the higher subscript to the one with the lower. Consequently, all sources in α'_0 are nodes in L_{m-1} , while all sinks are in L_0 . If we let $S'_0, \dots, S'_{\lambda'-1}$ denote $G[K_m]$'s sink decomposition according to α'_0 , then we have

$$S'_{\lambda'-1} \subseteq L_{m-1}$$

and

$$S'_0 \subseteq L_0.$$

Lemma 4.5: Orientation α'_0 is in a period of length p in which each node operates exactly once.

Proof: By Corollary 3.13, it suffices to show that each node in S'_0 has at least one neighbor in $S'_{\lambda'-1}$. For recall from Theorem 3.11 that G admits a p -color, m -tuple coloring of its nodes, such that each node with color c_l has at least one neighbor with color $c_{(l-1) \bmod p}$, $0 \leq l \leq p-1$. This coloring is such that a node has color c_0

among its colors if and only if it is a sink. A corresponding p -color, 1-tuple coloring for the nodes of $G[K_m]$ can be obtained as follows. A node in L_{m-1} gets the color with the highest subscript among the m colors assigned to the corresponding node in G , a node in L_{m-2} gets the color with the second highest subscript, and so on. Clearly, each node in S'_0 gets color c_0 . Each such node must have a neighbor with color c_{p-1} , which must be a node in L_{m-1} . That this node is a member of $S'_{\lambda'-1}$ comes from the fact that in G every node with color c_{p-1} is a source, by Theorem 3.11. ■

It is a consequence of Lemma 4.5 that $\lambda'=p$.

Before we proceed to the main result of this section, we need a few more definitions. Let κ denote an undirected cycle in G , i.e. a sequence of nodes $i_1, \dots, i_{|\kappa|}, i_1$, where $|\kappa|$ denotes the number of nodes in the cycle κ . This cycle may be simple or not, i.e. it is possible that more than one of the nodes $i_1, \dots, i_{|\kappa|}$ are actually the same node in N . We say that κ is traversed in the clockwise direction if it is traversed from i_1 to $i_{|\kappa|}$. Otherwise, it is said to be traversed in the counter-clockwise direction. These directions defined, we let $n_{cw}(\kappa, \omega)$ denote the number of edges in κ oriented clockwise by ω , and $n_{ccw}(\kappa, \omega)$ the number of edges in κ oriented counter-clockwise by ω . Notice that a same edge may be counted in both of these quantities, since it may appear more than once in κ , sometimes in the clockwise direction, sometimes in the counter-clockwise direction. Define the ratio

$$\rho(\kappa, \omega) = \min \left\{ \frac{n_{cw}(\kappa, \omega)}{|\kappa|}, \frac{n_{ccw}(\kappa, \omega)}{|\kappa|} \right\},$$

and let K denote the set of G 's undirected cycles.

Lemma 4.6: Consider a cycle $\kappa \in K$. The ratio $\rho(\kappa, \omega)$ is the same for all ω in the same connected component in OTD .

Proof: By Theorem 3.7, each of the connected components of OTD is strongly connected, which means that any two orientations in that component are reachable from each other under Scheduling by Edge Reversal. If ω and ω' are two such orientations, then $n_{cw}(\kappa, \omega) = n_{cw}(\kappa, \omega')$ and $n_{ccw}(\kappa, \omega) = n_{ccw}(\kappa, \omega')$, since the reversal of sinks in κ does not change these numbers. Thence the lemma. ■

Theorem 4.7: If G is not a tree, then $\gamma_o(\omega) = \min_{\kappa \in K} \rho(\kappa, \omega)$, for all $\omega \in \Omega$.

Proof: Let α_0 be the first periodic orientation reachable from ω . By Lemma 4.6, it suffices to show the assertion for α_0 . Also, if the period in which α_0 participates has length p and in it each node operates m times, then it suffices, by Theorem 4.2, to show that

$$\frac{m}{p} = \min_{\kappa \in K} \rho(\kappa, \alpha_0).$$

We first show that $\rho(\kappa, \alpha_0) \geq \frac{m}{p}$ for all $\kappa \in K$. For such, consider the graph formed by the $|\kappa|$ nodes in κ . This graph is a simple cycle with κ nodes (a same node in N may have several instances of it in this graph). Let $\#_{\kappa}(\omega)$ denote the number of sinks in it under orientation ω . Similarly, let $\#_{\kappa, G}(\omega)$ denote the number of sinks in it which are also sinks in G , i.e. members of $sinks(\omega)$. Clearly, we have the inequalities

$$\rho(\kappa, \omega) \geq \frac{\#_{\kappa}(\omega)}{|\kappa|}$$

$$\geq \frac{\#\kappa, G(\omega)}{|\kappa|},$$

for all $\omega \in \Omega$. In particular, if we take the sum on the period $\alpha_0, \dots, \alpha_{p-1}$, we get

$$\begin{aligned} \sum_{\alpha \in \{\alpha_0, \dots, \alpha_{p-1}\}} \rho(\kappa, \alpha) &\geq \sum_{\alpha \in \{\alpha_0, \dots, \alpha_{p-1}\}} \frac{\#\kappa, G(\alpha)}{|\kappa|} \\ &= \frac{m |\kappa|}{|\kappa|} \\ &= m. \end{aligned}$$

Since the left-hand side of the inequality above is equal to $p\rho(\kappa, \alpha_k)$ for any $0 \leq k \leq p-1$, we get the desired inequality

$$\rho(\kappa, \alpha_0) \geq \frac{m}{p},$$

for all $\kappa \in K$.

To complete the proof, we need to show that

$$\min_{\kappa \in K} \rho(\kappa, \alpha_0) \leq \frac{m}{p}.$$

To do this, we resort to the graph $G[K_m]$ introduced earlier, and to its orientation α'_0 , which by Lemma 4.5 is in a period of length p in which each node operates once. By Corollary 3.14, a simple cycle exists in $G[K_m]$ with a number of nodes which is multiple of λ' (recall that $S_0, \dots, S_{\lambda'-1}$ is $G[K_m]$'s sink decomposition according to α'_0). This simple cycle may be traversed as follows. Pick the first node in $S_{\lambda'-1}$, the next in $S_{\lambda'-2}$, etc., until a node in S_0 is picked, then another one in $S_{\lambda'-1}$, and so on, until the simple cycle is closed. If we extend the definition of the ratio ρ to the graph $G[K_m]$, we immediately see that, if κ' is the cycle we just

described, then $\rho(\kappa', \alpha'_0) = \frac{1}{p}$. This is so because, if the traversal we described is in the clockwise direction, then there exist $p-1$ consecutive edges in the clockwise direction for each edge encountered in the counter-clockwise direction, since $\lambda' = p$. This simple cycle κ' in $G[K_m]$ corresponds to a cycle κ , not necessarily simple, in G . To see how the value of $\rho(\kappa, \alpha_0)$ behaves, we notice that, as the cycle κ' is traversed as described above, a traversal also takes place on the cycle κ . This traversal of κ may only encounter edges oriented in the opposite (i.e. counter-clockwise) direction when the traversal of κ' moves from one of the layers L_0, \dots, L_{m-1} to another, since while the traversal is in the same layer the orientations of edges in κ and κ' are the same. Consequently, we see that $\rho(\kappa, \alpha_0) \leq \frac{m}{p}$, thus completing the proof. In Figure 4.1, we offer an illustration of the second part of this proof. Namely, we show the same graph of Figure 2.3 oriented by the respective α'_0 . Dashed edges correspond to the simple cycle κ' discussed above. The outermost pentagon is L_1 , the innermost one being L_0 . ■

Corollary 4.8: If G is not a tree, then let κ^* be a cycle for which $\gamma_\omega(\omega) = \rho(\kappa^*, \omega)$, for some $\omega \in \Omega$. Furthermore, suppose that $\rho(\kappa^*, \omega) = \frac{n_{cw}(\kappa^*, \omega)}{|\kappa^*|}$. Any segment of κ^* oriented in the clockwise direction must be a shortest directed path in G oriented by ω . Similarly, any segment of κ^* oriented in the counter-clockwise direction must be a longest directed path in G oriented by ω .

Proof: By Theorem 4.7, cycle κ^* must exist. By hypothesis, κ^* has at least as many edges oriented in the counter-clockwise direction as it has in the clockwise direction. Then it must be true, by Theorem 4.7, that

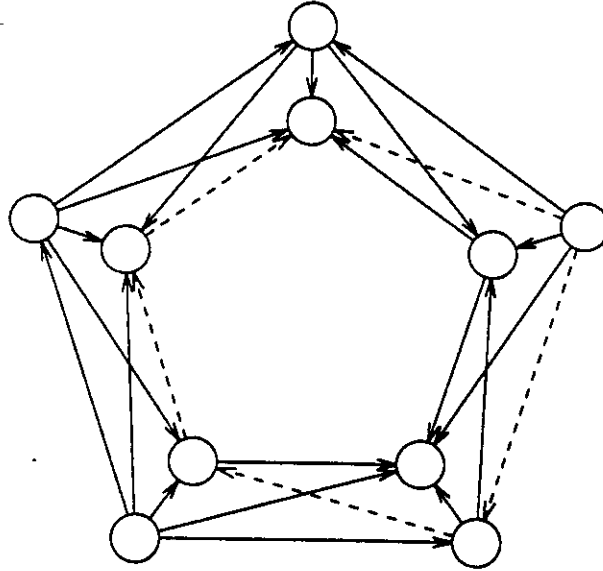


Figure 4.1. Used in the Proof of Theorem 4.7

$$\frac{n_{cw}(\kappa^*, \omega)}{|\kappa^*|} \leq \frac{n_{cw}(\kappa, \omega)}{|\kappa|},$$

for all $\kappa \in K$. Now let l_{cw} be the number of edges in a segment of κ^* oriented by ω in the clockwise direction. Similarly, let l_{ccw} be the number of edges in a segment of κ^* oriented by ω in the counter-clockwise direction. The first claim is that l_{cw} must be the length of a shortest directed path in G oriented by ω . For, if not, then there must be another segment oriented in the clockwise direction between the same two nodes, whose length is $l'_{cw} = l_{cw} - l$, for some $l > 0$. Consequently, there must exist a cycle κ' such that

$$\frac{n_{cw}(\kappa', \omega)}{|\kappa'|} = \frac{n_{cw}(\kappa^*, \omega) - l}{|\kappa^*| - l} < \frac{n_{cw}(\kappa^*, \omega)}{|\kappa^*|},$$

contradicting the assumed optimality of κ^* . The second claim is that l_{ccw} is the length of a longest directed path in G oriented by ω . For otherwise there must exist another segment oriented in the counter-clockwise direction between the same two nodes, with $l'_{ccw} = l_{ccw} + l$ edges, for some $l > 0$. This implies the existence of a cycle κ' such that

$$\frac{n_{cw}(\kappa', \omega)}{|\kappa'|} = \frac{n_{cw}(\kappa^*, \omega)}{|\kappa^*| + l} < \frac{n_{cw}(\kappa^*, \omega)}{|\kappa^*|},$$

again contradicting the assumed optimality of κ^* . ■

Let $K_s \subseteq K$ denote the set of all of G 's simple cycles. In the next corollary, we show that the result of Theorem 4.7 can actually be made more precise.

Corollary 4.9: If G is not a tree, then $\gamma_o(\omega) = \min_{\kappa \in K_s} \rho(\kappa, \omega)$, for all $\omega \in \Omega$.

