NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

Canada
ON STOCHASTIC SORTING

by

JACK R. ZGIERSKI, B.Sc.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario
April 8, 1993.

© copyright 1993,
Jack Zgierski
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L’auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L’auteur conserve la propriété du droit d’auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-84095-1
Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.
The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

"ON STOCHASTIC SORTING"

submitted by Jack R. Zgierski, B.Sc.
in partial fulfillment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Director, School of Computer Science

Carleton University

May 6, 1993.
ABSTRACT

The problem of stochastic sorting is extensively examined in this thesis. This problem is also known as sorting with errors or sorting under a stochastic environment. Relevant topics from the literature are first reviewed and presented. These include the problems of noisy binary searching and of designing fault tolerant sorting circuits. We introduce the concept of the deterministic filter which attenuates any errors which occur during the comparison of individual pairs of values. These deterministic filters can then be used by standard sorting algorithms to achieve stochastic sorting. Hierarchical structures of such filters are also examined.

The second part of the thesis deals with stochastic sorting algorithms based on list organizing methods. Such algorithms are ergodic in nature and are thus able to function in a non-stationary environment. The concept of ergodic deterministic filters is then introduced. These ergodic filters operating in tandem with any list organizing scheme are effectively able to filter out errors. All stochastic algorithms are fully presented, and implemented. Extensive simulation results of the stochastic sorting ability of the various algorithms are also included.
ACKNOWLEDGMENTS

I would like to express my sincere thanks towards Dr. B. John Oommen, my supervisor, for his helpful critique, wisdom, and guidance which allowed me to see the way towards the completion of this thesis. His unyielding enthusiasm and determination in research has been a constant source of inspiration to me. I am also extremely grateful to him for the patience, support, and advice that he has extended towards me during my graduate years, as well as his attention to detail which ensured the professional quality of this work.

I would also like to thank the School of Computer Science at Carleton University for allowing me this opportunity, and in particular, Marlene Wilson who worked tirelessly to get the administrative tasks accomplished so that I could graduate early. I am also grateful to Dr. E. Kranakis and Dr. I. Stojmenovic who were on my defence committee, for their helpful suggestions.
# TABLE OF CONTENTS

## CHAPTER 1:
**INTRODUCTION TO STOCHASTIC SORTING** ........................................ 1
  1.1 General Introduction ...................................................................... 1
  1.2 The Roots Of Stochastic Sorting And Its Applications .................. 3
  1.3 Formal Problem Definition And Model ........................................ 5
  1.4 The Failure Probability .................................................................. 8
  1.5 Outline Of Thesis .......................................................................... 8

## CHAPTER 2:
**A REVIEW OF PREVIOUS LITERATURE** .......................................... 12
  2.1 General Review .............................................................................. 12
  2.2 Searching with Errors .................................................................... 13
    2.2.1 Results Due To Rivest et al. .................................................. 14
    2.2.2 Results Due To Pelc ............................................................... 19
    2.2.3 Results Due To Feige et al. ..................................................... 22
  2.3 Fault Tolerant Sorting Circuits ..................................................... 25
    2.3.1 Results Due To Yao and Yao .................................................... 26
    2.3.2 Results Due To Assaf and Upfal ............................................. 28
    2.3.3 Results Due To Leighton et al. ................................................. 30
  2.4 Tournaments Algorithms .................................................................. 32
    2.4.1 The Monrad Tournament ......................................................... 32
    2.4.2 The Butterfly Tournament ....................................................... 34
    2.4.3 The Points Tournament ........................................................... 35
  2.5 Improvement To Pelc’s Algorithm ................................................... 36
  2.6 A Comparison Of Presented Algorithms ....................................... 38
  2.7 Conclusions .................................................................................. 41

## CHAPTER 3:
**STOCHASTIC SORTING USING DETERMINISTIC FILTERS** .......... 43
  3.1 Introduction .................................................................................. 43
  3.2 Majority Filter .............................................................................. 45
3.3 Optimal Majority Filter ........................................... 53
3.4 Consecutive Filter ............................................... 61
3.5 Leader Filter .................................................... 66
3.6 Stochastic Leader Filter ......................................... 69
3.7 Stochasticizing The Other Filters ................................. 73
3.8 A Comparison Of Deterministic Filters ..................... 74
3.9 A Comparison Of Stochastic Sorting Algorithms Using Filters ... 80
3.10 Conclusions ................................................... 84

CHAPTER 4:
USING MULTIPLE LEVELS OF FILTERS .............................. 87
  4.1 Introduction .................................................. 87
  4.2 Analysis Of The Hierarchical Filtering Structure .......... 89
  4.3 A Hierarchy Of Majority Filters ............................. 90
  4.4 A Hierarchy Of Optimal Majority Filters ................ 92
  4.5 A Hierarchy Of Consecutive Filters ....................... 93
  4.6 A Hierarchy Of Leader Filters .............................. 97
  4.7 Conclusions ............................................... 98

CHAPTER 5:
STOCHASTIC SORTING USING LIST ORGANIZING METHODS ...... 99
  5.1 Introduction .................................................. 99
  5.2 Choosing Values For Comparison ............................ 100
    5.2.1 The Random Scheme .................................. 101
    5.2.2 The Neighbour Scheme ................................. 102
  5.3 Move To Front .............................................. 103
  5.4 Move To Back ............................................... 108
  5.5 Move To Front and Back .................................... 110
  5.6 Transposition Rule ......................................... 115
  5.7 Transpose Forwards And Backward .......................... 117
  5.8 Exchange Rule .............................................. 119
  5.9 Simulation Methods And Results ............................ 120
  5.10 Conclusions .............................................. 136
CHAPTER 6:
STOCHASTIC SORTING USING ERGODIC DETERMINISTIC FILTERS. 138
  6.1 Introduction .................................................................. 138
  6.2 The Tsetlin Filter ............................................................ 139
  6.3 The Krinsky Filter ............................................................ 146
  6.4 Simulation Methods And Results ..................................... 149
  6.5 Conclusions ................................................................. 155

CHAPTER 7:
FINAL REMARKS AND CONCLUSIONS ..................................... 157
  7.1 Overall Conclusion .......................................................... 157
  7.2 Comparison Of All Algorithms Presented ......................... 158
  7.3 Future Research Topics ................................................... 165

REFERENCES ..................................................................... 167
LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Pelc ½</td>
<td>22</td>
</tr>
<tr>
<td>Algorithm Feige</td>
<td>23</td>
</tr>
<tr>
<td>Algorithm Monrad</td>
<td>33</td>
</tr>
<tr>
<td>Algorithm Butterfly</td>
<td>34</td>
</tr>
<tr>
<td>Algorithm Points</td>
<td>35</td>
</tr>
<tr>
<td>Algorithm Majority</td>
<td>46</td>
</tr>
<tr>
<td>Algorithm Optimal Majority</td>
<td>54</td>
</tr>
<tr>
<td>Algorithm Consecutive</td>
<td>62</td>
</tr>
<tr>
<td>Algorithm Leader</td>
<td>67</td>
</tr>
<tr>
<td>Algorithm Stochastic Leader</td>
<td>70</td>
</tr>
<tr>
<td>Algorithm Random Choice</td>
<td>101</td>
</tr>
<tr>
<td>Algorithm Neighbour</td>
<td>102</td>
</tr>
<tr>
<td>Algorithm Move To Front</td>
<td>103</td>
</tr>
<tr>
<td>Algorithm Move To Back</td>
<td>109</td>
</tr>
<tr>
<td>Algorithm Move To Front And Back</td>
<td>111</td>
</tr>
<tr>
<td>Algorithm Transposition</td>
<td>115</td>
</tr>
<tr>
<td>Algorithm Transpose Forward Backward</td>
<td>118</td>
</tr>
<tr>
<td>Algorithm Exchange</td>
<td>120</td>
</tr>
<tr>
<td>Algorithm Tsetlin Filter</td>
<td>140</td>
</tr>
<tr>
<td>Algorithm Krinsky Filter</td>
<td>147</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1
Comparison of mean number of stochastic comparisons performed by the majority filter and optimal majority filter for equivalent filter parameters.......................... 59

Table 3.2
Simulation results for the majority filter for a failure probability of q = 0.2............. 75

Table 3.3
Simulation results for the optimal majority filter for a failure probability of q = 0.2..... 76

Table 3.4
Simulation results for the consecutive filter for a failure probability of q = 0.2......... 76

Table 3.5
Simulation results for the leader filter for a failure probability of q = 0.2............. 77

Table 3.6
Simulation results for the stochastic leader filter for a failure probability of q = 0.2... 77

Table 3.7
The convergence probabilities of the various filters introduced in Chapter 3............. 85

Table 4.1
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the majority filter. The initial failure probability was q = 0.49.......................................................... 91

Table 4.2
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the optimal majority filter. The initial failure probability was q = 0.49.......................................................... 92

Table 4.3
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the consecutive filter. The initial failure probability was q = 0.49.......................................................... 94
Table 4.4
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using different filters. In places where no error is given values are exact and were determined theoretically. .......................................................... 95

Table 4.5
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using different filters. In places where no error is given values are exact and were determined theoretically. .......................................................... 95

Table 4.6
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using different filters. In places where no error is given values are exact and were determined theoretically. .......................................................... 96

Table 4.7
Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the consecutive filter. The initial failure probability was \( q = 0.49 \). .............................................................................................................. 97

Table 5.1
Table listing the sixteen different cases that occur in the Move To Front Rule, the corresponding probability of each case occurring and the effect on the value of \( \pi P_j (t) \) ........................................ 106

Table 5.2
Table listing the sixteen different cases that occur in the Move To Front And Back rule, the corresponding probability of each case occurring and the effect on the value of \( \pi P_j (t) \) .............................................................................................................. 113

Table 5.3
Example of a scenario in which the Move To Front And Back list organizing rule, estimates the wrong precedence order for the values......................................................... 123

Table 5.4
Asymptotic precedence probabilities \( \pi P_j \) as calculated for the scenario given in Table 5.3 for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. Observe that the value of \( \pi P_4 \) for the Move To Front And Back................. 124

Table 7.1
Asymptotic disorder results for algorithms introduced in Chapter 2........................................ 160

Table 7.2
Results for stochastic sorting algorithms using the deterministic leader filter introduced in Chapter 3.............................................................................................................. 161
Table 7.3
Asymptotic disorder results for list organizing schemes introduced in Chapter 5. The random scheme was used to choose values for comparison. ................................................. 162

Table 7.4
Asymptotic disorder results for list organizing schemes introduced in Chapter 5. The neighbour scheme was used to choose values for comparison............................................. 163

Table 7.5
Asymptotic disorder results for list organizing schemes operating in conjunction with the ergodic deterministic Tsetlin filter introduced in Chapter 6. The random scheme was used to choose values for comparison............................................................. 164

Table 7.6
Asymptotic disorder results for list organizing schemes operating in conjunction with the ergodic deterministic Tsetlin filter introduced in Chapter 6. The neighbour scheme was used to choose values for comparison............................................................. 165
LIST OF FIGURES

Figure 1.1
Diagram of distributions for two values which are known only stochastically. Two values are presented stochastically from each distribution and compared. A 'failure' may occur when either value is picked from the overlapping shaded region. ........................................... 6

Figure 1.2
Two values, v_i and v_j, are stochastically presented and compared. In this case an error will occur since v_j is chosen as being less than v_i, and the means of the distributions are ordered oppositely - v_i is less than v_j................................................................. 6

Figure 2.1
Diagram of the extended noisy binary search tree developed in [Feige et al., 1990]. In this example the search list is [2, 5, 7].................................................................24

Figure 2.2
Diagram of all possible scenarios encountered by a single comparator. V represents the larger value, and v the smaller. .................................................................26

Figure 2.3
Diagram of a simple 4-sorter network α.................................................................27

Figure 2.4
Diagram of the network β which makes the simple 4-sorter network shown in Figure 2.3 1-tolerant.................................................................27

Figure 2.5
Plot of average disorder vs. number of stochastic comparisons performed by Feige's algorithm for different input parameters. The failure probability was set at q = 0.25. The list being sorted consisted of 32 values. The inset graph depicts t...........................................39

Figure 2.6
Plot of average disorder vs. number of stochastic comparisons performed by the Monrad Tournament for different success probabilities. The list being sorted consisted of 32 values.................................................................40

Figure 2.7
Plot of average disorder vs. number of stochastic comparisons performed by the Monrad, Wins, Points, and Butterfly tournament operating under different value picking schemes. The failure probability was set to q = 0.25. The list being sorted consisted of 32 values. ... 41
Figure 3.1
Diagram of a sorting mechanism which uses a deterministic filter.................................44

Figure 3.2
The transition map of the state space for the majority scheme. All horizontal transitions are with probability p, and all vertical transitions are with probability q, where \( p + q = 1 \). The states with bold lettering represent the final result states when \( 2M-1 \) stochastic comparisons are performed. States shown in italics denote final result states when \( 2M \) stochastic comparisons are made in total........................................................................52

Figure 3.3
The transition map of the result state space for the optimal majority scheme. All horizontal transitions are with probability \( p \), and all vertical transitions are with probability \( q \), where \( p + q = 1 \). The bold circles represent the final result states.........................57

Figure 3.4
Graph showing the number of stochastic comparisons performed vs. the filter parameter for both the majority filter and the optimal majority filter. The shaded region represents the range of answers for the optimal majority filter.................................................................60

Figure 3.5
Markov chain corresponding to the consecutive scheme. Stochastic comparisons are performed until M consecutive similar answers are obtained. The shaded states imply absorption.................................................................62

Figure 3.6
Markov chain corresponding to the leader scheme. Stochastic comparisons are performed until one stochastic answer has occurred M more times than the other. The shaded states represent absorption.................................................................66

Figure 3.7
Markov chain corresponding to the stochastic leader scheme. Stochastic comparisons are performed until the chain becomes absorbed. The shaded states represent absorption..............70

Figure 3.8
Graph of \( \Pr [D_p] \) vs. number of stochastic comparisons for the majority, optimal majority, consecutive, and leader filters. The failure probability in all cases was \( q = 0.2 \) …78

Figure 3.9
Graph of \( \Pr [D_p] \) vs. number of stochastic comparisons for the majority, optimal majority, consecutive, and leader filters. The failure probability in all cases was \( q = 0.4 \) …78

Figure 3.10
Graph of \( \Pr [D_p] \) vs. number of stochastic comparisons for leader, stochastic leader, optimal majority, and majority filters. The failure probability in all cases was \( q = 0.2 \) …79
Figure 3.11
Graph of average disorder vs. number of stochastic comparisons for majority, optimal majority, consecutive, and leader filters. In each case Insertion Sort was used to order a set of 32 values. The failure probability was set at $q = 0.2$, and the filter parameters were assigned so as to achieve a final accuracy of 0.99 for each comparison. .................................................. 80

Figure 3.12
Graph of average disorder vs. number of stochastic comparisons for majority, optimal majority, consecutive, and leader filters. In each case Shell’s Sort was used to order a set of 32 values. The failure probability was set at $q = 0.2$, and the filter parameters were assigned so as to achieve a final accuracy of 0.99 for each comparison. The inset graph is a magnification of the region near low disorder values. ........................................................... 81

Figure 3.13
Graph of average disorder vs. number of stochastic comparisons for Insertion Sort using a Leader filter with different filter parameters. The failure probability was set at $q = 0.25$. In each case a set of 32 values was sorted. The corresponding filtered success probability, $\alpha$, is given for each filter parameter. .............................................................. 82

Figure 3.14
Graph of average disorder vs. number of stochastic comparisons for Heap Sort using a leader filter with different filter parameters. The failure probability was set at $q = 0.25$. In each case a set of 32 values was sorted. The corresponding filtered success probability, $\alpha$, is given for each filter parameter. .............................................................. 83

Figure 3.15
Graph of average disorder vs. number of stochastic comparisons for Insertion Sort, Shell’s Sort, and Heap Sort. The failure probability was set at $q = 0.25$, a leader filter was used with parameter $M = 3$. In each case a set of 32 values was sorted. ............................. 84

Figure 4.1
The filtering process using a single filter................................................................. 87

Figure 4.2
The filtering process using a hierarchical structure of filters. The structure depicted contains two levels of filters. Although three filters appear in the first level they are in fact one and the same filter being used three times over. No purpose is served by using three physically distinct filters if each level contains the same type of filter. .................. 88

Figure 4.3
Diagram showing some intermediate level $i$ of the filtering structure, with incoming results, and an outgoing filtered result. The corresponding success probabilities are given for both sides. .......................................................... 89
Figure 4.4
Graph showing performance of different optimal majority filter structures. The curve is the result for a single filter level. Other filter structures are plotted as scattered points. The data for this graph is shown in Table 4.2.................................................................93

Figure 4.5
Graph showing performance of hierarchical consecutive filter structures and single majority, optimal majority, and leader filters. The failure probability was \( q = 0.49 \). The graph shows that consecutive filters arranged in a multi level structure outperforms majority and optimal majority filters, but not the leader filter. The data for this graph is given in Tables 4.4 to 4.6 .................................................................96

Figure 5.1
The neighbour scheme showing successive choice of values picked for comparison. An even number of values in the input list causes no problems. If we have an odd number of elements the scheme alternates between starting from the first and starting from the second values in the input list. ........................................102

Figure 5.2
Diagrammatic representation of the Move To Front list organizing scheme. Values 3 and 1 are picked and the smaller of the two is moved to the front. Subsequently values 2 and 4 are picked, and since 2 is already at the front it remains there unmoved ...............103

Figure 5.3
Diagrammatic representation of the Move To Back list organizing scheme. Values 3 and 1 are picked and the greater of the two is moved to the back. Subsequently values 2 and 4 are picked, and the process is repeated.................................................................109

Figure 5.4
Diagrammatic representation of the Move To Front And Back list organizing scheme. Values 3 and 1 are picked and the smaller of the two is moved to the front, while the larger is moved to the back. Subsequently values 2 and 4 are picked and the process is repeated.................................................................111

Figure 5.5
The Transposition list organizing scheme. Values 3 and 1 are picked and the smaller of the two is transposed forwards by one place. Subsequently values 2 and 4 are compared, and since 2 is already at the front of the list it remains there untransposed ......................115

Figure 5.6
The Transpose Forward And Backward list organizing scheme. Values 3 and 1 are picked and the smaller of the two is moved forwards by two places, the greater value is then moved backwards by three places. Subsequently values 2 and 4 are picked and the process is repeated. \( N_f = 2 \) and \( N_b = 3 \) in the above transpositions............117
Figure 5.7
The Transpose Forward And Backward list organizing scheme. Values 3 and 1 are picked and the greater of the two is moved backwards by three places, the lesser value is then moved forwards by two places. Subsequently values 2 and 4 are picked and the process is repeated. $N_f = 2$ and $N_b = 3$ in the above transpositions.

Figure 5.8
Diagrammatic representation of the exchange list organizing scheme. Values 3 and 1 are picked and are found to be out of order, and so they are exchanged. Subsequently values 2 and 4 are picked and found to be in the correct order so they remain unswapped.

Figure 5.9
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front list organizing scheme for different success probabilities. Observe that even with no errors the Move To Front rule does not achieve a zero disorder state. A set of 32 values was used in these results.

Figure 5.10
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A set of 32 values was used in these results, the failure probability was set at $q = 0.25$.

Figure 5.11
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Backwards, and Transpose Forwards And Backwards list organizing schemes. A set of 32 values was used in these results, the failure probability was set at $q = 0.25$.

Figure 5.12
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Transpose Forwards And Backwards by log $N$, and Exchange list organizing schemes. A set of 32 values was used in these results, the failure probability was set at $q = 0.25$.

Figure 5.13
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards And Backwards list organizing scheme for different values of $N_f$ and $N_b$. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$.

Figure 5.14
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards And Backwards list organizing scheme for different values of $N_f$ and $N_b$. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$. 

xvi
Figure 5.15
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at q = 0.  

Figure 5.16
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forward, Transpose Backward, and Transpose Forward and Backward list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at q = 0.  

Figure 5.17
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Backwards, and Transpose Forwards And Backwards rules. In each case the records were moved by log N, list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at q = 0.  

Figure 5.18
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Transpose Forwards And Backwards by log N, and Exchange list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at q = 0.  

Figure 5.19
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A set of 32 values was used in these results, the failure probability was set at q = 0.25, and the neighbour scheme was used to pick values.  

Figure 5.20
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Backwards, and Transpose Forwards And Backwards list organizing schemes. A set of 32 values was used in these results, the failure probability was set at q = 0.25, and the neighbour scheme was used to pick values.  

Figure 5.21
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Forwards And Backwards, and Exchange list organizing schemes. A set of 32 values was used in these results, the failure probability was set at q = 0.25, and the neighbour scheme was used to pick values.  

Figure 5.22
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at q = 0.25.
Figure 5.23
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$... 133

Figure 5.24
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Back list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$... 134

Figure 5.25
Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front And Back list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$... 134

Figure 5.26
Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Fowards And Backwards ($N_F = N_B = \log N$) list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$... 135

Figure 5.27
Plot of average disorder vs. number of stochastic comparisons performed for the Exchange list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$... 135

Figure 6.1
Diagrammatic representation of a sorting mechanism which uses a deterministic filter and a list organizing scheme... 139

Figure 6.2
State diagram of the ergodic Tsetlin automaton with $M$ states per action and a central state... 140

Figure 6.3
State diagram of the ergodic Tsetlin automaton with $M$ states per action and no central state... 144

Figure 6.4
State diagram of the ergodic Krinsky automaton with $M$ states per action and a central start state of no return... 147
Figure 6.5
Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating in conjunction with the ergodic Tssetlin filter for different parameters. The failure probability was $q = 0.25$, the list size was 32, and the random scheme was used to pick the values to be compared. The error-free case is included as well. ......................................................... 150

Figure 6.6
Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Tssetlin filter for different parameters. The failure probability was $q = 0.25$, the list size was 32, and the neighbour scheme was used to pick values. The error-free case is included as well. ......................................................... 150

Figure 6.7
Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Tssetlin filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values. The error-free case is included as well. ......................................................... 151

Figure 6.8
Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Tssetlin filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the neighbour scheme was used to pick values. The error-free case is included as well. ......................................................... 151

Figure 6.9
Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Krinsky filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used. The error-free case is also included. ......................................................... 152

Figure 6.10
Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Krinsky filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the neighbour scheme was used. The error-free case is also included. ......................................................... 153

Figure 6.11
Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Krinsky filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values. The error-free case is included as well. ......................................................... 153
Figure 6.12
Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Krinsky filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the neighbour scheme was used to pick values. The error-free case is included as well. ...................................................... 154

Figure 6.13
Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Tsetlin and Krinsky filters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values. ...................................................... 155

Figure 6.14
Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Tsetlin and Krinsky filters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values. ...................................................... 155
Chapter 1

Introduction To Stochastic Sorting

1.1 General Introduction

The field of Computer Science encompasses a vast variety of problems ranging from the simplest to those which are extremely complex. For instance, we may wish to find the maximum of a given set of numbers, or check if two quantities are equal. The solution to these rather trivial questions is quite straightforward and, in fact, people† come up with the correct answer without even giving it a second thought. However, computers offer an immense advantage in problems which involve computations which are more intricate. Consider, for example, the problem of plotting the graph of a function. A computer is able to do this both quickly and accurately, although the task, though simple, is tedious and can be painstaking. At the other end of the spectrum there exist problems which even computers find difficult to solve. Many of these problems are intrinsically more abstract and often even hard to formulate. Besides this, there may, for example, be no single correct

† The literature in learning theory [Narendra and Thathachar, 1989] gives us interesting cases when even primitive creatures such as rats, can be taught to learn the best of a set of actions.
solution to the problem, or there may be far too many feasible potential solutions, as in the traveling salesman problem†.

The problem of finding the maximum of a set of numbers can clearly be done in linear time. However, the solution to the traveling salesman problem, typically, takes a time which grows exponentially with the size of the map. Notice that in both these cases we are assuming that the data presented to the system is error free.

The complexity of the problem increases drastically when noise and errors have to be considered. If, for example, we were to compute the largest of a set of numbers, and the actual numbers themselves were erroneously represented, the question of what we mean by the maximum itself has to be considered. Indeed, if we were comparing two random variables A and B, it could very well occur that whereas, a single comparison of the variables yields the result that A is smaller than B, the average of multiple comparisons could lead to a contrary conclusion.

An identical scenario is encountered in the traveling salesman problem. If the inter-city distances (costs) were not error-free constants, the computations concerned would now involve random variables. Thus, computing the best path could require two levels of computation, the first of which estimated the mean inter-city costs, and the second which computed the optimal path utilizing the latter estimates. Indeed, to our knowledge, we are not aware of any reported solution to the ‘noisy’ traveling salesman problem.

In this thesis we will be investigating the realm of error and uncertainty. This entire realm is quite insidious to most computer programs which do not possess flexibility. Indeed, in many cases, a single error may cause an avalanche effect leading to multiple errors in subsequent computations.

Consider our simple problem of comparing two quantities to see which of them is the larger value. Operating in an error free environment we would simply compare these values once, and infer the solution from this comparison. The solution would however, be quite different if the computations involved were erroneous. If errors occur in executing the

† The travelling salesman problem is well known to computer scientists, and is described as follows. A salesman is given a map with the associated distances between the various cities. The problem involves finding a minimal tour for the salesman which visits every city on a map at most once.
latter comparison, or if the values themselves are 'fuzzy' (i.e., the individual values are not known with certainty), we would be very hesitant about basing our conclusion on the results of just a single comparison. In fact, it makes more sense to perform several comparisons and somehow conclude the final result from the ensemble of results obtained for each comparison.

To illustrate our example, suppose that the two values to be compared, A and B, are compared ten times and we find that, of these ten comparisons, A is less than B twice. We could then conclude that A is probably the larger of the two, and simultaneously be fairly certain about this conclusion. On the other hand, if A was less than B four times out of ten we would be less certain of our answer. We thus see that the introduction of error and ambiguity renders even this elementary problem non-trivial. Indeed, it is not hard to envision how the complexity of the traveling salesman problem would increase if the inter-city distances (costs) were ambiguous, or 'fuzzy', or even worse, if these costs are dynamically varying with time.

It is this realm of error and uncertainty that becomes the focus of this thesis. More specifically we will be considering the problem of sorting in the presence of erroneous computations or equivalently sorting in a noisy environment. We will refer to this problem as the problem of stochastic sorting, where the expression 'stochastic' implies that each value or comparison result is stochastically (or probabilistically) evaluated.

1.2 The Roots Of Stochastic Sorting And Its Applications

The problem of stochastic sorting originally arose from the study of the related problems involving noisy binary search procedures and fault tolerant sorting circuits. Pioneer work in these fields began ten to fifteen years ago when researchers delved into the problem of investigating the behaviour of the binary search when the comparisons were noisy. In a relatively short period of time the literature reported various results on the topic of noisy binary searching, and simultaneously various papers on the related topic of fault tolerant sorting circuits and algorithms were published. Research into both these problems has continued up to the present day.
The motivation for study of these problems initially arose from the design of large scale systems. One of the major problems encountered in all large scale systems is the presence of faulty elements - the larger the size of the system, the greater the probability that some fraction of it will fail to operate correctly. Since it may not always be possible to eliminate faulty components, the focus of the research involved developing algorithms that would function efficiently despite the errors. Since the most elementary problems that are tackled involve searching and sorting, this led to the growth in research in the fields of binary searching and fault tolerant sorting.

Stochastic sorting has also various applications in many other areas. Consider the case when a learning machine is required to order a set of actions under a generalized environment [Ng et al., 1993]. Traditionally, a learning system attempts to learn the best or optimal action which the environment offers. However, when we intend to order the actions in terms of their optimality, we wish to not only find the best action, but also to rank them in terms of a performance criterion, from best to worst. Observe that the simpler problem of finding the best action is a direct parallel to the problem of finding the maximum (or minimum) from a set of values under a stochastic environment.

Another application of stochastic sorting would involve the following scenario. Suppose we have N processes with process i requiring $V_i$ time units of execution time. Let us suppose that we are to rank the priority queue in terms of these execution times. If the $V_i$'s are random, the problem which we encounter is essentially one of stochastically sorting them. Other applications would involve sorting distributions by their mean values, when only random values of the variables are observable. A final application example would be one in the area of autonomous intelligent robots, or vehicles. Suppose we have a robot exploring an (unknown) environment. At each point in its decision making process it would be preferable if the system provided it with not only the best decision but also a list of plausible decisions ordered in terms of their optimality. This is because, in many cases, the robot may not be able to perform the best action due to its physical limitations. In such cases, a list of plausible decisions will definitely be advantageous.
1.3 Formal Problem Definition And Model

At the root of all types of sorting algorithms lies the comparison of two values. We intend to arrange the values in a sorted order by utilizing the information gained by comparing the magnitudes of these values.

At this juncture, it is appropriate to mention that there are currently two points of view which can be taken when we are considering the question of comparing two values. The perspective taken by researchers in the field of fault tolerant sorting, (and most researchers in the area of the noisy binary searching) is that the values being compared are known exactly, but the comparator executing the comparison returns an erroneous answer. In this case it can be said that whereas the values themselves are deterministic, there is a finite probability for the comparator returning an erroneous answer. This quantity is often referred to as the failure probability, and will be used throughout this thesis.

The other point of view, assumes that the comparisons themselves may be error-free, but considers the scenario where the values themselves are known probabilistically, and are best described by a distribution. The true value underlying each distribution can be considered to be the respective mean. Thus, when two stochastic values are compared, we essentially are presented with two deterministic (well-defined) values from the corresponding distributions and they are subsequently compared in an error-free manner. Indeed, in this case the failure probability becomes the probability that the comparison returns erroneous information about their means due to the stochasticity of the values themselves. A clarification of this is given in Figures 1.1 and 1.2.

In this thesis, we prefer to choose the latter point of view. This view is, perhaps, a little more general and is probably more adept to physical scenarios encountered in practical applications. Indeed, in this setting, it becomes more apparent that the failure probability need not be a constant, but may itself be some complex function of the two values being compared. Also, in this setting, it is clear that the problem of stochastic sorting and that of sorting a set of distributions in terms of their mean values are equivalent.

Whatever method we use to model our problem the end result is essentially the same. In either formulation, it is easy to see that the important quantity is the failure probability - the probability of receiving erroneous information from a comparison. Thus,
what contributes to the error, whether it be the comparator, the values themselves, or a combination of both of these is irrelevant to the formulation of our problem.

Throughout this thesis, we will denote the set of values to be sorted sort as the set \( \{ V_1, V_2, ..., V_N \} \), where \( N \) is the total number of values. We assume that the values are unique, although it can be shown that for identical values any comparison is faulty with probability \( \frac{1}{2} \). Under a stochastic environment these values will only be known

\[
\begin{align*}
&f_i & v_j \\
\text{Function describes distribution.} & \text{A value is chosen stochastically from the distribution.}
\end{align*}
\]

Underlying true values: \( V_i \) \( V_j \)

\text{Figure 1.1: Diagram of distributions for two values which are known only stochastically. Two values are presented stochastically from each distribution and compared. A 'failure' may occur when either value is picked from the overlapping shaded region. For more detail on this please refer to Figure 1.2.}

\[
\begin{align*}
&V_i & V_j & V_i & V_j \\
\text{Values chosen for comparison.}
\end{align*}
\]

\text{Figure 1.2: Two values, } v_i \text{ and } v_j \text{ are stochastically presented and compared. In this case an error will occur since } v_j \text{ is chosen as being less than } v_i \text{, and the means of the distributions are ordered oppositely - } V_i \text{ is less than } V_j \text{.}
stochastically, where \( v_i \) and \( v_j \) are respective occurrences of the random variables \( V_i \) and \( V_j \). The distribution for each random variable \( V_i \) is given by a function \( f_i \) (see Figure 1.1) unknown to the stochastic sorting algorithm. With no loss of generality we assume that each \( f_i \) satisfies:

\[
\int_{-\infty}^{\infty} f_i(x) \, dx = 1.
\]

To render the problem non-trivial, we also assume that the means of the distributions are themselves unknown to the sorting algorithms.

Whenever we compare \( V_i \) with \( V_j \) we shall say that we are performing a stochastic comparison. This is distinct from a deterministic comparison which would be done if we are comparing \( v_i \) with \( v_j \). A stochastic comparison of \( V_i \) with \( V_j \) involves obtaining (requesting) the values \( v_i \) and \( v_j \) from each distribution and comparing them deterministically. Thus, a stochastic comparison is essentially an abstract operation and will be denoted by the corresponding abstract operators \( \prec \) and \( \succ \). Observe that the operation \( V_i < V_j \) is not possible from within the stochastic environment, and that the operation \( v_i \prec v_j \) is exactly equivalent to \( v_i < v_j \) since \( v_i \) and \( v_j \) are deterministic.

We will refer to the mean of the distribution from which \( v_i \) is chosen as its true value. In a deterministic environment, \( V_i \) would be deterministically known and the corresponding distribution would be a single spike (Dirac delta function) at the value \( V_i \). Thus the task of comparing \( V_i \) with \( V_j \) would reduce to a deterministic comparison. Indeed, in this case, we will use the term deterministic sorting to describe such a process. Notice that in this scenario, the failure probability is zero.

Although it is not necessary in the implementation of any stochastic sorting scheme, we shall now briefly discuss the stochastic equality operator, \( \equiv \). As before, \( V_i \equiv V_j \) is an abstract operation, and is achieved by evaluating \( v_i = v_j \) where \( v_i \) and \( v_j \) are occurrences of \( V_i \) and \( V_j \). If the distributions for \( V_i \) and \( V_j \) are continuous then the result of the stochastic equality operator is always false. A result of ‘true’ can only be obtained if \( f_i \) and \( f_j \) both describe discrete distributions, since only then is it possible to pick \( v_i \) and \( v_j \) such that \( v_i = v_j \). Using the above definition of the stochastic equality operator we can generalize our definitions to yield the abstract operators \( \approx \) and \( \lessapprox \).
1.4 The Failure Probability

Typically, in the model currently used, in which the comparators are faulty and the values are deterministic, the failure probability is a quantity that is assumed to be known in the analysis. As opposed to this, under the model of obtaining stochastic values and error-free comparisons described in the previous section, each value is only described stochastically by its distribution. The true values are the means of the corresponding distributions. If two values $V_i$ and $V_j$ are described by their distributions $f_i$ and $f_j$ respectively, the failure probability $q_{i,j}$ for stochastically comparing $V_i$ and $V_j$ can always be found. In fact, if with no loss of generality, if $V_i$ is less than $V_j$, it can be easily shown that:

$$q_{i,j} = \int_{-\infty}^{\infty} f_i(v_i) \int_{-\infty}^{v_i} f_j(v_j) \, dv_j \, dv_i,$$

In addition, for the case when we compare $V_i$ to itself, the failure probability is $q_{i,i} = \frac{1}{2}$.

