

EXPERIMENTS IN DISTRIBUTED PROBLEM SOLVING
WITH
ITERATIVE REFINEMENT
TR # 82-25

A Dissertation Presented
By
RICHARD SAMUEL BROOKS

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 1983

Computer and Information Science

© Richard Samuel Brooks 1982
All Rights Reserved

This work was supported in part by:

The National Science Foundation
Grant Number MCS78-04212

EXPERIMENTS IN DISTRIBUTED PROBLEM SOLVING

WITH


ITERATIVE REFINEMENT

A Dissertation Presented

By

RICHARD SAMUEL BROOKS

Approved as to style and content by:



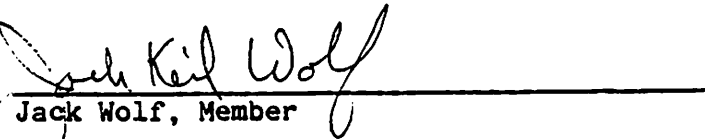
Victor R. Lesser, Chairperson of Committee



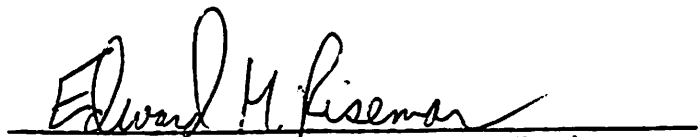
Michael A. Arbib, Member



William Kilmer, Member



Jack Wolf, Member



Edward M. Riseman, Department Head
Computer and Information Science

ACKNOWLEDGEMENTS

I would like to thank Victor Lesser for his invaluable guidance and helpful comments on many drafts of this thesis and Michael Arbib, Bill Kilmer, and Jack Wolf for their suggestions. I would also like to thank Yechiam Yemini for many fruitful discussions concerning my extension of his balance principle, Andy Barto for listening to my ideas and providing excellent feedback, and Dan Corkill for help with the formatting of this thesis.

ABSTRACT

Experiments in Distributed Problem Solving
with
Iterative Refinement

February 1982

Richard S. Brooks

B.S., Massachusetts Institute of Technology

M.S., Ph.D., University of Massachusetts

Directed by: Professor Victor Lesser

A general method for distributed problem solving which tolerates incomplete and inconsistent local databases is developed, based on a centralized iterative refinement technique. It is shown that uncertainty introduced as a result of distributing the iterative refinement technique can be resolved through additional iteration if an appropriate control scheme is employed to limit simultaneous refinements.

To support this conclusion, a number of variations of distributed iterative refinement network traffic light control algorithm with modest communication requirements are shown to find comparable solutions in less time than an existing, centralized network traffic light control algorithm. Processors are assumed to be located at intersections equipped with signals, and may only communicate with processors at adjacent intersections. Simulation results also indicate that the algorithms tolerate the use of approximations that limit communication and the uncertainty that is introduced if communication is noisy.

As a by-product of this research, a general balance principle and adaptive balancing algorithm for Pareto-optimal multi-access control of distributed resources is also developed and tested. The balance principle and adaptive balancing algorithm represents an extension of work by Kleinrock and Yemini on packet-radio broadcast communication.

TABLE OF CONTENTS

LIST OF TABLES	x
LIST OF FIGURES	xi

Chapter	
I.	INTRODUCTION 1
1.1	Objectives 5
1.2	Results 6
1.3	Summary of Remaining Chapters 7
II.	BACKGROUND 11
2.1	Distributed Problem Solving 11
2.2	Iterative Refinement 13
2.2.1	Classical iterative refinement 15
2.2.2	Non-classical iterative refinement 17
III.	DISTRIBUTED PROBLEM SOLVING WITH ITERATIVE REFINEMENT . . . 19
3.1	A Centralized Iterative Refinement Algorithm 19
3.2	Distributing the Iterative Refinement Technique 21
3.2.1	The basic algorithm 24
3.2.2	The case of localized interaction 29
3.2.3	The case of non-local interaction 30
IV.	THE SIMULTANEOUS-UPDATE PROBLEM 33
4.1	Access Control 34
4.1.1	Access control for update control 36
4.1.2	Some fundamental access control schemes 37
4.1.3	Discussion 40
4.2	Pareto-Optimal Access Control for Two Schemes 41
4.2.1	A balance principle for optimal access control 42
4.2.2	An adaptive balancing algorithm 59
4.3	Eight Update Control Schemes 67

V.	EXPERIMENTS WITH LOCALIZED INTERACTION	74
5.1	A Simple Numbering Problem	75
5.2	Initial Numbering Problem Experiments	79
5.2.1	Overall behavior	80
5.2.2	Speed of convergence and quality of solutions	83
5.2.3	Update control statistics	85
5.2.4	Communication requirements	91
5.3	Sensitivity Experiments	94
5.3.1	Sensitivity to network size	94
5.3.2	Sensitivity to network topology	97
5.4	Discussion	100
VI.	AN APPLICATION: DISTRIBUTED NETWORK TRAFFIC LIGHT CONTROL	102
6.1	Introduction	102
6.2	A Traffic Light Control Problem	105
6.3	The SIGOP II Algorithm	111
6.4	Discussion	114
VII.	THE PROBLEM OF NON-LOCAL INTERACTION	118
7.1	The Introduction of State Variables	121
7.2	The Amended Algorithm	124
7.3	The Local-View Problem	129
7.3.1	A local-view approximation	130
7.3.2	A worst-case assumption	133
7.3.3	Cooperative gathering of non-local information	137
VIII.	EXPERIMENTS WITH NON-LOCALIZED INTERACTION	142
8.1	Arterial Traffic Light Control Experiments	143
8.1.1	Performance with local-view approximation	143
8.1.2	Performance with decoupling technique	149
8.1.3	Performance with information gathering technique	149
8.1.4	Discussion	150
8.2	Network Traffic Light Control Experiments	152
8.2.1	Performance on small network	152
8.2.2	Performance on large network	157
8.2.3	Performance with communication errors	163
8.2.4	Performance with partial state updating	167
8.2.5	Discussion	168
IX.	CONCLUSIONS	171
9.1	Summary of Experimental Results	173
9.2	Access Control	174
9.3	Open Questions	176

SELECTED BIBLIOGRAPHY	179
Appendix A. TESTS OF ADAPTIVE BALANCING ALGORITHM	187
I.1. Theoretical Performance	187
I.2. Simulation Results	191
Appendix B. TWO-PLATOON TRAFFIC FLOW SYSTEM MODEL	209

LIST OF TABLES

1. Statistics on Update Control for Ring Numbering Problem	85
2. Statistics on Update Control for ATLC Problem	146
3. Statistics on Errors in Departure Times for ATLC Problem . . .	148

LIST OF FIGURES

1. Assignment of Processing Elements to Controls	22
2. Basic Algorithm for a Processing Element	26
3. Utilization Operator	49
4. An Example of the Numbering Problem	77
5. Global Disutility vs. Iteration for RING of 12 PEs	82
6. Speed of Convergence and Quality of Solutions	84
7. Histograms of Utilizations with some Update Control Schemes . .	89
8. Communication Requirements of Update Control Schemes	91
9. Sensitivity to Network Size	96
10. Sensitivity to Network Topology	98
11. Network Topologies for Numbering Problem Experiments	99
12. A Time/Space Diagram	107
13. The Arterial Traffic Light Control Problem	110
14. First Sample Run of SIGOP II	115
15. Second Sample Run of SIGOP II	116
16. An Example of Non-Local Interaction	119
17. Traffic System State Variables	122
18. The Amended Distributed Iterative Refinement Algorithm	125
19. The Danger of Employing a Local View	134
20. Performance on ATLC Problem with Local View Approximation . .	145
21. Performance with Local View vs. Global View	151
22. Small Network Used in First NTLC Experiments	153
23. Performance on ATLC vs. NTLC Problem with 12 PEs	156
24. Large Network Used in Final NTLC Experiments	159
25. Performance with Local View on Large NTLC Problem	162
26. Sensitivity to Communication Errors	164
27. Individual Runs with 0, 1 and 5 Percent Communication Errors .	166
28. Performance with Partial vs. Full State Updating	169
29. Expected Utilization vs. Demand with N PEs and no Penalty . .	188
30. Expected Utilization vs. Demand with 2 PEs and Penalty α . . .	189
31. Probabilistic Scheme at Equilibrium, Various Step-sizes . . .	194
32. Probabilistic Scheme at Equilibrium, Various Window-sizes . .	196
33. Transient Behavior of Probabilistic Scheme, Large Step-size .	199
34. Transient Behavior of Probabilistic Scheme, Small Step-size .	201
35. Probabilistic Scheme near Equilibrium, Various Penalties . . .	203
36. Transient Behavior of Urn Scheme, Large Step-size	206
37. Transient Behavior of Urn Scheme, Small Step-size	208
38. Two-platoon Interaction with a Signal - Case 1	211
39. Two-platoon Interaction with a Signal - Cases 2 and 3	212
40. Two-platoon Interaction with a Signal - Cases 4 and 5	213
41. Two-platoon Interaction with a Signal - Cases 6 and 7	214
42. Two-platoon Interaction with a Signal - Case 8	215

CHAPTER I

INTRODUCTION

The cost of processing elements and communication networks is at a level where distributed problem solving is now practical. This is mainly due to recent advances in microprocessor and network technologies [LELA81]. The potential advantages of a distributed problem solving approach over a centralized, uniprocessor approach include:

- o enhanced real-time response;
- o lower processing costs;
- o lower communication costs;
- o increased reliability and flexibility;
- o the ability to handle increased complexity.

These potential advantages have yet to be exploited in a wide range of applications because of a lack of appropriate theories on structuring distributed problem solving systems. Only in the areas of distributed databases [PEEB78, KIMB81, HOLL81] networking [TOBA80, KLEI79, THUR81a, THUR81b] and process control [DIMM76, SCHO76, MERR81] have some of the promises of distributed processing been realized.

There are many applications that could benefit from new theories on structuring distributed problem solving systems. This is especially so for applications which exhibit a natural spatial distribution of sensors and/or effectors such as network traffic light control, air traffic control, and sensor networks. Such applications often involve sensing and/or control of many objects over a very large spatial area, making a

centralized, uniprocessor approach unattractive and often impractical.

The advantages of distributed problem solving can be realized by exploiting the natural spatial decomposition of a problem as well as by exploiting processing resources (parallelism), technological advances, and good design. Enhanced real-time response, for example, can be achieved by placing processing elements near sensors and effectors as well as by exploiting parallelism. The use of cheaper, less complex processing elements that can be mass produced, and the use of load-sharing can contribute to lower processing costs. Abstracting data for transmission, and placing processing elements near the data can lower communication costs. And, increased reliability and flexibility can be achieved through redundancy in communication paths and processing elements, and through modularity of design. Often failures can be localized, yielding graceful degradation. Finally, the ability to handle increased complexity can be achieved by decomposing the problem into subproblems, each of lower complexity than the overall problem.

On the other hand, there are potential problems with a distributed problem solving approach. Many of these problems are not normally encountered in centralized problem solving because centralized problem solving techniques generally rely on maintaining correctness and consistency in all aspects of the problem solving process.¹ While easily maintained in a centralized problem solving system, correctness

1. There are important exceptions, however, to this where there is a need to introduce approximations in order to solve (with some degree of accuracy) a very large problem.

and consistency is often difficult and expensive to maintain in a distributed problem solving system because some of the information required by a processing element to solve a subproblem often resides at some other processing element.

When information required to solve a subproblem does not reside at a processing element responsible for solving the subproblem, the information must be obtained from another processing element in order to correctly solve the subproblem. However, there are costs associated with communicating this information: the costs of the communication network and the processing resources wasted while processors wait to receive information.² To eliminate the communication of some information or to substitute for information not yet received, approximations may be utilized in the local problem solving processes. For example, worst-case information might be assumed in place of communicating some actual information. This limits the information needed by and transferred among processing elements, but at the expense of possibly introducing error, inconsistency, and incompleteness into the distributed problem solving system. Unreliable or noisy communication channels can be another source of error and inconsistency.

2. High speed point to point communication between arbitrary processors in a large network is generally impractical, so less costly but slower communication networks must often be employed in large distributed systems.

Regardless of the source, error, inconsistencies, and incompleteness leads to uncertainty in a problem solving system about the correctness and completeness of the information received from other processing elements, and thus the results of its own processing. The main focus of this thesis is on how to deal with uncertainty in a distributed problem solving system. Communication errors and approximations employed to limit communication and permit more effective utilization of processing resources are two important sources of uncertainty in a distributed problem solving system. Another, more subtle source of uncertainty may be viewed as a synchronization problem. A processing element working in parallel with other processing elements faces uncertainty as to the validity of the solution to its subproblem if the solution is based on information obtained from other processing elements which may be changed while the subproblem is being solved.³

The idea explored in this thesis is that a centralized problem solving algorithm which is capable of resolving uncertainty in a centralized setting may be capable of handling additional uncertainty introduced as a result of adapting the algorithm for use in a distributed problem solving system. Furthermore, the ability to deal effectively with uncertainty should lead to many of the advantages of

3. If information that must be acquired from other processing elements to solve a subproblem is obtained early and is invalidated before it is used, or if this information is invalidated after it has been used to solve a subproblem, the solution to the subproblem may also be invalidated. This, however, is a problem not unique to distributed problem solving (in planning this is called subgoal interaction [NILS80]) and is usually addressed by finding a sequence for solving the subproblems that avoids these problems.

distributed problem solving, i.e., less inter-process communication, more effective processor utilization (i.e., parallelism), and more robustness in presence of hardware error.

The centralized problem solving technique selected for this research is iterative refinement. An iterative refinement algorithm starts with an approximate solution and repeatedly refines (improves upon) the approximation. Because of the iterative nature of the technique and the fundamental use of approximate solutions, uncertainty is not only tolerated, but is resolved as an integral part of the problem solving method.

1.1 Objectives

The primary objective of this research is to demonstrate that it is not necessary to maintain full accuracy, consistency, and completeness in distributed problem solving with iterative refinement because of the iterative refinement algorithm's ability to resolve uncertainty. This uncertainty may arise from working in parallel on interacting subproblems⁴, using approximations to limit communication, and sustaining occasional communication errors. To this end, a number of distributed iterative refinement algorithms were developed and tested on

4. Two subproblems "interact" if part of the solution to one subproblem is needed to solve the other subproblem. This relation is often symmetric (i.e., the solution of each subproblem may require a part of the solution to the other subproblem).

their ability to solve a number of problems, including a network traffic light control problem.

In addition to the usual questions of how much parallelism can be effectively utilized (what degree of speed-up results) and how network size and topology affect performance, the following questions are also addressed:

- o To what extent can interaction among subproblems be tolerated and at what expense? Can interaction be limited?
- o Can processing elements work with local views of the effects of their actions or decisions, or must a global view be employed?
- o How accurate must information needed by a processing element be about the rest of the system's actions or decisions?

Many of these questions are covered by asking how much uncertainty in control, algorithm, and data can be tolerated in a distributed problem solving system.

1.2 Results

It is shown that additional uncertainty introduced as a result of distributing the centralized iterative refinement algorithm can be resolved through the basic iterative nature of the algorithm if an appropriate control scheme is employed to limit but not necessarily eliminate the simultaneous solution of interacting subproblems, and hence the uncertainty that results. To support this conclusion, a number of versions of distributed iterative refinement network traffic

light control algorithm with modest communication requirements are shown to find comparable solutions in less time than an existing centralized network traffic light control algorithm. Simulation results also indicate that the distributed iterative refinement algorithms tolerate the use of approximations that limit communication and the uncertainty that is introduced if communication is noisy.

As a by-product of this research, a general balance principle and adaptive balancing algorithm for Pareto-optimal multi-access control of distributed resources is also developed. The balance principle and adaptive balancing algorithm are used in some versions of the distributed iterative refinement algorithm to control simultaneous refinements. The new balance principle and adaptive balancing algorithm is an extension of work by Kleinrock and Yemini [KLEI78, YEMI78, YEMI79] concerning a balance principle and adaptive algorithms for multi-access control of a shared communication channel in a packet radio network.

1.3 Summary of Remaining Chapters

The remainder of this thesis begins with a background chapter on distributed problem solving and iterative refinement. In Section 2.1 the relationship of this work to others' work in distributed problem solving and distributed processing is discussed. The iterative refinement technique is discussed in Section 2.2. A number of variations of the basic iterative refinement paradigm are identified in this section based on the way in which refinements are obtained.

The adaptation of an iterative refinement technique for distributed problem solving is discussed in Chapter 3. Two main problems are encountered in distributing the iterative refinement technique: the simultaneous-update problem and the non-local interaction problem. The simultaneous-update problem is the synchronization problem described earlier which arises because simultaneous refinements may interact to produce an unexpected result. The non-local interaction problem is the problem of coping with interaction between processing elements which are not neighbors (spatially adjacent).

The simultaneous-update problem is discussed at length in Chapter 4. The simultaneous-update problem arises because processing elements are allowed to make refinements in parallel. In order to alleviate the simultaneous-update problem, decentralized update control schemes are considered to control refinements. Because this problem of update control can be viewed as a problem of decentralized access control of distributed resources, this more general problem is addressed. For this reason, much of Chapter 4 concerns this access control problem and may be applicable to other problems involving access control of shared resources.

Experiments with a distributed iterative refinement algorithm for solving a simple numbering problem are discussed in Chapter 5. Interaction is localized with the numbering problem, so a local view of the effects of a refinement is equivalent to a global view, and thus proposed refinements are always appropriate from a global context. For this reason, the numbering problem is ideal for testing the ability of

update control schemes to alleviate the simultaneous-update problem without the added complication of the local-view problem.

In Chapter 6, a network traffic light control problem is described. This application exhibits clear, strong non-local interaction and possesses a well defined but complex objective function. An existing centralized algorithm based on iterative refinement which solves this problem, SIGOP II [LIEB76], is also described in this chapter; this algorithm is used for comparisons in Chapter 8.

The problem of coping with non-local interaction is addressed in Chapter 7. The non-local interaction problem arises because communication between processing elements that are not spatially adjacent is costly, and possibly errorful or delayed. For this reason, it is desirable to introduce mechanisms that limit the information processing elements must acquire from non-neighboring processing elements, and to have processing elements employ only a limited, local view of the overall problem. As a consequence, though, refinements may no longer be appropriate from a global perspective.

Because a local view may not be best, a number of ideas are explored in Chapter 7. First, the local-view approximation is developed under the assumption that effects beyond neighboring processing elements may be simply ignored; it is hoped that the robustness of the iterative refinement technique will permit this approximation. A worst-case assumption is also developed for use with the local-view approximation if decoupling of non-neighboring processing elements is desired. Finally, an information gathering technique is developed for acquiring

larger views.

Chapter 8 presents and discusses results of experiments with arterial and network traffic light control. These experiments focus on the ability of the distributed iterative refinement technique to handle the combination of the non-local interaction and simultaneous-update problems. Communication errors are also introduced in some of the experiments.

Conclusions and open questions are presented in Chapter 9.

C H A P T E R II

BACKGROUND

This chapter consists of background sections on distributed problem solving and iterative refinement. The relationship of the work presented in this thesis to others' work in distributed problem solving is discussed in Section 2.1. In Section 2.2 the basic (centralized) iterative refinement paradigm is described and a number of variations are identified and discussed.

2.1 Distributed Problem Solving

The research presented in this thesis concerns low-level distributed problem solving. Distributed problem solving is defined as a cooperative computation involving a network of processing elements that share a common goal [DAVI80]. Low-level distributed problem solving involves a close coupling of dedicated processing elements; processors are often located near sensors and/or effectors, and are arranged with a fixed organization.

In contrast to this, high-level distributed problem solving approaches which have recently appeared in the artificial intelligence literature exhibit more loosely coupled (nearly autonomous) interaction among processing elements and permit more variability in organization. Some of these high-level distributed problem solving approaches include ACTORS [HEWI77], BEINGS [LENA75], a distributed Hearsay approach

[LESS80, LESS81, LESS82], and a contract-net approach [SMIT80, SMIT81].

The difference between distributed problem solving and distributed processing algorithms (for networking, distributed data base systems, etc.) is the nature of the problem to be solved. With distributed processing, there are many localized tasks that are to be performed by a network of processing elements that share resources; with distributed problem solving, on the other hand, there is a single, decomposed task, a larger scope of interaction, and often a need for some global coordination and/or calculations.

Unlike most control theoretic approaches to problem solving such as aggregation, singular and non-singular perturbation, decentralized feedback control, and multi-level optimization [ATHA78, SAND78, LARS79b,c] the approach followed in this thesis is applicable to problems with complex, multi-modal or discrete problems which need not have special properties such as fast and slow modes.

Finally, the difference between the approach to distributed problem solving presented in this thesis and an approach by Tenny [TENN79, TENN81a, TENN81b] is the non-reliance on iteration to resolve uncertainty; Tenny dismisses iteration as inefficient and favors complex information gathering techniques.

2.2 Iterative Refinement

Iterative refinement is a very general and powerful centralized problem solving technique. The technique is also called the method of successive approximations and, according to Bellman and Dreyfus [BELL62], is "the most powerful of all tools of analysis" because of its general applicability as a method of solving problems of high dimensionality. Abstractly, the idea is as follows: given a problem and an objective function which can be used to determine the worth of a potential solution to the problem, one starts with an initial guess as to the solution of the problem and then repeatedly refines (improves upon) this approximate solution using the objective function. The algorithm terminates when further improvements are not possible or when the approximation is within prescribed limits of accuracy.

Iterative refinement is distinct from relaxation [KILM69, WALT75, ROSE76, DAVI76, ZUCK77, RISE77, HANS78], which starts with probability (or confidence) values for each alternative of each component of a solution and uses local constraints between solution components to modify these probability values until a single alternative for each component dominates. Thus, many alternatives for a single component of the solution vector are considered simultaneously with relaxation, whereas only a single alternative for each component of the solution vector is considered at any one time with iterative refinement. An earlier study [BROO79] found a relaxation approach to distributed network traffic light control to be quite interesting, but limited in

its ability to handle non-local interaction between distant traffic lights without introducing a hierarchy. Also, as the complexity of the application grows, so do the number of alternatives for each component of the solution. This significantly affects the speed of the relaxation algorithm.

Iterative refinement is also distinct from iterative synthesis, which iteratively builds a complete solution from components (often maintaining many alternative, partial solutions at the same time) until an appropriate complete solution is constructed. Examples of the iterative synthesis technique include spatial dynamic programming [LARS79a, CLIN79] and the Hearsay approach [LESS77, ERMA79, ERMA80].

Still, it is possible to apply the iterative refinement technique in a variety of ways depending on how refinements are computed. Classical iterative refinement algorithms include fixed-point and other "iterative methods" of numerical analysis [CONT65], and the well-known hill-climbing or gradient methods [COOP70]. Non-classical iterative refinement algorithms such as the SIGOP II network traffic light control algorithm [LIEB76] and the Dynamic Programming Successive Approximations method [LARS70] have appeared more recently.

2.2.1 Classical iterative refinement.

Classical iterative refinement algorithms typically use the value of the objective function at the current approximate solution or the derivative of the objective function at the current approximate solution to compute refinements. Fixed-point iterations, for example, use an objective function, in this case f , directly to find a solution, x , such that $x=f(x)$. This is accomplished by starting with an initial guess, x_0 , and computing $x_1=f(x_0)$, $x_2=f(x_1)$, etc. until $x_t=x_{t-1}$ or the difference is small enough. Unfortunately, there are restrictions which must be satisfied in order for convergence to be assured. The function, f , must be continuous and differentiable, and the magnitude of the derivative must be less than 1 in the neighborhood of the solution. Solutions may also be vectors, in which case f is called an operator. Other, similar classical iterative refinement algorithms have been developed to find the roots of non-linear functions and to solve differential equations numerically.

Baudet [BAUD78] has shown that certain fixed-point iterations can be performed quickly using an asynchronous multi-processor system. Similar restrictions to those encountered with centralized fixed-point iterations apply to these asynchronous iterations. Very few additional restrictions, however, are introduced by distributing the computation. This was an important indication that the distribution of another type of iterative refinement algorithm might succeed.

Hill-climbing or gradient methods are also well-known variations of the classical iterative refinement technique. Here the gradient of an objective function at approximate solutions is repeatedly utilized to perturb the current approximate solution towards the maximum or minimum of the objective function. In general, the objective function must be continuously differentiable; there are, however, hill-climbing techniques that sample points in the neighborhood of a potential solution to estimate the gradient when it is difficult or impossible to calculate. Although ideally suited for problems with convex objective functions, gradient methods generally yield poor results when started with a poor initial prime and faced with a non-convex or multi-modal objective function. Gradient methods are also inappropriate for problems where there is no scalar relationship between possible values for solution components.

Gallager [GALL77] developed a routing algorithm for a packet-switched communication network such as the ARPANET, which is a distributed gradient algorithm. A routing table at each node specifies what fraction of packets destined for other nodes should leave on each of the node's communication links. The distributed gradient algorithm incrementally updates each node's routing table based on information communicated between adjacent nodes about the marginal delay to each destination. Unfortunately, the algorithm is not applicable to problems with complex, non-convex or multi-modal objective functions. But again, this successful distribution of an iterative refinement algorithm is encouraging.

2.2.2 Non-classical iterative refinement.

The objective function is used to evaluate a set of possible refinements (perturbations) to the current approximate solution in a non-classical iterative refinement algorithm. The best perturbation is then chosen and perturbations of the new solution are considered. A Dynamic Programming Successive Approximation algorithm [LARS70] is an example of a non-classical iterative refinement algorithm. The SIGOP II network traffic light control algorithm is another non-classical iterative refinement algorithm.

Larson's Dynamic Programming Successive Approximations technique is an iterative refinement technique that has been applied to a reservoir system flow control problem [LARS70]. Each component of a control vector determines the flow from a reservoir through a unique dam, in a system of connected reservoirs. The algorithm starts with an initial control vector and determines the water levels in all reservoirs over time, that result from the use of the initial vector. Then, each flow from a dam and the associated reservoir's water level is readjusted, in turn, so as to minimize a global objective function. This sequence of readjustments is then repeated, until no further improvements are possible.

From the standpoint of distributing this algorithm, it is unfortunate that when changes in a dam's control and associated reservoir's water level are considered, other dams' controls must be readjusted to keep other reservoir water levels constant. This implies

that each processing element cannot have sole control over its assigned dam, and must possess knowledge of other dams, reservoirs, and their relationship.

The iterative refinement technique underlying the SIGOP II algorithm, however, does utilize a problem decomposition that appears to be well suited for distributed problem solving. Again, a solution vector is sought, but each refinement concerns only a single component of the solution vector.⁵ This iterative refinement technique is useful in a centralized system because the problem of refining a single component of an approximate solution vector is generally of lower complexity than the original problem of finding an optimal solution vector, and overall computational requirements are reduced. This type of iterative refinement is described in more detail in the next chapter and provides a basis for the distributed iterative refinement algorithms developed in this thesis.

5. This version of iterative refinement is most clearly explained when there is one subproblem for each component of the solution vector. The technique is easily extended to allow for the solution of multiple components in one subproblem, but for clarity only the simpler case is discussed in this thesis.

CHAPTER III

DISTRIBUTED PROBLEM SOLVING WITH ITERATIVE REFINEMENT

In this chapter a distributed iterative refinement technique is developed based on a centralized iterative refinement technique. The centralized iterative refinement technique is first described in detail. Distributing the iterative refinement technique among processing elements (PEs) is then discussed, the basic distributed iterative refinement algorithm is described, and the cases of localized and non-localized interaction are introduced.

3.1 A Centralized Iterative Refinement Algorithm

As a beginning, consider a very general optimization problem to be solved with an iterative refinement algorithm,

$$\begin{aligned} & \text{MIN}_{\underline{u}} C(\underline{u}) && (3.1) \\ & \text{subject to } \underline{u} \in \underline{U}, \end{aligned}$$

where the solution, \underline{u} , is a vector of solution components, (u_1, u_2, \dots, u_N) . Assuming the objective function, C , is well defined for all solution vectors, \underline{u} , (i.e., $C(\underline{u})$ takes on finite value for all $\underline{u} \in \underline{U}$), the i th subproblem can be stated as

$$\begin{aligned} & \underset{u_i}{\text{MIN}} \quad C(\hat{u}_1, \dots, \hat{u}_{i-1}, u_i, \hat{u}_{i+1}, \dots, \hat{u}_N) & (3.2) \\ & \text{subject to } u_i \in \{u \mid (\hat{u}_1, \dots, \hat{u}_{i-1}, u, \hat{u}_{i+1}, \dots, \hat{u}_N) \in \underline{U}\}. \end{aligned}$$

The centralized iterative refinement algorithm proceeds from an initial guess, \underline{u}_0 (commonly called a prime), repeatedly solving the sequence for $i=1,2,\dots,N$ of subproblems given by (3.2). The most recent value for u_j is represented by \hat{u}_j , and the algorithm terminates when a sweep of N optimizations results in no change.

Since a global view of updates (changes to components of \underline{u}) is assumed, convergence to a solution is easily guaranteed provided that only a single update is considered each iteration and up to date information is used. Of course, updates must only be allowed if they do not result in an increase of the objective function. But unfortunately, convergence to non-optimal solutions (local minima, points along a ridge, etc.) may result for problems with multi-modal search spaces, especially when \underline{u}_0 is a poor prime (i.e., it is far from the optimal solution).

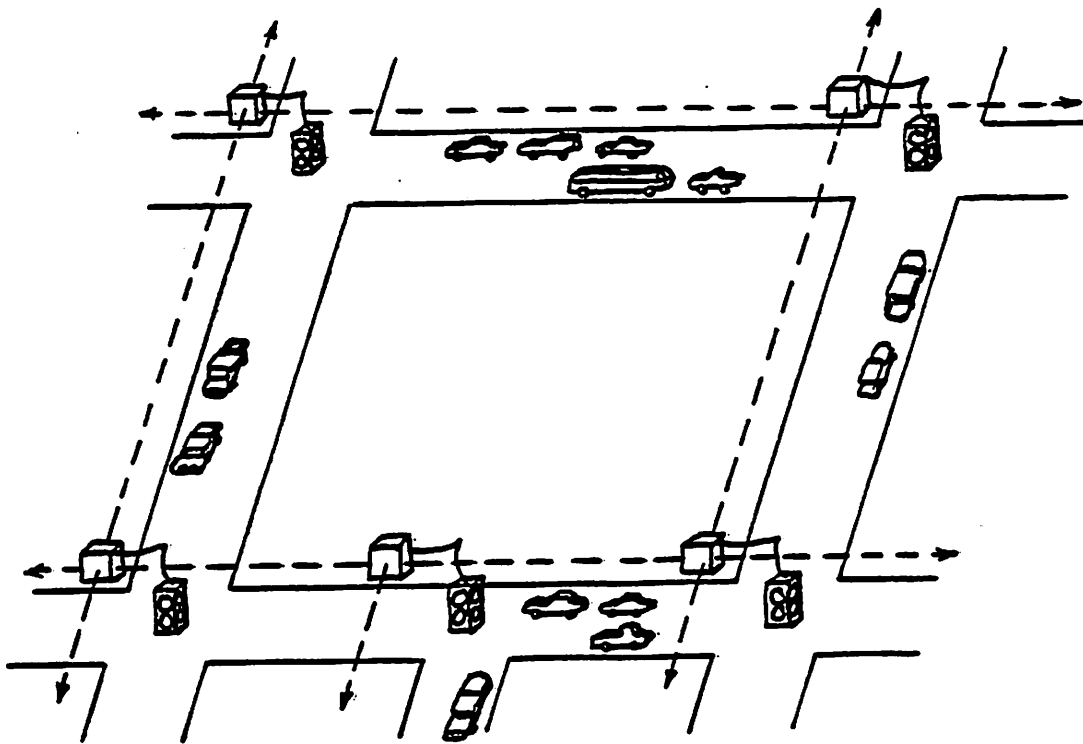
The inability to always find a global optimum results because only updates that involve single control variables are considered each iteration. Thus, improvements requiring coordinated changes to \underline{u} are not considered. However, this version of iterative refinement is less susceptible to convergence to non-optimal solutions than gradient methods because a line search (i.e., a complete search with respect to the control variable) rather than a gradient (i.e., the effect of an incremental change of the control variable) is utilized in the

modification of each u_i .

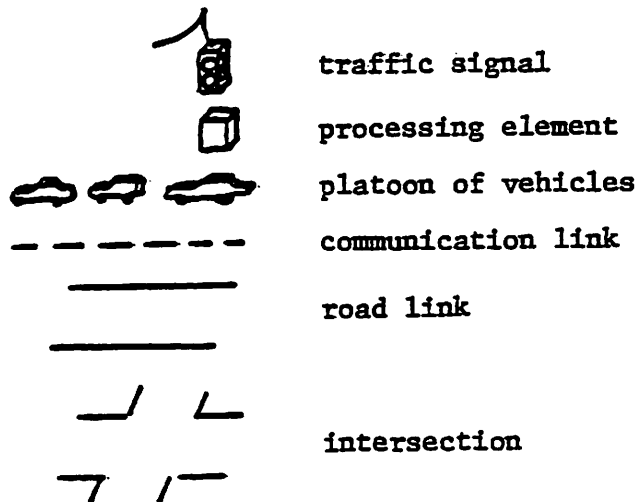
3.2 Distributing the Iterative Refinement Technique

The first step in distributing the centralized iterative refinement technique described above is to choose a problem decomposition. Generally, the components of a control vector are associated with spatially distributed sensors and/or effectors. This spatial distribution lends itself quite well to a decomposition where a unique processing element is assigned to each of the N components.

The assignment of processing elements to subproblems, one for each component (control), is depicted in Figure 1. To solve equation (3.2), however, each processing element (PE) must possess knowledge of the global objective function, and must acquire current values for all other PEs' controls unless some property of the application limits the effects of changing controls. In a large distributed system point to point communication between arbitrary processing elements is generally impractical. For this reason, approximations to (3.2) that limit a processing element's need for non-local information are examined in Chapter 7. Also, a cooperative non-local information gathering technique is developed in Chapter 7 that provides processing elements with larger than local views using neighbors-only communication.



KEY:



The assignment of processing elements (PEs) to controls is depicted for a traffic light control application. A unique PE is assigned to solve for the control of each traffic signal. Ideally, PEs should communicate only with PEs associated with adjacent signals.

Figure 1: Assignment of Processing Elements to Controls.

A basic, often unstated assumption of the centralized iterative refinement technique is that solutions to other subproblems (other processing elements' current controls) will not change during the computation of each subproblem. Because there is a critical region between the attainment of information from another processing element and the calculation of a new local control based on this information, the simultaneous solution by another processing element of an interacting subproblem may invalidate the calculation and produce an outdated result. Thus, the simultaneous refinement of subproblems by two or more PEs may actually lead to an increase in the global objective function rather than a decrease. This is the simultaneous-update problem. In fact, a straight forward, lock-step parallel approach where on each iteration any PE could make a refinement almost always resulted in oscillation, as is demonstrated with the numbering problem experiments of Chapter 5.

Because of the simultaneous-update problem, a key issue in developing an effective distributed iterative refinement algorithm is the limiting of simultaneous updates between interacting processing elements or collisions, thereby controlling uncertainty introduced by simultaneous updates. Schemes to restrict simultaneous updates are called update control schemes. A major focus of this thesis is the investigation of such schemes. Note that there is already a tolerance to error in the iterative refinement algorithm so that it is not a matter of completely avoiding collisions, but rather a matter of limiting the frequency of collisions and the magnitude of effect of

collisions. Of course, convergence is assured if an update control scheme is employed that does not allow simultaneous updates among interacting processing elements.⁶

The simultaneous-update problem is specifically addressed in Chapter 4. The problem is treated as a problem of access control of distributed resources. Below the basic distributed iterative refinement algorithm is described.

3.2.1 The basic algorithm.

In the remaining chapters a number of distributed iterative refinement algorithms are developed. A distributed iterative refinement algorithm is executed by a network of processing elements, but all processing elements perform the same computation in a lock-step (synchronized) parallel fashion. Thus, a distributed iterative refinement algorithm can be described by specifying the basic algorithm for an individual processing element. Each processing element is responsible for its own control variable; however, the common goal of all processing elements is to find a control vector for which no further refinements can be found.

6. The simplest such scheme is the sequential, serial update control scheme.

The basic algorithm for an individual processing element is shown in Figure 2. The basic algorithm consists of a number of phases: initialization, acquisition, calculation, and alteration. These phases are described below. At the top-most level, the initialization phase is run and then the sequence of acquisition, calculation, alteration phases is repeated until convergence (or a time-out) is detected in the evaluation phase.

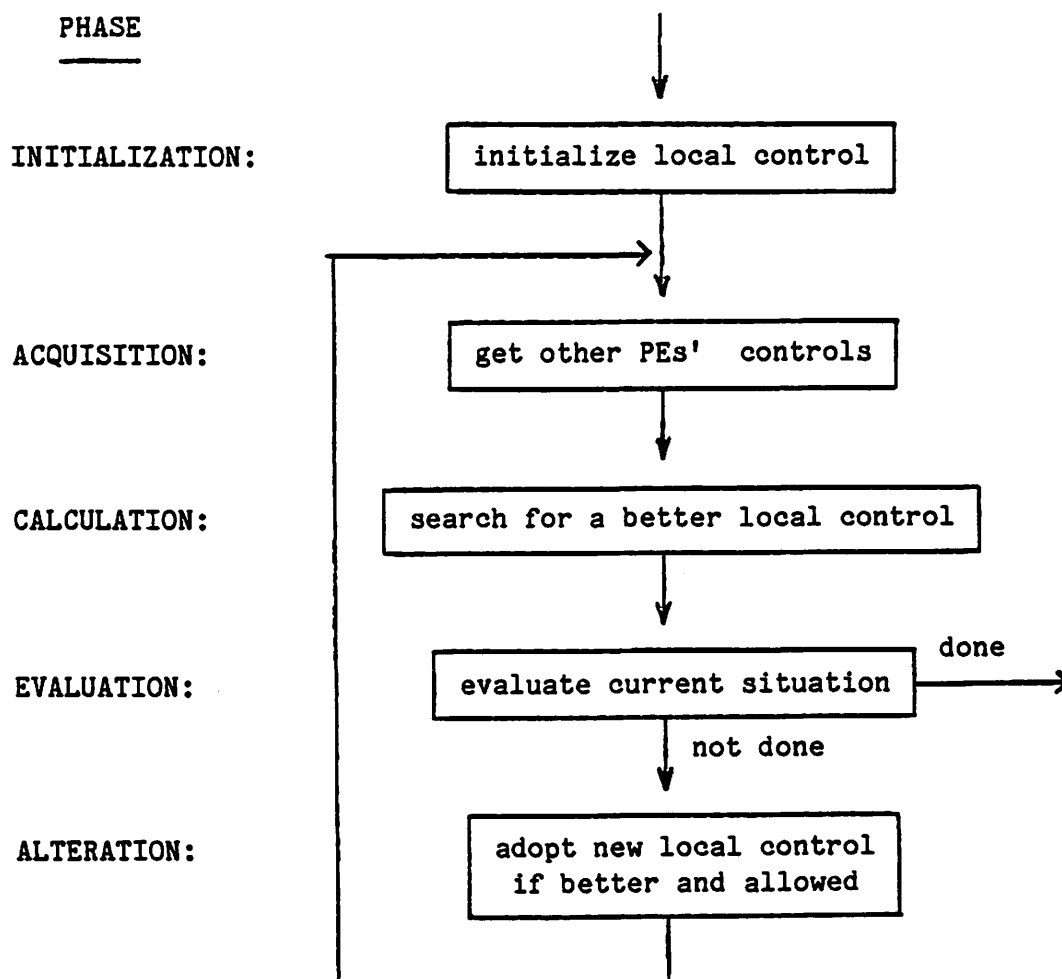
Initialization Phase. During the initialization phase a PE initializes its local control to "prime" the iterative refinement algorithm. If the algorithm is run periodically in a quasi-static (slowly changing) environment to obtain an more appropriate control vector, the control vector used in the previous control period may serve as a prime; otherwise, a fixed, random, or heuristically obtained prime may be used. This process is described below for PE_i .

```
(* INITIALIZATION PHASE: initialize local control      *)
(* other variables are also initialized at this time   *)
(* u[i] is PEi's control to be initialized to PRIME[i] *)
(* it_count will keep count of the number of iterations *)

u[i] := PRIME[i];
it_count := 0;
```

Other initializations may take place at this time. Unless these other initializations require communication, no communication is required for the initialization phase.

Acquisition Phase. A PE acquires through communication other PEs' controls during the acquisition phase. Note that a PE must make its local control available to other PEs during this phase, since other PEs are in their acquisition phase too. This process is described below for



The basic algorithm for an individual processing element (PE) consists of a number of phases: initialization, acquisition, calculation, evaluation, and alteration. All PEs perform this computation in a lock-step, parallel fashion, until no PE can find a better control or a pre-specified number of iterations have passed. All phases except the calculation phase may involve communication with other PEs.

Figure 2: Basic Algorithm for a Processing Element.

PE_i.

```
(* ACQUISITION PHASE: get other PEs' controls *)
(* N is number of PEs *)
(* u[j] is PEj's current control *)

for j := 1 to N
  do if i≠j and PEj's control is needed for PEi's calculation phase
     then get u[j] from PEj;
```

In many cases only a subset of other PEs' controls are needed for the calculation phase and, thus, need to be communicated during the acquisition phase.

Calculation Phase. Each processing element solves its subproblem during the calculation phase to search for a better local control. This phase is generally more time-consuming than all the other phases of the algorithm. Because a processing element needs other PEs' current controls for the solution of its subproblem, this information was communicated during the acquisition phase. The calculation process is depicted below.

```
(* CALCULATION PHASE: search for a better local control *)
(* u[i] is current local control *)
(* cost[i] is current cost associated with using u[i] *)
(* û[j] is PEi's copy of PEj's current control *)
(* best[i] will be better local control *)
(* demand[i] is set to indicate if PEi wants to *)
(* alter its control *)

best[i] := u[i];
cost[i] := C(û[1],...û[i-1],u[i],û[i+1],...û[N]);
min_cost := cost[i];
for u := all values for u[i] such that constraints are satisfied
  do if C(û[1],...û[i-1],u,û[i+1],...û[N]) < min_cost
     then begin
         min_cost := C(û[1],...û[i-1],u,û[i+1],...û[N]);
         best[i] := u
     end;
demand[i] := (best[i] = u[i]);
```

Because a PE is interested only in changing its local control, it is often the case that a simpler cost function will suffice. This often happens because the local control seldom interacts with all other PEs' controls. By simplifying the cost functions PEs use, the number of other PEs' controls acquired during the acquisition phase may be reduced. No communication is required for the calculation phase.

Evaluation Phase. After the acquisition phase processing elements generally have consistent views of the situation they are faced with given the current solution vector.⁷ Local and possibly global evaluations are performed at this time. Convergence (or a time-out) may also be detected during this phase. An evaluation process is given below.

```
(* EVALUATION PHASE: evaluate current situation *)
(* demand[i] was set during the calculation phase to *)
(* indicate if PEi wants to alter its control *)
(* converged[i] will be set to indicate if convergence *)
(* has been achieved *)
(* it_count is current iteration number *)

it_count := it_count + 1;
converged[i] := (not demand[i]);
for j := 1 to N
  do if ((j<>i) and converged[i])
      then begin
            get converged[j] from PEj;
            converged[i] := (converged[i] and converged[j])
          end;
if ((it_count > it_max) or converged[i])
  then exit;
```

7. Communication errors or delays can lead to errors and inconsistent views. However, if errors are infrequent, the general robustness of the distributed iterative refinement algorithm resolves these additional errors. This is demonstrated in experiments reported in Chapter 7.

Communication is required for convergence detection and for global evaluations.

Alteration Phase. A processing element that finds a better local control must not always adopt the new control, else the simultaneous-update problem results. For this reason, an alteration phase is included in the algorithm, during which processing elements use an update control scheme to determine which processing elements may adopt new controls. This phase is depicted below.

```
(* ALTERATION PHASE: adopt new control if better and allowed *)
(* demand[i] was set during the calculation phase to *)
(* indicate if PEi wants to alter its control *)
(* u[i] is current local control *)
(* best[i] is new local control *)
(* allowed(i) is a call to an update control scheme *)

if demand[i]
  then if allowed(i)
    then u[i] := best[i];
```

Eight update control schemes are described in detail in the next chapter. Because some update control schemes require communication, some communication may be required for the alteration phase.

3.2.2 The case of localized interaction.

The distributed iterative refinement algorithm described above has been designed for applications where there is a natural spatial distribution of effectors (and sensors) and a unique processing element is assigned to each effector. Based on this spatial distribution, two processing elements are said to be neighbors if they are spatially

adjacent. Communication between neighboring processing elements is assumed to be direct (point to point), inexpensive, and fast, while communication between non-neighboring processing elements is assumed to be indirect, expensive, and slow.