Proof: Let $\kappa \in K$ be a cycle for which the minimum stated in Theorem 4.6 is achieved. We show that there must be a simple cycle enclosed in κ for which the same ratio is achieved. For let κ consist of the nodes $i_1, \dots, i_{|\kappa|}$. We can partition these nodes into simple cycles as follows. Starting at i_1 , let a simple cycle be formed whenever a node, say j_{i_1} , is visited twice for the first time. From this node, let another simple cycle be formed when a node found after j_{i_1} is repeated for the first time. Let this node be j_{i_2} . If this process is continued, a number, say l , of such nodes will be found, until all of the $|\kappa|$ nodes are exhausted and i_1 is reached again. The cycle κ is then constituted by $l+1$ edge-disjoint simple cycles, l of them formed when the nodes j_{i_1}, \dots, j_{i_l} are found, the $l+1$ st being $i_1, \dots, j_{i_1}, \dots, j_{i_l}, \dots, i_1$.

Clearly, because these simple cycles are edge-disjoint, at least one of them must yield a ratio no larger than that of κ . Because in κ a minimum ratio is achieved, these simple cycles whose ratios are no larger than κ 's have actually a ratio which is the same as κ 's. ■

4.3.3. The Length of a Period

Consider one of G 's simple cycles, say κ , for which the minimum in Corollary 4.9 is achieved. Let G_κ denote the graph induced by the nodes in κ . If ω is an orientation of G 's edges, let ω_κ be the corresponding orientation of G_κ 's edges. One immediate consequence of Corollary 4.9 is that $\gamma_o(\omega_\kappa) = \frac{m}{p}$, if this is the value of $\gamma_o(\omega)$. By Theorem 3.11, ω_κ is itself in a period of length p in which each of G_κ 's nodes operates m times. What this means is that, as far as periodicity is concerned, the whole graph G behaves as one of its simple cycles. In other words, there is a simple cycle in G whose number of nodes is a multiple of p . Consequently, the value of p for any graph which is not a tree cannot exceed the size of its largest simple cycle.

These observations were previewed in Chapter 3, where we also pointed out that this bound on the length of a period readily yields a method for telling whether an orientation is periodic or not in $O(n^3)$ steps.

4.4. The Concurrency of a Graph

4.4.1. The Optimality of Scheduling by Edge Reversal

We have seen that the question of how much concurrency can be achieved in the synchronous model from an initial orientation ω amounts essentially to finding the quantity $\gamma_o(\omega)$, which is also the amount of concurrency provided by the greedy synchronous schedule from ω . One question that remains to be answered is how much concurrency G can provide, i.e. what the optimal initial orientation is in terms of the value of $\gamma_o(\omega)$.

We define the concurrency of a graph to be given by the number $\gamma^*(G)$ such that

$$\gamma^*(G) = \max_{\omega \in \Omega} \gamma_o(\omega).$$

In this section we present a criterion according to which Scheduling by Edge Reversal is optimal. For such, consider a q -color, k -tuple coloring of G 's nodes using the colors c_0, \dots, c_{q-1} . Let it be called ξ . In our synchronous environment, one possible way to schedule independent sets of G for operation at each of the steps would be to let all nodes with color c_0 operate, then those with color c_1 , and so on. The concurrency attained with this scheduling would be $\frac{k}{q}$. As we will see in the next section, there are colorings of this type which would provide a concurrency larger than $\gamma^*(G)$.

Now consider two neighboring nodes i and j in G . Let the q -color, k -tuple coloring described above assign colors c_{i_1}, \dots, c_{i_k} and c_{j_1}, \dots, c_{j_k} to nodes i and j , respectively. Assume without loss of generality that $i_1 < \dots < i_k$ and that

$j_1 < \dots < j_k$. We say that this coloring is *interleaved* if and only if either $i_1 < j_1 < i_2 < j_2 < \dots < i_k < j_k$, or $j_1 < i_1 < j_2 < i_2 < \dots < j_k < i_k$.

Theorem 4.10: If ξ is interleaved, then $\gamma^*(G) \geq \frac{k}{q}$.

Proof: Orient G by an orientation ω such that $\omega((i, j)) = j$ if and only if $j_1 < i_1$. All sinks with color c_0 are sinks in ω . If we reverse all of them, we get an orientation in which all nodes with color c_1 are sinks. Reversing all of these sinks, the fact that ξ is interleaved implies that the orientation obtained is such that in it all nodes with color c_2 are sinks, and so on. This sequence of orientations in which all nodes with color c_0 operate, then those with color c_1 , etc., is therefore an Edge Reversal schedule. The ratio $\frac{k}{q}$ is the concurrency of this schedule, which must then be no larger than $\gamma^*(G)$, by definition. ■

As a consequence of Theorem 4.10, $\frac{1}{\gamma^*(G)}$ may be called the *interleaved multichromatic number* of G , in the spirit of the definition of $\chi^*(G)$.

4.4.2. Concurrency, Chromatic and Multichromatic Numbers

In this section, we establish a relation between $\gamma^*(G)$ and G 's chromatic and multichromatic numbers, denoted by $\chi(G)$ and $\chi^*(G)$, respectively, and discussed in Chapter 2.

Theorem 4.11: $\frac{1}{\chi(G)} \leq \gamma^*(G) \leq \frac{1}{\chi^*(G)}$.

Proof: The first inequality follows immediately from Theorem 4.10, since any 1-tuple coloring of G 's nodes is interleaved. To see the second inequality, recall

that $\gamma^*(G) = \frac{k}{q}$ for some q -color, k -tuple coloring, thence

$$\begin{aligned}\gamma^*(G) &\leq \frac{k}{\chi^k(G)} \\ &\leq \frac{1}{\chi^*(G)}.\end{aligned}$$

■

By Theorems 4.10 and 4.11, any q -color, k -tuple coloring of G 's nodes which is not interleaved must be such that

$$\gamma^*(G) < \frac{k}{q} \leq \frac{1}{\chi^*(G)}.$$

4.4.3. The Concurrency of Bipartite Graphs, Cycles, and Complete Graphs

In this section, we investigate the behavior of the quantity $\gamma^*(G)$, when G is taken to be a complete graph, a bipartite graph, or a cycle. For these graphs, we also discuss how $\gamma^*(G)$ relates to $\chi(G)$ and $\chi^*(G)$. Values of $\chi^*(G)$ that we report are from [Stah76].

The simplest case is that in which G is a complete graph, since by Theorem 3.20 we have that $\gamma_\omega(\omega) = \frac{1}{n}$ for all $\omega \in \Omega$. Consequently, we have $\gamma^*(G) = \frac{1}{n}$. Also, we have that $\chi(G) = n$ and $\chi^*(G) = n$. Therefore, for a complete graph,

$$\frac{1}{\chi(G)} = \gamma^*(G) = \frac{1}{\chi^*(G)} = \frac{1}{n}.$$

If G is a bipartite graph, then $\chi(G) = 2$ and $\chi^*(G) = 2$. Also, we can obtain an acyclic orientation of its edges by orienting all edges from one of the two partitions

to the other. Such an orientation is clearly periodic, by Corollary 3.13, with $p=2$ and $m=1$. By Theorem 4.2, $\gamma_o(\omega)=\frac{1}{2}$, and by Corollary 4.3, $\gamma^*(G)=1/2$. So we have

$$\frac{1}{\chi(G)} = \gamma^*(G) = \frac{1}{\chi^*(G)} = \frac{1}{2},$$

if G is a bipartite graph. This case includes trees (in accordance with Corollary 4.4), and cycles with an even number of nodes.

As an interesting example, we consider the hypercubic network [Seit85]. This is a bipartite graph, and therefore achieves as much concurrency as is possible according to our definitions and our constraints. A hypercube has 2^d nodes, all of the same degree d , for some $d \geq 1$. Figure 4.2 shows two hypercubes, for $d=3$ and $d=5$, respectively. The numbers shown next to each node are useful in figuring out the interconnection pattern that yields the hypercubic configuration. Two nodes are connected to each other if and only if their numbers differ in exactly one bit of their binary representations.

When G is a cycle, we use Corollary 4.9 to find the optimal value of $\gamma_o(\omega)$. If n is even, such an optimal orientation orients $\frac{n}{2}$ edges in the clockwise direction and $\frac{n}{2}$ in the counter-clockwise direction. In this case, $\gamma^*(G)=\frac{1}{2}$, in full conformity with the previous paragraph on bipartite graphs.

If n is odd, then optimality is achieved by orienting $\frac{n-1}{2}$ edges in one direction, and the other $\frac{n+1}{2}$ in the opposite direction. We then get $\gamma^*(G)=\frac{n-1}{2n}$, and since we have $\chi(G)=3$ and $\chi^*(G)=\frac{2n}{n-1}$, we get

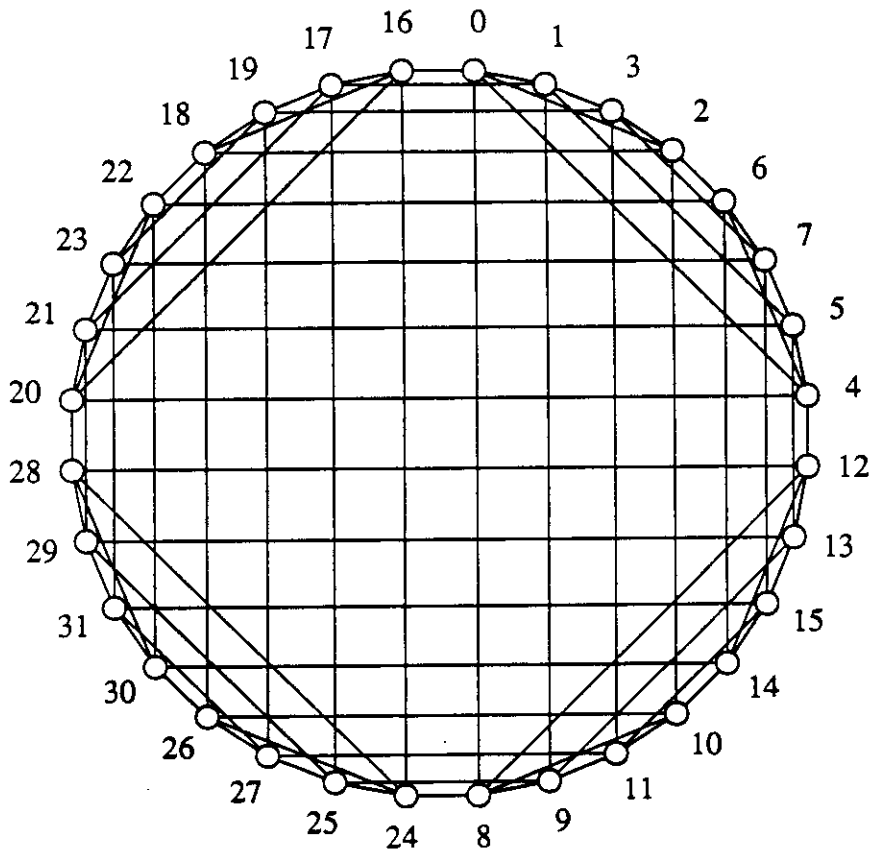
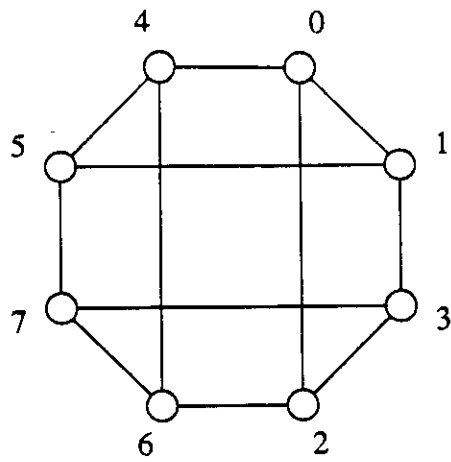


Figure 4.2. A 3-Cube and a 5-Cube

$$\frac{1}{3} = \frac{1}{\chi(G)} < \gamma^*(G) = \frac{1}{\chi^*(G)} = \frac{n-1}{2n}.$$

In Figure 4.3, we show optimal orientations (which also happen to be periodic, by Lemma 3.18) of a 6-node and a 5-node cycle.