1.5 Outline Of Thesis

In Chapter 2 we will present a review of the literature related to our topic. The review will deal with the topics of noisy binary searching and fault tolerant sorting circuits. The depth of discussion on each paper reviewed will be dependent on the applicability of its results to this thesis. Previous work and algorithms which can be directly applied will be presented fully, and in detail; in addition, any improvements, or drawbacks to the algorithms examined will also be discussed. Simulation results of the implemented algorithms and comparisons with other algorithms will also be presented at the end of Chapter 2.

Chapter 3 will deal with the use of standard sorting algorithms like Shell's sort, or insertion sort under a stochastic environment. The information gained from stochastic comparisons will be filtered to reduce errors through the use of deterministic filters which will be introduced in the chapter. By filtering stochastic comparisons one is able to
essentially attenuate any errors, and obtain as accurate an answer as is desired. This filtered comparison information can than be utilized by any deterministic sorting algorithm. For each deterministic filter, a corresponding algorithm will be given, and a formal analysis will also be presented. At the end of the chapter we will present simulations which compare the filters under a wide range of parameters. The simulation results will further provide a means of comparing the performance of each stochastic sorting algorithm coupled with various filters, and operating under different stochastic environments.

In Chapter 4 we examine the use of structures of filters arranged hierarchically, rather than just in a single layer. It will be shown that such structures are sometimes advantageous. Simulations which support these conclusions will also be included in Chapter 4.

The algorithms presented in Chapter 3 assume that the stochastic values are stationary with respect to time. That is, the sorted solution remains the same through time, and the distributions do not change. For the case when the stochastic environment is not stationary, we shall develop several list organizing schemes to enhance the sorting. This is done in Chapter 5. The schemes presented here can, of course, can also be applied to stationary stochastic environments as well. Each of the list organizing schemes use information gained from stochastic comparisons to update an ordering of the values, and are able to present its current solution at any point in time. The solution presented by each scheme attempts to match the current optimal solution, which may itself keep changing in a non-stationary environment. Besides this, Chapter 5 will also deal with the investigation of how values to be compared can be chosen.

As a continuation of Chapter 3, in Chapter 6 we will introduce the use of ergodic filters which improve the performance of the list organizing scheme. These filters will then be used in tandem with the organizing schemes discussed in Chapter 5 to reorder a list of values, into its proper sorted order. Ergodic filters are used to retain the ability of the overall list organizing scheme to function under a non-stationary stochastic environment. The theoretical results obtained in this case too are validated with extensive simulations.
Chapter 7 concludes the thesis with an overall critical comparison of the various stochastic sorting algorithms. In this chapter we will also highlight future research directions in stochastic sorting and the related research avenues.

In all brevity, an overall survey of the novel results presented in this thesis is given below.

(i) Chapter 3 introduces several deterministic filters: majority, optimal majority, consecutive and leader filters. When two values are to be compared, the filters perform several comparisons and report an answer which is correct with probability higher than each individual comparison. All filters are proven to be asymptotically optimal - meaning that a correct answer may be obtained with probability as close to unity as desired. The leader filter has the best performance. The consecutive filter outperforms both majority filters for cases where the failure probability is less than 0.25. Experimentally its performance degrades drastically for higher failure probabilities. The optimal majority filter always outperforms the majority filter.

(ii) Stochasticized versions of the above filters are also examined briefly. The stochastic leader filter has been analyzed in depth. The stochastic versions of the filters exhibit poorer performance than their deterministic counterparts.

(iii) Chapter 4 introduces the concept of multi-layered filter structures. Such structures are shown to ameliorate performance for the consecutive filter. However, experimentally, the performance degrades for either majority filter. No improvement is obtained for the leader filter. Heirarchical filtering structures composed of consecutive filters are shown to outperform both majority filters, but not the leader filter.

(iv) Various list organizing methods are introduced and discussed in Chapter 5, such as the Move To Front, Exchange, and Transposition rules. An accuracy-speed tradeoff is observed - more destructive rules such as the Move To Front which move elements to the front of the list quickly reorganize the list but are unable to approach a sorted order. Less destructive rules like the Transposition rule take longer to converge but are able to more closely approximate the sorted order.
(v) To attenuate errors ergodic filters are introduced in Chapter 6. List organizing methods operating in conjunction with ergodic filters exhibit a marked increase in performance. Two ergodic filters are analyzed - the Tsetlin and Krinsky filters; the Tsetlin filter is shown to be superior. The Exchange rule operating with ergodic filters is able to completely sort a list of values. Furthermore, since list organizing schemes are ergodic, they are able to function in a non-stationary environment, and in particular, the Exchange rule tends to perform optimally.

Throughout this thesis, we will refer to results that we have derived as Lemmas or Theorems. Results obtained from the other previous research will be merely referred to as ‘Results’. The ideas and results presented in this thesis are the culmination of cooperative work between my supervisor, Dr. John Oommen and myself. This study has resulted in four potential journal publications [Oommen and Zgierski, 1993a; Oommen and Zgierski, 1993b; Oommen and Zgierski, 1993c; Oommen and Zgierski, 1993d].
CHAPTER 2

A REVIEW OF PREVIOUS
LITERATURE

2.1 General Review

The literature on the problem of stochastic sorting is scanty. Most of the relevant work done so far concentrates on two principal areas. The first is that of searching with errors, and the second that of designing fault tolerant sorting circuits by using a network of comparators.

Results obtained from studying noisy searching [Rivest et al., 1980; Pelc, 1989; Feige et al., 1990] can be easily applied to the problem of stochastic sorting. Indeed, since every member of the family of 'Insertion Sort' algorithms uses some kind of a searching strategy, a stochastic sort can be implemented by reckoning each search to be noisy. However, it is not so easy to apply the latter analytic results directly. In all the papers surveyed the noisy search assumes that the list being searched is sorted correctly. The resulting probability of finding the correct place of inserting an element thus hinges on the latter assumption. Observe that during the process of sorting by insertion, there is a larger probability that the list to be searched will not be perfectly ordered, rendering the analyses in [Rivest et al., 1980; Pelc, 1989; Feige et al., 1990] ineffective for our problem.
Chapter 2: A Review of Previous Literature

The work done on designing fault tolerant sorting circuits [Yao and Yao, 1985; Assaf and Upfal, 1990; Leighton et al., 1990; Leighton and Plaxton, 1991] also offers some insights into the problem of stochastic sorting. But, just as in the case of noisy searching, these do not provide all the answers sought after. A fault tolerant circuit designed to operate within some minimum depth - usually of $O(\log N)$, can be thought of as an algorithm which sorts in parallel using multiple processors. Instead of minimizing the total number of matches or comparisons carried out, the goal in these cases is to minimize the number of rounds or, rather, the depth of the circuit. It is clear that at least $N \log N$ comparisons are needed to sort $N$ numbers when the numbers are deterministically known. Parallel sorting algorithms and sorting circuits attempt to achieve the same efficiency, and this endeavour becomes more complex when the numbers themselves are only known stochastically. In this chapter we shall compare the two different methods in order to evaluate their relative strengths and weaknesses.

Aside from the areas of searching with errors and fault tolerant sorting circuits several tournament algorithms (such as the Monrad scheme) have also been reported in the literature [Block, 1987]. These schemes will also be discussed in this chapter.

We now present a brief survey of some of the most relevant results in the field of searching with errors, and of fault tolerant sorting circuits. The algorithms which are more pertinent to this thesis will be explained in greater detail. In addition, the applicability of the respective results to the problem of stochastic sorting will also be catalogued.

2.2 Searching with Errors

The problem of coping with erroneous information in search procedures has been studied by many pioneering researchers [Rivest et al., 1980; Pelc, 1989; Feige et al., 1990]. This problem can, perhaps, be most conveniently stated in terms of a two-person game between a Questioner and a Responder. The Responder first chooses some element $X$ from a given set, referred to as the search space. The Questioner attempts to identify $X$ by asking questions of some prescribed form. The Responder answers each question asked by the Questioner in a manner which may or may not be erroneous. Based on the assumption
that the total number of erroneous answers is 'controlled', the problem is one of finding an optimal searching strategy to be used by the Questioner.

Depending on the different rules governing this searching game, many variations on the problem exist. For example, the search space may be continuous or discrete. In the continuous case, the unknown number $X$ is usually chosen from the interval $[0,1]$, in which case the true value of $X$ can never be identified. As opposed to this, a solution can be found if we restrict the problem to be one of finding some interval in $[0,1]$ of size $\epsilon$ which contains $X$. In the discrete, case researchers have attempted to identify $X$ from some finite set $\{1, 2, ..., N\}$ or from some infinitely countable set.

In addition to this, there are also various scenarios for restricting the types of questions which the Questioner may ask. Since the search space is most often a set of numbers, the most common queries comprise of comparison queries. However, more general queries of the type "Is $X \in T$?" may also be asked, where $T$ is a subset of the search space.

Finally, the errors that can be encountered are also often controlled. In some of the research conducted the number of errors may be bounded by some constant, or they may depend on the size of the search space or on the total number of questions asked.

We shall now survey the literature on this subject, attempting to do this chronologically.

2.2.1 Results Due To Rivest et al.

One of the first papers in this field was due to Rivest and his colleagues [Rivest et al., 1980]. In [Rivest et al., 1980] the authors consider the problem of identifying some unknown value $X \in \{1, 2, ..., N\}$ using only comparisons of $X$ to constants. Among these comparisons the model permits a maximum of $E$ erroneous responses. This problem is equivalent to that of performing a noisy binary search where at most $E$ comparisons can fail. The results obtained in [Rivest et al., 1980] are related to the topic of stochastic searching in that, a noisy binary search is performed often when sorting using insertion techniques. The number of comparisons required to perform such a noisy binary search is
found to be \( \log N + E \log \log N + O(E \log E) \). The 'non-noisy' binary search would, of course, require only \( \log N \) comparisons.

Suppose there is an unknown number \( X \) which we wish to identify by asking some simple two-answer "Yes/No" questions. The objective in [Rivest et al., 1980] is to minimize the number of questions required in the worst case to identify this number given the possibility that some of the answers obtained might be wrong. Depending on the possible values of \( X \) and the natures of the "Yes/No" questions the problem takes on many forms. One can have the discrete case where \( X \) is a member of some finite set \( \{1, 2, ..., N\} \) or a continuous case where \( X \) is a member of the half-open interval \( (0,1] \). In the continuous case the goal is to identify \( X \) by finding some region \( A \) of size \( \epsilon \) in which \( X \) is known to lie. In either case \( U \) denotes the universe of possible values for \( X \). The type of questions that may be asked about \( X \) may include "Is \( X \in T \)?", where \( T \) is some subset of \( U \). Alternatively one may allow questions restricted to comparisons only, such as "Is \( X \leq a? \)", where \( a \in U \) [Rivest et al., 1980].

In [Rivest et al., 1980] a worst case analysis of such an identification problem was carried out under the assumption that up to \( E \) of the answers may be incorrect. The bound \( E \) may be a constant \( o(N) \), a function of \( N \), in the discrete case, and of \( \epsilon \) in the continuous case. The authors first show that there is a unique optimal strategy for the continuous problem with only comparison questions allowed. For a given number of questions, \( Q \), this strategy minimizes the worst case size of the region \( A \) which contains \( X \).

The authors then show that \( E \) extra comparisons are sufficient to cut down the region \( A \) into a single interval. From the results for the continuous case the discrete case is derived by setting \( \epsilon = 1/N \), whenever \( A \) becomes a single interval. This yields a comparison strategy for the discrete problem which uses no more than \( \log N + E \log \log N + O(E \log E) \) questions or comparisons.

We now present the analytic results from [Rivest et al., 1980] in all brevity. The proofs of the results were abstracted from the paper. The notation

\[
\binom{n}{m} \text{ is used to denote } \sum_{i=0}^{m} \binom{n}{i}.
\]
Result 2.1: For any two non-negative integers \( Q \) and \( E \) let \( \varepsilon(Q,E) \) denote the smallest \( \varepsilon \) such that \( Q \) arbitrary “Yes/No” questions about an unknown \( X \in (0,1] \), up to \( E \) of which may receive erroneous answers, are sufficient in the worst case to determine a subset \( A \) of \( (0,1] \) with \( X \in A \) and \( |A| \leq \varepsilon \). Then

\[
\varepsilon(Q,E) = \left( \left\lceil \frac{Q}{E} \right\rceil \right)^Q.
\]

Proof: The state of knowledge of the Questioner at any point during the questioning can be summarized by the number \( q \) of questions remaining and the \((E+1)\)-tuple \( A_q = (A_q^0, A_q^1, \ldots, A_q^E) \) defined by \( X \in A_q^e \) if and only if exactly \( e \) of the previous answers were incorrect. Clearly, we always have

\[
X \in \bigcup_{q=0}^{E} A_q^e.
\]

Define the weight \( w \) of a state \( A_q \) when there are \( q \) questions remaining by:

\[
w(q,A_q) = \sum_{e=0}^{E} \left( \left\lceil \frac{q}{e} \right\rceil \right) |A_q^e|.
\]

We now define an ‘adversary answering strategy’ to answer the next question. Suppose the question is “Is \( X \in T? \)”. A “Yes” answer results in a state described by \( yA_{q-1} = (yA_{q-1}^0, yA_{q-1}^1, \ldots, yA_{q-1}^E) \), with,

\[
yA_{q-1}^e = (A_{q}^e \cap T) \cup (A_{q}^{e-1} \cap T), \quad \text{for } 1 \leq e \leq E,
\]

and

\[
yA_{q-1}^0 = A_{q-1}^0 \cap T.
\]

Similarly a “No” answer will result in a state described by \( nA_{q-1} = (nA_{q-1}^0, nA_{q-1}^1, \ldots, nA_{q-1}^E) \), with,

\[
nA_{q-1}^0 = (A_{q}^e \cap T) \cup (A_{q}^{e-1} \cap T), \quad \text{for } 1 \leq e \leq E,
\]

and

\[
nA_{q-1}^1 = A_{q-1}^1 \cap T.
\]
and
\[
{nA_{q-1}}^0 = A_q^0 \cdot T.
\]

It is straightforward to verify that \(w(q-1, YA_{q-1}) + w(q-1, nA_{q-1}) = w(q, A_q)\), using the identity
\[
\binom{n+1}{m} = \binom{n}{m} + \binom{n}{m-1}.
\]

The adversary strategy consists in answering "Yes" if \(w(q-1, YA_{q-1}) \geq w(q-1, nA_{q-1})\) and "No" otherwise, thereby ensuring that \(w(q-1, A_{q-1}) \geq w(q, A_q)/2\).

The definitions of \(w\) and \(A_q\) imply that
\[
w(Q, A_Q) = \binom{Q}{E} \quad \text{and} \quad w(0, A_0) = \sum_{e=0}^{E} |A_0^e|.
\]

Hence no \(e = \sum_{e=0}^{E} |A_0^e|\) smaller than \(\binom{Q}{E} 2^{-Q}\) is achievable against the adversary answering in the manner described above.

The above analysis also shows that the best questioning strategy is to choose the next question "Is \(X \in T\)?" such that the two possible weights are equal. Any such strategy does, in fact, achieve a value of \(e = \sum_{e=0}^{E} |A_0^e|\) which is smaller than \(\binom{Q}{E} 2^{-Q}\) in the worst case. This can be done with a comparison "Is \(X \leq a\)?" because \(w(q-1, YA_{q-1})\) and \(w(q-1, nA_{q-1})\) are continuous functions of \(a\). Indeed, for \(a = 0\) we have
\[
w(q-1, YA_{q-1}) < w(q-1, nA_{q-1}),
\]
and for \(a = 1\) we have
\[
w(q-1, YA_{q-1}) \geq w(q-1, nA_{q-1}).
\]

Hence there is at least one value of \(a\) where the two weights are equal. Hence the result.
One can of course rephrase the above result into a statement involving the number $Q = Q(\varepsilon, E)$ of questions necessary to achieve a tolerance of $\varepsilon$ with up to $E$ erroneous answers, as follows:

$$Q(\varepsilon, E) = \min \left\{ Q' \mid E \geq \left( \frac{Q'}{E} \right)^2 \right\}.$$  

In the discrete case [Rivest et al., 1980] prove the following results:

**Result 2.2:** Consider the optimal comparison strategy for the continuous problem as described in the proof of Result 2.1. Then $E$ additional comparison questions asked after the end of the above strategy suffice to reduce the set $\bigcup_{e=0}^{E} A_0^e$ to a single interval.

**Proof:** The set $\bigcup_{e=0}^{E} A_0^e$ consists of a finite number of intervals, say $I_1, I_2, \ldots, I_r$, where $r \leq E+1$. Define depth $(I_j)$ to be the number of sets $A_0^e$, $0 \leq e \leq E$, that intersect $I_j$, and define depth $(A_0) = \sum_{j=1}^{r} \text{depth } (I_j)$. Using the fact that $A_0 = (A_0^0, A_0^1, \ldots, A_0^E)$ can be described completely by no more than $E+1$ previous "Yes" answers and $E+1$ previous "No" answers, it is not hard to see that depth $(A_0) \leq E+1$. The proof can then be completed by observing that any comparison question placed between two intervals reduces depth $(A_0)$ by at least unity, no matter what the answer is. $lacksquare$

**Result 2.3:** For any non-negative integer $E$ and positive integer $N$, let $Q(N, E)$ denote the number of comparison questions necessary in the worst case to identify an unknown $X \in \{1, 2, \ldots, N\}$ when up to $E$ of the questions may receive an erroneous answer. Then

$$\min \left\{ Q' \mid 2^{Q} \geq N \left( \frac{Q'}{E} \right)^2 \right\} \leq Q(N, E) \leq \min \left\{ Q' \mid 2^{Q' + E} \geq N \left( \frac{Q'}{E} \right)^2 \right\}.$$  

The above result also holds when $E$ is a constant as well as when $E$ is a function of $N$. With some tedious manipulations the inequalities given in this result can be shown to imply

$$Q(N, E) = \log N + E \log \log N + O(E \log N).$$
Proof: A detailed proof may be seen in [Rivest et al., 1980].

The results are only valid when E is a constant or some function of N. Due to this, they cannot be directly employed in our study of stochastic sorting, since, in our case, E would have to be a function of the total number of comparisons made.

2.2.2 Results Due To Pelc

In [Pelc, 1989] Pelc studies the problem of interactive searching in a set of numbers using comparison queries, under the assumption that each answer can be erroneous with a constant probability q and that a given reliability $0 < \rho < 1$ of the result is required. The search is considered in three versions, the continuous, the discrete bounded, and the discrete unbounded. Pelc proves that in all three cases the search is feasible for any N, which is the space size, and $\rho$ if and only if $q \neq \frac{1}{2}$. For $q \neq \frac{1}{2}$ an $O(\log N)$ searching algorithm is given in the continuous case and $O(\log^2 N)$ algorithms in the discrete cases. For $q < \frac{1}{2}$ or $q > \frac{1}{2}$, $O(\log N)$ algorithms are also given for each of the two discrete cases [Pelc, 1989].

Questions about the unknown X are always assumed to be in comparison form and errors in the answers occur independently with some probability $q$. The goal is to obtain a value for X that is correct with probability $\rho$. Pelc was interested in the feasibility of such a search, and in optimal search strategies. The length of any search strategy is the worst case number of queries necessary to carry it out, this length being considered as a function of the size of the search space. In the continuous case where we wish to find some set A of size $\varepsilon$ in which X is known to lie, the size is the number $N = \lceil 1/\varepsilon \rceil$. In the bounded discrete case, this size is simply N where the discrete set to search from is $\{1, 2, ..., N\}$.

We will now describe the six different types of searching games analyzed in [Pelc, 1989]. The first three games correspond to the probabilistic error setting described earlier.

Let $0 < q < 1$ be the probability of error, and $0 < \rho < 1$ the required reliability. The Responder gives an answer after each query which is incorrect with some probability $q$. 
The responses are stochastically independent. The Questioner has to carry out the search and identify the unknown $X$ correctly with probability $\rho$. We thus have:

(a) The continuous game $G_{q,\rho}([0,1], \varepsilon)$. The Responder thinks of a number $X \in [0,1]$ unknown to the Questioner who tries to find a set $A \in [0,1]$ of size $\leq \varepsilon$ such that $X \in A$. The Questioner may ask questions of the form "Is $X < a$?", where $a \in [0,1]$.

(b) The discrete bounded game $G_{q,\rho}(1, 2, \ldots, N)$. The Responder thinks of an integer $X \in \{1, 2, \ldots, N\}$ unknown to the Questioner who tries to identify it by asking questions of the form "Is $X < a$?", where $a \in \{1, 2, \ldots, N\}$.

(c) The discrete unbounded game $G_{q,\rho}(N)$, where $N$ is the set of positive integers. The Responder thinks of any positive integer unknown to the Questioner who tries to identify it by asking questions of the form "Is $X < a$?", where $a \in N$.

It should be noted that it is sufficient to consider the above games only for error probabilities in $(0, \frac{1}{2})$. If the Responder answers incorrectly with probability $1 > q > \frac{1}{2}$, the Questioner can always take the negation of the answer and assume the Responder is wrong with probability $0 < 1-q < \frac{1}{2}$. The respective games for $q$ and $1-q$ are thus equivalent due to the underlying symmetry [Pelc, 1989].

The next three games are slight modifications of the above. The error probability is again $0 < q < 1$, but it is assumed that, for an initial series of $m$ answers, where $m$ is equal to or exceeding a given parameter $M$, the number of errors cannot exceed $qm$. The Questioner has to assure that the determined value for $X$ is correct with probability $\rho=1$. The rest of the rules are exactly the same as before. We thus have:

(a') The continuous game $G^*_{q,M}([0,1], \varepsilon)$.

(b') The discrete bounded game $G^*_{q,M}(1, 2, \ldots, N)$.

(c') The discrete unbounded game $G^*_{q,M}(N)$.

In this set of three games, the role of the error parameter $q$ is different from the probabilistic setting described above and cannot be reduced by symmetry to only
considering cases when $0 < q \leq \frac{1}{2}$. For the last three games Pelc proves the following result:

**Result 2.4:** The Questioner wins each of the games $G^*_{q,M}([0,1], 1/N)$, $G^*_{q,M}\{1, 2, \ldots, N\}$, and $G^*_{q,M}(N)$ for all positive integers $N, M$ iff $q < \frac{1}{2}$. For $q < \frac{1}{2}$ he has a winning 'greedy' strategy with complexity $O(N^\alpha)$ and $\Omega(N^\beta)$, for some positive $\alpha, \beta$. 

This first result is based on a greedy algorithm in which the Questioner repeats each query until he is sure of the answer. For superior algorithms the following results can be shown to be true.

**Result 2.5:**
(a) If $q < \frac{1}{2}$, the Questioner can win the game $G^*_{q,M}([0,1], 1/N)$ for any $M$ and $N$ in time $O(\log N)$.
(b) If $q < \frac{1}{2}$, the Questioner can win the game $G^*_{q,M}\{1, 2, \ldots, N\}$ for any $M$ and $N$ in time $O(\log N)$.
(c) If $q < \frac{1}{2}$, the Questioner can find the unknown integer $X = N$ thus winning the game $G^*_{q,M}(N)$ for any $M$ and $N$ in time $O(\log N)$.

The results which are most relevant to the problem of stochastic sorting are ones describing the probabilistic game scenario. Again, all the proofs are omitted here but may be found in [Pelc, 1989].

**Result 2.6:** Let $q \leq \frac{1}{2}$. The Questioner wins each of the games $G_{q,\rho}([0,1], 1/N)$, $G_{q,\rho}\{1, 2, \ldots, N\}$, $G_{q,\rho}(N)$ with any fixed reliability $0 < \rho < 1$ and for any $N \in \mathbb{N}$ iff $q < \frac{1}{2}$. If $q < \frac{1}{2}$ he has a 'greedy' winning strategy using $O(\log^2 N)$ queries.

**Result 2.7:**
(a) If $q < \frac{1}{2}$, the Questioner can win the game $G_{q,\rho}([0,1], 1/N)$ for any $0 < q < 1$ and any $N$ in time $O(\log N)$.
(b) If $q < \frac{1}{2}$, the Questioner can win the game $G_{q,\rho}\{1, 2, \ldots, N\}$ for any $0 < q < 1$ and any $N$ in time $O(\log N)$.
(c) If $q < \frac{1}{2}$, the Questioner can find the unknown integer $X = N$ thus winning the game $G_{q,\rho}(N)$ for any $0 < q < 1$ and any $N$ in time $O(\log N)$. 

Of all the above results the ones that are most pertinent to our topic of stochastic sorting are Result 2.6 and Result 2.7 (b). The corresponding algorithm from [Pelc, 1989]
for the discrete bounded case when \( q < \frac{1}{2} \) is given below. The algorithm for the case when \( q < \frac{1}{2} \) is omitted since it restraints the failure probability excessively.

**Algorithm Pelc \( \frac{1}{2} \):**

*Aim:* An algorithm to find \( X \in \{1, 2, ..., N\} \) with error probability \( q < \frac{1}{2} \) and reliability \( 0 < \rho < 1 \). It uses \( O(\log^2 N) \) comparisons.

*Input:* N: size of search space.
q: error probability.
\( \rho \): reliability.

*Output:* Unknown element \( X \) correct with probability at least \( \rho \).

*Method:*

\[
\varepsilon = (\frac{1}{2} \cdot q)/2; \\
c = q(1-q)/\varepsilon^2; \\
m = \lceil \log N \rceil; \\
k = \lceil c/(1-\rho^l/m) \rceil; \\
\]

Perform usual binary search in \( \{1, 2, ..., N\} \) repeating each question \( k \) times and taking the majority answer.

End Algorithm Pelc \( \frac{1}{2} \).

### 2.2.3 Results Due To Feige et al.

In [Feige et al., 1990] the authors develop a noisy binary search algorithm which runs in \( O(\log N/Q) \) time, where \( N \) is the size of the list to be searched and \( Q \) is a tolerance parameter in \( (0, \frac{1}{2}) \). The algorithm computes the correct location to insert a new element \( X \) into a sorted list \( V = (V_1, V_2, ..., V_N) \) for all \( Q < \frac{1}{2} \) with probability which is greater than or equal to \( 1-Q \). The general model used in [Feige et al., 1990] is a comparison tree in which each node gives the correct answer with some probability \( \geq p \), where \( p \) is a fixed constant in \( (\frac{1}{2},1) \). The node faults are independent. The authors study the depth of the computation tree in terms of the tolerance parameter \( Q \in (0,\frac{1}{2}) \). At any instant the computation tree leads to a leaf yielding the correct answer at that instant with probability at least \( 1-Q \). The principles motivating the algorithm are described below.

A binary search tree is created in which each node represents an interval of the input vector \( V \). A leaf of the search tree represents an interval between two consecutive input
values. There are thus N+1 leaves with the \(i\)th leaf \((1 \leq i \leq N+1)\) representing the interval \((V_{i-1}, V_i)\). The boundary values are set as \(V_0 = -\infty\) and \(V_{N+1} = \infty\). Each internal node \(u\) of the tree represents the interval \((V_l, V_h)\), \(0 \leq l \leq h \leq N+1\), where \(V_l\) is the smallest value at a leaf subrooted at \(u\) and \(V_h\) the largest. The left child of \(u\) represents the interval \((V_l, V_z)\) and the right child \((V_z, V_h)\), where \(z = \lceil (l+h)/2 \rceil\).

In order to perform a noisy search the tree is extended such that each leaf \(V_l\) is the parent of a chain of length \(m' = O(\log N/Q)\). An example of such an extended tree is shown in Figure 2.1. All the nodes of this chain represent the same interval as the leaf. Let \(X\) denote the value being searched for in the tree. The search starts at the root of the tree, and advances or backtracks according to the results of the various comparisons. Upon reaching some node \(u\), the algorithm checks to see if \(X\) belongs to the interval represented by that node by comparing it to the two endpoints. If it does not, it implies that some inconsistent answers have been obtained, and the algorithm backtracks to the parent of \(u\). If \(X\) is found to be within the interval it is then compared to \(V_z\) - the central value of the interval of \(u\), and the algorithm proceeds to the proper child depending on the result of this comparison.

The algorithm is run for exactly \(m\) steps, where \(m = O(\log N/Q)\) and \(m < m'\). The endpoint of the chain is thus never reached. The outcome of the algorithm is the left endpoint of the interval represented by the node at which the search has terminated.

**Algorithm Feige:**

**Aim:** An algorithm to find the place of insertion of some value \(X\) into the sorted list \([1, 2, ..., N]\). Each comparison is erroneous with probability \((1-p)\). The correct position of insertion of \(X\) is found with probability \(1-Q\). The algorithm uses \(O(\log N/Q)\) comparisons.

**Input:**
- \(N\): size of search space.
- \(p\): probability of comparison giving correct answer.
- \(Q\): tolerance parameter - result is wrong with probability \(Q\).

**Output:** Place of insertion for \(X\).
Figure 2.1: Diagram of the extended noisy binary search tree developed in [Feige et al., 1990]. In this example the search list is \(\{2, 5, 7\}\).

**Method:**

1. \(l = 0\).
2. \(h = N + 1\).
3. \(m = \lceil \log N/Q \rceil \times \text{Multiplier} \) \(^*\) recall \(m = O(\log N/q)\), \(m < m' *\).
4. for \(i = 1\) to \(m\)
   1. if \(X \in (V_l, V_h)\) then
      1.1. \(z = \lceil (1+h)/2 \rceil\).
      2. if \(X \in V_z\) then
         2.1. \(h = z\).
      3. else
         3.1. \(l = z\).
   4. else
      4.1. go to parent, reset \(l\) & \(h\) accordingly if necessary.
endfor

End Algorithm Feige.
Indeed, the above algorithm satisfies Result 2.8:

**Result 2.8:** For every $Q < \frac{1}{2}$, Algorithm Feige computes the correct location of $X$ with probability $\geq 1 - Q$.

**Proof:** The proof is given in [Feige et al., 1990].

### 2.3 Fault Tolerant Sorting Circuits

All sorting networks presented in the papers reviewed in this section are built from comparators, where each comparator is a two input, two output device capable of sorting two numbers (see Figure 2.2). Most of the work deals with constructing sorting networks for $N$ inputs using a minimum number of comparators. Other goals in constructing fault tolerant sorting networks include minimizing the depth of the network. Although it was well known that at least $\Omega(N \log N)$ comparators were needed to sort $N$ numbers [Knuth, 1975]. The problem of determining a practical circuit to achieve this was left open. This problem was finally solved by Ajtai, Komlos, and Szemeredi in [Ajtai et al., 1983] who constructed a network for sorting $N$ inputs using $O(N \log N)$ comparators.

The work being done currently deals with designing sorting networks which tolerate faulty comparators. There are basically two types of comparator faults referred to as passive and destructive faults. Passive comparators do not compare the input values but, rather output them in exactly the same order as they came in. As opposed to this, destructive comparators are more dangerous. They actually output the input values in their reverse sorted order. They thus tend to unsort any sorting achieved previously. A diagram showing all four scenarios that a single comparator can encounter is shown in Figure 2.2. It is clear that passive faults only cause a problem in case (c), whereas, destructive faults cause problems in both cases (c) and (d).

Most sorting networks are viewed on a level by level basis. At each level a pair of values is compared at most once. The number of levels in the network is its depth. The sorting network can thus be thought of as an algorithm for sorting in parallel. We will now discuss the work done on fault tolerant sorting circuits in a chronological fashion.
2.3.1 Results Due To Yao and Yao

Perhaps one of the first papers dealing with the problem of fault tolerant sorting networks is the one due to Yao and Yao [Yao and Yao, 1985]. In it the authors study the use of redundancy to enhance reliability for sorting and related networks when they themselves are built from unreliable comparators. It is shown that if no more than k comparators are faulty in any sorting network, an additional k(2N-3) comparators are sufficient to render this sorting network reliable, where N is the size of the set to be sorted.

Consider a sorting network as shown in Figure 2.3, where from left to right each comparator \([i_k : j_k]\) is drawn as a vertical bar connecting the \(i^{th}\) and \(j^{th}\) lines. The input vector \(x = [x_1, x_2, ..., x_N]\) is fed in from the left end with line \(i\) carrying \(x_i\). When a comparator \([i_k : j_k]\) is encountered, the smaller of the two incoming numbers moves to the upper line, \(i_k\), and the larger to the lower line, \(j_k\). At the right end of the network the numbers should exit in sorted order. A sorting network for \(N\) input values is called an \(N\)-sorter if it correctly sorts all \(N!\) permutations of input values. The network shown in Figure 2.3 is clearly a 4-sorter.

The next step is to consider the situation when faulty comparators are present. The types of faults considered by Yao and Yao are exclusively passive faults. The effect of
having a passive fault comparator is the same as that of removing it completely from the network. We shall now discuss, in greater detail, the case when at most \( k \) comparators are passively faulty.