With this in mind, interaction is said to be localized if only neighboring processing elements' subproblems interact. This case is interesting because a limited, local view of the effects of changing a control is equivalent to a global view, and because during the acquisition phase of iterative refinement algorithm a processing element need only acquire neighbors' current controls. Thus, the lack of a global view presents no problem, and the only source of uncertainty which must be faced by processing elements, unless communication errors are considered, is the simultaneous-update problem which is discussed in Chapter 4. Experiments pertaining to the case of localized interaction are discussed in Chapter 5.

3.2.3 The case of non-local interaction.

When there is interaction between non-neighboring PEs' subproblems interaction is said to be non-local. Many real applications exhibit non-local interaction, including the network traffic light control problem examined later in this thesis. The presence of non-local interaction has the following implications: the simultaneous-update problem extends beyond neighbors (i.e., a PE may collide with a non-neighboring PE), each iteration PEs must acquire non-neighbors'

current controls to accurately assess its current situation, and a global view of the effects of changing a control is needed to accurately search for a better control.

Because of these implications, it may not be possible to maintain correctness and consistency at all times during the problem solving process in order to solve a problem with non-local interaction distributively with an effective utilization of processing resources and with limited communication. It may be necessary to trade off some accuracy for increased speed and decreased communication by introducing quick approximations based on local information (e.g., use a limited view). The lost accuracy will hopefully be regained through a few extra, less expensive iterations as neighbors' controls eventually embody the considerations of non-neighbors.

The first implication, that the simultaneous update problem extends beyond neighbors, is simply ignored (i.e., no attempt is made to control collisions between non-neighboring PEs). Preliminary experiments [BROO79] indicated that if simultaneous updates between neighboring PEs were controlled, the distributed iterative refinement algorithm resolved the additional uncertainty introduced by simultaneous updates between non-neighboring PEs. Since it is possible that oscillation will occur making it difficult to choose a solution if a deterministic update control scheme is employed and simultaneous updates between non-neighboring PEs is ignored, periodic global evaluations of tentative solutions is considered as an alternative to simply taking the last solution after some fixed number of iterations.

The second implication, the need to communicate non-neighboring PEs' controls is addressed through the introduction of state variables (which abstract the interaction of non-neighboring PEs) and state updating (which propagates non-local interactions). This allows a PE to solve at least part of its subproblem using only information obtained from neighboring PEs, and is discussed in detail in Chapter 7. Although state updating requires communication between neighboring PEs, partial state updating is also considered to limit this communication.

The third implication, that PEs require knowledge of the global objective function is called the local-view problem. The local-view problem is addressed in a number of ways. A local-view approximation assumes a local view is a good approximation to a global view. With a local-view approximation communication is minimized, but desired accuracy may not always be achieved. Furthermore, complications arise in assuring convergence and selecting a solution. The local-view approximation is described in detail in Section 7.1. A worst-case assumption is considered in Section 7.2 as a means of decoupling non-neighbors' subproblems. Cooperative gathering of non-local information (with abstraction) is discussed in Section 7.3. Although more communication is needed for the cooperative gathering technique, better solutions may result. The effectiveness of the various techniques is determined via experiments discussed in Chapter 8.

C H A P T E R I V

THE SIMULTANEOUS-UPDATE PROBLEM

The simultaneous-update problem is addressed in this chapter. The simultaneous-update problem is the problem of encountering unexpected, often negative results from processing elements simultaneously solving interacting subproblems. The approach taken here is to view this as a problem of access control of distributed resources, and to use access control schemes to control simultaneous updates. This approach is then tested in experiments that follow in Chapter 5.

To see how the simultaneous-update problem may be viewed as an access control problem, consider a processing element which has solved its subproblem and obtained a solution. Because processing elements operate in parallel, the processing element faces uncertainty as to the validity of its solution because solutions to other subproblems that the processing element's subproblem was dependant on may have (simultaneously) changed. If one imagines an abstract entity, called "the right to update," which is a resource shared by processing elements whose subproblem interact, then in order to effectively utilize this shared resource it must be accessed by a single processing element at a time. Thus, the problem of update control for iterative refinement is transformed into a problem of access control.

4.1 Access Control

Unlike the problem of update control for an iterative refinement algorithm, the access control problem has been studied by a number of researchers. Two well known applications, packet radio [KLEI78, TOBA80, THUR81b] and multidrop coax bus (Ethernet) [METC76, SHOC80, THUR81b] broadcast communication concern decentralized access control of a communication medium. Since both these applications involve communication over a shared path, a determination must somehow be made as to which PE is allowed to transmit.

Unfortunately, these applications involve controlling access to a single shared resource (e.g., the packet radio communication schemes generally assume all PEs transmit to a single satellite). However, much of the theory is applicable to the more general case of access control of multiple, distributed resources. What stands in the way of using this theory for the more general case of packet radio communication is the inability of processing elements to observe all collisions with their transmission, and the fact that acknowledgements which would supply this information require some of the very resource which is to be controlled. The update control application, on the other hand, does not involve the control of a communication resource.

In general, there are many variations of the decentralized access control problem. Variables include:

- o how the resources are shared (which processing elements have access to what resources),
- o how demand for resources arises (model with probabilities or rates which specify what resources are needed and for how long resources are needed when demand arises), and
- o what the payoffs for various events (exclusive access, collision, etc.) are, and how they are combined in an objective function for processing elements.

Different access control problems, as defined by these variables, often require different solutions. The access control equivalent of the update control problem is discussed in the next section.

Access control strategies can be either fully controlled or partially controlled. With a fully controlled strategy, which PEs are allowed to access resources is determined before any attempt is made to utilize resources. Controlled strategies avoid collisions, and include reservation schemes and selection schemes. Reservation schemes work by establishing a schedule of resource use in advance statically (as with TDMA, time division multiplexed access) or dynamically. Selection schemes involve arbitration by priority, voting, or sequencing before resources are used.

Partially controlled access control strategies include random access or contention schemes. With a partially controlled access control strategy, PEs access resources whenever demand for them arises, and if a collision occurs a retry scheme is employed. Sometimes a little discretion is used and PEs do not attempt to access resources if

they are known to be in use. Collision detection can be done any number of ways, depending largely on characteristics of the specific application. There are also quite a few retry schemes, including random, fixed, adaptive, and even reservation schemes. Some fundamental fully controlled and partially controlled access control schemes will be discussed after the following section on access control for iterative refinement update control.

4.1.1 Access control for update control.

The characteristics of the iterative refinement update control problem (viewed as an access control problem) can be summarized as follows:

- o an arbitrary network is assumed with many shared resources, and use of resources is time-slotted with a single time-slot needed when demand arises;
- o demand for resources arises in an unknown, random manner, and PEs are "greedy" (i.e., an attempt is made to access all resources accessible to a PE when demand arises); and,
- o the payoff for exclusive access to shared resources is positive but variable and there is a variable penalty for collisions.

In choosing an access control scheme for update control, these characteristics must be considered.

There are important differences between the iterative refinement update control problem and the communication channel access control problems. For example, a single time-slot is needed to satisfy demand for the update control problem, whereas many consecutive time-slots may be useful for broadcast communication applications. This difference makes carrier-sense techniques that have proved to be quite successful for Ethernet [SHOC80] inappropriate for update control in an distributed iterative refinement algorithm. A second difference is that a collision does not necessarily lead to zero utilization for the update control application (i.e., it may be negative), whereas most communication applications consider a lost packet to be zero utilization. These differences make it necessary to extend some of the existing work on access control for its use in alleviating the simultaneous-update problem.

4.1.2 Some fundamental access control schemes.

Unless demand is sporadic, collisions will regularly occur with distributed iterative refinement if no access control scheme is utilized, resulting in oscillation. There are a variety of fundamental access control schemes of potential interest for the iterative refinement update control application. These include both controlled and uncontrolled schemes.

A serial control scheme gives PEs access to shared resources one at a time in some order. Thus, the serial scheme is a controlled, static reservation scheme like TDMA. The particular order is generally considered to be unimportant. If there are N PEs, then all N PEs have access to shared resources once and only once each N iterations. Unfortunately, this permits no parallelism, thus defeating much of the potential advantage of a distributed approach.

An alternating control scheme cycles between pre-specified groups of PEs, giving all PEs in the currently chosen group access rights. The alternating control scheme is also a controlled, static reservation scheme. Usually, each PE is a member of only a single group, and PEs that share resources are members of different groups. Note that this scheme degenerates to a serial control scheme if all PEs share some resource.

A negotiated control scheme requires communication before each time-slot and allows only a subset of PEs which do not compete to access resources. The negotiated control scheme is a controlled selection scheme. A number of variations of this basic approach are possible, some employing fixed or dynamic priority mechanisms. The version employed in later experiments has neighboring PEs communicate their demand and employs a rotating priority mechanism based on an alternating PE ordering to resolve conflicts in a fair manner. A negotiated scheme can give near perfect utilizations, but at the cost of communicating with contending PEs.

A probabilistic control scheme involves the use of a policy, p_i , by each PE_i to determine if it should have access rights for each time-slot. The policy is a probability value, and represents the probability that PE_i is given access rights given that PE_i has demand. The probabilistic scheme is an uncontrolled, random access scheme. The optimum value(s) for $\underline{p}=(p_1, p_2, \dots, p_N)$ depend upon the optimality conditions chosen. Below, Pareto optimality is used as a decentralized optimality condition to derive a balance principle for optimal access control. Because the policy may be varied over time, it is shown how this control scheme can be made demand adaptive.

An urn control scheme [KLEI78, YEMI78, YEMI80] is an interesting alternative to the probabilistic decision mechanism described above. The urn scheme is another uncontrolled random access schemes. Under the urn scheme, each PE_i draws k_i numbers from identical pseudo-random number generators using a common seed and if it drew its number it may attempt utilization of shared resources. The probability of drawing a particular number from a pseudo-random number generator in a sample of k numbers is similar to the probability of drawing a ball of a particular color from an urn containing various colored balls in a sample of k balls; hence, the name "urn" scheme. The use of a common seed and identical pseudo-random number generators coordinates the decisions of PEs without communication.

Actually, a variety of urn schemes are possible, depending upon how numbers (colors) are assigned to PEs and what numbers can be drawn from a pseudo-random number generator (how many balls of each color are in an urn). Perhaps the simplest urn scheme assigns a unique number to each PE, and has PEs draw numbers from identical, randomly ordered lists of the assigned numbers. A better urn scheme, though, assigns the fewest numbers to PEs such that each PE is assigned a number yet in each neighborhood no PE has the same number, and has PEs draw numbers from identical, randomly ordered lists of the (fewer) assigned numbers. Note that when each PE may interfere with all PEs, the schemes are equivalent. The use of multiple urns has been proposed [YEMI80] which produces a scheme similar to the tree scheme [CAPE79].

The balance principle derived below for the probabilistic scheme may also be used to determine an optimal k for an urn scheme. The urn scheme, as Kleinrock and Yemini showed in [KLEI78] and will be shown below, achieves a utilization similar to the probabilistic scheme when demand is light, and significantly better utilization when demand is heavy.

4.1.3 Discussion.

Update control schemes of all the types listed above are employed in distributed iterative refinement algorithms to control simultaneous updates. These update control schemes are described in greater detail at the end of this chapter along with the SIGOP update control scheme (a

serial scheme with heuristics) and the attenuated update control scheme (a parallel scheme that limits the magnitude of updates rather than their frequency). The theory needed for the probabilistic and urn update control schemes is developed next; readers not interested in this theory should continue with Section 4.3.

4.2 Pareto-Optimal Access Control for Two Schemes

In this section the general problem of optimal, distributed multi-access control of shared resources in a network of processing elements is addressed outside the context of the iterative refinement application. The balance principle derived below is appropriate for access control schemes for which the probability that a processing element will attempt to access shared resources given it has demand can be expressed as a function of the processing element's access control policy. The probabilistic and urn schemes described above are this type of access control scheme. The balance principle can be used to determine optimal access control policies for these schemes in a number of ways. An adaptive balancing algorithm is also developed to handle variable demand.

The research presented in this section is based on recent contributions to the field of Communication and Networking by Yechiam Yemini and Leonard Kleinrock [KLEI78, YEMI78, YEMI79] concerning a balance principle and adaptive algorithms for multi-access control of a shared communication channel in a packet radio network. This research

extends their work to cover a broader range of objective functions which, in addition to specifying the utilization of a success (exclusive access to shared resources), may specify a penalty for collisions and/or sacrifices (decisions to not attempt access though in need of resources). A more general balance principle results because an alternative interpretation of the necessary conditions is found to be appropriate. Furthermore, a new adaptive balancing algorithm that has been developed and tested is presented in Subsection 4.2.2.

4.2.1 A balance principle for optimal access control.

In this section a necessary condition for (Pareto) optimality is applied to the access control problem and an interpretation results in a general balance principle for optimal access control. The balance principle is quite general and is shown to hold for a wide range of models of utilization. Direct use of the balance principle is discussed, and theoretical performance figures are presented, comparing two schemes for which the balance principle was used to obtain Pareto-optimal policies and two standard schemes that do not use the balance principle.

It is assumed that each PE has access to a subset of all resources, and demand for resources arises in a random manner. Demand is assumed to be irregular, so a scheme which is demand adaptive is necessary to avoid a waste of resources. Use of resources is assumed to be time-slotted (i.e., use is synchronized to slots in time) and a PE

requires a single time-slot of all accessible resources when it has demand. This "greedy" assumption simplifies the analysis, but a probability distribution may be used to specify probabilistically which resources accessible to a PE will be needed when demand arises.

When two or more PEs attempt to access a shared resource simultaneously, a collision occurs, and for many applications the shared resource cannot be properly utilized. Because of this, an access control scheme that decides which PEs with need of resources should have the right to access the resources is desirable. The objective of an access control scheme is to maximize the expected utilization of resources. Note that the utilization of resources when a collision occurs is taken to be zero for many applications, but this need not be the case.

4.2.1.1 Objective functions.

The major difficulty in determining an optimal control scheme in a distributed manner is the lack of a global view of demand and resources in the network. For this reason, the optimization will be based on a simple relationship among local, decentralized objective functions rather than on a global objective function. A network utilization operator mathematically expresses the expected utilization of resources by each PE; it is a vector of local utilization operators, one for each PE. These local utilization operators are used as local objective functions.

A number of utilization operators are possible, depending for instance on which accessible resources will be needed when demand arises and how possible outcomes (success, collision, sacrifice, etc.) are combined. Different applications will call for different operators. While the balance principle will be developed below using only one utilization operator, it will later be shown to apply to a broad class of utilization operators.

Formally, let $\underline{\tilde{d}}^t = (\tilde{d}_1^t, \tilde{d}_2^t, \dots, \tilde{d}_N^t)$ designate the demand process for time-slot t .⁸ That is,

$$\tilde{d}_i^t = \begin{cases} 1 & \text{if PE}_i \text{ requires resources at time-slot } t, \\ 0 & \text{otherwise.} \end{cases}$$

There is no distinction between resources in this model because it is assumed that when a processing element attempts to access resources, it attempts to access all resources to which it has access. Let $\pi^t(\underline{d})$ designate the distribution of $\underline{\tilde{d}}^t$ (i.e., the likelihoods that PEs have demand). This distribution is not known, and is assumed to change slowly with time.

Define the resource utilization process of PE_1 during time-slot t as

8. A random variable and its mean are represented by \tilde{x} and \bar{x} , respectively. Vectors are underlined and sets are capitalized.

$$\tilde{u}_i^t = \begin{cases} 1 & \text{if PE}_i \text{ has demand and gains exclusive access} \\ & \text{to all shared resources,} \\ -\alpha & \text{if PE}_i \text{ has demand and one or more PEs} \\ & \text{interfere with PE}_i, \\ 0 & \text{if PE}_i \text{ has demand but does not attempt to access} \\ & \text{shared resources, or if PE}_i \text{ has no demand.} \end{cases}$$

Thus, a PE is awarded one unit of utilization for a success and is penalized α units for a collision; a sacrifice or being idle (having no demand) results in no utilization.⁹ For the packet broadcast problem examined by Yemini and others [KLEI78], [TOBA80] a penalty was not used (i.e., $\alpha=0$) since a packet whose transmission is interfered with is not lost because a copy is always held for possible retransmission. It is not the case that collisions are harmless for distributed iterative refinement.

Consider a probabilistic access control scheme¹⁰ where a utilization policy, $\underline{p} = (p_1, p_2, \dots, p_N)$, is a vector of probabilities, such that for each time-slot PE_i will attempt to access shared resources with probability p_i if it has demand. Assuming that a PE attempts to access all resources to which it has access when demand arises the mean utilization of PE_i when policy \underline{p} is used, conditioned upon $\underline{d}_i^t = \underline{d}$ is

9. A single penalty for a collision with any number of PEs and no penalty for a sacrifice is used here, but many other utilization processes are possible.

10. Other control schemes may be used provided the probability a PE will attempt utilization can be expressed as a function of the policy; another control scheme, the urn scheme, is described in Section 4.

$$\bar{u}_i^t(\underline{d}, \underline{p}) = \begin{cases} p_i \cdot \prod_{j \in I_i(\underline{d})} (1-p_j) - \alpha p_i (1 - \prod_{j \in I_i(\underline{d})} (1-p_j)) & \text{if } d_i=1 \\ 0 & \text{if } d_i=0, \end{cases}$$

where $I_i(\underline{d}) = \{j \mid (j \neq i) \wedge (d_j=1) \wedge (j \in I(i))\}$,

and $I(i) = \{j \mid \text{PE}_j \text{ potentially interferes with PE}_i\}$.

Clearly, PE_i 's expected utilization given demand \underline{d} and policy \underline{p} is zero if $d_i=0$ (i.e., PE_i has no demand). If $d_i=1$, however, PE_i 's expected utilization is 1 times the probability that it attempts and no PEs interfere minus α times the probability that it attempts and some PE interferes.

Assuming a particular distribution of demand (specified by π , the expected utilization of PE_i during time-slot t with policy \underline{p} is given by the local utilization operator,

$$\bar{u}_i^t(\underline{p}) = \sum_{\underline{d} \in \{0,1\}^N} \pi^t(\underline{d}) \cdot \bar{u}_i^t(\underline{d}, \underline{p}).$$

To simplify notation, the time index, t , will be eliminated in expressions where there is no confusion. The network utilization operator can now be described by

$$\underline{U}(\underline{p}) = \begin{bmatrix} \bar{u}_1(\underline{p}) \\ \bar{u}_2(\underline{p}) \\ \cdot \\ \cdot \\ \cdot \\ \bar{u}_N(\underline{p}) \end{bmatrix} .$$

Given a policy, \underline{p} , the network utilization operator yields the expected utilization of PEs.

4.2.1.2 Pareto optimality and a necessary condition.

Pareto optimality, a concept of mathematical economics and game theory [LUCE57] is an attractive choice of a decentralized optimality criterion because it is a weak form of optimality that requires minimal coordination of members of a decentralized community. At a Pareto-optimum it is impossible to increase the utilization of any PE by changing PEs' policies without decreasing the utilization of some other PE(s); thus, PEs are not selfish. A Pareto-optimal policy yields a Pareto-optimal utilization.

The following derivation of the necessary conditions for Pareto optimality is based on a similar derivation by Yemini [YEMI79]. Formally, a utilization vector, \underline{u}^* , is Pareto-optimal if and only if

- 1) it is attainable, i.e., $\underline{u}^* = \underline{U}(\underline{p}^*)$ for feasible policy \underline{p}^* , and
- 2) it is not dominated by another attainable utilization vector (i.e., $\nexists \underline{u}' \mid \forall i \ u'_i \geq u^*_i$, with at least one strict inequality).

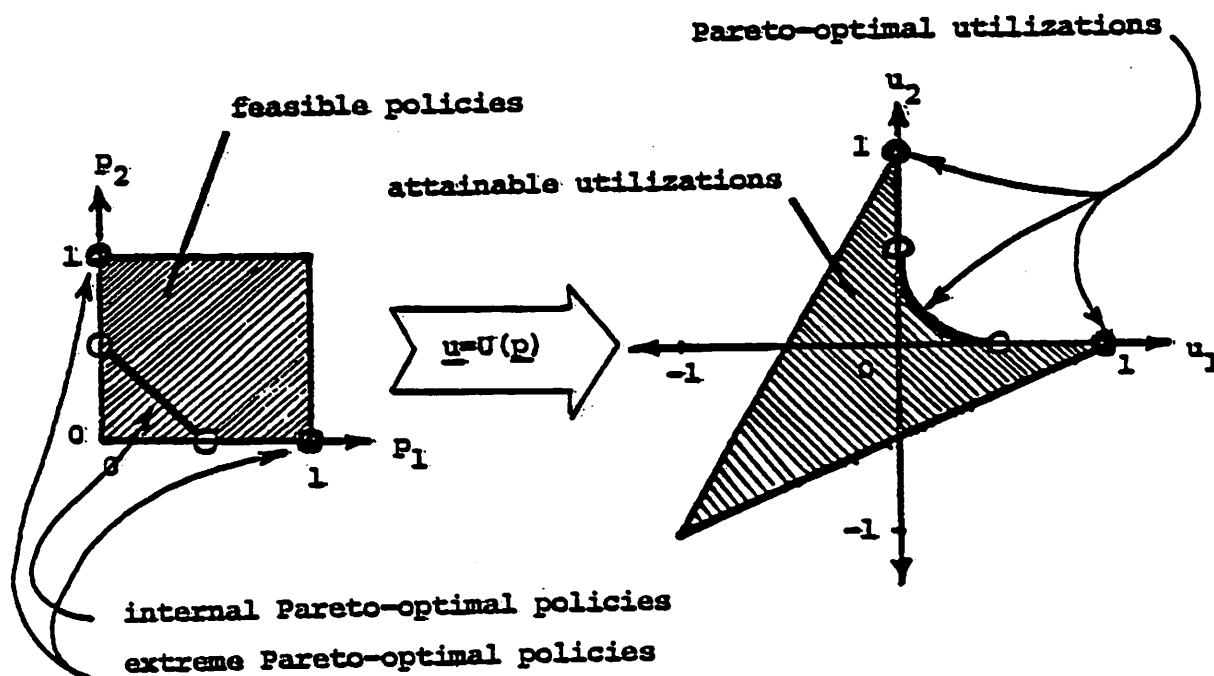
The concept of Pareto optimality has a simple geometric interpretation. To see this, consider the case of two PEs that both have demand for a single, shared resource. If the penalty for a collision is one (i.e., $\alpha=1$), the utilization operator is

$$\underline{U}(\underline{p}) = \begin{cases} \bar{u}_1(\underline{p}) = p_1(1-p_2) - p_1p_2 \\ \bar{u}_2(\underline{p}) = p_2(1-p_1) - p_2p_1 \end{cases}$$

and Figure 3 shows how the space of feasible policies is mapped onto attainable utilizations.

In this example, feasible policies are probability vectors for deciding whether to access shared resources when there is demand, so all components must lie between 0.0 and 1.0, inclusive. Each feasible policy vector has a corresponding utilization vector determined by the network utilization operator; these utilizations constitute the attainable utilizations. Pareto-optimal utilizations are all utilizations on the upper-right boundary of the region of attainable utilizations because such utilizations are not dominated by any other attainable utilizations (i.e., there is no other utilization that gives more to one PE without taking utilization away from another PE).

The set of feasible policies includes internal policies and extreme policies. A policy, \underline{p}' , is an extreme policy if $p'_i=0$ or $p'_i=1$ for some i . The necessary condition for Pareto optimality to be derived characterizes internal Pareto-optimal policies and does not necessarily hold for extreme policies. This is so because the extremality of $\underline{U}(\underline{p}')$



The utilization operator, U , maps feasible policies onto attainable utilizations. A feasible policy is a probability vector, so each component is a real number in the range $[0,1]$; attainable utilizations are all the utilization vectors associated with the feasible policies. Utilizations on the upper right boundary of the region of attainable utilizations are called Pareto-optimal policies, and the policies that map into these utilizations are called Pareto-optimal policies.

Figure 3: Utilization Operator.

may be directly caused by the extremality of \underline{p}' . Extreme Pareto-optimal policies must be found by other means.

Consider an internal Pareto-optimal policy, \underline{p}^* , and let $\underline{u}^* = \underline{U}(\underline{p}^*)$ be the resulting Pareto-optimal utilization. Provided $\underline{U}(\underline{p})$ is continuous, a small perturbation of \underline{p}^* leads to a small perturbation of \underline{u}^* . The utilization of these perturbed policies are related to \underline{u}^* by the following linear approximation:

$$\underline{U}(\underline{p}^* + \Delta \underline{p}) = \underline{u}^* + \Delta \underline{p} \left. \frac{\partial \underline{U}(\underline{p})}{\partial \underline{p}} \right|_{\underline{p}=\underline{p}^*}$$

where $\left. \frac{\partial \underline{U}(\underline{p})}{\partial \underline{p}} \right|_{\underline{p}=\underline{p}^*}$ is the Jacobian matrix of $\underline{U}(\underline{p})$ at \underline{p}^* .

Because \underline{p}^* is an internal point of the set of feasible policies, it admits perturbations in all directions. The extremality of \underline{u}^* implies that the attainable perturbations of \underline{u}^* must not admit perturbations in all directions. This condition occurs if the Jacobian matrix at \underline{p}^* is singular, because (according to the linear approximation) when this is the case there is no perturbation of \underline{p}^* that perturbs \underline{u}^* in the direction perpendicular to the boundary surface of attainable utilizations. Note that this condition is necessary but not sufficient for determining these extrema.

The singularity of the Jacobian at \underline{p}^* implies that there is a non-zero linear combination of rows of the Jacobian at \underline{p}^* which yields a zero vector. If $\underline{c} = (c_1, c_2, \dots, c_N)$ are the non-zero coefficients of such a linear combination, the necessary condition for Pareto optimality

of an internal policy can be stated as

$$\forall i \quad \sum_{j=1 \dots N} c_j \frac{\partial \bar{u}_j(\underline{p})}{\partial p_i} = 0 .$$

4.2.1.3 The balance principle.

For the utilization operator defined in Section 4.2.1.1, the elements of the Jacobian at \underline{p} are

$$\frac{\partial \bar{u}_i(\underline{p})}{\partial p_j} = \begin{cases} \phi_i(\underline{p}) & \text{if } i=j \\ \psi_{ij}(\underline{p}) & \text{if } i \neq j. \end{cases}$$

$$\text{where } \phi_i(\underline{p}) = \sum_{\underline{d} \in \{0,1\}^N} \pi(\underline{d}) \frac{\partial \bar{u}_i(\underline{d}, \underline{p})}{\partial p_i},$$

$$\psi_{ij}(\underline{p}) = \sum_{\underline{d} \in \{0,1\}^N} \pi(\underline{d}) \frac{\partial \bar{u}_i(\underline{d}, \underline{p})}{\partial p_j},$$

$$\frac{\partial \bar{u}_i(\underline{d}, \underline{p})}{\partial p_i} = \begin{cases} \prod_{k \in I_i(\underline{d})} (1-p_k) - \alpha (1 - \prod_{k \in I_i(\underline{d})} (1-p_k)) & \text{if } d_i=1 \\ 0 & \text{if } d_i=0, \end{cases}$$

$$\frac{\partial \bar{u}_i(\underline{d}, \underline{p})}{\partial p_j} = \begin{cases} -p_i \prod_{k \in I_1(\underline{d}) \wedge k \neq j} (1-p_k) - \alpha p_i \prod_{k \in I_1(\underline{d}) \wedge k \neq j} (1-p_k) & \text{if } d_i = 1 \wedge j \in I_1(\underline{d}) \\ 0 & \text{otherwise.} \end{cases}$$

Obviously these expressions are the marginal expected utilizations for PE_i given an incremental change in p_i or p_j ; but, these expressions also have another meaning. To see the alternative interpretations for $\phi_i(\underline{p})$ and $\psi_{ij}(\underline{p})$ it is necessary to look at the expressions by themselves (i.e., not as derivatives).

If PE_i has demand, $\phi_i(\underline{p})$ is the probability that no PEs with demand that may interfere with PE_i attempt to access shared resources minus a times the probability that some PE with demand that may interfere with PE_i does attempt (and hence collides with PE_i). If PE_i does not have demand, $\phi_i(\underline{p})$ is 0. Thus, $\phi_i(\underline{p})$ can be interpreted as PE_i 's expected utilization with policy \underline{p} given that it attempts to access shared resources if it has demand. Note that if $\alpha=0$, $\phi_i(\underline{p})$ may alternatively be interpreted (as Yemini did) as the probability that shared resources to which PE_i has access will not be utilized given that PE_i does not attempt to access them if it has demand.

To interpret $\psi_{ij}(\underline{p})$, note that

$$\psi_{ij}(\underline{p}) = \begin{cases} -X_{ij}(\underline{p}) + Z_{ij}(\underline{p}) & \text{if } j \in I(i) \\ 0 & \text{otherwise,} \end{cases}$$

$$X_{ij}(p) = \sum_{\underline{d} \in \{0,1\}^N} \pi(\underline{d}) \cdot \bar{x}_{ij}(\underline{d}, p),$$

$$\text{where } Z_{ij}(p) = \sum_{\underline{d} \in \{0,1\}^N} \pi(\underline{d}) \cdot \bar{z}_{ij}(\underline{d}, p),$$

$$\bar{x}_{ij}(\underline{d}, p) = \begin{cases} p_i \prod_{j \in I_1(\underline{d}) \wedge k \neq j} (1-p_k) - \alpha p_i (1 - \prod_{j \in I_1(\underline{d}) \wedge k \neq j} (1-p_k)) & \text{if } d_i = d_j = 1 \\ p_i \prod_{k \in I_1(\underline{d})} (1-p_k) - \alpha p_i (1 - \prod_{k \in I_1(\underline{d})} (1-p_k)) & \text{if } d_i = 1 \wedge d_j = 0 \\ 0 & \text{if } d_i = 0. \end{cases}$$

$$\bar{z}_{ij}(\underline{d}, p) = \begin{cases} -\alpha p_i \prod_{j \in I_1(\underline{d}) \wedge k \neq j} (1-p_k) - \alpha p_i (1 - \prod_{j \in I_1(\underline{d}) \wedge k \neq j} (1-p_k)) & \text{if } d_i = d_j = 1 \\ p_i \prod_{k \in I_1(\underline{d})} (1-p_k) - \alpha p_i (1 - \prod_{k \in I_1(\underline{d})} (1-p_k)) & \text{if } d_i = 1 \wedge d_j = 0 \\ 0 & \text{if } d_i = 0, \end{cases}$$

through the addition of some dummy terms. For $j \in I(i)$, $X_{ij}(p)$ can be interpreted as PE_i 's expected utilization with policy p given PE_j does not attempt to access shared resources if it has demand. And, for $j \in I(i)$, $Z_{ij}(p)$ can be interpreted as PE_i 's expected utilization with policy p given PE_j does attempt to access shared resources if it has demand. Note that $\bar{z}_{ij}(\underline{d}, p) = -\alpha p_i$ after a little algebraic manipulation for $d_i = d_j = 1$ as it should since PE_i attempts to access shared resources

with probability p_i and will suffer a collision for sure if it attempts (because it is given that PE_j will attempt to access shared resources and PE_j interferes with PE_i).

The Jacobian of the utilization operator at \underline{p} can now be stated as

$$\frac{\partial \underline{U}(\underline{p})}{\partial \underline{p}} = \begin{bmatrix} \phi_1(\underline{p}) & \psi_{12}(\underline{p}) & \psi_{13}(\underline{p}) \dots & \psi_{1N}(\underline{p}) \\ \psi_{21}(\underline{p}) & \phi_2(\underline{p}) & \psi_{23}(\underline{p}) \dots & \psi_{2N}(\underline{p}) \\ \psi_{31}(\underline{p}) & \psi_{32}(\underline{p}) & \phi_3(\underline{p}) \dots & \psi_{3N}(\underline{p}) \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{N1}(\underline{p}) & \psi_{N2}(\underline{p}) & \psi_{N3}(\underline{p}) \dots & \phi_N(\underline{p}) \end{bmatrix} .$$

And, the necessary condition for Pareto optimality of an internal policy (that there exist a non-zero linear combination of rows of the Jacobian at \underline{p} that yields a zero vector) can be stated as

$$\exists \underline{c} \mid \underline{c} \neq (0, 0, \dots, 0) \wedge \forall i=1, 2, \dots, N$$

$$c_i \phi_i(\underline{p}) + \sum_{j \in I(i)} c_j Z_{ji}(\underline{p}) = \sum_{j \in I(i)} c_j X_{ji}(\underline{p}) .$$

For convenience, a PE which may interfere with PE_i will be called a neighbor of PE_i , and a PE and its neighbors form a neighborhood. Noticing that \underline{c} can be interpreted as a vector of pay-off coefficients similar to a vector of Lagrangian multipliers, and discounting the case where PE_i has no demand (since then $\phi_i(\underline{p})=0$ and $Z_{ji}(\underline{p})=X_{ji}(\underline{p})$), the following interpretation, the balance principle, is possible:

At a Pareto-optimal policy there exists a non-zero vector of payoff coefficients such that for each PE, expected neighborhood utilization payoff given the PE has demand and attempts to access shared resources equals expected neighborhood utilization payoff given the PE has demand but does not attempt to access shared resources.

This balance principle differs from Yemini's which, for each PE, equates expected silence (empty slots) given the PE has demand with expected utilization by the PE's neighbors given the PE has demand. In the next section the balance principle developed above is shown to hold for a wide range of utilization operators which may include a penalty for collisions and/or sacrifices. Yemini's balance principle is less general as it does not hold for such a wide range of utilization operators.

4.2.1.4 Proof of generality.

It will now be shown that the general balance principle developed in the previous section applies for a broad range of utilization operators. Let $E[\tilde{u}_j | p^{\text{attempt}}(i)]$ be PE_j 's expected utilization with policy p given that PE_i attempts to access shared resources. Similarly, let $E[\tilde{u}_j | p^{\text{sacrifice}}(i)]$ and $E[\tilde{u}_j | p^{\text{idle}}(i)]$ be PE_j 's expected utilization with policy p given that PE_i has demand but does not attempt to access shared resources and PE_i has no demand, respectively.

The local utilization operator for some PE_j can now be written as

$$\bar{u}_j(\underline{p}) = \sum_{\underline{d} \in \{0,1\}^N} \pi(\underline{d}) \cdot \bar{u}_j(\underline{d}, \underline{p}),$$

where

$$\bar{u}_j(\underline{d}, \underline{p}) = \begin{cases} p_i E[\bar{u}_j | \underline{p}^{\text{attempt}}(i)] + (1-p_i) E[\bar{u}_j | \underline{p}^{\text{sacrifice}}(i)] & \text{if } d_i=1 \\ E[\bar{u}_j | \underline{p}^{\text{idle}}(i)] & \text{if } d_i=0. \end{cases}$$

And, providing $E[\bar{u}_j | \underline{p}^{\text{attempt}}(i)]$, $E[\bar{u}_j | \underline{p}^{\text{sacrifice}}(i)]$, and $E[\bar{u}_j | \underline{p}^{\text{idle}}(i)]$ are independent of p_i for all j (including $j=i$)

$$\frac{\partial \bar{u}_j(\underline{p})}{\partial p_i} = \sum_{\underline{d} \in \{0,1\}^N} \pi(\underline{d}) \frac{\partial \bar{u}_j(\underline{p}, \underline{d})}{\partial p_i},$$

where

$$\frac{\partial \bar{u}_j(\underline{d}, \underline{p})}{\partial p_i} = \begin{cases} E[\bar{u}_j | \underline{p}^{\text{attempt}}(i)] - E[\bar{u}_j | \underline{p}^{\text{sacrifice}}(i)] & \text{if } d_i=1 \\ 0 & \text{if } d_i=0. \end{cases}$$

Using the necessary conditions for pareto optimality and discounting the case where $d_i=0$ (since $0=0$), the general balance principle results:

$$\exists \underline{c} \mid \forall i=1,2,\dots,N$$

$$c_i E[\bar{u}_i | \underline{p}^{\text{attempt}}(i)] + \sum_{j \in I(i)} c_j E[\bar{u}_j | \underline{p}^{\text{attempt}}(i)] =$$

$$c_i E[\bar{u}_i | \underline{p}^{\text{sacrifice}}(i)] + \sum_{j \in I(i)} c_j E[\bar{u}_j | \underline{p}^{\text{sacrifice}}(i)].$$

This balance principle is valid for a wide range of utilization processes and operators; however, for some applications a special case of the balance principle is appropriate. This case arises when a PE's utilization of one shared resource is not dependent on the PE gaining exclusive access to other shared resources. When this condition holds, a PE's neighbors' utilization of resources not shared with the PE is independent of the PE's actions, and neighbors' conditional expected utilization payoff of resources shared with the PE may be used in place of neighbors' conditional expected utilization payoff of all shared resources.

4.2.1.5 Direct use of the balance principle.

A direct way of using the balance principle just derived involves manipulating the balance principle equations into expressions for optimal policies given the demand, payoff coefficients, and penalty (if applicable). These expressions are then used with estimates of the current demand, current payoff coefficients, and estimate of the penalty (if applicable) to solve for optimal policies.

To understand how this works, consider the case of two PEs that share a single resource with penalty for collisions. Assuming that the probability that each PE has demand for the shared resource is estimated to be $\underline{\delta} = (\delta_1, \delta_2)$, the balance principle states that

$$\text{for PE}_1: \quad c_1((1-\delta_2p_2)-\alpha\delta_2p_2) + c_2(-\alpha\delta_2p_2) = c_2\delta_2p_2,$$

$$\text{and for PE}_2: c_2((1-\delta_1 p_1) - \alpha \delta_1 p_1) + c_1(-\alpha \delta_1 p_1) = c_1 \delta_1 p_1.$$

Here $c_1((1-\delta_2 p_2) - \alpha \delta_2 p_2) + c_2(-\alpha \delta_2 p_2)$ is PE₁'s expected neighborhood utilization payoff given it has demand and attempts to access the resource because $c_1((1-\delta_2 p_2) - \alpha \delta_2 p_2)$ is PE₁'s expected utilization payoff given that it has demand and attempts to access the resources and $c_2(-\alpha \delta_2 p_2)$ is PE₂'s expected utilization payoff given that PE₁ has demand and attempts to access the resource. Also, $c_2 \delta_2 p_2$ is PE₁'s expected neighborhood utilization payoff given it has demand yet does not attempt to access the resource because PE₁'s expected utilization payoff given that it has demand yet does not access the resource is zero and PE₂'s expected utilization payoff given that PE₁ has demand yet does not access the resource is $c_2 \delta_2 p_2$. The situation for PE₂ is similar.

After some manipulation, the expressions for optimal policies are

$$p_1 = c_1 / (c_1 + c_2) \delta_1 (1 + \alpha)$$

$$\text{and } p_2 = c_2 / (c_1 + c_2) \delta_2 (1 + \alpha).$$

Thus, if $\delta_1 = \delta_2 = 1$ (i.e., PE₁ and PE₂ are both known or believed to be busy) and $\alpha = 1$ (i.e., the penalty for collisions is known or believed to be one) then $c_1 / c_2 = 1$ selects the Pareto-optimal policy $p_1 = p_2 = 1/4$. One can, in fact, prove that when $\delta_1 = \delta_2 = 1$ and $\alpha = 1$ (as is the case shown in Figure 1) all internal Pareto-optimal policies lie on the line $p_2 = -p_1 + 1/2$. To obtain the expected utilization payoffs for PEs one merely plugs the optimal policies into the network utilization operator.

It should be noted that the vector of payoff coefficients, \underline{c} , which may be used to select a particular Pareto-optimal solution does not directly set the relative value of utilization for PEs. For example, if $c_1/c_2=2$ for the example just described and depicted in Figure 3, the policy $p_1=1/6, p_2=1/3$ is selected, for which $u_1/u_2=1/4$. At an internal Pareto-optimal policy the ratio of the coefficients that satisfies the balance principle defines the orientation of the utilization boundary surface, not the ratio of expected utilizations. By fixing the coefficients and then solving for a Pareto-optimal policy, a particular Pareto-optimal policy can be chosen; however, care must be taken to choose a vector of coefficients for which a solution exists.

4.2.2 An adaptive balancing algorithm.

It is possible for the balance principle developed in the previous section to serve as the basis for an adaptive, distributed access control algorithm that is quasi-static (i.e., appropriate for slowly changing environments) and entirely decentralized. The algorithm implements a decentralized, iterative process that estimates and balances PEs' conditional expected neighborhood utilization payoffs (expected neighborhood utilization payoff given the PE has demand and attempts to access shared resources, and expected neighborhood utilization payoff given the PE has demand but sacrifices). While a detailed analysis of the algorithm is beyond the scope of this thesis, a description is included below and proper operation is verified in a

controlled environment (see Appendix A).

Conventional adaptive access control algorithms rely on an estimate of the number of PEs with demand and an optimality condition similar to the balance principle to compute an optimal policy [KLEI78, SEGA76]. In much the same way, an estimate of the number of PEs with demand can be used in conjunction with the balance principle derived in Subsection 4.2.1 to compute optimal policies. Estimates of the number of PEs with demand can be maintained by heuristic means [KLEI78] or by recursive estimation [SEGA76]. These schemes are limited to the case where there is a single resource shared by all PEs.

The alternative adaptive access control scheme proposed below incrementally adjusts policies based on estimated conditional expected utilization payoffs. The estimates of conditional expected utilization payoffs are maintained through observations and/or communicated information concerning PEs' actual utilizations, and may be supplemented by inferences of PEs' conditional expected utilizations. Note that estimates of PEs' demand need not be maintained, and a precise model of the utilization process need not be known if actual utilizations are observed or communicated. Furthermore, the algorithm is appropriate for arbitrary network topologies.

4.2.2.1 The underlying concepts.

For reasons soon to be explained, it is actually best that each PE maintain separate estimates of conditional expected utilization payoffs for itself and each of its neighbors. This involves obtaining neighbors' utilizations via communication each iteration a PE has demand unless neighbors' utilization of resources is observable, some approximation is made, or advantage is taken of some special property. With a utilization processes for which the special case of the balance principle discussed briefly at the end of section 4 holds, only a PE's neighbors' utilization of resources shared with the PE are needed to update the PE's estimates; this can be obtained without communication if utilization of resources to which a PE has access is observable by the PE.

The payoff coefficients, \underline{c} , determine the particular Pareto-optimal policy and utilization that is obtained when balance is achieved. Thus, the adaptive balancing algorithm does not strive to find some random Pareto-optimum, it strives to find the particular Pareto-optimum selected by the coefficients, \underline{c} . These coefficients may be preset or manipulated, via a central processor, a hierarchy, or a second level of adaptation, to distribute utilization among PEs of the network. Unfortunately, for irregular networks $\underline{c}=(1,1,\dots,1)$ does not give equal utilization to all PEs. Although a PE need not keep separate estimates because of this, a PE must distinguish between neighbors when observing utilizations if $\underline{c}=(1,1,\dots,1)$ so that the utilization payoffs can be

multiplied by the appropriate payoff coefficients.

Given a particular vector of payoff coefficients, \underline{c} , let

$$E[nup_i | p^{\wedge} \text{attempt}(i)] =$$

$$c_i E[\tilde{u}_j | p^{\wedge} \text{attempt}(i)] + \sum_{j \in I(i)} c_j E[\tilde{u}_j | p^{\wedge} \text{attempt}(i)]$$

be PE_i 's estimate of its expected neighborhood utilization payoff given PE_i attempts to access shared resources; and, let

$$E[nup_i | p^{\wedge} \text{sacrifice}(i)] =$$

$$c_i E[\tilde{u}_j | p^{\wedge} \text{sacrifice}(i)] + \sum_{j \in I(i)} c_j E[\tilde{u}_j | p^{\wedge} \text{sacrifice}(i)]$$

be PE_i 's estimate of its expected neighborhood utilization payoff given PE_i sacrifices. Furthermore, assuming a probabilistic scheme, let

$$E[up_j | p^{\wedge} \text{demand}(i)] =$$

$$p_i c_j E[\tilde{u}_j | p^{\wedge} \text{attempt}(i)] + (1-p_i) c_j E[\tilde{u}_j | p^{\wedge} \text{sacrifice}(i)]$$

be PE_i 's estimate of a neighbor's expected utilization payoff (given PE_i has demand).

Now, if $E[nup_i | p^{\wedge} \text{attempt}(i)] < E[nup_i | p^{\wedge} \text{sacrifice}(i)]$ then PE_i knows that its neighbors are over-utilizing shared resources and are interfering with its attempts to access shared resources. For this reason, PE_i should notify its neighbors to decrease their attempts to access shared resources. It is important to note that changes in PE_i 's policy will not directly affect the imbalance seen by PE_i , but changes to neighbors' policies will. For this reason, changes to a PE's policy are, for the most part, "neighbor-directed" rather than self-directed.