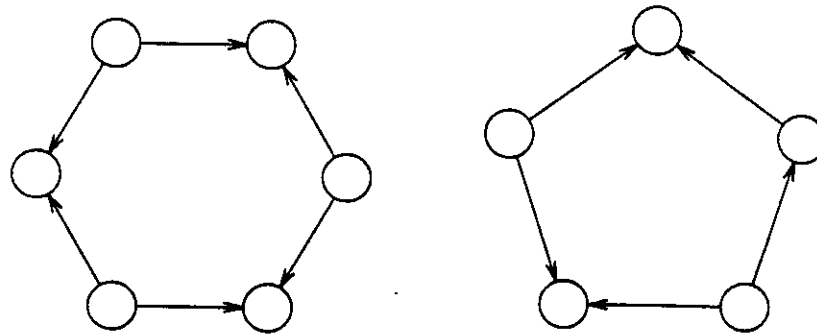


Figure 4.3. Optimal Orientations of a 6-Node and a 5-Node Cycle

4.4.4. Concurrency and Multichromatic Number May Be Distinct

In all of the examples in the previous section, it is the case that $\gamma^*(G) = \frac{1}{\chi^*(G)}$. Since the problem of finding $\chi^*(G)$ has been shown in [Gröt81] to be *NP*-hard (see [Gare79]), establishing the fact that the two quantities are in general the same would provide an immediate proof of the *NP*-hardness of the problem of finding $\gamma^*(G)$ as well. Though it is indeed the case that this problem is intractable, as we show in the next section, here we wish to provide an example for which the equality above does not hold. Consider the graph G of Figure 4.4. In [Stah76], it is shown that G 's k -chromatic number $\chi^k(G)$ is given by

$$\chi^k(G) = 2k+1 + \left\lfloor \frac{k-1}{2} \right\rfloor$$

$$= \begin{cases} 2k+1 + \frac{k-1}{2}, & \text{if } k \text{ is odd,} \\ 2k+1 + \frac{k-2}{2}, & \text{if } k \text{ is even.} \end{cases}$$

As a result, we have

$$\frac{\chi^k(G)}{k} = \begin{cases} \frac{5}{2} + \frac{1}{2k}, & \text{if } k \text{ is odd,} \\ \frac{5}{2}, & \text{if } k \text{ is even.} \end{cases}$$

Therefore, the value that we get for $\chi^*(G) = \min_{k \geq 1} \frac{\chi^k(G)}{k}$ is $\chi^*(G) = \frac{5}{2}$. Also, we have

that the value for $\gamma^*(G)$ is $\gamma^*(G) = \frac{3}{8}$, and as a consequence it is established that

$$\gamma^*(G) \neq \frac{1}{\chi^*(G)}.$$

For the graph of Figure 4.4, it can be easily seen that $\chi(G) = 3$, and we have

$$\frac{1}{\chi(G)} < \gamma^*(G) < \frac{1}{\chi^*(G)}.$$

These strict inequalities show that in some cases $\gamma^*(G)$ is different from both $\frac{1}{\chi(G)}$

and $\frac{1}{\chi^*(G)}$.

4.4.5. Intractability of Maximizing Concurrency

Consider the definition of Problem Π below.

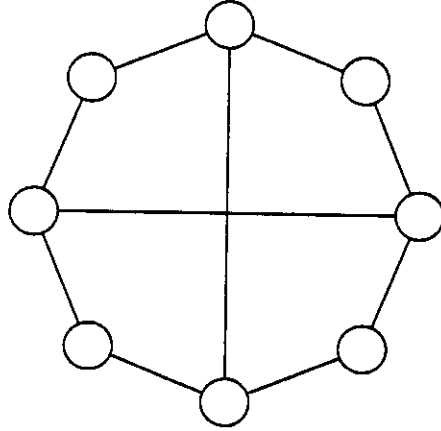


Figure 4.4. An Example for which $[\chi(G)]^{-1} < \gamma^*(G) < [\chi^*(G)]^{-1}$

Problem Π : Given the graph $G=(N,E)$ and the integers $p \geq 2$ and $m \geq 1$, is there an orientation $\omega \in \Omega$ of G such that $\gamma_o(\omega) > \frac{m}{p}$?

Π is the decision problem corresponding to the optimization problem of finding $\gamma^*(G)$. We show in this section that Π is *NP*-complete.

Lemma 4.12: There exists a polynomial-time algorithm to find $\gamma_o(\omega)$, for any $\omega \in \Omega$.

Proof: By Theorem 4.7, it suffices to show that $\min_{\kappa \in K} \rho(\kappa, \omega)$ can be found in polynomial time, for any $\omega \in \Omega$. We describe an algorithm which operates on the following table. There is one row for each rational $\frac{x}{y}$, where x is the number of edges

oriented in a path in the direction in which the path is traversed, and y is the total number of edges in it. Clearly, no more than n^2 such values can exist, and so this is the number of rows in the table. Each row is indexed by a tuple of the form (x,y) , where $\frac{x}{y}$ is the corresponding value of the ratio that it stands for, $1 \leq x,y \leq n$. Our table has one column for each of the n^2 ordered pairs of nodes in G . Each column is therefore indexed by the pair (i,j) , $i,j \in N$.

The algorithm we propose proceeds in stages, and the table is updated from stage to stage. At stage k , the entry $(x,y),(i,j)$ of the table contains an integer k' if and only if k' is the largest integer such that $1 \leq k' \leq k$ and: (1) there is a path from i to j in G containing k' segments of which the first, third,... are oriented in the direction in which the path is traversed from i to j , whereas the second, fourth,... are oriented in the opposite direction; and (2) the ratio of the number of edges oriented in the direction of the traversal to the total number of edges in the path is equal to $\frac{x}{y}$. If no such k' exists in the range specified, the entry is set to infinity. Clearly, no more than n stages are needed, since Corollary 4.9 allows us to ignore any candidate $k \in K$ which is not a simple cycle. By Corollary 4.8, each of the segments comprising the path described above has to be either a shortest directed path, if it is traversed in the forward direction, or a longest directed path, if traversed in the backward direction.

The algorithm is initialized by computing, for each ordered pair of nodes (i,j) , the shortest and the longest directed paths from i to j in G oriented by ω . These values are kept in two separate arrays, and are used during the algorithm to build the table. At the same time, the values of the shortest paths are themselves stored in the table, as follows. Entry $(x,x),(i,j)$ of the table gets value 1 if the shortest directed

path from i to j has length x . All other entries get value infinity. This constitutes the first stage of the algorithm. At the k th stage, $1 < k \leq n$, entry $(x, y), (i, j)$ of the table gets value k if there is a node, say l , such that: (1) entry $(x', y'), (i, l)$ of the table had value $k-1$ at the end of the previous stage; and (2) either $x-x'$ is the length of the shortest directed path from l to j , if k is odd (in this case, $y-y'=x-x'$), or $y-y'$ is the length of the longest directed path from j to l , if k is even (in this case, $x=x'$). At the end of the n th stage, each column of the form (i, i) is scanned and the minimum ratio is selected among the rows which have finite values in that column. Then the minimum of these minima is taken.

Clearly, each stage of the algorithm takes $O(n^4)$ time. The algorithm, therefore, operates in a time which is $O(n^5)$. ■

We now let Π' denote the usual Colorability Problem [Karp72] (known to be NP -complete), given by the following formulation.

Problem Π' : Given the graph $G'=(N', E')$ and the positive integer $K \leq n$, does G' admit a 1-tuple coloring of its nodes using K colors?

In Theorem 4.13 below, we use Problem Π' in showing that Problem Π is NP -complete.

Theorem 4.13: Problem Π is NP -complete.

Proof: In order to show that Π is NP -complete, we have to show that it is a member of NP and that all problems in NP are polynomially reducible to it [Karp72, Gare79]. That $\Pi \in NP$ is directly implied by Lemma 4.12. Notice that this assertion relies essentially on the closed-form expression found earlier for the concurrency of

an orientation. In this sense, this membership in NP constitutes the hardest part of this proof, unlike most NP -completeness proofs.

We show that all problems in NP are polynomially reducible to Π by showing that there is an NP -complete problem which is polynomially reducible to Π . We do so by showing that $\Pi' \leq \Pi$, i.e. that Problem Π' is polynomially reducible to Problem Π .

Consider the instance of problem Π' consisting of the graph $G'=(N',E')$ and the positive integer $K \leq |N'|$. Let $G=(N,E)$, together with the integers $p \geq 2$ and $m \geq 1$, be the instance of problem Π obtained as follows. G is obtained from G' by adding a new node, say k , to N' , and connecting it to all other nodes in N' , i.e. $N=N' \cup \{k\}$ and $E=E' \cup \{(k,i), i \in N'\}$. Furthermore, let $p=K+2$ and $m=1$. The action of problem Π on this instance consists of deciding whether there is an orientation ω of G such that $\gamma_\omega(\omega) > \frac{1}{K+2}$. By Corollary 3.13, all periods in G 's state space are such that each node operates only once. This is seen by noting that, in any such period, the orientation resulting from an operation of node k , whose sink decomposition we denote by $S_0, \dots, S_{\lambda-1}$, is such that each node in S_0 has as neighbor the node in $S_{\lambda-1}=\{k\}$. For this reason, Π actually decides whether there is an orientation ω of G whose sink decomposition has length smaller than $K+2$ or, equivalently, by the remarks in Chapter 2, whether G 's nodes can be colored by less than $K+2$ colors. This is true if and only if the nodes of G' can be colored by less than $K+1$ colors, i.e. by K colors.

■

CHAPTER 5

EDGE REVERSAL IN AN ASYNCHRONOUS MODEL

5.1. Introduction

In this chapter we present a discussion of Scheduling by Edge Reversal in an asynchronous environment. Our asynchronous model is such that each node has access to a local clock only, and therefore no global timing basis exists. Messages sent over communication channels remain in transit for a finite, though arbitrary, time. In this model, the following is how Scheduling by Edge Reversal functions. Sinks in the initial acyclic orientation operate and send messages out on their incident channels, in order to reverse the orientation of the corresponding edges. Whenever a node gets similar messages on all of its incident channels, it proceeds likewise. As in the synchronous model, these messages may carry additional information, depending on the application at hand.

In Section 5.2, we present details of the model for the case of a generic asynchronous computation. The concepts of events, orders, local and global states are reviewed, following [Lamp78, Brac84, Chan85]. We also discuss the representation of this event system by a precedence graph, i.e. a directed acyclic graph which represents the interdependence among events. Section 5.3 specializes the concepts described in the preceding section to the case of Scheduling by Edge Reversal. We discuss how this affects the notions of events, orders and states, and discuss another precedence graph that can be used in this case. To finalize, we discuss the concepts

of an execution and of the timing of an event. We show that the synchronous model of Chapter 3 can be regarded as a special case of the asynchronous model described in this chapter. A concurrency measure is then presented for the asynchronous case, and we relate it to the synchronous measures discussed in Chapter 4.

5.2. Events and Global States: the General Case

5.2.1. Events and Orders

Let Φ_i be the sequence of *states* of node i in a computation. Notice that it is in perfect accordance with our assumptions to talk about the state of a node at a given time, where it is understood that time means local time, as given by that node's clock. In the beginning of a computation, each node is in its *initial* state. As the computation progresses, a node's state changes, and these changes are caused by *events*. At the end of the computation at node i , i.e. when no more events involve node i , we say that node i is in its *final* state.

