



# Cellular Automata as an Approximate Method in Structural Analysis

by

**M. P. Hindley**

A dissertation submitted in partial fulfillment  
of the requirements for the degree of

**Master of Engineering**

in the Department of Mechanical and Aeronautical Engineering,  
University of Pretoria

November 2001

Supervisor:

Prof. Albert A. Groenwold



# Abstract

**Title:** Cellular Automata as an Approximate Method in Structural Analysis

**Author:** Michael Philip Hindley

**Supervisor:** Prof. A.A. Groenwold

**Department:** Department of Mechanical Engineering

**Degree:** Master of Engineering

**Keywords:** Cellular automata, structural analysis, finite difference method, finite element method, boundary element method

This thesis deals with the mathematical idealization denoted cellular automata (CA) and the applicability of this method to structural mechanics. When using CA, all aspects such as space and time are discrete. This discrete nature of CA allows for ease of interaction with digital computers, while physical phenomena which are essentially discrete in nature can be simulated in a realistic way. The application of such a novel numerical method opens up new possibilities in structural analysis.

In this study, the fundamentals of CA are studied to determine how the parameters of the method are to be evaluated and applied to the established field of structural analysis. Attention is given to the underlying mathematics of structural mechanics, as well as approximate methods currently used in structural analysis, e.g. the finite element method (FEM) and the boundary element method (BEM).

For structural simulations performed with the CA implemented in this study, machine learning based on a genetic algorithm (GA) is used to determine optimum rules for the CA, using finite element, boundary element and analytical approximations as the basis for machine learning.

Rather unconventionally, symmetric problems in structural analysis are analyzed using asymmetric rules in the machine learning process, where the symmetry of the solution found is used as a quantitative indication of the quality of the solution. It is demonstrated that the quality of the asymmetric rules is superior to the quality of symmetric rules, even for those



problems that are symmetric in nature.

Finally, exploiting the inherent parallelism of CA, it is shown that distributed computing can greatly improve the efficiency of the CA simulation, even though the speed-up factor is not necessarily proportional to the number of sub lattices used.

The distributed computing device itself is constructed by combining 18 obsolete Pentium computers in a single cluster. In terms of CPU performance the constructed distributed computer is not state-of-the-art, but it is constructed with no hardware costs whatsoever. In addition, the software used in assembling the cluster is in the public domain, and is also available free of charge. Such a parallel configuration is also known as the poor man's computer. However, faster and more modern machines can simply be added to the existing cluster as and when they become available.

While CA are recent additions to the 'tools' used in structural analysis, increased use of CA as distributed computing becomes more widely available is envisaged, even though the CA rules are at this stage not transferable between different problems or even between meshes of varying refinement for a given problem.



# Opsomming

- Titel:** Sellulêre Outomata as 'n Benaderingsmetode in Struktuuranalise
- Outeur:** Michael Philip Hindley
- Leier:** Prof. A.A. Groenwold
- Departement:** Departement van Meganiese Ingenieurswese
- Graad:** Meester van Ingenieurswese
- Sleutelwoorde:** Sellulêre outomata, struktuuranalise, eindige verskil metode, eindige element metode, rand element metode

Hierdie verhandeling is gemoed met die numeriese idealiseringsmetode genoem sellulêre outomata (SO), en die moontlike toepassings van hierdie metode in struktuurmeganika. SO is 'n metode waarin alle fisiese parameters soos tyd en ruimte met diskrete waardes benader word. Die diskrete aard van SO laat eenvoudige interaksies met rekenaars toe, terwyl fisiese verskynsels wat hoofsaaklik diskreet van aard is, op 'n realistiese wyse benader kan word. Die toepassing van so 'n nuwe metode in struktuuranalise laat uiteraard nuwe moontlikhede toe.

In hierdie studie word die fundamentele werking van die SO bestudeer om te bepaal hoe die parameters van die metode benader kan word, om dan ook toegepas te kan word op die gevestigde gebied van struktuuranalise. Aandag word verder ook gegee aan die onderliggende wiskundige begrippe van struktuurmeganika, asook benaderingsmetodes wat reeds gevestig is in struktuuranalise, soos byvoorbeeld die eindige element metode (EEM) en die rand element metode (REM).

In struktuursimulasies met behulp van die SO wat ontwikkel is, word masjien-onderrig met behulp van 'n genetiese algoritme (GA) gebruik om die optimale reëls vir die SO te bepaal. Die basis vir die leerproses is eindige element modelle, rand element modelle, asook analitiese benaderings.

Simmetriese probleme in struktuuranalise word op 'n onkonvensionele manier geanaliseer met behulp van onsimmetriese reëls in die masjienonderrig proses, terwyl die simmetrie van die voorspelde oplossings gebruik word as 'n kwalitatiewe aanduiding van die geskiktheid



van die reëls. Daar word aangetoon dat die kwaliteit van onsimmetriese reëls beter is as die kwaliteit van simmetriese reëls, selfs vir die probleme wat inherent simmetries is.

Laastens, deur gebruik te maak van die inherente parallelisme van die SO, word aangetoon dat verspreide berekenings die effektiwiteit van SO-berekenings kan verhoog, alhoewel die versnellingsfaktor nie noodwendig eweredig aan die aantal onderverdelings in die struktuur is nie.

Die verspreide rekenaar is saamgestel deur 18 verouderde Pentium-rekenaars in 'n enkele berekeningsgroep saam te stel. In terme van SVE-prestasie is die verspreide rekenaar nie op die voorgrond van tegnologie nie, maar is saamgestel sonder enige hardeware koste. Die sagteware wat gebruik is om die berekeningsgroep saam te stel is in die publieke domein, en is ook kosteloos beskikbaar. So 'n parallelle rekenaar staan ook bekend as die "arm man se parallellerekenaar". Vinniger en beter rekenaars kan egter eenvoudig tot die berekeningsgroep toegevoeg word wanneer hulle beskikbaar is.

Terwyl SO 'n onlangse stuk 'gereedskap' is wat in struktuuranalise gebruik word, word verwag dat SO toenemend gebruik sal word soos verspreide berekenings meer algemeen beskikbaar raak, nieteenstaande die feit dat die reëls op hierdie stadium nie oordraagbaar is tussen verskillende probleme nie, en ook nie vir verskillende grade van maasverfyning vir 'n bepaalde probleem nie.



# Acknowledgments

I would like to express my sincere gratitude towards the following persons:

- Prof. Albert A. Groenwold, my supervisor, for his guidance and support throughout this study.
- To both my parents for their support, and for granting me the opportunity to study. I would also like to thank them for their financial backing.
- To my fellow students for turning a hard working environment into a pleasurable social experience.
- To every person involved with development of GNU (GPL) software. All the work performed in this thesis, including programming, editing and writing of this thesis, was accomplished using free software granted under the GNU public license.

I would like to dedicate this thesis to Laura Vatta.



# Contents

Abstract	ii
Opsomming	iv
Acknowledgments	vi
List of Figures	xiv
List of Tables	xv
List of Abbreviations	xvi
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objectives . . . . .	3
1.3 Approach . . . . .	4
<b>2 Introduction to cellular automata</b>	<b>5</b>
2.1 Informal definition . . . . .	5
2.1.1 Simple example . . . . .	6
2.2 Formal definition . . . . .	6
2.3 Neighborhood definition and size in 2-D . . . . .	7
2.4 Boundary conditions . . . . .	8
2.5 Initial conditions . . . . .	9
2.6 Geometry . . . . .	9
2.6.1 One dimension . . . . .	10
2.6.2 Two dimensions . . . . .	10
2.6.3 Three dimensions . . . . .	14
2.7 Cell state and precision . . . . .	14

2.8	Cell rule . . . . .	15
2.9	Cellular automata classification . . . . .	16
2.10	CA internal representation possibilities . . . . .	18
2.11	Partial differential equations, continuous systems and cellular automata . . .	19
<b>3</b>	<b>Cellular automata in structural analysis</b>	<b>22</b>
3.1	Introduction . . . . .	22
3.1.1	CA parameters in structural analysis . . . . .	23
3.1.2	CA implementation . . . . .	25
3.2	Problem formulation . . . . .	25
3.2.1	Problem descriptions . . . . .	25
3.2.2	Neighborhood descriptions . . . . .	26
3.2.3	Analysis criteria . . . . .	27
3.3	Determination of the optimum cell rule . . . . .	30
3.3.1	Motivation . . . . .	30
3.3.2	Cell rule comparison definition . . . . .	31
3.4	Problem analysis and results . . . . .	32
3.4.1	Extended neighborhood . . . . .	32
3.4.2	Moore neighborhood . . . . .	37
3.5	Cellular automata computational conclusions . . . . .	52
<b>4</b>	<b>Parallel processing implementation</b>	<b>53</b>
4.1	Introduction . . . . .	53
4.1.1	Parallel computing concepts . . . . .	53
4.1.2	Technology . . . . .	54
4.1.3	Classification of parallel machines . . . . .	54
4.1.4	Interconnecting structures and graphs . . . . .	55
4.1.5	Parallel computing models and complexity measures . . . . .	55
4.2	Parallel virtual machine, linux xpvm . . . . .	56
4.2.1	Linux . . . . .	56
4.2.2	Parallel virtual machine . . . . .	57
4.2.3	Xpvm . . . . .	59
4.3	Parallelization of cellular automata . . . . .	62
4.3.1	Parallelization method . . . . .	62
4.3.2	Sequential sub-lattice update method . . . . .	64
4.3.3	Parallel lattice computation . . . . .	66