If $E[n_{p_i|p}^{\text{attempt}(i)}] > E[n_{p_i|p}^{\text{sacrifice}(i)}]$ then PE_i knows that its neighbors are under-utilizing shared resources and are wasting PE_i 's sacrifices. It is tempting to say that PE_i should notify neighbors with low $E[u_{j|p}^{\text{demand}(i)}]$ to increase their attempts to access shared resources; however, under-utilization of shared resources by neighbors may be due to neighbors' attempting too much (and colliding) as well as attempting too little.

Two approaches to resolving this uncertainty are for a PE to try to determine if neighbors are under-utilizing because they are attempting too much or too little, or for a PE to let the neighbors determine that. For a PE to determine why neighbors are under-utilizing shared resources it might look at its own expected utilization payoff given it attempts to access shared resources; if this is low then neighbors are attempting too much, and if high then neighbors are attempting too little. For a neighbor to determine if it is attempting too much or too little, the neighbor need only look at its policy or note whether or not it just collided or sacrificed; if its policy is high or it just collided then it is attempting too much, and if its policy is low or it just sacrificed then it is attempting too little.

To distribute utilization quickly in the manner specified by the payoff coefficients it is important that PE_i direct stronger notifications to increase/decrease attempts to access shared resources to neighbors with low/high $E[u_{j|p}^{\text{demand}(i)}]$. This is why separate estimates of conditional expected utilization payoffs for PE_i and its neighbors should be kept. Note that if $c_i \neq c_j$ for some neighbor, PE_j ,

then PE_i must distinguish between different neighbors' utilization of resources anyway.

In general a PE should readjust its likelihood of attempting to access given demand in accordance with notifications received from neighbors, by changing its policy. Because of the problem mentioned in the previous paragraph, though, a PE might be required to decrease its attempts to access if it has just collided or its policy is high, and otherwise readjust its attempts to access according to notifications received from neighbors. PEs' policies are bounded, so a PE cannot increase its policy beyond 1.0 or decrease it beyond 0.0 for the probabilistic scheme; the upper and lower limits for the urn scheme are N and 0.0, respectively.

The adaptive balancing algorithm used throughout this thesis can be described by the following rules:

- 1) If a PE's expected neighborhood utilization payoff given it attempts to access resources is greater than its expected neighborhood utilization payoff given it sacrifices, then it should notify neighbors with minimal estimated utilization payoff to increase their policies by two step-sizes, and should notify other neighbors to increase their policies by one step-size;

- 2) If a PE's expected neighborhood utilization payoff given it attempts to access resources is less than its expected neighborhood utilization payoff given it sacrifices, then it should notify neighbors with maximal estimated utilization payoff to increase their policies by two step-sizes, and should notify other neighbors to increase their policies by one step-size;

- 3) Each PE lowers its policy by one step-size if it suffers a collision; otherwise, it increases its policy by an amount determined by summing notifications received from neighbors and normalizing by dividing by the number of neighbors. Policies are only permitted to take on values between 0.0 and 1.0 (inclusive) for the probabilistic scheme,

or between 0.0 and N (inclusive) where N is the number of PEs for the urn scheme.

A step-size specifies the basic magnitude of changes to policies, thus controlling the reactivity of the algorithm.

Remember that the balance principle represents a necessary but not sufficient condition for Pareto optimality of an internal policy. For example, the rude policy, $p=(1,1,\dots,1)$, satisfies the balance principle if $a=0$ and demand is high, yet is not Pareto-optimal. This does not, however, appear to present a problem for the adaptive balancing algorithm just described, which will not remain at the rude policy if there are any collisions. Other non-pareto optimal equilibria have not been encountered.

4.2.2.2 Discussion.

A balance principle for optimal access control has been derived which covers a wide range of utilization processes including the case where a penalty is incurred for collisions. This was necessary because the model of utilization for the iterative refinement application differs from the model of utilization for the packet radio broadcast application, and a simultaneous update in an iterative refinement algorithm often has a negative effect.

An adaptive balancing algorithm based on observed utilizations is especially useful when knowledge of the demand and penalty processes is unknown. For the iterative refinement application the penalty can be hard to determine because the effect of a collision (simultaneous update among interacting neighbors) depends on characteristics of the search space and the location in the search space of the control vector before the update.

Communication is kept to a minimum with the adaptive balancing algorithm by inferring the limited utilization information required for the adaptive balancing algorithm from the required exchange of current solution components between neighbors each iteration. Notifications are the only messages communicated; they are small messages and are never sent beyond neighbors.

The adaptive balancing algorithm developed above has been tested in a controlled environment. These results are included in Appendix A. The adaptive balancing algorithm is also used in two iterative refinement update control schemes used extensively in experiments discussed in later chapters. These update control schemes are described in detail next.

4.3 Eight Update Control Schemes

Of the eight update control schemes discussed in this section, six are parallel schemes that are appropriate for the distributed iterative refinement technique. These update control schemes are the pure parallel, alternating, attenuated, probabilistic, urn, and negotiated update control schemes. The probabilistic and urn schemes employ the adaptive balancing algorithm developed in Section 4.2. The other two, the serial and SIGOP schemes, are sequential schemes included for comparison. Many of these schemes are based on the fundamental access control schemes described above.

The Serial Scheme. The serial update control scheme gives PEs the right to update one at a time and in a fixed order. Below is a serial update control scheme for PE_i.

```
(* SERIAL UPDATE CONTROL SCHEME: *)
(* PEs are numbered 1 through N *)
(* PEi will be allowed to update on iteration i and *)
(* every N iterations thereafter *)
(* it_count is current iteration number *)

if (((it_count-1) mod N)+1) = i
  then PEi is allowed to update;
```

The particular order is generally taken to be unimportant with the serial scheme, but experience has shown the order to be, in fact, very important and most effective when taken along a spanning tree of the network. This ordering is generally established at system creation.

The SIGOP Scheme. The SIGOP update control scheme is a serial scheme which employs some specialized heuristics. These heuristics were found to be important for an existing centralized network traffic light control program, SIGOP II, which is discussed in detail later in Chapter 6. The first heuristic involves the use of a maximal spanning tree based on link importance to provide an global ordering for updates. Another part of this heuristic is to reverse the sequence after each "sweep" through the network. The second heuristic is employed in the calculation phase during the first sweep through the network that serves to prime the control settings. This second heuristic is to include only links connected to processing elements with primed controls in the calculation of another processing element's prime.

Below is a SIGOP update control scheme for PE_1 . The modifications to the calculation phase that are part of the SIGOP scheme are not shown.

```
(* SIGOP UPDATE CONTROL SCHEME: *)
(* PEs are numbered 1 through N *)
(* PEi will be allowed to update once every N iterations *)
(* MST[1], MST[2],... MST[N] defines an ordering *)
(* of PEs along a maximal spanning tree *)
(* it_count is current iteration number *)

if (it_count mod (2*N)) < N
  then begin (* during forward sweep *)
    if i = MST[((it_count-1) mod N)+1]
      then PEi is allowed to update
    end
  else begin (* during backward sweep *)
    if i = MST[N-((it_count-1) mod N)]
      then PEi is allowed to update
    end;
end;
```

The maximal spanning tree ordering should be established during the

initialization phase of the distributed iterative refinement algorithm.

The Parallel Scheme. The (pure) parallel update control scheme allows unrestricted, lock-step parallel updates to take place. Below is a parallel update control scheme for PE_i .

```
(* PARALLEL UPDATE CONTROL SCHEME: *)
(* all PEs are allowed to update *)
```

```
PEi is allowed to update;
```

Since many simple examples can be generated which demonstrate how oscillation often results from not controlling updates, this scheme should not be used unless demand is extremely low.

The Alternating Scheme. The alternating update control scheme cycles among pre-specified groups of PEs, giving each PE in the currently chosen group the right to update. PEs are assigned to groups such that each PE is a member of only a single group, no neighboring PE is a member of the same group, and the fewest groups possible are formed. An alternating update control scheme for PE_i is given below.

```
(* ALTERNATING UPDATE CONTROL SCHEME: *)
(* GROUP[i] indicates to which one group (numbered 1 *)
(* to number_of_groups) PEi belongs; these groups *)
(* are arranged such that no PE in a group requires *)
(* another group member's control for its *)
(* calculation phase *)
(* it_count is current iteration number *)
```

```
if GROUP[i] = (((it_count-1) mod number_of_groups)+1)
  then PEi is allowed to update;
```

This group membership can be established at system creation or during the initialization phase of the distributed iterative refinement algorithm.

The Negotiated Scheme. The negotiated control scheme requires communication before each time-slot and allows only a subset of PEs which do not compete to access resources. A negotiated update control scheme is given below for PE_i .

```
(* NEGOTIATED UPDATE CONTROL SCHEME: *)
(* PEi is allowed to update if it has higher priority *)
(* than any other PE which needs PEi's control for *)
(* its calculation phase and wishes to update its *)
(* control *)
(* priority(i) is a function that returns the priority of *)
(* PEi, ((GROUP[i]+it_count) mod number_of_groups) *)
(* GROUP[i] indicates to which one group (numbered 1 to *)
(* number_of_groups) PEi belongs; these groups are *)
(* arranged such that no PE in a group requires another *)
(* group member's control for its calculation phase *)
(* demand[i] was set during the calculation phase to *)
(* indicate if PEi wants to alter its control *)
```

```
high_priority := priority(i);
for j := 1 to N
  do if i ≠ j and PEi needs PEj's control for its calculation phase
    then if priority(j) > high_priority
      then begin
        get demand[j] from PEj;
        if demand[j]
          then high_priority := priority(j)
        end;
  if priority(i) = high_priority
    then PEi is allowed to update;
```

The negotiated scheme described above and employed in later experiments uses an alternating PE ordering to implement a rotating priority mechanism for resolving conflicts in a fair manner. This ordering can be established at system creation or during the initialization phase of the distributed iterative refinement algorithm.

The Attenuated Scheme. The attenuated update control scheme allows unrestricted parallel updates, but restricts the magnitude of changes to control variables. An attenuated update control scheme is given below for PE_i .

```
(* ATTENUATED UPDATE CONTROL SCHEME: *)
(* all PEs are allowed to change their controls, *)
(* but only by a small amount *)
```

PE_i is allowed to change its control by a small amount;

In terms of access control, perhaps this would be viewed as partial utilization of some kind - it is not clear; the attenuated scheme does not fit the model of utilization previously discussed. In later experiments controls are integers between 0 and 39, inclusive, and the attenuated update control scheme allows a change of 1 if a change of 5 or less is desired, else it allows a change of 5. The use of two step-sizes increases the speed of convergence.

The Probabilistic Scheme. A policy, p_i , is used by each PE_i in the probabilistic control scheme to determine if PE_i should have the right to update for a given time-slot. The policy is a probability value, and represents the probability that PE_i is given the right to update given that PE_i has demand. Below is a probabilistic update control scheme for PE_i .

```
(* PROBABILISTIC UPDATE CONTROL SCHEME: *)
(* PEi is allowed to update with probability POLICY[i]; *)
(* this probability is maintained by an adaptive *)
(* balancing algorithm *)
(* random is a random real number greater than or equal *)
(* to 0 but less than 1 *)
```

```
if random >= POLICY[i]
```

then PE_i is allowed to update;

The policies are adapted over time using the adaptive balancing algorithm developed in Section 4.2. Payoff coefficients of 1 are used in all experiments, as well as a step-size of 0.05 and window-size of 2 for the adaptive balancing algorithm.

The Urn Scheme. Under the urn scheme, each PE_i draws k_i numbers from identical pseudo-random number generators using a common seed to determine which PEs may attempt utilization of shared resources. An urn update control scheme for PE_i is given below.

```
(* URN UPDATE CONTROL SCHEME: *)
(* PEi is allowed to update if GROUP[i] is among the *)
(* first POLICY[i] numbers in array URN; POLICY[i] *)
(* is a number between 0 and number_of_groups maintained *)
(* by an adaptive balancing algorithm, and the values in *)
(* array URN are determined each iteration by a *)
(* pseudo-random number generator and seed common *)
(* to all PEs *)
(* GROUP[i] indicates to which one group (numbered 1 to *)
(* number_of_groups) PEi belongs; these groups are *)
(* arranged such that no PE in a group requires another *)
(* group member's control for its calculation phase *)
(* random is a random real number greater than or equal *)
(* to 0 but less than 1 *)

for k := 1 to number_of_groups
  do URN[k] := k;
for k := 1 to number_of_groups-1
  do begin
    l := trunc(random * (number_of_groups - k + 1)) + k;
    URN[k] := URN[l];
    URN[l] := k;
  end;
found := false;
for k := 1 to POLICY[i]
  do if GROUP[i] = URN[k]
    then found := true;
if found
  then PEi is allowed to update;
```

Like the probabilistic update control scheme, the policies are adapted

over time using the adaptive balancing algorithm developed in Section 4.2. Payoff coefficients of 1 are used in all experiments, as well as a step-size of 0.5 and window-size of 2 for the adaptive balancing algorithm.

These eight update control schemes are tested in experiments described in Chapters 5 and 8. Five of these update control schemes, the alternating, negotiated, attenuated, probabilistic, and urn schemes are decentralized schemes which allow parallel updating, although three of these schemes, the alternating, negotiated, and urn schemes, employ global processing element orderings.

C H A P T E R V
EXPERIMENTS WITH LOCALIZED INTERACTION

The objective of the set of experiments discussed in this chapter is to study the behavior of the basic distributed iterative refinement algorithm on a simple application where interaction is localized (i.e., only neighboring processing elements' subproblems interact). For such an application, the simultaneous-update problem is the principle difficulty faced by the distributed iterative refinement algorithm; this is because a local view is sufficient to predict the effects of changing a control. Thus, the focus for these experiments is on the performance of the update control schemes.

The experiments discussed in this chapter concern a simple numbering problem which is described below. Initial experiments involve a ring of 12 processing elements. First the gross behavior of the distributed iterative refinement algorithm with various update control schemes is described. The sensitivity of the distributed iterative refinement algorithm to network size and topology is then evaluated through a set of experiments involving rings of various sizes, a grid network, and two other networks with many connections.

5.1 A Simple Numbering Problem

The numbering problem described below is a simple problem which can be used to test different update control schemes. In this problem a network of nodes are to be assigned integers in the range $[0, C-1]$. Nodes are connected by directed links, and the assignment of numbers is to be one such that the sum of the squares of the difference between certain pre-specified offsets and obtained offsets along all links is minimized; an offset is the difference between two nodes' assigned numbers, taken in a pre-specified direction.¹¹

In the distributed version of this problem, a processing element (PE) is associated with each node and has the responsibility for determining what number should be assigned to its node. A PE can communicate only with its neighbors, PEs whose nodes are connected to the PE's node by a link. The overall objective of the network of processing elements is to determine an appropriate numbering in the least time.

As an example, consider a ring of 4 PEs as shown in Figure 4. Assume that $C=40$, the initial numbering is 0 for all PEs, and the pre-specified offsets are 10 between PE_1 and PE_2 , 15 between PE_2 and PE_3 , 15 between PE_3 and PE_4 , and 20 between PE_4 and PE_1 . Since the initial numbering is 0 for all PEs, the actual offsets between PEs is

11. A cyclic numbering system is assumed for this problem, so most differences must be carried out modulo C . However, the difference between two offsets, ϕ and ϕ' , is the minimum of $((\phi+C-\phi') \bmod C)$ and $((\phi'+C-\phi) \bmod C)$.

An example of the numbering problem used to test update control schemes is illustrated. Four processing elements must solve for numbers such that the difference between actual and pre-specified offsets is minimized, using a distributed iterative refinement algorithm. If a pure parallel update control scheme is employed, all PEs are allowed to update each iteration and the algorithm oscillates between two poor solutions. If, however, only the first and third PEs are allowed to update the algorithm converges immediately to a reasonable solution. This demonstrates the need for update control schemes other than the pure parallel update control scheme.

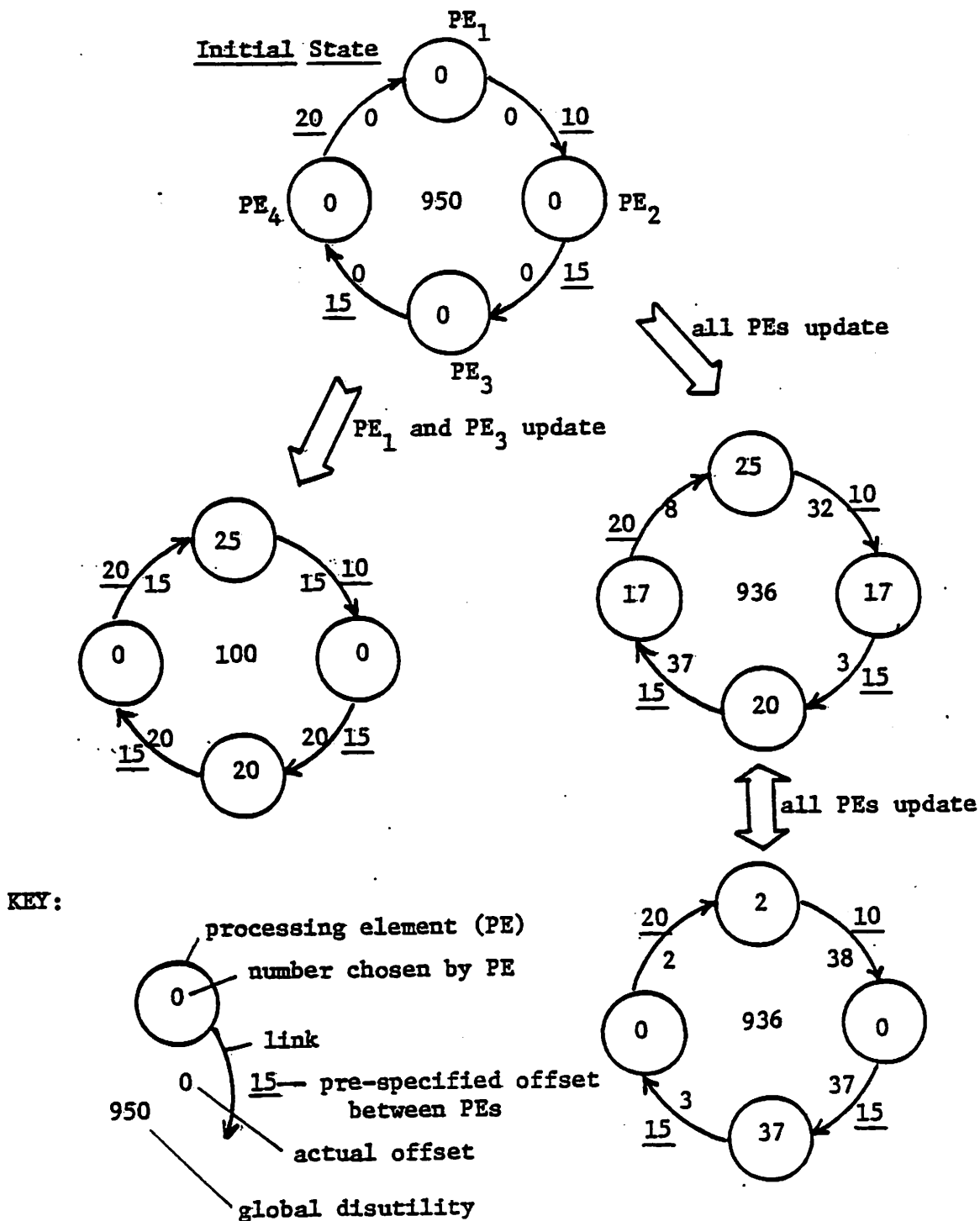


Figure 4: An Example of the Numbering Problem.

initially 0.

To compute the initial global disutility, the sum of the squares of the differences between pre-specified offsets and actual offsets. The difference between the pre-specified offset between PE_1 and PE_2 of 10 and the actual offset between PE_1 and PE_2 of 0 is 10. For the link between PE_2 and PE_3 and between PE_3 and PE_4 this difference is 15, and for the link between PE_4 and PE_1 this difference is 20. Thus, the initial global disutility is $10^2+15^2+15^2+20^2$ or 950 units. Notice how interaction is localized for the numbering problem (i.e., a PE's choice of number only affects that part of the global objective function associated with links connected to the PE's node).

Processing elements execute the basic distributed iterative refinement algorithm described in Chapter 3. Processing elements are synchronized (time is slotted) and start with an initial numbering called a prime. At the beginning of each iteration (time-slot) processing elements obtains neighbors' numbers. The processing elements then use their neighbors' numbers and their local objective functions to determine if it is desirable to change (update) their numbers. At the end of each iteration, the processing elements use one of the update control schemes described at the end of Chapter 4 to determine which processing elements are allowed to change their numbers. Convergence is detected globally for these experiments.

The results of some iterations of the basic iterative refinement algorithm are depicted in 4. From the initial state oscillation sets in quickly if all PEs update their numbers simultaneously. On the other hand, if PE_1 and PE_3 only are permitted to update their numbers, a solution is obtained with a global disutility of 100. Thus, the need for update control is illustrated.

5.2 Initial Numbering Problem Experiments

Initial numbering problem experiments were carried out on a ring network of 12 PEs. Thus, each processing element has two neighbors. A variety of update control schemes are tested. These update control schemes include the serial, pure parallel, alternating, attenuated, adaptive probabilistic, adaptive urn, negotiated, and SIGOP schemes which were described in Section 4.3.

The overall behavior, speed of convergence, quality of solutions, and sensitivity to prime are discussed below. For the probabilistic and urn control schemes, results of sixty runs were averaged together (twenty primes for each of three random number generator seeds). For the other, deterministic control schemes, results of twenty runs (with the same twenty primes) were averaged together. Statistics on the effectiveness of the update control schemes are also given and communication requirements are discussed.

5.2.1 Overall behavior.

The overall behavior of 12 PEs in a ring network using the distributed iterative refinement algorithm with various update control schemes to solve the numbering problem is depicted in Figure 5. The dimensions shown are average global disutility and iteration number. The average global disutility of a prime is approximately 1600; the average global disutility of a solution is approximately 100.

Reasonable performance was obtained with four of the non-serial update control schemes: the negotiated scheme, the alternating scheme, the urn scheme, and the probabilistic scheme. Convergence for these schemes was exponential, and the quality of solutions found by the distributed iterative refinement algorithm employing these schemes was good.

The iterative refinement algorithm with SIGOP update control scheme found the best solutions, but was slower than the negotiated, alternating, urn, and probabilistic schemes. Since the SIGOP scheme is serial, convergence is linear during each sweep, but the slope decreases abruptly after the first sweep because SIGOP's priming heuristics employed during the first sweep are so effective that there is little room for improvement on subsequent sweeps. A serial scheme without special heuristics (S in later figures) was even slower.

The average global disutility over time of 20 runs on the numbering problem is shown for the distributed iterative refinement algorithm with the pure parallel, SIGOP, serial, probabilistic, attenuated, alternating, urn, and negotiated update control schemes (P, SIGOP, S, PROB, ATT, ALT, URN, and N, respectively). A ring network with 12 processing elements was used. Exponential convergence is observed with the negotiated, urn, alternating, attenuated, and probabilistic schemes. Convergence with the SIGOP or serial schemes is piece-wise linear; full convergence takes more than the 18 iterations shown above. The pure parallel scheme results in an algorithm that does not converge.

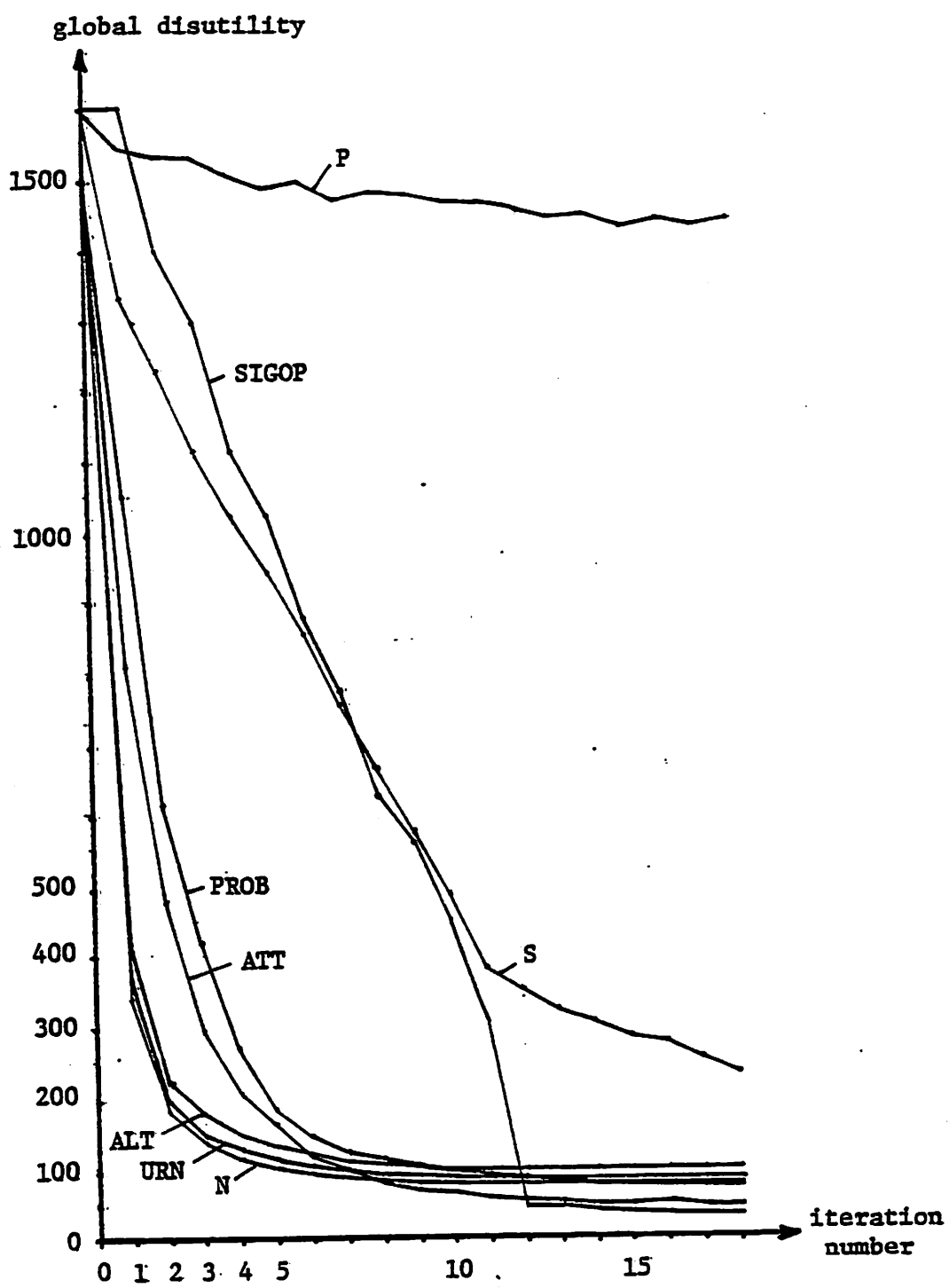


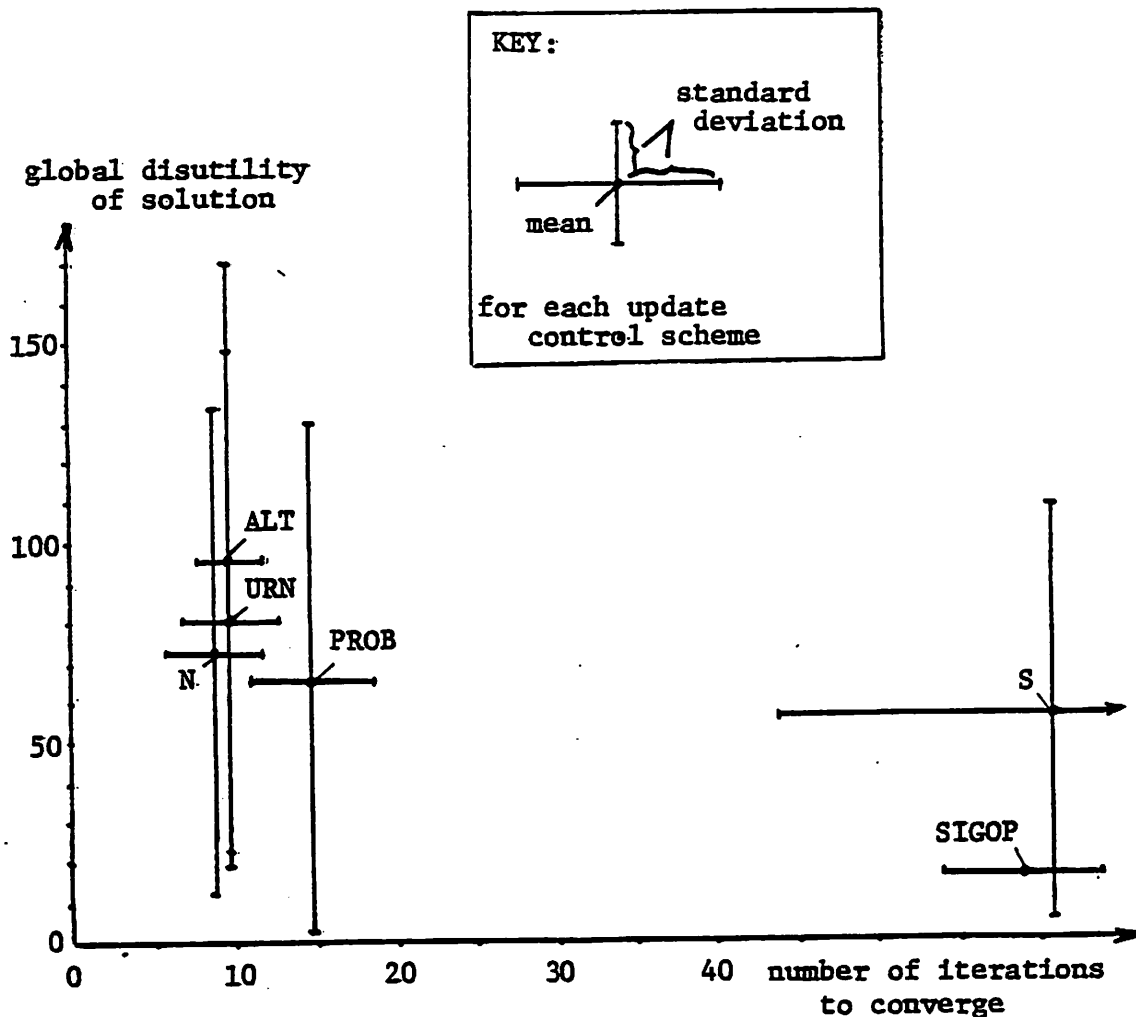
Figure 5: Global Disutility vs. Iteration for RING of 12 PEs.

The iterative refinement algorithm with attenuated update control scheme did not always converge, but often found very good solutions quickly. When oscillation set in, it spanned many iterations; convergence was often erratic. Finally, oscillation always resulted when the pure parallel update control scheme was used. Little improvement was generally obtained over the prime with the pure parallel scheme.

5.2.2 Speed of convergence and quality of solutions.

Speed of convergence and quality of solutions found of the distributed iterative refinement algorithm with various update control schemes is depicted in Figure 6. Each iteration of the algorithm takes the same amount of time to execute, regardless of the particular update control scheme used. The mean and standard deviation of the number of iterations to fully converge and the quality of solutions found is shown for each control scheme. The algorithm with negotiated, alternating, urn, and probabilistic schemes took 9, 10, 13, and 15 iterations, respectively, to fully converge. With the SIGOP update control scheme, on the other hand, took 59 iterations on average to fully converge.

Sensitivity to the quality of the prime is not shown. In general, convergence is faster with a good prime (i.e., a prime that is close to a solution). The exceptions to this occurred with the SIGOP update control scheme, which generates its own prime and thus was insensitive to the given primes, and the pure parallel scheme which resulted in



The average speed of convergence and quality of solutions of 20 runs on the numbering problem is shown for the iterative refinement algorithm with the negotiated, urn, alternating, probabilistic, SIGOP, and serial update control schemes. A ring network of 12 processing elements was used to obtain these results. Standard deviations are also shown. The use of the negotiated, urn, alternating, and probabilistic schemes clearly result in a faster algorithm than use of the SIGOP or serial schemes, although use of the SIGOP scheme produced the best solutions.

Figure 6: Speed of Convergence and Quality of Solutions.

oscillation even when started with a good prime.

5.2.3 Update control statistics.

Update control statistics for the eight update control schemes are presented in Table 1. The statistics shown include the percent of iterations a PE has demand, the percent of iterations a PE with demand is successful, is involved in a collision, or sacrifices, and the

UPDATE CONTROL	DEMAND	FREQUENCY OF EVENTS GIVEN DEMAND			AVERAGE AND STAN. DEV. MAGNITUDE OF UTILIZATION	
		success	collision	sacrifice	success	collision
S	46%	11%	0%	89%	20+44	NA
SIGOP	27%	9%	0%	92%	49+91	NA
P	95%	0%	100%	0%	13+10	0+15
ATT	47%	5%	95%	0%	2+ 2	2+11
ALT	33%	87%	0%	13%	22+55	NA
N	33%	84%	0%	16%	26+63	NA
PROB	43%	32%	33%	36%	12+34	18+43
URN	31%	79%	2%	19%	25+60	3+ 5

Update control statistics for the serial, SIGOP, pure parallel, attenuated, alternating, negotiated, probabilistic, and urn update control schemes (S, SIGOP, P, ATT, ALT, N, PROB, and URN, respectively) used to control simultaneous updates in an iterative refinement algorithm are shown. Included are the percent of iterations a processing element has demand, the percent of iterations a processing element is successful, is involved in a collision, or sacrifices given it has demand, and the average magnitude and standard deviation of utilizations for successes and collisions. NA indicates that the statistic is not applicable.

Table 1: Statistics on Update Control for Ring Numbering Problem.

average magnitude and standard deviation of utilizations (decreases in disutility) for successes and collisions.

The serial and SIGOP update control schemes both have low success rates and high sacrifice rates as one would expect from a sequential scheme. A processing element often has demand (i.e., wishes to update), but must often wait for the right to update. Because the first N iterations are more effective with the SIGOP scheme due to its heuristics, processing elements have less demand and the magnitude of utilization given a success is significantly higher.

The pure parallel update control scheme produces many collisions and almost no successes. There are no sacrifices and demand is high. This is expected, the fact that the average utilization for a PE is nearly zero is indicative of oscillation.

When the negotiated and alternating update control schemes are used, processing elements have moderate success rates given they had demand (84-87%), and low sacrifice rates. This indicates that the schemes work very well (i.e., PEs are likely to be allowed to update whenever they wish to). PEs wish to update approximately a third of all iterations, and PEs' demand processes seem to fall into a nice alternating pattern which matches well with either update control scheme. The zero collision rates are expected since these schemes do not permit collisions.

The probabilistic update control scheme has a lower success rate than the negotiated and alternating schemes, and has a moderate collision rate; however, the effect of collisions is highly beneficial for the probabilistic scheme! It seems that a property of the update control application is that collisions involving many PEs are generally more harmful than collisions involving few PEs. The probabilistic scheme limits simultaneous updates and the likelihood that many PEs will collide; this reduces both the frequency of collisions and the ill effects of collisions.

The urn update control scheme performs much like the alternating scheme at first when demand is high. When demand is low, however, PEs are given the right to update more often than is called for with the alternating scheme. This results in an occasional collision, but there is a speed-up over the alternating scheme since PEs need not always wait their turn.

Finally, the attenuated scheme has a very high collision rate like the pure parallel scheme, but attenuation limits the magnitude rather than the frequency of error thereby reducing uncertainty. This is reflected in the low but positive magnitude of utilizations for success and collisions.

Histograms showing the results of attempted updates are included in figure 7. The first histogram shows the distribution of the magnitude of utilization (decrease in disutility) for collisions when the pure parallel scheme is used. With a pure parallel update control scheme, the iterative refinement algorithm suffers many collisions and

Histograms of utilizations encountered with the pure parallel, negotiated, and probabilistic update control schemes which are employed in distributed iterative refinement algorithms is illustrated. Oscillation results when the pure parallel update control scheme is used; thus, utilizations, which are positive when collisions do not result in an increase in disutility and negative when they do result in an increase in disutility, are centered around zero. The negotiated update control scheme permits only successes, which always result in positive utilizations. However, the probabilistic update control scheme limits but does not eliminate collisions. When successes with the probabilistic scheme occur, utilizations similar to those observed with the negotiated scheme result; when collisions occur some negative utilizations result, but more positive than negative utilizations result because collisions involving many processing elements that are more harmful are unlikely.

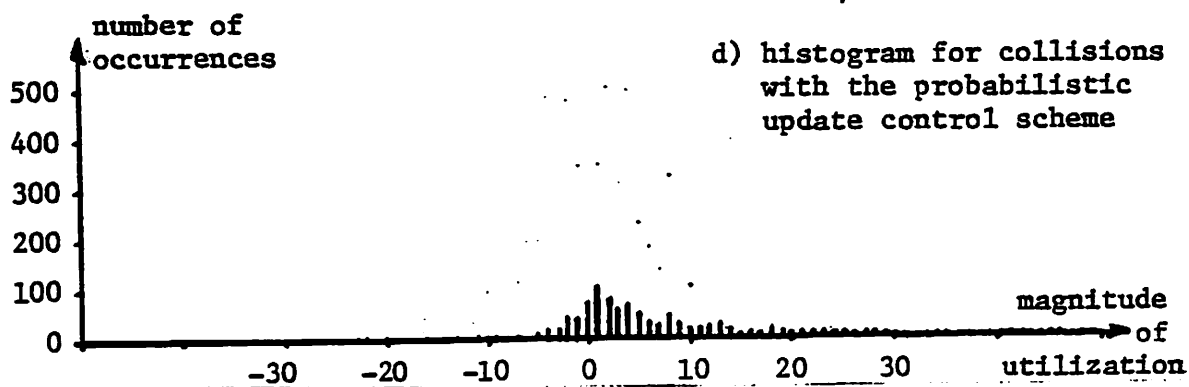
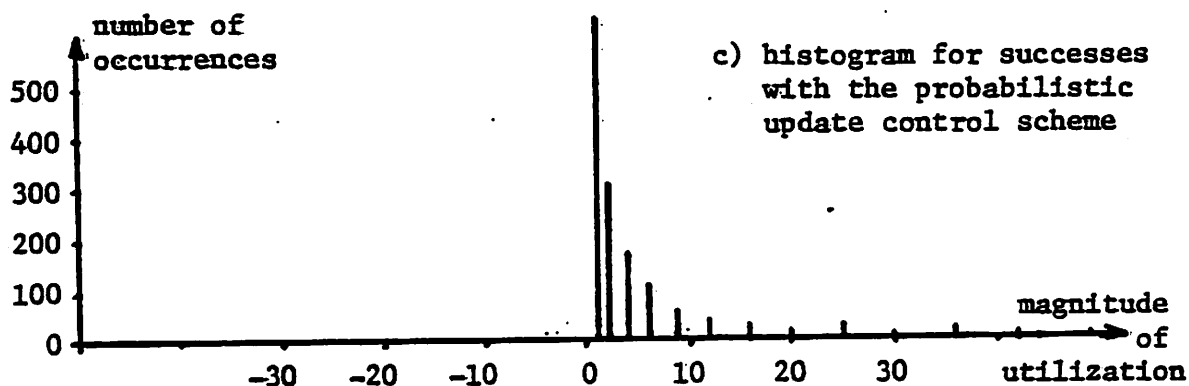
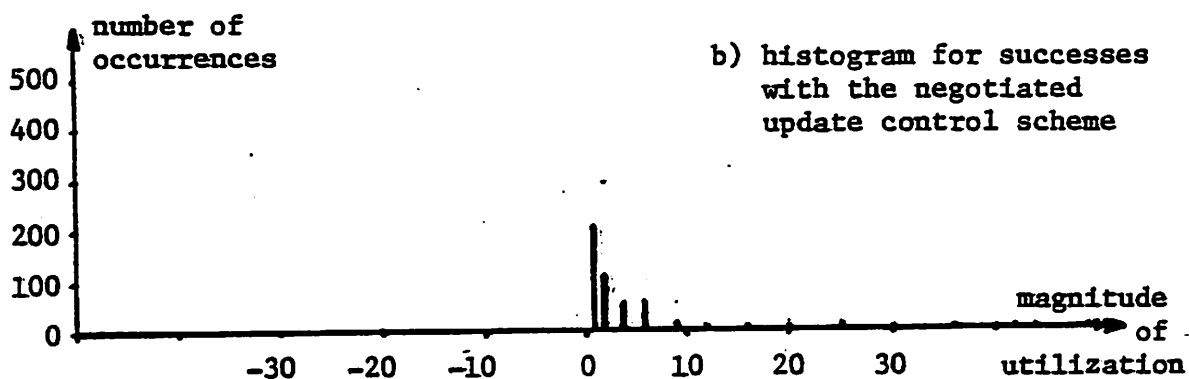
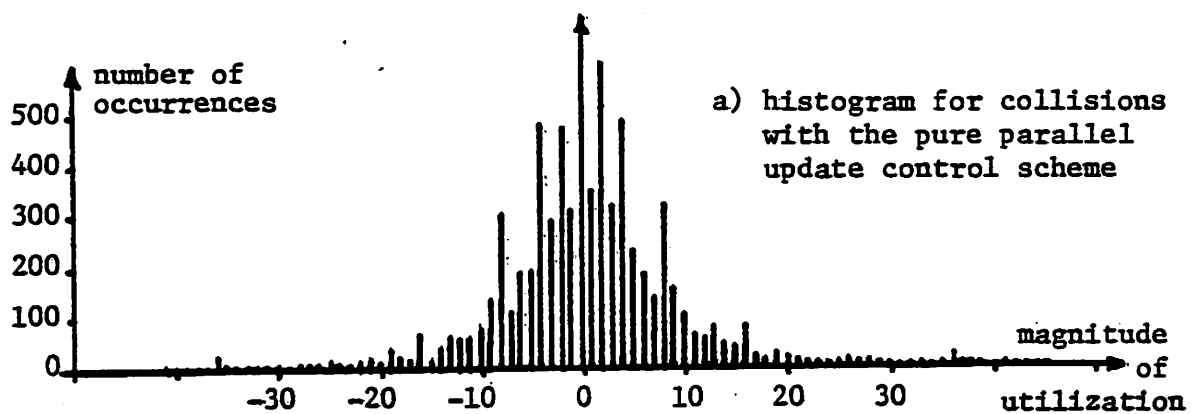


Figure 7: Histograms of Utilizations with some Update Control Schemes.

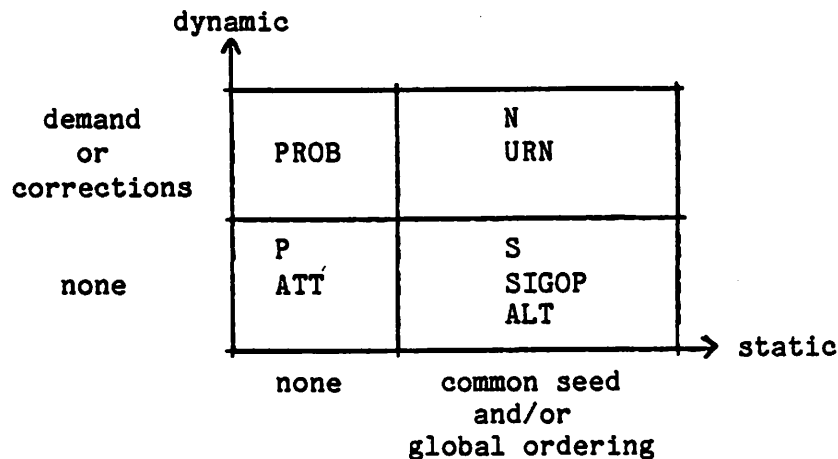
oscillates. The roughly gaussian distribution of the effects of collisions with mean near zero is indicative of this oscillation.

The second histogram shows the distribution of the magnitude of utilization for successes when the negotiated scheme is used. All utilizations are positive because there are no simultaneous updates between interacting subproblems, and small utilizations (1-6 units of disutility) are more likely than large utilizations. The third histogram shows the distribution of the magnitude of utilization for successes when the probabilistic scheme is used. Again, all utilizations are positive (because utilizations from collisions are not included). Note the similarity between this and the previous histogram.

The final histogram shows the distribution of the magnitude of utilization for collisions when the probabilistic scheme is used. The interesting thing here is that this distribution is not like that of the magnitude of utilization for collisions when the pure parallel scheme is used. Rather, this distribution has a surprisingly high mean (approximately 18 units of disutility). Again, the difference between collisions encountered with the probabilistic scheme and the pure parallel scheme is that collisions involving many PEs are generally worse than collisions involving few PEs, and the collisions encountered with the probabilistic scheme involved fewer PEs than collisions encountered with the pure parallel scheme.