An event v has associated with it a node i and a time t , as given by i 's clock, at which v happened. It may have been triggered by a message from one of i 's neighbors, or may be a spontaneous internal event. Its effect is to make node i change its state from a member of Φ_i to the next. Finally, it may or may not cause i to send a message to one or more of its neighbors. We assume that it takes no time for an event to happen, from the receipt of the message that triggers it, if any, to the eventual sending of messages, if any. In the sequel, we let V denote the set of events for a given computation.

Despite the absence of a concept of global time in our model, there is still some causality among the events. Define the following relation, called *BEFORE*, on the set V of events. Two events $v, v' \in V$ are such that v *BEFORE* v' if and only if: (i) v and v' involve the same node, and v occurs before v' , without any other event happening in the meantime, or (ii) a message is sent by v and received at some neighboring node by v' . Notice that there are at most two events v such that v *BEFORE* v' , for any event v' . Let B denote the transitive closure of *BEFORE*. B is a partial order of the events in V , since it is irreflexive and transitive [Ende77].

Let V_i denote the set of events which happened at node i , $i \in N$. Each of the sets V_i is totally ordered by the relation B . We call $v \in V_i$ the *first* event at node i if and only if no other event $v' \in V_i$ exists such that $v' B v$. Similarly, it is the *last* event at node i if and only if no event $v' \in V_i$ exists such that $v B v'$. In the same vein, let ϕ_i be the state of node i at some local time. We denote by $se(\phi_i)$ the event, if any, that happens at node i when it is in state ϕ_i , and by $pe(\phi_i)$ the event, if any, that happens at node i resulting in its transition to state ϕ_i .

An event v' is in the *past* of event v if and only if it is a member of

$$PAST(v) = \{v' \in V \mid v' B v\},$$

and is in its *future* if and only if it is a member of

$$FUTURE(v) = \{v' \in V \mid v B v'\}.$$

5.2.2. Global States

In this section, we review the concept of a *global state* (or *snapshot*) of the system. A *system state* is defined to be an n -tuple whose i th component is a member

of Φ_i . This definition is usually extended to include the states of the communication channels as well. Here, however, we restrict ourselves to discussing node states only, since this serves our purposes quite adequately. Below we characterize system states which are global states.

Let ϕ be a system state and ϕ_i the corresponding local state of node $i \in N$. Define the past of ϕ to be the set of events

$$PAST(\phi) = \bigcup_{i \in N} \left[\{pe(\phi_i)\} \cup PAST(pe(\phi_i)) \right],$$

and its future to be

$$FUTURE(\phi) = \bigcup_{i \in N} \left[\{se(\phi_i)\} \cup FUTURE(se(\phi_i)) \right].$$

This system state is a global state if and only if

$$v \in PAST(\phi) \rightarrow PAST(v) \subseteq PAST(\phi).$$

In other words, ϕ is a global state if and only if the past of an event in the past of ϕ is contained in the past of ϕ . The set of global states will be denoted by $\Psi \subseteq \Phi$. Likewise, a generic member of Ψ will be denoted by ψ .

5.2.3. Precedence Graphs

All the concepts discussed so far in this chapter can be better understood if we consider a graph representation of them. In this section, we discuss a *precedence graph* that represents the interdependence among events, and in which the concept of a global state can be clearly visualized.

The precedence graph we consider is the acyclic directed graph $PG=(V,BEFORE)$. That is, PG 's set of vertices is the set of events V , and its directed edges are specified by the relation $BEFORE$ as follows. If v and v' are members of V , then an edge exists in PG from v to v' if and only if v $BEFORE$ v' . As a consequence, the in-degree of any vertex in PG is at most 2. In PG all events $v \in V_i$ form a directed chain from the first event in i to the last. Each edge in this chain can be thought of as representing the state of node i between the events represented by the endpoints of that edge. Notice that this representation does not include an edge to represent either a node's initial or final state. Edges exist between these chains as dictated by the messages sent between nodes.

An example of a precedence graph is given in Figure 5.1. Each dashed horizontal line represents a local time axis for a node. Local time increases from the left to the right. The rest of the drawing represents the precedence graph PG for some computation. We have exemplified three candidates for global states. Each of these candidates is represented by a vertically oriented line, each of which crosses a number of edges. The chain edges crossed by one such line stand for the local state of the corresponding node in that candidate to global state. If a chain is crossed without an edge in that chain being crossed (e.g. line (a)), then the corresponding node is either in its initial or final state in that system state. The interchain edges crossed by the line should represent messages which would be in transit in that global state, had we included the state of communication channels in our definition of a system state. By our definitions of a global state, one of these vertically oriented lines is a global state if and only if it is not crossed by any edge from the right to the left. It is clear that the lines labeled (a) and (c) in the figure are global states, whereas the one labeled (b) is not.

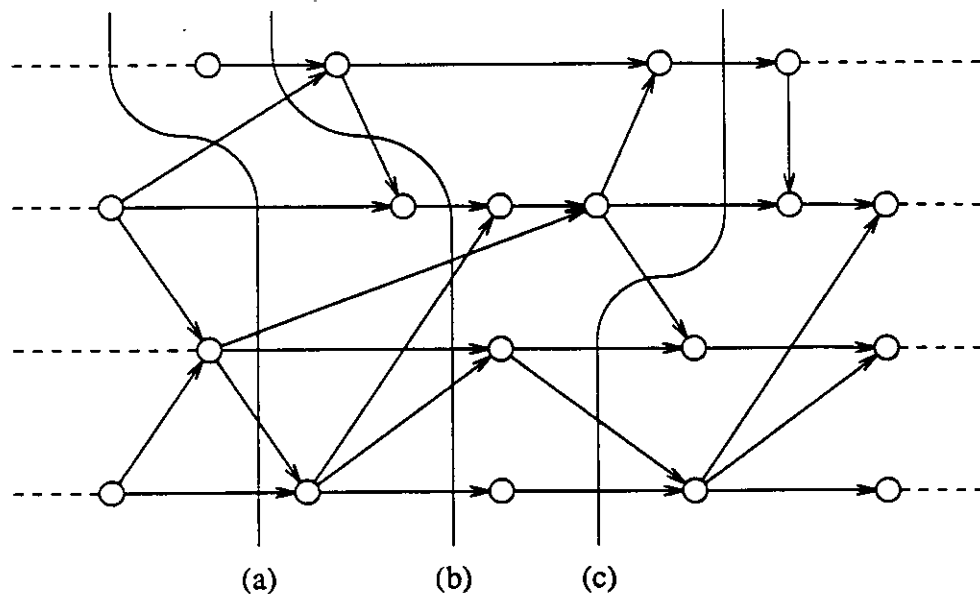


Figure 5.1. Precedence Graph for a Generic Asynchronous Computation

5.3. Events and Global States: the Case of Edge Reversal

5.3.1. Events, Orders, and Global States

We now restrict ourselves to asynchronous computations governed by Scheduling by Edge Reversal. We assume that a node's state may only change when the node operates. In other words, if a node receives a message from one of its neighbors which is not the last message it has to wait for before operating, then the event corresponding to this message does not change its state. Based on this assumption, we may classify the events in V into the following two types. A *type-1 event*

constitutes the receipt of a message at a node, which is not the last message that node has to wait for in order to operate. As a consequence, its state does not change, and no message is sent out. In a typical *type-2 event*, a node receives the last message for which it is waiting, operates, and then changes its local state. A message is then broadcast to all of its neighbors. In the sequel, V_1 and V_2 denote the set of type-1 and type-2 events, respectively.

Because of the assumption that a node's state may only change when a type-2 event happens, we see that two global states are distinct from each other if and only if at least one type-2 event in the past of one of them is in the future of the other. This motivates the following theorem.

Theorem 5.1: In an asynchronous computation in which nodes are scheduled for operation by Edge Reversal, at most $\frac{|V_2|+n}{n}2^{n-1}$ distinct global states may exist.

Proof: Let ϕ_i be a local state of node i , and consider the number of distinct global states ψ such that $\psi_i=\phi_i$, i.e. the number of distinct global states in which node i participates with its local state ϕ_i . We show that this number is no greater than 2^{n-1} . A global state involving ϕ_i can be constructed as follows. Visit a neighbor of i , and select one of its local states which can participate with ϕ_i in a global state. Then a neighbor of that neighbor, and so on. At each step, at most two distinct local states can be considered for inclusion in the global state. Since the distance from i to another node is in the worst case equal to $n-1$ edges, we get the number 2^{n-1} of possible distinct global states in which ϕ_i participates. Considering that there are at most $|V_2|+n$ distinct local states during the computation (the n initial states, plus

the states generated by type-2 events), we see that there may be no more than

$$\frac{|V_2|+n}{n}2^{n-1}$$

distinct global states. The division by n comes from the fact that in the number

$$(|V_2|+n)2^{n-1},$$

each global state is accounted for n times. ■

5.3.2. A Modified Precedence Graph

Because of the assumption that only type-2 events may change the local states of nodes, it becomes convenient to define a precedence graph whose set of vertices is V_2 . We denote this graph by $PG'=(V_2,BEFORE)$. Each of its vertices is a type-2 event. Also, if $v,v' \in V_2$, the directed edge $(v \rightarrow v')$ exists if and only if d type-1 events, say $v_{i_1}, \dots, v_{i_d} \in V_1$, can be found such that

$$v \text{ BEFORE } v_{i_1} \text{ BEFORE } \dots \text{ BEFORE } v_{i_d} \text{ BEFORE } v'$$

for some $d \geq 0$. The graph PG' can be viewed as obtained from PG by lumping together all consecutive type-1 events in a same set V_i to the succeeding type-2 event in that same set, if any. Notice that a vertex in PG' typically has both its in-degree and its out-degree equal to 1 plus the degree of the node in G at which the corresponding event happened.

We illustrate the construction of this graph in Figure 5.2, where three graphs are shown. From the top, one finds G , then the corresponding PG , then the corresponding PG' . The initial orientation of G is that shown in its drawing. In the drawing of PG , events enclosed in an ellipse are replaced in PG' with one single vertex. One should notice that global states can still be graphically represented in PG'

as they can in PG , through the use of vertically oriented lines that separate its vertex set. However, each line in PG' may correspond to more than one in PG . All of these lines in PG , however, correspond to the same global state. As a consequence, the lines that we can draw in PG' suffice to represent all possible distinct global states, so in this aspect nothing is lost in moving from PG to PG' . As an example, in Figure 5.2 we show two such lines in PG and the corresponding line in PG' .

5.3.3. Executions and the Timing of Events

In this section, we investigate the impact of message delays on the *timing* of events. We assign different delays to messages, and investigate the effects of doing this. When it takes messages different times to be delivered, the structure of PG may change. Nevertheless, because the computation is based on Scheduling by Edge Reversal, the relative order of the type-2 events remains unchanged. What may change is how type-1 events get organized in between two type-2 events. The structure of PG' , the graph in which only type-2 events participate, remains the same.

Define an *execution* θ to be an assignment of finite real numbers to messages, and denote by Θ the collection of all such assignments. Each of the members of Θ is a possible scenario for the occurrence of the events in V . We restrict these real numbers assigned to messages to being in the interval $[0, 1]$.

Each execution $\theta \in \Theta$ is an assignment of real numbers in the interval $[0, 1]$ to the edges in PG' which correspond to messages being sent between nodes. If we assume that all other edges get value 0, then we define the timing $\tau_\theta(v)$ of a type-2 event v under execution θ to be the weighted length of a longest path in PG' from a source to v .