![Diagram of a simple 4-sorter network \( \alpha \).](image)

**Figure 2.3:** Diagram of a simple 4-sorter network \( \alpha \).

Let \( k \) be an integer greater than zero. A \( k \)-tolerant \( N \)-sorter is an \( N \)-sorter such that if any \( k \) or fewer of its comparators are removed the resulting network is still an \( N \)-sorter. Let \( S_k(N) \) be the minimum number of comparators needed in any \( k \)-tolerant \( N \)-sorter. It is elementary to see that \( S_k(N) \leq (k+1) S(N) \), since one can obtain a \( k \)-tolerant \( N \)-sorter by simply replacing every comparator in an optimal \( N \)-sorter with \( k+1 \) copies. In an attempt to improve on this the following result was developed in [Yao and Yao, 1985].

**Result 2.9:** If network \( \alpha \) is an \( N \)-sorter, then there exists an \( N \)-network \( \beta \) with \( k(2N-3) \) comparators such that \( \alpha\beta \) is a \( k \)-tolerant \( N \)-sorter.

**Proof:** Proof is given in [Yao and Yao, 1985]

A diagram of a 4-network \( \beta \) is given in Figure 2.4. The resulting network \( \alpha\beta \) is 1-tolerant. Appending \( \beta \) again would create \( \alpha\beta\beta \) - a 2-tolerant 4-sorter. As a consequence of Result 2.9 any \( N \)-sorter can be made \( k \)-tolerant by appending to it a network of \( O(kN) \) comparators.

![Diagram of the network \( \beta \) which makes the simple 4-sorter network shown in Figure 2.3 1-tolerant.](image)
The assumption that at most \( k \) comparators are faulty may be too restrictive for very large networks. A better model would be the stochastic fault model. In the latter we let \( 0 < \varepsilon, \delta < 1 \), and \( N \) to be a positive integer as before. An \( N \)-network \( \alpha \) is an \((\varepsilon, \delta)\)-stochastic \( N \)-sorter if the random \( N \)-network \( \alpha' \) obtained from \( \alpha \) by deleting independently each comparator with some fixed probability \( \delta' \leq \delta \), is an \( N \)-sorter with probability at least \( 1-\varepsilon \).

The most conventional method of achieving reliability is to replace each basic component by several unreliable components which simulate the basic component with high reliability. In this case, a series of \( m \) comparators in sequence would fail with probability \( \delta^m \). Suppose \( \alpha \) is an \( N \)-network with no faulty comparators, the network \( \beta \) obtained from \( \alpha \) by replacing each comparator with \( m \) comparators in series is called the canonical \( m \)-redundant network of \( \alpha \). The probability for \( \beta \) to be a network performing the same mapping as \( \alpha \), is at least \( (1-\delta^m)^N \), which is greater than \( 1-\varepsilon \) for large \( N \) if \( m > (\log (N/\varepsilon))/\log(1/\delta) \).

The stochastic model presented above can be easily applied to the problem of stochastic sorting. The only consideration that must be taken into account is that although both passive and destructive faults occur during stochastic sorting, Yao and Yao only consider passive faults in their paper. A fault tolerant sorting circuit which tolerates both passive and destructive faults is the topic of the paper by Assaf and Upfal [Assaf and Upfal, 1990] which will now be discussed.

2.3.2 Results Due To Assaf and Upfal

Assaf and Upfal developed a general technique for enhancing the reliability of sorting networks and other comparator-based networks [Assaf and Upfal, 1990]. Their technique converts any network that uses unreliable comparators to a fault tolerant network that produces the correct output with very high probability even when each comparator is faulty with some probability less than \( \frac{1}{2} \) independent of the other comparators. The depth of the fault tolerant network is a constant times the depth of the original network, and the width of the network is increased by a logarithmic factor.
Chapter 2: A Review of Previous Literature

The work of Assaf and Upfal essentially extends on the work of [Yao and Yao, 1985]. Whereas the latter studied sorting networks under a weak model of faults, the former consider more significant faults in which a faulty comparator may output its two input values in any arbitrary order. This, in effect, combines both passive and destructive faults into a single fault model. The goal is then to construct a fault tolerant network, under the present model of obtaining faults with the hope that the network generates the correct sorted list with a high probability, even when each comparator is faulty with some fixed probability less than \( \frac{1}{2} \) independent of the other comparators.

Since sorting networks usually use \( N \) registers - one for each input value, Assaf and Upfal instantly conclude that no \( N \)-register network can achieve their goal. Their reasoning is as follows: suppose register \( R_s \) is designated to hold the smallest input value. It is obvious that \( R_s \) must be compared at least once in the network. Let \( R_n \) represent the other register with which \( R_s \) is last compared to. If we have no more than \( N \) registers in the whole network then either \( R_s \) or \( R_n \) do not store the smallest element. In this case, if the comparator is faulty the network does not output the correct order. We thus have the following result:

**Result 2.10:** The failure probability of a fault tolerant network that sorts \( N \) elements with \( N \) registers cannot be smaller than the failure probability of each individual comparator.

**Proof:** Sketched above.

In order to obtain a smaller failure rate one must allow the network to use more than \( N \) registers. This is akin to essentially storing information about the results of previous comparisons. The network will then have two main sections. The first section simulates the comparators of the original sorting network. Through this, each register \( R_n \) of the \( N \) registers of the original network is simulated by a set \( R_n = \{ R_n^1, R_n^2, ..., R_n^m \} \) where \( m = \log N \).

A register \( R_n^k \) is said to store the correct value at a given stage in computation if it stores the same value as register \( R_n \) of the original network at the corresponding stage. To guarantee the correctness of the computation we require that at each step of the computation all but a fixed fraction of each set \( R_n \) store the correct value. When register \( R_i \) is compared with \( R_j \) in the original network, the new network compares each \( R_i^k \) with \( R_j^k \), for \( k = 1, 2, \ldots, m \).
..., m. Since a fraction of the comparators might be faulty, each comparison might increase the fraction of wrong values. At each stage there is a constant depth majority-expander component that reinforces the majority in each set \( R_n \), after simulating each comparison of the original network.

The second section of the network generates the output. The input to this are \( N \) sets, each with \( m = \log N \) values. With a high probability each set contains only a small fraction of values that are not the correct value for the set. The purpose of the second part of the network is to reduce the \( m \) values from each set into a single correct value. This is achieved through the use of a majority-preserving component. Details of how the majority-expander and majority-preserver components work are given in [Assaf and Upfal, 1985]. For the sake of completeness we present their final result in all brevity.

Result 2.11: Given a sorting network with depth \( d \) and width \( N \), there is a fault tolerant sorting network of depth \( O(d) \) and width \( O(N \log N) \) that sorts every list of \( N \) elements correctly with probability \( 1 - 1/N \), even if each comparison in the network is faulty with some fixed probability smaller than \( \frac{1}{2} \).

Proof: Proof is given in [Assaf and Upfal, 1985].

2.3.3 Results Due To Leighton et al.

In [Leighton and Plaxton, 1990] the authors analyze a natural k-round tournament over \( N=2^k \) players and demonstrate that it possesses a strong ranking property. The authors then exploit this ranking property to develop efficient parallel sorting algorithms.

The k-round tournament is a simple butterfly tournament where, in the first round, \( N/2 \) matches are played according to a random pairing of the \( N \) players. The following \( k-1 \) rounds are defined by recursively running two separate butterfly tournaments between the \( N/2 \) winners and \( N/2 \) losers in parallel. At the completion of such a butterfly tournament each player will have achieved a unique sequence of wins and losses of length \( k \). The players are ordered based on this sequence. Obviously the best player has a sequence of straight wins and the worst player a sequence of straight losses. The other players are
ranked according to the magnitude of the binary number resulting from the sequence by equating a win with a 0 and a loss with a 1 - the lower numbers representing better players.

There are many problems associated with this type of ranking. The most common one can be illustrated with a simple example. If the second-best player plays the best player before the final round it will not be assigned its true rank. In order for it to get its true rank the second-best player must meet the best player in the last round so that its sequence is 0...01. In fact, a player of rank \( n \), \( 1 \leq n \leq N \) can obtain at most \( k_l = \lfloor \log (n-1) \rfloor \) losses and at most \( k_w = \lfloor \log (N-n+1) \rfloor \) wins. Thus, the 19th best player among 32 players can be ranked to be anywhere from being the 4th best to the second last \( (k_l = 4, k_w = 3 \) leading to sequences of 00011 and 11110).

**Theorem 2.1:** Any player of true rank \( n \), \( 1 \leq n \leq N \), can obtain at most \( k_l = \lfloor \log (n-1) \rfloor \) losses and at most \( k_w = \lfloor \log (N-n+1) \rfloor \) wins, by competing in a butterfly tournament.

**Proof:** Since we have \( N \) players there will be at most \( \lceil \log N \rceil \) rounds in the butterfly tournament. Let each player \( n \), \( 1 \leq n \leq N \) have true rank with 1 being the best rank, and \( N \) being the worst. In order for player \( n \) to win as many matches as possible it must play as many worse players as possible (players with rank greater than \( n \)). There are exactly \( N-n+1 \) worse players than player \( n \) in the first round, out of which at most half will remain in the second round, and a quarter in the third. After \( \log (N-n+1) \) rounds none of these will remain and player \( n \) will lose. Thus, player \( n \) can win at most \( \lfloor \log (N-n+1) \rfloor \) matches since it may only play one match per round. A similar argument follows for the maximum number of losses. There are exactly \( n-1 \) better players than player \( n \) and thus player \( n \) can lose at most \( \lfloor \log (n-1) \rfloor \) matches.

Leighton and Plaxton eventually develop a sorting circuit which sorts all but a superpolynomially small fraction of the \( n! \) possible input permutations in 7.44 \( \log N \) rounds.

There is, however, a more complex problem when the matches between players are stochastic rather than deterministic (a better player need not necessarily always win) as is often the case in real life. Under stochastic environments the previous solution breaks down. To remedy this the authors of [Leighton et al., 1991] have constructed a different sorting circuit in which each comparator is allowed to fail with some constant probability \( q \).
Chapter 2: A Review of Previous Literature

A differentiation is made between passive and destructive faults. As before, passive faults simply output the two inputs in the same order - thus effecting no comparison. Destructive faults output them in the reversed, unsorted order. Destructive faults are thus harder to handle. The authors develop a $O(\log N \log \log N)$ depth passive-fault-tolerant circuit which sorts most permutations of $N$ inputs. In addition a $O(\log^2 N)$ destructive-fault-tolerant circuit is developed which approximately sorts most permutations of $N$ inputs. The authors also prove that any destructive-fault-tolerant circuit capable of approximate insertion must have depth $\Omega(\log^2 N)$.

2.4 Tournaments Algorithms

In [Block, 1987] the author analyzes many sorting algorithms used in sports tournaments, and develops improvements for several of them. Some of the most common tournaments use a points scale to rank the players in some order. An example of such a tournament is the Monrad tournament used in chess matches. Points are added or subtracted depending on whether a player wins or loses a game. Other tournaments consider only wins - the more wins a player has the higher it is ranked. Of course, many drawbacks exist in each of these philosophies.

We shall now present and discuss different types of tournaments in detail. The strongest point of these is the fact that a tournament algorithm remains exactly the same whether it is being performed within a deterministic or stochastic environment.

2.4.1 The Monrad Tournament

The Monrad tournament is the one being currently used in chess competitions. Initially, each player is placed in some random order - everyone starts off with zero points. Adjacent players on the points scale play each other, with the winner gaining half a point and the loser losing half a point. After every player has played once, one round is considered to be finished. The players are deterministically sorted according to their points and the next round begins [Block, 1987]. The treatment of a 'left over' player in the event of the total number of players being odd is not well defined. There are two choices, the left out player could simply wait until the next round or play its neighbour on the points scale -
in which case the latter player would have to play twice in a single round. In this thesis, we have implemented the tournament using the former option, thus ensuring that a player plays at most one match within a round. The Monrad algorithm is formally presented below:

**Algorithm Monrad:**

**Aim:** An algorithm to sort \( N \) numbers by using a Monrad tournament. The algorithm has complexity \( O(N \log N) \) per round due to deterministically sorting the \( N \) values according to their points.

**Input:**
- \( N \): number of players.
- \( V \): List of \( N \) values to be sorted.
- maxRounds: Maximum number of rounds to be performed.

**Output:** \( V \) sorted according to tournament results.

**Method:**

```plaintext
for i = 1 to N
  Points [i] = 0.
endfor

for eachRound = 1 to maxRounds
  for i = 1 to N step 2
    if \( V_i < V_{i+1} \)
      Points [i] = Points [i] + \( \frac{1}{2} \).
      Points [i+1] = Points [i+1] - \( \frac{1}{2} \).
    else
      Points [i] = Points [i] - \( \frac{1}{2} \).
      Points [i+1] = Points [i+1] + \( \frac{1}{2} \).
    endif
  endfor
endfor

Sort list \( V \), and array Points according to values in Points array.
```

End Algorithm Monrad.

Several drawbacks exist in the Monrad tournament. The foremost one is that the players never settle down to their correct positions but rather oscillate around them. This is true even under a deterministic environment. It may therefore be impossible to obtain a correctly sorted output even if we wanted to get one with a probability as close to 1 as desired. The points obtained by the best and worst players tend to increase and decrease in an uncontrollable way, even asymptotically, while the points of the intermediate players remain within fixed limits. This behaviour is similar to that of the butterfly tournament.
where the best and worst players can be accurately determined but a correct ordering of the intermediate players is not so easily achieved.

2.4.2 The Butterfly Tournament

The Butterfly tournament discussed in Section 2.3.3 was used by Leighton et al., in fault tolerant sorting; but it is a tournament algorithm in its own right, and we shall formally present it here. In the butterfly tournament early wins carry a lot more weight than subsequent wins resulting in its erratic points system. The drawbacks of the butterfly tournament have already been discussed in Section 2.3.3. As in the case of the Monrad tournament, the best and worst players can be always found accurately in a deterministic environment. However, a correct ordering of the intermediate players, is generally speaking not possible.

Algorithm Butterfly:

Aim: An algorithm to sort N numbers by using a Butterfly tournament. The algorithm has complexity O(N log N) per round due to deterministically sorting the N values according to their points.

Input: N: number of players. For simplicity we assume N is a power of 2.
V: List of N values to be sorted.
maxRounds: Maximum number of rounds to be performed.

Output: Sorted V according to Points.

Method:
for i = 1 to N
    Points [i] = 0.
endfor
for R = (maxRounds-1) to 0 by -1
    for i = 1 to N step 2
        if \( V_i < V_{i+1} \)
            Points [i] = Points [i] + 2^R.
        else
            Points [i+1] = Points [i+1] + 2^R.
        endif
    endfor
    Sort list V, and array Points according to values in Points array.
endfor
End Algorithm Butterfly.
2.4.3 The Points Tournament

The Points tournament may count the total number of victories and losses for each player and ranks them accordingly. As before, each player is initially placed in some random rank, every player starts off with zero points. Players receive points for each victory and lose points for each defeat. If the players are chosen to play each other in a way analogous to the Monrad tournament and the number of points received for a win is equal to the number of points lost for a defeat the Points tournament becomes the Monrad tournament discussed in Section 2.4.1.

If every player plays all the other players under a deterministic environment, a correct ordering is always possible in the Points tournament. Of course, this same does not hold true for a stochastic environment. neither does this hold when each player does not play all the others.

The players can also be chosen randomly or otherwise. A more detailed discussion of how this can be achieved is given in Section 5.2. The points tournament algorithm is now presented below:

**Algorithm Points:**

**Aim:** An algorithm to sort $N$ numbers by using a Points tournament. The choice of values for comparison is done according to any pre-determined scheme. The algorithm has complexity $O(N)$ per round. The final complexity is $O(N \log N)$ due to the deterministic sort at the end.

**Input:**
- $N$: number of players.
- $V$: List of $N$ values to be sorted.
- maxRounds: Maximum number of rounds to be performed.

**Output:** Sorted $V$ according to Points.

**Method:**

```plaintext
for i = 1 to N
    Points [i] = 0.
endfor
for eachRound = 1 to maxRounds
    pick \( \frac{N}{2} \) distinct pairs \((i,j)\) from \(\{1,2,\ldots,N\}\) according to some scheme s.t. \(i \neq j\).
```

for eachPair
    if \( V_i < V_j \)
        Points \([i]\) = Points \([i]\) + WinPoints.
        Points \([j]\) = Points \([j]\) - LossPoints.
    else
        Points \([j]\) = Points \([i]\) + WinPoints.
        Points \([i]\) = Points \([i]\) - LossPoints.
    endif
endfor
Sort list \( V \), and array Points according to values in Points array.
End Algorithm Points.

A similar tournament, which we will call the Wins tournament, and which counts only victories is also frequently used in real life. For example, most sport teams playing in leagues are ranked by points which they receive for wins and ties - no points are received for a loss. The Wins tournament can be emulated by the Points tournament by setting LossPoints to zero in, and WinPoints to 1.

Our simulation results seem to indicate that the Wins tournament is inferior to the Points tournament, which penalizes a team for a loss. However, this is only true when the players for each game are picked randomly.

This terminates our discussion on tournament algorithms. We shall now critically compare the various algorithms presented in this chapter, proposing improvements whenever possible.

2.5 Improvement To Pelc’s Algorithm

In this section we shall first of all show that Pelc’s Algorithm (Algorithm Pelc \( \frac{1}{2} \)) presented in Section 2.2.2 can be drastically improved. The most important improvement is that we have obtained a better lower bound on the value of \( k \) - the number of times each comparison must be repeated. The improvements that we will report come from results obtained in Chapter 3 (Section 3.2). These results 3.2 dictate that the minimum value of \( k \) necessary to achieve a reliability of \( p \) is given by \( k = 2M_{\text{min}} - 1 \), where we have found \( M_{\text{min}} \) is:
\[ M_{\min}(\alpha) = \text{Min} \left\{ M \mid \frac{B_{(1-q)}(M,M)}{B(M,M)} \geq \alpha \right\}, \]

where \( B_{(1-q)}(M,M) \) and \( B(M,M) \) are the incomplete and complete Beta functions respectively. The value \( \alpha \) is the desired reliability of a set of \( k \) comparison operations and is equal to \( \rho^{1/m} \).

The improvement on the lower bound of \( k \) can be best seen by an example. For simplicity let us assume \( m = 1 \). If \( q = 0.2 \) and the desired reliability is \( \rho = 0.99 \), then \( k = 1067 \) when calculated by Algorithm Pelc ½. However \( k = 13 \) when calculated by the above formula. The improvement is obvious. Another example involves a list of length 32, whence \( m = 5 \). For a failure probability of \( q = 0.25 \) we wish to obtain a reliability of \( \rho = 0.95 \). Algorithm Pelc ½ yields a lower bound value of \( k = 1176 \); our bound yields the value of \( k \) to be 19.

Furthermore, Algorithm Pelc ½ uses the standard majority scheme - this means that \( k \) comparisons are performed and the majority is taken. We will see, in Chapter 3, that schemes which are far superior to the majority scheme can be obtained. A quick improvement would involve the use of an 'optimal majority' scheme which will be presented in Section 3.3. The improvement lies in the fact that we need not always play all \( k \) games to find the majority. For example, suppose five games are being played. If one player wins the first three, playing the two remaining games is a wasted effort since we know that the other player has no chance of catching up.

Algorithm Pelc ½ can thus be improved even more by using an optimized majority scheme. Additionally, we shall see that an even better improvement can be achieved by using a 'leader' scheme which will be introduced in the next chapter. The leader scheme is itself superior to the optimized majority scheme, and its use would further improve the algorithm.

We will now continue with the presentation of simulation results for the rest of the algorithms presented in this chapter. Simulation results for Pelc's improved algorithm will be presented in Chapter 3, since it is basically an Insertion Sort which uses a deterministic filter.
2.6 A Comparison Of Presented Algorithms

In this section we will present simulation results for the algorithms presented in this chapter. The algorithms were implemented in C, and run on a Sparc/Sun station. For each algorithm a set of 100 independent experiments was carried out. Statistical means were used to obtain the standard deviation error.

Since we are dealing with the problem of sorting, we would like to have some kind of metric which would quantitatively tell us how much order or sorting exists within a list of values. One such metric is that of disorder, and has been used extensively in Block's work [Block, 1987]. The disorder of any list of values is given by:

\[ D = \sum_{i=1}^{N} | \text{pos}(V_i) - \text{ord}(V_i) |^2, \]

where \( \text{pos}(V_i) \) is the position or index of the value \( V_i \) in the list, and \( \text{ord}(V_i) \) is the position of the value \( V_i \) in the corresponding sorted list. The disorder of a sorted list is clearly zero. Maximum disorder occurs when the list is sorted in its reverse from its desired order. For a list of \( N \) values the maximum disorder can be shown to be:

\[ D_{\text{max}} = \sum_{i=1}^{\lceil N/2 \rceil} 2 (N - 2i + 1)^2. \]

We thus define the percentage disorder as the ratio of equation (2.5.1) to equation (2.5.2) or:

\[ \hat{D} = \frac{D}{D_{\text{max}}} = \frac{\sum_{i=1}^{N} | \text{pos}(V_i) - \text{ord}(V_i) |^2}{\sum_{i=1}^{\lceil N/2 \rceil} 2 (N - 2i + 1)^2}. \]

The value of \( \hat{D} \) thus ranges from 0 for a sorted list, to 1 for a reversely sorted list. Values in between indicate quantitatively how close a list of values is to being sorted. The
percentage disorder is particularly good since it not only takes into account values in wrong positions but also how far away they are from their true positions. Other metrics for measuring the 'sortedness' of a list of values do, of course, exist, we have opted to use this one in our simulations.

Results for Feige's algorithm are shown in Figure 2.5. The algorithm is able to fully sort a list. For example, in our simulations, a list of 32 values was fully sorted 54% of the time, after about 1200 stochastic comparisons, even though 25% of all comparisons were erroneous. After about 2500 stochastic comparisons, the algorithm was able to fully sort a list of 32 values 98% of the time.

![Disorder vs. Number of Stochastic Comparisons](image)

**Figure 2.5:** Plot of average disorder vs. number of stochastic comparisons performed by Feige's algorithm for different input parameters. The failure probability was set at $q = 0.25$. The list being sorted consisted of 32 values. The inset graph depicts the magnification of the individual curves for values of disorder close to zero.

The Monrad tournament was simulated for various failure probabilities, and the results are shown in Figure 2.6. The final disorder values obtained by the Monrad
tournament are quite good, although the list never attains a truly sorted order as predicted in Section 2.4.1. Even with no errors (p=1.0) the asymptotic disorder reached by the Monrad tournament is about 0.0012. For a success probability of p = 0.75 the asymptotic disorder reached hovers around 0.006 and takes about 3000 stochastic comparisons to reach this value.

Figure 2.7 compares results for the Monrad, Wins, Points, and Butterfly tournaments. The Wins tournament counts only victories, as was mentioned earlier. For the Points tournament the parameters were set as WinPoints = LossPoints = 1. For both the Wins and Points tournaments two different schemes were used in choosing values for comparison - the random scheme which picks values with equal probability, independently of each other, and the neighbour scheme which picks values in a manner analogous to the Monrad tournament. Performance results for the six resulting tournaments are plotted in Figure 2.7.

![Graph showing disorder vs. number of stochastic comparisons]

**Figure 2.6:** Plot of average disorder vs. number of stochastic comparisons performed by the Monrad Tournament for different success probabilities. The list being sorted consisted of 32 values.
Figure 2.7: Plot of average disorder vs. number of stochastic comparisons performed by the Monrad, Wins, Points, and Butterfly tournament operating under different value picking schemes. The failure probability was set to $q = 0.25$. The list being sorted consisted of 32 values.

Notice that the Butterfly Tournament has little hope of achieving a sorted order, although it does provide a reasonable amount of ordering. The Points tournament outperforms the Wins tournament when values to be compared are picked randomly, this is expected since it makes use of more information, namely, the losses.

When the neighbour scheme is used to pick the values to be compared the Wins, Points, and Monrad tournaments all perform equally well. In fact, the Points Tournament operating under the neighbour scheme and the Monrad Tournament are equivalent algorithms.

2.7 Conclusions

This chapter served mostly as a review of pertinent literature in topics related to stochastic sorting. Papers from the fields of noisy binary searching and fault tolerant sorting were condensed, and all the relevant algorithms and results presented. The
applicability of the various results, to stochastic sorting were also discussed. Also, an improvement to Pelc’s algorithm was suggested.

In this chapter we also considered several well known tournament algorithms. They were all implemented and tested. Simulations of the different algorithms were carried out in order to provide a common basis on which they could be compared. To evaluate their performance the disorder metric was used to quantitatively describe how ‘sorted’ a list of values is. The various algorithms were compared on how well they could sort in the presence of errors, and how fast they could accomplish this goal i.e., how many stochastic comparisons they required.

The simulation results demonstrated, that all the tournament algorithms were unable to achieve a perfectly sorted order, although they did tend to decrease the overall disorder of the list substantially. It was found that, in the limit, the different tournament algorithms were essentially equivalent (under the neighbour scheme), except for the Butterfly tournament which has the poorest performance in all cases. When the values to be compared were randomly chosen for comparison the Wins tournament was less effective than the Points tournament. Selecting values randomly generally resulted in a decreased performance. However, in the limit, the random scheme was eventually able to attain a lower disorder.

Starting with the next chapter we will begin the presentation of the new results that we have obtained in the field of stochastic sorting.
3.1 Introduction

One way to sort under a stochastic environment is to adapt previous deterministic algorithms to work under the new conditions. The easiest and most straightforward way is to somehow convert stochastic comparisons into deterministic ones. Although, strictly speaking this cannot be done, we can, however, obtain the correct deterministic result with a very high probability. There are many ways in which several stochastic comparisons of the same two input values can be converted into a single deterministic comparison result. The most common way to do this is through the use of repetition. For example, we could repeat the stochastic comparison many times and reckon that the majority result is the correct one. Another method would be to use some kind of learning scheme to learn the 'more correct result' of the two possible outcomes for each comparison. Once a deterministic result is obtained we can employ any one of the many deterministic sorting algorithms - like insertion sort, bubble sort, Shell's sort, heap sort, or others to actually achieve the stochastic sorting. We refer to this strategy as one which uses a deterministic
filter. A diagrammatic representation of stochastic sorting using deterministic filters is given in Figure 3.1.

![Diagram of sorting mechanism](image)

**Figure 3.1:** Diagram of a sorting mechanism which uses a deterministic filter.

Various ways of converting stochastic comparison results into a deterministic result will now be presented, and discussed. First of all, we shall refer to the probability that the stochastic comparison results in a correct answer as \( p \), where \( p \) belongs to the half-open interval \( (\frac{1}{2}, 1] \). The probability that the stochastic comparison results in an erroneous answer will be given by \( q = 1 - p \). Typically, when two values, \( V_i \) and \( V_j \) are being compared, the probability \( p \) is often a function of their difference, or in the most general case, a function of both \( V_i \) and \( V_j \). In view of this, we define \( p_{ij} \) to be the probability that a correct stochastic answer is obtained upon comparing \( V_i \) and \( V_j \), and \( q_{ij} \) to be the corresponding failure (or error) probability. In order to ease calculations we set \( p = \min \{ p_{ij} \} \), for all \( i \) and \( j \). Computations which utilize this value of \( p \) will, for the most part, overestimate the failure probability for all stochastic comparisons except one, and will thus yield a conservative bound on the performance of the sorting mechanism.

Each stochastic comparison results in any of two answers. The correct one, occurring with probability \( p \), will be denoted by \( S_p \). The erroneous one, occurring with probability \( q \) will be denoted by \( S_q \). Similarly, the deterministic filter also results in any of two answers. The correct deterministic answer will be denoted by \( D_p \), and the incorrect one by \( D_q \). Thus, although \( \Pr \{ S_p \} = p \), and \( \Pr \{ S_q \} = q \), the probabilities \( \Pr \{ D_p \} \) and \( \Pr \{ D_q \} \) are typically functions of \( p \) and the parameter of the filter - say \( M \), where \( M \) is usually related to the number of stochastic comparisons being done. It would be desirable that in the limit as \( M \to \infty \), \( \Pr \{ D_p \} \) should tend to unity. Additionally we would like these probabilities to converge to their asymptotic values as quickly as possible.
Chapter 3: Stochastic Sorting Using Deterministic Filters

We define a scheme as asymptotically optimal if

\[
\lim_{M \to \infty} \Pr [D_p] = \begin{cases} 
1, & \text{if } p > \frac{1}{2} \\
0, & \text{if } p < \frac{1}{2} \\
\frac{1}{2}, & \text{if } p = \frac{1}{2}
\end{cases}
\]  

(3.1.1)

Throughout this thesis we will be making extensive use of a special function called the Beta function, \( B(z,w) \), used in probability and statistics. This function is defined as:

\[
B(z,w) = \int_0^1 t^{z-1} (1-t)^{w-1} \, dt.
\]  

(3.1.2)

As well, as the incomplete Beta function defined below, will repeatedly be used:

\[
B_k(z,w) = \int_0^k t^{z-1} (1-t)^{w-1} \, dt.
\]  

(3.1.3)

In the next sections we will present some deterministic filters and analyze each rigorously to examine its efficiency, and speed of convergence. We close the chapter with simulation results of each deterministic filter which will rank the filters in terms of efficiency. In addition, actual simulations of stochastic sorting using these filters will also be presented.

3.2 Majority Filter

The most commonly used method for obtaining reliable results from faulty information is through the use of repetition. The concept of majority uses exactly this. We perform a stochastic comparison between the same two input values \( k \) times and take the majority result as the correct one. In the first scenario, all of the \( k \) stochastic comparisons are always performed. We shall now analyze this scenario.

For simplicity we let \( k = 2M-1 \), where \( M \) is a positive integer and is a parameter of the majority filter; \( k \) is thus always odd. This assumption is justified by the fact that draws are not permitted in the majority scheme.
The algorithm for the majority filter is quite simple and is shown below. This particular example illustrates how a deterministic ‘greater-than-comparison’ would be implemented in terms of stochastic ‘greater-than-comparisons’. As mentioned in Chapter 1, generally speaking, if the values are real numbers an equality operator is not defined under a stochastic environment. However, if the number of stochastic comparisons is even and the two types of results occur an equal number of times, the deterministic result is an equality. In this case, to be impartial, we will assign the value to be \( D_p \) with probability \( \frac{1}{2} \) and \( D_q \) with probability, also \( \frac{1}{2} \).

For the sake of simplicity we ignore the case when an equality is obtained. However, if \( 2M \) stochastic comparisons are performed, the result of the majority filter could be an equality, but we shall show the non-trivial result that even in such a case an equally likely tie breaker would result in the same values for \( \Pr \{ D_p \} \) and \( \Pr \{ D_q \} \).

Algorithm Majority:

Aim: An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the majority deterministic filter.

Input: \( M \): Majority parameter. Answer occurring \( M \) or more times is majority answer.
\( \text{Value1}, \text{Value2} \): Values to be compared.

Output: A deterministic answer using stochastic comparisons. A stochastic comparison is represented by \( \gg \).

Method:

\[
\text{NumGreaterAnswers} = \text{NumLesserAnswers} = 0.
\]

for \( i = 1 \) to \( 2M-1 \)

if (\( \text{Value1} \gg \text{Value2} \)) then

\[
\text{NumGreaterAnswers} = \text{NumGreaterAnswers} + 1.
\]

else

\[
\text{NumLesserAnswers} = \text{NumLesserAnswers} + 1.
\]
endif

endfor

return (\( \text{NumGreaterAnswers} > \text{NumLesserAnswers} \)).

End Algorithm Majority.

Theorem 3.1: If \( p \) is the probability of obtaining a correct stochastic answer, then the majority filter with parameter \( M \) will yield a correct deterministic answer with probability

\[
\Pr \{ D_p \mid \text{majority filter used} \} = \frac{B_p(M,M)}{B(M,M)},
\]
where, \( B(.,.) \) and \( B(.,.) \) are the incomplete and complete Beta functions respectively. This probability increases monotonically with \( M \). Furthermore, the scheme is asymptotically optimal.

**Proof:** For the given values of \( p \) and \( q \) we shall prove the above expression for \( \Pr [D_p | \text{majority filter being used}] \) and also the asymptotic result that:

\[
\lim_{M \to \infty} \Pr [D_p | \text{majority filter being used}] = \begin{cases} 
1, & \text{if } p > \frac{1}{2} \\
0, & \text{if } p < \frac{1}{2} \\
\frac{1}{2}, & \text{if } p = \frac{1}{2}
\end{cases}
\]

(3.2.1)

Recall that \( k = 2M - 1 \). For the sake of ease of notation, let \( N = 2M \) (i.e., \( N = k + 1 \)). To obtain a correct deterministic answer for this comparison using the majority scheme we must obtain a majority of correct stochastic answers. Hence,

\[
\Pr [D_p | \text{majority filter being used}] = \Pr [i \text{ correct stochastic answers } | i \geq \lceil k/2 \rceil]
\]

\[
= \sum_{i=\lceil k/2 \rceil}^{k} \binom{k}{i} p^i q^{k-i}
\]

(3.2.2)

and,

\[
\Pr [D_q | \text{majority filter being used}] = \Pr [i \text{ wrong stochastic answers } | i < \lceil k/2 \rceil]
\]

\[
= \sum_{i=0}^{\lceil k/2 \rceil} \binom{k}{i} p^i q^{k-i}
\]

(3.2.3)

Clearly, the sum of both these probabilities is unity, since,

\[
\sum_{i=0}^{k} \binom{k}{i} p^i q^{k-i} = (p+q)^k = 1.
\]

(3.2.4)

Using the notation that \( N = k + 1 \) (thus \( N \) is always even), we substitute into equation (3.2.3) to obtain,

\[
q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^i.
\]

where \( e = \frac{p}{q} \)

(3.2.5)
Similarly equation (3.2.4) transforms into,

\[ q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^i. \]  

(3.2.6)

Evaluating equation (3.2.3) is equivalent to evaluating (3.2.5). Furthermore, since (3.2.6) sums to unity, we can evaluate (3.2.5) by equating it to the ratio of (3.2.5) and (3.2.6). Let \( H_N \) denote this ratio of equation (3.2.5) to (3.2.6). From [Oommen and Hansen, 1984] we know that:

\[ H_N = \frac{q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^i}{q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^i} = \frac{B_q(N, N)}{B(N-1, 2)} = \frac{B_q(M, M)}{B(M, M)} . \]

(3.2.7)

and thus,

\[ \Pr \left[ D_q \right | \text{majority filter being used}] = \frac{B_q(M, M)}{B(M, M)}. \]

(3.2.8)

The monotonicity follows from the properties of the incomplete Beta function. Also, we know from [Oommen and Hansen, 1984] that \( \lim_{N \to \infty} H_N = 1 \), when \( 0 < e < 1 \). Indeed, this is the case when \( p < \frac{1}{2} \). Thus the probability of obtaining a correct answer must tend to zero.