5.2.4 Communication requirements.

It is assumed in this thesis that communication between neighbors is fast and inexpensive, whereas communication between non-neighbors is expensive and slow. For the numbering problem, numbers must be communicated each iteration during the acquisition phase. In addition to this, some of the update control schemes require communication. These additional communication requirements are shown in Figure 8.



The communication requirements of the update control schemes employed by a distributed iterative refinement algorithm are presented. These requirements can be static (i.e., to be fulfilled once, before the update control schemes are used) or dynamic (i.e., to be fulfilled each iteration). The pure parallel and attenuated schemes require no communication, while the negotiated and urn schemes require both static and dynamic communication. Update control schemes which require communication generally perform better than those that do not.

Figure 8: Communication Requirements of Update Control Schemes.

Two categories of communication requirements are depicted: static and dynamic. Static communication requirements are fulfilled once, when the network of PEs is set up. Static communication requirements include communication necessary to set up a global ordering (e.g., for S, N, ALT, URN, and SIGOP) and/or to distribute a common seed (e.g., for URN). Dynamic communication requirements are fulfilled each iteration. Dynamic communication requirements include communication necessary to obtain the demand of neighboring PEs before each iteration (e.g., for N) or to send periodic corrections of neighbors' policies (e.g., for PROB and URN).

The pure parallel and attenuated update control schemes require no communication to operate. Oscillation always resulted with the pure parallel scheme, but only occasionally resulted with the attenuated scheme. A probabilistic scheme with a fixed policy would also have no communication requirements and would be effective if a reasonable policy is chosen, but would not be as effective as the (adaptive) probabilistic scheme (PROB) which utilizes communication. A fixed probabilistic component added to the attenuated scheme might make a nice hybrid scheme.

The (adaptive) probabilistic scheme does not rely on a numbering for PEs or a common seed, and hence has no static communication requirements. Corrections to neighboring PEs' policies, however, must be communicated periodically. For best results, these corrections should be communicated at the end of each iteration, but less frequent communication will most likely only result in slower convergence.

The negotiated, alternating, urn, SIGOP, and serial update control schemes all require communication when the network is set up to establish global numberings for the PEs. Ideally, the proper numbering for the negotiated, alternating, and urn schemes is one which uses the fewest numbers yet assigns a number to each PE such that no PE has the same number as a neighbor. A proper numbering for the SIGOP or serial scheme assigns to each PE a unique number. In addition, a maximal spanning tree must be formed for the SIGOP scheme if the application involves a complex network topology.

There is a problem in relying on a global ordering if PEs are either mobile or can be created and destroyed. This problem arises because the network topology may be changed. If this is the case, additional communication may be needed to maintain a reasonable numbering, and less effective numberings (i.e., numberings which uses more than the fewest numbers possible, or do not form a maximal spanning tree) may need to be used. Note that this problem is finessed by the probabilistic scheme.

The alternating, SIGOP, and serial schemes have no dynamic communication requirements. This demonstrates how static communication can almost be as effective as dynamic communication. The negotiated and urn schemes are the most effective schemes, but have both static and dynamic communication requirements.

5.3 Sensitivity Experiments

Results of experiments on the numbering problem with networks of different sizes and topologies are discussed in this section.

5.3.1 Sensitivity to network size.

Sensitivity to network size is depicted in Figure 9. The dimensions of this graph are the number of processing elements and the average number of iterations to fully converge. Twenty runs on ring networks with 12, 18, 24, 30, 36, and 42 processing elements were averaged together to obtain these results.

Note the large increase in the number of iterations to fully converge observed with the serial and SIGOP schemes as the number of PEs increases. The results of using the SIGOP scheme are quite erratic due to variations in the effectiveness of SIGOP's priming heuristics on the different networks. In contrast, the viable parallel schemes (N, ALT, URN, and PROB) were almost unaffected by an increase in the number of PEs. With 12 PEs the negotiated scheme is four times as fast as the SIGOP scheme; with 42 PEs the negotiated scheme is ten times as fast.

The results of these experiments on sensitivity to network size serve to confirm that since collision effects are local (because interaction is localized) the viable parallel schemes have an increasing advantage over the serial and SIGOP schemes as network size increase.

The distributed iterative refinement algorithm's sensitivity to network size is illustrated. Whereas the speed of convergence for the most part decreases as network size increases when the serial or SIGOP update control scheme (S and SIGOP, respectively) are used, there is almost no decrease in speed as the network size increases when the probabilistic, urn, alternating, and negotiated update control schemes (PROB, URN, ALT, and N, respectively) are used.

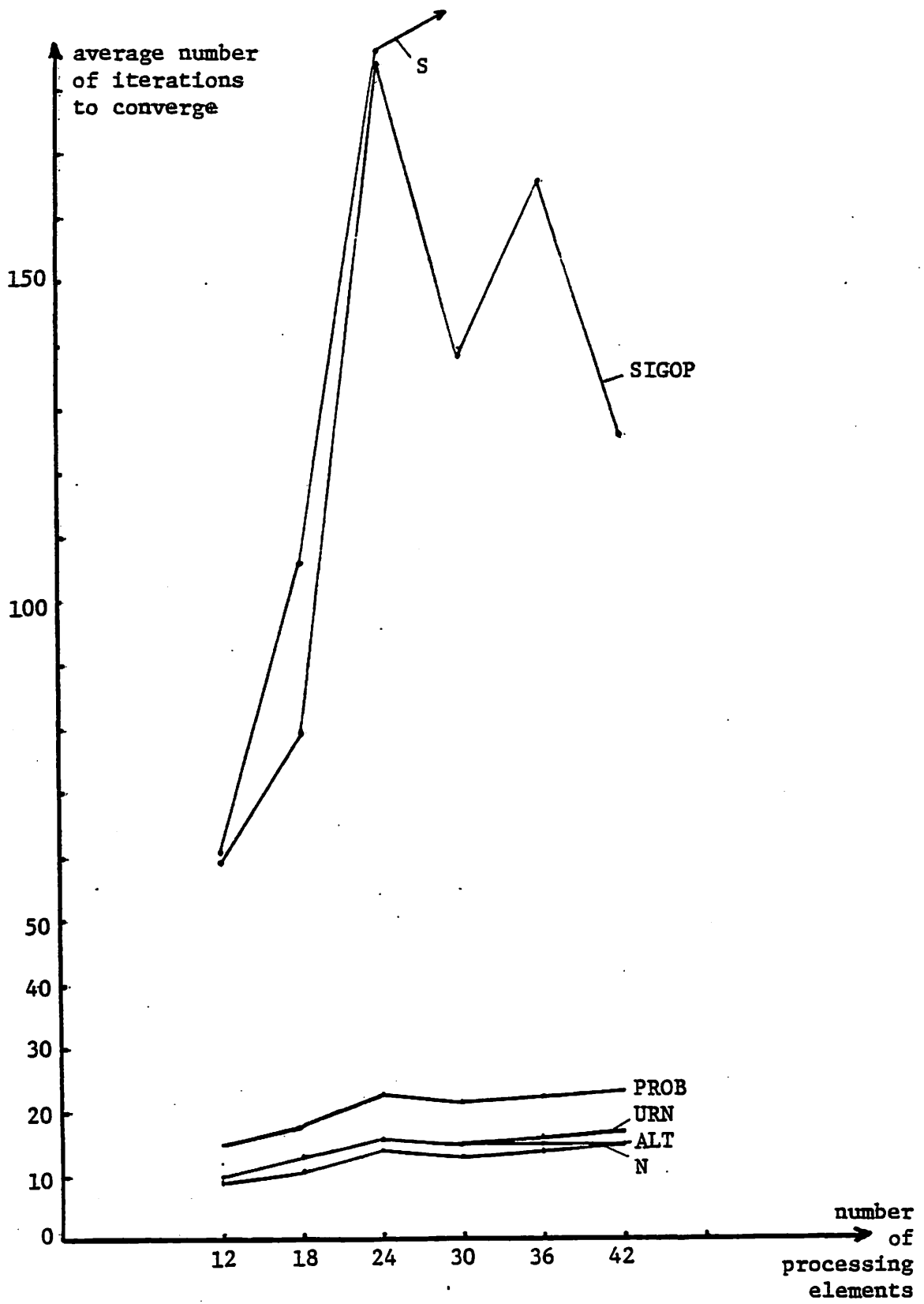


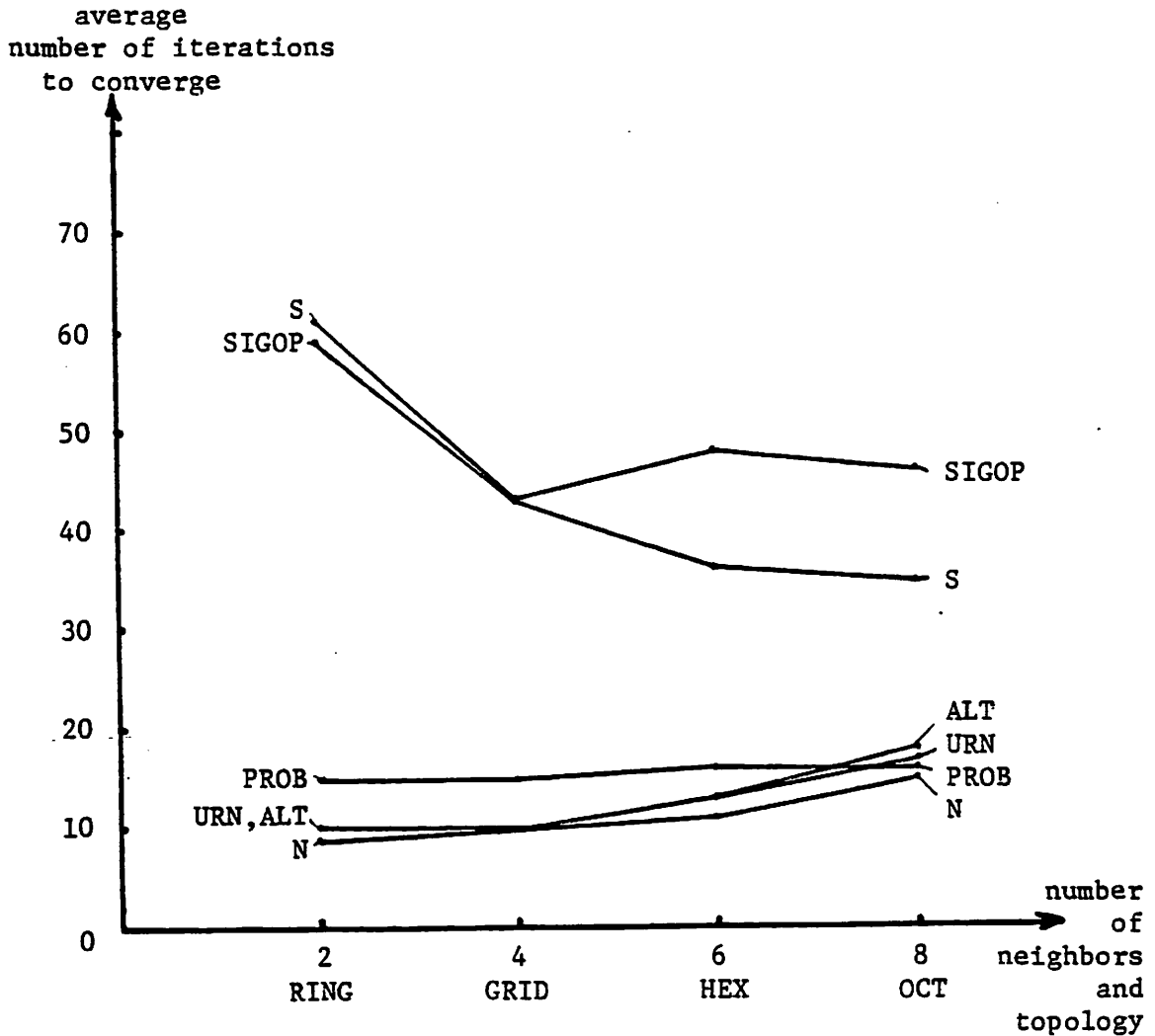
Figure 9: Sensitivity to Network Size.

5.3.2 Sensitivity to network topology.

Sensitivity to network topology (connectivity) is depicted in Figure 10. The dimensions of this graph are the network topology and the average number of iterations to fully converge. Four topologies were examined: a ring network where each PE has 2 neighbors, a grid network where each PE has 4 neighbors, a network called HEX where each PE has 6 neighbors, and a network called OCT where each PE has 8 neighbors. These networks are depicted in Figure 11. Each network is made up of 12 PEs, and is homogenous in the number of connections had by each PE.

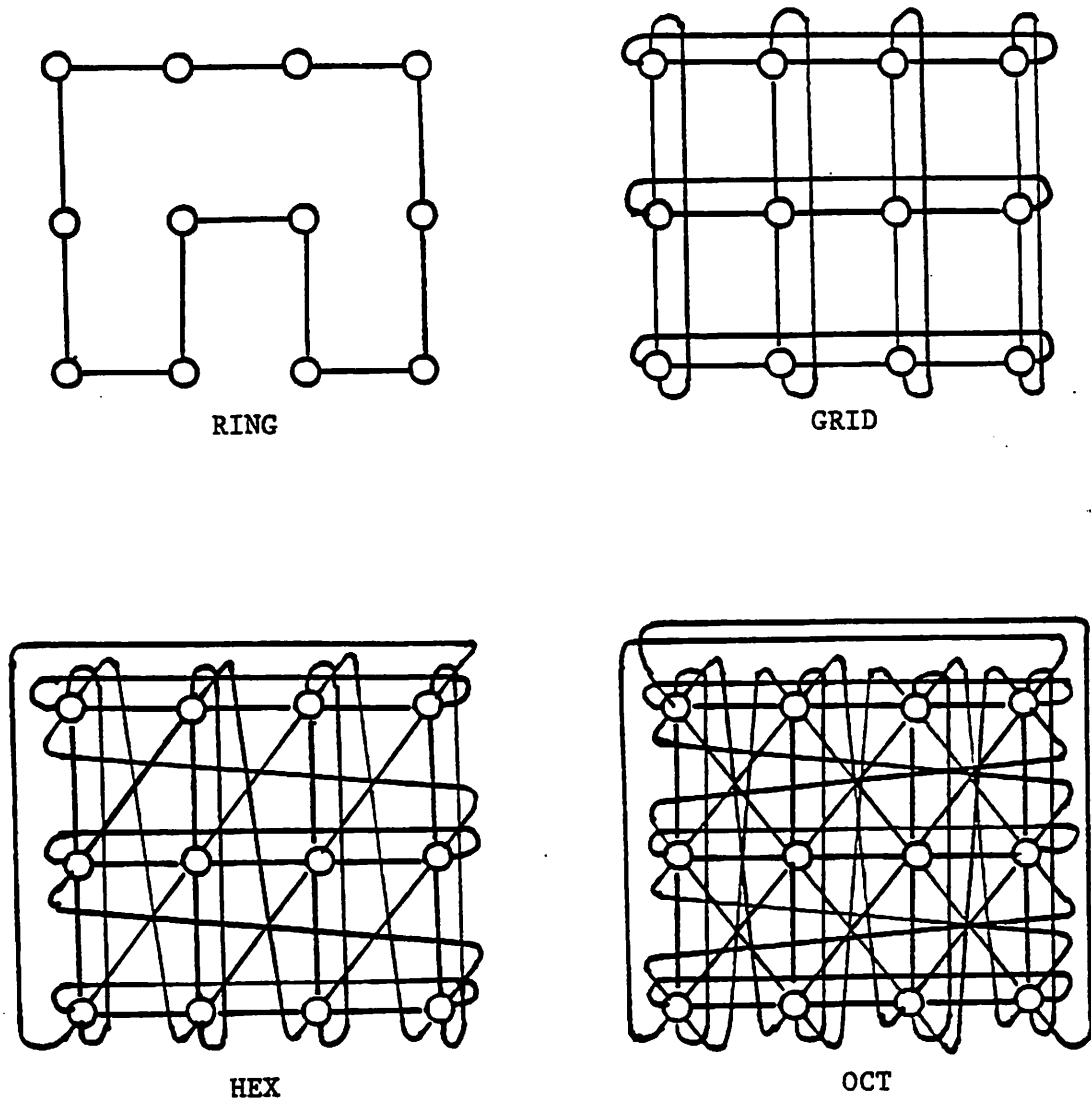
The distributed iterative refinement algorithm with alternating and negotiated update control schemes take slightly longer as the number of neighbors increases because PEs must wait longer before it is their turn to update. The urn scheme's advantage over the alternating scheme finally shows up when PEs each have 8 neighbors; this advantage would be significant if PEs had 50 or more neighbors. The algorithm with SIGOP update control scheme varied in its performance, but seemed to take less time when PEs have more neighbors.

These results show that the advantage in terms of speed of convergence that a distributed iterative refinement algorithm has over a centralized iterative refinement algorithm decreases somewhat as the connectivity of the network of an applications increases. This is not unexpected, because according to theoretical results on access control (see Appendix A) the update control schemes should become less effective



The distributed iterative refinement algorithm's sensitivity to network topology is illustrated. With the probabilistic, urn, alternating, and negotiated update control schemes (PROB, URN, ALT, and N, respectively) there is little decrease in speed of convergence as network topology changes, whereas there is an increase in speed of convergence with the serial and SIGOP update control schemes (S and SIGOP, respectively). The probabilistic, urn, alternating, and negotiated schemes, however, maintain their advantage in speed of convergence over the serial and SIGOP schemes.

Figure 10: Sensitivity to Network Topology.



KEY: ○ processing element
 — link

Four networks of 12 processing elements with different topologies that have been used to determine the distributed iterative refinement algorithm's sensitivity to network topology are shown. They are named RING, GRID, HEX, and OCT, and processing elements in these networks each have 2, 4, 6, and 8 neighbors, respectively.

Figure 11: Network Topologies for Numbering Problem Experiments.

as connectivity increases since there is increased competition for the right to update.

5.4 Discussion

A number of control schemes were quite effective in controlling the simultaneous-update problem. The best parallel update control schemes in decreasing order of performance were as follows: the negotiated scheme, the urn scheme, the alternating scheme, and the probabilistic scheme. These parallel schemes have a distinct advantage over the SIGOP scheme in speed of convergence, which increases as the network size grows.

The negotiated scheme performs well because it is nearly a perfect scheme.¹² The negotiated scheme is decentralized with the possible exception of obtaining a global ordering once, when the network is established. However, PEs with demand must communicate with neighbors each iteration.

The alternating scheme performs well because PEs have few neighbors, and so a busy PE must wait only half an iteration on average with a ring network or one iteration on average with a grid network to obtain the right to update. In general, for a network topology requiring NP phases to number PEs such that no PE is neighbors with a PE

12. A perfect scheme would give the right to update to as many PEs as possible each iteration while avoiding collisions. A perfect scheme would require a global view of which PEs have demand.

of the same phase, PEs must wait $(NP-1)/2$ iterations on average to obtain the right to update.

The urn scheme performs well because it performs like the alternating scheme when demand is high, and performs better than the alternating scheme when demand is low. The urn scheme's advantage over the alternating scheme is not very apparent when PEs have few neighbors.

The probabilistic scheme starts PEs off with a 0.4-0.7 probability of having the right to update when busy, which rises to nearly 1.0 in about a dozen iterations. The probabilistic scheme works well because collisions are more often beneficial than harmful.

The attenuated scheme has potential because limiting the magnitude of changes is quite effective in controlling most simultaneous-updates, but occasional oscillation is a problem. This oscillation results because small changes are not attenuated. Adding a probabilistic, alternating, or negotiated component to the attenuated scheme to resolve oscillation near a solution might result in a viable scheme.

Finally, the SIGOP scheme's heuristics give it a distinct advantage over all the other schemes in finding good solutions, but is not as fast as the good parallel schemes. This disadvantage clearly grows as the size of the network increases.

C H A P T E R VI

AN APPLICATION: DISTRIBUTED NETWORK TRAFFIC LIGHT CONTROL

Network traffic light control (NTLC) involves controlling signals of a network so as to optimize flow in the network [WAGN71, DOT76]. The NTLC problem domain is a very rich source of difficult optimization problems despite the conceptually simple and familiar structure of NTLC problems. As will become apparent, solving NTLC problems in a distributed problem solving environment often requires considerable coordination in the solution of local subproblems due to strong interactions between neighboring as well as non-neighboring subproblems. Furthermore, the combinatorics of many NTLC problems make complete searches infeasible. Thus, the NTLC problem domain exhibits many of the key issues that need to be explored in order to develop a general approach to distributed problem solving.

6.1 Introduction

For network traffic light control there are three principle flow regimes: light, moderate, and congested. Traffic flow in a network is most accurately modeled with individual vehicles; however, other flow models are most often utilized for reasons of efficiency. For light flows, traffic can be modeled with individual vehicles. For moderate flows the the vehicles tend to form platoons, or groups of vehicles which travel together, permitting a platoon flow model. For congested

flows, vehicles tend to form queues in addition to platoons; since the platoons seldom clear the lights without encountering (and joining) a queue, a queueing model is often most appropriate. Coordination among signals is mandatory for the moderate and congested flow regimes.

The control to be applied to signals of the network is specified as a network timing plan. A network timing plan specifies a set of signal timing plans, one for each signal in the network. A signal timing plan specifies when a signal will be green and when it will be red for each approach to the signal. A control period is a length of time for which a timing plan is specified (during a control period, signals may go through a number of green-red cycles).

Optimality of a network timing plan is determined on the basis of global disutility. Disutility is a measure of stops and delay incurred at signals by traffic traversing roads (links) of the network. Global disutility is the sum of disutility on all links in the network; in general it is a function of the network topology, traffic volumes, and the control applied to the signals.

For many NTLC systems a signal timing plan consists of a cycle length, a split, and an offset; this completely specifies the behavior of the signal for a control period if it is assumed that the signal repeats the cycle of control for the length of the control period. If such an assumption is not desirable a sequence of cycle-split-offset triples may be required. The cycle length of a signal is the length of time for a complete sequence of signal phases. A simple signal has two phases, a green phase and a red phase, as viewed from one of the

approaches chosen as a point of reference. The split specifies what portion of the cycle is taken up by each phase. An offset expresses the time relationship between the start of a cycle and a reference point common to all PEs. An alternate specification of a signal timing plan consists of a cycle length and switching times which indicate when each phase starts relative to a global reference.

A network timing plan for many NTLC systems consists of a common cycle length for all signals, and pairs of splits and offsets, one for each signal of the network. The assumption of a common cycle length simplifies matters by keeping the phase relationship between signals fixed for the length of the control period.

An optimal steady-state network timing plan is a network timing plan which is optimal assuming that the timing plan is repeated indefinitely with constant traffic volumes until transient behavior resulting from the change in signal timing plans has died out and a "steady state" has been achieved. An optimal transient or real-time network timing plan is optimal under existing conditions, although usually an optimal real-time solution is approximated by planning up to a finite horizon instead of forever. A typical on-line NTLC system repeatedly senses and/or predicts traffic volumes for an upcoming control period, and computes for implementation an optimal steady-state or real-time network timing plan.

A control period, the interval of time between computations of new timing plans, is on the order of 10-15 minutes, and cycle lengths are on the order of 80 seconds. As long as traffic volumes do not change drastically between control periods an optimal steady-state network timing plan may be appropriate; however, if traffic volumes do change drastically between control periods a real-time solution is more appropriate.

Advantages of real-time solutions include the elimination of the need for smoothing to handle abrupt changes in timing plans between control periods, more flexibility in control, and easier interfacing with special events. A steady-state timing plan may, however, be easier to find and may be preferred to a real-time solution obtained using a short finite horizon.

6.2 A Traffic Light Control Problem

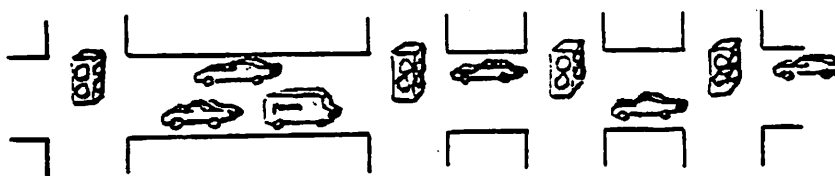
The particular traffic light control problem examined in this thesis is the problem of finding optimal steady-state network timing plans for an artery or network under moderate traffic conditions. This is the problem solved by SIGOP II, a recently developed centralized network traffic light control algorithm, described in the next section of this chapter, that has gained wide acceptance in the traffic control community. For this reason, it has been chosen to provide a comparison for the distributed network traffic light control algorithms developed in this thesis.

A key assumption used in developing an optimal network timing plan for moderate traffic flows is that vehicles traveling along links between signals are released together in platoons from a signal, and travel together in platoons to the next signal. The use of a platoon-based traffic flow model can be represented graphically with a time-space diagram, as shown in Figure 12. Several important features are illustrated in the figure. Progressive movement of platoons in the network is represented as green waves. The location of a green wave, also termed a green band or through band, is determined by the control settings of signals in the network and a set of rules governing the structure of the green wave. The bandwidth indicates the period of time available for traffic flow within the green wave.

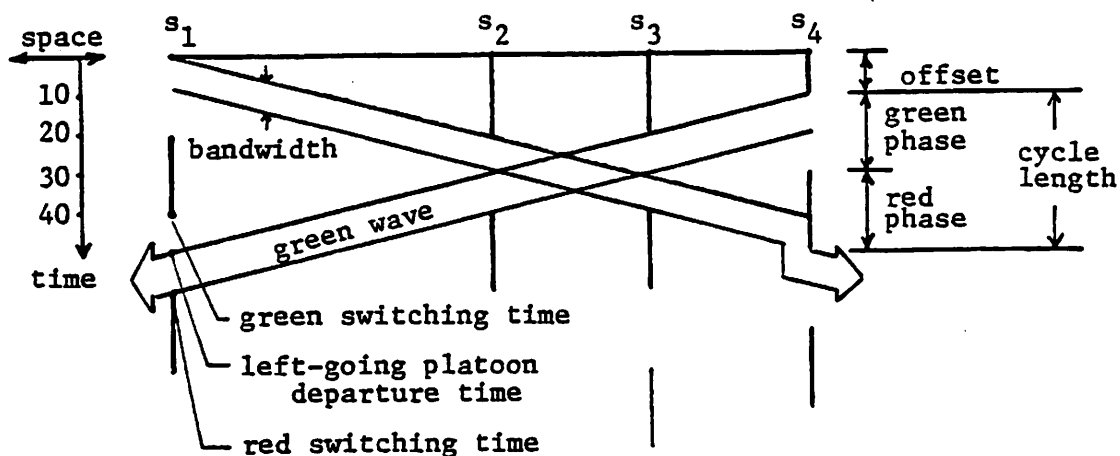
This explicit modeling of the platoon structure utilized in SIGOP II as well as the GLC Combination Method [HUDD69], TRANSYT [ROBE69], and CYRANO [LIEB74], represents an important improvement for light traffic over previous NTLC algorithms such as the original combination method [HILL65, HILL69, GART75] and the first SIGOP [TRC66] which assume the primary platoon is evenly distributed over the entire green phase. A secondary platoon which enters a link during the red phase of the leading signal is often added to model turning traffic, and even more precise platoon models may be employed.

To keep this discussion simple, consider an artery with a single (primary) platoon flow for each direction along the artery. If a platoon is split at a light, an approximation is used to produce a

An artery:



Corresponding time/space diagram:



A time/space diagram is shown for an artery consisting of four traffic signals. Vehicles are assumed to travel in groups called platoons, which enter the artery at both ends when the end signals turn green, and continue to flow at free-flow speed unless they interact with a signal. A common cycle length is used by all signals, and signals repeat the same patterns for a control period (approximately 15 minutes), so an offset and split (ratio of the length of the green phase to the cycle length) completely specifies the timing of a signal during the control period. A signal's timing may also be specified by a green and red switching time.

Figure 12: A Time/Space Diagram.

single platoon for the next link.¹³ The major features of the traffic light control problem, the multi-modal disutility functions and non-local interaction, are exhibited by this simple arterial traffic light control problem.

Disutility incurred by traffic traversing an individual link depends on the control settings for the signals at each end of the link and the arrival times and bandwidths of platoons entering the link. The arrival time of a platoon at signal s_j on link l_{ij} is the departure time for the platoon at s_i on l_{ij} plus the time it takes platoons to traverse l_{ij} . The departure time of a platoon from s_i on l_{ij} is the start of the green phase of s_i if s_i is at the beginning or end of an artery, else it is a function of the arrival time at s_i on the link, l_{hi} , that feeds l_{ij} , and the control applied to s_i . Since these platoon arrival times depend on the control settings and platoon arrival times for other signals, disutility on a link is, in general, a global function of all signal control settings. As will be seen in future discussions, this is a very important characteristic of the problem.

By using state variables which represent platoon departure times, the link disutility calculations can be performed using only information available locally or from (spatial) neighbors. This leads to the steady-state problem specification depicted in Figure 13. The major ramification of utilizing platoon departure times in specifying the

13. A more complex model with a second platoon for turning traffic is employed in the network traffic light control experiments discussed in Chapter 8. This two-platoon model is described in Appendix B.

The arterial traffic light control problem used for initial tests of the distributed iterative refinement algorithm in the presence of non-local interaction and multi-modal, discontinuous objective functions is formally described. The objective is to minimize the sum of disutility, a measure of stops and delay incurred by platoons traversing a link, over all links. Disutility on a link is a function of the green switching time of the receiving signal, the departure time from the sending signal of the single platoon traversing the link, and some constants, the platoon's bandwidth, the time it takes a platoon to traverse the link, the common cycle length, and the duration of the green phase. Constraints expressed with a platoon structure function determine the platoon departure times.

$$\begin{aligned} & \text{MIN}_{\underline{sg}} \sum_{\forall i,j | l_{ij} \in L} D_{ij}(sg_j, td_{ij}) \\ & \text{subject to } td_{ij} = \begin{cases} sg_i & \text{if } \exists h | l_{hi} \in L \wedge h \neq j \\ PS_{hi}(sg_i, td_{hi}) & \text{if } \exists h | l_{hi} \in L \wedge h \neq j \end{cases} \end{aligned}$$

CONSTANTS:

- S is the set of all signals, where s_i is a signal,
 L is the set of all links, where l_{ij} is a link upon which traffic flows from s_i to s_j ,
 C is the common cycle length for all signals,
 gd_i is the duration of the green phase for signal s_i ,
 d_{ij} is the time it takes a platoon to traverse link l_{ij} , and
 pb_{ij} is the bandwidth for platoons traversing link l_{ij} .

CONTROL VARIABLES:

- $\underline{sg} = (sg_1, sg_2, \dots, sg_N)$ is a vector of green switching times where sg_i is the green switching time for signal s_i .

STATE VARIABLES:

- td_{ij} is the platoon departure time from signal s_i on link l_{ij} .

AUXILLARY VARIABLES:

- $ta_{ij} = (((td_{ij} + d_{ij} + sg_j) \bmod C) - sg_j)$ is the platoon arrival time at signal s_j on link l_{ij} , and
 $sr_i = sg_i + gd_i$ is the red switching time for signal s_i .

THE DISUTILITY FUNCTION:

$$D_{ij}(sg_j, td_{ij}) = \begin{cases} 0 & \text{if } sg_j \leq ta_{ij} \leq sr_j - pb_{ij} \\ \alpha (ta_{ij} + pb_{ij} - sr_j)(C - sr_j + \theta) & \text{if } sr_j - pb_{ij} \leq ta_{ij} \leq sr_j \\ \alpha \cdot pb_{ij} (sg_j + C + ta_{ij} + \theta) & \text{if } sr_j \leq ta_{ij} \leq sg_j + C \end{cases}$$

THE PLATOON STRUCTURE FUNCTION:

$$PS_{hi}(sg_i, td_{hi}) = \begin{cases} ta_{hi} & \text{if } (sg_i \leq ta_{hi} \leq sr_i - \frac{pb_{hi}}{2}) \wedge (pb_{hi} \leq gd_i) \\ sg_i & \text{otherwise.} \end{cases}$$

Figure 13: The Arterial Traffic Light Control Problem.

problem is that the platoon departure times for each link of the network may need to be updated whenever any signal's setting in the network is altered because the platoon structure on a link depends on all upstream signals which feed traffic into the link. These state variables can be viewed as representing an environment which reacts to changes in signal settings.

For real-time control a sequence of timing plans would be sought for each signal. With a 10-15 minute control period and 80 second cycle length a sequence of approximately 10 cycles of timing plan is required for each signal in the network. Although additional constraints to ensure proper transitions in each signal's timing plans cuts down on the size of the overall search space, it is still considerably larger than the search space of a similar steady-state NTLC problem.

6.3 The SIGOP II Algorithm

The SIGOP II network traffic light control algorithm [LIEB76] that provides a comparison for the distributed network traffic light control algorithms developed in the next chapter is now discussed. The SIGOP II algorithm searches for a steady-state network timing plan that minimizes network disutility using a flow model which represents traffic flow explicitly as platoons of vehicles traversing links in the network. Knowledge of the network geometry (link lengths) and platoon bandwidths for platoons of each network link (predicted on the basis of link volumes) is used to determine a network timing plan.

The centralized SIGOP II algorithm is structured as an iterative refinement algorithm in order to reduce the complexity of problem solving. The problem of finding a network timing plan which minimizes network disutility is broken up into subproblems which involve finding a signal's timing plan which minimizes local disutility. Local disutility is disutility incurred by vehicles traversing links of a "mini-network" which consists of a signal, its neighbors, and the connecting links. The overall optimization process proceeds as follows:

- 1) Determine the sequence of signals along a maximal spanning tree (MST) of the network. This ordering lists signals attached to high volume links before signals attached to low volume links.

- 2) Prime the control settings at all signals to optimize traffic operations along this ordering. Also platoon departure times are determined.

- 3) Reverse the sequence of signals to get a new ordering.

- 4) For each signal, in the order indicated by the MST sequence, optimize the control setting based on local disutility and update local platoon departure times. In these calculations, the most recent values for signal settings and platoon departure times are used.

- 5) Calculate network-wide disutility (this entails updating platoon departure times until they are consistent, calculating local disutilities, and summing the local disutilities) and if zero or unchanged, stop.

- 6) If a predetermined number of sweeps over the network has been completed and convergence has not been achieved, select the best solution and stop, else continue with step 3.

The optimization is carried out by starting with an approximate solution (the prime) which has been obtained heuristically (steps 1 and 2). This priming process involves setting the first signal arbitrarily and choosing settings for the remaining signals in the indicated sequence on the basis of local disutility, ignoring links attached to signals where signal settings have not yet been determined. The prime is then repeatedly improved on by successive applications of local optimizations (steps 3 through 6).

The use of a maximal spanning tree (MST) to determine a sequence for processing signals is an important heuristic employed by the SIGOP II algorithm. The MST is determined on the basis of traffic volumes; processing signals along this spanning tree amounts to "sweeping" updates along major flows of the network.¹⁴ This tends to preserve the validity of the platoon structure at the site of updates, especially in an artery. Thus, updating of the network platoon structure can be eliminated after individual updates. Complete platoon structure updating, however, is still required periodically and is performed after each sweep to evaluate potential solutions because a valid platoon structure at all signals is required to evaluate a network timing plan accurately.

14. Note that the maximal spanning sequence for an artery is just a sequence along the artery since an artery is a tree with no branches.

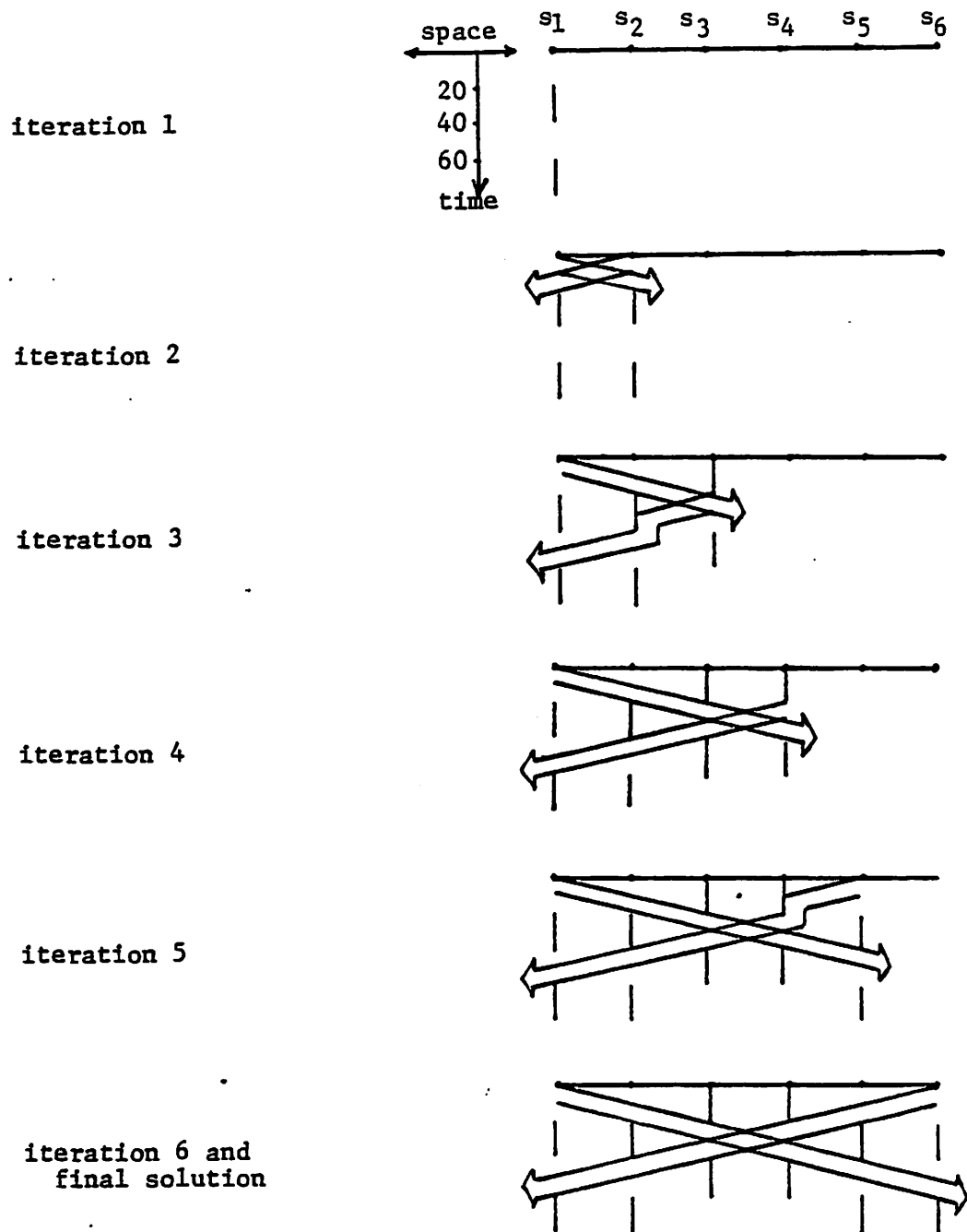
Another, more subtle feature of the MST heuristic ordering is the reversing of the sweep direction after each sweep. This reversal, in addition to allowing each signal to be updated exactly once on each sweep, aids in finding a good solution. It should be noted that subproblems (local optimizations) involve only variables for a small spatial area (a signal and its immediate neighbors). This makes the SIGOP II problem decomposition appear well suited for placing PEs at signals and utilizing neighbors-only communication.

Two sample runs of a SIGOP II test version utilized in a preliminary study [BRO079] are depicted in Figures 14 and 15.

6.4 Discussion

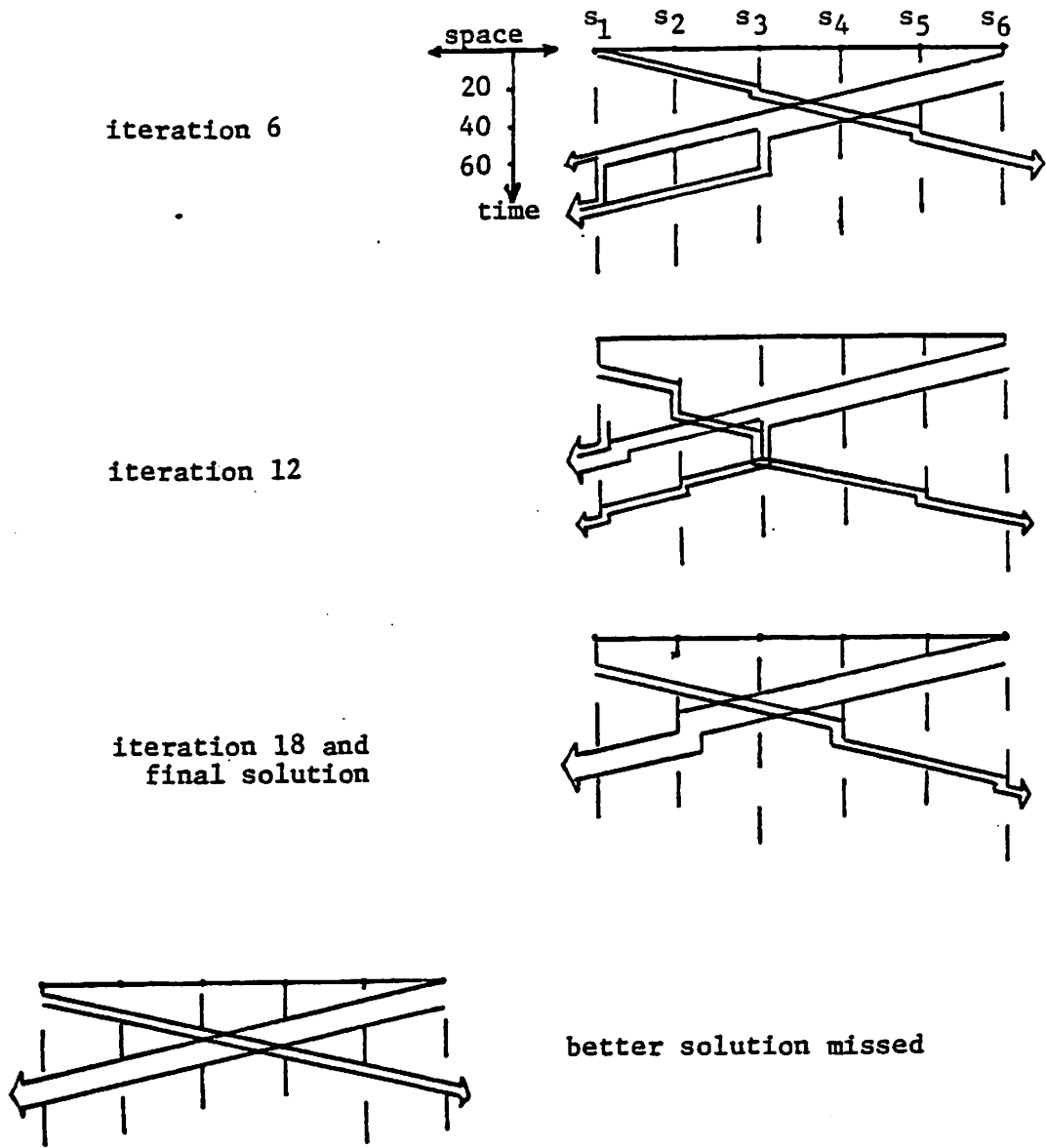
The SIGOP II algorithm described above happens to be a centralized iterative refinement algorithm of the type that the distributed iterative refinement algorithm developed in this thesis is based upon. In fact, the main difference between SIGOP II and the distributed iterative refinement network traffic light control algorithms developed in the next chapter is SIGOP II's serial update control scheme and heuristics.

Sometime after SIGOP II was developed, Lum, Kinney, and Kumar at the University of Minnesota examined the feasibility of a distributed computer traffic control system [LUM79] based on SIGOP II. The study was performed at the same time as some of the research presented in this thesis. The proposed system is based on a centralized multi-processor



The first of two sample runs of SIGOP II on the arterial traffic light control problem is illustrated. The algorithm begins by arbitrarily assigning a timing to the first signal, and proceeds to set the timing of the other signals in turn, so as to minimize disutility on each link that is added. In this case a globally optimum solution is found after just six iterations.

Figure 14: First Sample Run of SIGOP II.



The second of two sample runs of SIGOP II on the arterial traffic light control problem is illustrated. After six iterations, the first "sweep" through the network has been completed, and a solution has not been found. The second sweep proceeds in the opposite direction, and each signal is readjusted so as to minimize disutility on links connected to the signal being readjusted. After a third sweep in the same direction as the first, a solution is found; however, a better solution is missed.

Figure 15: Second Sample Run of SIGOP II.

system. The central processor provides synchronization and performs simple global computations such as periodic evaluations and convergence detection.

The method uses the same platoon-based traffic model and subproblems as SIGOP II. To avoid the simultaneous-update problem, an alternating scheme is employed. $N/2$ processors first solve for half the signals' local subproblems to refine their timings based on the other $N/2$ signals' timings, and then the other $N/2$ subproblems are solved. This process is then repeated until convergence is detected or a pre-specified number of iterations pass.