4.4	Parallelization of a genetic algorithm . . . . .	70
4.4.1	Introduction . . . . .	70
4.4.2	Implementation method . . . . .	70
4.5	Computational conclusions . . . . .	72
<b>5</b>	<b>Concluding Remarks</b>	<b>73</b>
5.1	Summary of contributions . . . . .	73
5.2	Conclusions . . . . .	74
5.3	Directions for future work . . . . .	74
<b>A</b>	<b>Established structural approximation methods</b>	<b>80</b>
A.1	Governing equations . . . . .	80
A.1.1	One dimension . . . . .	82
A.1.2	Two dimensions . . . . .	82
A.1.3	Mechanics of the boundary value problem . . . . .	82
A.2	Finite difference method . . . . .	86
A.2.1	Introduction . . . . .	86
A.2.2	Methodology . . . . .	86
A.2.3	Methods of successive approximations . . . . .	89
A.2.4	Practicality for structural analysis . . . . .	91
A.3	Finite element method . . . . .	93
A.3.1	Introduction . . . . .	93
A.3.2	Methodology . . . . .	94
A.3.3	Implementation . . . . .	101
A.4	Boundary element method . . . . .	102
A.4.1	Introduction . . . . .	102
A.4.2	Basic principles . . . . .	103
A.4.3	Fundamental solution . . . . .	105
A.4.4	Boundary integral formulation . . . . .	106
A.4.5	Boundary element formulation . . . . .	108
A.4.6	Implementation . . . . .	109
A.5	Finite element method and boundary element method comparison . . . . .	115
<b>B</b>	<b>Numerical optimization</b>	<b>117</b>
B.1	Optimization formulation . . . . .	117
B.1.1	Mathematical definition . . . . .	117



# List of Figures

2.1	Various CA neighborhood definitions . . . . .	8
2.2	Shear mapping of the hexagonal lattice to the square lattice . . . . .	11
2.3	Mapping of the nearest neighbor in the shear mapping . . . . .	11
2.4	Shift mapping of the hexagonal lattice to the square lattice . . . . .	11
2.5	Mapping of the nearest neighbors in the shift mapping . . . . .	12
2.6	Mapping of the triangular lattice to the square lattice . . . . .	13
2.7	Mapping of the nearest neighbors in the triangular mapping . . . . .	13
2.8	Mapping of two triangular cells into one square cell . . . . .	13
3.1	One-dimensional bar problem . . . . .	24
3.2	One-dimensional computational cost comparison . . . . .	24
3.3	Formulation of CA problem 1 investigated . . . . .	26
3.4	Formulation of CA problem 2 investigated . . . . .	26
3.5	Formulation of CA problem 3 investigated . . . . .	26
3.6	Extended neighborhood description . . . . .	27
3.7	Moore neighborhood description . . . . .	27
3.8	Extended neighborhood RMS error comparison . . . . .	34
3.9	Extended neighborhood RMS error closer comparison . . . . .	34
3.10	Extended neighborhood computational cost comparison . . . . .	35
3.11	Problem 1 FEM $8 \times 8$ stress plot . . . . .	36
3.12	Problem 1 CA optimized 32 cell states stress plot . . . . .	36
3.13	Problem 1 FDM 4 neighbors stress plot . . . . .	36
3.14	Problem 1 CA optimized 1024 cell states stress plot . . . . .	36
3.15	Problem 1 CA optimized 32768 cell states stress plot . . . . .	36
3.16	Problem 1 Extended FDM stress plot . . . . .	36
3.17	Cell state accuracy round off occurrence . . . . .	38
3.18	Problem 1 FEM $16 \times 16$ stress plot . . . . .	39
3.19	Problem 1 FDM 4 neighbors stress plot . . . . .	39



3.20 Problem 1 CA optimized 32 cell states stress plot . . . . .	39
3.21 Problem 1 CA optimized 256 cell states stress plot . . . . .	39
3.22 Problem 1 CA optimized 256 cell states stress plot . . . . .	39
3.23 Problem 1 CA optimized 32768 cell states stress plot . . . . .	39
3.24 Problem 2 BEM 16 × 16 stress plot . . . . .	41
3.25 Problem 2 FDM 4 neighbors stress plot . . . . .	41
3.26 Problem 2 CA optimized 32 cell states stress plot . . . . .	41
3.27 Problem 2 CA optimized 256 cell states stress plot . . . . .	41
3.28 Problem 2 CA optimized 2048 cell states stress plot . . . . .	41
3.29 Problem 2 CA optimized 32768 cell states stress plot . . . . .	41
3.30 Problem 2 BEM 8 × 8 stress plot . . . . .	42
3.31 Problem 2 BEM 16 × 16 stress plot . . . . .	42
3.32 Problem 2 FEM 8 × 8 stress plot . . . . .	42
3.33 Problem 2 FEM 16 × 16 stress plot . . . . .	42
3.34 Problem 3 FEM 8 × 8 stress plot . . . . .	44
3.35 Problem 3 CA optimized 32 cell states stress plot . . . . .	44
3.36 Problem 3 CA optimized 64 cell states stress plot . . . . .	44
3.37 Problem 3 CA optimized 512 cell states stress plot . . . . .	44
3.38 Problem 3 CA optimized 1024 cell states stress plot . . . . .	44
3.39 Problem 3 CA optimized 32768 cell states stress plot . . . . .	44
3.40 Problem 3 FEM 16 × 16 stress plot . . . . .	46
3.41 Problem 3 CA optimized 32 cell states stress plot . . . . .	46
3.42 Problem 3 CA optimized 128 cell states stress plot . . . . .	46
3.43 Problem 3 CA optimized 512 cell states stress plot . . . . .	46
3.44 Problem 3 CA optimized 2048 cell states stress plot . . . . .	46
3.45 Problem 3 CA optimized 32768 cell states stress plot . . . . .	46
3.46 Problem 3 FEM 16 × 16 stress plot . . . . .	48
3.47 Problem 3 CA optimized 64 cell states stress plot . . . . .	48
3.48 Problem 3 CA optimized 128 cell states stress plot . . . . .	48
3.49 Problem 3 CA optimized 512 cell states stress plot . . . . .	48
3.50 Problem 3 CA optimized 8192 cell states stress plot . . . . .	48
3.51 Problem 3 CA optimized 32768 cell states stress plot . . . . .	48
3.52 Asymmetric neighborhood problem description . . . . .	49
3.53 Asymmetric rule orientation for Problem 3 FEM 8 × 8 1024 cell states . . . .	50
3.54 Asymmetric rule one problem 3 stress plot, FEM 8 × 8 1024 cell states . . .	51

3.55	Asymmetric rule two problem 3 stress plot, FEM $8 \times 8$ 1024 cell states . . .	51
3.56	Asymmetric rule three problem 3 stress plot, FEM $8 \times 8$ 1024 cell states . .	51
3.57	Asymmetric rule four problem 3 stress plot, FEM $8 \times 8$ 1024 cell states . . .	51
3.58	Asymmetric rule five problem 3 stress plot, FEM $8 \times 8$ 1024 cell states . . .	51
3.59	Asymmetric rule six problem 3 stress plot, FEM $8 \times 8$ 1024 cell states . . . .	51
4.1	Photo of the 18 node Linux “Souper” Computer, named GOH, constructed in this study . . . . .	60
4.2	XPVM screen shot . . . . .	61
4.3	Original square lattice . . . . .	63
4.4	Four sub-lattice split . . . . .	63
4.5	Nine sub-lattice split . . . . .	63
4.6	Typical convergence for four sub-lattices . . . . .	70
4.7	GA calculation with nine population members . . . . .	71
A.1	The main coordinates, displacements, stresses and tractions in three dimensions	81
A.2	One-dimensional bar problem . . . . .	83
A.3	Equilibrium of an infinitesimal bar element . . . . .	83
A.4	Finite difference two-dimensional grid . . . . .	87
A.5	Finite difference torsion bar example cross section . . . . .	88
A.6	Typical ritz function used in the analysis of one-dimensional bar problem . .	94
A.7	Linear combination of the ritz functions . . . . .	94
A.8	Simple bar FEM mesh . . . . .	95
A.9	Simple membrane FEM element . . . . .	97
A.10	FEM element patch test . . . . .	102
A.11	The fundamental solution in two dimensions BEM . . . . .	107
A.12	Constant element solution BEM . . . . .	110
A.13	Corner point . . . . .	110
A.14	Simple boundary element problem . . . . .	112
A.15	Applying boundary conditions . . . . .	112
A.16	Applying complex load . . . . .	113
A.17	Constant deformation of a simple geometry . . . . .	113
A.18	Constant deformation of a refined mesh . . . . .	114
C.1	Boundary cell calculation . . . . .	124
C.2	CA simulation main . . . . .	125
C.3	Stress contour plot . . . . .	126