For the case when \( p > \frac{1}{2} \) we transform equation (3.2.2) into:

\[ q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^i = q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^{i+\frac{N}{2}} = q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^{i+\frac{N}{2}}. \]

(3.2.9)

By reversing the order of summation (3.2.9) becomes

\[ q^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} e^{N-1-i} = p^{N-1} \sum_{i=0}^{N-1} \binom{N-1}{i} \left( \frac{1}{e} \right)^i. \]

(3.2.10)
where \(0 < \frac{1}{e} < 1\). We also transform equation (3.2.4) into:

\[
p^k \sum_{i=0}^{k} \left( \frac{k}{i} \right) \left( \frac{1}{e} \right)^{k-i} = p^k \sum_{i=0}^{k} \left( \frac{k}{i} \right) \left( \frac{1}{e} \right)^{i} = p^k \sum_{i=0}^{k} \left( \frac{k}{i} \right) \left( \frac{1}{e} \right)^{i} = p^{N-1} \sum_{i=0}^{N-1} \left( \frac{N-1}{i} \right) \left( \frac{1}{e} \right)^{i}.
\]

(3.2.11)

As before we take the ratio of equation (3.2.10) to (3.2.11) to obtain \(G_N\):

\[
G_N = \frac{p^{N-1} \sum_{i=0}^{N-1} \left( \frac{N-1}{i} \right) e^{i}}{p^{N-1} \sum_{i=0}^{N-1} \left( \frac{N-1}{i} \right) e^{i}} = \frac{B_p\left(\frac{N}{2}, \frac{N}{2}\right)}{B\left(\frac{N}{2}, \frac{N}{2}\right)}, \text{ where this time } e = \frac{q}{p}.
\]

(3.2.12)

From these, we obtain,

\[
\Pr \{D_{p} \text{ majority filter being used} \} = \frac{B_p\left(M, M\right)}{B\left(M, M\right)}.
\]

(3.2.13)

As before \(\lim_{N \to \infty} G_N = 1\), when \(0 < e < 1\), which is always true if \(p > \frac{1}{2}\). Thus, in this case, the probability of obtaining a correct answer tends to unity.

Finally, when \(p = q = \frac{1}{2}\) equation (3.2.3) simplifies to: \(q^{N-1} \sum_{i=0}^{N-1} \left( \frac{N-1}{i} \right)\), and we know that

\[
\sum_{i=0}^{m-1} \left( \frac{2m-1}{i} \right) = \frac{1}{2} \sum_{i=0}^{2m-1} \left( \frac{2m-1}{i} \right) = 2^{2m-2},
\]

(3.2.14)

and thus,

\[
q^{N-1} \sum_{i=0}^{N-1} \left( \frac{N-1}{i} \right) = 2^{N+1} 2^{N-2} = \frac{1}{2}.
\]

(3.2.15)

Hence Theorem 3.1 is proved.

The importance of this theorem lies not only in the fact that the asymptotic probability of obtaining a correct deterministic answers approaches unity, but more so in
the closed form expression presented in equation (3.2.13). Since Beta functions are well tabulated, we are able to quickly calculate the expression for Pr [D_p] for any odd number of stochastic comparisons if we are given the filter parameter, M.

It is possible to rewrite equation (3.2.2) in a form which allows us to evaluate the minimal M required to obtain a desired degree of accuracy using the majority filter. Quite simply, this is obtained as:

\[ M_{\text{min}}(\alpha) = \min \left\{ M \left| \sum_{i=0}^{M-1} \binom{2M-1}{i} p^i q^{2M-1-i} \geq \alpha \right. \right\}, \tag{3.2.16} \]

or alternately, in terms of the Beta functions as,

\[ M_{\text{min}}(\alpha) = \min \left\{ M \left| \frac{B_p(M,M)}{B(M,M)} \geq \alpha \right. \right\}. \tag{3.2.17} \]

where \( \alpha \) is the desired accuracy and belongs to the interval \((\frac{1}{2}, 1)\). For example, when \( p = 0.8 \), to obtain an accuracy of 0.99, \( M_{\text{min}} \) would be equal to 7.

It is quite clear that the majority filter will always perform \( 2M-1 \) stochastic comparisons. A more efficient version of this filter will be presented in the following section. We close this section with an interesting theorem which completes our analysis of the majority filter.

Interestingly enough, performing \( k+1 \) stochastic comparisons (recall that \( k \) is odd) would not alter \( \text{Pr} [D_p] \) or \( \text{Pr} [D_q] \). This is definitely not obvious because, if an even number of stochastic comparisons are performed, the set of states representing the \( D_p \) and \( D_q \) events will be changed even though the condition for a tie breaker is ignored. The following theorem proves this statement.

**Theorem 3.2:** With regard to the majority filter, performing \( 2M-1 \) stochastic comparisons is equivalent to performing \( 2M \) stochastic comparisons if, in the event of a tie, the outcome of the filter is assigned the value \( D_p \) or \( D_q \) with equal likelihood.

**Proof:** For the case when \( k = 2M-1 \) we rewrite equation (3.2.2) by reversing the order of summation to obtain:
Chapter 3: Stochastic Sorting Using Deterministic Filters

\[
P(D_p | \text{majority filter being used}) = \sum_{i=0}^{M-1} \binom{2M-1}{i} q^i p^{2M-1-i}.
\]

(3.2.18)

Now, for the case when we perform an extra stochastic comparison, \(k = 2M\), and we have,

\[
P(D_p | \text{majority filter being used}) = \sum_{i=0}^{M-1} \binom{2M}{i} q^i p^{2M-i} + \frac{1}{2} \binom{2M}{M} q^M p^M.
\]

(3.2.19)

The extra term added on corresponds to the case when \(M\) results of type \(S_p\), and \(M\) results of type \(S_q\) have occurred. Clearly we do not have a majority, and we opt to choose either \(D_p\) or \(D_q\) with equal probability.

To prove the equality of equations (3.2.18) and (3.2.19) we let \(<i,j>\) denote the result after \(i+j\) stochastic comparisons, indicating that \(i\) stochastic responses of type \(S_p\) and \(j\) stochastic responses of type \(S_q\) have occurred. The majority scheme will thus have the state space described in Figure 3.2. The filter starts at state \(<0,0>\) with probability 1 and keeps requesting stochastic comparisons until it reaches one of the final states represented in bold (see Figure 3.2).

Equation (3.2.18) is nothing more than the sum of the probability masses of states \(<2M-1-i, i>\), where \(i \in \{0, 1, \ldots, M-1\}\), since each of these states will result in the deterministic answer \(D_p\). Similarly, equation (3.2.19) is the sum of the probability masses of states \(<2M-i, i>\), where \(i \in \{0, 1, \ldots, M-1\}\) and half the mass associated with \(<M, M>\).

Now, suppose \(2M-1\) stochastic comparisons have occurred and we are in some state \(<2M-1-i, i>\). Another stochastic result will put us in state \(<2M-i, i>\) or state \(<2M-1-i, i+1>\).

If \(i < M-1\), all the probability mass is transferred to states which still result in the \(D_p\) answer. Thus the total probability is conserved and remains unchanged no matter how it is distributed. The only exception is when \(i = M-1\). In this case, a transition is made from state \(<M, M-1>\) to state \(<M, M>\). Let \(Pr[<i, j>]\) denote the asymptotic probability mass of converging to the state \(<i, j>\). Using the first passage probabilities we know that,

\[
Pr[<M, M>] = p \Pr[<M-1, M>] + q \Pr[<M, M-1>]
\]
Figure 3.2: The transition map of the state space for the majority scheme. All horizontal transitions are with probability $p$, and all vertical transitions are with probability $q$, where $p + q = 1$. The states with bold lettering represent the final result states when $2M-1$ stochastic comparisons are performed. States shown in italics denote final result states when $2M$ stochastic comparisons are made in total.

In order for the total probability mass to be conserved in the states corresponding to the $D_p$ answer we need to show that the probability mass lost, $q \Pr [\langle M, M-1 \rangle]$, is equal to the probability mass gained, which is $\frac{1}{2} \Pr [\langle M, M \rangle]$.

Now, $\Pr [\langle M, M-1 \rangle]$ is given by:

$$\left(\frac{2M-1}{M-1}\right) q^{M-1} \frac{M}{p}. \quad (3.2.20)$$

Likewise $\Pr [\langle M, M \rangle]$ is:
\[
\binom{2M}{M} q^M p^M.
\]

We thus have the probability mass lost given by:
\[
q \binom{2M-1}{M-1} q^{M-1} p^M = \binom{2M-1}{M-1} q^M p^M.
\]

and the probability mass gained by:
\[
\frac{1}{2} \binom{2M}{M} q^M p^M = \frac{1}{2} \left[ \binom{2M-1}{M} + \binom{2M-1}{M-1} \right] q^M p^M
\]
\[
= \frac{1}{2} \left[ \binom{2M-1}{M-1} + \binom{2M-1}{M-1} \right] q^M p^M = \binom{2M-1}{M-1} q^M p^M.
\]

Thus the total probability mass is conserved and we have,
\[
\sum_{i=0}^{M-1} \binom{2M-1}{i} q^i p^{2M-1-i} = \sum_{i=0}^{M-1} \binom{2M}{i} q^i p^{2M-i} + \frac{1}{2} \binom{2M}{M} q^M p^M.
\]

Hence the theorem.

3.3 Optimal Majority Filter

A non-trivial improvement can be easily made to the previous majority scheme. It is quite obvious that all 2M-1 stochastic comparisons need not always be done in order to find the majority answer. As soon as any one answer is obtained M times we have a majority since the other answer can only be obtained at most M-1 times. It is therefore unnecessary to perform the remaining stochastic comparisons. We thus modify the majority scheme so that the number of stochastic comparisons is a variable (albeit, random) ranging from M to 2M-1. It is interesting to note that the converging accuracy (i.e. the probability of D_p) of the two majority schemes is identical. But, clearly, this majority scheme is more efficient. The algorithm for the optimal majority filter will now be presented.
Algorithm Optimal Majority:

Aim: An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the optimal majority deterministic filter.

Input: M: Majority parameter. Answer occurring M or more times is majority answer. Value1, Value2: Values to be compared.

Output: A deterministic answer.

Method:

NumGreaterAnswers = NumLesserAnswers = 0.
while (NumGreaterAnswers < M) and (NumLesserAnswers < M) do
    if (Value1 >= Value2) then /* stochastic comparison */
        NumGreaterAnswers = NumGreaterAnswers + 1.
    else
        NumLesserAnswers = NumLesserAnswers + 1.
    endif
endwhile
return (NumGreaterAnswers > NumLesserAnswers).
End Algorithm Optimal Majority.

We shall now prove that this majority scheme has the same properties as the previous naive one.

Theorem 3.3: If p is the probability of obtaining a correct stochastic answer, then the optimal majority filter with parameter M will yield a correct deterministic answer with probability:

\[
\Pr[D_p | \text{optimal majority filter being used}] = \frac{B_p(M,M)}{B(M,M)}.
\]

This quantity increases monotonically with M. Also, the scheme is asymptotically optimal.

Proof: As usual, we consider the case when \( \Pr[S_p] = p \), and \( \Pr[S_q] = q \). We shall first derive the expression independently, without referring to the previous majority filter. We let \( <i,j> \) denote the result after \( i+j \) stochastic comparisons, indicating that \( i \) correct and \( j \) incorrect stochastic answers have been received. It is clear that this majority scheme has the space state described in Figure 3.3.

The optimal majority scheme starts at state \( <0,0> \) with probability 1 and keeps requesting stochastic comparisons until it reaches one of the final states represented by the
bold circles in Figure 3.3. The deterministic answer is clearly of the form \(<M,i>\), or \(<i,M>\) where \(i < M\). Clearly, this probability is given by:

\[
\Pr [\langle M,i \rangle] = \binom{M+i-1}{i} p^{M-i} q^i.
\] (3.3.1)

The probability of obtaining a correct deterministic answer is thus the sum of the probability masses of all states \(<M,i>\), for \(0 \leq i \leq M-1\). Therefore,

\[
\Pr [D_p | \text{optimal majority filter being used}] = \sum_{i=0}^{M-1} \binom{M+i-1}{i} p^{M-i} q^i.
\] (3.3.2)

Indeed, using Lemma 2.3 of [Oommen et al., 1990] we can see that if

\[
f_n(p) = p^n \sum_{k=0}^{n-1} \binom{n+k-1}{k} q^k,
\] (3.3.3)

then

\[
\lim_{n \to \infty} f_n(p) = \begin{cases} 
1, & \text{if } p > \frac{1}{2} \\
0, & \text{if } p < \frac{1}{2} \\
\frac{1}{2}, & \text{if } p = \frac{1}{2}
\end{cases}
\] (3.3.4)

Clearly (3.3.2) is equivalent to \(f_n(p)\). Hence the asymptotic result.

For finite values of \(M\), notice that if \(I[.]\) is the indicator function of the event in parentheses, then by definition:

\[
I[D_p | \text{majority filter being used}] \Rightarrow I[D_p | \text{optimal majority filter being used}].
\] (3.3.5)

Thus, if \(D_p\) has occurred because \(2M-1\) comparisons have been made, clearly \(D_p\) will be the decision of the optimal majority filter. In addition observe that,

\[
I[D_p | \text{optimal majority filter being used}] \Rightarrow I[D_p | \text{majority filter being used}].
\] (3.3.6)

Equation (3.3.6) is true because, if \(D_p\) has occurred due to the operation of the optimal majority filter, the original majority filter will have to report \(D_p\) as well, in view of the fact that, the maximum value of comparisons supporting \(D_q\) will be definitely bounded by \(M-1\).
Using the set equality of the underlying events it is clear from equations (3.3.5) and (3.3.6) that

\[ 1 \{D_p \mid \text{majority filter being used} \} \leftrightarrow 1 \{D_p \mid \text{optimal majority filter being used} \} \quad (3.3.7) \]

and so equation (3.2.13) will also give us the quantity we are looking for. This completes the proof.

As before we can easily find the expression for \(M_{\min}(\alpha)\) - the minimum value for the parameter \(M\) required to obtain an accuracy of \(\Pr \{D_p \geq \alpha\}\). From equation (3.3.2) we find:

\[ M_{\min}(\alpha) = \min \left\{ M \left| \sum_{i=0}^{M-1} \binom{M+i-1}{i} p^i q^{M-i} \geq \alpha \right. \right\}. \quad (3.3.8) \]

Both majority schemes produce the same deterministic answer with the same probability since they both represent the exact same physical event in the underlying event space. Thus we can equate equations (3.3.2) and (3.2.18). Combining this with (3.2.13) yields the combinatorial equality:

\[ \sum_{i=0}^{M-1} \binom{M+i-1}{i} p^i q^{M-i} = \sum_{i=0}^{2M-1} \binom{2M-1}{i} p^i q^{2M-1-i} \]

\[ = \frac{B_M(M,M)}{B(M,M)}. \quad (3.3.9) \]

Apart from the physical significance of equation (3.3.9), equation (3.2.23) from Theorem 3.2 gives us an interesting (non-trivial) combinatorial equivalence. Due to this the expression for \(M_{\min}(\alpha)\) can also be obtained by equations (3.2.16) and (3.2.17) presented in the previous section. As before, to obtain an accuracy of 0.99 with \(p = 0.8\) we need \(M_{\min}(0.99) = 7\).

In the case of the optimal majority filter, the average number of stochastic comparisons performed varies randomly from \(M\) to \(2M-1\). We shall now consider the expected value of the number of comparisons that are performed by this filter.

**Theorem 3.4:** The expected number of stochastic comparisons performed by the optimal majority filter is asymptotically linear with the coefficient:
Proof: For any value of $p$ we know that

$$\Pr[<M,i>] = \binom{M+i-1}{i} p^i q^{M-i}.$$  \hspace{1cm} (3.3.10)

Similarly, for $0 \leq i \leq M-1$,

$$\Pr[<i,M>] = \binom{M+i-1}{i} p^i q^{M-i}.$$  \hspace{1cm} (3.3.11)

Figure 3.3: The transition map of the result state space for the optimal majority scheme. All horizontal transitions are with probability $p$, and all vertical transitions are with probability $q$, where $p + q = 1$. The bold circles represent the final result states.

Note that exactly $M+i$ stochastic comparisons will be performed whenever states $<M,i>$ and $<i,M>$ are reached. Thus, the probability of performing exactly $M+i$ stochastic
comparisons, is clearly the sum of the probabilities of reaching states \(<M,i>\) and \(<i,M>\) since they are mutually exclusive. Hence,

\[
\Pr \{\text{performing } M+i \text{ stochastic comparisons}\} = \binom{M+i-1}{i} p^{M+i} q^i + \binom{M+i-1}{i} p^i q^M \\
= \binom{M+i-1}{i} (p^{M+i} + p^i q^M)
\]

(3.3.12)

The expected value of the number of stochastic comparisons is therefore,

\[
E \{\text{# st. cmps.}\} = \sum_{i=0}^{M-1} \Pr \{\text{performing } M+i \text{ stochastic comparisons}\} (M+i)
\]

(3.3.13)

We further analyze equation (3.3.13) by expansion to yield.

\[
\sum_{i=0}^{M-1} \binom{M+i-1}{i} p^M q^i (M+i) + \sum_{i=0}^{M-1} \binom{M+i-1}{i} q^M p^i (M+i)
=
M \sum_{i=0}^{M-1} \binom{M+i}{i} p^M q^i + M \sum_{i=0}^{M-1} \binom{M+i}{i} q^M p^i
\]

(3.3.14)

\[
= 1 \frac{M}{p} \sum_{i=0}^{M} \binom{M+i}{i} p^{M+i} q^i - M \binom{2M}{M} p^M q^M + \frac{1}{q} M \sum_{i=0}^{M} \binom{M+i}{i} q^{M+i} p^i - M \binom{2M}{M} p^M q^M
\]

(3.3.15)

Finally, using equation (3.3.9) we obtain,

\[
E \{\text{# st. cmps.}\} = \frac{1}{p} \frac{M B_p(M+1,M+1)}{B(M+1,M+1)} + \frac{1}{q} \frac{M B_q(M+1,M+1)}{B(M+1,M+1)} - 2M \binom{2M}{M} p^M q^M
\]

(3.3.16)

We can easily see that equation (3.3.16) becomes linear as \(M \to \infty\). Depending on the value of \(p\), one of the Beta function ratios tends towards zero, due to equation (3.2.1) the other to unity, and the negative term tends to zero due to the dominant exponent. Hence the theorem.
Table 3.1: Comparison of mean number of stochastic comparisons performed by the majority filter and optimal majority filter for equivalent filter parameters.

<table>
<thead>
<tr>
<th>M</th>
<th>Majority</th>
<th>Optimal Majority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>p = 1.0</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>3.00</td>
<td>2.00</td>
</tr>
<tr>
<td>3</td>
<td>5.00</td>
<td>3.00</td>
</tr>
<tr>
<td>4</td>
<td>7.00</td>
<td>4.00</td>
</tr>
<tr>
<td>5</td>
<td>9.00</td>
<td>5.00</td>
</tr>
<tr>
<td>6</td>
<td>11.00</td>
<td>6.00</td>
</tr>
<tr>
<td>7</td>
<td>13.00</td>
<td>7.00</td>
</tr>
<tr>
<td>8</td>
<td>15.00</td>
<td>8.00</td>
</tr>
<tr>
<td>9</td>
<td>17.00</td>
<td>9.00</td>
</tr>
<tr>
<td>10</td>
<td>19.00</td>
<td>10.00</td>
</tr>
</tbody>
</table>

From equation (3.3.13) or (3.3.16) we are able to compare the efficiency of both majority filters by calculating the mean number of stochastic comparisons that will be performed for the same filter parameter. Table 3.1 lists the results. As could have been anticipated, the optimal majority filter always outperforms the majority filter. It performs best when \( p \) approaches unity, and is at its worst when \( p = \frac{1}{2} \). On the average, the latter filter performs the same number of stochastic comparisons for values \( p \) and \( 1-p \), although the corresponding deterministic answers will be complementing each other. Thus, when \( p = 0.8 \), for the filter parameter of \( M = 7 \) the optimal majority filter performs, on the average, 8.73 stochastic comparisons as opposed to 13 for the majority filter - 33% fewer stochastic comparisons are done.

A plot of the average number of stochastic comparisons versus the filter parameter \( M \) is shown in Fig. 3.4. Equation (3.3.13) was used to calculate the expected number of comparisons for the optimal majority filter. Depending on the value of \( p \) this number may lie anywhere in the shaded region. A \( p \) value of 1.0 results in the least possible number of stochastic comparisons being done, a \( p \) value of 0.5 results in the most.
Figure 3.4: Graph showing the number of stochastic comparisons performed vs. the filter parameter for both the majority filter and the optimal majority filter. The shaded region represents the range of answers for the optimal majority filter.

The fact that this equation is $O(M)$ can be clearly seen from Figure 3.4, where the curve is always bounded from above by $2M-1$ which is linear, and below by $M$. Since the upper and lower bounds are linear, clearly the expression in equation (3.3.16) or (3.3.13) is linear as well. The power of Theorem 3.4 is that we have been able to derive the asymptotic value of the leading coefficient in the linear term.

This completes the analysis for the majority filters, and we will now continue with some other types of deterministic filters based on a completely different perspective.
3.4 Consecutive Filter

Another way in which a deterministic answer could be obtained is to repeat the stochastic comparison until \( M \) consecutive similar results are obtained. Intuitively, given that the same number of total comparisons have been performed, this method would yield the correct deterministic answer with higher probability than the original majority scheme. This is because, in the majority filter, we always perform a maximum number of stochastic comparisons, independent on whether the probability of failure is large or small. In the consecutive filter, on the average a smaller number of stochastic comparisons will be performed for a small failure probability, since \( k \) consecutive correct results will be reached more quickly. The consecutive scheme will thus perform more stochastic comparisons for values of \( p \) closer to \( \frac{1}{2} \) than for values of \( p \) closer to 1. This is similar to the optimal majority scheme, and it is therefore interesting to compare the two filters in terms of the closed form expressions for the probability of obtaining the correct deterministic answer.

An interesting point to note is that the filtered deterministic answer arising from the consecutive filter need not be the same as the majority answer. In fact we may have a high majority of one type of stochastic result such as \( S_p \) and still choose \( D_q \) as the deterministic result because the minority answer of type \( S_q \) may have coincidentally happened to occur consecutively enough number of times. In the long run, of course, we expect the superior answer to appear enough times consecutively and thus the probability that the two deterministic answers will be the same will, typically, be high.

The consecutive scheme is quite simple to implement, and can be represented by the Markov chain shown in Figure 3.5. Initially we start at state \( \Phi_0 \) - when no stochastic comparison has yet been made. This is a non-return state, since after we have obtained at least one stochastic result we will not return to it. State \( P_i \) is reached when exactly \( i \) consecutive correct stochastic answers are obtained. Similarly, the state \( Q_i \) represents exactly \( i \) consecutive incorrect stochastic answers being obtained. The scheme ends when one of the two absorbing states \( P_M \) or \( Q_M \) are reached, with state \( P_M \) denoting the event of reporting \( D_p \) and \( Q_M \) denoting the event of reporting \( D_q \). The deterministic result thus depends on which state the process gets absorbed into.
The algorithm for the consecutive filter is given below.

Algorithm Consecutive:

Aim: An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the consecutive deterministic filter.

Input: M: Consecutive parameter. Answer occurring M consecutive times is returned. Value1, Value2: Values to be compared.

Output: A deterministic answer.

Method
LastAnswer = CurrentAnswer = nil.
NumConsecutive = 0.
while (NumConsecutive < M) do
    CurrentAnswer = (Value1 > Value2). /* stochastic comparison */
    if (CurrentAnswer = LastAnswer) then
        NumConsecutive = NumConsecutive + 1.
    else
        NumConsecutive = 1.
    endif
    LastAnswer = CurrentAnswer.
endwhile
return (CurrentAnswer).
End Algorithm Consecutive.

We now derive a closed form expression for obtaining a correct deterministic answer by using the consecutive scheme. In order to do so we first review the well known result of computing the first passage probabilities [Ross, 1980].

![Markov chain diagram]

Figure 3.5: Markov chain corresponding to the consecutive scheme. Stochastic comparisons are performed until M consecutive similar answers are obtained. The shaded states imply absorption.

Result 3.1: Let F be the transition matrix of a homogeneous Markov chain, where the probability associated with the transition from states i to j is $F_{i,j}$. Let $x_{i,A}$ be the probability
of being absorbed into an absorbing state $A$, where $i \in \tau$, the set of transient states. Then $x_{i,A}$ obeys the recursive equation:

$$x_{i,A} = F_{i,A} + \sum_{j \in \tau} (F_{i,j} \cdot x_{j,A})$$

(3.4.1)

Using the above result we derive the probability for obtaining the correct result, $D_p$, with the consecutive filter.

**Theorem 3.5:** If $p$ is the probability of obtaining a correct stochastic answer, then the consecutive filter with parameter $M$ will yield a correct deterministic answer with probability:

$$\Pr [D_p | \text{consecutive filter being used}] = \frac{(1 - q^M)}{(p q^{M-1}) + (1 - q^{M-1})}.$$  

This probability increases monotonically with $M$ and demonstrates the asymptotic optimality of the scheme.

**Proof:** The proof is done by solving the Markov chain representing the consecutive scheme. Let $a_i$ be the probability of converging to $P_M$ given that the starting state is $P_i$. Similarly let $b_i$ be the probability of converging to $P_M$ given that the starting state is $Q_i$. Also let $f_0$ be the probability of converging to $P_M$ given that the starting state is $\phi_0$. Observe that $f_0$ is the quantity we aim to compute. Using Result 3.1, it can be seen that the quantities $\{a_1\}, \{b_1\}$, and $\phi_0$ themselves obey a set of rather interesting difference equations. Indeed, they satisfy:

$$a_M = 1;$$  

(3.4.2)

$$a_i = p \cdot a_{i+1} + (1 - p) \cdot b_1, \text{ where } 1 \leq i \leq M-1;$$  

(3.4.3)

$$b_M = 0;$$  

(3.4.4)

$$b_i = q \cdot b_{i+1} + (1 - q) \cdot a_1, \text{ where } 1 \leq i \leq M-1;$$  

(3.4.5)

$$f_0 = p \cdot a_1 + q \cdot b_1.$$  

(3.4.6)

Solving the difference equation (3.4.3) using (3.4.2) as a boundary condition yields the value of $a_1$ as:
\[ a_1 = p^{M-1} + (1-p) b_1 \sum_{j=0}^{M-2} p^j. \] (3.4.7)

Similarly, solving the difference equation (3.4.5) using (3.4.4) as a boundary condition yields the value of \( b_1 \):

\[ b_1 = (1-q) a_1 \sum_{j=0}^{M-2} q^j. \] (3.4.8)

Using (3.4.7) and (3.4.8) we now solve for \( a_1 \) and \( b_1 \) to find,

\[ a_1 = \frac{p^{M-1}}{1 - (1-p) p^{M-1}(1-q^{M-1})}. \] (3.4.9)

and

\[ b_1 = \frac{p^{M-1}(1-q^{M-1})}{1 - (1-p) p^{M-1}(1-q^{M-1})}. \] (3.4.10)

We now substitute the above into (3.4.6) to obtain,

\[ f_0 = \frac{p^M + q p^{M-1} - p q M^{M-1} - q^{M-1}}{q^{M-1} + p^{M-1} - p q^{M-1} - q^{M-1}} = \frac{p^{M-1}(1-q^{M})}{q^{M-1} + p^{M-1}(1-q^{M-1})}. \] (3.4.11)

Thus \( \Pr \{D_p \text{ consecutive filter being used}\} = \frac{p^{M-1}(1-q^{M})}{q^{M-1} + p^{M-1}(1-q^{M-1})}. \) (3.4.12)

Now clearly,

\[
\lim_{M \to \infty} \phi_0 = \lim_{M \to \infty} \frac{(1-q^M)}{(q^{M-1}) + (1-q^{M-1})}
= \begin{cases} 
1, & \text{if } p > \frac{1}{2} \\
0, & \text{if } p < \frac{1}{2} \\
\frac{1}{2}, & \text{if } p = \frac{1}{2}
\end{cases}
\] (3.4.13)
and thus the main statement of Theorem 3.5 and the asymptotic optimality is proved.

The monotonicity follows from examining the expressions for \( \Pr \{ D_p \} \) for filter parameters \( M \), and \( M + 1 \), given by:

\[
\frac{1 - q^M}{q^{M-1} + (1 - q^{M-1})}, \quad \text{and} \quad \frac{1 - q^{M+1}}{q^M + (1 - q^M)}
\]

respectively. We use the fact that,

\[
\frac{A}{B} < \frac{A + \Delta}{B + \Delta}, \quad \text{for} \ \Delta > 0, \quad \text{and} \ \frac{A}{B} > 0
\]

(3.4.15)

to transform the first part of equation (3.4.14) as follows:

\[
\frac{1 - q^M}{q^{M-1} + (1 - q^{M-1})} < \frac{1 - q^M + q^M}{q^{M-1} + (1 - q^{M-1}) + q^M} = \frac{1 - q^{M+1}}{q^{M-1} + (1 - q^M)}
\]

(3.4.16)

Furthermore, since \( p > q \), then \( \left( \frac{q}{p} \right)^M < \left( \frac{q}{p} \right)^{M-1} \) and so,

\[
\frac{1 - q^{M+1}}{\left( \frac{q}{p} \right)^{M-1} + (1 - q^M)} < \frac{1 - q^{M+1}}{\left( \frac{q}{p} \right)^M + (1 - q^M)}
\]

(3.4.17)

from which the monotonicity quickly follows. \( \square \)

The expression for \( M_{\min}(\alpha) \) is given by,

\[
M_{\min}(\alpha) = \text{Min} \left\{ M \mid \frac{p^{M-1}(1 - q^M)}{q^{M-1} + p^{M-1}(1 - q^M)} \geq \alpha \right\}.
\]

(3.4.18)

In this case, \( M_{\min}(0.99) = 4 \), when \( p = 0.8 \).
An analysis of the expected number of stochastic comparisons performed would involve the mean time of convergence of a Markov chain. Unfortunately we have not succeeded in obtaining an elegant expression for this quantity, but we believe that it will be the mean of a mixture of two geometric distributions. We now continue with the presentation of another deterministic filter. Unlike the consecutive filter, the next filter investigated chooses an answer which is also the majority answer occurring among all the results of stochastic comparisons done, but this 'majority' is evaluated in a slightly different way.

3.5 Leader Filter

Another new deterministic filter† (which happens to be the most superior one) is one in which the deterministic result corresponds to that stochastic comparison answer which has occurred M more times than the other. In effect the former is said to lead the latter by M. The corresponding Markov chain for such a scheme is shown in Figure 3.6.

State \( \phi_0 \) is the initial state and represents the state when both types of stochastic answers have occurred exactly the same number of times. State \( P_i \) is reached when \( i \) more correct stochastic answers than incorrect ones have been obtained. Similarly state \( Q_i \) represents the case when \( i \) more incorrect stochastic answers than correct answers have been obtained. The scheme ends when one of the two absorbing states \( P_M \) or \( Q_M \) are reached.

![Diagram of Markov chain](image)

**Figure 3.6:** Markov chain corresponding to the leader scheme. Stochastic comparisons are performed until one stochastic answer has occurred \( M \) more times than the other. The shaded states represent absorption.

† The concept of using leaders and consecutivity is entirely novel in the area of both stochastic sorting, game playing, and tournament resolving.
The algorithm for the leader filter is given below.

**Algorithm Leader:**

**Aim:** An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the leader deterministic filter.

**Input:** M: Leading parameter. Answer leading by M is returned.

Value1, Value2: Values to be compared.

**Output:** A deterministic answer.

**Method:**

NumLeading = 0.
while (NumLeading < M) & (NumLeading > -M) do
    if (Value1 ≳ Value2) then
        /* stochastic comparison */
        NumLeading = NumLeading + 1.
    else
        NumLeading = NumLeading - 1.
    endif
endwhile
return (NumLeading > 0)

**End Algorithm Leader.**

We shall now proceed to prove the properties of the leader filter.

**Theorem 3.6:** If $p$ is the probability of obtaining a correct stochastic answer, then the leader filter with parameter $M$ will yield a correct deterministic answer with probability:

$$
Pr[D_p | \text{leader filter being used}] = \frac{1}{\binom{\frac{n}{p}}{M} + 1},
$$

which increases monotonically with $M$. The scheme is also asymptotically optimal.

**Proof:** Consider a formal renumbering of the states such that state $Q_i = \theta_{M-i}$, state $\phi_0 = \theta_{n/2}$, and state $P_i = \theta_{n/2+i}$, where $n = 2M$. Also, let $u_i$ denote the probability of reaching state $\theta_0$ given that the process started at state $\theta_i$. This chain can then be viewed as the Markov chain obtained for the Gambler's Ruin problem [Kartin and Taylor, 1975], whose solution is:
\[ u_i = \begin{cases} \binom{q}{p} \cdot \frac{q^n}{(p^n - 1)}, & \text{if } p \neq q, \text{ and } u_i = \frac{n - i}{n}, & \text{if } p = q. \end{cases} \] (3.5.1)

To prove Theorem 3.6 we take the case when \( i = M, n = 2M \) and find the limit as \( M \to \infty \).