Unfortunately, no attention is paid to uncertainty introduced by non-local interaction among subproblems. This uncertainty arises from simultaneous updates between subproblems not avoided with the alternating scheme, and from not fully propagating the effects of changes in controls between iterations. Special attention is paid in this thesis to these and other uncertainties introduced by distributing the SIGOP II algorithm.

C H A P T E R VII

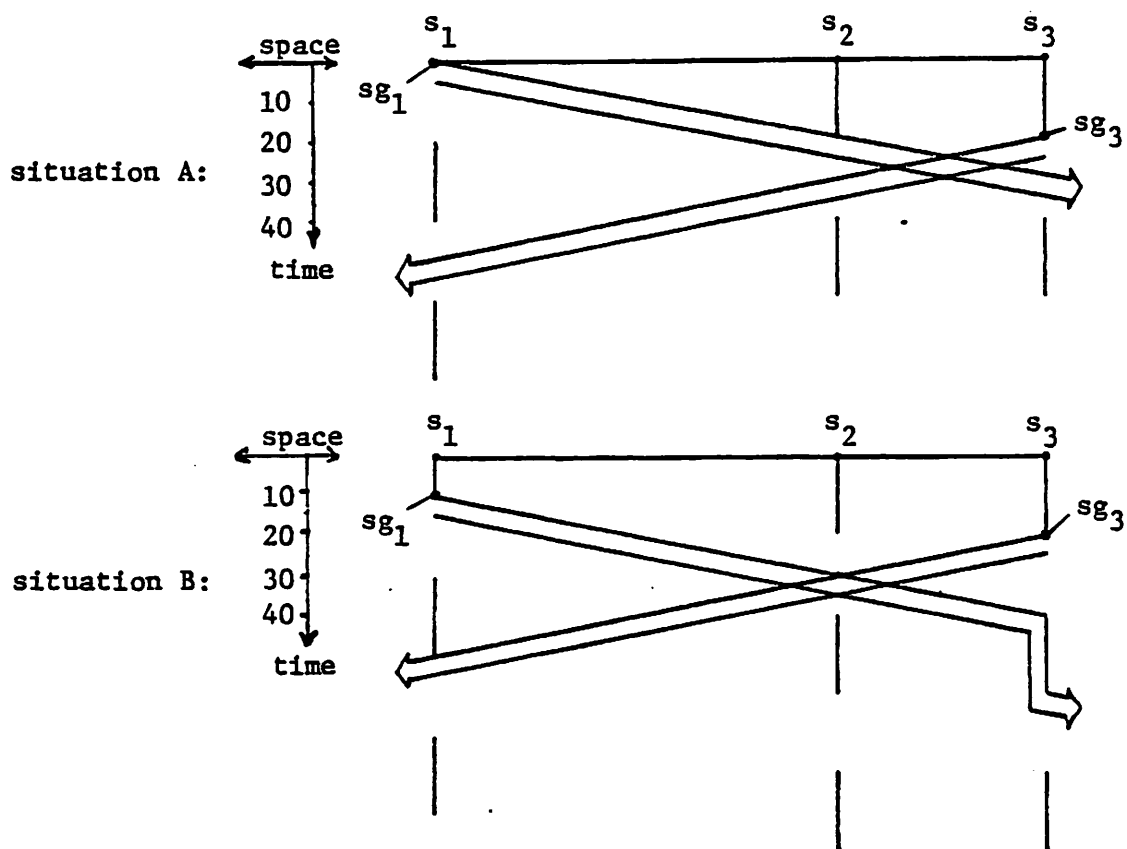
THE PROBLEM OF NON-LOCAL INTERACTION

When there is interaction between non-neighboring PEs' subproblems interaction is said to be non-local. Many real applications exhibit non-local interaction, including the traffic light control problem introduced in the previous chapter. An example of non-local interaction in an arterial traffic light control problem is depicted in Figure 16. Note how a change in PE_1 's control at one end of the artery affects the search for a better control by PE_3 , a non-neighboring processing element at the other end of the artery.

In order to solve a problem with non-local interaction in a distributed manner with an effective utilization of processing resources and with limited communication, it may not be possible to maintain correctness and consistency at all times during the problem solving process. It may be necessary to trade off some accuracy for increased speed and decreased communication by introducing quick approximations based on local information (e.g., by having PEs employ only limited views). The lost accuracy will hopefully be regained through a few extra, less expensive iterations.

The presence of non-local interaction has the following implications:

- o the simultaneous-update problem extends beyond neighbors (i.e., a PE may collide with a non-neighboring PE),



An example of non-local interaction between the processing elements assigned to the first and third signals is illustrated. Notice how the choice of timing for the first signal affects the third processing element's search for a better control (i.e., a green switching time of 20 is optimal for the third signal in Situation A, whereas a green switching time of 25 would be better in Situation B).

Figure 16: An Example of Non-Local Interaction.

- o each iteration PEs must acquire non-neighbors' current controls to accurately access its current situation, and
- o a global view of the effects of changing a control is needed to accurately search for a better control.

These implications are addressed as follows.

First, the problem of simultaneous updates between non-neighbor processing elements is simply ignored, and uncertainty encountered as a result is resolved through additional iteration. The success of the probabilistic and urn update control schemes in alleviating the simultaneous-update problem for the case of localized interaction (see Chapter 5) indicates that at least some uncertainty introduced by simultaneous updates can be resolved through iteration. Furthermore, preliminary experiments with distributed iterative refinement algorithm for traffic light control [BR0079] support this approach.¹⁵

Second, the need for processing elements to acquire non-neighbors' controls to determine its current situation is addressed through the introduction of state variables and state updating which allows a processing element to compute its local disutility using information obtained from neighboring processing elements only. The introduction of state variables and state updating is discussed in Section 7.1.

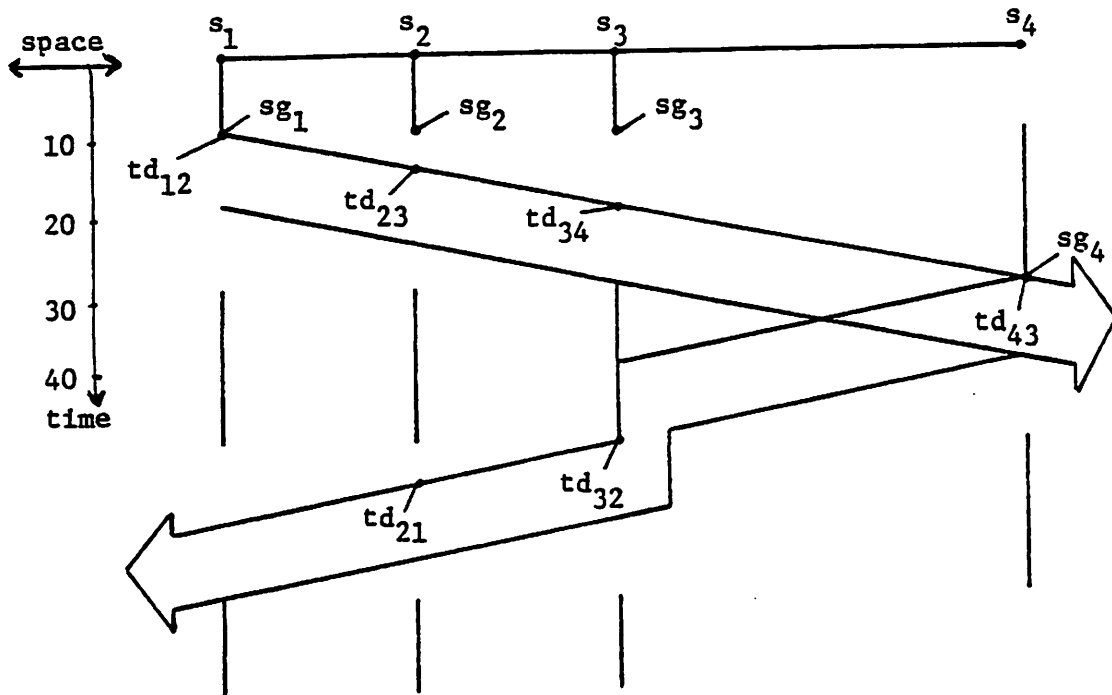
15. Since it is possible that oscillation will occur if collisions between non-neighbor PEs' updates is not controlled, periodic global evaluations of tentative solutions is considered as an alternative to simply taking the last solution after some fixed number of iterations.

And third, the need for processing elements to employ global views to accurately search for better controls is addressed by assuming that a local view of the effects of changing a control is sufficient, by decoupling non-neighboring processing elements through a worst-case approximation, or by acquiring larger views through cooperative gathering of non-local information. These techniques address the problem of retaining accuracy while limiting PEs' views to limit communication; this problem is called the local-view problem, and is discussed in Section 7.3.

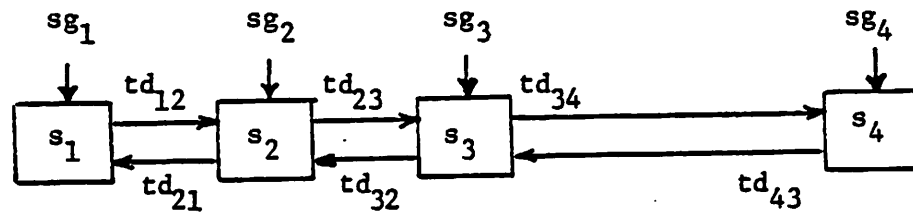
7.1 The Introduction of State Variables

The introduction of state variables and state updating allows a processing element to accurately assess its local situation using only information obtained from neighboring processing elements. This is illustrated in Figure 17 for the traffic light control problem. The state variables for this traffic light control problem are platoon departure times. Now a PE can accurately compute its local disutility (i.e., the disutility on all links entering and leaving the PE's intersection) with information obtained only from neighboring PEs, their current controls and states. Note how these platoon departure times abstract interaction from upstream signals, and how the concept of state variables is natural since interaction between non-neighboring processing elements must pass through some neighboring processing element.

sample time/space diagram:



model of traffic system:



$$td_{ij} = \begin{cases} sg_i & \text{if there is no upstream signal} \\ PS_{hi}(sg_i, td_{hi}) & \text{if } s_h \text{ is an upstream signal} \end{cases}$$

A time/space diagram with controls and states labeled is illustrated along with a model of the traffic flow system. Controls are the signals' green switching times; states are the platoon departure times.

Figure 17: Traffic System State Variables.

Unfortunately, to retain accuracy a global relationship among these state variables must be maintained. This requires state updating where all processing elements repeatedly update their state variables based on their controls and neighbors' current states until consistency has been achieved.¹⁶ State updating does requires communication between neighboring processing elements, but without the introduction of state variables communication between non-neighboring processing elements and knowledge of the network topology and other parameters pertaining to distant parts of the network would be needed.

An amended distributed iterative refinement algorithm is described next for the traffic light control problem. A simulation phase has been inserted before the acquisition phase of the basic algorithm described in Chapter 3, during which state updating takes place. It is called a simulation phase because state updating for network traffic light control involves simulating the flow of platoons through the traffic network to determine platoon departure times. Depending on how the local-view problem is addressed, the calculation phase has also been changed.

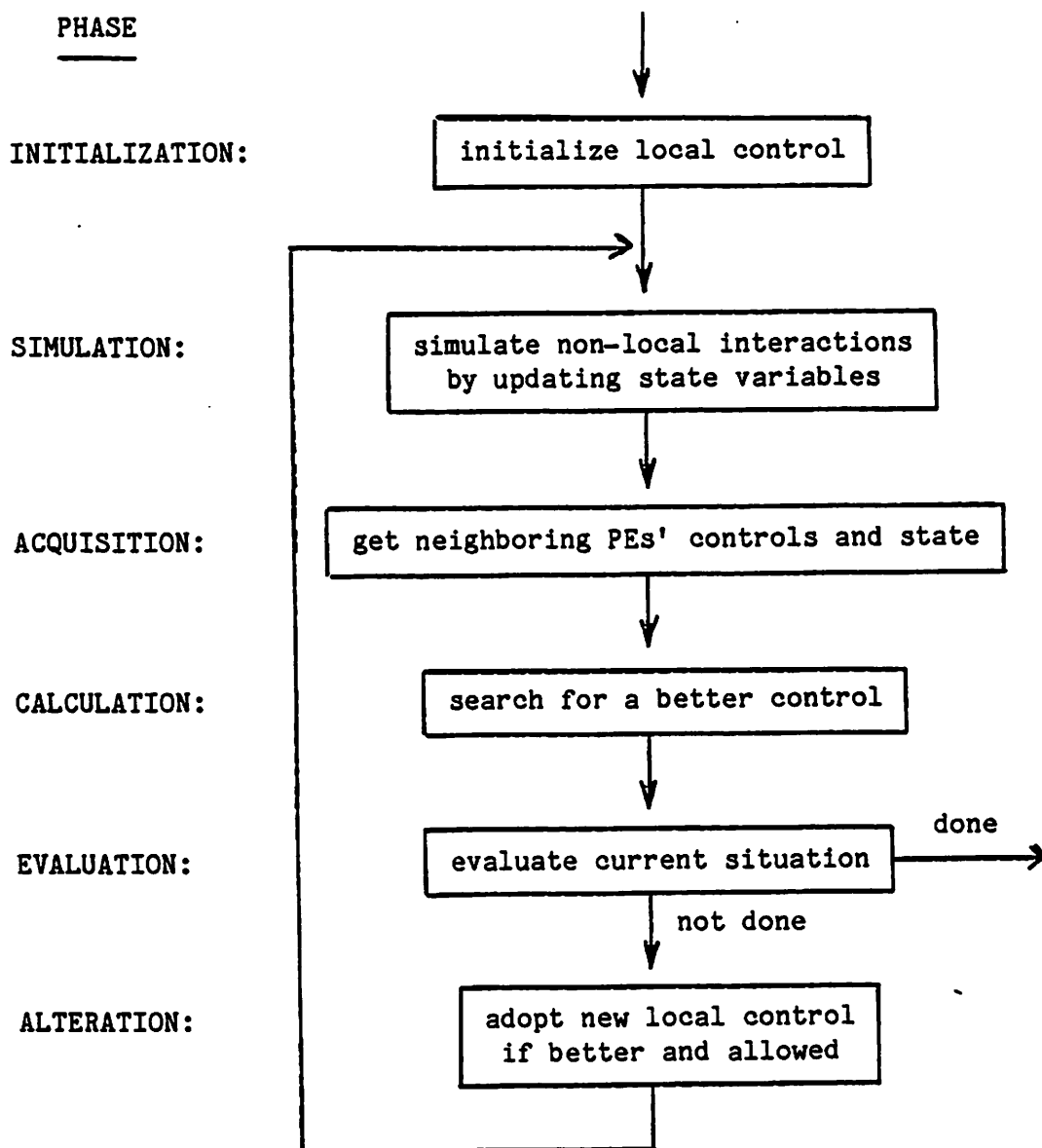
16. Experiments with incomplete (partial) as well as complete state updating are discussed in the next chapter.

7.2 The Amended Algorithm

A number of distributed iterative refinement network traffic light control algorithms are described below, and results of tests are then presented in the next chapter. In all algorithms, as with the basic algorithm described in Chapter 3, processing elements operate in a lock-step (synchronized) parallel fashion. Each processing element is responsible for its own control variable; however, the common goal of all processing elements is to find a control vector for which no further refinements can be found.

The amended algorithm for a processing element is shown in Figure 18. The amended algorithm consists of a number of phases: initialization, simulation, acquisition, calculation, evaluation, and alteration. These phases are described below. At the top-most level, the initialization phase is executed and then the sequence of simulation, acquisition, calculation, alteration phases is repeated until convergence (or a time-out) is detected in the evaluation phase.

Initialization Phase. During the initialization phase PEs initialize their controls to "prime" the iterative refinement algorithm. If the algorithm is run periodically in a quasi-static (slowly changing) environment to obtain an more appropriate control vector, the control vector used in the previous control period may serve as a prime; otherwise, a fixed, random, or heuristically obtained prime may be used. This process is described below for PE_i .



The amended algorithm for a PE consists of a number of phases: initialization, simulation, acquisition, calculation, evaluation, and alteration. All PEs perform the same computation in a lock-step, parallel fashion, until no PE can find a better control or a pre-specified number of iterations have passed. All phases may require some communication.

Figure 18: The Amended Distributed Iterative Refinement Algorithm.

```

(* INITIALIZATION PHASE: initialize local control *)
(* other variables are also initialized at this time *)
(* sg[i] is PEi's control to be initialized to PRIME[i] *)
(* it_count will keep count of the number of iterations *)

sg[i] := PRIME[i];
it_count := 0;

```

Other initializations may take place at this time. Unless these other initializations require communication, no communication is required for the initialization phase.

Simulation Phase. During the simulation phase PEs repeatedly update their state variables based on their current controls and neighbors' current states obtained through communication. Fewer than N state updates may suffice depending upon network topology and desired accuracy. This process is described below for PE_i .

```

(* SIMULATION PHASE: simulate non-local interactions *)
(* by updating state variables *)
(* td[i,j] is PEi's current state variable for traffic *)
(* flow to neighbor PEj *)
(* td[h,i] is neighbor PEh's current state variable for *)
(* traffic flow to PEi *)
(* N is the number of PEs *)
(* upstream(i,j) returns the index of the upstream neighbor *)
(* that feeds traffic through PEi to neighbor PEj if such *)
(* a PE exists, else it returns 0 *)

for count := 1 to N
  do for j := neighbors of PEi
    do if upstream(i,j) > 0
      then begin
        h := upstream(i,j);
        get td[h,i] from PEh;
        td[i,j] := PShi(td[h,i],sg[i])
      end
    else td[i,j] := sg[i];
  end
end

```

Note that PE_i must make its current state variables available to neighboring PEs during this phase, since other PEs are in their

simulation phase too.

Acquisition Phase. PEs acquire neighboring PEs' controls and states through communication during this phase. This process is described below for PE_i .

```
(* ACQUISITION PHASE: get neighboring PEs' controls and states *)
(*  sg[j] is neighbor PEj's current control                      *)
(*  td[j,i] is neighbor PEj's current state variable for        *)
(*    traffic flow to PEi                                       *)
```

```
for j := neighbors of PEi
  do get sg[j] and td[j,i] from PEj;
```

Note that a PE must make its current control and states available to neighboring PEs during this phase since other PEs are in their acquisition phase too.

Calculation Phase. Processing elements solve their subproblems during the calculation phase to search for better controls. In Section 7.3, The Local-View Problem, a number of calculation phases are developed. This phase is generally the most time-consuming of all the phases of the algorithm. Because processing elements need neighboring PEs' current controls and states for the solution of its subproblem, this information was communicated during the acquisition phase.

Evaluation Phase. After the acquisition phase processing elements generally have consistent views of the situation they are faced with given the current solution vector unless communication errors are considered. Local and possibly global evaluations are performed at this time, and convergence (or a time-out) may be detected. A simple evaluation process is given below for PE_i .

```

(* EVALUATION PHASE: evaluate current situation *)
(* demand[i] is true if PEi wants to alter its control *)
(* converged[i] will be set to indicate if convergence *)
(* has been achieved *)
(* it_count is current iteration number *)

```

```

it_count := it_count + 1;
converged[i] := not demand[i];
for i := 1 to N
  do for j := neighbors of PEi
    do if converged[i]
      then begin
        get converged[j] from PEj;
        converged[i] := converged[i] and converged[j]
      end;
  if (it_count > it_max) or converged[i]
    then exit;

```

Communication is required for convergence detection and for global evaluations.

Alteration Phase. Processing elements that find better controls must not always adopt their new controls, else the simultaneous-update problem results. For this reason, an alteration phase is included in the algorithm, during which processing elements use an update control scheme to determine which processing elements may adopt new controls. This phase is depicted below for PE_{*i*}.

```

(* ALTERATION PHASE: adopt new control if better and allowed *)
(* demand[i] indicates if PEi wants to alter its control *)
(* sg[i] is current control *)
(* best[i] is new control found in the calculation phase *)
(* allowed(i) is a call to an update control scheme *)

if demand[i]
  then if allowed(i)
    then sg[i] := best[i];

```

Eight update control schemes were described in detail in the Chapter 4. These same update control schemes are employed during the alteration phase; however, only neighboring PEs are considered to interact with a

PE. Because some update control schemes require communication, communication between neighboring PEs may be required for this phase.

7.3 The Local-View Problem

Although it is desirable to have processing elements in a distributed problem solving system employ a limited, local view of the overall problem, uncertainty introduced as a result of employing a local view may degrade performance if there is significant non-local interaction. In this section the local-view approximation is first discussed. A worst-case decoupling technique that eliminates interaction with non-neighbors and cooperative information gathering technique for obtaining global views are then developed as alternatives to the local-view approximation. These techniques are tested in experiments described in Chapter 8.

The network traffic light control application provides a convenient context in which to explain the local-view approximation, the worst-case decoupling assumption, and the cooperative information gathering technique. The traffic light control problem is the problem of determining an optimal control vector, a steady-state network timing plan designated sg, as described in Chapter 6. This involves minimizing global disutility, the sum of disutility on all links of the traffic network.

A platoon-based traffic model (see Chapter 6 or Appendix B) is employed for the traffic light control problem, and disutility is calculated on each link based on a measure of stops and delay incurred by traffic. State variables, platoon departure times, make it possible to calculate the disutility on a link using only local information (see Section 7.1). State variables are related through a state dependency function, in this case a function that defines the platoon structure. Thus, the disutility for some link l_{ij} that carries traffic from signal s_i to signal s_j , is denoted as $D_{ij}(sg_j, td_{ij})$ where sg_j is the control component assigned to PE_j , and td_{ij} is a state variable. Furthermore, td_{ij} is $PS_{hi}(sg_i, td_{hi})$ if there exists an upstream neighbor of PE_i , PE_h , such that traffic flows from signal s_h through s_i to s_j .¹⁷

7.3.1 A local-view approximation.

A PE employs a local-view approximation if it assumes that a local view of the effects of changing its control is a good approximation of the global view of the effects of changing its control. For the traffic light control problem, this involves searching for a better control based on local disutility, disutility incurred by traffic in a "mini-network" comprised of only links entering and leaving a PE's intersection.

17. For compactness in discussions below, let $upstream(i,j)$ denote the index of PE_i 's neighbor (if it exists) that feeds traffic through s_i to s_j .

Local disutility for a PE is a function of the PE's control and neighboring PEs' current controls and states. This may be written for PE_i as

$$\text{Local_Disutility}_i = \sum_{j|1_{ij} \in L} D_{ij}(sg_j, td_{ij}) + D_{ji}(sg_i, td_{ji}),$$

$$\text{where } td_{ij} = \begin{cases} sg_i & \text{if } \exists h|h=\text{upstream}(i,j), \\ PS_{hi}(sg_i, td_{hi}) & \text{if } \exists h|h=\text{upstream}(i,j). \end{cases}$$

Now PE_i 's subproblem may be expressed as

$$\begin{array}{ll} \text{MIN} & \text{Local_Disutility}_i \\ & sg_i \end{array} \quad (7.1)$$

and a calculation phase for the amended distributed iterative refinement algorithm that employs this local-view approximation can be specified for PE_i as follows.

```
(* CALCULATION PHASE: search for a better control *)
(* using a local-view approximation *)
(* sg[i] is current control *)
(* td[i,j] is PEi's current state variable for traffic *)
(* flow to neighbor PEj *)
(* cost[i] is current local disutility associated with *)
(* using sg[i] *)
(* ŝg[j] is PEi's copy of PEj's current control *)
(* t̂d[j,i] is neighbor PEj's current state variable for *)
(* traffic flow to PEi *)
(* upstream(i,j) returns the index of the upstream neighbor *)
(* that feeds traffic through PEi to neighbor PEj if such *)
(* a PE exists, else it returns 0 *)
(* best[i] will be better control *)
(* demand[i] will be set to indicate if PEi wants to *)
(* alter its control *)

best[i] := sg[i];
cost[i] := 0;
for j := neighbors of PEi
  do cost[i] := cost[i] + Dij(ŝg[j], t̂d[i,j])
```

```

                                + Dji(sg[i],t̂d[j,i]);
min_cost := cost[i];
for sg := all values for sg[i]
do begin
  temp := 0;
  for j := neighbors of PEi
  do if upstream(i,j) > 0
  then begin
    h := upstream(i,j);
    temp := temp + Dij(ŝg[j],PShi(sg,t̂d[h,i]))
              + Dji(sg,t̂d[j,i])
  end
  else temp := temp + Dij(ŝg[j],sg)
              + Dji(sg,t̂d[j,i]);
if temp < min_cost
then begin
  min_cost := temp;
  best[i] := sg
end
end;
demand[i] := (best[i] = sg[i]);

```

The local-view approximation described above is based on an assumption that a local view of the effects of changing a control is a good approximation of the global view. Unfortunately, this is not always the case, and a locally beneficial change in control may be globally detrimental; thus, even a collision-free update may result in a global increase in the objective function. This can not only lead to oscillation (even when a serial control scheme is utilized) but also means that periodic global evaluations may be necessary. Uncertainty as to the value of a change in control based on a local view is called local-view uncertainty.

The SIGOP II network traffic light control algorithm described in Section 6.3 is an example of a serial iterative refinement algorithm which employs local view approximation. The approximation greatly reduces communication and computation, but as Figure 19 illustrates,

updates based on a local view may actually cause an increase in global disutility. Thus, convergence cannot be guaranteed and oscillation may occur. Periodic evaluations of the approximate solutions are therefore performed to offset the fact that the algorithm may not always improve on the current approximation. Lieberman and others [LIEB74, LIEB76] considered it worthwhile to trade off having to compute global views for some accuracy in the centralized SIGOP II.

7.3.2 A worst-case assumption.

An additional approximation utilizing a worst-case assumption may be used in conjunction with the local view approximation to permit the complete decoupling of a PE's subproblem from non-neighboring PE's subproblems. This approximation eliminates the need for state updating (the simulation phase) as it transforms a problem with non-local interaction into a problem with localized interaction, but may yield less desirable solutions.

Worst-case local disutility for a PE is a function of the PE's controls and neighboring PE's controls (but not neighboring PE's states). This may be written for PE_i as

Worst_Case_Local_Disutility_i =

$$\sum_{j: i, j \in L} \text{MAX}_{td_{hi}, td_{kj}} D_{ij}(sg_j, td_{ij}) + D_{ji}(sg_i, td_{ji}) ,$$

$$\text{where } td_{ij} = \begin{cases} PS_{hi}(sg_i, td_{hi}) & \text{if } \exists h: h = \text{upstream}(i, j), \\ sg_i & \text{if } \sim \exists h: h = \text{upstream}(i, j), \end{cases}$$

$$\text{and } td_{ij} = \begin{cases} PS_{kj}(sg_j, td_{kj}) & \text{if } \exists k: k = \text{upstream}(j, i), \\ sg_j & \text{if } \sim \exists k: k = \text{upstream}(j, i), \end{cases}$$

Now PE_i's subproblem may be expressed as

$$\text{MIN}_{sg_i} \text{Worst_Case_Local_Disutility}_i . \quad (7.2)$$

Since interaction is now localized, the basic distributed iterative refinement algorithm described in Chapter 3 with the following calculation phase:¹⁸

```
(* CALCULATION PHASE: search for a better control *)
(* using a local-view approximation and worst-case *)
(* decoupling technique *)
(* sg[i] is current control *)
(* cost[i] is current worst-case local disutility *)
(* associated with using sg[i] *)
(* ŝg[j] is PEi's copy of PEj's current control *)
(* upstream(i,j) returns the index of the upstream neighbor *)
(* that feeds traffic through PEi to neighbor PEj if such *)
(* a PE exists, else it returns 0 *)
(* best[i] will be better control *)
(* demand[i] will be set to indicate if PEi wants to *)
(* alter its control *)
```

18. Note that the controls $sg[i]$ and $\hat{sg}[j]$ are referred to as $u[i]$ and $\hat{u}[j]$ in the basic algorithm of Chapter 3.

```

function worst(i,j: 1..N; sgi,sgj: control_range): real;
var worstin,worstout,tempin,tempout: real;
    tdhi,tdkj: state_range;
begin
    worstout := 0;
    worstin := 0;
    if upstream(i,j) > 0
        then for tdhi := possible state values
            do begin
                tdij := PShi(sgi,tdhi);
                tempout := Dij(sgj,tdij);
                if tempout > worstout
                    then worstout := tempout
                end
            end
        else worstout := Dij(sgj,sgi);
    if upstream(j,i) > 0
        then for tdkj := possible state values
            do begin
                tdji := PSkj(sgj,tdkj);
                tempin := Dji(sgi,tdji);
                if tempin > worstin
                    then worstin := tempin
                end
            end
        else worstin := Dji(sgi,sgj);
    worst := worstin + worstout
end;

best[i] := sg[i];
cost[i] := 0;
for j := neighbors of PEi
    do cost[i] := cost[i] + worst(i,j,sg[i], $\hat{sg}[j]$ );
min_cost := cost[i];
for sg := all values for sg[i]
    do begin
        temp := 0;
        for j := neighbors of PEi
            do temp := temp + worst(i,j,sg, $\hat{sg}[j]$ );
        if temp < min_cost
            then begin
                min_cost := temp;
                best[i] := sg
            end
        end
    end;
demand[i] := (best[i] = sg[i]);

```

This calculation phase will only require that neighboring PEs' controls be acquired during the acquisition phase.

Note that a solution found using (7.2) is often better than predicted assuming worst-case interaction (platoon structure) because worst-case interaction is seldom actually encountered for all subproblems. It may even be possible to take further advantage of this by readjusting local controls given the actual interaction pattern. This must be done carefully, however, so as not to disturb interactions since the non-local effects of changing interactions are still unknown to processing elements.

It is interesting that until recently a decoupling assumption like the worst-case decoupling assumption described above has been used in network traffic light control. Instead of assuming the worst-case interaction from neighboring processing elements, interaction was fixed (i.e., primary platoons were assumed to leave intersections always at the start of the green phase). Other decoupling techniques (e.g., most-likely interaction, best-case interaction) are also possible.

7.3.3 Cooperative gathering of non-local information.

A local view approximation is not always appropriate. Non-local constraints, for example, can be violated if such an approximation is used when such constraints are present. For multi-commodity flow problems a local view approximation is often inappropriate simply because the local view yields a poor approximation of the global view (bottlenecks outside of the local view are not seen). For such problems there appears to be an unavoidable need for certain non-local

information.

To gather larger views of the effects of changing controls (assuming other controls don't change), cost-to-go vector parameterized on possible interaction between PEs can be computed utilizing a technique similar to spatial dynamic programming [BELL62, LARS79a, CLIN79]. These cost-to-go vector can be computed iteratively.

Instead of assuming optimal controls and utilizing the basic dynamic programming recurrence relation,

$$X_{hi}[td_{hi}] = \begin{cases} \begin{matrix} \text{MIN} & D_{hi}(sg, td_{hi}) \\ & sg \end{matrix} & \text{if } \sim \exists j|h=\text{upstream}(i,j), \\ \begin{matrix} \text{MIN} & D_{hi}(sg, td_{hi}) + X_{ij}[PS_{hi}(sg, td_{hi})] \\ & sg \end{matrix} & \text{if } \exists j|h=\text{upstream}(i,j), \end{cases}$$

a simpler relation is used because PEs' controls are assumed to be frozen. This simpler recurrence relation is

$$X_{hi}[td_{hi}] = \begin{cases} D_{hi}(sg_i, td_{hi}) & \text{if } \sim \exists j|h=\text{upstream}(i,j), \\ D_{hi}(sg_i, td_{hi}) + X_{ij}[PS_{hi}(sg_i, td_{hi})] & \text{if } \exists j|h=\text{upstream}(i,j). \end{cases} \quad (7.3)$$

Note that PE_i must obtain X_{ij} from PE_j to determine X_{hi} (assuming PE_h feeds traffic through PE_i to PE_j). More generally, the cost-to-go vectors, X_{ij} 's, are passed up "against" chains of dependencies (upstream for traffic light control) and give the costs associated with presenting PEs further along chains of dependencies (downstream for traffic light control) with certain states.

Now, using the cost-to-go vectors, the global disutility of all links affected by changes in PE_i 's control, sg_i , can be expressed as follows.

Sufficiently_Global_Disutility_i =

$$\text{MIN}_{sg_i} X_{ij}[td_{ij}] + D_{ji}(sg_i, td_{ji})$$

$$\text{where } td_{ij} = \begin{cases} PS_{hi}(sg_i, td_{hi}) & \text{if } \exists h: h = \text{upstream}(i, j) \\ sg_i & \text{if } \sim \exists h: h = \text{upstream}(i, j) \end{cases}$$

And, the new subproblem for PE_i is

$$\text{MIN}_{sg_i} \text{Sufficiently_Global_Disutility}_i \quad . \quad (7.3)$$

A calculation phase is described below which involves iterating on the information gathering recurrence relation described above, and then searching for a better control with a sufficiently global view.

```
(* CALCULATION PHASE: search for a better control *)
(*   after cooperative information gathering *)
(*   using a sufficiently global view *)
(*   sg[i] is current control *)
(*   td[i,j] is PEi's current state variable for traffic *)
(*   flow to neighbor PEj *)
(*   cost[i] is current sufficiently global disutility *)
(*   associated with using sg[i] *)
(*    $\hat{sg}[j]$  is PEi's copy of PEj's current control *)
(*    $\hat{td}[j,i]$  is neighbor PEj's current state variable for *)
(*   traffic flow to PEi *)
(*   upstream(i,j) returns the index of the upstream neighbor *)
(*   that feeds traffic through PEi to neighbor PEj if such *)
(*   a PE exists, else it returns 0 *)
(*   downstream(h,i) returns the index of the downstream *)
(*   neighbor that receives traffic through PEi from *)
(*   neighbor PEh if such a PE exists, else it returns 0 *)
```

```

(* Xhi is PEi's cost-to-go vector for the flow from PEh *)
(* that passes through PEi *)
(* best[i] will be better control *)
(* demand[i] will be set to indicate if PEi wants to *)
(* alter its control *)

for count := 1 to N
  do for h := neighbors of PEi
    do if downstream(h,i) > 0
      then begin
        j := downstream(h,i);
        get Xij from PEj;
        for tdhi := all states
          do Xhi[tdhi] := Dhi(sg[i],tdhi)
            +  $\hat{X}_{ij}$ [PShi(sg[i],tdhi)]
        end
      else for tdhi := all states
        do Xhi[tdhi] := Dhi(sg[i],tdhi);
    best[i] := sg[i];
    cost[i] := 0;
    for j := neighbors of PEi
      do cost[i] := cost[i] +  $\hat{X}_{ij}$ [td[i,j]] + Dji(sg[i], $\hat{td}$ [j,i]);
    min_cost := cost[i];
    for sg := all values of sg[i]
      do begin
        temp := 0;
        for j := neighbors of PEi
          do if upstream(i,j) > 0
            then begin
              h := upstream(i,j);
              temp := temp +  $\hat{X}_{ij}$ [PShi(sg,td[h,i])]
                + Dji(sg, $\hat{td}$ [j,i])
            end
          else temp := temp +  $\hat{X}_{ij}$ [sg] + Dji(sg, $\hat{td}$ [j,i]);
        if temp < min_cost
          then begin
            min_cost := temp;
            best[i] := sg
          end
        end;
    demand[i] := (best[i] = sg[i]);

```

Unfortunately, repeated communication of the cost-to-go vectors is required during this calculation phase in addition to the communication during the acquisition phase of neighbors' controls and states. This amounts to significant communication if N information gathering

iterations are performed each time the calculation phase is executed, although it is neighbors-only communication. However, if the longest chain of dependencies (continuous flow of traffic) involves M PEs where $M < N$, the total number of PEs, then M information gathering iterations will suffice with no loss of accuracy. Furthermore, fewer than M information gathering iterations will provide a compromise between local and global views.

C H A P T E R VIII

EXPERIMENTS WITH NON-LOCALIZED INTERACTION

The results of experiments using the distributed iterative refinement algorithm for arterial and network traffic light control developed in the previous chapters is discussed in this chapter. These experiments test the distributed iterative refinement algorithm with the local-view approximation, the local-view approximation with decoupling assumption, and the information gathering technique also developed in the previous chapter. Tolerance of the iterative refinement algorithm to communication errors and partial state updating is measured too.

The traffic light control problem solved by the iterative refinement algorithm in these experiments was introduced in Section 6.2 of this thesis. Complications encountered with the traffic light control application include:

- o a complex, multi-modal search space,
- o non-local interaction, and
- o bi-directional flows.

The most important of these differences between the traffic light control problem and the numbering problem discussed in Chapter 5 is the presence of interaction between non-neighboring processing elements with the traffic light control problem (through the platoon structure) not encountered with the numbering problem.

8.1 Arterial Traffic Light Control Experiments

The first traffic light control experiments were conducted on 12-signal arteries with two-way traffic. The arterial traffic light control problem was precisely specified in Figure 13 (in Section 6.2, page 110). These tests focus on the performance of the algorithm with the update control schemes described at the end of Chapter 4 and the local-view approximation, the local-view approximation with decoupling assumption, and the information gathering technique.

In all experiments, results of twenty runs, each starting with a different prime, were averaged together. A common cycle length of 40 seconds and constant green durations of 20 seconds is assumed, and the right- and left-going platoon bandwidths are 10 seconds. The eleven free-flow delays between signals s_1 through s_{12} are 20, 10, 10, 10, 10, 5, 15, 30, 20, 30, and 20 seconds, in order.

8.1.1 Performance with local-view approximation.

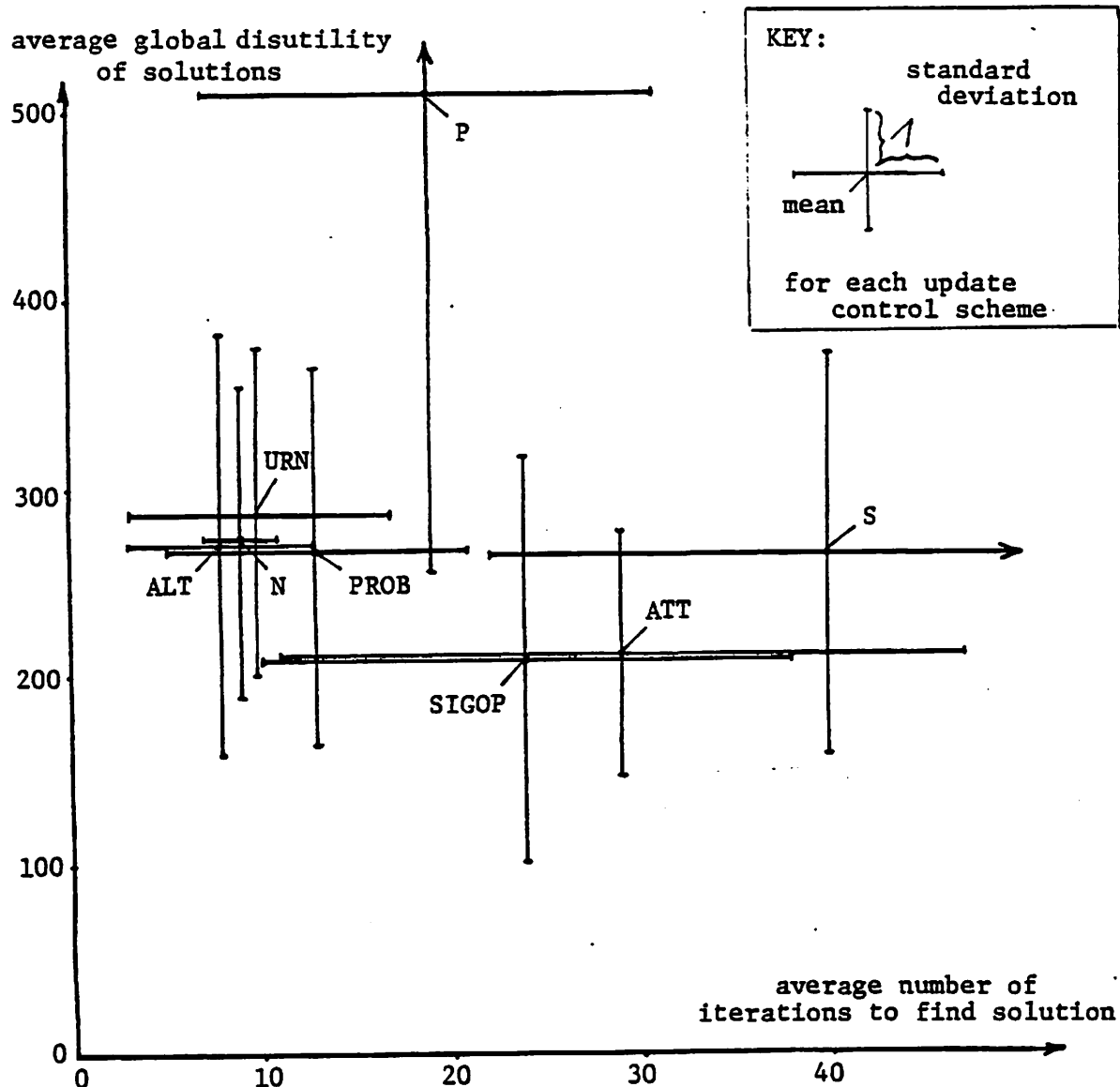
The local-view approximation described in detail in Subsection 7.3.1 works under the assumption that a local view of the effects of changing a control is a good approximation of the global effects of changing the control. Such an approximation is used in the existing traffic light control algorithm used for comparison, SIGOP II.

As a result of using the local-view approximation, convergence cannot be guaranteed with any control scheme because a successful update can result in an increase in global disutility. For this reason a global evaluation is performed after each iteration and the best solution found is remembered; if the algorithm has not converged within a predetermined number of iterations, the algorithm is just stopped.

Performance on the arterial traffic light control problem with the local-view approximation and a variety of update control schemes is shown in Figure 20. Performance on the arterial traffic light control problem is quite similar to the performance on the numbering problem with ring network (see Section 5.2).

With the pure parallel update control scheme the algorithm oscillated and failed to find decent solutions. The SIGOP II algorithm found the best solutions, but the distributed iterative refinement algorithm with alternating, negotiated, urn, and probabilistic schemes produce solutions very close in quality to those found by SIGOP II while taking half the time. The algorithm with attenuated and serial update control schemes took longer than SIGOP II; and with the attenuated scheme the algorithm occasionally did not converge.

Statistics on the frequency of events (success, collision, sacrifice, and idle) and the average magnitude of utilization (decrease in disutility) for certain events and each of the update control schemes are presented in Table 2. Table 1 containing the statistics on update control for the numbering problem (see Chapter 5) has been reproduced above Table 2 for comparison. Note the strong similarity between these



The average global disutility and average number of iterations to find a solution is shown for the distributed iterative refinement arterial traffic light control algorithm with alternating, negotiated, urn, probabilistic, pure parallel, attenuated, and serial update control schemes (ALT, N, URN, PROB, P, ATT, and S, respectively). The local-view approximation introduced in the previous chapter is employed. Results of runs of SIGOP II (labeled SIGOP) are included for comparison. Standard deviations are also shown. The alternating, negotiated, urn, and probabilistic schemes produced faster algorithms than the SIGOP II algorithm, but did not find as good solutions. The serial and attenuated schemes produced slower algorithms, and the pure parallel scheme resulted in oscillation and very poor solutions.

Figure 20: Performance on ATLC Problem with Local View Approximation.

UPDATE CONTROL	DEMAND	FREQUENCY OF EVENTS GIVEN DEMAND			AVERAGE AND STAN. DEV. MAGNITUDE OF UTILIZATION	
		success	collision	sacrifice	success	collision
S	46%	11%	0%	89%	20+44	NA
SIGOP	27%	9%	0%	92%	49+91	NA
P	95%	0%	100%	0%	13+10	0+15
ATT	47%	5%	95%	0%	2+ 2	2+11
ALT	33%	87%	0%	13%	22+55	NA
N	33%	84%	0%	16%	26+63	NA
PROB	43%	32%	33%	36%	12+34	18+43
URN	31%	79%	2%	19%	25+60	3+ 5

Table 1: Statistics on Update Control for Numbering Problem

UPDATE CONTROL	DEMAND	FREQUENCY OF EVENTS GIVEN DEMAND			AVERAGE AND STAN. DEV. MAGNITUDE OF UTILIZATION	
		success	collision	sacrifice	success	collision
S	28%	12%	0%	89%	28+24	NA
SIGOP	26%	15%	0%	85%	29+37	NA
P	70%	3%	97%	0%	8+20	2+32
ATT	48%	10%	90%	0%	1+16	6+29
ALT	32%	60%	0%	40%	21+32	NA
N	34%	64%	0%	36%	19+30	NA
PROB	33%	24%	34%	42%	27+32	6+35
URN	27%	42%	21%	37%	19+30	3+35

Update control statistics gathered during arterial traffic light control experiments for the serial, SIGOP, pure parallel, attenuated, alternating, negotiated, probabilistic, and urn update control schemes (S, SIGOP, P, ATT, ALT, N, PROB, and URN, respectively) are shown. Table 1 on update control statistics gathered during number problem experiments where interaction is localized is included for comparison. Included are the percent of iterations a processing element has demand, the percent of iterations a processing element is successful, is involved in a collision, or sacrifices given it has demand, and the average magnitude and standard deviation of utilizations for successes and collisions. NA indicates that the statistic is not applicable.