C.4 Genetic algorithm . . . . . 127

C.5 CA parallel implementation number one . . . . . 128

C.6 CA parallel implementation number two . . . . . 129

C.7 CA parallel implementation number three . . . . . 130

C.8 GA parallel implementation . . . . . 131

2.1 A simple one-dimensional, two state, cellular automata . . . . . 14

2.2 Number of possible combinations . . . . . 14

2.3 Possible particles . . . . . 15

2.4 Neighborhood size differs use for different mesh sizes . . . . . 15

2.5 Internal Computer Representation for a computerized CA mapped to a finite  
mechanical lattice . . . . . 15

3.1 Displacement state comparison of BEM and CA in a one dimensional FEM . . . . . 17

3.2 Optimized rules for 5 x 5 mesh on Pyram at frequency of 100 Hz and 1000 Hz . . . . . 21

3.3 Optimized rules for 16 x 16 mesh on Moore graphlet and FEM problem . . . . . 23

3.4 Optimized rules for 16 x 16 mesh on Moore graphlet and 5 x 5 BEM problem . . . . . 24

3.5 Optimized rules for 8 x 8 mesh on Moore graphlet and FEM problem . . . . . 25

3.6 Optimized rules for 16 x 16 mesh on Moore graphlet and FEM Problem . . . . . 26

3.7 Optimized rules for 16 x 16 spectral moment state and internal displacement  
FEM Problem 2 . . . . . 27

3.8 Optimized rules 5 x 8 FEM problem 1 to 2000 Hz . . . . . 28

4.1 Sequential block update method with GPU and state update of 100 Hz  
rule (N, Number of operations) . . . . . 30

4.2 Local convergence speed large system for 100 Hz BEM . . . . . 31

4.3 Parallel convergence speed performance of CA vs FEM . . . . . 32

5.1 Established finite difference computational rules . . . . . 33

5.2 Displacements calculated by FEM single element state . . . . . 34

5.3 Stresses Calculated by FEM single element state . . . . . 35

5.4 Internal stresses (BEM) . . . . . 36

5.5 Internal stresses refined mesh (BEM) . . . . . 37

5.6 Internal displacements refined mesh (BEM) . . . . . 38

6.1 Displacements calculated by BEM . . . . . 39

6.2 Stresses calculated by BEM . . . . . 40

6.3 Displacements calculated by FEM . . . . . 41

6.4 Stresses calculated by FEM . . . . . 42

# List of Tables

2.1	A simple one-dimensional, two state, cellular automata . . . . .	6
2.2	Number of possible combinations . . . . .	14
2.3	Possible precision . . . . .	15
2.4	Neighborhood size difference for difference mesh types . . . . .	18
2.5	Internal Computer Representation for a complicated CA mapped to a square computational lattice . . . . .	19
3.1	Displacement error comparison of BEM and CA in comparison to FEM . . .	28
3.2	Optimized rules for $8 \times 8$ mesh on Extended neighborhood FEM problem 1 .	33
3.3	Optimized rules for $16 \times 16$ mesh on Moore neighborhood FEM problem 1 .	37
3.4	Optimized rules for $16 \times 16$ mesh on Moore neighborhood BEM problem 2 .	40
3.5	Optimized rules for $8 \times 8$ mesh on Moore neighborhood FEM problem 3 . .	43
3.6	Optimized rules for $16 \times 16$ mesh on Moore neighborhood FEM Problem 3 .	45
3.7	Optimized rules for $16 \times 16$ mesh for more cell states on a Moore neighborhood FEM Problem 3 . . . . .	47
3.8	Optimized rules $8 \times 8$ FEM problem 3 (asymmetric) . . . . .	49
4.1	Sequential block update method with 32768 cell states and finite difference rule(N, Number of sub-lattices) . . . . .	65
4.2	Local convergence speed improvement for 4 sub-lattices . . . . .	66
4.3	Parallel computing speed performance of CA with PVM . . . . .	69
A.1	Established finite difference computational molecules . . . . .	92
A.2	Displacements calculated by FEM (single membrane element) . . . . .	101
A.3	Stresses Calculated by FEM (single membrane element) . . . . .	102
A.4	Internal stresses (BEM) . . . . .	114
A.5	Internal stresses refined mesh (BEM) . . . . .	115
A.6	Internal displacements refined mesh (BEM) . . . . .	115



# List of Abbreviations

1-D	One dimensional
2-D	Two dimensional
3-D	Three dimensional
BEM	Boundary element method
CA	Cellular automata
CPU	Central processing unit
CSE	Computational science and engineering
FDM	Finite difference method
FEM	Finite element method
GA	Genetic algorithm
IPC	Inter-process communications
MIMD	Multiple instruction multiple data
MPP	Massive parallel processors
NFS	Networked file system
NOW	Network of workstations
OS	Operating system
PDE	Partial differential equation
PVM	Parallel virtual machine
RISC	Reduced instruction set computing
RMS	Root mean square
SIMD	Single instruction multiple data
VLSI	Very large scale integration



# Chapter 1

## Introduction

### 1.1 Motivation

The ultimate purpose of most scientific investigations is to determine how physical or other systems will behave in particular circumstances [1]. However, the adaptive capabilities of the brain and many other biological systems go far beyond those of any artificial systems thus far constructed by conventional engineering means [2]. Hence many phenomena can still not be studied using existing modeling capabilities.

In our never-ending quest to predict the reactions of physical systems, more and more methods are developed to achieve this goal. Much has been discovered about the nature of the components in physical and biological systems, but little is known about the mechanisms by which these components act together to give the overall complexity observed [3]. Nature provides many examples of systems whose basic components are simple, but whose overall behavior is extremely complex.

Mathematical models, such as cellular automata (CA) capture many essential features of such systems, and provide some understanding of the basic mechanisms by which complexity is produced [2]. It may be speculated that they are widespread in natural systems, perhaps occurring wherever nonlinearity is present [3]. To illustrate the above Stephan Wolfram makes use of the laws that govern the freezing of water and the conduction of heat. These laws have long been known, but analyzing their consequences for the intricate patterns of snowflake growth has not yet been possible.

While many complex systems may be broken down into identical components, each obeying simple laws, the huge number of components that make up the whole system act together to generate very complex behavior [4]. There is, however, extensive evidence that at a functional level, the basic components of such complex natural systems are quite simple, and could, for example, be emulated with a variety of technologies [2]. The complexity is generated by the cooperative effect of many simple identical components.

To discover and analyze the mathematical basis for the generation of complexity, simple mathematical systems that capture the essence of the process have to be identified [4]. However, as indicated above, it is not yet known how a large number of these components

act together to perform complex tasks. There seem to be general principles which govern such complex behavior, and allow this behavior to be molded to achieve particular goals. If these principles could be found and applied, they would initiate new forms of engineering solutions.

As an analysis tool, CA are sufficiently simple to allow detailed mathematical analysis, yet sufficiently complex to exhibit a wide variety of complicated phenomena [5]. While many natural systems do tend toward disorder, a large class of systems, biological ones being prime examples, show a reverse trend. In the latter case a disordered or structure-less initial state may spontaneously change to a structured state, in due course.

The development of digital computers during the last few decades has caused numerical calculations to become a discipline of increasing importance [6]. Today computers no longer are relatively expensive tools available to a very limited group of scientists, but have evolved into a tool available to almost every scientist and engineer. Consequently the interest in numerical calculations is now greater than ever. Conventional engineering or computer programming systems are built to achieve goals by following strict patterns, which specify the detailed behavior of each of their component parts [2]. The overall behavior of these systems must therefore be simple enough to ensure complete predictability.

As an example: Motion in many conventional mechanical engineering devices is simply constrained to be periodic. When simulating motion in conventional computer programming, each step consists of a single operation performed on small amounts of data elements. More complex behavior could be obtained from the basic components, whether mechanical or logical, but the principles necessary to make use of such behavior are not always known yet [2].

In structural analysis, researchers have suggested the “lattice analogy” since 1906 to solve continuum problems. The continuum is approximated by a regular mesh of elastic bars, and the method is based on well-known methods for the analysis of framed structures. More recently in structural mechanics and stress analysis, a very powerful numerical method known as the finite element method (FEM) appeared as an alternative to the finite difference method (FDM), which was the first widespread numerical method able to solve differential equations by dividing the domain of interest into elements [6]. One of the main advantages of FEM over FDM is its ability to handle elements of different sizes. FEM was developed at the same time as powerful computers. This fact combined with the universal “use-ability” of FEM contributed to its enormous success.

As a result FEM was for many years practically the only numerical method used in structural mechanics and today FEM is still the dominating numerical method in that area. FEM’s popularity is a consequence of the use of “piece-wise” interpolated polynomials as basic functions. Such functions are not infinitely smooth. Even their first derivatives are usually discontinuous. Piece-wise polynomials are effective for a number of reasons: they fulfill essential boundary conditions<sup>1</sup>, they provide a sparse resultant matrix and they can handle the complicated geometry of structures. Therefore, most commercial analysis systems are based on FEM. FEM has its drawbacks however, and the sources of inaccuracies are widely

---

<sup>1</sup>The physical constraints to movement without which the body will experience rigid body motion and be in a stress-free state.

discussed in the literature.