Clearly the probability of obtaining a correct deterministic answer using this scheme is equivalent to \( 1 - u_M \). Thus,

\[ \Pr[D_p \text{ leader filter being used}] = 1 - u_M = \frac{\left(\frac{q}{p}\right)^M}{\left(\frac{q}{p}\right)^{2M} - 1} \]
\[ = \frac{1}{\left(\frac{q}{p}\right)^M + 1}. \] (3.5.2)

And,

\[ \lim_{M \to \infty} (1 - u_M) = \lim_{M \to \infty} \frac{1}{\left(\frac{q}{p}\right)^M + 1} = \begin{cases} 1, & \text{if } p > \frac{1}{2}, \\
0, & \text{if } p < \frac{1}{2}, \\
\frac{1}{2}, & \text{if } p = \frac{1}{2}. \end{cases} \] (3.5.3)

Thus Theorem 3.6 is proved.

The monotonicity is, of course, due to the fact that

\[ \frac{1}{\left(\frac{q}{p}\right)^M + 1} < \frac{1}{\left(\frac{q}{p}\right)^{M+1} + 1} \text{ whenever } p > \frac{1}{2}, \text{ since } \left(\frac{q}{p}\right)^M > \left(\frac{q}{p}\right)^{M+1}. \] (3.5.4)

The minimal \( M \) required for a given accuracy \( \alpha \) is given by,
\[ M_{\text{min}}(\alpha) = \text{Min} \left\{ M \mid \frac{1}{\binom{q}{p}^M + 1} \geq \alpha \right\}. \quad (3.5.5) \]

For a \( p \) value of 0.8, \( M_{\text{min}}(0.99) = 4 \).

An interesting point to note is that the expressions for \( \Pr[D_p] \) for the leader and consecutive filters become 'almost equivalent' for large \( M \). Equation (3.5.2) with a parameter of \( M \) gives an answer which asymptotically approaches the answer obtained from equation (3.4.12) for a parameter of \( M+1 \). The difference is that the two filters will perform a different number of stochastic comparisons in attaining the same \( \Pr[D_p] \).

We now present a new family of deterministic filters based on artificially stochasticizing the transition probabilities between states.

### 3.6 Stochastic Leader Filter

The state transition probabilities for the leader filter presented in the previous section are deterministic; we move forward to \( P_M \) with probability \( p \) and backward (i.e., away from it) with probability \( q \), no matter where in the Markov chain we are. This is except, of course, for the absorbing states. In this section we will consider what happens when we artificially stochasticize these transition probabilities. Thus, if we are in some state we will proceed forwards with a probability which is dependent on that state. Figure 3.9 gives the Markov chain that we are describing, along with the transition probabilities. This entire concept of artificially stochasticizing transitions is novel in the field of stochastic sorting, and tournament resolving.

The motivation behind the idea of stochasticizing the leader filter is based on the fact that we are constantly obtaining stochastic results which may be erroneous and we may have made a wrong decision along the way in the filtering process. We will thus stochasticize the filter in such a way so as to increase the probability of making a forward transition as we are in a more 'advanced' state in the chain. Similarly, we will do the same symmetrically for the other side of the Markov chain as well. Thus, if have received more
responses of one type then we will increase the probability of being absorbed into the corresponding deterministic response.

![Markov chain diagram]

Figure 3.7: Markov chain corresponding to the stochastic leader scheme. Stochastic comparisons are performed until the chain becomes absorbed. The shaded states represent absorption.

As done in the proof of Theorem 3.6 we renumber the states, such that state $Q_i = \theta_{M \cdot i}$, state $\phi_0 = \theta_{n/2}$, and state $P_i = \theta_{n/2+i}$, where $n = 2M$. Thus, if we are in state $\theta_i$ we want to weight the forward transition by $i / 2M$, and the backward transition by $(2M - i) / 2M$. We thus obtain:

$$f_i = \frac{i p}{2M}, \quad \text{and} \quad u_i = \frac{(2M - i) q}{2M},$$

(3.6.1)

where $f_i$ is the probability of moving to state $\theta_{i+1}$ from state $\theta_i$, and $u_i$ is the probability of moving to state $\theta_{i-1}$ from state $\theta_i$. Furthermore we remain in the same state $\theta_i$ with probability $1 - f_i - u_i$. It should be noted that $u_{2M} = 0$ and $f_{2M} = 1$, which makes state $\theta_{2M}$ to be absorbing. Additionally, $u_0 = 1$ and $f_0 = 0$, which renders state $\theta_0$ to be absorbing. We have thus achieved a stochastic version of the previous leader scheme.

Note that the deterministic answer returned no longer corresponds to the stochastic answer which leads by $M$. This is due to the self loops. In fact, we may have obtained a majority of type $S_q$ answers and still end up being absorbed into $D_p$. The formal algorithm follows.

Algorithm Stochastic Leader:

Aim: An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the stochastic leader deterministic filter.
Input: M: Leading parameter. Answer leading by M is returned. Value1, Value2: Values to be compared.

Output: A deterministic answer.

Method:
    NumLeading = 0.
    while (NumLeading < M) & (NumLeading > -M) do
        prob = (NumLeading + M) / 2M.
        stochasticResult = (Value1 > Value2).
        num = RandomFloatIn (0,1).
        if (stochasticResult & (num < prob)) then /* stochastic comparison */
            NumLeading = NumLeading + 1.
        else if (not (stochasticResult) & (num > prob))
            NumLeading = NumLeading - 1.
        endif
    endwhile
    return (NumLeading > 0)

End Algorithm Stochastic Leader.

We can now proceed to analyze the stochastic leader filter and show that it is not as promising as intuition suggests.

Theorem 3.7: If p is the probability of obtaining a correct stochastic answer, then the stochastic leader filter with parameter M will yield a correct deterministic answer with probability†:

\[
Pr \left[ D_p \mid \text{stochastic leader filter being used} \right] = \frac{B_p(M,M)}{B(M,M)}. 
\]

This probability increases monotonically with M. The scheme is also asymptotically optimal.

Proof: It is clear that the underlying Markov chain is a random walk whose transition probabilities are state dependent. An analysis of these chains is given in [Oommen, 1986]. It is shown (in equation (11) of [Oommen, 1986]) that the probability of being absorbed into state \( \theta_{2M} \) can be obtained by:

† The expressions obtained in Theorems 3.7, 3.1, and 3.3 are identical. This is absolutely coincidental and shows that stochastic algorithms can play amazing tricks which defy intuition.
Chapter 3: Stochastic Sorting Using Deterministic Filters

\[ \Pr \{ \text{being absorbed in } \theta_{2M} \} = \frac{\sum_{i=0}^{M-1} R_i}{\sum_{i=0}^{2M-1} R_i}, \text{ where } R_i = \prod_{j=1}^{i} u_j f_{ij}, \text{ and } R_0 = 1. \]  

(3.6.2)

Now, for this chain \( R_i \) can be written as:

\[ R_i = \frac{u_1 u_2 \ldots u_i}{f_1 f_2 \ldots f_i} = \frac{(2M-1)(2M-2)\ldots(M+1)}{1 \cdot 2 \cdot \ldots \cdot i} \binom{q}{i} = \binom{2M-1}{i} \binom{q}{p}. \]  

(3.6.3)

Using this, and equation (3.6.2) we get:

\[ \Pr \left\{ D_p \mid \text{stochastic leader filter being used} \right\} = \frac{\sum_{i=0}^{M-1} \binom{2M-1}{i} \binom{q}{p}}{\sum_{i=0}^{2M-1} \binom{2M-1}{i} \binom{q}{p}}. \]  

(3.6.4)

We now use equation (3.3.9) to obtain,

\[ \Pr \left\{ D_p \mid \text{stochastic leader filter being used} \right\} = \frac{\binom{1}{p}^{2M-1} B_p (M,M)}{B (M,M)} \frac{B_p (M,M)}{(1 + \frac{q}{p})^{2M-1}} \]  

\[ = \frac{B_p (M,M)}{B (M,M)} \frac{B_p (M,M)}{(p + q)^{2M-1}}. \]  

(3.6.5)

Amazingly enough, this is exactly the same expression obtained for both the majority filters discussed already. The monotonicity, and asymptotic expressions follow quickly from the above equation.

This completes the proof of this theorem.
As mentioned earlier, it is interesting to note that the expression for $\Pr[D_p]$ in the stochastic leader filter is the same as that which is obtained for either majority scheme. The difference between these three filters lies in the mean number of stochastic comparisons each will perform to determine the deterministic answer. For example, to obtain $\Pr[D_p] = 0.967$, when $p = 0.8$, the majority filter performs exactly 7 stochastic comparisons, while the optimal majority and stochastic leader filters perform, on the average, 4.927 and 9.322 stochastic comparisons respectively. It becomes quickly clear, even though it looks promising, that the stochastic leader filter provides no improvement to its deterministic counterpart.

During the course of this research endeavour we have analyzed various (at least half a dozen) other stochasticized versions of the leader filters including those involving self-loops like the scheme just described, and those without any self-loops. Unfortunately, we have not been able to discover (invent) a stochastic filter which is superior to its deterministic counterpart. In fact, contrary to the results obtained in learning theory, it appears that adding stochasticity will always degrade the performance of the leader filter rather than improve it. We believe that this is true as a consequence of the symmetry of the Markov chain - whatever is done on one side must be symmetrically done for the other as well. Thus, if we improve the probability of converging to one side, we could also improve the probability of converging to the other side and these effects could simultaneously degrade the performance of the overall filter.

### 3.7 Stochasticizing The Other Filters

As was done with the leader filter earlier, we also attempted to improve the performance of the consecutive, majority, and optimal majority filters through the introduction of stochasticity. In each case the result was the same - the slightest inclusion of any artificially introduced stochastic behaviour caused a degradation in performance. It appears as if the deterministic versions of the filters are optimal, and that improvements cannot be achieved through introducing stochasticity.

We shall conclude this chapter with a comparison of the filters presented. For each filter, extensive simulation results will be presented to evaluate their performance under
different conditions. Furthermore, results of actual deterministic sorting algorithms using these filters will also be presented.

3.8 A Comparison Of Deterministic Filters

In this section we present a complete comparison of all the different deterministic filters introduced in this chapter. In each case a set of simulations was carried out to see if the experimental values agreed with the theoretically predicted ones. The simulations were carried out on a Sparc/Sun station and were written in C. A total of 1000 independent simulations were carried out for each filter. In each simulation 10,000 filterings were executed to obtain a value for Pr \( [D_p] \). An ensemble average was then evaluated for Pr \( [D_p] \) from the simulations. To render the computations comparable, for each set of simulations done on the deterministic filters the random number generator was seeded with exactly the same value. In the reported results the error in standard deviations is given, as well as the statistical agreement between the predicted and experimental values. In addition to this, the mean number of stochastic comparisons performed in all modes of filtering is also reported. These results experimentally support the theoretical derivations. Also, the results allow us to compare the speed of convergence of the different filters by examining the mean number of stochastic comparisons needed to reach some acceptable Pr \( [D_p] \).

For any experimentally observable set of values \( \{x_i\} \) the standard deviation may be calculated by using the well known statistical formula [Barford, 1987]:

\[
\sigma_x = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N-1}}, \quad \text{where} \quad \mu = \frac{1}{N} \sum_{i=1}^{N} x_i \quad \text{is the mean},
\]

\( (3.8.1) \)

\( N \) is the total number of results and \( x_i \) are the individual results of each simulation. The standard deviation of the mean itself is given by:

\[
\sigma_\mu = \frac{\sigma_x}{\sqrt{N}}.
\]

\( (3.8.2) \)
The agreement value between the experimental result and the true, or theoretical result, may calculated by:

\[
\text{agreement} = \frac{\text{theoretical Pr } [D_p] - \text{experimental Pr } [D_p]}{\sigma}.
\]  

(3.8.3)

The experimental results are said to agree with the theoretically predicted ones if the experimental value lies within two standard deviations of the theoretical value, implying that the agreement must thus be less than or equal to 2. This is, in fact, the case for all results obtained, the agreement is of the order of 0.1 to 0.01 which is quite excellent. These experimental results are tabulated in Tables 3.2 to 3.6.

Table 3.2: Simulation results for the majority filter for a failure probability of \( q = 0.2 \).

<table>
<thead>
<tr>
<th>M</th>
<th>Majority</th>
<th>Simulation Results</th>
<th>Agreement</th>
<th># Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.80000000</td>
<td>0.79991090 ± 0.00413917</td>
<td>0.02152605</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.89600000</td>
<td>0.89597250 ± 0.00306849</td>
<td>0.00896206</td>
<td>3.00</td>
</tr>
<tr>
<td>3</td>
<td>0.94208000</td>
<td>0.94214140 ± 0.00245430</td>
<td>0.02501732</td>
<td>5.00</td>
</tr>
<tr>
<td>4</td>
<td>0.96665600</td>
<td>0.96677770 ± 0.00172338</td>
<td>0.07061704</td>
<td>7.00</td>
</tr>
<tr>
<td>5</td>
<td>0.98041856</td>
<td>0.98049480 ± 0.00139859</td>
<td>0.05451204</td>
<td>9.00</td>
</tr>
</tbody>
</table>
Table 3.3: Simulation results for the optimal majority filter for a failure probability of \( q = 0.2 \).

<table>
<thead>
<tr>
<th>M</th>
<th>Optimal Majority</th>
<th>Simulation Results</th>
<th>Agreement</th>
<th># Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.80000000</td>
<td>0.79991050 ± 0.00413917</td>
<td>0.02152605</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.89600000</td>
<td>0.89596960 ± 0.00310174</td>
<td>0.00980095</td>
<td>2.32</td>
</tr>
<tr>
<td>3</td>
<td>0.94208000</td>
<td>0.94213000 ± 0.00239143</td>
<td>0.02090799</td>
<td>3.63</td>
</tr>
<tr>
<td>4</td>
<td>0.96665600</td>
<td>0.96667520 ± 0.00175993</td>
<td>0.01090952</td>
<td>4.93</td>
</tr>
<tr>
<td>5</td>
<td>0.98041856</td>
<td>0.99999999 ± 0.00000000</td>
<td>0.07360169</td>
<td>6.20</td>
</tr>
</tbody>
</table>

Table 3.4: Simulation results for the consecutive filter for a failure probability of \( q = 0.2 \).

<table>
<thead>
<tr>
<th>M</th>
<th>Consecutive</th>
<th>Simulation Results</th>
<th>Agreement</th>
<th># Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.80000000</td>
<td>0.79991090 ± 0.00413917</td>
<td>0.02152605</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.91428572</td>
<td>0.91427870 ± 0.00286543</td>
<td>0.00244989</td>
<td>2.57</td>
</tr>
<tr>
<td>3</td>
<td>0.97017115</td>
<td>0.97017170 ± 0.00170431</td>
<td>0.00032271</td>
<td>4.62</td>
</tr>
<tr>
<td>4</td>
<td>0.99084481</td>
<td>0.99087390 ± 0.00090510</td>
<td>0.03214009</td>
<td>7.14</td>
</tr>
<tr>
<td>5</td>
<td>0.99737979</td>
<td>0.99739770 ± 0.00050770</td>
<td>0.03527674</td>
<td>10.23</td>
</tr>
</tbody>
</table>
Table 3.5: Simulation results for the leader filter for a failure probability of $q = 0.2$.

<table>
<thead>
<tr>
<th>M</th>
<th>Leader</th>
<th>Simulation Results</th>
<th>Agreement</th>
<th># Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.80000000</td>
<td>0.79991090 ± 0.00413917</td>
<td>0.02152605</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.94117647</td>
<td>0.94118520 ± 0.00229069</td>
<td>0.00381108</td>
<td>2.94</td>
</tr>
<tr>
<td>3</td>
<td>0.98461538</td>
<td>0.98470060 ± 0.00123948</td>
<td>0.06875464</td>
<td>4.85</td>
</tr>
<tr>
<td>4</td>
<td>0.99610895</td>
<td>0.99612310 ± 0.00059730</td>
<td>0.02368994</td>
<td>6.61</td>
</tr>
<tr>
<td>5</td>
<td>0.99902439</td>
<td>0.99903090 ± 0.00030979</td>
<td>0.02101424</td>
<td>8.32</td>
</tr>
</tbody>
</table>

Table 3.6: Simulation results for the stochastic leader filter for a failure probability of $q = 0.2$.

<table>
<thead>
<tr>
<th>M</th>
<th>Stochastic Leader</th>
<th>Simulation Results</th>
<th>Agreement</th>
<th># Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.80000000</td>
<td>0.79991090 ± 0.00413917</td>
<td>0.02152605</td>
<td>2.00</td>
</tr>
<tr>
<td>2</td>
<td>0.89600000</td>
<td>0.89618700 ± 0.00962937</td>
<td>0.01941975</td>
<td>4.62</td>
</tr>
<tr>
<td>3</td>
<td>0.94208000</td>
<td>0.94210100 ± 0.00734280</td>
<td>0.00285994</td>
<td>7.07</td>
</tr>
<tr>
<td>4</td>
<td>0.96665600</td>
<td>0.96614300 ± 0.00572340</td>
<td>0.08963204</td>
<td>9.32</td>
</tr>
<tr>
<td>5</td>
<td>0.98041856</td>
<td>0.98006900 ± 0.00441766</td>
<td>0.07912786</td>
<td>11.46</td>
</tr>
</tbody>
</table>
Figure 3.8: Graph of $\text{Pr}[D_p]$ vs. number of stochastic comparisons for the majority, optimal majority, consecutive, and leader filters. The failure probability in all cases was $q = 0.2$.

Figure 3.9: Graph of $\text{Pr}[D_p]$ vs. number of stochastic comparisons for the majority, optimal majority, consecutive, and leader filters. The failure probability in all cases was $q = 0.4$. 
Figure 3.10: Graph of $\Pr [D_p]$ vs. number of stochastic comparisons for leader, stochastic leader, optimal majority, and majority filters. The failure probability in all cases was $q = 0.2$.

Figures 3.8 and 3.9 clearly indicate that under all conditions the leader filter is the most efficient filter. The consecutive filter is also quite effective but only for cases when the failure probability is not too large. For cases when the failure probability is larger, the consecutive filter cannot be used to yield any reasonably good result. The optimal majority filter seems to take the second place in terms of its performance.

In general, the stochastic leader filter did not perform well at all - in fact it had the poorest performance of all the examined filters (see Figure 3.10). Also, the other stochasticized versions of the filters performed much worse than their deterministic counterparts.

In the next section we shall present simulation results of actual stochastic sorting algorithms which utilize these filters.
3.9 A Comparison Of Stochastic Sorting Algorithms Using Filters

The next set of simulations involved the use of actual deterministic sorting algorithms which work in conjunction with the filters. The disorder metric described in Section 2.5 was used, to compute the ‘sortedness’ of a list.

![Graph showing disorder vs. number of stochastic comparisons for different filters.]

Figure 3.11: Graph of average disorder vs. number of stochastic comparisons for majority, optimal majority, consecutive, and leader filters. In each case Insertion Sort was used to order a set of 32 values. The failure probability was set at \( q = 0.2 \), and the filter parameters were assigned so as to achieve a final accuracy of 0.99 for each comparison.

In all the simulations dealing with actual deterministic sorting algorithms, the initial list was reversely sorted, implying that the initial disorder was unity in all cases. A total of 100 independent experiments were performed, and the disorder and standard error were calculated at various points in time. The standard error or deviation was calculated according to equation (3.8.1). Figure 3.11 shows the results for using Insertion Sort, using the four different filters and in each case the filter achieved the minimum success probability of 0.99. It can be seen that Insertion Sort using the Leader filter achieves the
lowest disorder in the shortest time. Indeed, after just 700 stochastic comparisons the list is essentially in the correct order. The failure probability for this simulation was 0.2, implying that, on the average, about 20% of the comparisons returned erroneous information. Notice that if the individual comparisons are error-free Insertion Sort would require only 160 comparisons to sort the 32 values†.

![Graph of average disorder vs. number of stochastic comparisons for majority, optimal majority, consecutive, and leader filters. In each case Shell's Sort was used to order a set of 32 values. The failure probability was set at $q = 0.2$, and the filter parameters were assigned so as to achieve a final accuracy of 0.99 for each comparison. The inset graph is a magnification of the region near low disorder values.](image)

**Figure 3.12:** Graph of average disorder vs. number of stochastic comparisons for majority, optimal majority, consecutive, and leader filters. In each case Shell's Sort was used to order a set of 32 values. The failure probability was set at $q = 0.2$, and the filter parameters were assigned so as to achieve a final accuracy of 0.99 for each comparison. The inset graph is a magnification of the region near low disorder values.

Since Insertion Sort does not sort *in situ* the disorder is calculated slightly differently. For Insertion Sort, at any point in the process, we essentially have two lists,

† Insertion Sort runs in $O(N \log N)$ time when we just consider the number of comparisons performed. The overall time including all data restructuring and inserting takes $O(N^2)$ time.
the original from which each value is taken, and the new list into which the values are being inserted. The disorder was calculated as the weighted average (according to the individual list sizes) of the disorder of the new growing list, and the old shrinking list.

Figure 3.12 depicts the results for the same scenario but using Shell's Sort. It can be seen that Shell's sort behaves quite differently from Insertion Sort. The disorder decreases much faster at the start. As before the Leader filter is the best filter for the job.

![Graph of average disorder vs. number of stochastic comparisons for Insertion Sort using a Leader filter with different filter parameters. The failure probability was set at q = 0.25. In each case a set of 32 values was sorted. The corresponding filtered success probability, α, is given for each filter parameter.](image)

The next set of simulations investigated the behaviour of the stochastic sorting algorithms for differing filter parameters. Figure 3.13 shows the results for Insertion Sort using the Leader filter for filters parameters of 1 through 5. The filtered success probability is shown on the graph. The individual failure probability for this set of experiments was set at 0.25. Figure 3.14 depicts the results for the same scenario using Heap Sort.
It becomes quickly evident that without the use of any filtering (filter parameter $M = 1$) the sorting algorithms will not achieve a sorted order. At best Insertion Sort achieves a disorder of 0.16, and Heap Sort fares worse with a disorder of 0.31. Only through filtering are we able to reach disorders which approach zero. It also becomes clear that Heap Sort is generally worse than Insertion Sort in terms of the number of stochastic comparisons performed to achieve a specific value for the measure of disorder.

![Graph of average disorder vs. number of stochastic comparisons for Heap Sort using a leader filter with different filter parameters. The failure probability was set at $q = 0.25$. In each case a set of 32 values was sorted. The corresponding filtered success probability, $\alpha$, is given for each filter parameter.](image)

Figure 3.14: Graph of average disorder vs. number of stochastic comparisons for Heap Sort using a leader filter with different filter parameters. The failure probability was set at $q = 0.25$. In each case a set of 32 values was sorted. The corresponding filtered success probability, $\alpha$, is given for each filter parameter.

Figure 3.15 shows the performance of Insertion Sort, Shell's Sort, and Heap Sort, on the same set of 32 values, with a failure probability of 0.25. The Leader filter with parameter $M = 3$ was used in all three deterministic sorts. From the graph shown in Figure 3.15 it is clear that Shell's Sort, and Insertion Sort are more efficient than Heap Sort.
Figure 3.15: Graph of average disorder vs. number of stochastic comparisons for Insertion Sort, Shell's Sort, and Heap Sort. The failure probability was set at $q = 0.25$, a leader filter was used with parameter $M = 3$. In each case a set of 32 values was sorted.

3.10 Conclusions

In this chapter we have examined the use of absorbing filters to reduce the failure probability. It was shown that such filters can effectively eliminate all errors provided that an error occurs with a lower probability than a correct answer. The most efficient filter was the leader filter, which very quickly converged to the correct deterministic result with high probability. The consecutive filter was also efficient but only for values of $p$ roughly between 0.75 and 1; for values of $p$ less than this, it performed very poorly. Attempting to attain an accuracy greater than 0.99, generally required far too many stochastic comparisons. Thus, typically, we recommend the optimal majority filter as the second best one.
It was also discovered that artificially stochasticizing a deterministic filter did not improve its performance. On the contrary, it tended to degrade it. This is quite an interesting observation since learning automata whose underlying Markov chains are stochastic are far superior to the corresponding ones whose Markov chains are deterministic. The reason for the degradation in performance of the stochasticized filters is the symmetry of the transition probabilities in the underlying Markov chain. If we attempt to stochasticize in a way so as to ‘push’ the filter into converging more quickly to the right answer, we unfortunately also increase the probability of it converging to the incorrect answer. We thus conjecture that the addition of stochasticity to deterministic filters degrades their performance. Table 3.7 sums up the major results of this chapter with respect to the convergence probabilities.

Table 3.7: The convergence probabilities of the various filters introduced in Chapter 3.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Probability of obtaining correct deterministic result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Majority</td>
<td>$\frac{B_p(M,M)}{B(M,M)}$</td>
</tr>
<tr>
<td>Optimal Majority</td>
<td>$\frac{B_p(M,M)}{B(M,M)}$</td>
</tr>
<tr>
<td>Consecutive</td>
<td>$\frac{(1 - q^M_p)}{\left(\frac{q}{p}\right)^{M-1} + (1 - q^M_p)}$</td>
</tr>
<tr>
<td>Leader</td>
<td>$\frac{1}{1 + \left(\frac{q}{p}\right)^M}$</td>
</tr>
<tr>
<td>Stochastic Leader</td>
<td>$\frac{B_p(M,M)}{B(M,M)}$</td>
</tr>
</tbody>
</table>

Apart from theoretically analyzing the filters we have also examined the performance of three different deterministic sorting algorithms - Insertion Sort, Shell's
Sort, and Heap Sort acting in tandem with them. It was found that Insertion Sort and Shell's Sort outperformed Heap Sort in all cases. Furthermore, Shell's Sort is a more 'aggressive' sort, in that it lowers the disorder much more quickly at the beginning. As opposed to this, Insertion Sort decreases the list disorder in a more even fashion. Both Insertion Sort and Shell's Sort achieve similar results for an equivalent number of stochastic comparisons.

In the following chapter we present a scheme by which we can concatenate deterministic filters. Indeed, we suggest the use of a hierarchy of filters, each producing intermediate results with successively lower failure probabilities. When arranged in a tree-like manner we will show that such a structuring is advantageous for certain filters.
CHAPTER 4

Using Multiple Levels Of Filters

4.1 Introduction

In this chapter we examine the use of many deterministic filters arranged in a hierarchical structure to achieve the filtering process. In the previous chapter all stochastic comparison results were fed into a single filter which yielded as its output the corresponding deterministic answer. This process is depicted in Figure 4.1 below.

![Diagram of filtering process using a single filter](image)

Figure 4.1: The filtering process using a single filter.

We now consider a method of filtering the stochastic comparisons in a manner which utilizes many filters of the same kind arranged in a tree-like structure as shown in Figure 4.2. In general we need not use the same type of filter at every stage. However, in this study, for the sake of regularity, we will generally use a single type of filter throughout
the structure\(^\dagger\). Also, we assume that at the same level the filters are all of the same type.

Figure 4.2: The filtering process using a hierarchical structure of filters. The structure depicted contains two levels of filters. Although three filters appear in the first level they are in fact one and the same filter being used three times over. No purpose is served by using three physically distinct filters if each level contains the same type of filter.

The motivation behind using such a filtering structure is to lower the number of stochastic comparisons done to achieve the same accuracy in the final deterministic result. The number of levels of filtering need not be confined to just two, as depicted in Figure 4.2, but can be any number in general. A simple example will clarify this.

Suppose we are using a two layer filter structure as shown in Figure 4.2, where each filter is a Majority filter with parameter \(M = 2\). Also suppose that the incoming nine stochastic comparison results are as follows: \(\{s_p, s_p, s_q, s_q, s_p, s_q, s_p, s_p, s_p\}\). The majority filter at the first level will produce the intermediate results \(i_p, i_q, i_p\) - corresponding to majorities chosen from \(p, p, q\), \(q, p, q\), and \(p, p, p\) respectively. The filter at the final level will take the majority again to produce \(d_p\).

It will be shown, that using many levels of filtering improves the performance of consecutive filters, but not of the majority or the leader filters. The reason for this has to do with the function that describes the mean number of stochastic comparisons performed to achieve a certain accuracy, and will be discussed in detail in the later sections.

\(^\dagger\) We choose to make our structures homogeneous in order to discover the type of filter for which improvement in performance may be expected. We already know that the leader filter is superior to all others - thus any filter structure composed of any variety of filters will at best perform as well as the same structure composed solely of leader filters.
4.2 Analysis Of The Hierarchical Filtering Structure.

In this section we analyze the probability $\Pr[D_p]$ for the final deterministic answer of a hierarchically structured filtering system. Let $H$ be the total number of levels in the filtering structure, also called the height. Let $\xi_i$ denote the intermediate failure probability at each level in the structure, where $0 \leq i \leq H$. By definition $\xi_0 = q$, the failure probability of an individual stochastic comparison. Let $\psi_i$ denote the intermediate success probability, i.e $\psi_i = 1 - \xi_i$. Also by definition, we have $\psi_0 = p$. Finally, let $\Psi(...)$ be the function which gives the success probability of the filtered result. $\Psi(...)$ is a function of the success probability of the incoming result, as well as the filter parameter. Similarly $\Xi(...)$ denotes the failure probability function. For clarification please refer to Figure 4.3.

In the single layer filtering process depicted in Figure 4.1, we would have $H = 1$, and $\psi_1 = \Psi(\psi_0, M_0) = \Psi(p, M) = \Pr[D_p]$, which can be obtained directly from Table 3.7. Similarly $\xi_1 = \Xi(\xi_0, M_0) = \Xi(q, M) = \Pr[D_q]$.

![Diagram showing some intermediate level i of the filtering structure, with incoming results, and an outgoing filtered result. The corresponding success probabilities are given for both sides.](image)

Figure 4.3: Diagram showing some intermediate level $i$ of the filtering structure, with incoming results, and an outgoing filtered result. The corresponding success probabilities are given for both sides.

In a multi-layer filtering structure final success probability would be evaluated by:

$$
\psi_H = \Psi(\psi_{H-1}, M_{H-1}) = \Psi(\psi_{H-2}, M_{H-2}, M_{H-1}) \ldots \tag{4.2.1}
$$

If we assume the same filter parameter for each level of filtering equation (4.2.1) simplifies to:

$$
\psi_H = \psi^H(\psi_0, M), \tag{4.2.2}
$$

where $\psi^H$ is the composite function $\Psi(\Psi(...))$ of degree $H$. 
Chapter 4: Using Multiple Levels Of Filters

In addition, to this, we are also interested in the total number of stochastic comparisons made by the whole filtering structure. For any deterministic filter we denote the number of stochastic comparisons performed by the function $\mathbf{K}(\psi, M_i)$ which is a function of the filter parameter, and the success probability. Thus the filters at each level will need an average of $\kappa_i = \mathbf{K}(\psi_i, M_i)$ incoming results to produce the outgoing result. Suppose that we have a two layer filtering structure as shown in Figure 4.2. Then the final layer needs $r_1$ inputs but to obtain a single input, the filters at the first level must perform, on the average, $\kappa_0$ stochastic comparisons. The total number of stochastic comparisons is thus $\kappa_0 \kappa_1$. In general for an $H$ layer filtering structure the mean number of stochastic comparisons required is:

\[
\text{Total # stochastic comparisons} = \prod_{i=0}^{H} \kappa_i.
\]

(4.2.3)

With this we are ready to study filtering structures for each type of deterministic filter. We will also provide simulation results at the end of each section, and a general conclusion at the end of the chapter.

4.3 A Hierarchy Of Majority Filters

In this section we will investigate the feasibility of creating hierarchical filtering structures composed of majority filters. For the majority filter we have

$$\Psi(p,M) = \psi(M,M) / \psi(M,M),$$

and

$$\mathbf{K}(p,M) = 2M - 1.$$

Using this we can directly calculate the final success probability, for any type of filter structure. We will now present a few such structures and compare them to the original single filter scheme.

---

\* The equality $E[XY] = E[X]E[Y]$ is not always true. It will be true if $X$ and $Y$ are nonrelated. In this case we know that the input is a set of independent occurrences of $S_p$ and $S_q$, and hence (4.2.3) is true.
Suppose, we use a two layer filtering structure, with filter parameters \( M=5 \), and \( M=5 \) for both layers, suppose also that the initial success probability is \( p = 0.51 \). We will let the notation \([M=5], [M=5]\) denote such a two layer filtering structure. It is easily found from equation (4.2.1), that the final success probability will be 0.560335. The total number of stochastic comparisons performed is obtained from equation (4.2.3) as 81. Table 4.1 lists the various results for different filter structures.

<table>
<thead>
<tr>
<th>Filtering Structure Majority</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>([M=41])</td>
<td>0.571649</td>
<td>81</td>
</tr>
<tr>
<td>([M=5], [M=5])</td>
<td>0.560335</td>
<td>81</td>
</tr>
<tr>
<td>([M=2], [M=14])</td>
<td>0.562515</td>
<td>81</td>
</tr>
<tr>
<td>([M=14], [M=2])</td>
<td>0.562513</td>
<td>81</td>
</tr>
<tr>
<td>([M=2], [M=2], [M=5])</td>
<td>0.555198</td>
<td>81</td>
</tr>
<tr>
<td>([M=2], [M=5], [M=2])</td>
<td>0.555197</td>
<td>81</td>
</tr>
<tr>
<td>([M=5], [M=2], [M=2])</td>
<td>0.555196</td>
<td>81</td>
</tr>
<tr>
<td>([M=2], [M=2], [M=2], [M=2])</td>
<td>0.550492</td>
<td>81</td>
</tr>
</tbody>
</table>

From the results given in Table 4.1, it becomes apparent that the addition of a second filtering layer degrades the performance. In fact, splitting a single layer into two or more layers degrades performance. Best results are achieved when every stochastic comparison is fed into a single filter. Thus hierarchical filtering structures using majority filters offer no advantage. We will now analyze if the same is true for optimal majority filters.
4.4 A Hierarchy Of Optimal Majority Filters

In this section we will examine if a hierarchy of optimal majority filters is an improvement over using just a single optimal majority filter. Table 4.2 lists the corresponding results for the same filter structures that were examined in Table 4.1 in the previous section. All values were calculated according to equations derived in Sections 3.3 and 4.2.