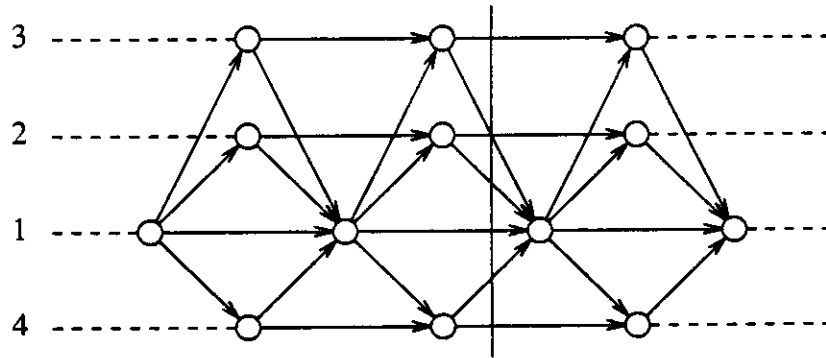
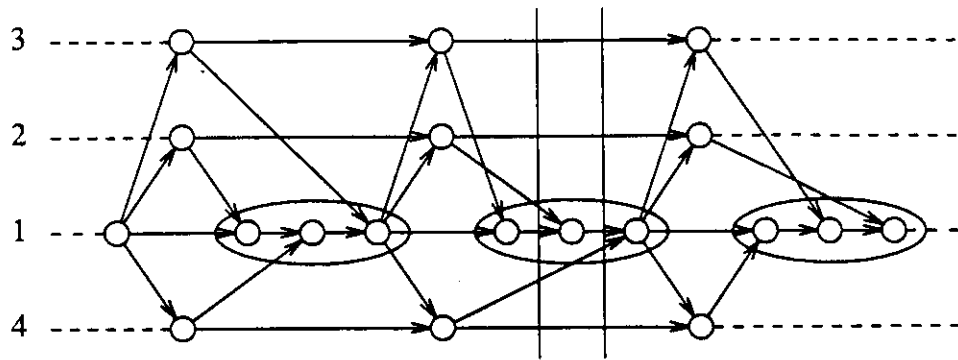
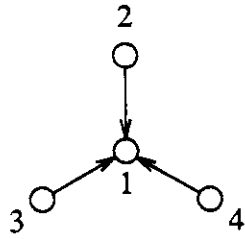


Figure 5.2. Precedence Graphs under Scheduling by Edge Reversal

In the spirit of Chapter 3, we now define $m_i(\theta, t)$ to be the number of times that node i operates under execution θ “up to time t .” More precisely, the value of $m_i(\theta, t)$ is the number of type-2 events v happening at node i such that $\tau_\theta(v) < t$. In the next section, we investigate some consequences of this definition.

5.3.4. Greedy Synchronous Operation

Consider the execution $\theta_s \in \Theta$ which assigns value 1 to all messages. One immediately notices that $\tau_{\theta_s}(v)$ is a nonnegative integer for all $v \in V_2$. In fact, if we let ω_1 denote the initial acyclic orientation of G , and furthermore let $\sigma_g \in \Sigma_g$ be the greedy synchronous schedule from ω_1 , then execution θ_s can be thought of as the realization in an asynchronous environment of the greedy synchronous schedule σ_g . To see this, recall from Chapter 3 that, for all $i \in N$,

$$m_i(\theta_s, 0) = m_i(\sigma_g, 1) = 0,$$

and for all $i \in N$ and all $q \geq 1$,

$$m_i(\theta_s, t) = m_i(\sigma_g, q+1),$$

where $q-1 \leq t < q$. In other words, the number of times that node i operates in the $q+1$ first orientations of the greedy synchronous schedule σ_g is the same as it does in execution θ_s up to time t , if $q-1 \leq t < q$.

Theorem 5.2 below provides us with a counterpart of Theorem 3.4. It states that nodes will not operate more frequently in an execution in which all messages take as much time as possible to be delivered than in any other execution.

Theorem 5.2: $m_i(\theta_s, t) \leq m_i(\theta, t)$, for all $i \in N$, all $\theta \in \Theta$, and all $t \geq 0$.

Proof: Let t be a nonnegative integer, and t' be a nonnegative real number such that $t \leq t' < t+1$. Since the timings of all type-2 events are nonnegative integers in θ_s , we see that $m_i(\theta_s, t) = m_i(\theta_s, t')$ for all $i \in N$. For all other executions $\theta \in \Theta$, it is true that $m_i(\theta, t) \leq m_i(\theta, t')$, and therefore it suffices to prove the theorem for $t \in \{0, 1, 2, \dots\}$.

We use induction on t . For $t=0$, the theorem holds trivially with equality. Assume that it holds for some integer $t \geq 0$. We then show it for $t+1$. It is sufficient to show that, if $m_i(\theta_s, t) = m_i(\theta, t)$ for some $\theta \in \Theta$, and i operates from t to $t+1$ in θ_s , then it must also operate in θ in the same interval. For let v be the event corresponding to i 's operation from t to $t+1$ in θ_s . Also, let v' be the event corresponding to the first operation of node i after t in θ . Clearly, we have $\tau_\theta(v') \leq \tau_{\theta_s}(v)$, and so v' must take place between t and $t+1$ as well. ■

5.3.5. A Measure of Concurrency

In this section we give a measure of concurrency for Scheduling by Edge Reversal in an asynchronous model. The definition we give for the concurrency attainable in the asynchronous model from an initial orientation ω is entirely analogous to the one provided in Chapter 4, and therefore we proceed directly to its final form. It will be a function of the form

$$\gamma_a : \Omega \rightarrow R ,$$

such that

$$\gamma_a(\omega) = \min_{\theta \in \Theta} \overline{\lim}_{t \rightarrow \infty} \frac{1}{(t+1)n} \sum_{i \in N} m_i(\theta, t) ,$$

for all $\omega \in \Omega$. The reason for the $t+1$ in the denominator, instead of simply t , is that we allow $t=0$. We can think of the ratio

$$\frac{1}{(t+1)n} \sum_{i \in N} m_i(\theta, t)$$

as representing the amount of concurrency achieved under execution θ “up to time t ,” $t \geq 0$.

The measure of concurrency $\gamma_a(\omega)$ indicates the very least amount of concurrency that can be achieved in the asynchronous environment from orientation ω . By Theorem 5.3 below, this minimum amount happens when all messages take as much time as they can to be delivered. It corresponds to the concurrency attainable in the synchronous environment from that same orientation if the duration of each synchronous step is the longest time that a message may spend in transit in the asynchronous model.

Theorem 5.3: For all $\omega \in \Omega$, $\gamma_a(\omega) = \gamma_o(\omega)$.

Proof: Recall that, if θ_s is the execution in which all messages take one unit of time to be delivered, then for all $i \in N$,

$$m_i(\theta_s, 0) = m_i(\sigma_g, 1) = 0,$$

and for all $i \in N$ and all $q \geq 1$,

$$m_i(\theta_s, t) = m_i(\sigma_g, q+1).$$

Here σ_g is the greedy synchronous schedule from ω , and $q-1 \leq t < q$. Theorem 5.2 states that $m_i(\theta_s, t) \leq m_i(\theta, t)$, for all $i \in N$, all $\theta \in \Theta$, and all $t \geq 0$. Consequently, we can actually write

$$\gamma_a(\omega) = \overline{\lim}_{q \rightarrow \infty} \frac{1}{(q+1)n} \sum_{i \in N} m_i(\sigma_g, q+1),$$

which immediately implies that

$$\gamma_a(\omega) = \gamma_o(\omega),$$

for all $\omega \in \Omega$. ■

CHAPTER 6

THE PROBLEM OF IDENTIFYING OPTIMAL GLOBAL STATES

6.1. Introduction

Here we consider the following problem on the asynchronous model of Chapter 5. With each state of a node, associate a real number. Find a global state for which the sum of these numbers over the participating local states is optimal (maximum or minimum). As we remarked in Chapter 1, a solution to this problem allows us to retrieve the minimum found by the distributed variation of the Simulated Annealing method.

We present a precise statement of the problem in Section 6.2. Since an asynchronous computation can be represented by the precedence graph PG introduced in Chapter 5, the problem can be thought of as a problem on precedence graphs. Despite the known NP -completeness of some similar problems on precedence systems [Seth75, Abde76, Abde78, Gare79], we show in this chapter that finding an optimal global state amounts essentially to finding a minimum flow in a modified form of the precedence graph. We discuss this Min-Flow formulation in Section 6.3. In Section 6.4, we concentrate on computations governed by Scheduling by Edge Reversal, and discuss the problem on PG' , the variation of PG introduced in Chapter 5. The Min-Flow formulation of Section 6.3 is of course also applicable here. A Max-Flow formulation is also possible, due to the special structure of PG' . We elaborate on the application to the distributed implementation of Simulated Annealing.

Both the maximum and the minimum flow computations can be performed efficiently in the case of Scheduling by Edge Reversal, by partitioning the graph in a convenient way.

6.2. Statement of the Problem

We first recall some notation from Chapter 5. For a general asynchronous computation, V denotes the set of events in that computation, Φ_i the sequence of states of node $i \in N$, Φ the set of all system states, and Ψ the set of all global states with respect to V . Here we assume that each node state is labeled with a number given by the function

$$g : \bigcup_{i \in N} \Phi_i \rightarrow \mathbb{R} .$$

The problem we wish to discuss is how to find a global state $\psi^* \in \Psi$ such that

$$\sum_{i \in N} g(\psi^*_i) \geq \sum_{i \in N} g(\psi_i) ,$$

for all $\psi \in \Psi$, where we recall that ψ_i denotes the local state of node i in the global state ψ .

One equivalent statement of this problem comes from the following observation. Each global state includes exactly one component for each node in N . For this reason, if $\psi^* \in \Psi$ is a solution to our problem, then it is also a solution if the function g is increased by a real constant on all of the members of Φ_i , for some $i \in N$. In particular, let \bar{g}_i be a real constant such that

$$\bar{g}_i \leq \min\{0, \min_{\phi_i \in \Phi_i} g(\phi_i)\} ,$$

for all $i \in N$. The problem we posed above is equivalent to the following problem.

Find a global state $\psi^* \in \Psi$ such that

$$\sum_{i \in N} h(\psi^*_i) \geq \sum_{i \in N} h(\psi_i),$$

for all $\psi \in \Psi$, where the function h is of the form

$$h : \bigcup_{i \in N} \Phi_i \rightarrow [0, \infty),$$

and is given by

$$h(\phi_i) = g(\phi_i) - \bar{g}_i,$$

for all $\phi_i \in \Phi_i$ and all $i \in N$. In other words, the solution to the original problem is still valid if the function g is replaced with the nonnegative function h . The fact that we can pose the problem using nonnegative labels only is of crucial importance to the solutions we propose in the sequel.

As a final observation, notice that we can also pose the problem as a minimization problem by just replacing g with $-g$. Of course, the same construction to obtain h is still valid. In Section 6.3 we deal with the formulation as a maximization problem, whereas in Section 6.4 we also regard it as a minimization problem.

6.3. Min-Flow Formulation

We can view the problem as a problem on the precedence graph PG . This is a graph which possesses one vertex for each of the events in V , and there are directed edges between vertices if and only if the respective events are related to each other through the relation *BEFORE*. It is an acyclic directed graph with a directed chain of edges for each node in N , from the first event in that node to the last. Events in the chain corresponding to node i are members of the set V_i of events happening at node

i. Each edge in this chain represents the state of node *i* between the two events that serve as endpoints to it. Clearly, a node's initial and final states are not represented by any such edge.