As opposed to FEM, the basic components of CA are discrete, but at least in some cases the aggregate behavior of large numbers of these components can be effectively continuous [7]. As a result, it is possible to use CA as models of continuum systems. This understanding may now be used to design systems whose complex behavior can be controlled and directed to particular tasks.

From complex scientific systems one must develop complex engineering systems. The laws governing the deformation of a structure are based on absorption of energy applied to the structure by the material, and one can now attempt to model this phenomenon using CA. The application of CA to structural mechanics will open an entire new direction of thought and with it new possibilities. A prerequisite for the application of a mathematical idealization like CA to continuous systems is a basic understanding of the manner in which the system reacts. Where no simple formula for the behavior of many natural systems can be given, the consequences of their evolution can only be found by direct simulation or observation.

Over the last decade, the role of computational simulations in all aspects of study in design in science and engineering has steadily increased. Algorithms and systems software are only two of the four main components required for the simulation and modeling technology for computational science and engineering (CSE) [8]. The advent of parallel computers has offered scalable high-performance engines to computational scientists and engineers. These may find applications in solving problems in shorter time periods and facing problems that were impossible to master by the use of sequential architectures. In order to make applications compatible for parallel machines, practitioners in this field are required to develop algorithms that are different from the ones previously used on sequential computers and to express their algorithms using new parallel languages and tools or parallel extensions to existing paradigms. In recent years there have been several attempts to provide more practical programming tools, environments for parallel programming and execution of CSE applications on parallel computer [9].

CA define a machine-independent parallel model of computation that can be used by various fields of computation. Despite the large amount of fields explored since the von Neumann definition of CA [10], researchers working in the field of CA have devoted their attention to the CA theory and abstract modeling of CA. By investigating the interaction of CA with parallel methods of computing it is possible to implement more complicated theories and models. This introduces a completely new fields of research exploration for the application of CA.

## 1.2 Objectives

The problem at hand is to approximate the mechanics of materials by CA.

To be able to effectively apply an idealization like CA to the field of structural mechanics, it is necessary to clearly understand the principles involved. It is thus required to fully understand the definition of CA and the inherent properties of CA.

The behavior of metallic materials on a microscopic level has been studied in great detail.

The behavior of such materials is believed to be understood to a reasonable extent and their properties are clearly described by various mathematical formulations. The definition of CA makes it very difficult to implement a purely mathematical description into a CA model. Integration of two fields with totally different approaches and description requires careful attention to detail, but also allows for new fields of exploration.

The inherent properties of CA such as the implicit parallelism make CA ideal for the use in current computing environments. CA are also capable of demonstrating systems with non-linearity and irreversibility. Hence it is desirable that these aspects are addressed in this study.

To summarize: This thesis aims to explore some of the possible methods of implementation of CA for obtaining approximations in structural analysis. Careful attention is to be given to the problems that are involved when using CA in structural mechanics. An exploration into the inherent parallelism in CA is desirable to obtain approximations with low, realistic computational effort.

### 1.3 Approach

Firstly, a perspective of all aspects involved is obtained by carefully investigating the details of CA construction and operation. Subsequently an overview is given of the mathematics describing structural mechanics and current methods that approximate the structural solution with discrete representations. These methods allow for better understanding of the governing aspects of the problem and how an approximate method like CA could be successfully applied in structural analysis.

Secondly, machine learning using a genetic algorithm is applied to the determination of optimum rules used in the CA, using finite element, boundary element and analytical approximations as the basis for machine learning. Cell rules are obtained for various problems and investigated for global validity and transferability.

Finally, the problems and benefits associated with parallel implementation of CA are investigated. To this extent, a parallel computer is constructed using inexpensive, available resources, and by combining them in a single computing cluster, called a Linux Beowulf 'Souper' computer. Using this infrastructure, the effects of symmetric versus asymmetric rules in the CA are studied.

## Chapter 2

# Introduction to cellular automata

### 2.1 Informal definition

Cellular automata (CA) are mathematical idealizations of physical systems in which physical quantities like space and time take on a finite set of discrete values [5].

Each point in a regular spatial lattice, called a cell, can have any one of a finite number of states. The states of the cells in the lattice are updated according to a local rule. That is, the state of a cell at any given time depends only on its own state in the preceding time step, and the states of its nearby neighbors at the previous time step. All cells on the lattice are updated synchronously. Thus the state of the entire lattice advances in discrete time steps [11]. CA provide a framework for a large class of discrete models with homogeneous interactions [12]. These are characterized by the following fundamental properties:

- They consist of a regular uniform discrete lattice (or “array”) of cells.
- The evolution takes place in discrete time steps.
- Each cell is characterized by a state taken from a finite set of states.
- Each cell evolves according to the same rule, which depends only on the state of the cell and a finite number of neighboring cells.
- The neighborhood relation is local and uniform.
- The variables at each site are updated simultaneously (synchronously), based on the values of the variables in their neighborhood at the preceding time step, and according to a definite set of “local rules”.

CA were originally introduced by von Neumann and Ulam (termed “cellular spaces”) as a possible idealization of biological systems with the particular purpose of modeling biological self-reproduction [5].

They have been applied and reintroduced for a wide variety of purposes, and referred to by a variety of names, which include:

- Tessellation automata
- Homogeneous structures
- Cellular structures
- Tessellation structures
- Iterative arrays

One of the most well-known CA rules, namely the “game of life” was conceived by Conway in the late 1960s [13, 14] and was shown to be computation-universal<sup>1</sup> [16].

CA have been used to study problems in number theory. They have also been applied to tapestry design [17, 18, 19, 20, 21].

### 2.1.1 Simple example

As an introduction to understanding CA, a simple one-dimensional CA with only two possible cell states (e.g., see [22]) is given.

Rule Table:								
Neighborhood:	111	110	101	100	011	010	001	000
Output bit:	1	1	1	0	1	0	0	0

Grid:															
$t = 0$	0	1	1	0	1	0	1	1	0	1	1	0	0	1	1
$t = 1$	1	1	1	1	0	1	1	1	1	1	1	0	0	1	1

Table 2.1: A simple one-dimensional, two state, cellular automata

Each cell has a possibility of two states, namely 0 and 1. The connectivity radius is  $r = 1$ , meaning that each cell has two neighbors, one to its left and one to its right. The grid size is 15, and spatially periodic boundary conditions are applied, viz. connectivity between cell 1 and cell 15 is assumed. The pseudo-time  $t$  is incremented in discrete steps of 1. Considering the first neighborhood and output bit, the cell in consideration has neighborhood 101 at  $t = 0$ , and hence, a state 1 at time  $t = 1$ . (For time  $t = 2$ , the state remains unchanged.)

## 2.2 Formal definition

We can now continue to give a mathematical description of CA with conventions:

- $d$  = dimension
- $k$  = states per cell

---

<sup>1</sup>They can generate arbitrarily complex patterns of symbols and process them [15].

- $r = \text{radius}$

Assume a  $d$ -dimensional cellular automaton with  $d = 1$  (for simplicity) takes as its underlying space the lattice  $S^Z$  ( $Z = \text{integers}$ , infinite in both positive and negative directions) where  $S$  is a finite set of  $k$  elements. The dynamics are determined by a global function.

$$F : S^Z \rightarrow S^Z \quad (2.1)$$

whose dynamics are determined “locally” as defined below. A “local (or neighborhood) function  $f$ ” is defined on a finite region.

$$f : S^{2r+1} \rightarrow S \quad (2.2)$$

The all-important property of CA is that this function is determined by a finite look-up table [11]. Both the domain and range of  $f$  are finite. The global function  $F$  arises from  $f$  by defining:

$$F(c)_i = f(c_{i-r}, \dots, c_{i+r}) \quad (2.3)$$

A concrete example with  $k = 2, r = 1$  would take a doubly infinite string of zeroes and ones to a new string at which each site is replaced by the logical and of its two neighbors [11]. Some relevant facts from a topological standpoint are:

- The base space  $S^Z$  is compact and the global function  $F$  is continuous. This makes CA an ideal meeting point between continuous dynamics and complexity theory, since they are discretely defined but exhibit continuous dynamics.
- The map  $F$  commutes with the shift operator which takes  $c_i$  to  $c_{i+1}$ .

Cellular automata are in fact characterized completely by these properties [11]. The transition to more dimensions is straightforward. The only difference is that the global function  $F$  is defined over  $S^Z$  (functions from a  $d$ -dimensional lattice to  $S$ ) and the local function  $f$  is determined by enumerating the image of all patches of size  $2^{(r+1)^d}$  [11].