Table 4.2: Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the optimal majority filter. The initial failure probability was $q = 0.49$.

<table>
<thead>
<tr>
<th>Filtering Structure Optimal Majority</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>[M=41]</td>
<td>0.571649</td>
<td>74.708910</td>
</tr>
<tr>
<td>[M=33]</td>
<td>0.564301</td>
<td>59.481197</td>
</tr>
<tr>
<td>[M=32]</td>
<td>0.563320</td>
<td>57.583626</td>
</tr>
<tr>
<td>[M=31]</td>
<td>0.562323</td>
<td>55.687572</td>
</tr>
<tr>
<td>[M=27]</td>
<td>0.558160</td>
<td>48.120178</td>
</tr>
<tr>
<td>[M=26]</td>
<td>0.557070</td>
<td>46.233009</td>
</tr>
<tr>
<td>[M=25]</td>
<td>0.555958</td>
<td>44.347934</td>
</tr>
<tr>
<td>[M=23]</td>
<td>0.553664</td>
<td>40.584603</td>
</tr>
<tr>
<td>[M=22]</td>
<td>0.552479</td>
<td>38.706653</td>
</tr>
</tbody>
</table>

| [M=5], [M=5]                        | 0.560335                    | 56.736327                        |
| [M=2], [M=14]                      | 0.562515                    | 59.456006                        |
| [M=14], [M=2]                      | 0.562513                    | 59.420987                        |
| [M=2], [M=2], [M=5]               | 0.555198                    | 47.046734                        |
| [M=2], [M=5], [M=2]               | 0.555197                    | 47.037396                        |
| [M=5], [M=2], [M=2]               | 0.555196                    | 47.033241                        |
| [M=2], [M=2], [M=2], [M=2]        | 0.550492                    | 39.001051                        |
As before, we see that, in the case of the optimal majority filter, the use of a complex filter structure is not advantageous. For example the single level filter structure \([M=31]\) outperforms the double layer structure \([M=5], [M=5]\). It achieves a higher success probability by performing less stochastic comparisons. These data points are plotted in Figure 4.4. All data points for higher order filter structures appear below the single layer curve implying that their performance is worse.

![Graph showing performance of different optimal majority filter structures. The curve is the result for a single filter level. Other filter structures are plotted as scattered points. The data for this graph is shown in Table 4.2.](image)

**Figure 4.4**: Graph showing performance of different optimal majority filter structures. The curve is the result for a single filter level. Other filter structures are plotted as scattered points. The data for this graph is shown in Table 4.2.

### 4.5 A Hierarchy Of Consecutive Filters

Results of filtered success probabilities for hierarchies of consecutive filters show great improvement. Table 4.3 lists the results. To obtain a filtered accuracy of about 0.6 when the failure probability is \(q = 0.49\), we need about 4041 stochastic comparisons when using a single consecutive filter. Using a two layer filter structure such as \([M=4],[M=4]\) we can drastically reduce this number to about 223 stochastic comparisons. Furthermore, using the three layer filter structure, \([M=2],[M=3],[M=3]\) the number of stochastic comparisons reduces still more to about 145.
Chapter 4: Using Multiple Levels Of Filters

The errors shown in Table 4.3 are the standard deviations of the mean, as calculated by equations (3.8.1) and (3.8.2). Equation (4.2.3) was used to calculate the mean number of stochastic comparisons for multiple layer structures. Also to calculate the corresponding error statistical error propagation methods were used [Barford, 1987].

Table 4.3: Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the consecutive filter. The initial failure probability was $q = 0.49$.

<table>
<thead>
<tr>
<th>Filtering Structure Consecutive</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>[M=11], [M=12]</td>
<td>$0.598752$</td>
<td>$2022 \pm 12$</td>
</tr>
<tr>
<td></td>
<td>$0.608301$</td>
<td>$4041 \pm 28$</td>
</tr>
<tr>
<td>[M=4], [M=4]</td>
<td>$0.605176$</td>
<td>$222.79 \pm 0.79$</td>
</tr>
<tr>
<td>[M=3], [M=5]</td>
<td>$0.599774$</td>
<td>$214.04 \pm 0.74$</td>
</tr>
<tr>
<td>[M=5], [M=3]</td>
<td>$0.599780$</td>
<td>$215.34 \pm 0.75$</td>
</tr>
<tr>
<td>[M=2], [M=7]</td>
<td>$0.599590$</td>
<td>$375.24 \pm 1.27$</td>
</tr>
<tr>
<td>[M=7], [M=2]</td>
<td>$0.599669$</td>
<td>$375.24 \pm 1.26$</td>
</tr>
<tr>
<td>[M=2], [M=3], [M=3]</td>
<td>$0.597130$</td>
<td>$145.14 \pm 0.48$</td>
</tr>
</tbody>
</table>

From our results it becomes apparent that the optimal type of hierarchical structure will include as many levels as possible and will be of the form [M=2],[M=2],...,[M=2]. Tables 4.4 through 4.6 tabulate results for just such filter structures and compare them to single filter results for the majority, optimal majority, and leader filters. From Table 4.4 one can see that the consecutive [M=2],[M=2] filter structure yields a better filtered success probability - using less comparisons than the majority and optimal majority filters, but not the leader filter. For example, the consecutive filter structure [M=2],[M=2] achieves a filtered success probability of 0.527755 and performs about 9.020 stochastic comparisons. The optimal majority filter achieves a lower filtered success probability of 0.527052 performing 9.290 stochastic comparisons in the process. Similar conclusions can be drawn from Tables 4.5 and 4.6.
Table 4.4: Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using different filters. In places where no error is given values are exact and were determined theoretically.

<table>
<thead>
<tr>
<th>Filter Structure</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consecutive [M=2],[M=2]</td>
<td>0.527755</td>
<td>9.020 ± 0.019</td>
</tr>
<tr>
<td>Majority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[M=5]</td>
<td>0.524560</td>
<td>9.000</td>
</tr>
<tr>
<td>[M=6]</td>
<td>0.527052</td>
<td>11.000</td>
</tr>
<tr>
<td>[M=7]</td>
<td>0.529303</td>
<td>13.000</td>
</tr>
<tr>
<td>Optimal Majority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[M=5]</td>
<td>0.524560</td>
<td>7.537</td>
</tr>
<tr>
<td>[M=6]</td>
<td>0.527052</td>
<td>9.290</td>
</tr>
<tr>
<td>[M=7]</td>
<td>0.529303</td>
<td>11.064</td>
</tr>
<tr>
<td>Leader</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[M=1]</td>
<td>0.510000</td>
<td>1.000</td>
</tr>
<tr>
<td>[M=2]</td>
<td>0.519990</td>
<td>4.014 ± 0.009</td>
</tr>
<tr>
<td>[M=3]</td>
<td>0.529968</td>
<td>8.973 ± 0.022</td>
</tr>
</tbody>
</table>

Table 4.5: Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using different filters. In places where no error is given values are exact and were determined theoretically.

<table>
<thead>
<tr>
<th>Filter Structure</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consecutive [M=2],[M=2],[M=2]</td>
<td>0.546183</td>
<td>27.059 ± 0.070</td>
</tr>
<tr>
<td>Majority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[M=16]</td>
<td>0.544695</td>
<td>31.000</td>
</tr>
<tr>
<td>[M=17]</td>
<td>0.546085</td>
<td>33.000</td>
</tr>
<tr>
<td>[M=18]</td>
<td>0.547434</td>
<td>35.000</td>
</tr>
<tr>
<td>Optimal Majority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[M=16]</td>
<td>0.544695</td>
<td>27.504</td>
</tr>
<tr>
<td>[M=17]</td>
<td>0.546085</td>
<td>29.362</td>
</tr>
<tr>
<td>[M=18]</td>
<td>0.547434</td>
<td>31.224</td>
</tr>
<tr>
<td>Leader</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[M=4]</td>
<td>0.539920</td>
<td>15.932 ± 0.040</td>
</tr>
<tr>
<td>[M=5]</td>
<td>0.549841</td>
<td>24.973 ± 0.063</td>
</tr>
<tr>
<td>[M=6]</td>
<td>0.559722</td>
<td>35.978 ± 0.092</td>
</tr>
</tbody>
</table>
Table 4.6: Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using different filters. In places where no error is given values are exact and were determined theoretically.

<table>
<thead>
<tr>
<th>Filter Structure</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consecutive, M=2, M=2, M=2</td>
<td>0.576623</td>
<td>80.932 ± 0.242</td>
</tr>
<tr>
<td>Majority, M=46</td>
<td>0.575867</td>
<td>91.000</td>
</tr>
<tr>
<td>Majority, M=47</td>
<td>0.576682</td>
<td>93.000</td>
</tr>
<tr>
<td>Majority, M=48</td>
<td>0.577487</td>
<td>95.000</td>
</tr>
<tr>
<td>Optimal Majority, M=46</td>
<td>0.575867</td>
<td>84.261</td>
</tr>
<tr>
<td>Optimal Majority, M=47</td>
<td>0.576682</td>
<td>86.175</td>
</tr>
<tr>
<td>Optimal Majority, M=48</td>
<td>0.577487</td>
<td>88.089</td>
</tr>
<tr>
<td>Leader, M=7</td>
<td>0.569555</td>
<td>48.683 ± 0.124</td>
</tr>
<tr>
<td>Leader, M=8</td>
<td>0.579335</td>
<td>63.304 ± 0.162</td>
</tr>
<tr>
<td>Leader, M=9</td>
<td>0.589052</td>
<td>80.130 ± 0.204</td>
</tr>
</tbody>
</table>

Figure 4.5: Graph showing performance of hierarchical consecutive filter structures and single majority, optimal majority, and leader filters. The failure probability was q = 0.49. The graph shows that consecutive filters arranged in a multi level structure outperforms majority and optimal majority filters, but not the leader filter. The data for this graph is given in Tables 4.4 to 4.6.
4.6 A Hierarchy Of Leader Filters

To conclude our study of hierarchical filter structures we shall now present results for the leader filter. Table 4.7 lists the various filter structures along with the resultant filtered success probabilities and mean number of stochastic comparisons performed in the filtering process. Observe that a hierarchical structure of leader filters offers no advantage, it also seems to offer no discernible disadvantage as far as we can tell from our results. A conclusive statement regarding this cannot be made but we believe that a hierarchical filtering structure composed of leader filters will on par with a single layer filter at best.

To conclusively determine whether a complex filter structure is disadvantageous, a closed form expression for the mean number of stochastic comparisons would be necessary. Our conjecture is based solely on simulation results, and as such may not be very accurate.

Table 4.7: Filtered success probabilities and mean number of stochastic comparisons performed for different filter structures using the consecutive filter. The initial failure probability was \( q = 0.49 \).

<table>
<thead>
<tr>
<th>Filtering Structure Leader</th>
<th>Filtered Success Probability</th>
<th>Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>[M=100], [M=64]</td>
<td>0.982023, 0.928265</td>
<td>4795 ± 14, 2755 ± 6</td>
</tr>
<tr>
<td>[M=10], [M=10]</td>
<td>0.982023</td>
<td>4814 ± 50</td>
</tr>
<tr>
<td>[M=8], [M=8]</td>
<td>0.928265</td>
<td>2743 ± 9</td>
</tr>
<tr>
<td>[M=4], [M=16]</td>
<td>0.928265</td>
<td>2728 ± 9</td>
</tr>
<tr>
<td>[M=16], [M=4]</td>
<td>0.928265</td>
<td>2738 ± 9</td>
</tr>
<tr>
<td>[M=2], [M=32]</td>
<td>0.928265</td>
<td>2754 ± 21</td>
</tr>
<tr>
<td>[M=32], [M=2]</td>
<td>0.928265</td>
<td>2723 ± 27</td>
</tr>
<tr>
<td>[M=4], [M=4], [M=4]</td>
<td>0.928265</td>
<td>2739 ± 11</td>
</tr>
</tbody>
</table>
4.7 Conclusions

This chapter examined the use of multiple layer filtering structures constructed from the deterministic filters presented in Chapter 3. We have shown from simulation results that a hierarchical filtering structure is advantageous only when composed of consecutive filters. The consecutive filter was shown to be second best in Chapter 3 when operating under a low failure probability. Its performance degrades rapidly for higher failure probabilities and it is an unfeasible option in such situations. A filtering structure composed of multiple layers can greatly improve the performance of the consecutive filter. Indeed, in some cases this improvement is so great that it outperforms the majority and optimal majority filters.

We have shown that hierarchical filtering structures of majority and optimal majority filters are disadvantageous and lead to a degraded performance. A conclusive statement about leader filter structures could not be made due to the standard deviation of the simulation results obtained in this case.

This chapter ends the investigation of absorbing filters and their application to stochastic sorting. Apart from these filters there is another class of ergodic filters which are to be used in a slightly different way. Ergodic filters are able to output a deterministic answer at any point in time, and thus differ from absorbing ones which only output the result once absorption has occurred. Use of such ergodic filters will be the subject of discussion in Chapter 6; but before their introduction, Chapter 5 will first describe the scenario in which such filters would be effective.
5.1 Introduction

In this chapter we shall consider how we can sort in a stochastic environment using ergodic methods. First of all we shall study a completely different approach to stochastic sorting which includes the use of a list organizing model. List organizing methods have been extensively used in the problem of self-organizing sequential search [McCabe, 1965; Rivest, 1976; Arnow and Tenenbaum, 1982; Tenenbaum and Nemes, 1982; Knuth, 1986; Chung, et al., 1988]. Briefly, the problem is as follows; we have a list of N records \( [R_1, R_2, \ldots, R_N] \), where each record \( R_i \) is identified by a unique key \( K_i \). At each instant of time we are presented with a key, \( K \), and asked to retrieve the associated record. This is done by sequentially comparing each key from the beginning of the list until some key \( K_i \) is found which equals \( K \). The number of comparisons is the cost of this sequential search and is \( i \) in this case. If \( p_i \) is the probability that the search argument \( K \) equals \( K_i \) then the expected cost of each search is given by.
\[
\text{cost} = \sum_{i=1}^{N} i \rho_i.
\]

List organizing strategies attempt to organize the list of records in such a way as to minimize this cost. In fact it can easily be shown that the cost will be at its minimum when the records are arranged in the descending order of the \( \rho \)'s. In this chapter we shall study how we can achieve stochastic sorting by using list organizing methods to organize the list into a sorted order.

Analogous deterministic sorting techniques to which the list organizing model readily applies. These include bubble sort, Shell’s sort, and any type of exchange sort in general.

To introduce this method, consider how sorting is achieved using various deterministic sorting schemes which sort \emph{in situ}. Essentially the process is as follows: two values are picked to be compared and are exchanged if they are out of order. What differentiates each of the different deterministic exchange type algorithms is the way in which successive values are picked for comparison. Bubble sort compares adjacent values until it finds one that is out of order, and slowly ‘bubbles’ it to a place where it is no longer out of order [Knuth, 1986]. Shell’s sort compares values at wide distances apart, and the comparison scheme for heap sort is even more intricate. We shall utilize identical principles in stochastic sorting. First of all we will present the different ways of choosing the two values to be compared. After this, we shall study a few list organizing methods. Finally, we shall conclude with the simulation results for all the algorithms presented.

### 5.2 Choosing Values For Comparison

When dealing with sorting under a stochastic environment the problem of choosing two values for comparison can be viewed differently from the same problem under a deterministic environment. In a deterministic environment the aim is to find a scheme that will perform the minimum number of comparisons necessary to fully sort the whole input list of values. Under stochastic conditions such a goal is no longer feasible since an optimal
solution that sorts a list in a deterministic environment may leave the list unsorted in a stochastic environment. We will therefore always be making more comparisons than the minimum required when dealing with a stochastic environment. The problem of choosing values to be compared, now, rather becomes one of choosing them is such a way that will allow the list to approach the sorted state as much as possible if no errors occurred during the stochastic comparisons. But we would also require that the list simultaneously becomes less ‘unsorted’ if errors occur.

5.2.1 The Random Scheme

The simplest way, by far, is to pick two different values at random and compare them. This random scheme, in addition to being highly straightforward, will also serve as a benchmark for the schemes to follow. Any scheme which chooses two values for comparison should outperform the random scheme - otherwise it is not even expedient. The scheme for picking elements randomly is, of course, trivial but is given below for the sake of completeness.

Algorithm Random Choice:

_Aim:_ An algorithm which picks two values out of the input list to compare. The choice is done randomly.

_Inputs:_ N: Size of input list.

_Output:_ Indices of two values to be compared in the input list.

_Method:_

```plaintext
index1 = index2 = 0.
while (index1 = index2) do
    index1 = randomIntegerIn (1,N).
    index2 = randomIntegerIn (1,N).
endwhile
return (index1, index2).
End Algorithm Random Choice.
```
5.2.2 The Neighbour Scheme

The neighbour scheme is based on the Monrad and Butterfly tournaments discussed earlier in sections 2.4.1, and 2.4.2. Neighbouring values are picked for comparisons sequentially. If the input list contains an odd number of values then we alternate the start to be either the first or the second value in the list on consecutive phases. Figure 5.1 depicts this scheme pictorially and explains how it works.

<table>
<thead>
<tr>
<th>Input List</th>
<th>Indices Picked:</th>
<th>Corresponding Values Picked:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 &amp; 2, 3 &amp; 4</td>
<td>3 &amp; 1, 5 &amp; 7</td>
</tr>
<tr>
<td></td>
<td>2 &amp; 3, 4 &amp; 5</td>
<td>1 &amp; 5, 7 &amp; 2</td>
</tr>
<tr>
<td></td>
<td>1 &amp; 2, 3 &amp; 4</td>
<td>3 &amp; 1, 5 &amp; 7</td>
</tr>
<tr>
<td></td>
<td>2 &amp; 3, 4 &amp; 5</td>
<td>1 &amp; 5, 7 &amp; 2</td>
</tr>
<tr>
<td></td>
<td>...and so on...</td>
<td>...and so on...</td>
</tr>
</tbody>
</table>

Figure 5.1: The neighbour scheme showing successive choice of values picked for comparison. An even number of values in the input list causes no problems. If we have an odd number of elements the scheme alternates between starting from the first and starting from the second values in the input list.

Algorithm Neighbour:

Aim: An algorithm which picks two values out of the input list to compare. The choice is done according to the neighbour scheme.

Input: 
N: Size of input list.

n: Index of last value chosen. Initially n = 0.

Output: Indices of two values to be compared in the input list.

Method:
if n = N then n = 0.
if n = (N-1) then n = 1.
index1 = n+1.
index2 = n+2.
return (index1, index2).
End Algorithm Neighbour.
5.3 Move To Front

A well known list organization method is the Move To Front rule which moves the accessed element to the front of the list. This rule has been studied extensively in the literature [McCabe, 1965; Rivest, 1976; Knuth, 1986; Chung, et al., 1988] in connection with the problem of self-organizing sequential search. If an element from the input list is accessed it is immediately moved to the front, with the hope that subsequent retrieval costs will be minimized. In our case, if we choose to compare two elements, we would move the lesser value to the front and leave the larger one where it is. If the lesser value is already at the front of the list it remains there unmoved. Figure 5.2 describes this process with an example.

![Diagram](image)

Figure 5.2: Diagrammatic representation of the Move To Front list organizing scheme. Values 3 and 1 are picked and the smaller of the two is moved to the front. Subsequently values 2 and 4 are picked, and since 2 is already at the front it remains there unmoved.

Algorithm Move To Front:

**Aim:** An algorithm which uses the Move To Front list organizing strategy to sort.

**Input:** N: Size of input list.  
L: Input list of values.

**Output:** Indices of two values to be compared in the input list.

**Method:**

```
repeat
    pick i and j from {1, 2, ..., N} according to some scheme s.t. i ≠ j.
    if V_i ⪰ V_j then /* stochastic comparison */
        temp = V_j.
        index = j.
    else
        temp = V_i.
```
index = i.
endif
for n = (index-1) to 1 by -1
   \[ V_{n+1} = V_n. \]
endfor
\[ V_1 = \text{temp}. \]
until satisfied
End Algorithm Move To Front.

We now present an analysis of the Move To Front rule as applied to stochastic sorting. In preparation for this we first review the notation that will be used in the forthcoming analysis.

Let the input list of values be represented by \( V_1, V_2, \ldots, V_N \), where \( N \) is the size of the list. We say that if \( V_i \preceq V_j \) then \( V_i \) wins, and denote \( W_{ij} \) as the conditional probability that \( V_i \preceq V_j \), given by \( W_{ij} = \Pr[V_i \text{ wins} \mid V_i \text{ and } V_j \text{ are compared}] \). This is simply,

\[
W_{ij} = \begin{cases} 
   p_{ij} & \text{if } V_i < V_j \\
   q_{ij} & \text{if } V_i > V_j 
\end{cases}
\tag{5.3.1}
\]

Also \( s_i \) denotes the probability of \( V_i \) being chosen as a value to be compared, and \( p_{ij} (t) \) the total probability that \( V_i \) precedes \( V_j \) in the list at time instant \( t \). The values of \( p_{ij} (\infty) \) denote the asymptotic precedence probabilities. Where there is no ambiguity we shall omit the reference to time 't', remembering that \( p_{ij} \) refers to \( p_{ij} (t) \).

Note that by definition we have \( W_{ij} = 1 - W_{ji} \), and \( p_{ij} = 1 - p_{ij} \). We also, let \( w_i \) be the total probability that the value compared, \( V_i \), will be reported as the winner. Indeed, this probability can be found from:

\[
w_i = \sum_{k \neq i} \Pr[V_i \text{ wins} \mid V_i \text{ compared to } V_k] \Pr[V_i \& V_k \text{ are compared}],
\tag{5.3.2}
\]

therefore
\[
w_i = \sum_{k \neq i} W_{ki} \left( \frac{s_i s_k}{1 - s_i} + \frac{s_k s_i}{1 - s_k} \right).
\tag{5.3.3}
\]

Observe that, \( w_i \) is the probability of \( V_i \) winning at any time instant - for this event to occur \( V_i \) must be picked as one of the two values being compared and must win. The
following theorem is only valid under the assumption that the probabilities $s_i$ are independent of one another.

**Theorem 5.1:** The Move To Front rule results in an ordering where any value $V_i$ precedes another value $V_j$ with the asymptotic probability $\pi P_j(\infty)$ given by:

$$\pi P_j(\infty) = \frac{w_i}{w_i + w_j}.$$

**Proof:** Consider the conditional probability $\pi P_j(t+1)$ given that $\pi P_j(t)$. When comparing two elements of the input list there are sixteen unique possible cases that can occur. These sixteen cases are given in Table 5.1, along with the corresponding probabilities of occurrence. The cases are mutually exclusive and totally exhaustive.

Initially, we have $\pi P_j(0) = \frac{1}{2}$. Consider the value for $\pi P_j(t+1)$ given $\pi P_j(t)$. If $V_i$ is moved to the front then $\pi P_j(t+1) = 1$. Similarly, if $V_j$ is moved to the front then $\pi P_j(t+1) = 0$; otherwise it is equal to $\pi P_j(t)$. Thus $\pi P_j(t+1)$ can be expressed in terms of $\pi P_j(t)$ as follows:

$$\pi P_j(t+1) | \pi P_j(t) = \begin{cases} 
1 & \text{with probability } A, \\
0 & \text{with probability } B, \\
\pi P_j(t) & \text{with probability } C = 1 - A - B,
\end{cases}$$

(5.3.4)

where

$$A = s_i \frac{s_j}{1 - s_i} W_j + s_i \sum_{k \neq i,j} \frac{s_k}{1 - s_i} W_k + s_j \frac{s_i}{1 - s_j} W_j + \sum_{k \neq i,j} s_k \frac{s_i}{1 - s_k} W_k,$$

(5.3.5)

and

$$B = s_i \frac{s_j}{1 - s_i} W_i + \sum_{k \neq i,j} s_k \frac{s_j}{1 - s_k} W_k + s_j \frac{s_i}{1 - s_j} W_i + s_j \sum_{k \neq i,j} \frac{s_k}{1 - s_j} W_k.$$

(5.3.6)
Table 5.1: Table listing the sixteen different cases that occur in the Move To Front Rule, the corresponding probability of each case occurring and the effect on the value of $iP_j(t)$.

<table>
<thead>
<tr>
<th>Values chosen for stochastic comparison &amp; winner</th>
<th>Event occurs with probability</th>
<th>Value of $iP_j(t+1) \mid iP_j(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_i$</td>
<td>$s_i \frac{s_j}{1 - s_i} W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_j$</td>
<td>$s_i \frac{s_j}{1 - s_j} W_i$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_j, V_i&gt; \rightarrow V_j$</td>
<td>$s_j \frac{s_i}{1 - s_j} W_i$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_j, V_i&gt; \rightarrow V_i$</td>
<td>$s_j \frac{s_i}{1 - s_j} W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_k&gt; \rightarrow V_i$</td>
<td>$s_i \sum_{k \neq i, j} \frac{s_k}{1 - s_i} W_k$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_k&gt; \rightarrow V_k$</td>
<td>$s_i \sum_{k \neq i, j} \frac{s_k}{1 - s_i} W_i$</td>
<td>$iP_j(t)$</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_k$</td>
<td>$\sum_{k \neq i, j} \frac{s_i}{1 - s_k} W_i$</td>
<td>$iP_j(t)$</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_i$</td>
<td>$\sum_{k \neq i, j} \frac{s_i}{1 - s_k} W_k$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_j, V_k&gt; \rightarrow V_j$</td>
<td>$s_j \sum_{k \neq i, j} \frac{s_k}{1 - s_j} W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_j, V_k&gt; \rightarrow V_k$</td>
<td>$s_j \sum_{k \neq i, j} \frac{s_k}{1 - s_j} W_i$</td>
<td>$iP_j(t)$</td>
</tr>
<tr>
<td>$&lt;V_k, V_j&gt; \rightarrow V_k$</td>
<td>$\sum_{k \neq i, j} \frac{s_j}{1 - s_k} W_j$</td>
<td>$iP_j(t)$</td>
</tr>
<tr>
<td>$&lt;V_k, V_j&gt; \rightarrow V_j$</td>
<td>$\sum_{k \neq i, j} \frac{s_j}{1 - s_k} W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_k$</td>
<td>$\sum_{k \neq i, j} \frac{s_i}{1 - s_k} W_k$</td>
<td>$iP_j(t)$</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_i$</td>
<td>$\sum_{k \neq i, j} \frac{s_i}{1 - s_k} W_i$</td>
<td>$iP_j(t)$</td>
</tr>
</tbody>
</table>
Simplifying the expression in (5.3.5) and (5.3.6) and using equation (5.3.3) we obtain the following values for A, and B:

\[
A = s_i \sum_{k \neq i} \frac{s_k}{1 - s_i} W_k + \sum_{k \neq i} \frac{s_k}{1 - s_k} W_k = w_i,
\]

\[
B = \sum_{k \neq j} \frac{s_j}{1 - s_k} W_k + s_j \sum_{k \neq j} \frac{s_k}{1 - s_j} W_k = w_j.
\]  

(5.3.7)

By computing the total probability of \( iP_j(t+1) \) in terms of \( iP_j(t) \) we obtain the difference equation in (5.3.4) to yield,

\[
iP_j(t+1) = A + C_iP_j(t).
\]  

(5.3.8)

Solving equation (5.3.8) yields:

\[
iP_j(t) = A \sum_{i=0}^{t-1} C^i + C^i_iP_j(0) = A \frac{1 - C^t}{1 - C} + C^t_iP_j(0).
\]  

(5.3.9)

Taking the limit as \( t \to \infty \) we obtain,

\[
limit_{t \to \infty} iP_j(t) = A \frac{1 - C^t}{1 - C} + C^t_iP_j(0) = \frac{A}{1 - C} = \frac{A}{A + B}.
\]  

(5.3.10)

Substituting for A and B from equation (5.3.7) we obtain the result of Theorem 5.1.

Furthermore, for the case when all \( s_i \)'s are equivalent we obtain Lemma 5.1.

**Lemma 5.1:** For the simplified case when all \( s_i \)'s are equal to one another, and when \( p = \min \{ p_{ij} \} \), and \( q = 1 - p \), the asymptotic probability for \( iP_j(\infty) \) becomes:

\[
iP_j(\infty) = \frac{\sum_{k \neq i} W_k}{\sum_{k \neq i} W_k + \sum_{k \neq j} W_k} = \frac{(i-1)q + (N-i)p}{(i+j-2)q + (2N-i-j)p}.
\]

**Proof:** From our assumptions we have,
\[ _iW_j = \begin{cases} p, & \text{if } V_i < V_j \\ q, & \text{if } V_i > V_j \end{cases} \]  

(5.3.11)

Additionally, since all \( s_i \)'s are equivalent we have, \( s_i = \frac{1}{N} \) and therefore,

\[ A = \frac{2}{N(N-1)} \sum_{k \neq i} _iW_k, \quad B = \frac{2}{N(N-1)} \sum_{k \neq j} _jW_k. \]  

(5.3.12)

We assume with no loss of generality that \( V_i < V_{i+1} \). Then,

\[ \sum_{k \neq i} _iW_k = (i-1)q + (N-i)p, \]  

(5.3.13)

and thus,

\[ _iP_j(\infty) = \frac{(i-1)q + (N-i)p}{(i-1)q + (N-i)p + (j-1)q + (N-j)p}. \]  

(5.3.14)

Hence the result.

In the next section we examine a list organizing scheme which is very similar to the Move To Front. It is called the Move To Back rule which moves the greater element to the rear.

### 5.4 Move To Back

A similar rule to the Move To Front just investigated is the Move To Back rule. Two values are picked for comparison and the greater of the two is immediately moved to the back of the list. Should the greater value already be at the rear it remains there unmoved. Figure 5.3 demonstrates this strategy.
Algorithm Move To Back:

Aim: An algorithm which uses the Move To Back list organizing strategy to sort.

Input: N: Size of input list.
L: Input list of values.

Output: Indices of two values to be compared in the input list.

Method:
repeat
    pick i and j from \{1, 2, ..., N\} according to some scheme s.t. \(i \neq j\).
    if \(V_i \gg V_j\) then
        \textbf{temp} = \(V_j\).
        index = \(j\).
    else
        \textbf{temp} = \(V_i\).
        index = \(i\).
    endif
    for \(n = \text{index} \text{ to } (N-1)\)
        \(V_n = V_{n+1}\).
    endfor
    \(V_N = \text{temp}\).
until satisfied
End Algorithm Move To Back.

We continue with the analysis of the Move To Back rule in exactly the same fashion as was done the previous section.

Theorem 5.2: The Move To Back rule results in an ordering where the value \(V_i\) precedes the value \(V_j\) with the asymptotic probability \(\Pr_j (\infty)\) given by:
\[ iP_j(\infty) = \frac{w_i}{w_i + w_j}. \]

**Proof:** We examine the sixteen possible cases that may occur and find that the corresponding values of \( iP_j(t+1) \) in each case are exactly the same as those listed in Table 5.1. The analysis is therefore exactly the same, we have the same values for A and B as in the Move To Front scheme. Hence the theorem. \( \blacksquare \)

**Lemma 5.2:** For the simplified case when all \( s_i \)'s are equal to one another, and when \( p = \min \{p_{ij}\} \), and \( q = 1 - p \), the asymptotic probability for \( iP_j(\infty) \) becomes:

\[ iP_j(\infty) = \frac{\sum_{k=i}^{N} w_k}{\sum_{k=i}^{N} w_k + \sum_{k=j}^{N} w_k} = \frac{(i-1)q + (N-i)p}{(i+j-2)q + (2N-i-j)p}. \]

**Proof:** The proof of this lemma is analogous to the proof of Lemma 5.1 \( \blacksquare \)

The next list organizing scheme that will be presented is a combination of the two schemes already discussed. This scheme is novel since there is no equivalent scheme for the self-organizing sequential search process.

### 5.5 Move To Front and Back

Since the problem of stochastic sorting is different from that of the sequential search we can now conceive of a new list organizing strategy which is not directly applicable to the problem of the sequential search. This new list organizing strategy is, indeed, the Move To Front And Back rule. It is similar to the Move To Front in that the lesser element is moved to the front of the list. Additionally, however, the greater element is simultaneously moved to the rear of the list. This new list organizing strategy is depicted in Figure 5.4 and the algorithm is given below.
Algorithm Move To Front And Back:

Aim: An algorithm which uses the Move To Front And Back list organizing strategy to sort.

Input: 
N: Size of input list.
L: Input list of values.

Output: Indices of two values to be compared in the input list.

Method:
repeat
  pick i and j from \{1, 2, ..., N\} according to some scheme s.t. i ≠ j.
  if \( V_i \geq V_j \) then
    \( \text{tempF} = V_j \), \( \text{indexF} = j \).
    \( \text{tempB} = V_i \), \( \text{indexB} = i \).
  else
    \( \text{tempF} = V_i \), \( \text{indexF} = i \).
    \( \text{tempB} = V_j \), \( \text{indexB} = j \).
  endif
  if (indexB < indexF) then (indexB = indexB + 1).
  for n = (indexF - 1) to 1 by -1
    \( V_{n+1} = V_n \).
  endfor
  for n = indexB to (N - 1)
    \( V_n = V_{n+1} \).
  endfor
  \( V_1 = \text{tempF} \), \( V_N = \text{tempB} \).
until satisfied
End Algorithm Move To Front And Back.

We now present an analysis of the Move To Front And Back rule as applied to stochastic sorting.
Theorem 5.3: The Move To Front And Back rule results in an ordering where each value \( V_i \) precedes another value \( V_j \) with the probability \( \mathbb{P}_j \) given by:

\[
\mathbb{P}_i = \frac{s_j}{1 - s_j} \sum_{k \neq i, j} s_k \mathbb{W}_j + s_j \sum_{k \neq i, j} \frac{s_k}{1 - s_k} \mathbb{W}_j
\]

\[
= \frac{s_j}{1 - s_j} \sum_{k \neq i, j} s_k \mathbb{W}_j + s_j \sum_{k \neq i, j} \frac{s_k}{1 - s_k} \mathbb{W}_j + w_j + \frac{s_i}{1 - s_i} \sum_{k \neq i, j} s_k \mathbb{W}_i + s_i \sum_{k \neq i, j} \frac{s_k}{1 - s_k} \mathbb{W}_i
\]

Proof: As in the case of Theorem 4.1, we analyze all possible disjoint and exhaustive cases that can occur in the comparison process. Table 4.2 lists all such cases along with the probability of each case occurring. The values of the corresponding \( \mathbb{P}_j \)'s that will result are also tabulated. Note that the cases are mutually exclusive and totally exhaustive.