In order to restate the problem in terms of the graph *PG*, we expand the vertex set of *PG* by two vertices *s* and *t*, such that there is a directed edge from *s* to the first event in each node, and from the last event in each node to *t*. We call this expanded graph $H=(J,A)$, where $J=V\cup\{s,t\}$, and *A* is the enlargement of *PG*'s edge set to include the edges directed away from *s* and those directed toward *t*. An *s-t cut* is a subset of *A* which disconnects *s* from *t* by partitioning the vertex set *J* into two components, call them *S* and \bar{S} , such that $s \in S$ and $t \in \bar{S}$. We shall denote by $S:\bar{S}$ the set of edges in the *s-t cut* which lead from a vertex in *S* to a vertex in \bar{S} , and by $\bar{S}:S$ the set of edges which lead from a vertex in \bar{S} to a vertex in *S*. From the discussion in Chapter 5, it is clear that any global state induces an *s-t cut* in *H*. The *s-t cut* corresponding to the global state $\psi \in \Psi$ is such that

$$S = \{s\} \cup PAST(\psi)$$

and

$$\bar{S} = \{t\} \cup FUTURE(\psi).$$

In fact, we can even say that *H* depicts global states in a better way than *PG* does, since now there is a directed edge corresponding to each node's initial and final states. These are the directed edges from *s* to the first event in each node, and from the last event to *t*. In *H*, there is a directed chain from *s* to *t* corresponding to each node, and this chain contains the corresponding chain that exists in *PG*. Although each global state induces an *s-t cut* in *H*, the converse statement is not necessarily true. For there are *s-t cuts* which do not correspond to global states. As an example,

consider the graph H depicted in Figure 6.1. This graph is obtained from Figure 4.1. In Figure 6.1, we show the same vertically oriented lines, labeled (a), (b), and (c), that appear in Figure 4.1. Clearly, all three lines represent $s-t$ cuts in H , though the line labeled (b) does not represent a global state. Nevertheless, it is immediate to see that every $s-t$ cut such that $\bar{S}:S=\emptyset$ induces a global state. In Figure 6.1, this is the case with the cuts labeled (a) and (c).

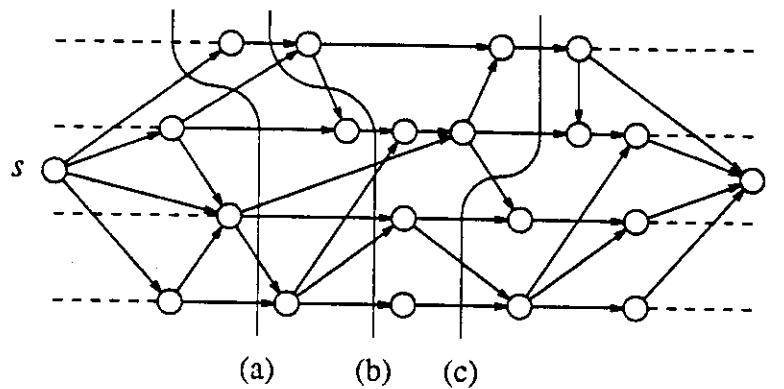


Figure 6.1. Graph H for a Generic Asynchronous Computation

The problem can then be restated as follows. Label each edge $a \in A$ with the 2-tuple $(b(a), c(a))$ such that:

- (i) $b(a) = h(\phi_i)$, if edge a corresponds to state ϕ_i of node i (i.e. it is in the chain corresponding to node i), or $b(a) = 0$, otherwise. Notice that this labeling assigns the value of h at initial states to edges from s . Similarly, it assigns the value of h at final states to edges to t .

$$(ii) \quad c(a) = \infty.$$

Define the *value* of an $s-t$ cut to be

$$\sum_{a \in \bar{S}: \bar{S}} b(a) - \sum_{a \in \bar{S}: S} c(a).$$

It can be seen that, if an $s-t$ cut is such that $\bar{S}:S \neq \emptyset$, then its value is minus infinity. Consequently, an $s-t$ cut induces a global state if and only if its value is nonnegative. Clearly, global state ψ is such that $\sum_{i \in N} h(\psi_i)$ is a maximum over Ψ if and only if the corresponding $s-t$ cut is of maximum value. By the Min-Flow Max-Cut Theorem [Dilw50, Lawl76], an $s-t$ cut of maximum value in H can be found by computing the minimum flow from s to t , using $b(a)$ as a lower bound to the flow on edge $a \in A$, and $c(a)$ as an upper bound, i.e. a 's capacity.

6.4. The Problem under Scheduling by Edge Reversal

6.4.1. Min-Flow Formulation

In all of Section 6.4, we investigate the problem in the case in which nodes are scheduled for operation by Edge Reversal. Recall that the simplified precedence graph PG' can be used in place of PG in this case, and still every distinct global state is represented. Clearly, the construction used previously to obtain graph H from PG can also be used to obtain a similar graph from PG' .

A graph $H'=(J',A')$ is obtained from PG' by enlarging its vertex set, V_2 , with the vertices s and t . Also, a directed edge is created from s to the first type-2 event in each of the nodes, and from the last such event in each of the nodes to t . By assigning the values described in Section 6.3 to the edges of H' , a global state of maximum

value can be found by computing the minimum flow from s to t in H' .

6.4.2. Max-Flow Formulation

As we remarked previously, in the remainder of Section 6.4 we assume that the problem is originally posed as a minimization problem. We have already seen that any $s-t$ cut in H' such that $\bar{S}:S=\emptyset$ induces a global state. While this is true for general asynchronous computations, it can be made more precise because H' is generated by Scheduling by Edge Reversal. Suppose that an $s-t$ cut in H' contains an edge in $\bar{S}:S$ which does not correspond to any message, i.e. an edge in the chain of some node. Let this edge be $(v \rightarrow v')$, and i the node at which v and v' happened. Messages are sent by v to i 's neighbors, and received by v' from those same neighbors. At least one of these messages must correspond to an edge which is also in $\bar{S}:S$, and therefore we get the following tighter characterization of a global state. In H' , any $s-t$ cut such that $\bar{S}:S$ contains no edge between different chains is such that $\bar{S}:S=\emptyset$, and so induces a global state.

After transforming PG' into the graph H' of the previous section, we take the additional step of transforming the graph H' into the graph $\bar{H}'=(J',\bar{A}')$ as follows. The vertex set of \bar{H}' is the same of H' . Its directed edge set is obtained from A' , the edge set of H' , by reversing the orientation of all edges which correspond to messages, i.e. all edges connecting events in different chains. Just as with the graph H' , every global state induces an $s-t$ cut in \bar{H}' , but not conversely. However, from the discussion above, we have that in \bar{H}' every $s-t$ cut such that $S:\bar{S}$ contains no edge between vertices in different chains induces a global state.

The following is a restatement of the problem. Label each edge $a \in \bar{A}'$ with the number $c(a)$ such that:

- (i) $c(a) = h(\phi_i)$, if edge a corresponds to state ϕ_i of node i .
- (ii) $c(a) = \infty$, otherwise.

Redefine the value of an $s-t$ cut to be

$$\sum_{a \in S:\bar{S}} c(a).$$

From this definition, one sees that, if an $s-t$ cut is such that $S:\bar{S}$ contains an edge between vertices in different chains, then its value is infinity. Consequently, every $s-t$ cut with finite value induces a global state, and vice-versa. A global state ψ is such that $\sum_{i \in N} h(\psi_i)$ is a minimum over Ψ if and only if the corresponding $s-t$ cut is of minimum value. By the Max-Flow Min-Cut Theorem [Law176], an $s-t$ cut of minimum value in \bar{H}' can be found by computing the maximum flow from s to t , using $c(a)$ as the capacity of edge $a \in \bar{A}'$.

We show in Figure 6.2 a graph PG' and the corresponding graphs H' and \bar{H}' .

6.4.3. An Example: Distributed Simulated Annealing

We described in Chapters 1 and 2 a distributed realization of the Simulated Annealing method to approximate the minimum of functions of many variables. We had the set $X = \{X_1, \dots, X_n\}$ of n variables, each of which takes values from a common domain D . The function f to be minimized was

$$f(x) = \sum_{Y \subseteq X} g_Y(x),$$

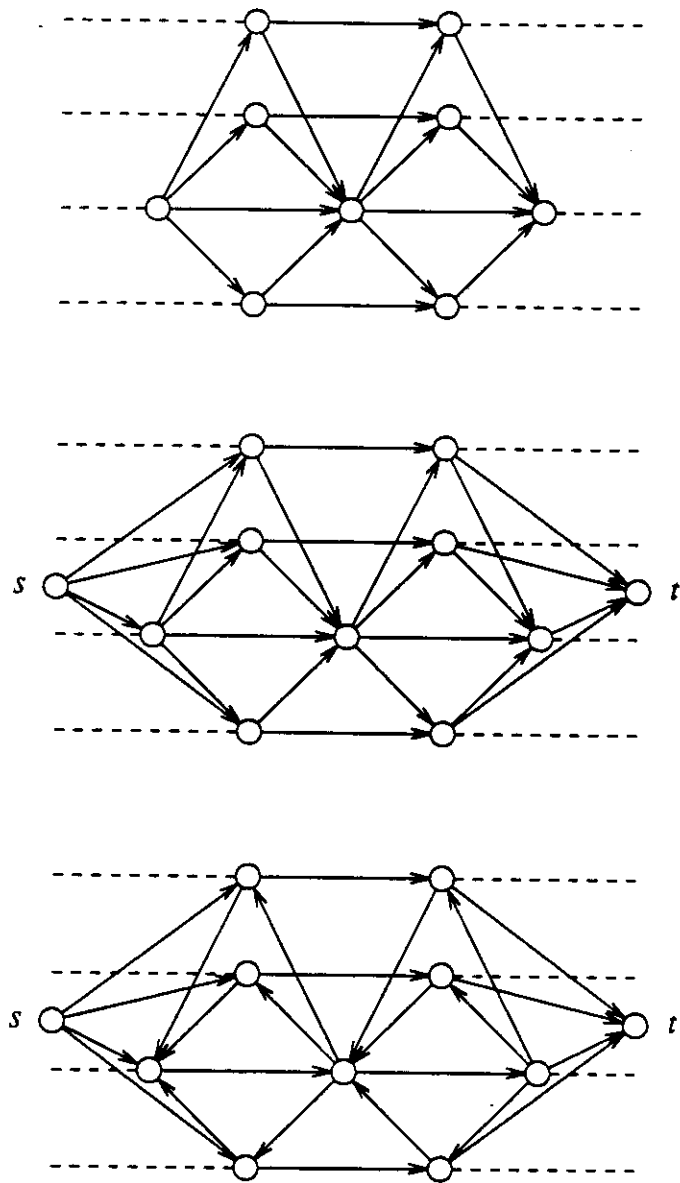


Figure 6.2. Graphs PG' , H' and $\overline{H'}$

for all $x \in D^n$, where $g_Y(x)$ depends only on those coordinates of x respective to variables in Y . The implementation we suggested associates node i with variable X_i , and connects two nodes to each other if and only if there is a subset Y of X of which both variables are members, such that $g_Y(x)$ is nonconstant. Nodes are scheduled for operation by Edge Reversal, and updated their variables probabilistically, using neighboring values only.

Here we want to cast that particular application in the model of Chapter 5. We start by noticing that each Φ_i has values from D . Consequently, $\Phi \subseteq D^n$. The function f can then be also regarded as a real function on Φ . As the simulation proceeds, global states are produced, and one wishes to obtain, at the end, a global state for which the function was found to be minimum during the simulation. We show below how this problem of retrieving a global state of minimum value can be regarded as an instance of our problem. In other words, we specify the function g whose sum over the local states participating in a global state is to be minimized over Ψ .