## 2.3 Neighborhood definition and size in 2-D

Various definitions of neighborhoods are possible. Considering a two-dimensional lattice, the following definitions are common and are shown in Figure 2.1 [23]

- **von Neumann neighborhood** (Figure 2.1.a):  
The von Neumann neighborhood of a cell is defined as the cells directly above and below, and to the right and to the left of the cell in discussion. The radius  $r = 1$ , as only the next layer is considered.

- **Moore neighborhood** (Figure 2.1.b):  
The Moore neighborhood is an enlargement of the von Neumann neighborhood, and includes the diagonal cells. As for the von Neumann neighborhood, the radius  $r = 1$ .
- **Extended von Neumann neighborhood** (Figure 2.1.c):  
The extended von Neumann neighborhood is equivalent to the Moore neighborhood with the cells in the second layer above and below and to the left and to the right included. Thus the radius  $r = 2$ .
- **Extended Moore neighborhood** (Figure 2.1.d):  
The extended Moore neighborhood is equivalent to the description of the Moore neighborhood above, but the neighborhood reaches over the distance of the next adjacent cells. Hence the radius  $r = 2$  (or larger).
- **Arbitrary neighborhood** (Figure 2.1.e):  
Any other neighborhood can be defined as long as it is uniform<sup>2</sup> and finite.

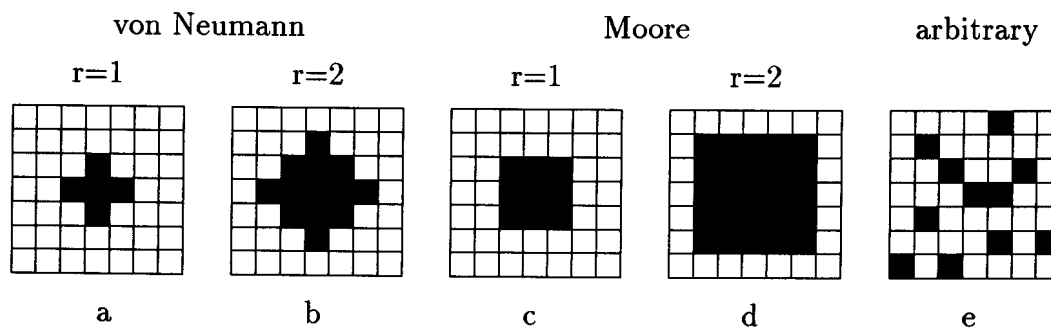


Figure 2.1: Various CA neighborhood definitions: (a) von Neuman neighborhood  $r = 1$ , (b) von Neuman neighborhood  $r = 2$ , (c) Moore neighborhood  $r = 1$ , (d) Moore neighborhood  $r = 2$ , (e) Arbitrary neighborhood

## 2.4 Boundary conditions

In the formal definition of CA, it is usually required that the lattice is infinite in all dimensions. For considerations of computability and complexity, this is reasonable and necessary. Since it is impossible to simulate a truly infinite lattice on a computer (unless the active region always remains finite), some boundary conditions have to be prescribed. Another reason for using certain boundary conditions is the natural boundary conditions of the problem under consideration.

Three kinds of boundaries will be considered. These are periodic, reflective and fixed value boundaries.

<sup>2</sup>The same neighborhood is used for each cell in consideration.



- Periodic boundaries are obtained by periodically extending the lattice. The one-dimensional version can be seen in Table 2.1. It is only necessary to store only the original array and a boundary of width  $r$ , where  $r$  is the radius of the neighborhood of the CA. These periodic boundary conditions come closest to simulating an infinite lattice, and are therefore often used [12].

The two-dimensional version of periodic boundaries, where the top and bottom are connected, and the left and right edges are connected, leads to the topology of a torus (it is not possible to obtain the topology of a sphere with a regular lattice of arbitrary size).

- Reflective boundary conditions are obtained by reflecting the lattice at the boundary, which means that in one dimension the end cell sees a copy of itself as the boundary. This type of boundary is best suited for boundaries where the system to be simulated also has a boundary, and where the value of the physical variables is not fixed (e.g. von Neumann, zero-flux boundary conditions in a diffusive system).
- Fixed value boundary conditions are obtained by simply prescribing a fixed value for the cells on the boundary. This value must be determined from application.

All three boundary conditions can be combined, so that different boundaries can have different conditions. If one edge has periodic boundary conditions, the opposite edge must also have periodic conditions. Often these boundary conditions can be usefully combined with one another. To simulate a long channel, periodic boundaries in the horizontal, and reflective boundaries in the vertical direction have to be used.

## 2.5 Initial conditions

Not only do the boundary conditions need to be specified, but the initial condition must also be specified. In most cases the initial condition greatly influences the subsequent evolution. Initial conditions can either be constructed, or they can be generated randomly.

An important consideration in the generation of initial conditions is that many CA rules conserve some quantities. This can be e.g., the total number of particles, the total momentum, or energy. It can also be some conserved quantities in the system to be modeled, e.g. the number of particles in one row or one column. In generating initial conditions, care must be taken that the intended value for the conserved quantities is reached (especially with random initial conditions), and that the conserved quantities do not generate undesired effects.

## 2.6 Geometry

An important choice is the selection of a specific lattice geometry. The definition of CA requires the lattice to be regular [12]. The different possibilities considered are for one, two, and three dimensions. Different geometries are constructed to establish the influence they may have on the specific problem.

### 2.6.1 One dimension

For one-dimensional automata, there is only one possibility: a linear array of cells as shown in the example in Section 2.1.1. The only variations that are possible are the number of neighbors to the left and right that are considered.

### 2.6.2 Two dimensions

In two dimensions there are three common regular lattices, namely triangular, square, and hexagonal lattices.

The main characteristics of the three lattices are [12]:

- **Triangular lattice:**
  - *Advantage:* Small number of nearest neighbors (three).
  - *Disadvantage:* Difficult to represent and visualize, since it must be mapped to square arrays and display pixels. The mapping is described in the following subsection.
- **Square lattice:**
  - *Advantage:* Simple representation using square arrays and simple visualization.
  - *Disadvantage:* In some cases the square lattice has insufficient isotropy<sup>3</sup>.
- **Hexagonal lattice:**
  - *Advantage:* The hexagonal lattice has the lowest anisotropy of all regular two-dimensional lattices. Often this lower anisotropy makes simulations appear more natural, and in some cases it is absolutely necessary to model the phenomena correctly.
  - *Disadvantage:* More difficult to represent and visualize, since it must be mapped to a square lattice. The mapping is described in the following section.

#### Mapping of hexagonal and triangular lattice to a square lattice

There are two possible mappings of a hexagonal lattice to a square lattice. The first is simple shear, as illustrated in Figure 2.2.

The cells are filled with random colors to show the correspondence. Mathematically the transformation can be described as follows. The cells of the hexagonal lattice have their centers at locations as described in (2.4)

$$\left( i + \frac{(j - 2 \cdot \text{floor}(\frac{j}{2}))}{2}, j \sqrt{\frac{3}{2}} \right) \text{ with } i, j \text{ in } Z \quad (2.4)$$

---

<sup>3</sup>The same properties in all directions.

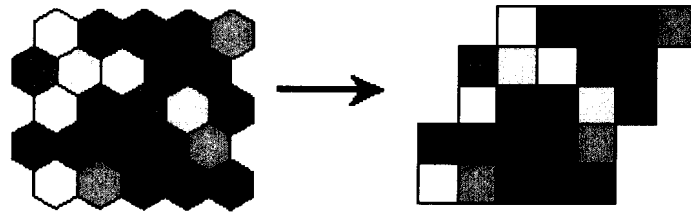


Figure 2.2: Shear mapping of the hexagonal lattice to the square lattice

where  $\text{floor}(x)$  denotes the largest integer smaller or equal to  $x$  and where the distance between cells is set to unity. In the shear mapping, the cell number  $(i, j)$  is mapped to position

$$\text{shear}(i, j) = \left( i + \text{floor}\frac{j}{2}, j \right) \quad (2.5)$$

The nearest neighbor relations are transformed as shown in Figure 2.3.

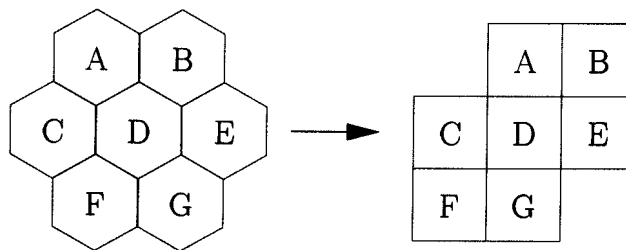


Figure 2.3: Mapping of the nearest neighbor in the shear mapping

The second possible mapping is to shift alternate rows in opposite directions, as shown in Figure 2.4. This mapping is represented by the formula

$$\text{shift}(i, j) = (i, j) \quad (2.6)$$

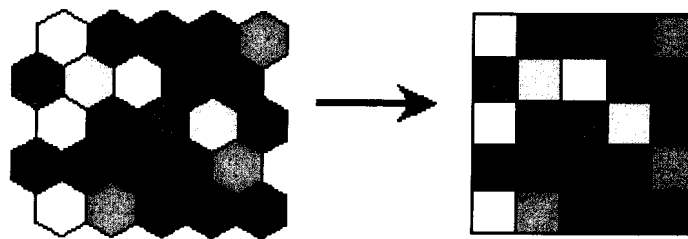


Figure 2.4: Shift mapping of the hexagonal lattice to the square lattice (the cells are filled with random colors to indicate the correspondence)

In this second (shift) mapping, the neighborhood relation is transformed differently depending on whether the row index  $j$  is even or odd, as shown in Figure 2.5.