Initially, we have \( \mathbb{P}_j (0) = \frac{1}{2} \). Again the conditional value for \( \mathbb{P}_j (t+1) \) given \( \mathbb{P}_j (t) \) can be expressed in terms of \( \mathbb{P}_j (t) \) as before:

\[
\mathbb{P}_j (t+1) \mid \mathbb{P}_j (t) = \begin{cases} 
1 & \text{with probability } A, \\
0 & \text{with probability } B, \\
\mathbb{P}_j (t) & \text{with probability } C = 1 - A - B, 
\end{cases}
\]

(5.5.1)

In this case the values of \( A \) and \( B \) simplify to:

\[
A = w_i + \frac{s_j}{1 - s_j} \sum_{k \neq i, j} s_k \mathbb{W}_j + s_j \sum_{k \neq i, j} \frac{s_k}{1 - s_k} \mathbb{W}_j, 
\]

(5.5.2)

and

\[
B = w_j + \frac{s_i}{1 - s_i} \sum_{k \neq i, j} s_k \mathbb{W}_i + s_i \sum_{k \neq i, j} \frac{s_k}{1 - s_k} \mathbb{W}_i.
\]

(5.5.3)

Thus, the difference equation in (5.5.1) yields the total probability recurrence equations (5.3.8) and (5.3.9) from Section 5.3, which simplify to:

\[
\mathbb{P}_j (t) = A \frac{1 - C^t}{1 - C} + C^t \mathbb{P}_j (0).
\]

(5.5.4)
### Table 5.2: Table listing the sixteen different cases that occur in the Move To Front And Back rule, the corresponding probability of each case occurring and the effect on the value of $iP_j(t)$.

<table>
<thead>
<tr>
<th>Values chosen for stochastic comparison &amp; winner</th>
<th>Event occurs with probability</th>
<th>Value of $iP_j(t+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_i$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_j$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_i$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_j, V_i&gt; \rightarrow V_j$</td>
<td>$\frac{s_i}{s_j \cdot (1-s_j)} \cdot W_i$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_j, V_i&gt; \rightarrow V_i$</td>
<td>$\frac{s_i}{s_j \cdot (1-s_j)} \cdot W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_k&gt; \rightarrow V_i$</td>
<td>$\frac{\sum_{k \neq i,j} s_k}{1-s_i} \cdot W_k$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_k&gt; \rightarrow V_k$</td>
<td>$\frac{\sum_{k \neq i,j} s_k}{1-s_i} \cdot W_i$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_k$</td>
<td>$\frac{\sum_{k \neq i,j} s_i}{1-s_k} \cdot W_i$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_i$</td>
<td>$\frac{\sum_{k \neq i,j} s_i}{1-s_k} \cdot W_k$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_j, V_k&gt; \rightarrow V_j$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_j)} \cdot W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_j, V_k&gt; \rightarrow V_k$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_k, V_j&gt; \rightarrow V_j$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_j)} \cdot W_j$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_j&gt; \rightarrow V_k$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_j&gt; \rightarrow V_j$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_j&gt; \rightarrow V_k$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_i$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_j)} \cdot W_j$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_k$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_i$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_j)} \cdot W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_k$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_k$</td>
<td>0</td>
</tr>
<tr>
<td>$&lt;V_i, V_j&gt; \rightarrow V_i$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_j)} \cdot W_j$</td>
<td>1</td>
</tr>
<tr>
<td>$&lt;V_k, V_i&gt; \rightarrow V_k$</td>
<td>$\frac{s_j}{s_i \cdot (1-s_i)} \cdot W_k$</td>
<td>0</td>
</tr>
</tbody>
</table>
Taking the limit as \( t \to \infty \) we have,

\[
\lim_{t \to \infty} P_j(t) = \frac{A}{A + B}.
\] (5.5.5)

Substituting the values of \( A \) and \( B \) proves the theorem.

Lemma 5.3: For the simplified case when all \( s_i \)'s are equal to one another, and when \( p = \min \{ p_{ij} \} \), and \( q = 1 - p \), the asymptotic probability for \( P_j(\infty) \) becomes:

\[
\lim_{t \to \infty} P_j(t) = \frac{N - 1 + (p - q)(j - i) - q}{2N - 3 + p - q}.
\]

Proof: We rewrite equation (5.5.2) as,

\[
A = \frac{2}{N(N-1)} \sum_{k \neq i} W_k + \frac{2}{N(N-1)} \sum_{k \neq j} W_j - iW_j.
\] (5.5.6)

As was done in Lemma 5.1, we assume with no loss of generality that \( V_i < V_{i+1} \). Then,

\[
\sum_{k \neq i} W_k = (i-1)q + (N-i)p, \quad \text{and}
\] (5.5.7)

\[
\sum_{k \neq j} W_j = (i-1)p + (N-i)q.
\] (5.5.8)

Using (5.5.7) and (5.5.8) we substitute into (5.5.6) and simplify to obtain:

\[
A = \frac{2}{N(N-1)} (N-1 + (p-q)(j-i) - iW_j), \quad \text{and}
\] (5.5.9)

\[
B = \frac{2}{N(N-1)} (N-1 + (p-q)(i-j) - jW_j).
\] (5.5.10)

The value of \( iW_j \) may be found from equation (5.3.11) but we first need to convert it to a total probability. Thus,

\[
iW_j = p P_j(\infty) + q (1 - P_j(\infty))
\] (5.5.11)
since the event $V_i < V_j$ will occur with probability $\pi P_j(\infty)$ in the limit. Likewise, we also have,

$$jW_i = p (1 - \pi P_j(\infty)) + q \pi P_j(\infty)$$  \hspace{1cm} (5.5.12)

We now evaluate equation (5.5.5) to yield,

$$\pi P_j(\infty) = \frac{N - 1 + (p - q)(j - i) - q}{2N - 3 + p - q}.$$  \hspace{1cm} (5.5.13)

Hence the lemma.

5.6 Transposition Rule

Another list organizing strategy used in the self-organizing sequential search problem is the Transposition rule, which has been proven to be superior to the Move To Front rule [Rivest, 1976]. In our adaptation to stochastic sorting the element reported as lesser will be moved forwards in the input list by one position. Thus, effectively, it will be transposed with the element preceding it, the other element will remain unmoved. Figure 5.5 demonstrates the process.

![Diagram](image)

Figure 5.5: The Transposition list organizing scheme. Values 3 and 1 are picked and the smaller of the two is transposed forwards by one place. Subsequently values 2 and 4 are compared, and since 2 is already at the front of the list it remains there untransposed.

Algorithm Transposition:

**Aim:** An algorithm which uses the transposition list organizing strategy to sort.

**Input:** $N$: Size of input list.
L: Input list of values.

Output: Indices of two values to be compared in the input list.

Method:

repeat
    pick i and j from \{1, 2, ..., N\} according to some scheme s.t. i \neq j.
    if \( V_i \geq V_j \) then *stochastic comparison*
        Swap \((V_{i-1}, V_j)\).
    else
        Swap \((V_{i-1}, V_i)\).
    endif
until satisfied

End Algorithm Transposition.

We now present an analysis of the Transposition rule. To do this we model it as a Markov chain with \( N! \) states - one for each permutation of the \( N \) values.

Theorem 5.4: The Markov chain for the Transposition rule is time reversible.

Proof: Suppose \( N = 3 \), and consider the following path from state \((1,2,3)\) to itself

\((1,2,3) \rightarrow (2,1,3) \rightarrow (2,3,1) \rightarrow (3,2,1) \rightarrow (3,1,2) \rightarrow (1,3,2) \rightarrow (1,2,3)\)

The transition from \((1,2,3)\) to \((2,1,3)\) occurs whenever the following events occur:

(i) \( <V_2, V_1> \rightarrow V_2, \)

(ii) \( <V_1, V_2> \rightarrow V_2, \)

(iii) \( <V_2, V_3> \rightarrow V_2, \)

(iv) \( <V_3, V_2> \rightarrow V_2. \)

Observe, that the four events shown above are, in fact, the same events described by equation (5.3.2). Thus the value \( V_2 \) is transposed at any time instant with probability \( w_i \).

The product of the transition probabilities in the forward direction for the path is thus,

\[
w_2^2 w_3^2 w_3^2 w_1^2 w_2 = w_1^2 w_2^2 w_3^2 \tag{5.6.1}\]

whereas in the reverse direction, it is

\[
w_3^2 w_3^2 w_2^2 w_1^2 w_1 = w_1^2 w_2^2 w_3^2. \tag{5.6.2}\]
As the general result follows in much the same manner the Markov chain is indeed time reversible.

From the above theorem we are able to obtain an interesting property of the Transposition rule. Let $\Pr \left( \mathbf{L}_p, \mathbf{V}_i, \mathbf{V}_j, \mathbf{L}_s \right)$ denote the limiting probability of the list being in the state $(\mathbf{L}_p, \mathbf{V}_i, \mathbf{V}_j, \mathbf{L}_s)$ where $\mathbf{L}_p$ is some prefix list, and $\mathbf{L}_s$ is some suffix list. By Theorem 5.4 we have

$$\frac{\Pr \left[ \mathbf{L}_p, \mathbf{V}_i, \mathbf{V}_j, \mathbf{L}_s \right]}{\Pr \left[ \mathbf{L}_p, \mathbf{V}_j, \mathbf{V}_i, \mathbf{L}_s \right]} = \frac{w_i}{w_j}$$

(5.6.3)

for all permutations.

5.7 Transpose Forwards And Backward

A generalization of the Move To Front and Transposition rules, and adaptation to the problem of stochastic sorting yields the Transpose Forward and Backward rule. In this list organizing method the lesser element is transposed $N_f$ positions forwards in the list, while the greater element is transposed $N_b$ positions backwards in the list just as is done in the Move-k-Ahead rule for the self-organizing sequential search. Each element may only be moved up as far as the front or down as far as the back of the list. Observe that in this case the order in which the two elements are moved is significant. Figure 5.6 demonstrates the process when the lesser element is moved first, and Figure 5.7 when it is moved last.

![Figure 5.6: The Transpose Forward And Backward list organizing scheme. Values 3 and 1 are picked and the smaller of the two is moved forwards by two places, the greater value is then moved backwards by three places. Subsequently values 2 and 4 are picked and the process is repeated. $N_f = 2$ and $N_b = 3$ in the above transpositions.](image-url)
Note that when $N_f = N$, and $N_b = 0$ we essentially have the Move To Front rule already discussed and analyzed in Section 4.3. Also, when $N_f = 0$, and $N_b = N$, we obtain the Move To Back rule which, in our setting, is probabilistically equivalent to the Move To Front rule and has been studied in Section 4.4. The Move To Front And Back rule is covered by setting the parameters to $N_f = N$, and $N_b = N$. Likewise, to obtain the Transposition rule we would set $N_f = 1$, and $N_b = 0$. It can thus be seen that this generalized Transpose Forward And Backward rule is able to cover all list organizing rules previously discussed.

A mathematical analysis of such a generalized scheme is unfeasible, and has never been fully undertaken in the literature - even for the simpler problem of self-organizing sequential search. We too have not succeeded in analyzing this scheme but simply present it here as a conclusion to the list organizing schemes discussed so far.

**Algorithm Transpose Forward Backward:**

**Aim:** An algorithm which uses the Move To Front list organizing strategy to sort.

**Input:**
- $N$: Size of input list.
- $L$: Input list of values.
- $ToFront$: Number of positions to transpose lesser element forwards.
- $ToBack$: Number of positions to transpose greater element backwards.

**Output:** Indices of two values to be compared in the input list.
Chapter 5: Stochastic Sorting Using List Organizing Methods

Method:
repeat
    pick i and j from \{1, 2, ..., N\} according to some scheme s.t. i ≠ j.
    if \(V_i \triangleright V_j\) then
        tempF = \(V_j\), indexF = j.
        tempB = \(V_i\), indexB = i.
    /* stochastic comparison */
    else
        tempF = \(V_j\), indexF = i.
        tempB = \(V_j\), indexB = j.
    endif
    front = indexF - ToFront.
    back = indexB + ToBack.
    if (front < 1) then front = 1.
    if ((indexB < indexF) and (front < back)) then
        back = back + 1.
        indexB = indexB + 1.
    endif
    if (back>N) then back = N.
    for n = indexF to (front+1) by -1
        \(V_n = V_{n-1}\).
    endfor
    L_front = tempF.
    for n = indexB to (back-1)
        \(V_n = V_{n+1}\).
    endfor
    \(V_{back} = tempB\).
until satisfied
End Algorithm Transpose Forward Backward.

5.8 Exchange Rule

In this section we shall present the exchange rule list organizing scheme. Of all the list organizing schemes presented so far this is the only one in which the values eventually become absorbed into their correct positions when interacting with a deterministic environment. Note that all the previous schemes are ergodic and thus unsorting could occur at any point even under deterministic conditions. In the exchange rule two values to be compared are picked by one of the schemes described in Section 4.2. If they are in the proper order they are left unchanged. However, if they are out of order they are exchanged. Figure 5.8 demonstrates the process.
Algorithm Exchange:

**Aim:** An algorithm which uses the exchange list organizing strategy to sort.

**Input:** L: Input list of values.

**Output:** Indices of two values to be compared in the input list.

**Method:**

```
repeat
    pick i and j from {1, 2, ..., N} according to some scheme s.t. i ≠ j.
    if (i>j) then Swap(i, j).
    if V_i > V_j then Swap (V_i, V_j).
    until satisfied

End Algorithm Exchange.
```

Under a deterministic environment the exchange rule is absorbing in nature. As opposed to this, the former rules were all ergodic. But under stochastic conditions even the exchange rule becomes ergodic. This is due to the possibility of errors occurring in the stochastic comparisons, and thus an 'unsorting' operation can occur.

We now close this chapter with simulation results for the different list organizing schemes presented.

### 5.9 Simulation Methods And Results

We will now describe the simulations that were carried out in order to experimentally compare each of the different list organizing methods presented earlier. Each of the schemes presented were simulated separately. The simulations were carried out on
the same workbench described in Section 3.8; for each set of simulations 100 independent experiments were carried out. Since list organizing methods are ergodic in nature, to calculate the disorder an ensemble time average was taken at different intervals, rather than the simple average of disorders for each of the different experiments as was done in Chapter 3. This means that for each experiment an average disorder was calculated for some interval in the simulation, and then from the independent experiments an average of the time averages was obtained. Additionally, the standard deviation error was calculated for every disorder data point from the 100 independent experiments. As before, initially, the list was reversely sorted. For the first set of results that will be presented, values picked for comparison were chosen using the random scheme. In order to compare the two value picking schemes described in this chapter (Algorithm Random Choice and Algorithm Neighbour) the set of simulations was repeated under the neighbour value picking scheme. Also, the set of simulations was repeated again for the deterministic environment (i.e., when the failure probability is zero). This was done in order to obtain a bound on how good a disorder can be achieved by each list organizing scheme under ideal conditions.

Figure 5.9 shows the plot of simulation results for the Move To Front rule for different success probabilities ranging from \( p = 0.6 \) to 1.0. As can be seen, even when no errors occur \( (p=1.0) \) the Move To Front rule is unable to properly order the list, although it does lower the disorder up to a point. The reason for this is that the Move To Front rule is too destructive in its behaviour. Suppose two large values are being compared, both of which would appear near the rear of the sorted list, the Move To Front rule will necessarily move one of these values to the very first position in the list.

Figure 5.10 compares the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A list of 32 values was reorganized, where the individual failure probability for each stochastic comparison was 25%. There is no visible difference in performance between the three schemes, except for the fact that the Move To Front And Back rule is slightly quicker in its convergence. One is able to more readily discern the difference between the three schemes in Figure 5.14 where \( q = 0 \). From this plot we see that the Move To Front has the same performance as the Move To Back - as expected from our theory. However, the Move To Front And Back rule is slightly worse. This too agrees with the analytic result since the Move To Front And Back rule is not optimal. Indeed, in
Figure 5.9: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front list organizing scheme for different success probabilities. Observe that even with no errors the Move To Front rule does not achieve a zero disorder state. A set of 32 values was used in these results.

Figure 5.10: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A set of 32 values was used in these results, the failure probability was set at \( q = 0.25 \).
some cases, it predicts the wrong ordering for values in terms of the asymptotic precedence probabilities \( p_j \).

Consider the case for a list of four values, where the quantities for \( s_i \) and \( w_j \) are given as shown in Table 5.3. It is clear from the values of \( w_j \) that the sorted order should be \( (V_1, V_2, V_3, V_4) \). Indeed, this is the order most likely chosen by the Move To Front, and Move To Back rules, as can be seen from the asymptotic precedence probabilities shown in Table 5.4. When these probabilities are calculated for the Move To Front And Back list organizing scheme, it is found that this scheme prefers to order \( V_4 \) before \( V_3 \) - which is clearly incorrect. This simple example proves that the Move To Front And Back rule gives rise to precedence probabilities, \( p_j \), which are not in accordance with the desired ordering. The Move To Front, and Move To Back rules on the other hand, do not have such a problem.

Table 5.3: Example of a scenario in which the Move To Front And Back list organizing rule, estimates the wrong precedence order for the values.

<table>
<thead>
<tr>
<th>Probabilities of choosing ( V_i ):</th>
<th>( s_1 = 0.3, s_2 = 0.2, s_3 = 0.4, s_4 = 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilities of ( V_i ) winning over ( V_j ):</td>
<td>( 1w_2 = 0.70 )</td>
</tr>
<tr>
<td></td>
<td>( 1w_3 = 0.80 )</td>
</tr>
<tr>
<td></td>
<td>( 1w_4 = 0.90 )</td>
</tr>
</tbody>
</table>
Table 5.4: Asymptotic precedence probabilities $P_j$ as calculated for the scenario given in Table 5.3 for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. Observe that the value of $P_4$ for the Move To Front And Back rule indicates that $V_4$ is more likely to appear before $V_3$ in the list.

<table>
<thead>
<tr>
<th>Precedence Probabilities</th>
<th>Move To Front &amp; Move To Back</th>
<th>Move To Front And Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1P_2$</td>
<td>0.636</td>
<td>0.599</td>
</tr>
<tr>
<td>$1P_3$</td>
<td>0.694</td>
<td>0.734</td>
</tr>
<tr>
<td>$1P_4$</td>
<td>0.885</td>
<td>0.748</td>
</tr>
<tr>
<td>$2P_3$</td>
<td>0.567</td>
<td>0.666</td>
</tr>
<tr>
<td>$2P_4$</td>
<td>0.811</td>
<td>0.641</td>
</tr>
<tr>
<td>$3P_4$</td>
<td>0.750</td>
<td>0.379</td>
</tr>
</tbody>
</table>

Figure 5.11: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Backwards, and Transpose Forwards And Backwards list organizing schemes. A set of 32 values was used in these results, the failure probability was set at $q = 0.25$. 
Figure 5.11 compares the Transpose Forwards (Transposition), Transpose Backwards, and Transpose Forwards And Backwards rules for the case when elements are transposed by a single place in the list. Similarly as before, the Transpose Forwards And Backwards rule yields a more aggressive list organizing scheme. It is hard to say whether or not the Transpose Forwards And Backwards rule gives a worse asymptotic disorder than either Transpose Forwards, or the Transpose Backwards rules.

![Graph showing disorder vs. number of stochastic comparisons](image)

**Figure 5.12:** Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Transpose Forwards And Backwards by log N, and Exchange list organizing schemes. A set of 32 values was used in these results, the failure probability was set at \( q = 0.25 \).

To investigate this possibility, simulations of the Transposition rule were made for the case when the transposing is done by \( \log N \) places at a time - the results are shown in Figure 5.17. Upon examination of Figure 5.17 it becomes clear that the Transpose Forward And Backwards rule is, in fact, worse than either Transpose Forwards, or Transpose Backwards, when \( N_f = N_b = \log N \). We have already seen that the same is true if \( N_f = N_b = N \) - the Move To Front and Back family of rules. We thus conjecture that for any values \( N_f, N_b \) the Transpose Forwards And Backwards rule is worse than the Transpose Forwards rule, and the Transpose Backwards rule.
Figure 5.12 compares the Move To Front, Transpose Forwards And Backwards by \( \log N \), and Exchange rules, under a failure probability of 25%. The pattern seems to be recurrent, the more aggressive a rule is at the start the poorer its final solution is. In this case we have the 'Transpose Forwards And Backwards by \( \log N \)' rule providing the lowest disorder but taking the longest time to converge.

Figure 5.13 plots the results of the Transpose Forwards And Backwards scheme, for different values of \( N_f \) & \( N_b \). From Figure 5.13 it is clear that better accuracy is achieved by using smaller transposition values (a less destructive list organizing scheme) although it does take a bit longer for the scheme to converge. This result is in agreement with results obtained from the self-organizing sequential search problem where it has been shown that a Move-k1-Ahead rule outperforms a Move-k2-Ahead rule where \( k_1 < k_2 \) [Arnold and Tenenbaum, 1982].

Moreover, results shown in Figure 5.13 imply that a transposition scheme with parameters \( N_f = N_b = n \) is faster but less accurate than the same scheme with parameters \( N_f = N_b = n - 1 \). Figure 5.14 plots the results of simulations of the Transpose Forwards And Backwards scheme under unequal values of \( N_f \) and \( N_b \). From Figure 5.13 and 5.14 a few basic behaviours become apparent. Let \( TFB(N_f,N_b) \) denote the Transpose Forwards And Backwards rule with parameters \( N_f \) and \( N_b \). From our simulations we conjecture that \( TFB(i,j) \) is equivalent to \( TFB(j,i) \) when the values compared are randomly chosen. Furthermore, we have observed that \( TFB(i,j) \) achieves a lower disorder (although it may take more stochastic comparisons to stabilize) than either \( TFB(i,j+1) \), or \( TFB(i+1,j) \). In fact further simulations seemed to indicate that \( TFB(i,j) \) achieves a lower disorder than either \( TFB(i+1,k | k \leq \text{Max}(i,j)) \) or \( TFB(k | k \leq \text{Max}(i,j),j+1) \) in general.

Figures 5.15 to 5.18 show the results for the scenario where no errors occur. From these figures it becomes quickly apparent that the Transpose Forwards, Transpose Backwards, and Exchange rules are the only list organizing schemes capable of approaching a disorder of zero. This behaviour was predicted for the Exchange rule, although Figure 5.12 shows that the Exchange rule does not perform as well under the presence of errors. For this reason the use ergodic filters will be discussed in Chapter 6 to attenuate any such errors. It will also be shown shortly that using the neighbour scheme to choose values offers an improvement over the random picking scheme.
Figure 5.13: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards And Backwards list organizing scheme for different values of $N_f$ and $N_b$. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$.

Figure 5.14: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards And Backwards list organizing scheme for different values of $N_f$ and $N_b$. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$. 


Figure 5.15: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0$.

Figure 5.16: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forward, Transpose Backward, and Transpose Forward And Backward list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0$. 
Figure 5.17: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Backwards, and Transpose Forwards And Backwards rules. In each case the records were moved by log N, list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0$.

Figure 5.18: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Transpose Forwards And Backwards by log N, and Exchange list organizing schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0$. 
Figure 5.19: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front, Move To Back, and Move To Front And Back list organizing schemes. A set of 32 values was used in these results, the failure probability was set at \( q = 0.25 \), and the neighbour scheme was used to pick values.

Figure 5.19 shows the results of the Move To Front, Move To Back, and Move To Front And Back list organizing rules when the values are picked using the neighbour rule discussed in Section 5.2.2. Clearly, the performance of each is different, which is no surprise since our theoretical results are true only when values are picked randomly. The neighbour scheme does not possess this property and thus the Move To Back and Move To Front rules result in different behaviour. Amazingly, it appears as if the Move To Back rule outperforms the Move To Front rule when neighbouring values are compared.

Results for the Transposition rules when comparing neighbouring values are given in Figure 5.20. In this case, we can more clearly see that the Transpose Forwards And Backwards rule is less accurate than either the Transpose Forwards or the Transpose Backwards rules. This property manifests itself even when the failure probability is quite high, \( q = 0.25 \). Recall that this observation was only possible when \( q=0 \), when the compared values are randomly chosen. Additionally, the Transpose Backwards rule appears to be slightly better than the Transpose Forwards rule. This is similar to the conclusion that was drawn from Figure 5.19.
Figure 5.20: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Backwards, and Transpose Forwards And Backwards list organizing schemes. A set of 32 values was used in these results, the failure probability was set at $q = 0.25$, and the neighbour scheme was used to pick values.

Figure 5.21: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards, Transpose Forwards And Backwards, and Exchange list organizing schemes. A set of 32 values was used in these results, the failure probability was set at $q = 0.25$, and the neighbour scheme was used to pick values.
Figure 5.21 compares the Transpose Forwards, Transpose Forwards And Backwards, and Exchange rules operating under the neighbour scheme, and a failure probability of 25%. It is interesting to note that the performance of the Exchange rule vastly improves under the neighbour value picking scheme. This is more obvious in Figure 5.27.

Figure 5.22 compares the Transposition rule under the random value picking scheme, and the neighbour scheme. It is evident that for the Transposition rule, the neighbour value picking scheme is far more efficient. The same observation can be made for the Transpose Forwards, and Transpose Forwards And Backwards schemes; the graphs are omitted here but look very much the same as Figure 5.22. Comparison results for the Move To Front, and Back family of rules under the two value picking schemes are a little disappointing. Under the neighbour scheme the Move To Front performs slightly worse than under the random choice scheme. The Move To Back rule performs a little better, and the Move To Front And Back much worse. The respective graphs are shown in Figures 5.23 to 5.25 respectively.

Perhaps the results would not be so interesting, except for the fact that the Transpose Forward And Backwards by log N also performs worse under the scheme comparing neighbouring values than under the random choice scheme (see Figure 5.26). We have, however, seen that the Transpose Forwards and Backwards rule performs better under the neighbour values picking scheme. Thus, it seems as if there must be some number, n, for which comparing neighbouring values yields an improvement over the random choice scheme if the transposing is done by n positions.

Figure 5.27 shows the results for the exchange rule operating under the random choice and neighbour schemes. Although the Exchange rule becomes slower under the neighbour scheme it is able to achieve a much lower disorder.
Figure 5.22: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$.

Figure 5.23: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$. 

Chapter 5: Stochastic Sorting Using List Organizing Methods
Figure 5.24: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Back list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$.

Figure 5.25: Plot of average disorder vs. number of stochastic comparisons performed for the Move To Front And Back list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$. 

Chapter 5: Stochastic Sorting Using List Organizing Methods
Figure 5.26: Plot of average disorder vs. number of stochastic comparisons performed for the Transpose Forwards And Backwards ($N_f = N_b = \log N$) list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$.

Figure 5.27: Plot of average disorder vs. number of stochastic comparisons performed for the Exchange list organizing rule operating with the random and neighbour schemes. A set of 32 values was used in these results, and the failure probability was set at $q = 0.25$. 

Chapter 5: Stochastic Sorting Using List Organizing Methods
5.10 Conclusions

In this chapter we have examined the use of list organizing schemes to achieve stochastic sorting. Several list organizing schemes were introduced and analyzed and extensive simulations were carried out to compare the different schemes in terms of their speed of convergence and accuracy.

From the theoretical analysis it was found that the Move To Front and Move To Back rules are stochastically equivalent in terms of their asymptotic precedence probabilities. The simulation results justify this conclusion. It was also analytically found, that the Move To Front And Back rule is inferior to either the Move To Front, or Move To Back rule. In fact it sometimes predicts the wrong asymptotic precedence probabilities. This behaviour also manifested itself in the simulation results. The theoretical results are valid only for a value picking scheme where each value is picked independently of the other values. The random scheme satisfies these conditions, while the neighbour scheme does not. Since in the latter there is a strong dependence. Simulation results of the Move To Front and Move To Back rules under the neighbour scheme corroborate this, since both manifest different performance under the neighbour scheme.

It was found that list organizing rules which transpose elements by large distances i.e. Move To Front rule, Move To Back rule, etc., were more aggressive in their behaviour. This means that the disorder of the list quickly decreased at the start, but, the final disorder achieved upon stabilization was not good. To achieve a low disorder, list organizing rules have to be used which are not so drastic in their reorganization. Such rules would include the Transpose Forwards, and Backwards family of schemes, which move elements one position at a time. Of course, such schemes have the drawback of taking a much longer time to stabilize.

These transposition schemes can be greatly improved by using the neighbour value picking scheme rather than the random scheme. In this case, not only is the speed of convergence increased but also the asymptotic disorder reached has a lower value. The scheme which picks neighbouring values becomes detrimental when used by more drastic rules like the Move To Front. The reason for this behaviour can be explained by noticing
that whereas, neighbouring values in a ‘fairly’ sorted list should tend to stay together, a destructive list organizing scheme picks and disperses them, thereby increasing the disorder.

Another way to cut down on the number of stochastic comparisons performed, and still attain a low disorder would be to vary the values of \( N_f \) and \( N_b \) dynamically. This approach would only work if the list was fairly unsorted to begin with. For example, initially, the values of \( N_f \) and \( N_b \) could be set to \( N \) (the Move To Front family); and this aggressive scheme would quickly restructure the list subsequently the parameters of the scheme could then be decreased until they reached the value unity permitting the list to reach a more sorted state. Of course, such a scheme would be detrimental when used upon a list which is already fairly sorted since the aggressive scheme would tend to unsort it at the beginning.

Another important conclusion is that the speed of convergence increases with the magnitudes of \( N_f \) and \( N_b \). This conclusion may be inferred from the fact that the Move To Front, and Back family of rules converge faster than the Transpose by \( \log N \) rule, which in turn converges faster than any of the Transpose by one place rules. The asymptotic disorder achieved upon convergence is also proportional to the magnitudes of \( N_f \) and \( N_b \). Generally speaking, the smaller the jumps made by the values in a list organizing scheme the lower the disorder that can be achieved upon stabilization.

It is also noticeable that these relations are monotonic in nature. Thus a transposition scheme with parameters \( N_f = N_b = n \) will be faster but less accurate than the same scheme with parameters \( N_f = N_b = n - 1 \). This result concurs with results obtained in the problem of self-organizing sequential search [Arnow and Tenenbaum, 1982] where it has been found that a Move-(k-1)-Ahead rule performs better than a Move-k-Ahead rule.

A comparison of the values picking schemes seems to indicate that the neighbour value picking scheme is advantageous when it is used in conjunction with a less destructive scheme.

In the following chapter we introduce ergodic filters which will attempt to attenuate the failure probabilities so as to yield more efficient ergodic stochastic sorting algorithms.
6.1 Introduction

This chapter will discuss the use of filters, as in Chapter 3. These ‘devices’ filter a number of stochastic results and present a corresponding deterministic result. The major difference between the filters that will be discussed in this section and those discussed in Chapter 3 is that they will be ergodic ones as opposed to the absorbing ones used earlier. The ergodic filters will be able to produce a deterministic result at any point in time regardless of how many stochastic comparisons have been performed earlier.

The motivation for using these ergodic filters is that they will be utilized in conjunction with list organizing schemes. An example in which the Move To Front rule is used will help clarify this. In the latter scheme two values were stochastically compared and the value reported to be the lesser was immediately moved to the front. The probability of failure for these individual stochastic comparisons was, of course, q. In the filter setting,
the stochastic comparison result will be used by the filter to update the deterministic answer that it produces, and the filtered deterministic answer will subsequently be used by the Move To Front rule to move the lesser value to the front of the list. Effectively, we shall aim to shrink the failure probability through the use of ergodic filters. It is our intent that as more and more stochastic comparisons are performed the failure probability of the filtered result will asymptotically approach zero.

In order to make use of these filters we must store a filter for each combination of comparison values; this forces us to use $O(N^2)$ space. Upon completion of a stochastic comparison we update the appropriate filter and obtain the filtered result which is then utilized by the list organizing scheme. A diagrammatic representation of stochastic sorting using list organizing methods in conjunction with filtering is given in Figure 6.1.

![Diagram](image)

**Figure 6.1:** Diagrammatic representation of a sorting mechanism which uses a deterministic filter and a list organizing scheme.

We now present the first ergodic filter which is based on the well known Tsetlin automaton.

### 6.2 The Tsetlin Filter

We will attempt to filter stochastic results through the use of the linear tactic or Tsetlin automaton [Tsetlin, 1973]. Given two results for a stochastic comparison the learning automaton will attempt to pick the most correct one using the principle of the well-known Tsetlin automaton. We will, however, introduce a slight modification to it by inserting a central state, $\Phi_0$, as shown in Figure 6.2.
State $\phi_0$ is the initial state where the processing is initiated. A stochastic comparison of type $S_p$ propels the automaton towards state $P_M$, the other result $S_q$ propels it towards the opposite state $Q_M$. At any point in time the corresponding deterministic answer is $D_p$ if the automaton is in any state $P_i$, and $D_q$ if it is in any state $Q_i$. Should the automaton be in state $\phi_0$, a deterministic answer $D_p$ or $D_q$ is chosen randomly with equal probability. The formal algorithm is presented below.

**Algorithm Tsetlin Filter:**

**Aim:** An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the Tsetlin deterministic filter.

**Input:**

$M$: Tsetlin filter parameter. Tsetlin automaton will have $M$ states.

$k$: Tsetlin filter parameter. Exactly $k$ stochastic comparisons will be performed.

$Value1$, $Value2$: Values to be compared.