Table 2: Statistics on Update Control for ATLC Problem.

statistics and the statistics gathered on the numbering problem. This similarity indicates that the non-local effects are handled reasonably well by the iterative nature of the algorithm, and local effects dominate. The effects of non-local interaction do, however, show.

The presence of non-local interaction tends to disrupt patterns in the demand process. That is, when interaction is localized a PE which successfully updates its control variable (without colliding with a neighbor) is always idle in the iteration immediately following its success, which presents its neighbors with an opportunity to update without interference. With non-local interaction, however, it is quite possible, even likely, that a successful PE will discover a change in platoon structure and wish to update its control again. This can be seen in the decrease in success rate and increase in sacrifice rates of the update control statistics for the negotiated and alternating schemes.

Table 3 presents statistics on the frequency, magnitude, and effect of errors in departure times encountered with the various update control schemes. By examining the errors in departure times given a success, one can get an idea of the frequency, magnitude, and effect of non-local interaction. Surprisingly, approximately a third of all departure times are in error given a success, due to changes in non-neighboring controls. The departure times are off by approximately 6.5 seconds when such an error occurs, and the error has the effect of increasing disutility by approximately 14 units. Note that this increase in disutility is usually more than offset by a decrease in disutility

<u>CONTROL SCHEME</u>	<u>ERRORS GIVEN A SUCCESS</u>			<u>ERRORS GIVEN A COLLISION</u>		
	<u>frequency</u>	<u>magnitude</u>	<u>effect</u>	<u>frequency</u>	<u>magnitude</u>	<u>effect</u>
S	0%	--	--	NA	NA	NA
SIGOP	0%	--	--	NA	NA	NA
P	36%	5.8	-13.0	89%	12.5	-39.0
ATT	45%	2.7	- 1.0	85%	4.5	- 4.0
ALT	32%	6.9	-15.0	NA	NA	NA
N	33%	6.5	-14.0	NA	NA	NA
PROB	19%	6.6	-14.0	76%	11.0	-33.0
URN	34%	6.7	-14.0	68%	9.7	-28.0

Statistics on errors in platoon departure times gathered during arterial traffic light control experiments with the local-view approximation and the serial, SIGOP, pure parallel, attenuated, alternating, negotiated, probabilistic, and urn update control schemes (S, SIGOP, P, ATT, ALT, N, PROB, and URN, respectively) are shown. Included are the frequency errors occur (percent of all platoon departure times), the average magnitude of the errors, and the average error in disutility calculations that result. NA indicates that the statistic is not applicable. Errors occurring during successes are attributable to non-local interaction.

Table 3: Statistics on Errors in Departure Times for ATLC Problem.

around the PE whose update caused the error since a locally successful change in a control that lowers local disutility is only rarely bad from a global perspective.

8.1.2 Performance with decoupling technique.

The worst-case decoupling technique introduced in Section 6.2 to be used in conjunction with the local-view approximation proved to be inappropriate for the traffic light control application. The decoupled problem was solved extremely quickly by the iterative refinement algorithms, but the quality of solutions was extremely poor. It seems the technique worked well in that interaction was localized, platoon structure updating was unnecessary, and convergence was monotonic, but the solutions of the resulting problem with localized interaction turn out to be poor solutions of the original problem with non-local interaction.

8.1.3 Performance with information gathering technique.

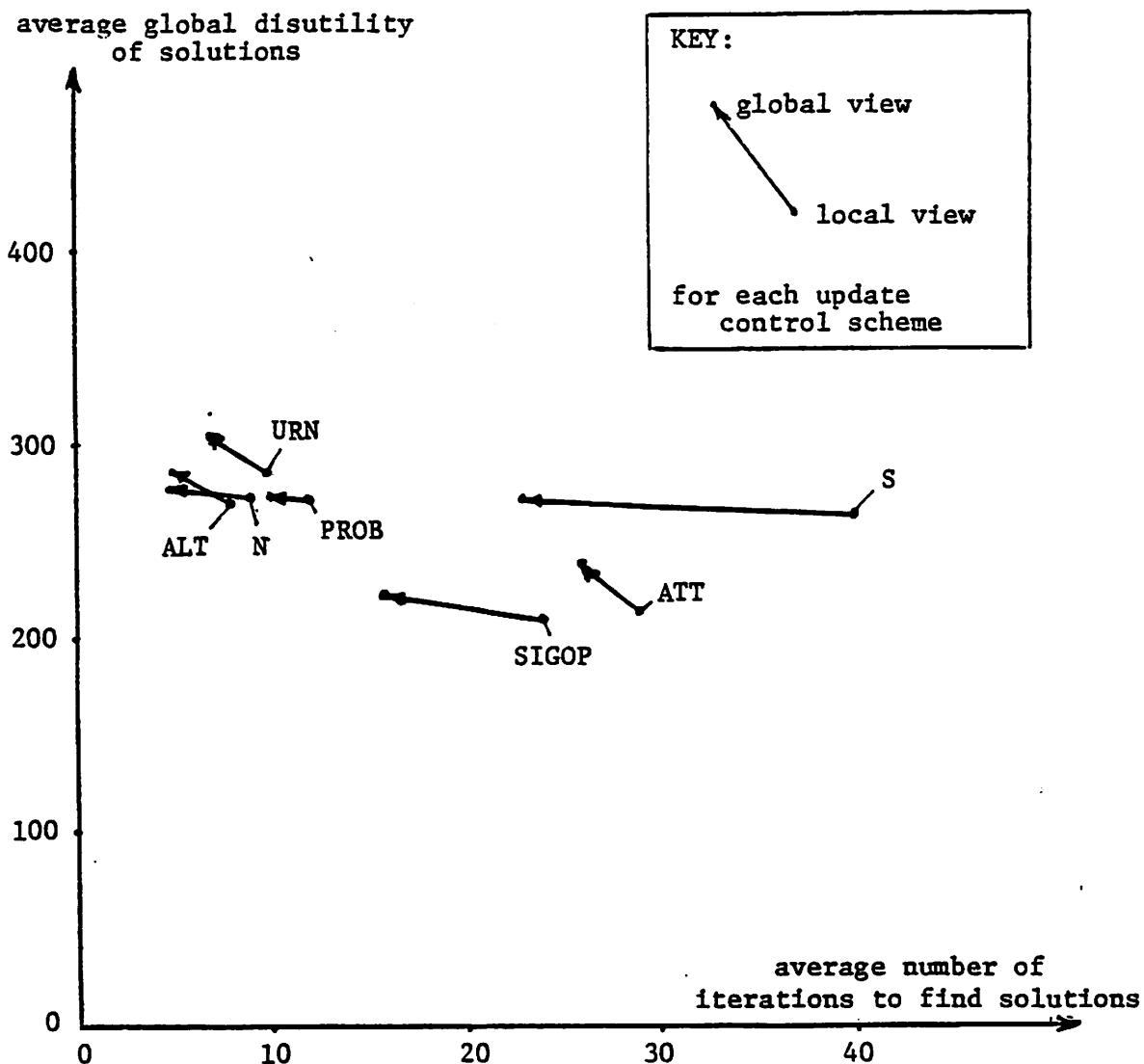
The information gathering technique discussed in Section 6.3 was also tested in a set of arterial traffic light control experiments. At each iteration, processing elements cooperate to obtain the global effects of changes in their controls. Unfortunately, this cooperative information gathering consumes time and space, and requires considerable communication; one iteration with information gathering takes approximately $N+1$ times as long as one iteration with the local-view approximation, where N is the number of PEs.

The performance of the iterative refinement algorithm with information gathering and a variety of update control schemes is compared to the performance of the iterative refinement algorithm with local-view approximation and the same update control schemes on the arterial traffic light control problem in Figure 21. In no case did information gathering improve upon the average quality of solutions found, and in all cases the algorithm took slightly fewer iterations (but considerably more time) with information gathering as opposed to the local-view approximation.

8.1.4 Discussion.

Best results on the arterial traffic light control problem were obtained with the local-view approximation and the alternating, negotiated, urn, and probabilistic update control schemes. Local effects dominate, and the algorithms easily recover from frequent errors introduced by non-local interaction. SIGOP II did find the best solutions, but the distributed iterative refinement algorithm with alternating, negotiated, urn, and probabilistic schemes were faster.

The decoupling technique to be used in conjunction with the local-view approximation to eliminate non-local interaction proved to be useless for the arterial traffic light control application. The information gathering technique did not result in any improvement over the local-view approximation in terms of the quality of solutions found, yet it cost a good deal in time, space, and communication.



The performance of the distributed iterative refinement arterial traffic light control algorithm with local versus global views of the effects of changing controls and the alternating, negotiated, urn, probabilistic, attenuated, and serial update control schemes (ALT, N, URN, PROB, ATT, and S, respectively) is illustrated. The SIGOP II algorithm (SIGOP) with local versus global views is also included for comparison. Surprisingly, the use of global views did not result in an improvement in the quality of solutions, but rather resulted in fewer iterations taken to find the solutions. Each iteration, however, takes approximately N times longer with the global view (where N is the number of processing elements), so it appears undesirable to obtain global views.

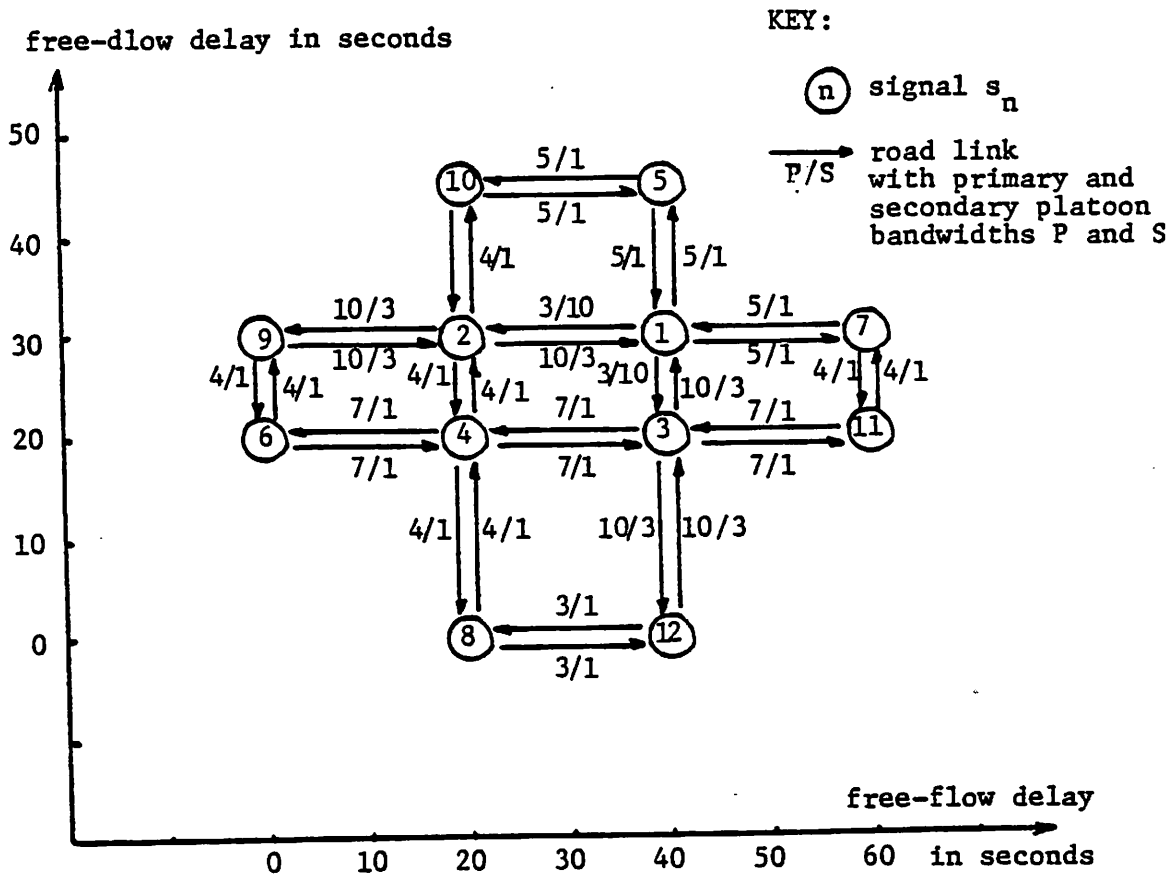
Figure 21: Performance with Local View vs. Global View

8.2 Network Traffic Light Control Experiments

In this section, results of experiments on network traffic light control are presented. The difference between these network traffic light control experiments and the preceding arterial traffic light control experiments is the use of a more complex, two platoon model and the increased number of neighbors each processing element has. The focus of these experiments is on the performance of the iterative refinement algorithm on networks with various topologies, noisy communication, and partial state updating.

8.2.1 Performance on small network.

The small network used for these experiments is shown in Figure 22. Values of a number of important parameters are shown. Each processing element is given a unique number. The numbering proceeds along a maximal spanning tree of links for use by the SIGOP II version. For each link there is a free-flow delay in seconds (indicated by the distance between PEs) and for each direction of travel there are primary and secondary platoons and associated bandwidths in seconds (numbers associated arrows). As with the arterial experiments, a 40 second cycle length and 20 second green durations are assumed, and results of twenty runs, each starting with a different prime, were averaged together.



The small network used in the first network traffic light control experiments is illustrated. The small network is made up of twelve processing elements, as was the artery used in previous experiments. The length of each link indicates the free-flow delay, or time it takes a platoon to traverse the link. A two-platoon traffic flow model is employed, so the primary and secondary platoon bandwidths are given for each link.

Figure 22: Small Network Used in First NTLC Experiments.

Note that there are 5 more links with the 12 PE small network than with the 12 PE artery, and secondary platoons have been added. For this reason, the average global disutility of solutions is higher for the small network experiments than for the arterial experiments. The random primes used for the small network experiments had an average global disutility of 1370 units vs. 947 units for the arterial experiments. The important differences are variations in speed of finding solutions, and the relative performance of each control scheme with respect to the performance of the others.

Overall performance of the iterative refinement algorithm with local-view approximation and various control schemes on the small network is compared to overall performance of similar algorithms on the artery, in Figure 23. The distributed iterative refinement algorithm with alternating, negotiated, urn, and probabilistic schemes performed well; the algorithm with negotiated scheme took no more time on the small network than it took on the artery, whereas the alternating, urn, and probabilistic schemes took slightly longer. The SIGOP II algorithm and distributed iterative refinement algorithm with attenuated schemes both took considerably longer to converge on the small network than on the artery. Surprisingly, the distributed iterative refinement algorithm with serial scheme was slightly faster on the small network.

The slight decrease in the speed of the algorithm on a network was expected with the negotiated, alternating, urn, and probabilistic schemes. This decrease is due to the additional competition for the right to update resulting from the increased number of neighbors each

The performance of the distributed iterative refinement traffic light control algorithm with local-view approximation and the alternating, negotiated, urn, probabilistic, attenuated, and serial update control schemes (ALT, N, URN, PROB, ATT, and S, respectively) on the artery versus the small network is illustrated. The SIGOP II algorithm (SIGOP) is also included for comparison. Solutions with higher disutility were obtained on the small network due to the increased number of links, although both the artery and the network contain 12 processing elements. The distributed refinement algorithm with alternating, negotiated, urn, and probabilistic update control schemes took very little additional time to find solutions on the network, whereas the distributed iterative refinement algorithm with attenuated update control scheme and SIGOP II algorithm took longer.

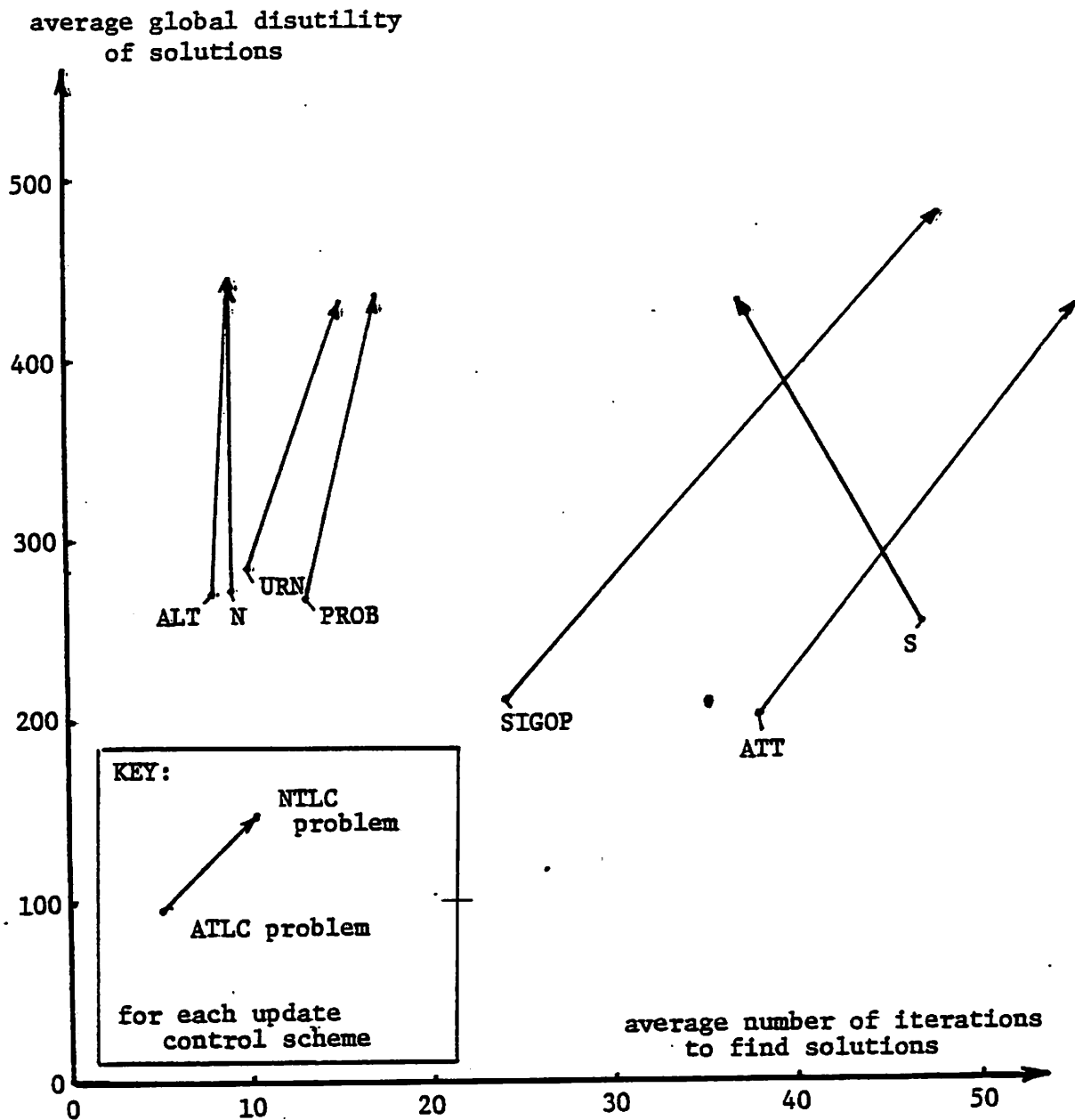


Figure 23: Performance on ATLC vs. NTLN Problem with 12 PEs.

processing element has. As it turns out, the probabilistic scheme is most affected by this increase in number of neighbors, and the negotiated scheme is able to compensate best for this increase.

The fact that the serial scheme performed better than SIGOP II on the small network was a surprise. It would appear that SIGOP II's maximal spanning tree heuristics get in the way when there are many flows in many directions. The distributed iterative refinement algorithm with attenuated scheme also took considerably more time arrive at solutions with the more complex network topology.

Results of runs with information gathering were carried out, but are not shown. At considerable expense, a small improvement in the quality of solutions (approximately 40 units) was realized with most schemes. As with the arterial experiments, a slight decrease in the number of iterations to find solutions was observed, but each iteration took much longer with information gathering.

8.2.2 Performance on large network.

The large network used in these experiments is shown in Figure 24 and represents a portion of the Washington D. C. central business district used for SIGOP II and CYRANO experiments [LIEB74, LIEB76]. Values of the important parameters (free-flow delays between intersections, platoon bandwidths, etc.) can be obtained from the figure. The network is made up of 38 signals and 91 links. As with the other traffic light control experiments, a 40 second cycle length and 20

The large network used in final traffic light control experiments is illustrated. The large network is made up of 38 processing elements and 91 links, and represents a portion of the Washington D. C. central business district. The length of each link indicates the free-flow delay, or time it takes a platoon to traverse the link. A two-platoon traffic flow model is employed, so the primary and secondary platoon bandwidths are specified for each link.

free-flow delay in seconds

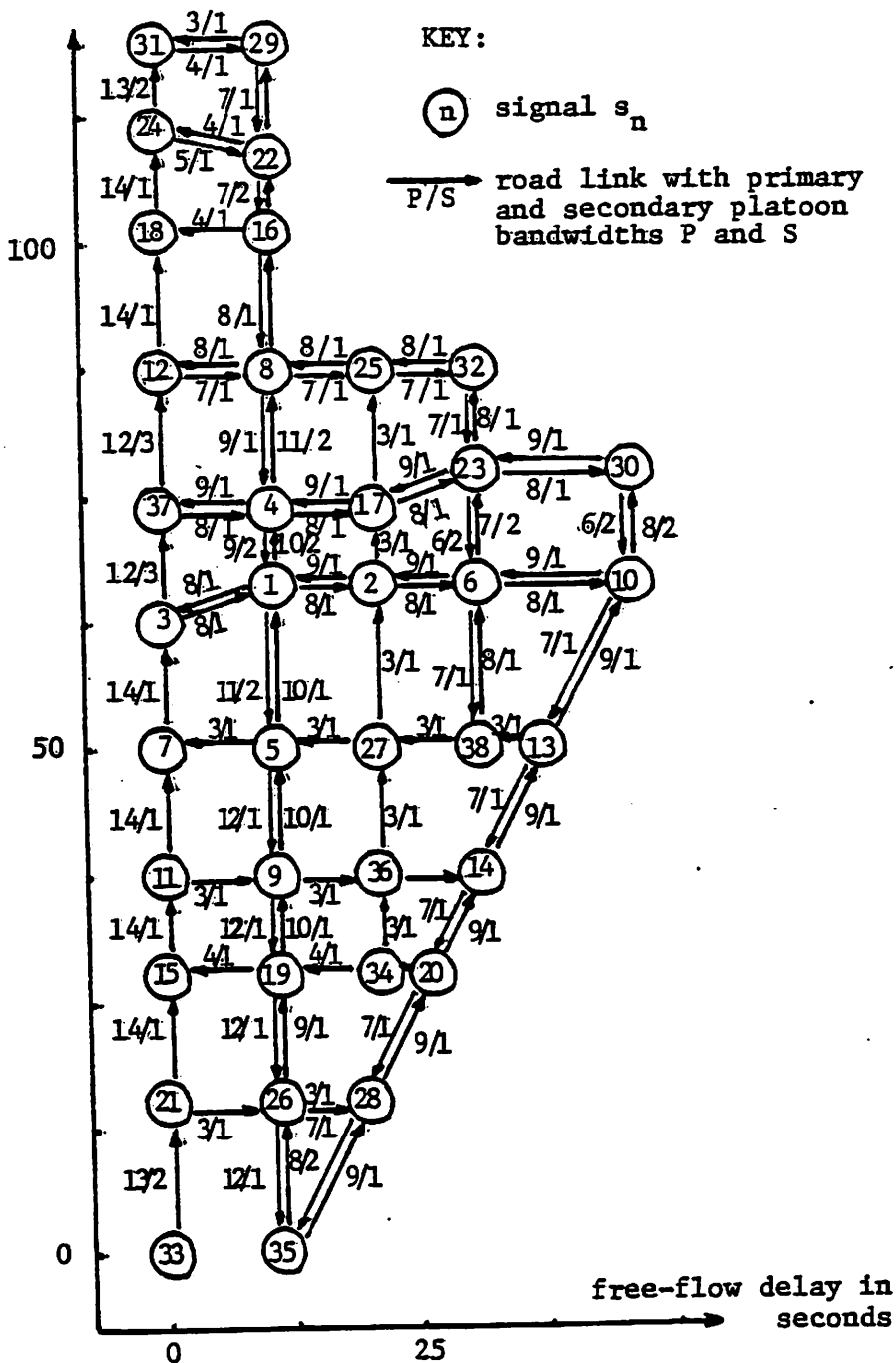


Figure 24: Large Network Used in Final NTLIC Experiments.

second green durations are assumed. Results of twenty runs, each starting with a different prime, were averaged together, except for experiments with noisy communication where due to time constraints only 4 runs were averaged together.

Performance of the distributed iterative refinement algorithm employing a local-view approximation and a variety of update control schemes on this large network traffic light control problem is depicted in Figure 25. The average global disutility of the 20 primes used in these experiments is approximately 4700; the global disutility of solutions is approximately 1400. The distributed iterative refinement algorithm with negotiated, alternating, urn, and probabilistic versions are clearly superior to the SIGOP II algorithm. Not only is the distributed iterative refinement algorithm 5 to 15 times faster than SIGOP II, but it generally finds better solutions. On a VAX 11/780 the simulated distributed iterative refinement algorithm with negotiated update control scheme took almost 10 hours to solve the 20 runs. This breaks down to 48 seconds per processing element for a single run of approximately 40 iterations, and fits easily into the 15 minute control period of the traffic light control application.

Note that the large network is no more tightly coupled than the small network introduced in the previous section. Thus, the additional time versions N, ALT, URN, and PROB take to solve the network traffic light control problem with large network is for the most part attributable to non-local interaction since arterial experiments showed an insensitivity to changes in network size. The number of iterations

The performance of the distributed iterative refinement network traffic light control algorithm with local-view approximation and negotiated, alternating, urn, probabilistic, and attenuated update control schemes (N, ALT, URN, PROB, and ATT, respectively) on the large network is illustrated. Performance of the SIGOP II algorithm (SIGOP) is also shown for comparison. The fastest distributed iterative refinement algorithms employ the negotiated, alternating, urn, and probabilistic update control schemes, and found better solutions than the SIGOP II algorithm. The distributed iterative refinement algorithm with attenuated update control scheme was slow and found the poorest solutions.

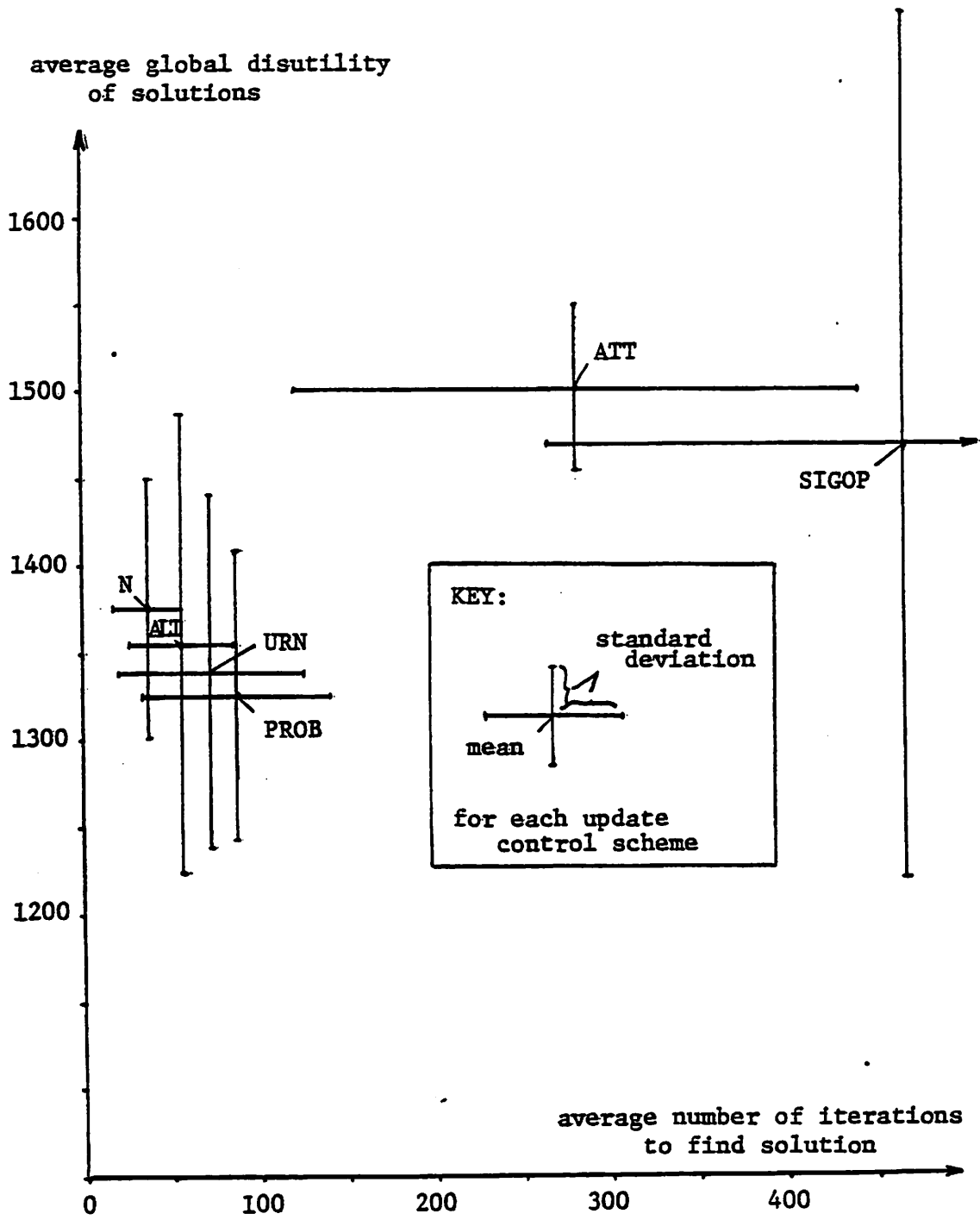


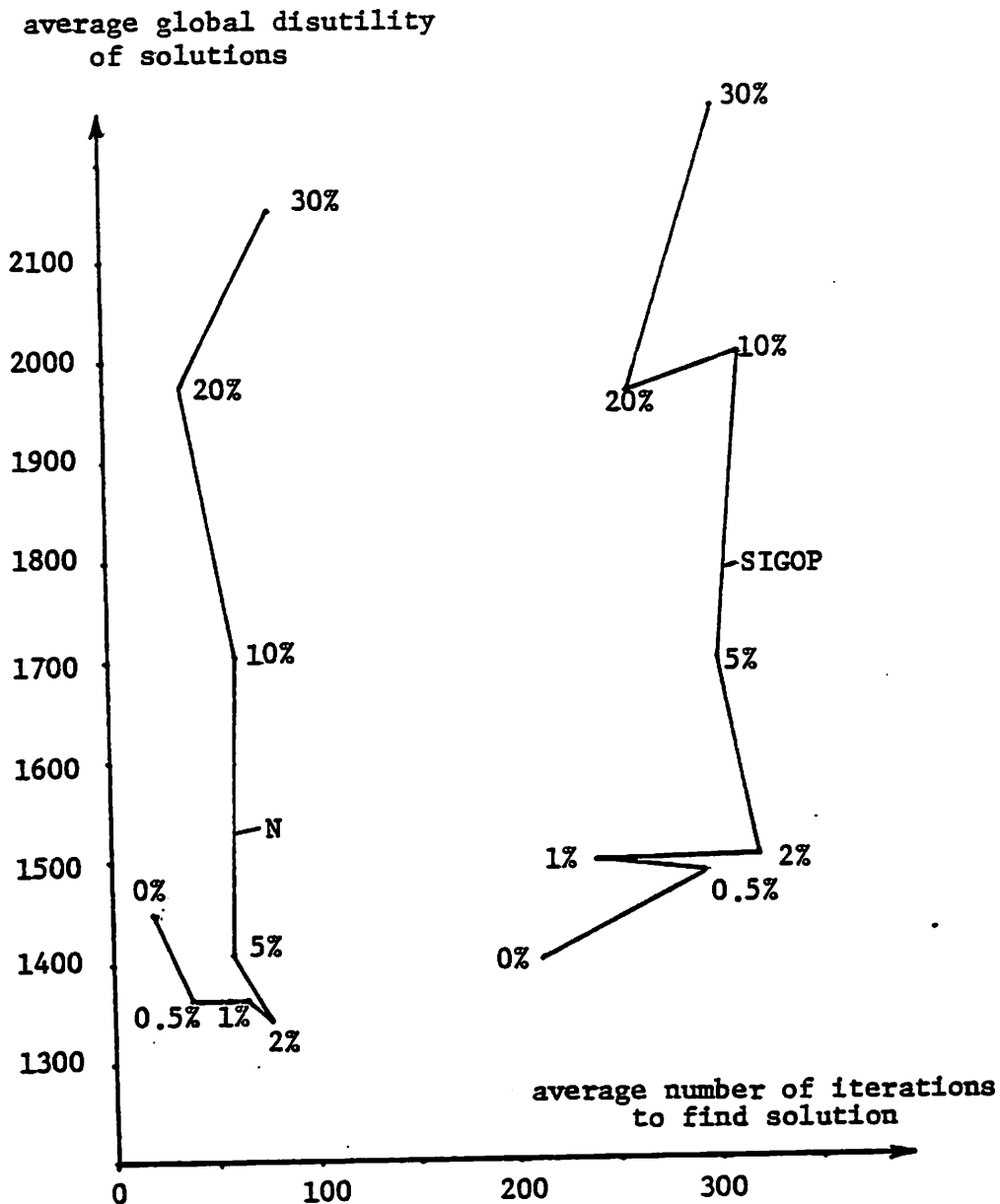
Figure 25: Performance with Local View on Large NTLC Problem.

to find solutions for SIGOP II, on the other hand, grows quickly as network size increases regardless of the topology.

8.2.3 Performance with communication errors.

Sensitivity of the distributed iterative refinement technique to communication errors is revealed in another set of experiments on the large network. Noise was added to a percentage of all controls and states communicated to simulate communication errors. When an error is added, a completely random control or state is substituted for the correct control or state. The results of these experiments are depicted in Figure 26. The traffic light control iterative refinement algorithm with local-view approximation described in Chapter 7 was employed for these experiments with a variety of the update control schemes.

With the negotiated update control scheme the algorithm is affected by the addition of communication errors in the following way. With very low error rates (up to 5 percent of all communications) the algorithm finds better solutions, but takes up to 4 times longer to find these better solutions. However, if solutions are desired of the quality found when no noise is added, the algorithm takes little or no additional time. This behavior is illustrated in Figure 27. With more than 5 percent communication errors the quality of solutions gradually degrades without a further increase in the number of iterations taken to find solutions.



The performance in the presence of communication errors of the distributed iterative refinement network traffic light control algorithm with local-view approximation and negotiated update control scheme (N) is compared to the performance of the SIGOP II algorithm (SIGOP). Points are plotted for the cases of 0, 0.5, 1, 2, 5, 10, 20, and 30 percent communication errors. The distributed iterative refinement algorithm actually benefits from the presence of up to 5 percent communication errors in that it finds better solutions than when there is no noise, whereas performance with SIGOP II degrades immediately.

Figure 26: Sensitivity to Communication Errors.

Individual runs of the distributed iterative refinement network traffic light control algorithm with local-view approximation and negotiated update control scheme are shown with 0, 1, and 5 percent communication errors. Notice how the presence of occasional communication errors results in a longer search that results in a better solution than is found when there are no communication errors, but if an improvement in the quality of solutions is not desired, little additional time is required.

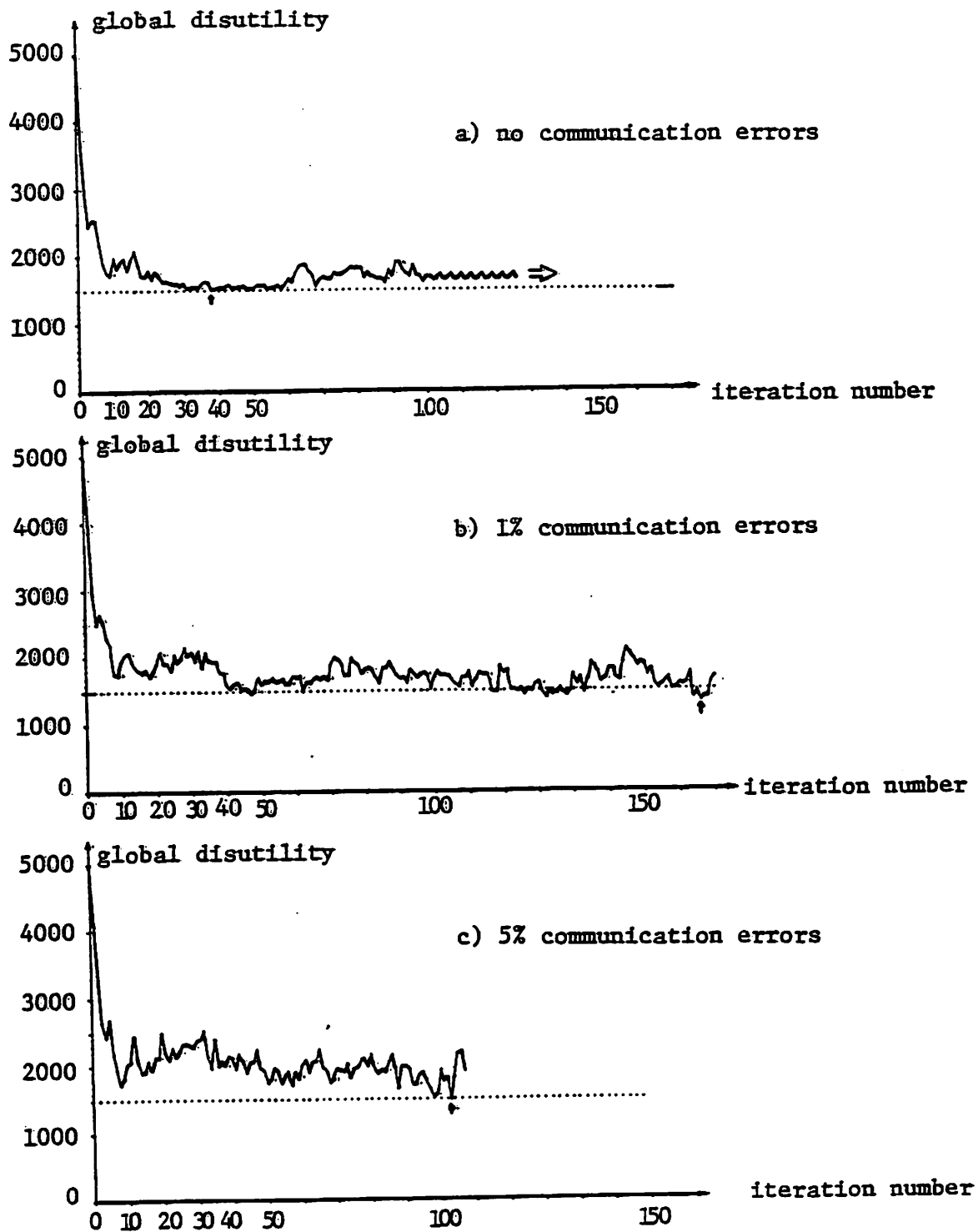


Figure 27: Individual Runs with 0, 1 and 5 Percent Communication Errors.

The distributed iterative refinement algorithm with urn, probabilistic, and alternating update control schemes react similarly to noisy communication. The performance of the SIGOP II algorithm, on the other hand, degrades when even a little noise is added to communications. This is most likely due to the fact that there is a longer waiting period between successive refinements to a control component with the sequential SIGOP II algorithm than with the distributed iterative refinement algorithm, so it takes longer to resolve errors. Thus, the distributed iterative refinement network traffic light control algorithm appears to be much more robust in the presence of communication errors than SIGOP II.

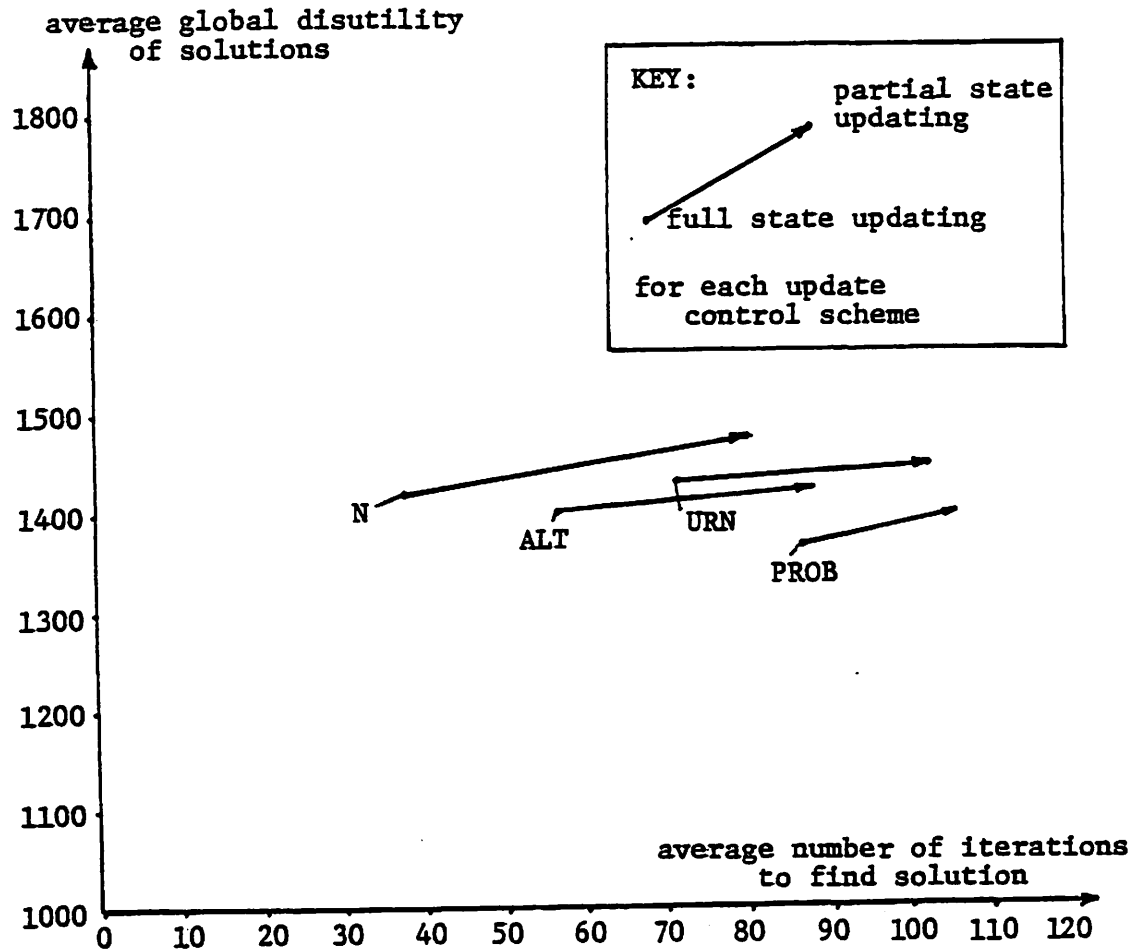
8.2.4 Performance with partial state updating.

The last experiment with the distributed iterative refinement network traffic light control algorithm and the large network involves partial state updating. State updating is the process of arranging state variables (platoon departure times) to be consistent with the current solution vector (signal timings). It is a cooperative computation involving all processing elements, and is performed during the simulation phase of the amended distributed iterative refinement algorithm described in Section 7.2.

Partial state updating involves performing only one state update during the simulation phase rather than N state updates, where N is the total number of processing elements, and cuts down on communication. Unfortunately, partial state updating introduces uncertainty into the problem solving, since state variables may be errorful. As is illustrated in Figure 28, the main effect of this error on the distributed iterative refinement algorithm with negotiated, alternating, urn, and probabilistic update control schemes is to slow it down; little accuracy is lost. This happens because once the algorithm nears a solution the controls change little, and after a few iterations most of the error in the state variables is resolved.