Using the terminology of Chapter 5, the state of a node only changes when a type-2 event happens, i.e. when messages have been received from all neighbors, and the node operates. A type-2 event v causes a change in f , denoted by $\delta_f(v)$, which is given by

$$\delta_f(v) = f(x') - f(x),$$

where x and x' are points in D^n which differ only in the coordinate corresponding to the node where v happened. In x , this coordinate is the state of that node preceding v , in x' its state succeeding v . Because of the special form of f , we have:

$$\begin{aligned}
\delta_f(v) &= \sum_{Y \subset X} g_Y(x') - \sum_{Y \subset X} g_Y(x) \\
&= \sum_{\substack{Y \subset X \\ X_i \in Y}} g_Y(x') + \sum_{\substack{Y \subset X \\ X_i \notin Y}} g_Y(x') - \sum_{\substack{Y \subset X \\ X_i \in Y}} g_Y(x) - \sum_{\substack{Y \subset X \\ X_i \notin Y}} g_Y(x) \\
&= \sum_{\substack{Y \subset X \\ X_i \in Y}} g_Y(x') - \sum_{\substack{Y \subset X \\ X_i \in Y}} g_Y(x),
\end{aligned}$$

since all functions g_Y such that $X_i \notin Y$ do not depend on the value of X_i . Notice that these are quantities that can be readily determined by node i , since they only involve local information. For the sake of simplicity, we let $\delta_f(v)=0$, whenever v is a type-1 event. At any global state ψ , we have

$$f(\psi) = f(\psi_0) + \sum_{v \in PAST(\psi)} \delta_f(v),$$

where ψ_0 is the initial global state. This can be rewritten as

$$f(\psi) = f(\psi_0) + \sum_{i \in N} \sum_{v \in V_i \cap PAST(\psi)} \delta_f(v),$$

thus leading to the following formulation as an instance of our problem. Let the function g be such that

$$g(\psi_i) = \sum_{v \in V_i \cap PAST(\psi)} \delta_f(v).$$

Clearly, global state ψ^* is such that

$$f(\psi^*) \leq f(\psi),$$

for all $\psi \in \Psi$, if and only if

$$\sum_{i \in N} g(\psi^*_i) \leq \sum_{i \in N} g(\psi_i),$$

for all $\psi \in \Psi$. Notice that $g(\psi_i)$ is the cumulative change caused in f by node i from

the beginning of the simulation up to local state ψ_i .

6.4.4. Centralized Implementation

We discuss in this section a centralized implementation of the Max-Flow solution proposed above. We leave the desirable distributed implementation to be developed in further research (see Chapter 7).

The solution we propose requires that a *leader* be elected in G . Such a leader is a node which all other nodes have agreed to be distinguished [Gall83]. After the computation governed by Scheduling by Edge Reversal has terminated, the precedence graph may be transmitted to the leader, which will then work on the solution.

We saw that a minimum global state in PG' may be obtained by finding a minimum $s-t$ cut in $\overline{H'}$ or, equivalently, by computing a maximum flow from s to t in $\overline{H'}$. Since the graph $\overline{H'}$ has $|V_2|+2$ vertices, a centralized implementation using the most efficient known max-flow algorithms [Malh78, Papa82, Gold86] could require as much as $O(|V_2|^3)$ time to run. The value of $|V_2|$ depends on the number of events in the computation to which $\overline{H'}$ refers. Since this number is in principle unpredictable, the Max-Flow computation may turn out to be quite costly.

In this section, we show that a minimum $s-t$ cut can be obtained by finding $n+|V_2|$ $s-t$ cuts in $\overline{H'}$ at a cost which depends polynomially on n only, and then taking the minimum of these cuts. As a result, the cost of finding a minimum cut is linear in $|V_2|$.

First, some definitions. The *immediate past* of a global state ψ in the set V_2 is the set

$$IPAST(\psi) = \bigcup_{i \in N} pe_2(\psi_i),$$

where $pe_2(\psi_i)$ is the type-2 event, if any, that immediately precedes ψ_i . Similarly, we define the *immediate future* of ψ in V_2 to be

$$IFUTURE(\psi) = \bigcup_{i \in N} se_2(\psi_i),$$

$se_2(\psi_i)$ being the type-2 event, if any, that immediately succeeds ψ_i . Now let v be an event in V_2 . A global state ψ_e is the *earliest* global state with respect to v if and only if $v \in IFUTURE(\psi_e)$ and, furthermore, no other global state ψ is such that $v \in IFUTURE(\psi)$ and $(V_2 \cap PAST(\psi)) \subset (V_2 \cap PAST(\psi_e))$. In the same vein, we say that ψ_l is the *latest* global state with respect to v if and only if $v \in IPAST(\psi_l)$ and, besides, no other global state ψ is such that $v \in IPAST(\psi)$ and $(V_2 \cap FUTURE(\psi)) \subset (V_2 \cap FUTURE(\psi_l))$.

Intuitively, the earliest global state with respect to a type-2 event v is the “leftmost” cut that can be made in PG' such that v appears immediately to its right. Similarly, the latest global state is the “rightmost” cut that can be made in PG' such that v appears immediately to its left.

We now define $SL(\phi_i)$, the *slice* of V_2 with respect to local state ϕ_i . If ϕ_i is the initial state of node i , then

$$SL(\phi_i) = V_2 \cap PAST(\psi_l),$$

where ψ_l is the latest global state with respect to event $se_2(\phi_i)$. If ϕ_i is the final state of node i , then

$$SL(\phi_i) = V_2 \cap FUTURE(\psi_e),$$

where ψ_e is the earliest global state with respect to event $pe_2(\phi_i)$. If neither of these is the case, then

$$SL(\phi_i) = V_2 \cap PAST(\psi_l) \cap FUTURE(\psi_e),$$

where ψ_l is the latest global state with respect to event $se_2(\phi_i)$, and ψ_e is the earliest global state with respect to event $pe_2(\phi_i)$.

Notice that the subgraph of PG' induced by $SL(\phi_i)$ includes all and only the global states in which ϕ_i can participate. Then we can find the minimum global state in PG' by first finding the minimum global state in the $n+|V_2|$ slices of V_2 , and then finding the minimum among them. We show below that a minimum global state in the subgraph induced by one of the slices can be found in a time which does not depend on $|V_2|$.

Theorem 6.1: The set $SL(\phi_i)$ has at most $\frac{1}{2}(n^2+3n)$ events, $\phi_i \in \Phi_i, i \in N$.

Proof: Consider node i . Let $d_G(i, j)$ denote the distance in G between nodes i and j . We show by induction on this distance that, if node j is such that $d_G(i, j)=k$ for some $0 \leq k \leq n-1$, then $SL(\phi_i)$ contains at most $k+2$ events happening at j . The assertion is clearly true for $k=0$, since $pe_2(\phi_i)$ and $se_2(\phi_i)$, if defined, are the only events in $SL(\phi_i)$ happening at node i . Assume it is true for $k-1$, and show it for k . Let j be such that $d_G(i, j)=k$, and let l be one of its neighbors such that $d_G(i, l)=k-1$. By the induction hypothesis, there are at most $k+1$ events in $SL(\phi_i)$ happening at l . Since the computation is governed by Scheduling by Edge Reversal, and considering the definition of $SL(\phi_i)$, no more than $k+2$ events may happen at j , thus proving the assertion. $SL(\phi_i)$ has the greatest number of events when there is exactly one node

in G whose distance from i is k , for each $0 \leq k \leq n-1$. In this case, there are

$$2 + 3 + \cdots + (n+1) = \frac{1}{2}(n^2+3n)$$

events in $SL(\phi_i)$, i.e.

$$|SL(\phi_i)| \leq \frac{1}{2}(n^2+3n).$$

■

The construction outlined previously can be applied to the graph induced by each of the slices $SL(\phi_i)$ to find the minimum global state involving ϕ_i . By Theorem 6.1, this has a time complexity of $O(n^6)$, using one of the Max-Flow algorithms found in the literature [Malh78, Papa82, Gold86]. The overall complexity of finding the optimal global state is then $O(|V_2|n^6)$. Notice that this complexity is achieved despite the fact that, by Theorem 5.1, the number of distinct global states is linear in $|V_2|$ and exponential in n .

In these constructions, the capacity of edges outgoing from s and incoming to t are set to infinity, since we do not wish a cut involving these edges to be considered. Exceptions to this rule are cases in which an edge from s is connected to the first event in a node, or an edge to t is connected to the last event in a node. Such edges are assigned capacities reflecting that node's initial or final state, respectively.

Though we have dwelt on the Max-Flow formulation in order to present this more efficient implementation, it should be noticed that we have essentially used a property of the global states generated by Edge Reversal. As a consequence, similar techniques apply to the Min-Flow formulation as well.

CHAPTER 7

DIRECTIONS FOR FURTHER RESEARCH

In this chapter we point out some directions in which the research presented in this dissertation can be extended. First, we discuss extensions pertinent to Scheduling by Edge Reversal, including applications and concurrency measures. Secondly, we consider extensions to the Max-Flow solution to the problem of identifying optimal global states. The extensions we consider have to do with a fully distributed, on-the-fly implementation of that solution when the computation is governed by Scheduling by Edge Reversal. Lastly, we comment on possible extensions to our distributed implementation of the Simulated Annealing method.

We describe a few lines of research to be pursued in the context of analyzing the Scheduling by Edge Reversal method itself. To begin with, one might look for different areas and problems to which the scheduling method would be applicable. For example, the model described in [Yemi83] for resource sharing systems requires that independent sets be scheduled for operation, and so our method might be useful.

Since the decision problem corresponding to determining $\gamma^*(G)$ is *NP*-complete, the search for heuristics to approximate its value might turn out to be of interest. With respect to the particular applications that we have considered, this would provide a means of coming up with an initial orientation yielding a value of concurrency approximately close to the optimum.

As a second direction in which this work can be extended, we argue towards the possibility of implementing distributedly the Max-Flow solution proposed in Section 6.4, when the precedence graph is generated by Scheduling by Edge Reversal. The idea is to have each node $i \in N$ store information about the events that involve it, i.e. the members of V_i which are type-2 events, and calculate, distributedly and on-the-fly, the maximum flow from s to t in the graph $\overline{H'}$. A newly proposed Max-Flow algorithm [Gold86] is quite convenient in this sense.

Unlike other Max-Flow algorithms, this algorithm first finds the cut of minimum value, in a first stage, and then computes the maximum flow, in a second stage. This makes it convenient to us, since it is the minimum cut that we are really looking for. Moreover, and more importantly, this algorithm operates fully distributedly by relying on local information only. It operates on a residual graph, which is a subgraph of the original graph containing nonsaturated edges only. Each vertex keeps an estimate of its distance to t in the residual graph, and pushes excess flow in the direction of t , based on this estimate. The only requirement is that the estimate be a lower bound on the actual distance to t in the residual graph.

We show in the following simple lemma that these distance estimates can be associated with the vertices of $\overline{H'}$ as soon as it starts to be built, i.e. during the computation itself.

Lemma 7.1: In $\overline{H'}$, a shortest undirected path from any event to t exists along the chain corresponding to the node at which that event happens.

Proof: Notice that any alternative to the path asserted by the lemma would be a path that goes to the chain of another node, and then heads to t . If these two

chains correspond to nodes which are k hops away in G , for some $1 \leq k \leq n$, then we see that k hops are needed to get from one chain to the other. If l is the length of the path asserted in the statement, then Corollary 3.3 implies that the alternative path cannot be shorter than $k + (l - k)$, thus showing the equivalence of the two. ■

The following is an alternative initial labeling of vertices which can be used by the algorithm. Vertex s gets label 0, and the k th event at each node gets label $-k$, $k > 0$.

Because possible global states cannot be too spread out in the precedence graph (see Theorem 6.1), not all of a node's history has to be kept during the Max-Flow computation, which may account for a good message complexity. We leave the details of this completely distributed, on-the-fly implementation for further research.

Finally, we discuss possible extensions to the distributed implementation of the Simulated Annealing method. First, we would like to investigate other characteristics of problems whose solution can be approximated through our distributed implementation of Simulated Annealing, and perform tests to find out the effectiveness of the proposal.