The two mappings each have advantages and disadvantages.

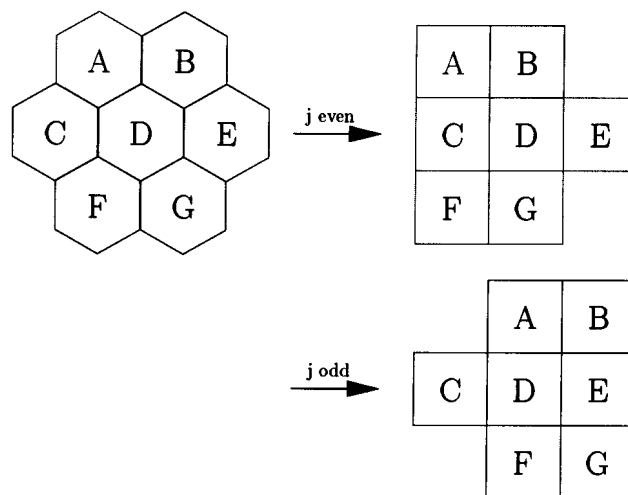


Figure 2.5: Mapping of the nearest neighbors in the shift mapping ( $j$  is the row index)

- Shear mapping
  - *Advantage:* Shear mapping has the advantage that the local neighborhood remains uniform in the square representation.
  - *Disadvantage:* It has the disadvantage that boundary conditions are more difficult to implement, and that the transformation has to be reversed for visualization to avoid presenting a sheared image.
- Shift mapping
  - *Advantage:* The shift mapping has the advantage that boundary conditions are as simple to implement as for the square lattice (in fact, the same boundary conditions can be used), and that visualization can also be very simple.
  - *Disadvantage:* The shift mapping has the disadvantage that the neighborhood now depends on the parity of  $j$ . This contradicts the definition of CA, where the neighborhood is homogeneous and the rules are homogeneous, too.

We can nevertheless turn the mapped automaton into a real CA by extending the neighborhood to eight cells, which covers the two possible neighborhoods for even and odd rows. One bit is added to the state of the CA to indicate whether the cell lies on an even or an odd row. The rules are modified accordingly and initial conditions are specified so that the indicator bit is correctly set. The rules do not modify this part of the state, it is only used to select the six relevant neighbors out of the total eight neighbors. Thus the mapped CA has uniform rules and neighborhoods again.

The mapping of the triangular automaton to the square lattice is similar to the shear mapping for the hexagonal lattice. In the triangular case, every second cell has a different orientation. The mapping is shown in Figure 2.6.

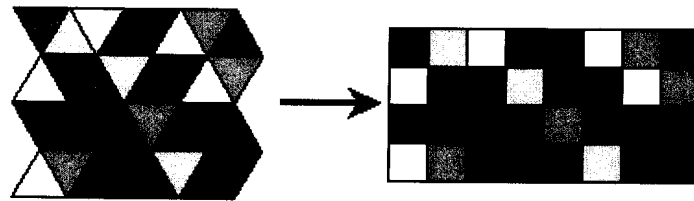


Figure 2.6: Mapping of the triangular lattice to the square lattice (the cells are filled with random colors to show the correspondence)

Each row of triangles is simply mapped to one row of squares. This leads to a non-uniform neighborhood depending, on the parity of the cell in a checkerboard coloring (parity of  $i+j$ ). The neighborhood is shown in Figure 2.7.

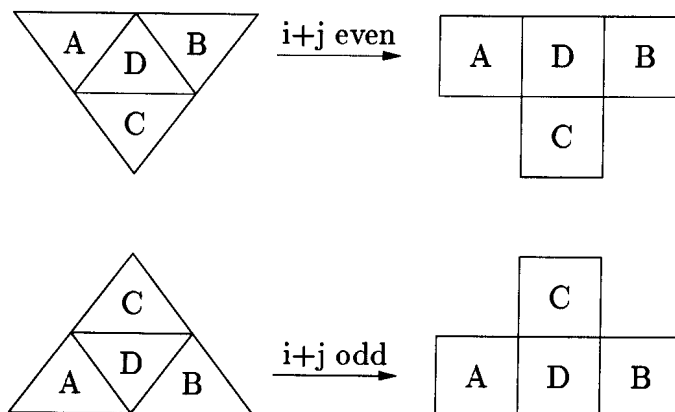


Figure 2.7: Mapping of the nearest neighbors in the triangular mapping ( $i$  is the column index and  $j$  is the row index)

A second, and somewhat surprising method is to map two triangular cells into one square cell (See Figure 2.8).

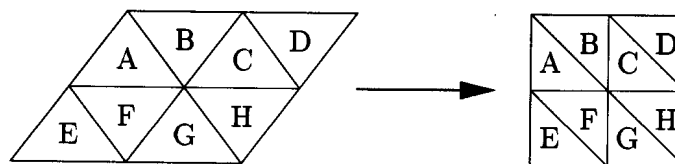


Figure 2.8: Mapping of two triangular cells into one square cell

It is clear that the state of the square cell has to be extended to  $S \times S$  to store the states of both triangular cells. The nearest neighbors of the two triangular cells are just the nearest neighbors of one square cell. This method has the advantage of a uniform transition rule and neighborhood, and the disadvantage of a larger state set. Consequently the transition table is larger for the compound cells [12].

### 2.6.3 Three dimensions

In three dimensions there are many possible regular lattices. One difficulty with three-dimensional CA is the graphical representation (on two-dimensional paper or screen) of the state [12]. In this study is restricted to one- and two-dimensional cases.

## 2.7 Cell state and precision

According to the definition of CA, each cell is a finite automaton, and therefore the set of states has a finite size [12]. In addition, the set is usually fairly small. There are several reasons for this. Previous researchers have investigated all possible combinations of CA with a given number of states and a given neighborhood size. This is only possible when the set of states is very small, since the number of possible CA combinations with  $s$  states and  $n$  cells in the neighborhood (including the cell to be updated), is  $(s^n)^n$ . This number rapidly becomes too big to explore all possibilities as  $n$  and  $s$  grow:

$s$	$n$	$s^n$	$(s^n)^n$
2	3	8	64
2	5	32	1024
5	3	125	15625

Table 2.2: Number of possible combinations

The second reason for the preference of small state sets, is that it is possible to specify the CA rules explicitly when there are few states and store them in a table (the table size is  $s^n$ ).

A reason for using large numbers of states is that CA can better approximate a continuous system with more states. This is clear when considering the representation of  $n$  particles in CA. When using CA with one bit per cell,  $O(n)$  cells are required. When particles are placed randomly in these cells, the fluctuations in particle number in a subset of cells is  $O(\sqrt{n})$ . Therefore the precision is  $O(\frac{\sqrt{n}}{n})$ . Conversely, with  $k$  bits per cell,  $2^k - 1$  particles can be represented in each cell. Therefore only  $O(\frac{n}{2^k - 1})$  cells are needed, and the fluctuations are  $O(\sqrt{\frac{n}{2^k - 1}})$ . Thus the precision is  $O(\frac{\sqrt{\frac{n}{2^k - 1}}}{n})$  for different choices of  $k$  and  $n$  [12].

It is clear from the above that low precision is obtained when using few bits. But modeling with high precision, when the equations are exactly known, is just the domain of numerical analysis and finite element or finite volume methods [12]. CA's are stronger for lower precision, and in situations where the fluctuations are important, or where the interactions in the system are not exactly known. They are more easily described using simple CA rules, rather than partial differential equations.

Even though a simulation using floating-point variables still uses a finite set of states (about  $2^{32}$  states for a single-precision floating point variable), such models are strictly speaking not a CA model. Such programs are usually a computer approximation of models using real, i.e., continuous numbers, and the discretization is poorly controlled. In models with

Number of particles	Bits per cell	Number of cells	Total number of bits	Precision $O$
$n$	$k$	$\frac{n}{2^k-1}$	$k\frac{n}{2^k-1}$	$\frac{\sqrt{\frac{n}{2^k-1}}}{n}$
$10^{12}$	1	$10^{12}$	$10^{12}$	$10^{-6}$
$10^6$	1	1000000	1000000	$1 \times 10^{-3}$
$10^6$	2	333334	666668	$5.8 \times 10^{-3}$
$10^6$	3	142858	428574	$3.8 \times 10^{-3}$
$10^6$	5	32259	161295	$1.8 \times 10^{-3}$
$10^6$	10	978	9780	$3.1 \times 10^{-4}$
$10^6$	15	31	465	$5.6 \times 10^{-6}$
$10^6$	20	1	20	$10^{-6}$

Table 2.3: Possible precision

real variables the names *coupled map lattice* [24, 25] or *finite difference method* might be appropriate for partial differential equation (PDE's).