8.2.5 Discussion.

Results of network traffic light control experiments support earlier results indicating a significant speed-up of a distributed iterative refinement algorithm over a sequential iterative refinement algorithm which grows as the size of the network increases. The negotiated, alternating, urn, and probabilistic schemes were, again, the viable parallel update control schemes. The negotiated scheme was the fastest of these, followed by the alternating, urn, and probabilistic schemes in order. SIGOP II was not only slower than the distributed iterative refinement algorithm, but appears to lose its advantage of finding better solutions when a network is used instead of an artery.



The performance of the iterative refinement network traffic light control algorithm with local-view approximation and negotiated, alternating, urn, and probabilistic update control schemes (N, ALT, URN, and PROB, respectively) is illustrated with full and partial state updating. Some accuracy is lost and the algorithms take considerably longer to find solutions if only partial state updating is utilized due to the inaccuracies introduced with partial state updating.

Figure 28: Performance with Partial vs. Full State Updating.

The distributed iterative refinement algorithm worked very well with the local-view approximation, and lost little accuracy when faced with infrequent communication errors; its performance slowly degrades as the communication error increase in frequency. Furthermore, partial state updating is tolerated with little loss in accuracy. This means that not only is communication neighbors-only, but communication bandwidth requirements are also flexible, and may be adjusted to fit real-world situations.

CHAPTER IX

CONCLUSIONS

The primary objective of the research presented in this thesis has been to demonstrate that a distributed problem solving system can be designed to deal with uncertainty arising from working in parallel on interacting subproblems, using approximations to limit communication, and sustaining communication errors. This objective has been achieved. A general method of distributed problem solving based on a centralized iterative refinement technique has been presented, which resembles a distributed gradient algorithm; however, line searches are employed to consider arbitrary rather than incremental changes in control or interpretation with the method presented. This difference is particularly useful for problems with complex (discontinuous, non-differentiable, multi-modal, discrete, etc.) objective functions.

The ability of the distributed iterative refinement technique to handle considerable interaction among subproblems is dependent on the use of an appropriate update control scheme. However, an update control scheme is needed only to control simultaneous updates between neighboring processing elements; the iterative refinement algorithm easily tolerates uncertainty introduced by non-local interaction. A variety of decentralized update control schemes were examined as a means of limiting collisions (simultaneous updates) including alternating, negotiated, attenuated, probabilistic, and urn schemes.

When interaction is localized, a local view of the effects of changing a control is equivalent to the global view. When interaction is not localized and communication is limited, non-local information requirements must be considered. State variables were introduced to permit processing elements to determine their local disutility with information obtained from neighboring processing elements only, and three approaches to handling the need for non-local information to evaluate the effects of changing controls were described.

Through experiments on a simple numbering problem and network traffic light control, empirical results have been obtained. These results are summarized in the next section. The distributed iterative refinement network traffic light control algorithm was based on an existing network traffic light control algorithm, SIGOP II. These experiments included a comparison of the update control schemes with a variety of network sizes and topologies. The local-view approximation, worst-case decoupling assumption, and cooperative information gathering techniques were also tested in these experiments. Finally, experiments with communication errors and partial state updating were included to further test the distributed iterative refinement algorithm's tolerance of uncertainty.

As a by-product of this research, a general balance principle and adaptive balancing algorithm for Pareto-optimal multi-access control of distributed resources has been developed. A summary of this research and open questions will follow a summary of the experimental results.

9.1 Summary of Experimental Results

The negotiated, alternating, probabilistic, and urn update control schemes produced a faster algorithm than the centralized, sequential SIGOP II algorithm. On the large network traffic light control problem (with 38 processing elements) the distributed versions were 5-15 times as fast with no loss in accuracy. Of these update control schemes, the negotiated scheme was most effective, followed in turn by the alternating, urn, and probabilistic schemes. Use of the attenuated update control scheme led to slow convergence, occasional oscillation, and poor solutions. As network size and/or connectivity increases, all versions of the distributed iterative refinement algorithm take longer, but are significantly less sensitive to changes in network size or topology than SIGOP II.

The best results with traffic light control experiments where there is non-local interaction were obtained using the local-view approximation. Although some improvement in the quality of solutions was occasionally realized with the information gathering technique, it required considerable time and communication to operate. This demonstrates that processing elements not only can tolerate the uncertainty introduced by assuming a local view of the effects of changing controls, but that it can be worthwhile to do so. Although interaction can be limited with a worst-case interaction assumption, experiments indicate that considerable accuracy is lost.

The distributed iterative refinement algorithm with local-view approximation and negotiated, alternating, urn, and probabilistic update control schemes proved to be quite tolerant of infrequent communication errors (up to 5% of all communications). Furthermore, partial state updating is tolerated. This means the distributed iterative refinement algorithm is quite flexible in its communication requirements. Because of the use of the local-view approximation the algorithm easily meets the time constraints of the traffic light control application.

9.2 Access Control

In Chapter 4, the problem of simultaneous updating of control variables was viewed as a problem of access control of shared resources. This led to an investigation of a general problem of decentralized access control of distributed resources, and the extension of work by Kleinrock and Yemini [KLEI78, YEMI78, YEMI79]. This extension allows one to add a penalty for collisions (simultaneous attempts to access shared resources). The results are summarized below.

A simple, yet powerful balance principle for optimal, multi-access control of distributed resources has been derived. The balance principle is based on Pareto optimality and covers a broad range of objective functions; the objective functions may include a penalty for collisions and other events. Providing the probability that a processing element will attempt to access shared resources given the processing element has demand can be expressed as a function of the

access control policy, the balance principle is appropriate.

Two important access control schemes of this type, the probabilistic and urn schemes, were examined in this thesis. The urn scheme is theoretically better than the probabilistic scheme because it performs better at high demand. A high penalty for collisions make the probabilistic and urn schemes less effective because processing elements must be more cautious to avoid collisions.

The theoretical performance of the probabilistic and urn schemes were also compared to some other well-known schemes, the serial and ideal (perfect) schemes. While a serial access control scheme (or better, an alternating scheme) is reasonably good for high demands, oscillation is a risk and the scheme is less effective for low demand; an urn scheme is generally better than a serial scheme because it is more effective for low demand and equivalent for high demand. A negotiated scheme is ideal, and is better than a probabilistic or urn scheme if there is sufficient communication to support the scheme. The benefits of a negotiated scheme are greater the more processing elements there are in neighborhoods; however, communication requirements increase as the number of processing elements increase.

The balance principle can be used in a number of ways. Given constant and known demand statistics, the balance principle can be used to derive an optimal, fixed policy. When demand varies over time, the balance principle can be used in a conventional manner (see Section 4.2.1) to determine the optimal policy given an estimate of the demand statistics. Alternatively, the balance principle can be used in an

adaptive balancing algorithm as described in Section 4.2.2.

The adaptive balancing algorithm is appropriate for quasi-static environments and is completely decentralized. Communication requirements are modest given a processing element can observe or cheaply obtain utilization statistics for shared resources to which the processing element has access. And, demand and penalty statistics need not be known or estimated as only utilization statistics are required by the algorithm.

The adaptive balancing algorithm has been tested in a controlled environment as well as in the iterative refinement update control environment. Results of the controlled environment tests show that the adaptive balancing algorithm works properly (i.e., that it settles on predicted theoretical pareto optima). These results are included in Appendix A. Results of the iterative refinement update control tests also show that the adaptive balancing algorithm works well even when the demand process is not independent each iteration and there are random payoffs for collisions as well as successes.

9.3 Open Questions

Below, a few open questions concerning the distributed iterative refinement algorithm developed in this thesis are briefly discussed. These questions concern the distributed iterative refinement algorithm's potential for asynchronous operation and its extensibility.

For example, the algorithm's tolerance to error, inconsistencies, and incompleteness might make it possible for it to be run asynchronously. The algorithm's performance with the probabilistic and urn update control schemes demonstrates the algorithm's ability to tolerate collisions, which would be inevitable if run asynchronously. Furthermore, the algorithm proved to be quite tolerant of the use of limited, local views and partial state updating. This eliminates two global computations which require processing elements to operate synchronously. The last global computation, periodic global evaluations of solutions, might also be eliminated, although solutions will not always be as good.

A second question is whether the algorithm will work as well for real-time network traffic light control like CYRANO [LIEB74] with an even more complex platoon flow model and variable splits, as it worked for steady-state network traffic light control with a two-platoon model and fixed splits. The most crucial factor will probably be one of speed, since these three extensions all enlarge the local search spaces.

A final question is whether the algorithm can easily be adapted for other applications. Some of these potential applications are distributed reservoir system water-level control (where the flow of water at multiple dams is controlled), distributed waste treatment control (where the treatment of water in the various parts of the plant is controlled), or distributed air traffic control. The properties of the interaction between processing elements' local subproblems will undoubtedly vary, and it would be useful to know how the variations

affect the distributed iterative refinement algorithm.

SELECTED BIBLIOGRAPHY

- [ATHA78] Michael Athans.
On Large-Scale Systems and Decentralized Control.
IEEE Transactions on Automatic Control AC-23(2):105-106, April 1978.
- [BAUD78] Gerard M. Baudet.
Asynchronous Iterative Methods for Multiprocessors.
Journal of the Association for Computing Machinery 25(2):226-244, April 1978.
- [BELL62] Richard E. Bellman and Stuart E. Dreyfus.
Applied Dynamic Programming.
Princeton University Press, 1962.
- [BROO79] Richard S. Brooks and Victor R. Lesser.
Distributed Problem Solving Using Iterative Refinement.
Technical Report 79-14, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, May 1979.
- [CAPE79] John I. Capetanakis.
Tree Algorithms for Packet Broadcast Channels.
IEEE Transactions on Information Theory IT-25(5):505-515, September 1979.
- [CLIN79] T. B. Cline and R. E. Larson.
Decision and Control in Large-Scale Systems Via Spatial Dynamic Programming.
In R. E. Larson, editor, Tutorial: Distributed Control, pages 289-293, IEEE, October 1979.
- [CONT65] S. D. Conte and Carl de Boor.
Elementary Numerical Analysis: an Algorithmic Approach
McGraw-Hill, 1972.
- [COOP70] Leon N. Cooper and David Steinberg.
Introduction to Methods of Optimization
W. B. Saunders, Philadelphia, Pennsylvania, 1970.
- [DAVI76] Larry S. Davis and Azriel Rosenfeld.
Applications of Relaxation Labeling, 2: Spring-Loaded Template Matching.
Proceedings of the Third International Joint Conference on Pattern Recognition, pages 591-597, IEEE, November 1976.

- [DAVI80] Randy Davis.
Report on the Workshop on Distributed AI.
SIGART Newsletter (73):42-43, October 1980.
- [DIMM76] D. G. Dimmler, N. Greenlaw, M. A. Kelley, D. W. Potter, S. Rankowitz, and F. W. Stubblefield.
Brookhaven Reactor Experiment Control Facility: A Distributed Function Computer Network.
IEEE Transactions on Nuclear Science NS-23(1):398-405, February 1976.
- [DOT76] U. S. Department of Transportation / Federal Highway Administration.
Traffic Control Systems Handbook.
Available as publication 050-001-00114-4 from U. S. Government Printing Office, Washington, D. C. 20402, June 1976.
- [ERMA79] Lee D. Erman and Victor R. Lesser.
The Hearsay-II System: A Tutorial.
In W. A. Lee, editor, Trends in Speech Recognition, chapter 16, Prentice-Hall, 1979.
- [ERMA80] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy.
The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty.
Computing Surveys 12(2):213-253, June 1980.
- [GALL77] Robert Gallager.
A Minimum Delay Routing Algorithm Using Distributed Computation.
IEEE Transactions on Communication COM-25:73-85, January 1977.
- [GART75] Nathan H. Gartner and John D. C. Little.
Generalized Combination Method for Area Traffic Control.
Transportation Research Record, report 531, pages 58-69, 1975.
- [HANS78] Allen R. Hanson and Edward M. Riseman.
Segmentation of Natural Scenes.
In Allen R. Hanson and Edward M. Riseman, editors, Computer Vision Systems, pages 129-163, Academic Press, 1978.
- [HEWI77] Carl Hewitt.
Viewing Control Structures as Patterns of Passing Messages.
Artificial Intelligence 8(3):323-364, fall 1977.

- [HILL65] J. A. Hillier.
Glasgow's Experiment in Area Traffic Control.
Two parts. Traffic Engineering and Control 7(8):502-509,
December 1965 and 7(9):569-571, January 1966.
- [HILL69] J. A. Hillier and J. Holroyd.
Area Traffic Control in Glasgow, A Summary of Results from
Four Control Schemes.
Traffic Engineering and Control 11(5):220-227, 1969.
- [HOLL81] Elmar Holler.
Multiple Copy Update.
In B. W. Lampson, M. Paul, and H. J. Siegert, editors,
Distributed Systems - Architecture and Implementation,
chapter 13, Springer-Verlag, 1981.
- [HUDD69] K. W. Huddart and E.D. Turner.
Traffic Signal Progressions - GLC Combination Method.
Traffic Engineering and Control 11:320-327, November 1969.
- [KILM69] W. L. Kilmer, W. S. McCulloch, and J. Blum.
A Model of the Vertebrate Central Command System.
International Journal of Man-Machine Studies 1:279-309, 1969.
- [KIMB81] Stephen R. Kimbleton, Pearl Wang, and Butler W. Lampson.
Applications and Protocols.
In B. W. Lampson, M. Paul, and H. J. Siegert, editors,
Distributed Systems - Architecture and Implementation,
chapter 14, Springer-Verlag, 1981.
- [KLEI78] Leonard Kleinrock and Yechiam Yemini.
An Optimal Adaptive Scheme for Multiple-Access Broadcast
Communication.
Proceedings of the 1978 International Conference on
Communications, pages 7.2.1-7.2.5, IEEE, June 1978.
- [KLEI79] Leonard Kleinrock.
On Flow Control in Computer Networks.
Proceedings of the 1978 International Conference on
Communications, pages 27.2.1-27.2.5, IEEE, June 1978.
- [LARS70] R. E. Larson and A. J. Korsak.
A Dynamic Programming Successive Approximations Technique with
Convergence Proofs.
Automatica 6:245-260, March 1970.

- [LARS79a] Robert E. Larson, Paul L. McEntire, and Thomas L. Steding.
Foundations of Spatial Dynamic Programming.
Proceedings of the First International Conference on
Distributed Computing Systems, pages 572-577, IEEE,
October 1979.
- [LARS79b] Robert E. Larson.
A Survey of Optimization Techniques.
In R. E. Larson, editor, Tutorial: Distributed Control,
chapter 1, IEEE, October 1979.
- [LARS79c] Robert E. Larson.
A Survey of Distributed Control Techniques.
In R. E. Larson, editor, Tutorial: Distributed Control,
chapter 5, IEEE, October 1979.
- [LELA81] Gerard LeLann.
Motivations, Objectives, and Characterization of Distributed
Systems.
In B. W. Lampson, M. Paul, and H. J. Siegert, editors,
Distributed Systems - Architecture and Implementation,
chapter 1, Springer-Verlag, 1981.
- [LENA75] Douglas B. Lenat.
Beings: Knowledge as Interacting Experts.
Advance Papers of the Fourth International Joint Conference on
Artificial Intelligence, pages 126-133, 1975.
Zerographic or microfilm copies available from University
Microfilms, 300 North Zeeb Road, Ann Arbor, Michigan
48106.
- [LESS77] Victor R. Lesser and Lee D. Erman.
A Retrospective View of the Hearsay-II Architecture.
Proceedings of the Fifth International Joint Conference on
Artificial Intelligence, pages 790-800, 1977.
Available from Department of Computer Science, Carnegie-Mellon
University, Pittsburgh, Pennsylvania 15213.
- [LESS80] Victor R. Lesser and Lee D. Erman.
An Experiment in Distributed Interpretation.
IEEE Transactions on Computers C-29(12):1144-1163, December
1980.
- [LESS81] Victor R. Lesser and Daniel D. Corkill.
Functionally Accurate, Cooperative Distributed Systems.
IEEE Transactions on Systems, Man, and Cybernetics
SMC-11(1):81-96, January 1981.

- [LESS82] Victor R. Lesser, Daniel D. Corkill, Jasmina Pavlin, Larry Lefkowitz, Eva Hudlicka, Richard S. Brooks, and Scott Reed
A High-Level Simulation Testbed for Cooperative Distributed Problem-Solving.
Technical Report 81-16, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, revised 1982.
- [LIEB74] E. B. Lieberman, W. R. McShane, and R. B. Goldblatt.
Variable Cycle Signal Timing Program, Volume 3: CYRANO: Cycle-Free Responsive Algorithms for Network Optimization: Moderate, Congested, and Light Flow Regimes.
Technical report FHWA-RD-74, KLD Associates, Huntington, New York 11743, May 1974.
Available as publication PB-241 720 from National Technical Information Service, Springfield, Virginia 22151.
- [LIEB76] Edward B. Lieberman and James L. Woo.
SIGOP II: A New Computer Program for Calculating Optimal Signal Timing Patterns.
Transportation Research Record, report 596, pages 16-21, 1976.
- [LUCE57] R. D. Luce and H. Raiffa.
Games and Decisions.
John Wiley and Sons, 1957.
- [LUM79] M. Lum, L. L. Kinney, and K. S. P. Kumar.
Feasibility of a Distributed Computer Traffic Control System.
Department of Electrical Engineering, University of Minnesota, Minneapolis, Minnesota, 1979.
- [MERR81] P. E. Merritt, Jr. and F. S. Ozdemir.
Distributed Control/Support Architecture for Direct Wafer Write E-beam Lithography Systems: Throughput Considerations.
Journal of Vacuum Science and Technology, 19(4):998-1000, November/December 1981.
- [METC76] R. M. Metcalfe and D. R. Boggs.
Ethernet: Distributed Packet Switching for Local Computer Networks.
Communications of the ACM, 19(7):395-404, July 1976.
- [NILS80] Nils J. Nilsson.
Principles of Artificial Intelligence.
Tioga, Palo Alto, California, 1980.

- [PEEB78] R. Peebles and E. Manning.
System Architecture for Distributed Data Management.
Computer 11:40-47, 1978.
- [RISE77] Edward M. Riseman and Michael A. Arbib.
Computational Techniques in the Visual Segmentation of Static
Scenes.
Computer Graphics and Image Processing 6(3):221-276, June
1977.
- [ROBE69] D. I. Robertson.
TRANSYT: A Traffic Network Study Tool.
RRL Report LR253, Road Research Laboratory, Crowthorne,
England, 1969.
- [ROSE76] Azriel Rosenfeld, Robert A. Hummel, and Steven W. Zucker.
Scene Labeling by Relaxation Operations.
IEEE Transactions on Systems, Man, and Cybernetics
SMC-6(6):420-433, June 1976.
- [SAND78] Nils R. Sandell Jr., Pravin Varaiya, Michael Athans, and
Michael G. Safanov.
Survey of Decentralized Control Methods for Large Scale
Systems.
IEEE Transactions on Automatic Control AC-23(2):108-128, April
1978.
- [SCHO76] James D. Schoeffl and Charles W. Rose.
Distributed Computer Intelligence for Data Acquisition and
Control.
IEEE Transactions on Nuclear Science NS-23(1):38-54, February
1976.
- [SEGA76] Adrian Segall
Recursive Estimation from Discrete-Time Point Processes.
IEEE Transactions on Information Theory IT-22(4):422-431, July
1976.
- [SHOC80] J. R. Shoch and J. A. Hupp.
Measured Performance of an Ethernet Local Network.
Communications of the ACM 23(12):711-721, December 1980.
- [SMIT80] Reid G. Smith.
The Contract Net Protocol: High-level Communication and
Control in a Distributed Problem Solver.
IEEE Transactions on Computers C-29(12):1104-1113, December
1980.

- [SMIT81] Reid G. Smith and Randall Davis.
Frameworks for Cooperation in Distributed Problem Solving.
IEEE Transactions on Systems, Man, and Cybernetics
SMC-11(1):61-70, January 1981.
- [TENN79] Robert. R. Tenny.
Distributed Decision Making Using a Distributed Model.
Ph. D. thesis, Massachusetts Institute of Technology, June
1979.
Available as Technical Report LIDS-TD-938, Laboratory for
Information and Decision Systems, Massachusetts Institute
of Technology, Cambridge, Massachusetts 02139.
- [TENN81a] Robert R. Tenny and Nils R. Sandell, Jr.
Structures for Distributed Decisionmaking.
IEEE Transactions on Systems, Man, and Cybernetics
SMC-11(8):517-527, August 1981.
- [TENN81b] Robert R. Tenny and Nils R. Sandell, Jr.
Strategies for Distributed Decisionmaking.
IEEE Transactions on Systems, Man, and Cybernetics
SMC-11(8):527-538, August 1981.
- [THUR81a] Kenneth J. Thurber.
Interprocess Communication Layer: Introduction.
In B. W. Lampson, M. Paul, and H. J. Siebert, editors,
Distributed Systems - Architecture and Implementation,
chapter 3, Springer-Verlag, 1981.
- [THUR81b] Kenneth J. Thurber.
Hardware Interconnection Technology.
In B. W. Lampson, M. Paul, and H. J. Siebert, editors,
Distributed Systems - Architecture and Implementation,
chapter 4, Springer-Verlag, 1981.
- [TOBA80] Fouad A. Tobagi.
Multiaccess Protocols in Packet Communication Systems.
IEEE Transactions on Communication COM-28(4):468-488, April
1980.
- [TRC66] Traffic Research Corporation
SIGOP: Traffic Signal Optimization Program, A Computer Program
to Calculate Optimal Coordination in a Grid Network of
Synchronized Traffic Signals.
Traffic Research Corporation, New York, 1966.

- [WAGN71] F. A. Wagner, F. C. Barnes, and D. L. Gerlough.
Improved Criteria for Traffic Signal Systems in Urban
Networks.
National Cooperative Highway Research Program report 124,
Highway Research Board, Washington D.C., 1971.
- [WALT75] David Waltz.
Understanding Line Drawings of Scenes with Shadows.
In Patrick H. Winston, editor, The Psychology of Computer
Vision, pages 19-91, McGraw-Hill, 1975.
- [YEMI78] Yechiam Yemini.
On Channel Sharing in Discrete-Time, Packet-Switched,
Multi-Access Broadcast Communication.
Ph. D. dissertation, University of California at Los Angeles,
1978.
- [YEMI79] Yechiam Yemini and Leonard Kleinrock.
On a General Rule for Access Control or, Silence is Golden...
In "Distributed Sensor Networks," ISI Working Paper 12, USC
Information Sciences Institute, Marina del Rey,
California, April 1979.
- [YEMI80] Yechiam Yemini and Leonard Kleinrock.
Broadcasting in an Urn: An Optimal Adaptive Multiaccess
Scheme.
Private communication, USC Information Sciences Institute,
Marina del Rey, California, 1980.
- [ZUCK77] Steven W. Zucker, Robert A. Hummel, and Azriel Rosenfeld.
An Application of Relaxation Labeling to Line and Curve
Enhancement.
IEEE Transactions on Computers C-26(4):394-403, April 1977.

A P P E N D I X A

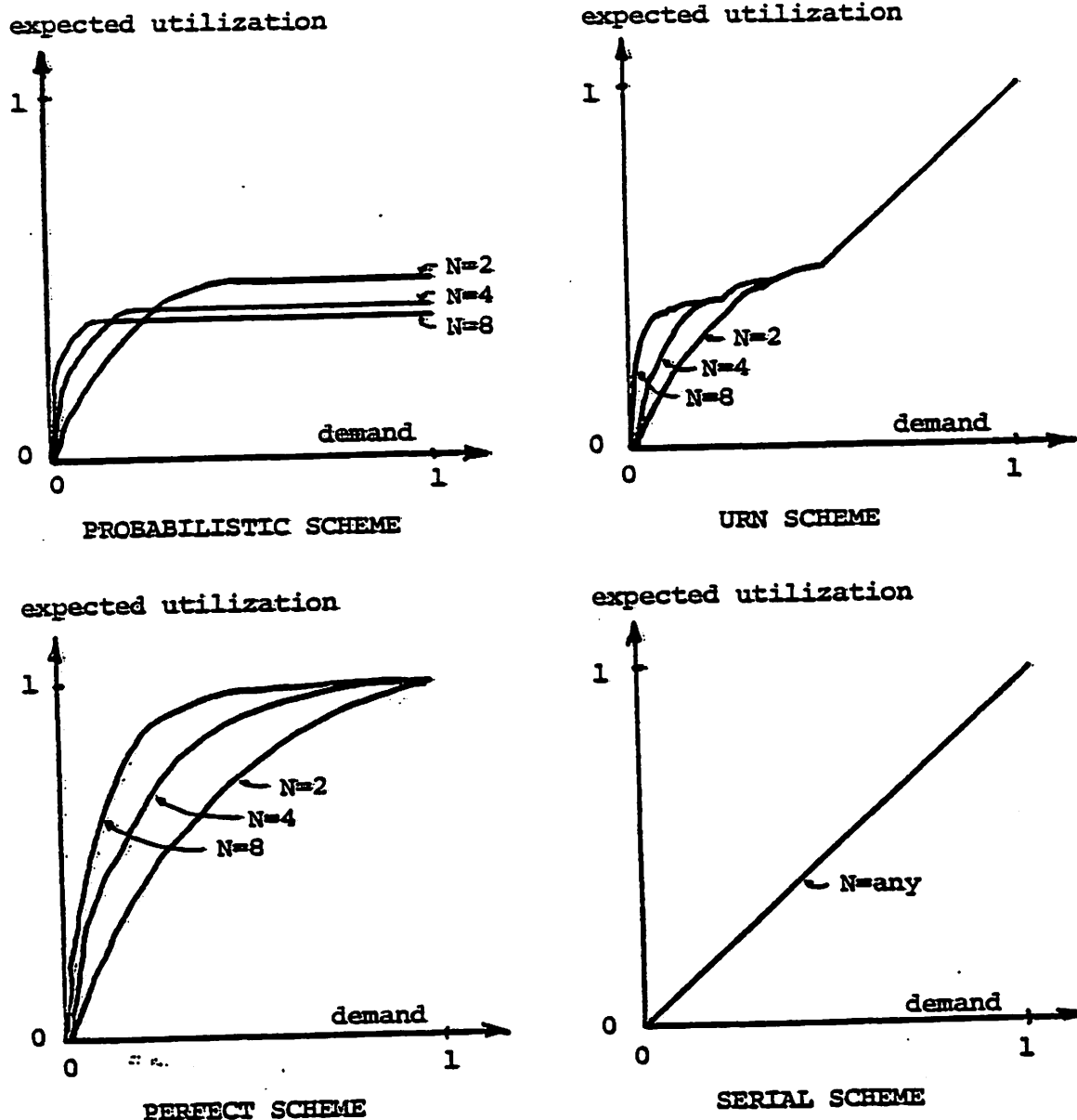
TESTS OF ADAPTIVE BALANCING ALGORITHM

In this appendix, the adaptive balancing algorithm developed in Chapter 4 and used in some of the experiments discussed in Chapters 5 and 8 is shown to perform correctly. First, the theoretical performance of the probabilistic and urn access control schemes are examined.¹⁹ Then, the adaptive balancing algorithm is tested through simulation in a controlled environment.

I.1. Theoretical Performance

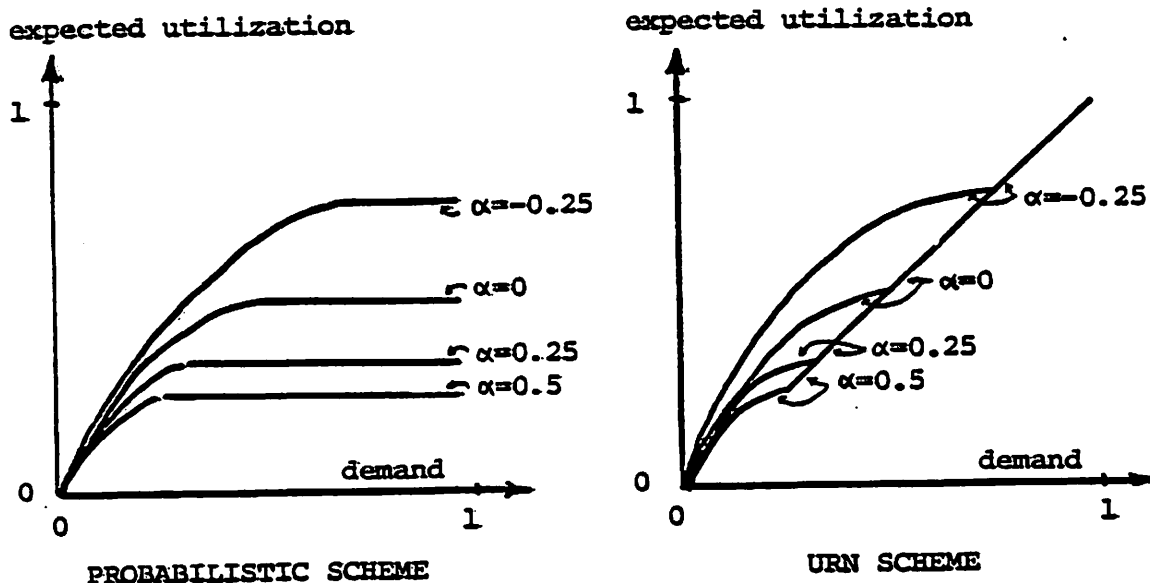
Figures 29 and 30 show the theoretical performance of the probabilistic and urn access control schemes, and compare their performance with a serial scheme (which gives access rights to one PE each iteration in a fixed order) and a perfect scheme (which gives access rights to one PE with demand each iteration if there is such a PE). These figures depict the case where a single resource is shared by a number of PEs, and each PE has demand with some fixed probability each time-slot, independent of other PEs' demands and other time-slots. Expected global utilization vs. demand for two, four, and eight identical PEs at a Pareto optimum assuming no penalty for collisions is

¹⁹ These schemes have been called update control schemes in the main part of this thesis because they are used to control simultaneous updates for distributed iterative refinement algorithms.



The expected utilization versus demand is given for N processing elements that share a single resource and employ a probabilistic, urn, perfect, and serial access control scheme. A processing element gains one unit of utilization if it successfully gains exclusive access to the shared resource, else it gains nothing. The demand is the probability each of the processing elements wants to access the shared resource. At low demands, the perfect scheme is best, followed by the probabilistic and urn schemes, and then the serial scheme. For high demands, the perfect scheme is again best, followed by the urn and serial schemes, and then the probabilistic scheme. As N increases, utilizations decrease.

Figure 29: Expected Utilization vs. Demand with N PEs and no Penalty.



The expected utilization versus demand is given for two processing elements that share a single resource and employ a probabilistic and urn access control scheme. A processing element suffers penalty (incurs - units utilization) if it collides with the other processing element in accessing the shared resource, gains one unit of utilization if it successfully gains exclusive access to the shared resource, or gains nothing if it does not attempt to access the shared resource. As the penalty increases, the effectiveness of both schemes is diminished.

Figure 30: Expected Utilization vs. Demand with 2 PEs and Penalty α .

shown in Figure 29. Figure 30 shows expected global utilization vs. demand for two identical PEs at a Pareto optimum assuming a range of penalties for collisions.

The effectiveness of the probabilistic scheme is clearly limited with moderate and high demand. The urn scheme performs better than the probabilistic scheme, but requires a common seed for the identical pseudo-random number generators. Note that the urn scheme performs like the probabilistic scheme for low demand, and like the serial scheme for high demand.

A perfect scheme (implemented with a central processor or global negotiation) is, of course, more effective; but, the communication costs of the perfect scheme can be prohibitive. The benefits of a perfect scheme are greatest for large N and low demand, but communication costs are high for large N since communication costs are geometrically related to N .

Figure 30 shows that a penalty for collisions forces the probabilistic and urn schemes to adopt more conservative policies; thus, they become less effective as the penalty increases. A penalty of $\alpha=1/2$, for example, cuts utilization in half. A negative penalty (actually a reward) causes the policies to be more optimistic.

I.2. Simulation Results

Simulation results of the adaptive balancing algorithm with the probabilistic and urn schemes to control access to a single shared resource are presented below. A controlled situation involving 4 PEs that share a single resource was chosen for these experiments. Demand is generated randomly and independently for each PE and each iteration. The probability that each PE has demand can be specified, and initial policies and estimates can be provided so that transient behavior can be observed.

A simple estimation scheme can be used to maintain estimates of conditional expected utilizations, from which conditional expected neighborhood utilization payoffs are calculated. A "window-size" is specified, and when PE_i has demand and attempts to access shared resources (say after time-slot t) it updates $E[\tilde{u}_j | p^{\text{attempt}}(i)]$ using the rule,

$$E[\tilde{u}_j | p^{\text{attempt}}(i)] := \\ ((\text{window_size}-1)E[\tilde{u}_j | p^{\text{attempt}}(i)] + \tilde{u}_j) / \text{window_size}$$

for each neighbor. Similar rules are used to update $E[\tilde{u}_j | p^{\text{attempt}}(i)]$ and, when PE_i has demand and sacrifices, $E[\tilde{u}_j | p^{\text{sacrifice}}(i)]$ for each neighbor and $E[\tilde{u}_j | p^{\text{sacrifice}}(i)]$. By varying window-size, the amount of history retained by PEs can be controlled. Although the simple estimation scheme works it can be many iterations between attempts/sacrifices if demand is high/low and estimates may get old.

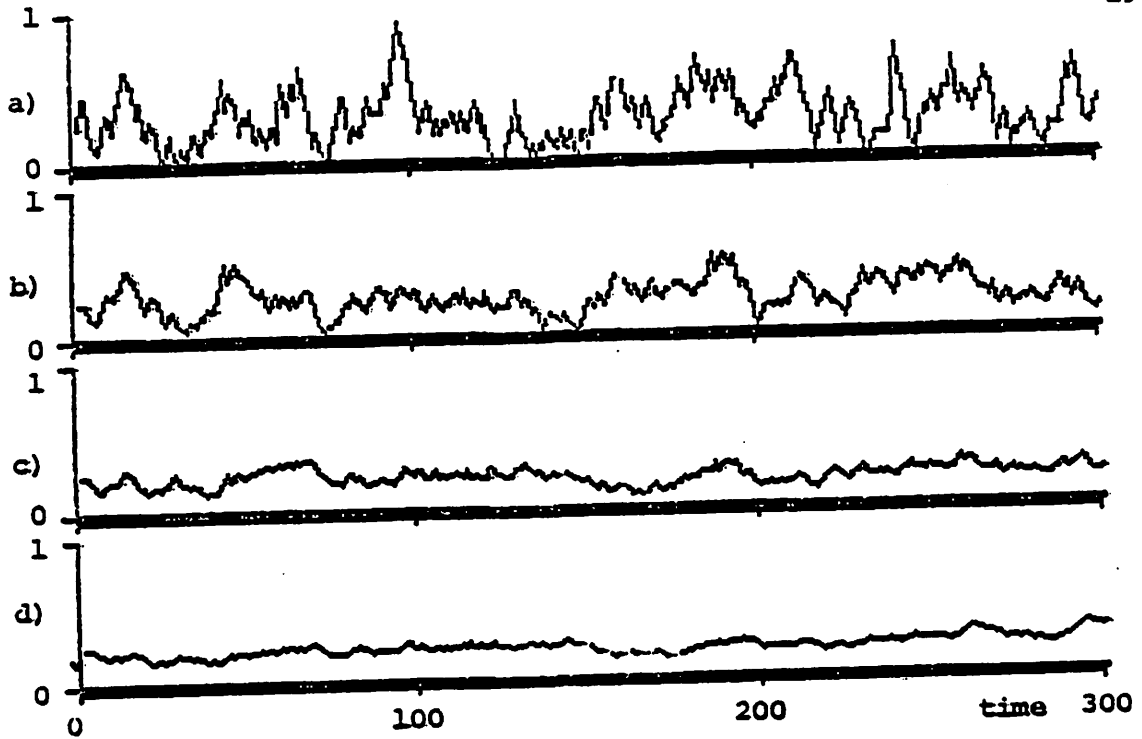
To obtain better results (shown below) a more complex estimation scheme can be used with the adaptive balancing algorithm. This estimation scheme takes advantage of PEs' ability to infer both conditional neighborhood expected utilization payoffs each iteration they have demand from knowledge of which other PEs attempt to access shared resources and knowledge of their own current and proposed new controls. This results in a more responsive algorithm.

All simulations involved the case of 4 PEs that share a single resource with payoff coefficient vector, $\underline{c}=(1,1,1,1)$. The next 7 figures show the behavior of the adaptive balancing algorithm. For each figure, PEs' policies are shown in the top half of the figure, and utilizations are shown in the bottom half. Because utilizations are discrete events, the average utilization is shown to the right of the graphs.

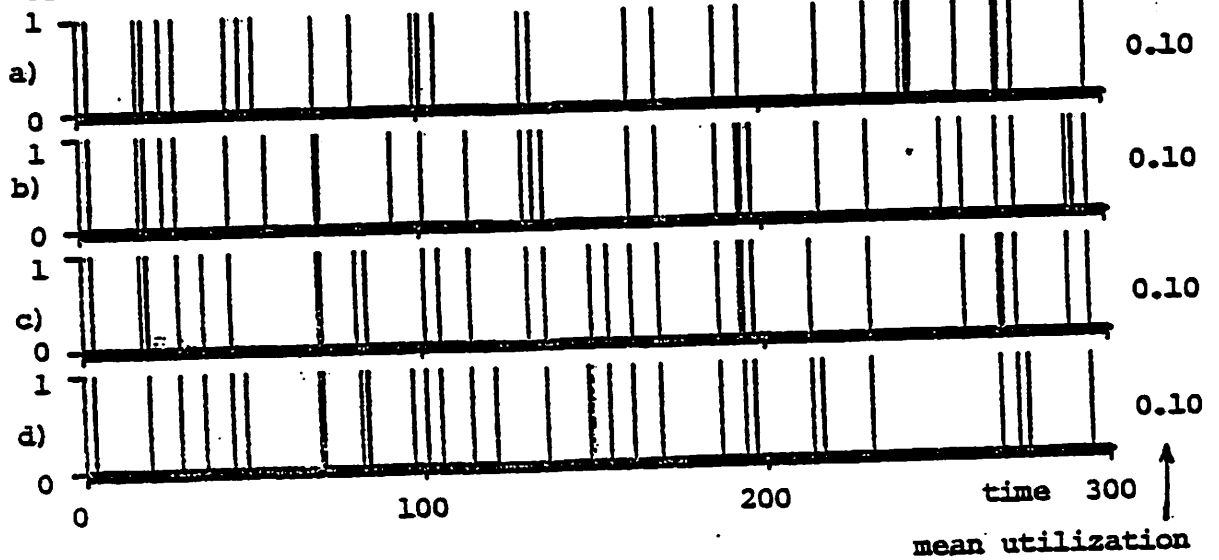
Figures 31 and 32 depict the probabilistic scheme starting at a Pareto-optimum. Figure 31 shows the policies and utilizations of a typical PE with various step-sizes and Figure 32 shows the policies and utilizations of a typical PE with various window-sizes. In both cases, all PEs have demand each iteration with probability 1.0 (i.e., $\underline{d}=(1.0,1.0,1.0,1.0)$ for all iterations) and there is no penalty for collisions, so the Pareto-optimal policy is $\underline{p}=(0.25,0.25,0.25,0.25)$; each PE's expected utilization payoff should be approximately 0.11. Note that the algorithm functions well even when the step-size is quite high.

The policy and utilization over time of one of four processing elements that share a single resource and employ a probabilistic access control scheme are shown with step-sizes of 0.1, 0.05, 0.02, and 0.01. A step-size is the magnitude of an incremental change in policy made by the probabilistic access control scheme. The processing elements have demand for the resource at all times. Policies for the probabilistic scheme are probability values; utilizations are discrete events. Good mean utilizations, close to the predicted 0.11, are achieved with any of the step-sizes. Policies stayed near the predicted 0.25 when a small step-size is used.

POLICIES:



UTILIZATIONS:

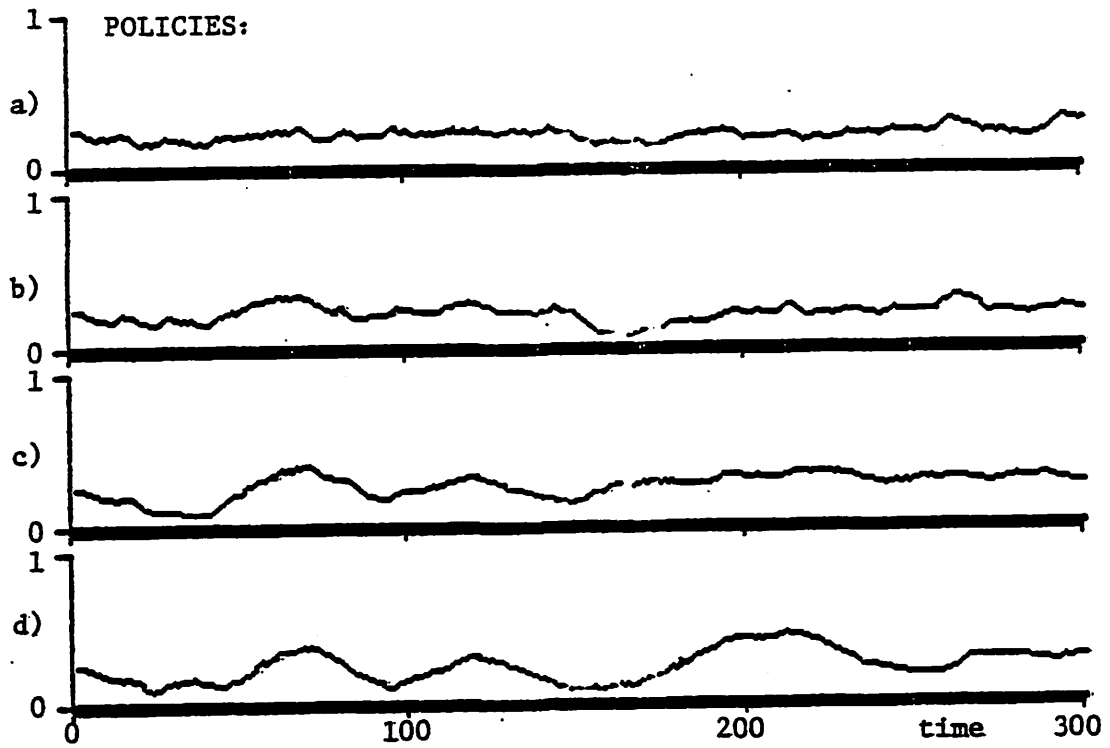


window-size = 2, penalty = 0,

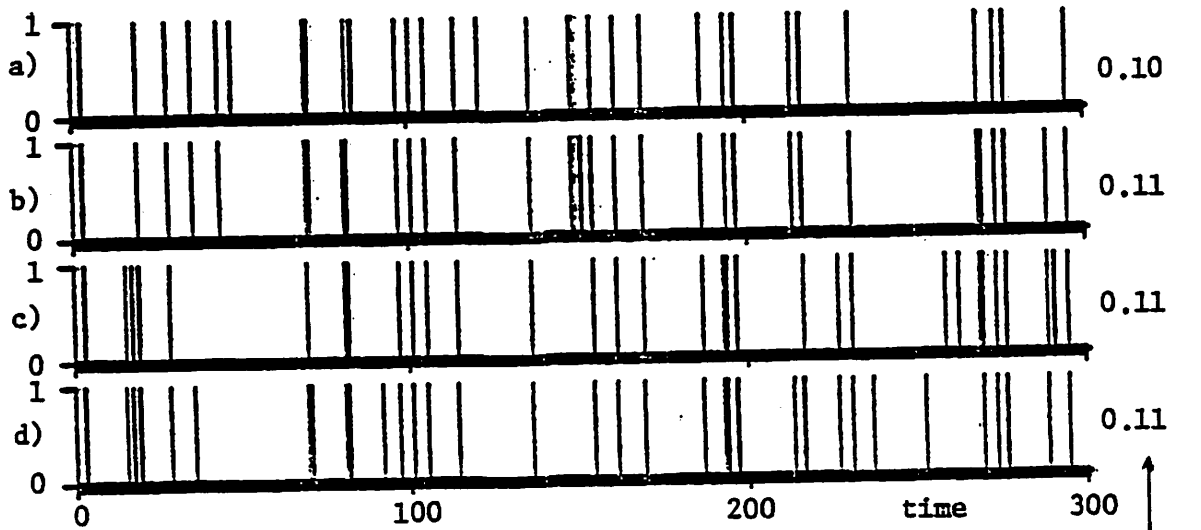
$$\text{step-size} = \begin{cases} \text{a) } 0.10 \\ \text{b) } 0.05 \\ \text{c) } 0.02 \\ \text{d) } 0.01 \end{cases}$$

Figure 31: Probabilistic Scheme at Equilibrium, Various Step-sizes.