Another direction in which investigation can be furthered is the following. Recall that our distributed implementation performs a stochastic search on the entirety of D^n . That the solution must lie inside the feasible set FS is something to be incorporated in the objective function through the use of penalties. The reason for this is that the convergence proof in [Gema84], in which we based our proposal, requires the set X of variables to behave as a Markov Random Field (MRF), as

explained in Chapter 2. For such, it is necessary that all points in D^n have a nonzero probability of being found during the simulation, be they feasible or not. The question that poses itself is then how to perform the search inside FS only. It has been shown in [Mous74] that the probabilities involved would still leave X with the necessary characteristics of an MRF. Nevertheless, convergence is yet to be proven. Even if such is the case, however, we tend to think that the original unconstrained approach is more efficient, since it allows infeasible intermediate configurations, possibly making the paths among feasible configurations shorter, and convergence faster.

REFERENCES

- [Abde76] Abdel-Wahab, H. M., "Scheduling with Applications to Register Allocation and Deadlock Problems," Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, Canada (1976).
- [Abde78] Abdel-Wahab, H. M. and T. Kameda, "Scheduling to Minimize Maximum Cumulative Cost Subject to Series-Parallel Precedence Constraints," *Operations Research* 26(1), pp.141-158 (January/February 1978).
- [Arag84] Aragon, C. R., D. S. Johnson, L. A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: an Experimental Evaluation," *Workshop on Statistical Physics in Engineering and Biology* (April 1984).
- [Awer85] Awerbuch, B., "Complexity of Network Synchronization," *Journal of the ACM* 32(4), pp.804-823 (October 1985).
- [Berg76] Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, The Netherlands (1976).
- [Berg] Berger, T. and F. Bonomi, "Parallel Updating of Certain Markov Random Fields," unpublished manuscript, Cornell University, Ithaca, NY.
- [Besa74] Besag, J., "Spatial Interaction and the Statistical Analysis of Lattice Systems," *Journal of the Royal Statistical Society, Series B* 36(2), pp.192-236 (1974).
- [Boll79] Bollobás, B. and A. Thomason, "Set Colourings of Graphs," *Discrete Mathematics* 25(1), pp.21-26 (January 1979).
- [Brac84] Bracha, G. and S. Toueg, "A Distributed Algorithm for Generalized Deadlock Detection," pp. 285-301 in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, BC, Canada (August 27-28, 1984).
- [Brig82] Brigham, R. C. and R. D. Dutton, "Generalized k -tuple Colorings of Cycles and Other Graphs," *Journal of Combinatorial Theory, Series B* 32(1), pp.90-94 (February 1982).

- [Camp86] Campbell, M., "Data Flow Graphs as a Model of Parallel Complexity," Ph.D. Dissertation, in preparation, Computer Science Department, University of California, Los Angeles, CA (1986).
- [Chan84] Chandy, K. M. and J. Misra, "The Drinking Philosophers Problem," *ACM Transactions on Programming Languages and Systems* 6(4), pp.632-646 (October 1984).
- [Chan85] Chandy, K. M. and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions on Computer Systems* 3(1), pp.63-75 (February 1985).
- [Chan80] Chang, E., " n -Philosophers: an Exercise in Distributed Control," *Computer Networks* 4, pp.71-76 (1980).
- [Clar76] Clarke, F. H. and R. E. Jamison, "Multicolorings, Measures and Games on Graphs," *Discrete Mathematics* 14(3), pp.241-245 (March 1976).
- [Cook71] Cook, S. A., "The Complexity of Theorem-Proving Procedures," pp. 151-158 in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, Shaker Heights, OH (May 3-5, 1971).
- [Demi79] Deming, R. W., "Acyclic Orientations of a Graph and Chromatic and Independence Numbers," *Journal of Combinatorial Theory, Series B* 26(1), pp.101-110 (February 1979).
- [Dijk71] Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," *Acta Informatica* 1, pp.115-138 (1971).
- [Dilw50] Dilworth, R. P., "A Decomposition Theorem for Partially Ordered Sets," *Annals of Mathematics* 51, pp.161-166 (1950).
- [Ende77] Enderton, H. B., *Elements of Set Theory*, Academic Press, New York, NY (1977).
- [Felt85] Felten, E., S. Karlin, and S. W. Otto, "The Traveling Salesman Problem on a Hypercubic, MIMD Computer," pp. 6-10 in *Proceedings of the International Conference on Parallel Processing*, Chicago, IL (August 20-23, 1985).
- [Fior77] Fiorini, S. and R. J. Wilson, *Edge-Colourings of Graphs*, Pitman, London, England (1977).
- [Gafn81] Gafni, E. M. and D. P. Bertsekas, "Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology," *IEEE Transactions on Communications COM-29*(1), pp.11-18 (January 1981).

- [Gall83] Gallager, R. G., P. A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Transactions on Programming Languages and Systems* 5, pp.66-77 (January 1983).
- [Gare79] Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA (1979).
- [Gell75] Geller, D. and S. Stahl, "The Chromatic Number and other Functions of the Lexicographic Product," *Journal of Combinatorial Theory, Series B* 19(1), pp.87-95 (August 1975).
- [Gema84] Geman, S. and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6(6), pp.721-741 (November 1984).
- [Gida85] Gidas, B., "Non-Stationary Markov Chains and Convergence of the Annealing Algorithm," *Journal of Statistical Physics* 39, pp.73-131 (1985).
- [Gold86] Goldberg, A. V. and R. E. Tarjan, "A New Approach to the Maximum Flow Problem," pp. 136-146 in *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, Berkeley, CA (May 28-30, 1986).
- [Gree86] Greene, J. W. and K. J. Supowit, "Simulated Annealing without Rejected Moves," *IEEE Transactions on Computer-Aided Design* CAD-5(1), pp.221-228 (January 86).
- [Gree74] Greenwell, D. and L. Lovász, "Applications of Product Colouring," *Acta Mathematica Academiae Scientiarum Hungaricae* 25(3-4), pp.335-340 (1974).
- [Grif76] Griffeath, D., "Introduction to Random Fields," pp. 425-458 in *Denumerable Markov Chains*, ed. J. G. Kennedy, J. L. Snell, and A. W. Knap, Springer-Verlag, New York, NY (1976).
- [Gröt81] Grötschel, M., L. Lovász, and A. Schrijver, "The Ellipsoid Method and Its Consequences in Combinatorial Optimization," *Combinatorica* 1(2), pp.169-197 (1981).
- [Haje84] Hajek, B., "Link Schedules, Flows, and the Multichromatic Index of Graphs," pp. 498-502 in *Proceedings of the 1984 Conference on Information Sciences and Systems*, Princeton, NJ (March 14-16, 1984).

- [Haje85] Hajek, B., "Cooling Schedules for Optimal Annealing," manuscript, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL (January 1985).
- [Hema84] Hemachandra, L. A. and V. K. Wei, "Using Simulated Annealing to Calculate Combinatorial Constants," pp. 545-552 in *Proceedings of the Twenty-Second Annual Allerton Conference on Communications, Control, and Computing*, Monticello, IL (October 3-5, 1984).
- [Hilt73] Hilton, A. J. W., R. Rado, and S. H. Scott, "A (<5)-colour Theorem for Planar Graphs," *The Bulletin of the London Mathematical Society* 5(15), pp.302-306 (November 1973).
- [Isha81] Isham, V., "An Introduction to Spatial Point Processes and Markov Random Fields," *International Statistical Review* 49, pp.21-43 (1981).
- [Karp72] Karp, R. M., "Reducibility among Combinatorial Problems," pp. 85-103 in *Complexity of Computer Computations*, ed. R. E. Miller and J. W. Thatcher, Plenum Press, New York, NY (1972).
- [Kind80] Kinderman, R. and J. L. Snell, *Markov Random Fields and Their Applications*, American Mathematical Society, Providence, RI (1980).
- [Kirk83] Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science* 220(4598), pp.671-680 (May 13, 1983).
- [Lamp78] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM* 21(7), pp.558-565 (July 1978).
- [Lawl76] Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, NY (1976).
- [Luen73] Luenberger, D. G., *Introduction to Linear and Nonlinear Programming*, Addison-Wesley Pub. Co., Reading, MA (1973).
- [Lync80] Lynch, N. A., "Fast Allocation of Nearby Resources in a Distributed System," pp. 70-81 in *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, Los Angeles, CA (April 28-30, 1980).
- [Lync81] Lynch, N. A., "Upper Bounds for Static Resource Allocation in a Distributed System," *Journal of Computer and System Sciences* 23(2), pp.254-278 (October 1981).

- [Malh78] Malhotra, V. M., M. P. Kumar, and S. N. Maheshwari, "An $O(|V|^3)$ Algorithm for Finding Maximum Flows in Networks," *Information Processing Letters* 7(6), pp.277-278 (October 1978).
- [Metr53] Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *Journal of Chemical Physics* 21, pp.1087-1091 (1953).
- [Mitr] Mitra, D., F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and Finite-Time Behavior of Simulated Annealing," *Advances in Applied Probability*. To appear.
- [Mous74] Moussouris, J., "Gibbs and Markov Random Systems with Constraints," *Journal of Statistical Physics* 10(1), pp.11-33 (January 1974).
- [Otte84] Otten, R. H. J. M. and L. P. P. van Ginneken, "Floorplan Design Using Simulated Annealing," pp. 96-98 in *Proceedings of the IEEE International Conference on Computer-Aided Design*, Santa Clara, CA (November 12-15, 1984).
- [Papa82] Papadimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1982).
- [Rabi81] Rabin, M. and D. Lehmann, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem," pp. 133-138 in *Proceedings of the Eighth Annual ACM Symposium on Principles of Programming Languages*, Williamsburg, VA (January 26-28, 1981).
- [Rome84a] Romeo, F. and A. Sangiovanni-Vincentelli, "Probabilistic Hill Climbing Algorithms: Properties and Applications," Memorandum No. UCB/ERL M84/34, University of California at Berkeley, Berkeley, CA (March 13, 1984).
- [Rome84b] Romeo, F., A. Sangiovanni-Vincentelli, and C. Sechen, "Research on Simulated Annealing at Berkeley," pp. 652-657 in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, NY (October 8-11, 1984).
- [Rota64] Rota, G.-C., "On the Foundations of Combinatorial Theory, I. Theory of Möbius Functions," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 2(4), pp.340-368 (March 13, 1964).

- [Seit85] Seitz, C. L., "The Cosmic Cube," *Communications of the ACM* 28(1), pp.22-33 (January 1985).
- [Seth75] Sethi, R., "Complete Register Allocation Problems," *SIAM Journal on Computing* 4(3), pp.226-248 (September 1975).
- [Spit71] Spitzer, F., "Markov Random Fields and Gibbs Ensembles," *American Mathematical Monthly* 78, pp.142-154 (1971).
- [Stah76] Stahl, S., " n -Tuple Colorings and Associated Graphs," *Journal of Combinatorial Theory, Series B* 20(2), pp.185-203 (April 1976).
- [Stah79] Stahl, S., "Fractional Edge Colorings," *Cahiers du Centre d'Etudes de Recherche Opérationnelle* 21(2), pp.127-131 (1979).
- [Stan73] Stanley, R. P., "Acyclic Orientations of Graphs," *Discrete Mathematics* 5(2), pp.171-178 (June 1973).
- [Vecc83] Vecchi, M. P. and S. Kirkpatrick, "Global Wiring by Simulated Annealing," *IEEE Transactions on Computer-Aided Design CAD-2*(4), pp.215-222 (October 1983).
- [Whit84] White, S. R., "Concepts of Scale in Simulated Annealing," pp. 646-651 in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, NY (October 8-11, 1984).
- [Yemi83] Yemini, Y., "A Statistical Mechanics of Distributed Resource Sharing Mechanisms," pp. 531-539 in *Proceedings of the Second IEEE INFOCOM*, San Diego, CA (April 18-21, 1983).