## 2.8 Cell rule

Up to this point cells arranged in a lattice represented a static state. Rules have to be integrated into these systems in order to add a dynamical dimension. The function rules is to define the state of the cells for *the next discrete time step*.

In CA a rule defines the state of a cell, which depends on the state of the neighborhood of the cell. The most important aspect of a cellular automaton is the transition rule or transition function. The transition rule depends on the lattice geometry, the neighborhood of cells, and the set cells states. Even though the transition rule determines the evolution, it is in many cases not possible to predict this evolution by other means than explicit simulation (one can prove that this is the case for those automata that can be shown to be computationally universal, such as simple automata like the “game of life”). The most direct specification is to establish the outcome of the transition rule for each possible state configuration in the neighborhood. An important property of CA is that a small change in the transition rule can have dramatic consequences.

Many CA rules merely require information about the number of neighbors in a certain state. This can be used to obtain a simpler specification and also a smaller transition table for the implementation. In classical CA theory an automation rule is called *totalistic* if it depends only on the sum of the states of all cells in the neighborhood. The rule is called *outer totalistic* if it also depends on the state of the cell to be updated [12].

In many cases it is convenient, for the specification and the implementation to split the automation rule into several sub-steps. In theory however, the sub-steps can be combined into a table or rule for the whole step.

An important class of transition rules are *probabilistic* rules. Functions of the transition rule do not have exactly one result for each neighborhood configuration. One or more possible

outcomes are possible with associated probabilities. The sum of probabilities of all outcomes must be one for each input configuration. The probabilistic choices of all cells are independent of one another (uncorrelated) [12].

The introduction of probabilistic rules is very important in many modeling situations, since many natural systems are noisy. In some cases it is sufficient to introduce random initial conditions, which also lead to a noisy or random behavior. In many cases probabilistic rules have advantages [12].

The possibility of such self-organization is therefore a consequence of the irreversibility of the cellular automaton evolution, and the structures obtained through self-organization are determined by the characteristics of the attractors [4].

## 2.9 Cellular automata classification

According to Wolfram [4], the final results obtained by CA simulations can be divided into four classes. These basic classes of behavior may be characterized empirically as follows:

- **Class 1** evolution leads to a homogeneous state in which, for example, all sites have value 0.
- **Class 2** evolution leads to a set of stable or periodic structures that are separated and simple.
- **Class 3** evolution leads to a chaotic pattern.
- **Class 4** evolution leads to complex structures, sometimes long-lived [4].

The different classes of cellular automaton behavior allow different levels of prediction of the outcome of cellular automaton evolution from particular initial states [26].

The existence of only four qualitative classes implies considerable universality in the behavior of CA. Many features of CA depend only on the class in which they lie and not on the precise details of their evolution [4]. Such universality is analogous, though probably not mathematically related, to the universality found in the equilibrium statistical mechanics of critical phenomena. In this case many systems with quite different detailed construction are found to lie in classes with critical exponents that depend only on general, primarily geometrical features of the systems and not on their detailed construction.

To proceed in analyzing universality in CA, a more quantitative definitions of the classes identified above must first be given. One approach to such definitions is to consider the degree of predictability of the outcome of cellular automaton evolution in conjunction with given knowledge of the initial state.

For class 1 CA complete prediction is trivial regardless of the initial state, since the system always evolves to a unique homogeneous state [4]. Such CA may be considered to evolve to simple “limit points” in phase space. Their evolution completely destroys any information on the initial state. Some exceptional configurations in finite class 1 CA may not evolve to a homogeneous state, but may in fact enter non-trivial cycles [26].



Class 2 CA have the feature that the effects of particular site values propagate only a finite distance, that is, only to a finite number of neighboring sites. Thus a change in the value of a single initial site affects only a finite region of sites around it, even after an infinite number of time steps [4]. The simple structures generated by class 2 CA are either stable, or are periodic, typically with small periods. The set of persistent structures generated by a given class 2 cellular automaton is typically quite simple [26].

Class 2 CA may be considered as “filters” that select particular features of the initial state. A class 2 CA may for example be constructed in which initial sequences 111 survive, but sites not in such sequences eventually attain value 0. Such CA are of practical importance for digital image processing: they may be used to select and enhance particular patterns of pixels. After a sufficiently long time any class 2 cellular automaton evolves to a state consisting of blocks containing nonzero sites separated by regions of zero sites. The blocks have a simple form, typically consisting of repetitions of particular site values or sequences of site values (such as 101010). The blocks either do not change with time or cycle between a few states.

In class 3 CA, the prediction of the value of a site at infinite time would require knowledge of an infinite number of initial site values. Class 3 CA exhibit chaotic *aperiodic behavior*<sup>4</sup>. Although chaotic, the patterns generated by class 3 CA are not completely random. In fact, they may exhibit important self-organizing behavior. In contrast to class 2 CA, the statistical properties of the states generated by many time steps of class 3 cellular automaton evolution are the same for almost all possible initial states. The large-time behavior of a class 3 cellular automaton is therefore determined by these common statistical properties [4].

Class 4 CA are distinguished by an even greater degree of unpredictability than class 3 [4]. For class 1, 2 and 3 CA, fluctuations in statistical quantities are typically found to become progressively smaller as larger numbers of sites are considered. Such systems therefore exhibit definite properties in the “infinite volume” limit. For class 4 CA, it seems likely that fluctuations do not decrease as larger number of sites are considered, and no simple smooth infinite volume limit exists. Important qualitative effects can arise from special sequences appearing with arbitrarily low probabilities in the initial state [26].

The complexity apparent in the behavior of class 4 CA suggests the conjecture that these systems may be capable of universal computation. The initial sequence may be considered as a program and data stored in computer memory, and part of the final sequence may be considered as the result of the computation. CA may be considered as computers, their initial configurations represent programs and initial data, and their configurations after a long time contain the results of computations [4]. A system is a universal computer if, given a suitable initial program, its time evolution can implement any finite algorithm [27].

However, if CA in the class are indeed capable of universal computation, then this dependence may be arbitrarily complex, and the behavior of the system can be found by no procedure significantly simpler than direct simulation. No meaningful prediction is therefore possible for such systems [26].

---

<sup>4</sup>Periodically returns to the same state.

The results from a single cell rule may indeed also hold promise for encryption of secure data. The evolving nature and invisibility of CA would make it impossible to encrypt the data without all the necessary CA variables.

The four classes of CA may be distinguished by the level of predictability of their “final” large time behavior given their initial state [26].

## 2.10 CA internal representation possibilities

CA as an idealization can be used for an approximation for the governing equations in structural analysis in a similar way as FDM. For structural analysis, the physical geometry has to be converted into a discrete mesh. For complex geometries it will not be possible to create a mesh with equally sized cells. In section 2.6 we discuss methods by which triangular and hexagonal physical meshes can be represented on a square CA computational map. A problem with this method is keeping track of the neighbors for each cell (indexing complications). As for the FDM mesh, different mesh sizes will complicate the calculation. The cells on the edges have fewer neighbors to keep track of than the cells in the middle. Table 2.4 shows how these neighbors can vary

Mesh type	Maximum neighbors	Minimum neighbors
Triangular	3	2
Square	8	3
Hexagonal	6	2

Table 2.4: Neighborhood size difference for difference mesh types

Thus as far as computational methods are concerned, it is only necessary to deal with a square lattice. As mentioned previously, the problem arises of keeping track of the neighbors in the physical geometry in the CA map. Since CA do not require the solution of systems of equations, an internal representation of the system under consideration may be used. Table 2.5 illustrates an example of data storage. The first columns store the physical coordinates of the cell center. The next set of columns represent the cell states. Each cell state corresponds to a physical quantity, (in structural analysis for example, both displacements and stresses may be stored), even though they evolve according to different cell rules, and could possibly be a function of different cell states. The following column contains the neighborhood flag, with each flag number corresponding to a specific number of neighbors and configuration. The last columns of Table 2.4 give the indices of the neighbors in accordance to the neighborhood flag.

Each column does not have to be in the same array. There must merely be corresponding indices. For instance, the first few columns that contain the coordinates require real values to be stored. For the remainder of the columns, an integer value is sufficient and would save memory.

The methodology mentioned above allows for the complicated expansion of the cell rule definition. Most CA simulations in existence have no need for a complicated representation

Physical coordinates		Cell States		Neighbor Flag	Neighbor Indexes	
$x_1$	$y_1 \dots$	State $l_1 \dots$	State $m_1$	Flag 1	Index $l_1 \dots$	Index $n_1$
$x_2$	$y_2 \dots$	State $l_2 \dots$	State $m_2$	Flag 2	Index $l_2 \dots$	Index $n_2$
$x_3$	$y_3 \dots$	State $l_3 \dots$	State $m_3$	Flag 3	Index $l_3 \dots$	Index $n_3$

Table 2.5: Internal Computer Representation for a complicated CA mapped to a square computational lattice

since they are idealized with simple geometries. This implies simple mapping and “neighbor look-up”. Now it is possible to not only represent for instance the displacements for each cell but also the stresses. The cell rules can be formed by using a combination of the cell states. One of the cell states can be represented by two possible states, one for existence and one for dead, where dead implies that the material of the cell is physically not present. If a cell dies, the program can easily change the neighborhood flag and indices of all its neighbors to ensure that the dead cell is not used for further computations. If the cell is not used for further computations it is not removed from the array the indices of each cell whether dead or alive remains unchanged. It is also possible to add cells to the bottom of the index in a similar fashion. Table 2.5 offers a method in which the CA can be handled such that the only significant problem lies with the initial setup of the cell map. The computational part of the problem is simplified and flexible and more attention can be given to the computational parameters.