The policy and utilization over time of one of four processing elements that share a single resource and employ a probabilistic access control scheme are shown with window-sizes of 2, 4, 8, and 16. A window-size governs the probabilistic access control scheme's memory of past utilizations. The processing elements have demand for the resource at all times. Policies for the probabilistic scheme are probability values; utilizations are discrete events. Good mean utilizations, close to the predicted 0.11, are achieved with any of the window-sizes, but utilizations are more evenly distributed over time and policies more constant with a small window-size.



UTILIZATIONS:



step-size=0.01, penalty=0

mean utilization

- window-sizes:
- a) 2
 - b) 4
 - c) 8
 - d) 16

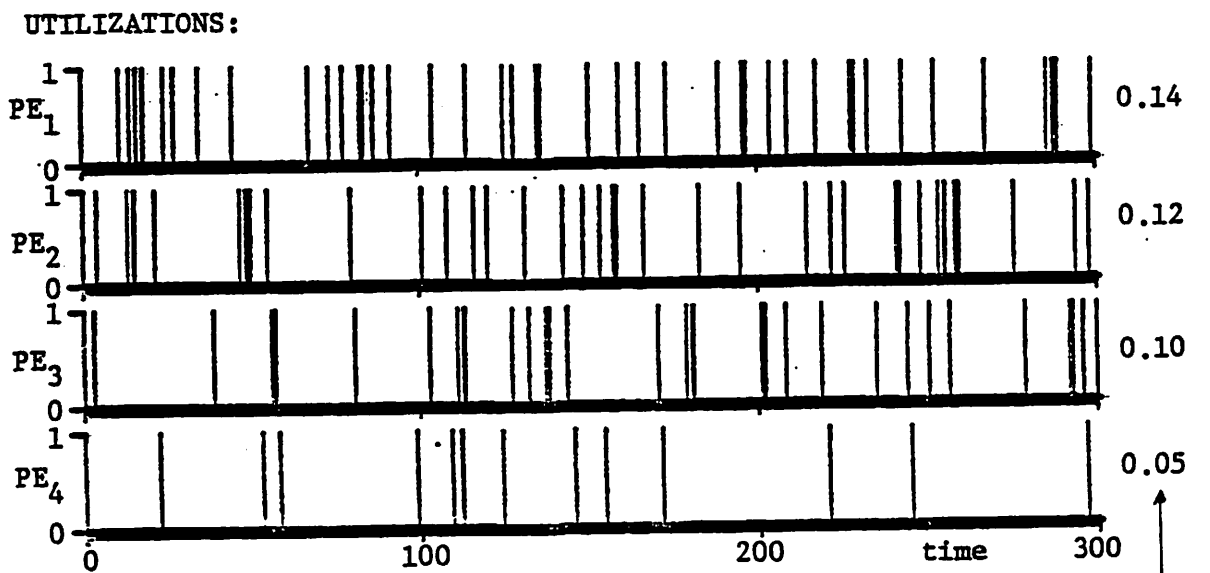
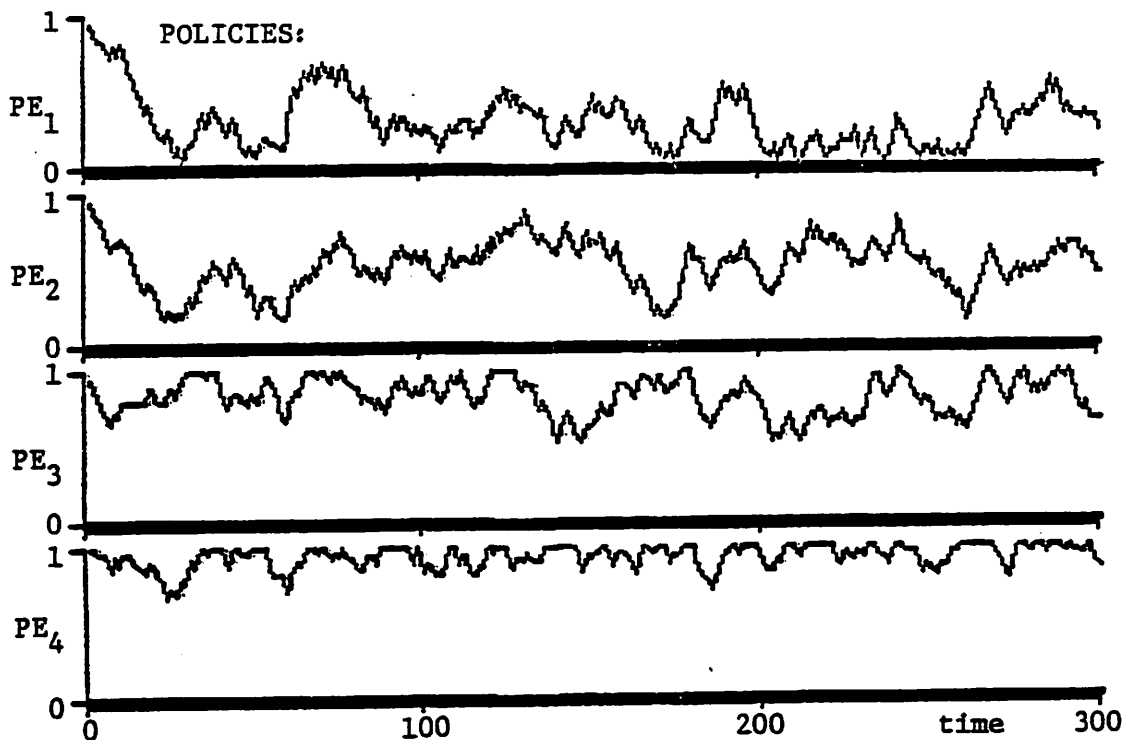
Figure 32: Probabilistic Scheme at Equilibrium, Various Window-sizes.

Figures 33 and 34 depict the probabilistic scheme coping with a dramatic change in load. The PEs begin with policies and estimates appropriate for $\underline{d}=(0.25,0.25,0.25,0.25)$, but are faced with $\underline{d}=(1.0,0.5,0.25,0.1)$ for all iterations. A window-size of 2 and step-size of 0.1 are assumed for Figure 33; a window-size of 2 and step-size of 0.01 are assumed for Figure 34. It took approximately 20 iterations to adapt with a step-size of 0.1, and approximately 120 iterations to adapt with a step-size of 0.01.

The Pareto-optimal policy sought by the network of PEs for runs depicted in Figures 33 and 34 is an extreme policy, $\underline{p}=(0.25,0.5,1.0,1.0)$. Notice that for $i=1, 2,$ and 3 $d_i p_i=0.25$, but $d_4 p_4=0.1$. In this situation the adaptive balancing algorithm is striving to find a \underline{p} such that $d_i p_i=0.25$ for all PEs. However, balance cannot be achieved with $\underline{c}=(1,1,1,1)$. Thus, after an initial transient period $PE_1, PE_2,$ and PE_3 repeatedly send messages to PE_4 requesting that it increase its attempts to access shared resources; PE_4 , though, cannot increase p_4 beyond 1.0. This behavior of the adaptive balancing algorithm at such extreme Pareto optima seems reasonable.

The probabilistic scheme at equilibrium, but with various penalties for collisions is depicted in Figure 35. The policies and utilizations are as expected. Note that the particular version of adaptive balancing algorithm that was simulated has each PE_i decrease p_i if it is involved in a collision; this may be inappropriate for negative penalties (i.e., when collisions are beneficial).

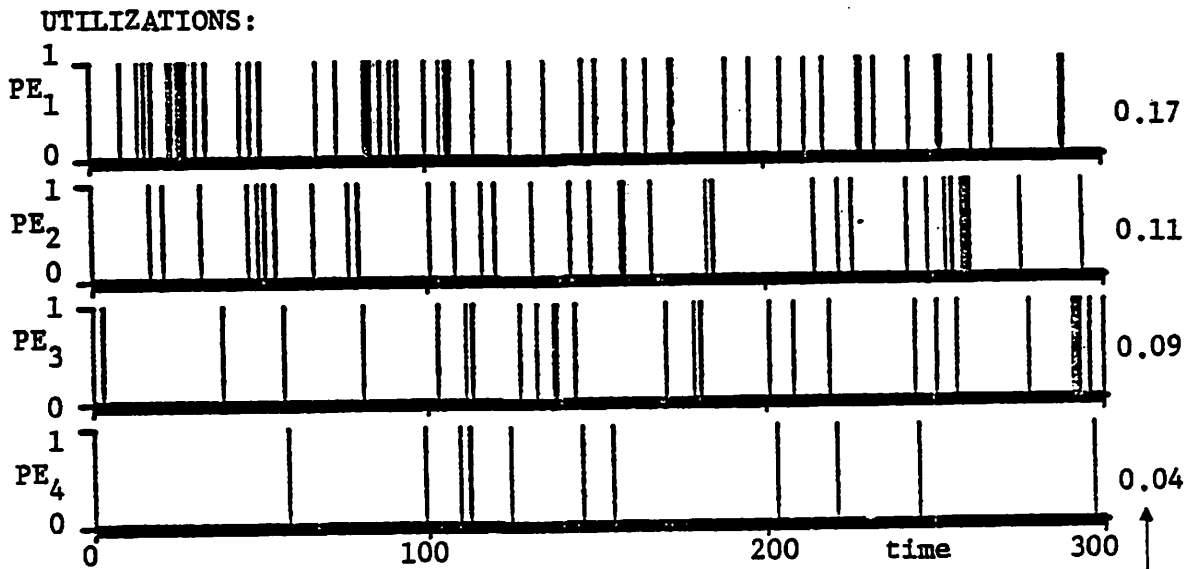
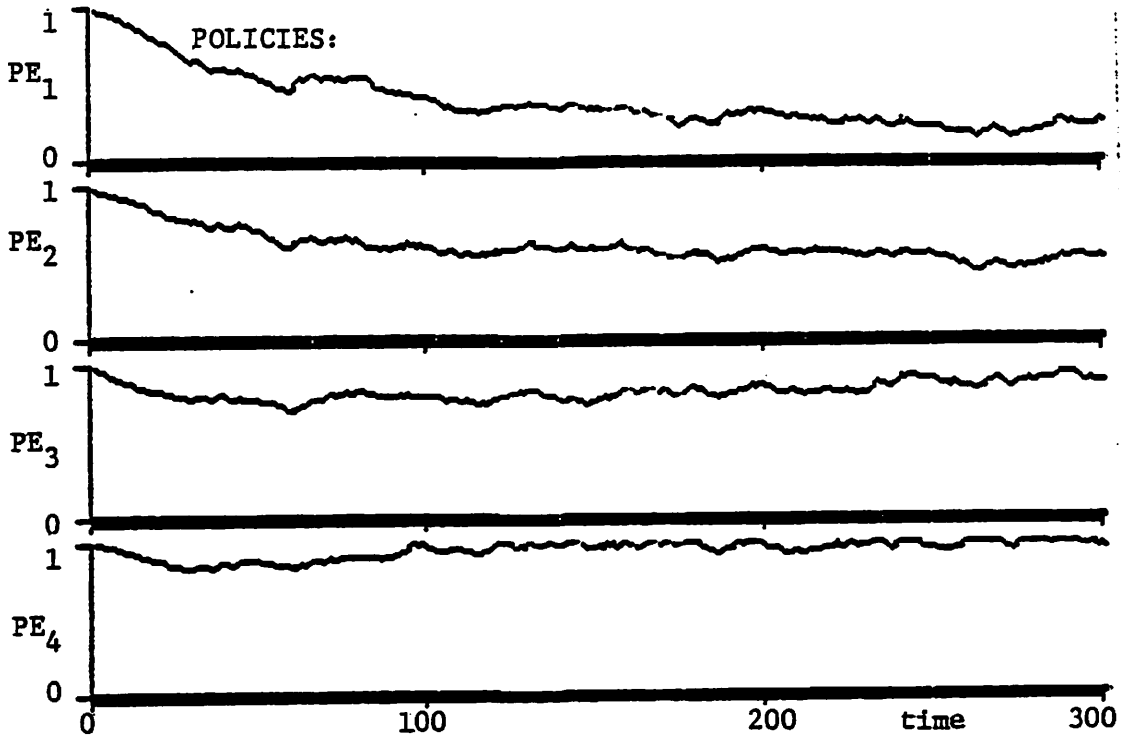
The transient behavior of the probabilistic access control scheme with large step-size is illustrated for four processing elements that share a single resource. The policies for all processing elements are probability values that are started at 1.0; utilizations are discrete events. Not all processing elements have demand for the shared resource at all times, but all processing elements must adapt their policies. The first three are reasonably successful at achieving the predicted 0.11 mean utilization, while the fourth's demand is too low for it to achieve a similar mean.



	\underline{PE}_1	\underline{PE}_2	\underline{PE}_3	\underline{PE}_4	mean utilization
demand	1.0	0.5	0.25	0.1	
initial policy	1.0	1.0	1.0	1.0	
initial estimate	0.4	0.4	0.4	0.4	

Figure 33: Transient Behavior of Probabilistic Scheme, Large Step-size.

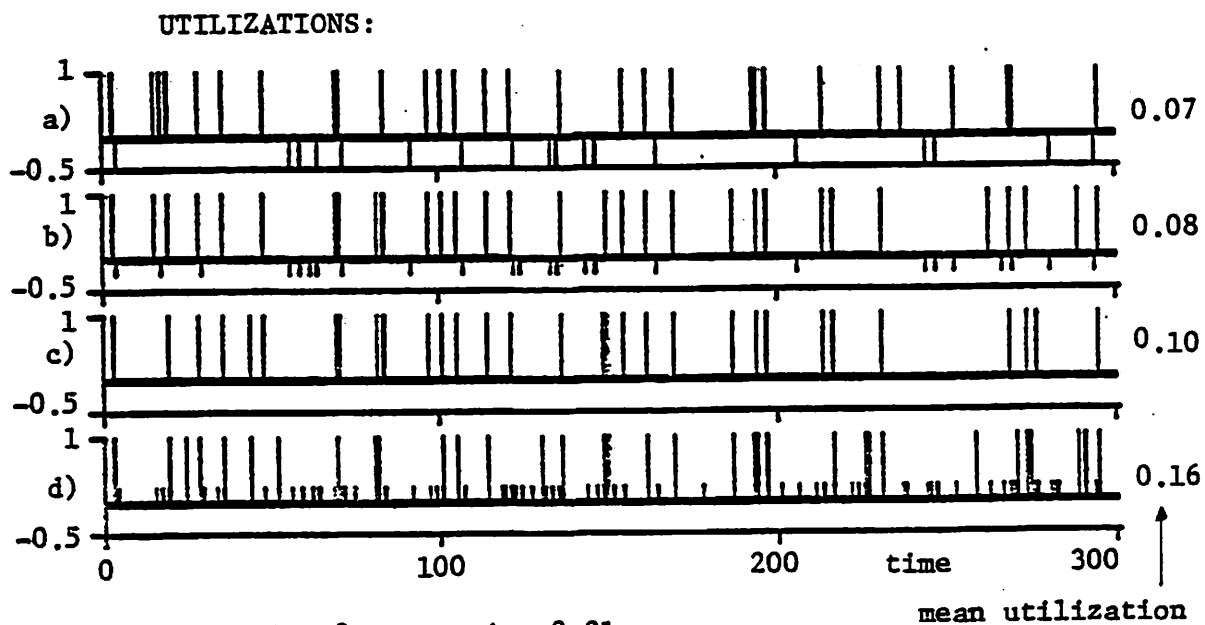
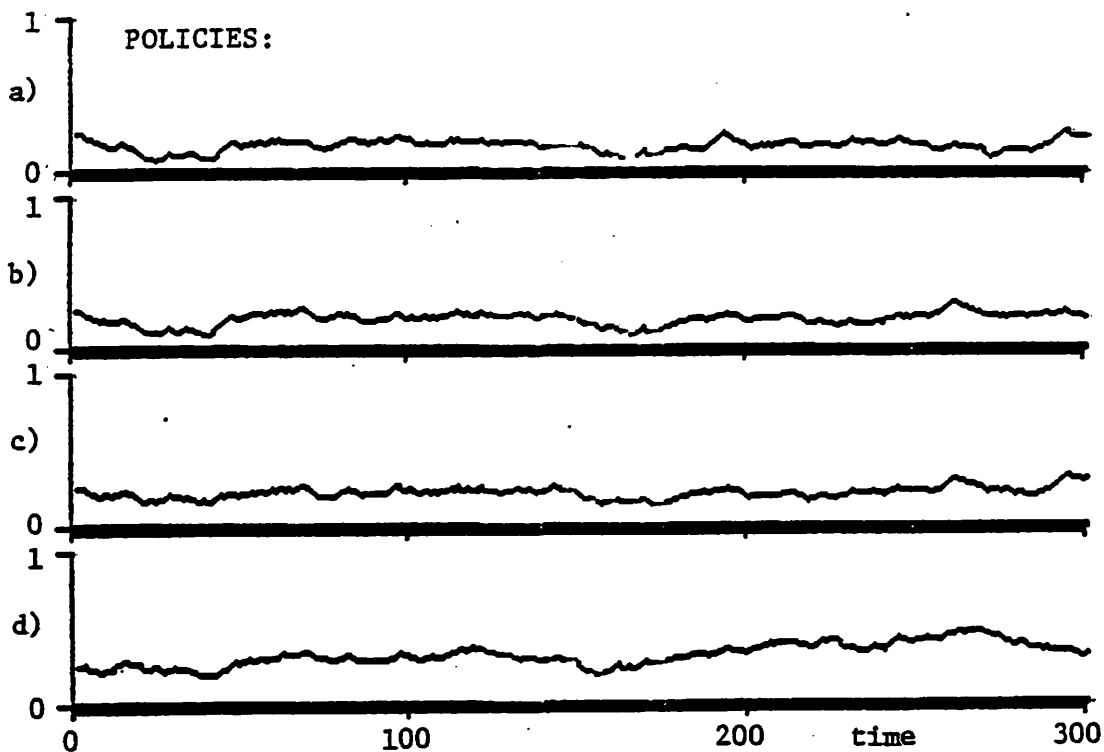
The transient behavior of the probabilistic access control scheme with small step-size is illustrated for four processing elements that share a single resource. The policies for all processing elements are probability values that are started at 1.0; utilizations are discrete events. Not all processing elements have demand for the shared resource at all times, but all processing elements must adapt their policies. The first three are quite slow at achieving the predicted 0.11 mean utilization due to the small step-size, while the fourth's demand is too low for it to achieve a similar mean.



	\underline{PE}_1	\underline{PE}_2	\underline{PE}_3	\underline{PE}_4	mean utilization
demand	1.0	0.5	0.25	0.1	
initial policy	1.0	1.0	1.0	1.0	
initial estimate	0.4	0.4	0.4	0.4	

Figure 34: Transient Behavior of Probabilistic Scheme, Small Step-size.

The policy and utilization over time of one of four processing elements that share a single resource and employ a probabilistic access control scheme are shown with penalties for collisions of 0.5, 0.25, 0, and -0.25. Policies are probability values; utilizations are discrete events. A high penalty makes a policy more conservative, and a negative policy makes a policy more liberal. This is reflected in the decrease in mean utilization observed when a high penalty is used, and an increase when a negative penalty is used.



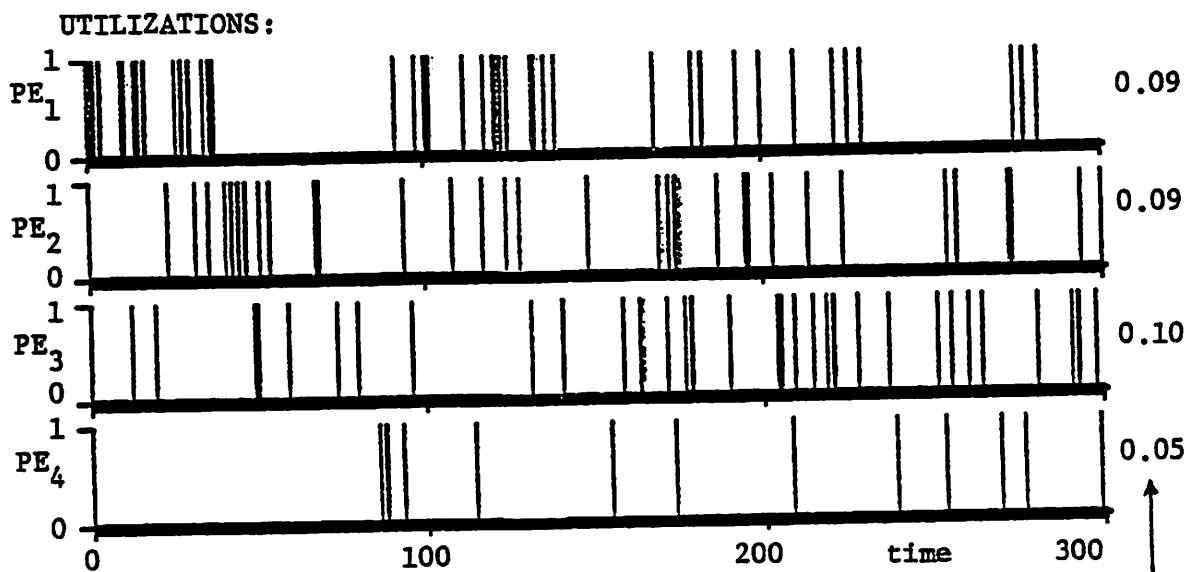
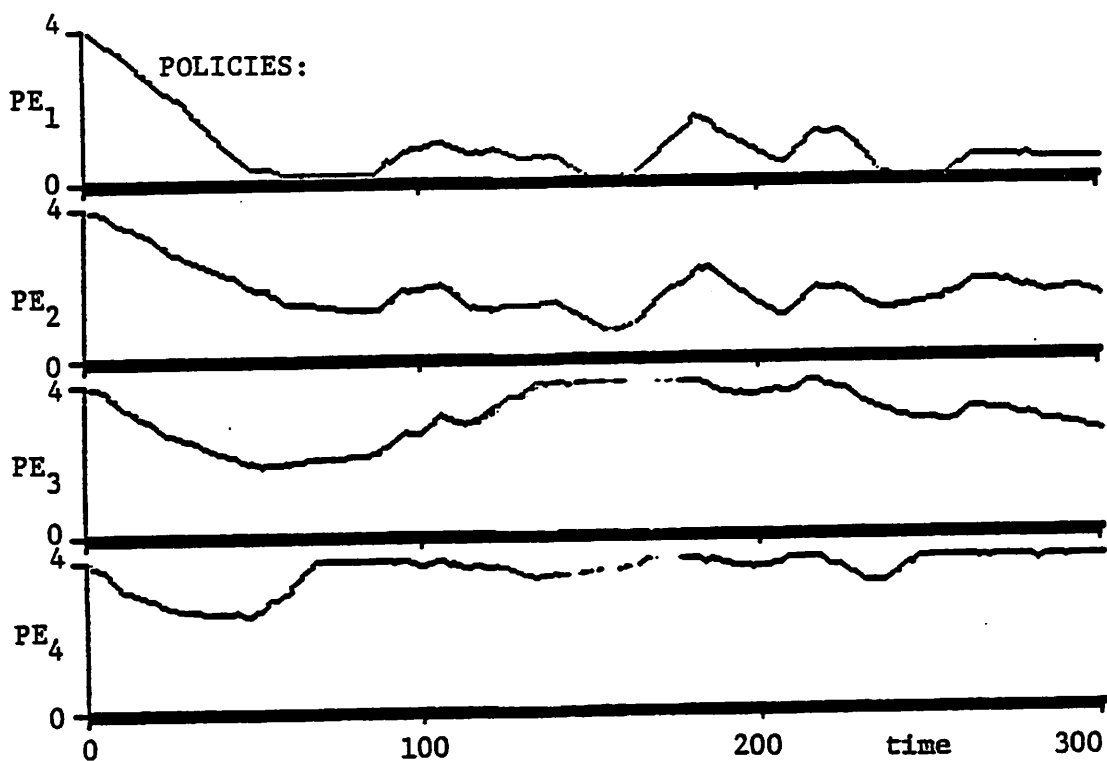
window-size=2, step-size=0.01

penalty:
a) 0.5
b) 0.25
c) 0.0
d) -0.25

Figure 35: Probabilistic Scheme near Equilibrium, Various Penalties.

Finally, Figures 36 and 37 depict the urn scheme coping with the same dramatic change in load as the probabilistic scheme faced in Figure 33 and Figure 34. With the urn scheme, policies range from 0.0 to N . Also, policies are rounded off to integers when used for the urn scheme because an integral number of numbers are to be drawn from pseudo-random number generators. Note that PE_1 would have a constant policy of 1.0 and utilization of 0.25 if the other PEs also had high demand, but other PEs' low demand forces PE_1 to sacrifice some of these slots.

The transient behavior of the urn access control scheme with large step-size is illustrated for four processing elements that share a single resource. The policies for all processing elements range between 0 and 4, are started at 4.0, and are rounded off to integers before use; utilizations are discrete events. Not all processing elements have demand for the shared resource at all times, but all processing elements must adapt their policies. The first three are reasonably successful at achieving predicted mean utilizations, while the fourth's demand is too low for it to achieve a similar mean.

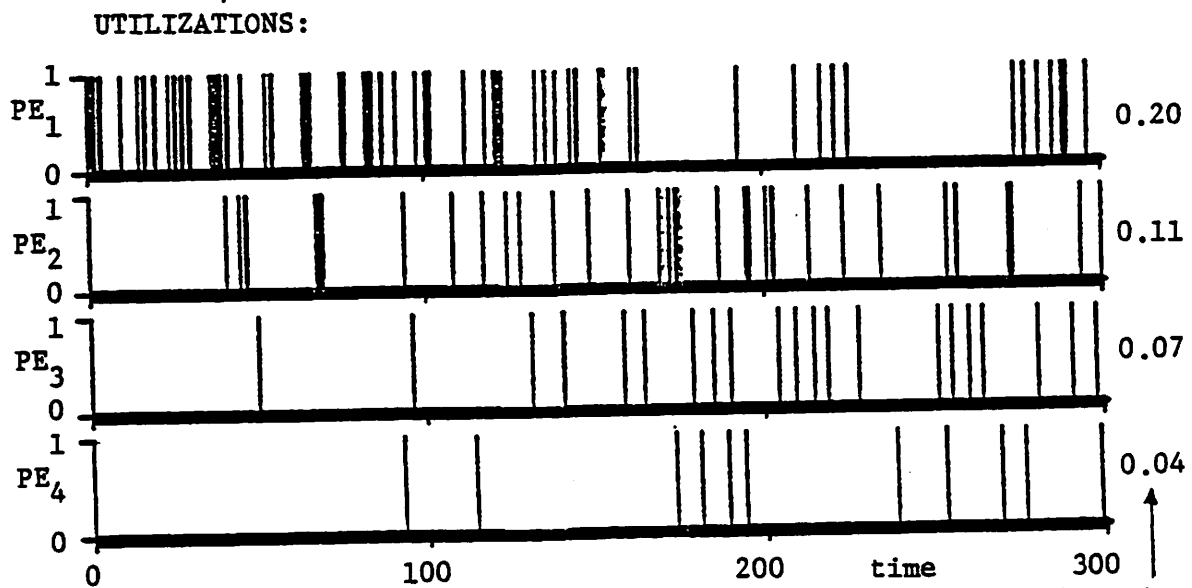
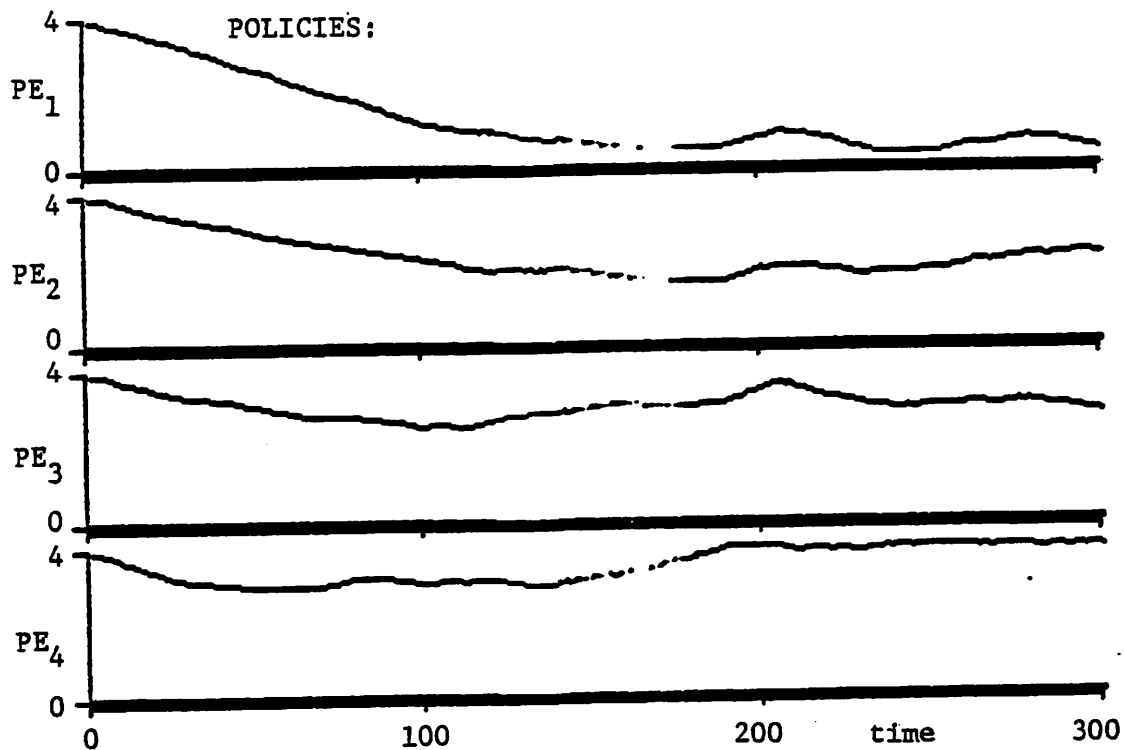


	\underline{PE}_1	\underline{PE}_2	\underline{PE}_3	\underline{PE}_4
demand	1.0	0.5	0.25	0.1
initial policy	4.0	4.0	4.0	4.0
initial estimate	0.4	0.4	0.4	0.4

mean utilization

Figure 36: Transient Behavior of Urn Scheme, Large Step-size.

The transient behavior of the urn access control scheme with large step-size is illustrated for four processing elements that share a single resource. The policies for all processing elements range between 0 and 4, are started at 4.0, and are rounded off to integers before use; utilizations are discrete events. Not all processing elements have demand for the shared resource at all times, but all processing elements must adapt their policies. All processing elements are quite slow to adapt their policies.



	\underline{PE}_1	\underline{PE}_2	\underline{PE}_3	\underline{PE}_4
demand	1.0	0.5	0.25	0.1
initial policy	4.0	4.0	4.0	4.0
initial estimate	0.4	0.4	0.4	0.4

Figure 37: Transient Behavior of Urn Scheme, Small Step-size.

A P P E N D I X B

TWO-PLATOON TRAFFIC FLOW SYSTEM MODEL

This appendix contains disutility and platoon structure calculations for a two-platoon traffic flow system model. The flow system is made up of signalized intersections and directed road links, connecting the intersections. Each link in such a traffic flow system carries traffic from one intersection to another; two-way traffic between intersections is represented by two links. Arbitrary traffic networks can be accommodated in the model.

Disutility is a measure of stops and delay incurred by platoons interacting with a signal after traversing a link. The disutility on an individual link depends on the time when the primary and secondary platoons arrive at the intersection at the end of the link, the timing of the signal at the intersection at the end of the link, and the platoons' bandwidths. The speed at which traffic flows and the length of a link determines the free-flow delay of a link (an approximation to the time it takes a platoon to traverse the link).

The rules that define the platoon structure of the network (the platoon departure times) are embodied in a set of functions that relate the times platoons leave intersections to the times platoons arrive at intersections. A primary platoon's departure time from an intersection on a link is the start of the green phase of the intersection's signal if no other link feeds traffic onto the link, else it is a function of the arrival time of platoons on the feeding link and the timing of the

signal. Secondary platoons represent turning traffic, and always depart signals at the start of their red phase.

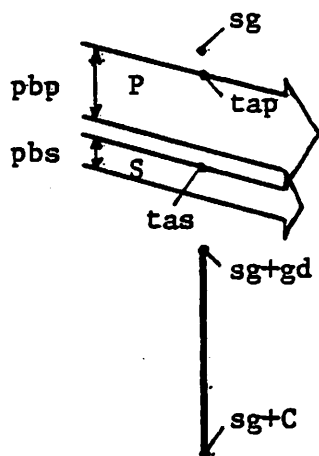
With a two-platoon traffic model there are eight ways in which the two platoons can interact with a signal. These eight situations are described and illustrated in 5 figures, starting with Figure 38. Disutility and platoon structure formulas are also included. The following definitions are used in these formulas.

sg is the green switching time of a signal,
 gd is the length of the green phase of a signal,
 C is the cycle length of all signals,
 pbp is the primary platoon bandwidth on a link,
 pbs is the secondary platoon bandwidth on a link,
 tap is a primary platoon's arrival time at a signal,
 tas is a secondary platoon's arrival time at a signal, and
 tdr is the departure time for the primary platoon that
 results from flow on a link,

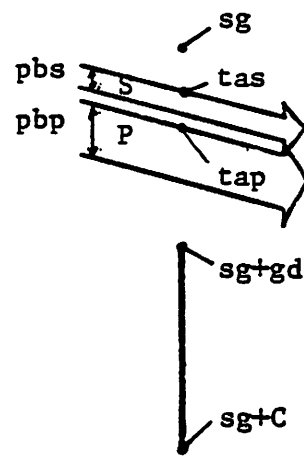
subject to the following constraints:

$0 \leq sg < C$
 $min_duration \leq gd \leq max_duration$
 $pbp + pbs \leq gd$
 $sg \leq tap < sg + C$
 $sg \leq tas < sg + C$
 $(tas \geq tap + pbp) \text{ or } (tas \leq tap - pbs)$
 $(tap \geq tas + pbs) \text{ or } (tap \leq tas - pbp)$

$$\text{conditions: } \begin{cases} sg \leq tap \leq sg + gd - pbp \\ sg \leq tas \leq sg + gd - pbs \end{cases}$$



(a)



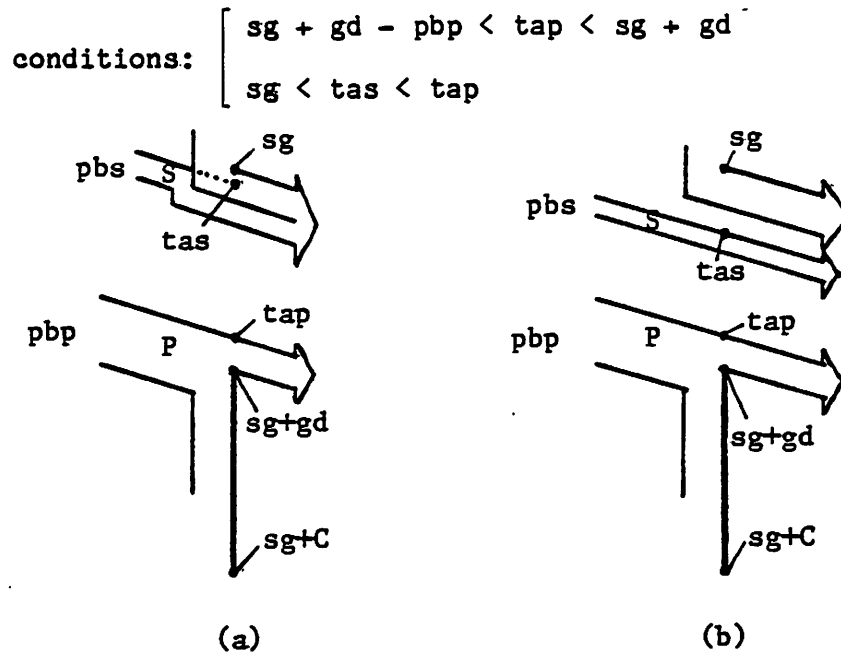
(b)

Disutility = 0

$$\text{tdr} = \begin{cases} \text{tap} & \text{if } pbp \geq pbs \\ \text{tas} & \text{if } pbp < pbs \end{cases}$$

Case 1 of the interaction with a signal that is encountered with the two-platoon network traffic light control experiments is illustrated. Both the primary and secondary platoons arrive during the green phase of the signal, so no disutility is incurred at all. The resulting primary platoon (not shown) is assumed to depart when the larger of the two platoons clear the signal.

Figure 38: Two-platoon Interaction with a Signal - Case 1.



if $tas < (tap + pbp - gd)$

then $Disutility = \alpha(tap + pbp - sg - gd)(C - gd + \beta) + \alpha pbs(tap + pbp - gd - tas + \beta)$

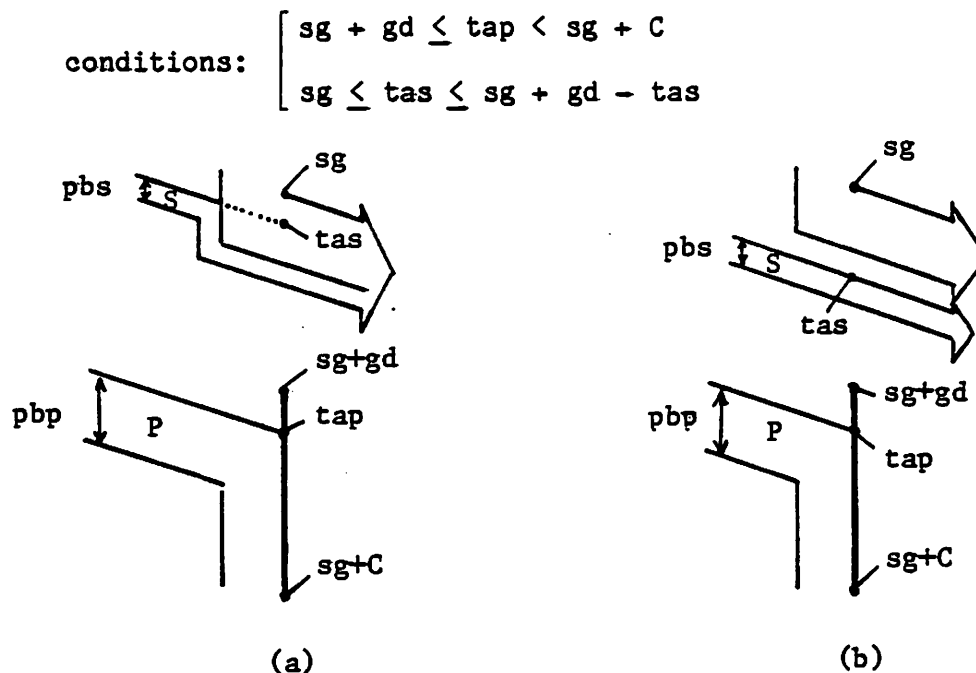
$$tdr = \begin{cases} sg & \text{if } tap + pbp - sg - gd + pbs \geq sg + gd - tap \\ tap & \text{if } tap + pbp - sg - gd + pbs < sg + gd - tap \end{cases}$$

else $Disutility = \alpha(tap + pbp - sg - gd)(C - gd + \beta)$

$$tdr = \begin{cases} sg & \text{if } (tap + pbp - sg - gd \geq sg + gd - tap) \wedge (tap + pbp - sg - gd \geq pbs) \\ tas & \text{if } (pbs > tap + pbp - sg - gd) \wedge (pbs \geq sg + gd - tap) \\ tap & \text{if } (sg + gd - tap > tap + pbp - sg - gd) \wedge (sg + gd - tap > pbs) \end{cases}$$

Case 2 of the interaction with a signal that is encountered with the two-platoon network traffic light control experiments is illustrated. Part of the primary platoon is stopped by the signal and the secondary platoon arrives during the signal's green phase. Disutility is incurred by the part of the primary platoon that is stopped, and by the secondary platoon if it encounters the primary platoon. The resulting primary platoon (not shown) is assumed to depart when the largest piece of platoon clears the signal. Case 3 is similar, and is obtained by exchanging the primary and secondary platoons.

Figure 39: Two-platoon Interaction with a Signal - Cases 2 and 3.



if $tas < sg + pbp$

then Disutility = $\alpha pbp(sg+C-tap+\beta) + \alpha pbs(sg+pbp-tas+\beta)$

$tdr = sg$

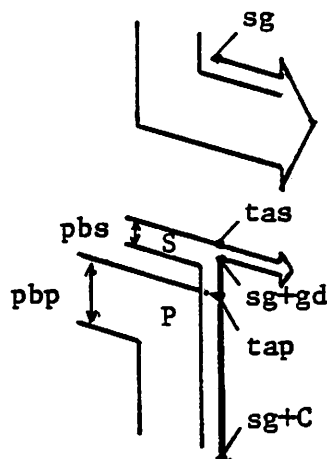
else Disutility = $\alpha pbp(sg+C-tap+\beta)$

$$tdr = \begin{cases} sg & \text{if } pbp \geq pbs \\ tas & \text{if } pbp < pbs \end{cases}$$

Case 4 of the interaction with a signal that is encountered with the two-platoon network traffic light control experiments is illustrated. The entire primary platoon is stopped by the signal and the secondary platoon arrives during the signal's green phase. Disutility is incurred by the primary platoon, and the secondary platoon if it encounters the primary platoon. The resulting primary platoon (not shown) is assumed to depart when the signal turns green if the primary platoon is larger than the secondary platoon, else it is assumed to depart when the secondary platoon clears the signal. Case 5 is similar, and is obtained by exchanging the primary and secondary platoons.

Figure 40: Two-platoon Interaction with a Signal - Cases 4 and 5.

$$\text{conditions: } \begin{cases} sg + gd \leq tap < sg + C \\ sg + gd - pbs < tas < sg + gd \end{cases}$$



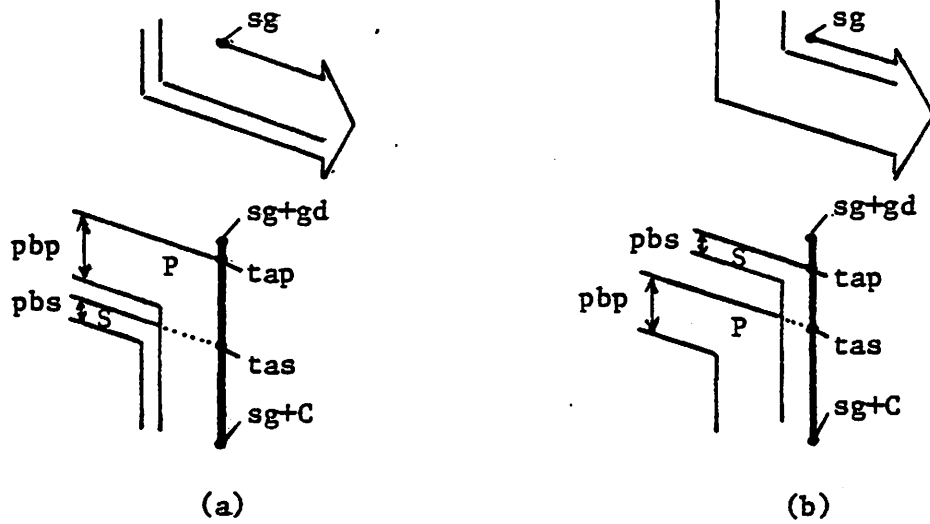
$$\text{Disutility} = \alpha(tas + pbs - sg - gd)(C - gd + \beta) + \alpha pbp(tas + pbs + C - gd - tap + \beta)$$

$$\text{tdr} = \begin{cases} sg & \text{if } pbp + tas + pbs - sg - gd \geq sg + gd - tas \\ tas & \text{if } pbp + tas + pbs - sg - gd < sg + gd - tas \end{cases}$$

Case 6 of the interaction with a signal that is encountered with the two-platoon network traffic light control experiments is illustrated. The entire primary platoon and part of the secondary platoon is stopped by the signal, so disutility is incurred by both platoons. The resulting primary platoon (not shown) is assumed to depart when the signal turns green, unless the piece of secondary platoon that clears the signal is larger than the other pieces. Case 7 is similar, and is obtained by exchanging the primary and secondary platoons.

Figure 41: Two-platoon Interaction with a Signal - Cases 6 and 7.

$$\text{conditions: } \begin{cases} sg + gd \leq tap < sg + C \\ sg + gd < tas < sg + C \end{cases}$$



if $tap < tas$

$$\text{then Disutility} = \alpha pbp(sg+C-tap+\beta) + \alpha pbs(sg+C+pbp-tas+\beta)$$

$$tdr = sg$$

$$\text{else Disutility} = \alpha pbp(sg+C+pbs-tap+\beta) + \alpha pbs(sg+C-tas+\beta)$$

$$tdr = sg$$

Case 8 of the interaction with a signal that is encountered with the two-platoon network traffic light control experiments is illustrated. Both platoons are stopped by the signal, and so incur disutility. The resulting primary platoon departs when the signal turns green.

Figure 42: Two-platoon Interaction with a Signal - Case 8.