**Output:** A deterministic answer.

**Method:**

```plaintext
State = M.
for i = 1 to k
    if (Value1 > Value2) then /* stochastic comparison */
        State = State - 1.
    else
        State = State + 1.
    endif
    if (State < 0) then State = 0.
    if (State > 2M) then State = 2M.
endfor
if (State < M) then return (True).
if (State > M) then return (False).
return (randomIn(0, 1) < 0.5).
End Algorithm Tsetlin Filter.
```

† In the absorbing filters a deterministic result was only reported once the process reached the end states.
Suppose we perform $k$ stochastic comparisons. It can be shown that if $2M - 1 \geq k$ the resulting deterministic answer obtained by this filter is exactly equivalent to that of either majority scheme presented in Sections 3.2 and 3.3. In view of this we present the following theorem.

**Theorem 6.1:** If $2M - 1 \geq k$ the resulting deterministic answer obtained by the ergodic Tsetlin filter is exactly equivalent to that of either absorbing majority filter.

**Proof:** It is quite clear that the Tsetlin filter shown in Figure 6.2 with infinite memory ($M = \infty$) will produce the same result as either majority filter. This is because the automaton will end up on the side whose stochastic answers were a majority and choose the corresponding deterministic result.

Now suppose the automaton has a finite number of states, $M$, on each side. Whenever the automaton is in state $P_M$ and receives the stochastic comparison response $S_p$, this information is not utilized since the automaton remains in the same state; a similar situation may occur at state $Q_M$. The effects of this phenomenon can only manifest themselves in the filtered result if the filter produces the result $D_q$ when it would have produced $D_p$ - had a majority filter been used. This can first occur when $M+1$ responses of type $S_p$ and $M$ responses of type $S_q$ occur in the order $S_p, S_p, \ldots, S_p, S_q, \ldots, S_q$. In our case, this sequence of stochastic comparison results will leave the automaton in state $\phi_0$, and hence $D_q$ may be chosen. We thus obtain a deterministic result which is different from the result of the majority filter.

As soon as another state is added on each side this event cannot occur. Even though loss of information may occur, and the automaton will most likely end up in a state different from that if infinite states were available, it will still choose the same deterministic result since it will still be on the same side.

We can thus see that if $2M - 1$ stochastic comparisons are performed on the $(2M+1)$-state Tsetlin filter depicted in Figure 6.2 the filtered result is the same as for either majority filter. Thus the theorem.
A general mathematical analysis of the Tsetlin filter described here will now be presented. We analyze the Markov chain depicted in Figure 6.2, to find the asymptotic probabilities of the chain being in some specific state. To do so we first review some notation. We denote the transition matrix of the Markov chain by $F$, and the state probability vector by $\Pi_t = [\pi_0, \pi_1, ..., \pi_{2M}]$, where:

$$
\pi_i(t) = \begin{cases} 
\Pr [\phi(t) = Q_{M-i}], & 0 \leq i \leq M-1, \\
\Pr [\phi(t) = \phi_0], & i = M, \\
\Pr [\phi(t) = P_{i-M}], & M+1 \leq i \leq 2M.
\end{cases}
$$

(6.2.1)

The stationary state probability vector, which we are trying to find is denoted by $\Pi^*$, and is simply:

$$
\Pi^* = \lim_{t \to \infty} \Pi_t.
$$

(6.2.2)

We know from Markov chain theory [Ross, 1980; Karlin and Taylor, 1975] that the stationary state obeys the expression:

$$
F^T \Pi^* = \Pi^*
$$

(6.2.3)

Using equation (6.2.3) we are thus able to find $\Pi^*$, this unique solution.

**Theorem 6.2:** The stationary state probability vector for the Markov chain in Figure 6.2 is $\Pi^* = [\pi^*_0, \pi^*_1, ..., \pi^*_{2M}]$, where

$$
\pi^*_i = \frac{1 - \frac{p}{q}}{1 - \left(\frac{p}{q}\right)^{2M+1}} \left(\frac{p}{q}\right)^i, \quad 0 \leq i \leq 2M.
$$
Proof: We let

\[
A = \frac{1 - \frac{p}{q}}{2^{M+1}}, \quad \text{and thus } \pi_i = A \left(\frac{p}{q}\right)^i, \quad 0 \leq i \leq 2M.
\]

(6.2.4)

Substituting the values of \(\pi_i\) back into the R.H.S. of equation (6.2.3) we see:

\[
\begin{bmatrix}
q & p & 0 & 0 & 0 & \ldots & 0 \\
q & 0 & p & 0 & 0 & \ldots & 0 \\
0 & q & 0 & p & 0 & \ldots & 0 \\
& & & & & \ddots & \ddots \\
0 & \ldots & 0 & q & 0 & p & 0 \\
0 & \ldots & 0 & 0 & q & 0 & p \\
0 & \ldots & 0 & 0 & 0 & q & p \\
\end{bmatrix}
\begin{bmatrix}
A \\
A (p/q) \\
A (p/q)^2 \\
& & \ddots \\
& & & A (p/q)^{2M} \\
\end{bmatrix}
= 
\begin{bmatrix}
\pi^*_0 \\
\pi^*_1 \\
\pi^*_2 \\
& \ddots \\
\pi^*_2M \\
\end{bmatrix}
\]

(6.2.5)

To find if the R.H.S. indeed equals the L.H.S. we proceed to evaluate each value of \(\pi^*_i\).

The value of \(\pi^*_0\) as calculated from equation (6.2.5) is

\[
\pi^*_0 = qA + qA \frac{p}{q} = A (q + p) = A,
\]

(6.2.6)

satisfying the boundary condition on the left extreme.

Similarly, the value of \(\pi^*_2M\) is

\[
\pi^*_2M = pA \left(\frac{p}{q}\right)^{2M-1} + pA \left(\frac{p}{q}\right)^{2M} = A \left(\frac{p}{q}\right)^{2M} (q + p) = A \left(\frac{p}{q}\right)^{2M},
\]

(6.2.7)

which is indeed the value of the right extreme.

Finally any value of \(\pi^*_i\), for \(1 \leq i \leq 2M-1\) is given by:
\[ \pi^*_i = pA\left(\frac{p}{q}\right)^{i+1} + qA\left(\frac{p}{q}\right)^i = A\left(\frac{p}{q}\right)^i(q + p) = A\left(\frac{p}{q}\right)^i. \] (6.2.8)

Therefore, the vector \( \Pi^* \) is the unique eigenvector for the eigenvalue unity. Furthermore, we see that \( \Pi^* \) is a probability vector since,

\[ \sum_{i=0}^{2M} \pi^*_i = A\left(1 + \frac{p}{q} + \left(\frac{p}{q}\right)^2 + \left(\frac{p}{q}\right)^3 + \ldots + \left(\frac{p}{q}\right)^{2M}\right) \]

\[ = A \frac{1 - \left(\frac{p}{q}\right)^{2M+1}}{1 - \frac{p}{q}} = 1. \] (6.2.9)

Hence the theorem.

We could decide to make our Markov chain consist of an even number of states in total as shown in Figure 6.3. In this situation we would begin in state \( P_1 \) or \( Q_1 \) with equal probability rather than state \( \phi_0 \) as before. However, since the chain is ergodic, the initial probability vector has no effect on the stationary probability vector.

Having obtained Theorem 6.2 we now proceed to analyze the probability of obtaining a correct filtered result.

Figure 6.3: State diagram of the ergodic Tsetlin automaton with \( M \) states per action and no central state.

Theorem 6.3: If \( p \) is the probability of obtaining a correct stochastic answer, then, on converging, the Tsetlin filter with parameter \( M \) will yield a correct deterministic answer with probability:
\[
\text{Pr} [D_p \mid \text{Tsetlin filter used, with a central state}] = \frac{\binom{p}{q}^M}{2 \left( 1 - \binom{p}{q}^{2M+1} \right)} \left( 1 + \frac{p}{q} + 2 \binom{p}{q}^{M+1} \right).
\]

\[
\text{Pr} [D_p \mid \text{Tsetlin filter used, with no central state}] = \frac{1}{1 + \binom{q}{p}^M}.
\]

These probabilities increase monotonically with M. Furthermore both the filters are asymptotically optimal.

**Proof:** We find the expression for \( \text{Pr} [D_p \mid \text{Tsetlin filter used, with a central state}] \) by using the results of Theorem 6.2.

\[
\text{Pr} [D_p \mid \text{Tsetlin filter used, with a central state}] = \sum_{i=M+1}^{2M} \pi^*_i + \frac{1}{2} \pi^*_M. \quad (6.2.10)
\]

Expanding equation (6.2.10) and simplifying we obtain:

\[
\text{Pr} [D_p \mid \text{Tsetlin filter used, with a central state}] = \frac{\binom{p}{q}^M \left( 1 + \frac{p}{q} - 2 \binom{p}{q}^{M+1} \right)}{2 \left( 1 - \binom{p}{q}^{2M+1} \right)}. \quad (6.2.11)
\]

The monotonicity and asymptotic optimality can be shown to quickly follow from the above equation.

Similarly to find \( \text{Pr} [D_p \mid \text{Tsetlin filter used, with no central state}] \) we write,

\[
\text{Pr} [D_p \mid \text{Tsetlin filter used, with no central state}] = \sum_{i=M}^{2M} \pi^*_i. \quad (6.2.12)
\]

Expanding and simplifying we obtain:
\[ \Pr \{ D_p \mid \text{Tsetlin filter used, with no central state} \} = \frac{\left( \frac{p}{q} \right)^M}{1 + \left( \frac{p}{q} \right)^M} = \frac{1}{1 + \left( \frac{q}{p} \right)^M}. \] (6.2.13)

Observe that equation (6.2.13) is, amazingly, the same as equation (3.5.2) derived in Section 3.5. We thus have monotonicity as proved in equation (3.5.4) and asymptotic behaviour as shown in the theorem. \(

For the case where there is no middle state the minimal \( M \) required for a given accuracy \( \alpha \) is given by,

\[ M_{\min}(\alpha) = \min \left\{ M \mid \frac{1}{1 + \left( \frac{q}{p} \right)^M} \geq \alpha \right\}. \] (6.2.14)

which is the same as equation (3.59). For a \( p \) value of 0.8 \( M_{\min}(0.99) = 4 \).

This terminates the discussion on the ergodic Tsetlin filter whose asymptotic probability of obtaining a correct filtered result happens to be the same as that for the absorbing leader filter. We now continue with the presentation of another filter based on the underlying principle of the Krinsky automaton.

6.3 The Krinsky Filter

The Krinsky automaton can also be used to develop a deterministic filter. An extra initial state \( \Phi_0 \) is added to decide the starting state of the scheme. State \( \Phi_0 \) is the initial state where we start. A stochastic comparison of type \( S_p \) propels the automaton to state \( P_M \), the other result \( S_q \) propels it to the opposite state \( Q_M \). The automaton takes a long time to ‘unlearn’ once it reaches an internal state \( P_M \) or \( Q_M \). At any point in time the corresponding deterministic answer is \( D_p \) if the automaton is in any state \( P_i \) and \( D_q \) if it is in a state \( Q_i \).

We can view this automaton (see Figure 6.4) as being exactly equivalent to the standard Krinsky M-state, 2-action automaton with probabilities of starting in boundary...
states given by \( q \) for \( Q_M \) and \( p \) for \( P_M \), rather than \( \frac{1}{2} \) for \( Q_1 \) and \( P_1 \). Subsequently, we can apply the results for the Krinsky automaton to yield its asymptotic performance. The formal definition of the algorithm follows.

![State diagram](image)

**Figure 6.4:** State diagram of the ergodic Krinsky automaton with \( M \) states per action and a central start state of no return.

**Algorithm Krinsky Filter:**

**Aim:** An algorithm which returns a deterministic answer obtained from a set of stochastic answers by using the Krinsky deterministic filter.

**Input:**
- \( M \): Krinsky filter parameter. Krinsky automaton will have \( M \) states.
- \( k \): Krinsky filter parameter. Exactly \( k \) stochastic comparisons will be performed.
- \( \text{Value1, Value2} \): Values to be compared.

**Output:** A deterministic answer.

**Method:**

```plaintext
if (Value1 ≥ Value2) then /* stochastic comparison */
    State = 0.
else
    State = 2M-1.
endif
for i = 2 to k
    if (Value1 ≥ Value2) then /* stochastic comparison */
        if (State < M) then (State = 0) else (State = State - 1).
    else
        if (State >= M) then (State = 2M-1) else (State = State + 1).
    endif
endfor
return (State < M).
End Algorithm Krinsky Filter.
```

We now present the mathematical analysis of the Krinsky filter.
Chapter 6: Stochastic Sorting Using Ergodic Deterministic Filters

**Theorem 6.4:** If \( p \) is the probability of obtaining a correct stochastic answer, then, on converging, the Krinsky filter with parameter \( M \) will yield a correct deterministic answer with probability:

\[
\Pr[D_p \mid \text{Krinsky filter being used}] = \frac{p^M}{p^M + q^M}.
\]

This probability increases monotonically with \( M \) and asymptotically approaches unity as \( M \) approaches infinity.

**Proof:** The first part of the theorem is just a well known result from literature concerning the Krinsky automaton [Narendra and Thathachar, 1989]. It is the probability of asymptotically being in the action \( D_p \). We immediately see that the equation for \( \Pr[D_p] \) given in Theorem 6.4 is identical to equations (6.2.13) and (3.5.6). Thus the monotonicity and the asymptotic behaviour are proven as well.

The minimal \( M \) required for a given accuracy \( \alpha \) is given by,

\[
M_{\min}(\alpha) = \min \left\{ M \mid \frac{p^M}{p^M + q^M} \geq \alpha \right\}.
\]

(6.3.1)

which is the same as equations (6.2.14) and (3.5.6). For a \( p \) value of 0.8 \( M_{\min}(0.99) = 4 \).

It is clear from Theorems 6.4, 6.3, and 3.5 that the Krinsky, Tsetlin, and leader filters all have the same expression for obtaining the correct deterministic answer \( D_p \). The distinct difference which is important to note is that each filter arrives at the deterministic answer differently. The leader filter performs some unbounded but finite number of stochastic comparisons. The ergodic Tsetlin, and Krinsky filters, on the other hand, have the same expression for \( \Pr[D_p] \). In addition, both Tsetlin and Krinsky filters behave differently for a particular filtering. The Krinsky filter is more interested in consecutive responses of the same type - it takes a longer to exit from its internal state \( Q_M \) or \( P_M \). Also a single stochastic response of its type will send the filter back into the same internal state.

\[\dagger\] The equality of the expression for completely different filters with completely different asymptotic properties is absolutely fascinating and intriguing.
The Tsetlin filter however does not have this property. It will be thus quite interesting to compare these two filters through the means of simulation, along with the filters discussed in Chapter 3, to see how they behave for finite values of k.

6.4 Simulation Methods And Results

We close this chapter on stochastic sorting using ergodic deterministic filters by describing the simulations that were carried out in order to experimentally compare each of the different list organizing methods presented earlier operating in conjunction with the ergodic deterministic filters. The simulations were carried out on the same platform as described in Section 3.8. Disorder values were calculated as explained in Section 5.10.

Figure 6.5 shows the graphs for the Transpose Forwards list organizing scheme acting in tandem with the ergodic Tsetlin filter. The error-free case for p = 1.0 is also plotted in this figure to act a benchmark, and to see if the ergodic filters are actually able to filter out all errors and eventually approach the same asymptotic disorder value as that reached in an error-free environment. The results show that performance increases with the filter parameter, at no loss to the speed of convergence. The final disorder reached by the Transpose Forwards rule operating with a 17-state Tsetlin filter (M=8) is practically the same as that reached in an error-free environment.

Figure 6.6 plots results for the same scenarios as in Figure 6.5 but using the neighbour scheme to pick values for comparison. The improvement obtained through the use of ergodic deterministic filters is clearly evident. Figure 6.7 and 6.8 repeat the same set of simulation, using the Exchange rule. Such results are typical.

In Figure 6.9 the results for Krinsky filter are plotted. The same scenarios were examined as in Figure 6.5. The Krinsky filter also helps to increase the overall performance, just as the Tsetlin filter does. However, if the parameter, M, is chosen to be too large the performance tends to decrease. Observe that in Figure 6.9 the performance curve for a Krinsky filter parameter of M = 8 is much worse than the corresponding curve for a parameter of M = 4.
Figure 6.5: Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating in conjunction with the ergodic Tsetlin filter for different parameters. The failure probability was \( q = 0.25 \), the list size was 32, and the random scheme was used to pick the values to be compared. The error-free case is included as well.

Figure 6.6: Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Tsetlin filter for different parameters. The failure probability was \( q = 0.25 \), the list size was 32, and the neighbour scheme was used to pick values.
Figure 6.7: Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Tsetlin filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values. The error-free case is included as well.

Figure 6.8: Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Tsetlin filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the neighbour scheme was used to pick values. The error-free case is included as well.
Another inherent characteristic of its behaviour also becomes noticeable - the higher the parameter of the Krinsky filter the slower the speed of convergence. This is due to the fact that the Krinsky filter takes a ‘long’ time to unlearn any particular decision. Since it could be initially put into the incorrect internal state with probability \( q \), it must first unlearn this incorrect decision before any improvement can take place. Of course, if \( M \) is too high, it will not have enough time to unlearn - thus the reason for the degradation in performance for larger values of \( M \).

![Graph](image)

**Figure 6.9:** Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Krinsky filter for different parameters. The failure probability was \( q = 0.25 \), the list size was 31, and the random scheme was used. The error-free case is also included.
Figure 6.10: Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Krinsky filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the neighbour scheme was used. The error-free case is also included.

Figure 6.11: Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Krinsky filter for different parameters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values. The error-free case is included as well.
Figures 6.10 through 6.12 show the analogous results of Figures 6.6 through 6.8 except that they utilize the Krinsky filter. The same degradation in performance is observed for larger values of M. This behaviour is further observed in Figures 6.13 and 6.14 which compare the performance of the Transpose Forwards list organizing scheme operating with the Tsetlin and Krinsky filters. For the filter parameter M = 2 both, the Tsetlin, and Krinsky filters are essentially identical, as can be seen in Figure 6.13. However, even for as small a value as M = 4 a difference in filter performances may be observed (see Figure 6.14). Note that the scheme using the Krinsky filter takes a little longer to converge, and the disorder is marginally little higher.

This completes our simulation results for the ergodic deterministic filters discussed in this chapter.

![Graph](image)

Figure 6.12: Plot of average disorder vs. number of stochastic comparisons performed by the Exchange rule operating with the ergodic Krinsky filter for different parameters. The failure probability was q = 0.25, the list size was 31, and the neighbour scheme was used to pick values. The error-free case is included as well.
Figure 6.13: Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Tsetlin and Krinsky filters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values.

Figure 6.14: Plot of average disorder vs. number of stochastic comparisons performed by the Transpose Forwards rule operating with the ergodic Tsetlin and Krinsky filters. The failure probability was $q = 0.25$, the list size was 31, and the random scheme was used to pick values.

6.5 Conclusions

In this chapter we introduced the concept of ergodic deterministic filters. Such filters are easily used in conjunction with any of the list organizing methods presented in
Chapter 5. The ergodic filters may be used to greatly attenuate any errors induced by the stochastic environment.

As expected, the performance of any list organizing scheme from Chapter 5 can be improved to the maximum by utilizing an ergodic deterministic filter. The improved method will eventually reach the same disorder as would be reached in an error-free environment for that particular rule, although it will take more time to do so.

In comparing the two different ergodic filters that were presented in this chapter we have come to the conclusion that the Tsetlin filter is the superior one of the two. This is due to the fact that the Krinsky filter takes too much time to unlearn a wrong decision. In this case a crucial decision involves the memory required by the Krinsky filter in order to minimize the overall number of stochastic comparisons. This decision is not so crucial for the Tsetlin filter.
CHAPTER 7

Final Remarks And Conclusions

7.1 Overall Conclusion

In this thesis the problem of stochastic sorting was examined. A review of the literature relevant to this topic was first presented. This review included a collection of several papers on the topics of noisy binary searching and fault tolerant sorting circuits. In addition to this, several popular tournament algorithms were also outlined. All the algorithms which could be applied to stochastic sorting were also presented, implemented, and compared through means of extensive simulations.

In the next part of the thesis the concept of deterministic filters was introduced. This concept is quite novel to the field of stochastic sorting. The different deterministic filters were utilized in conjunction with ordinary deterministic sorting algorithms to produce stochastic sorting algorithms. Perhaps the most valuable result obtained in the study of these filters is the fact that we have devised methods able to pick out the superior element in a comparison with an accuracy larger than what is possible using conventional methods.
In order to ameliorate the performance of deterministic filters, hierarchical filter structures were investigated. It turned out that multiple level structures of deterministic filters are sometimes advantageous depending on what type of filter the structure is composed of.

The second part of the thesis dealt with ergodic stochastic sorting algorithms. In this part list organizing methods were introduced and analyzed. Although most of these algorithms are not able to completely sort, they do very closely approximate the sorted order even under a non-stationary environment. These algorithms were further enhanced through the introduction of ergodic deterministic filters. The ergodic deterministic filters attenuate any errors and cause the list organizing schemes to function as if they were operating within an almost error-free environment. With the aid of ergodic deterministic filters a list organizing scheme which is able to completely sort is possible. Furthermore, this scheme is ergodic and is able to 'track' the current solution in a non-stationary environment.

The following section presents an overall comparison of all the algorithms presented in this thesis in terms of their stochastic sorting ability.

7.2 Comparison Of All Algorithms Presented

In this section we will present an overall comparison of the algorithms discussed in this thesis. The algorithms will be compared in terms of the asymptotic disorder that is achievable, and the mean number of stochastic comparisons necessary to converge to this disorder. These numbers were obtained through extensive simulations.

Results from the presented algorithms fall into two categories, the first one includes algorithms which approach a sorted order but never attain it. To this group belong the Monrad tournament and the Move To Front list organizing scheme, as well as others. The second category includes algorithms which are eventually able to achieve the exact sorted order. In this category fall the Exchange rule operating with an ergodic filter, as well as the stochastic sorting algorithms which work in conjunction with deterministic filters.

The asymptotic disorder achieved by the first category of algorithms was calculated as the weighted mean of the individual disorders evaluated at different time intervals after
the scheme has come to its steady state mode of operation. This weighted mean is obtained by the formula:

\[ x_{wm} = \frac{\sum_{i=1}^{N} \frac{x_i}{\sigma_{x_i}}}{\sum_{i=1}^{N} \frac{1}{\sigma_{x_i}}}, \text{ and } \sigma_{x_{wm}} = \left( \frac{\sum_{i=1}^{N} \frac{1}{\sigma_{x_i}}}{} \right)^{\frac{1}{2}}. \]  

(7.1.1)

For the second category of algorithms, for each case, a set of 100 independent experiments was performed to obtain an estimate on the percentage of correctly sorted results, as well as the mean number of stochastic comparisons to achieve this sorted state. The whole set of 100 experiments was then repeated 10 times to obtain standard deviation errors on both the measured values.

In all the simulations carried out a list of 31 values was used, and the failure probability was q=0.25. As usual, the list was initially in a reversely sorted order. Table 7.1 lists the results for algorithms presented in Chapter 2. Algorithm Feige is the only one capable of sorting the original list exactly. The tournament algorithms, on the other hand, are capable of sorting in a non-stationary environment much like the list organizing schemes discussed in Chapters 5 and 6. For this category of algorithms the value for the mean number of stochastic comparisons was approximately extrapolated from the place where the algorithm stabilized.

Table 7.2 tabulates the results for stochastic sorting algorithms introduced in Chapter 3. The algorithm with the best performance turns out to be the stochastic Insertion Sort. It is able to correctly sort, a list of 31 values, 99% of the time. It performs, on the average, only 1570 stochastic comparisons to achieve this when the failure probability is q=0.25.

Results for the list organizing algorithms introduced in Chapter 5 are presented in Tables 7.3 and 7.4. Tables 7.5 and 7.6 report the results using the same algorithms, but acting in conjunction with the ergodic Tsetlin filter. The memory of the filter was set to M=1000.
Table 7.3 lists results for all list organizing schemes which pick values randomly for comparison but use no additional filter mechanism. The Transposition family and Exchange rules are the superior ones. Table 7.4 reports results on the same simulations as performed in Table 7.3 but uses the neighbour scheme to choose values for comparison. When neighbouring values are picked for comparison substantial improvement in performance is obtained for the Transpose Forwards and the Exchange rules.

Performance results of list organizing schemes acting in conjunction with ergodic deterministic filters are tabulated in Table 7.5. When compared to the results in Table 7.3 one can easily notice that, as expected, the use of the ergodic Tsetlin filter improves the performance. Table 7.6 again compares the schemes when neighbouring values are picked for comparison. It can be seen that the exchange rule operating in conjunction with the ergodic Tsetlin filter is able to completely sort, and its performance is further improved by the used of the neighbour scheme.

Table 7.1: Asymptotic disorder results for algorithms introduced in Chapter 2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic Disorder Reached</th>
<th>Mean Number of Stochastic Comparisons</th>
<th>Mean Percentage Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feige</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q=0.01, mMultiplier=3</td>
<td>0.0</td>
<td>1195 ± 2</td>
<td>54 ± 2</td>
</tr>
<tr>
<td>Q=0.005, mMultiplier=4</td>
<td>0.0</td>
<td>1648 ± 2</td>
<td>86 ± 2</td>
</tr>
<tr>
<td>Q=0.001, mMultiplier=5</td>
<td>0.0</td>
<td>2341 ± 2</td>
<td>98 ± 1</td>
</tr>
<tr>
<td>Monrad</td>
<td>0.0017 ± 0.0004</td>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>Wins - Random</td>
<td>0.0014 ± 0.0002</td>
<td>30000</td>
<td>0</td>
</tr>
<tr>
<td>Wins - Neighbour</td>
<td>0.0035 ± 0.0004</td>
<td>5000</td>
<td>0</td>
</tr>
<tr>
<td>Points - Random</td>
<td>0.0010 ± 0.0001</td>
<td>30000</td>
<td>0</td>
</tr>
<tr>
<td>Points - Neighbour</td>
<td>0.0018 ± 0.0003</td>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>Butterfly</td>
<td>0.3656 ± 0.0376</td>
<td>80</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 7.2: Results for stochastic sorting algorithms using the deterministic leader filter introduced in Chapter 3.

<table>
<thead>
<tr>
<th>Deterministic Sorting Algorithm</th>
<th>Mean Number of Stochastic Comparisons</th>
<th>Mean Percentage Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insertion Sort</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leader M=5</td>
<td>974 ± 4</td>
<td>71 ± 5</td>
</tr>
<tr>
<td>Leader M=6</td>
<td>1181 ± 4</td>
<td>88 ± 3</td>
</tr>
<tr>
<td>Leader M=7</td>
<td>1373 ± 8</td>
<td>96 ± 1</td>
</tr>
<tr>
<td>Leader M=8</td>
<td>1571 ± 3</td>
<td>99 ± 1</td>
</tr>
<tr>
<td><strong>Shell's Sort</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leader M=4</td>
<td>1342 ± 8</td>
<td>57 ± 2</td>
</tr>
<tr>
<td>Leader M=5</td>
<td>1637 ± 5</td>
<td>86 ± 4</td>
</tr>
<tr>
<td>Leader M=6</td>
<td>1969 ± 12</td>
<td>94 ± 3</td>
</tr>
<tr>
<td>Leader M=7</td>
<td>2286 ± 4</td>
<td>98 ± 1</td>
</tr>
<tr>
<td><strong>Heap Sort</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leader M=5</td>
<td>1915 ± 18</td>
<td>54 ± 4</td>
</tr>
<tr>
<td>Leader M=6</td>
<td>2336 ± 10</td>
<td>80 ± 3</td>
</tr>
<tr>
<td>Leader M=7</td>
<td>2742 ± 14</td>
<td>95 ± 2</td>
</tr>
<tr>
<td>Leader M=8</td>
<td>3135 ± 14</td>
<td>97 ± 1</td>
</tr>
</tbody>
</table>
Table 7.3: Asymptotic disorder results for list organizing schemes introduced in Chapter 5. The random scheme was used to choose values for comparison.

<table>
<thead>
<tr>
<th>List Organizing Scheme - Random / No Filter</th>
<th>Asymptotic Disorder Reached</th>
<th>Mean Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move To Front</td>
<td>0.3663 ± 0.0302</td>
<td>200</td>
</tr>
<tr>
<td>Move To Back</td>
<td>0.3821 ± 0.0343</td>
<td>200</td>
</tr>
<tr>
<td>Move To Front And Back</td>
<td>0.3657 ± 0.0324</td>
<td>50</td>
</tr>
<tr>
<td>Transpose Forwards</td>
<td>0.0713 ± 0.0153</td>
<td>4000</td>
</tr>
<tr>
<td>Transpose Backwards</td>
<td>0.0715 ± 0.0294</td>
<td>4000</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards</td>
<td>0.0716 ± 0.0064</td>
<td>2500</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards by log N</td>
<td>0.1857 ± 0.0108</td>
<td>300</td>
</tr>
<tr>
<td>Exchange</td>
<td>0.3189 ± 0.0120</td>
<td>200</td>
</tr>
</tbody>
</table>
Table 7.4: Asymptotic disorder results for list organizing schemes introduced in Chapter 5. The neighbour scheme was used to choose values for comparison.

<table>
<thead>
<tr>
<th>List Organizing Scheme - Neighbour / No Filter</th>
<th>Asymptotic Disorder Reached</th>
<th>Mean Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move To Front</td>
<td>0.3376 ± 0.0160</td>
<td>200</td>
</tr>
<tr>
<td>Move To Back</td>
<td>0.3273 ± 0.0183</td>
<td>200</td>
</tr>
<tr>
<td>Move To Front And Back</td>
<td>0.4010 ± 0.0185</td>
<td>50</td>
</tr>
<tr>
<td>Transpose Forwards</td>
<td>0.0066 ± 0.0006</td>
<td>2000</td>
</tr>
<tr>
<td>Transpose Backwards</td>
<td>0.0980 ± 0.0073</td>
<td>2000</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards</td>
<td>0.0238 ± 0.0020</td>
<td>1500</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards by log N</td>
<td>0.1653 ± 0.0098</td>
<td>500</td>
</tr>
<tr>
<td>Exchange</td>
<td>0.0053 ± 0.0004</td>
<td>2500</td>
</tr>
</tbody>
</table>
Table 7.5: Asymptotic disorder results for list organizing schemes operating in conjunction with the ergodic deterministic Tsetlin filter introduced in Chapter 6. The random scheme was used to choose values for comparison.

<table>
<thead>
<tr>
<th>List Organizing Scheme - Random / Tsetlin</th>
<th>Asymptotic Disorder Reached</th>
<th>Mean Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move To Front</td>
<td>0.2359 ± 0.0198</td>
<td>4000</td>
</tr>
<tr>
<td>Move To Back</td>
<td>0.2375 ± 0.0342</td>
<td>4000</td>
</tr>
<tr>
<td>Move To Front And Back</td>
<td>0.2525 ± 0.0261</td>
<td>3000</td>
</tr>
<tr>
<td>Transpose Forwards</td>
<td>0.0352 ± 0.0045</td>
<td>8000</td>
</tr>
<tr>
<td>Transpose Backwards</td>
<td>0.0353 ± 0.0034</td>
<td>8000</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards</td>
<td>0.0361 ± 0.0021</td>
<td>7000</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards by log N</td>
<td>0.0981 ± 0.0040</td>
<td>5000</td>
</tr>
<tr>
<td>Exchange</td>
<td>0.0020 ± 0.0011</td>
<td>18 000</td>
</tr>
</tbody>
</table>
Table 7.6: Asymptotic disorder results for list organizing schemes operating in conjunction with the ergodic deterministic Tsetlin filter introduced in Chapter 6. The neighbour scheme was used to choose values for comparison.

<table>
<thead>
<tr>
<th>List Organizing Scheme - Neighbour / Tsetlin</th>
<th>Asymptotic Disorder Reached</th>
<th>Mean Number of Stochastic Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move To Front</td>
<td>0.2434 ± 0.0055</td>
<td>3000</td>
</tr>
<tr>
<td>Move To Back</td>
<td>0.1658 ± 0.0058</td>
<td>3000</td>
</tr>
<tr>
<td>Move To Front And Back</td>
<td>0.3199 ± 0.0047</td>
<td>2'000</td>
</tr>
<tr>
<td>Transpose Forwards</td>
<td>0.0012 ± 0.0001</td>
<td>4000</td>
</tr>
<tr>
<td>Transpose Backwards</td>
<td>0.2272 ± 0.0166</td>
<td>4000</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards</td>
<td>0.0015 ± 0.0002</td>
<td>3000</td>
</tr>
<tr>
<td>Transpose Forwards And Backwards by log N</td>
<td>0.0242 ± 0.0025</td>
<td>4000</td>
</tr>
<tr>
<td>Exchange</td>
<td>0</td>
<td>4000</td>
</tr>
</tbody>
</table>

7.3 Future Research Topics

As of date, we believe that this thesis contains the first documented complete coverage of stochastic sorting. It is our hope that it will serve as a review and general introduction and also as a stepping stone for future researchers. Indeed, various avenues of research still lie unexplored.

Although we have introduced the concept of deterministic filters and designed and utilized many such filters in this thesis, further work on this topic of deterministic filters would be valuable. Especially important would be the search for a filter which would improve on the characteristics of the leader filter. Also, a calculation of the mean number of stochastic comparisons performed by the leader, and consecutive filters would be useful, since this would allow the researcher to quantitatively determine the advantage of
hierarchical filter structures. The whole area of obtaining superior stochastic filters still remains open.

In the area of stochastic sorting using list organizing methods, research into the problem of choosing values to be compared is relatively scanty. Analysis of these schemes in terms of their stochastic behaviour would be interesting.

Finally, the application of learning methods to the problem of stochastic sorting still lies uninvestigated. These learning methods such as the Linear Reward-Inaction (LRI) automaton and the discretized schemes [Narendra and Thathachar, 1989] could possible be used to order the stochastic values, and also to estimate their relative magnitudes.
REFERENCES