## 2.11 Partial differential equations, continuous systems and cellular automata

The mathematical formulation of most problems in science involving rates of change with respect to two or more independent variables leads to Partial differential equations (PDEs) or a set of such equations [28]. Equation 2.7 shows a typical two dimensional second-order equation.

$$a \frac{\partial^2 \phi}{\partial x^2} + b \frac{\partial^2 \phi}{\partial x \partial y} + c \frac{\partial^2 \phi}{\partial y^2} + d \frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y} + f \phi + g = 0 \quad (2.7)$$

Where  $a, b, c, d, e, f$  and  $g$  are a function of independent variables  $x$  and  $y$  and the dependent variable  $\phi$ . By definition (2.7) is considered to be *elliptic* when  $b^2 - 4ac < 0$ , *parabolic* when  $b^2 - 4ac = 0$ , and *hyperbolic* when  $b^2 - 4ac > 0$ .

Several conditions are necessary for the overall behavior of a system with discrete elements to appear continuous. First, continuum behavior must be associated with some kind of extensive quantity. Such a quantity must be additive, and must be conserved in the dynamical evolution of the system. In a gas, one example of such a quantity is particle number. Other examples are energy and momentum [7].

CA rules may be compared with traditional approaches to emulating these continuum sys-

tems on digital computers. In the conventional approach, one starts from partial differential equations, then makes discrete numerical approximations to them. These approximations involve considering a discrete lattice of points. But unlike in CA, each of these lattice points carries a continuous variable which represents the precise value of a continuum quantity, such as particle density, at that point. In actual computer implementations, the continuous variable is represented by a floating-point number, say 64 bits in length. The number is updated in a sequence of time steps, according to a discrete rule. The rule in general involves arithmetic operations, which cannot be carried out precisely on the finite precision number. As a result, low-order bits of the number are truncated. Numerical analysis has studied in detail the propagation of such round-off errors, and has suggested schemes which minimize their effects. Instead of systematically truncating the numbers, their low-order bits are modified according to dynamics which yields effectively random behavior. The result is similar to random round-off, but includes a precise particle conservation law. By adjusting the number of unary and digital bits, one can determine the trade-offs between cellular automaton and numerical analysis approaches [7].

One of the major issues in numerical analysis is *convergence*. This is very difficult to prove for all but the simplest equations and the simplest schemes. But in CA, the analogue of convergence is the process of coming to equilibrium. Thus the problem of convergence is related to a fundamental problem of physics [7].

CA may thus be considered as discrete idealizations of the partial differential equations often used to describe natural systems [4]. Physical systems containing many discrete elements with local interactions are often conveniently modeled as CA [5]. Any physical system satisfying differential equations may be approximated as a cellular automaton by introducing finite differences and discrete variables [29]. The time evolution given particular initial conditions is represented by a trajectory in the space of variables described by the differential equations. In the simplest cases (such as those typical for chemical concentrations described by the Boltzmann transport equations), all trajectories tend at longer times to a small number of isolated limit points, or approach simple periodic limit cycle orbits [5].

While there is every evidence that the fundamental microscopic laws of physics are intrinsically reversible (information-preserving, though not precisely time-reversal invariant), many systems behave irreversibly on a macroscopic scale and are appropriately described by irreversible laws. For example, while the microscopic molecular interactions in a fluid are entirely reversible, macroscopic descriptions of the average velocity field in the fluid, using, say, the Navier-Stokes equations, are irreversible and contain dissipative terms. CA provide mathematical models at this macroscopic level [4].

The second law of thermodynamics implies that isolated microscopically reversible physical systems tend with time to states of maximal entropy and maximal disorder [5]. However, dissipative systems involving microscopic irreversibility, or those open to interactions with their environment, may evolve from disordered to more ordered states. In almost all cases, cellular automaton evolution is irreversible [26]. Every configuration in a cellular automaton has a unique successor in time. A configuration may however have several distinct predecessors [30]. The presence of multiple predecessors implies that the time evolution mapping is not invertible but is instead contractive. The CA thus exhibits irreversible behavior in which

information on initial states is lost through time evolution [30]. The existence of configurations with multiple predecessors implies that some configurations have no predecessors [5]. These configurations occur only as initial states, and may never be generated in the time evolution of the cellular automaton. They appear on the periphery of the state transition. Their presence is an inevitable consequence of irreversibility and of the finite number of states [30].

Trajectories in the configuration space for CA therefore merge with time, and after many time steps, trajectories starting from almost all initial states become concentrated onto attractors [26]. These attractors typically contain only a very small fraction of possible states. Evolution to attractors from arbitrary initial states allows for self-organizing behavior, in which structure may evolve at large times from structure-less initial states. The nature of the attractors determines the form and extent of such structures. Statistical mechanics, and the continuum equations derived from it, provide a considerably reduced description of these systems [7].

i 1160 3174x  
b 15431253

## Chapter 3

# Cellular automata in structural analysis

### 3.1 Introduction

In this chapter the CA parameters in structural analysis and the CA implementation for structural analysis are described. The problems to be solved using CA are formulated and the method of determining the optimum cell rule is discussed. Finally example problems are analyzed and the results compared to current established methods in structural analysis.

The question whether CA can model not only general phenomenological aspect of our world, but also model the laws of physics themselves was raised by Toffoli [31]. A primary theme of this research is the computational models of physics that are *information-preserving*, and thus retain one of the most fundamental features of microscopic physics namely reversibility [32]. CA have been used to provide extremely simple models of common differential equations of physics, such as the heat and wave equations [33] and the Navier-Stokes equation [34, 35]. CA also provide a useful discrete model for a branch of dynamical systems theory which studies the emergence of well-characterized collective phenomena such as ordering, turbulence, chaos, etc. [36, 37]. Systematic research of CA in this context was pioneered by Wolfram and studied extensively by him, identifying four qualitative classes of CA behavior as discussed in Section 2.9.

In Appendix A a brief summary of the governing equations in structural mechanics as well as current approximation methods in structural analysis are discussed. Knowledge of these equations and the approximation methods is essential in order to determine how well the CA approximation will compare with these methods. Most of the approximations discussed in Appendix A are derived from the governing equations and their relations.

The discrete formulation of CA makes it very difficult to predict the outcome of the method, as is clearly shown in [5]. In most cases the simulation requires too many components, and a direct approach of construction fails to predict the outcome of the simulation. One must instead attempt to extract the mathematical essence of the process by observing the results obtained by various formulations.

The governing equations in structural mechanics describe the way in which the material in a structure adjusts itself to absorb energy input into the structure. These fundamental mathematical components are independent of the size of the problem, making it possible to analyze small structures to understand the mechanisms common to such problems. The possibility embedded in such an approach is to identify fundamental mathematical mechanisms that are common to many different structural systems. Such commonly used mechanisms would correspond to universal features in the behavior of very different complex natural systems [4].

### 3.1.1 CA parameters in structural analysis

To analyze a structural analysis problem using cellular automata, various parameters have to be taken into consideration. Section 2.6 deals with established methods of handling the physical geometry to establish a structural mesh. In addition to the mesh that must be taken into consideration, there are other aspects to consider in CA simulations for practical problems in structural analysis:

1. How many cell states are sufficient to simulate the deformation of a structure?
2. How would one determine the cell rule to be used?
3. How would the boundary conditions be obtained and implemented?
4. What would the influence of the initial condition be and how would the initial conditions be obtained?
5. Which neighborhood size would be easy to handle yet sufficient to simulate the problem at hand?

We have already discussed the accuracy dependency of the CA solution in Section 2.7 and showed how this depends on the number of cell states. We now need to attempt a prediction on the influence the number of cell states has on the simulation time of the problem at hand.

#### **Example problem: Influence of the number of cells and number of cell states**

A very simple model is now set up, using a simple one-dimensional model in which we change the number of cell states and the number of cells in the system: Consider a bar which is clamped at one end and with an axial force applied, as depicted in Figure 3.1.

We assume that the point where the force is applied has the biggest displacement (the maximum or highest cell state used) and that the clamped side has zero displacement (lowest cell state used). Obviously the boundary conditions prescribe the smallest cell state as 0.

The discrete nature of CA is one of the major advantages when used as a computational mechanics method, since it allows for easy interaction on digital computers. It is logical to use cell states that can be stored effectively in a conventional computer's memory. Conventional computers use binary logic, thus we will use cell states that are of the form  $2^n$ , where  $n$

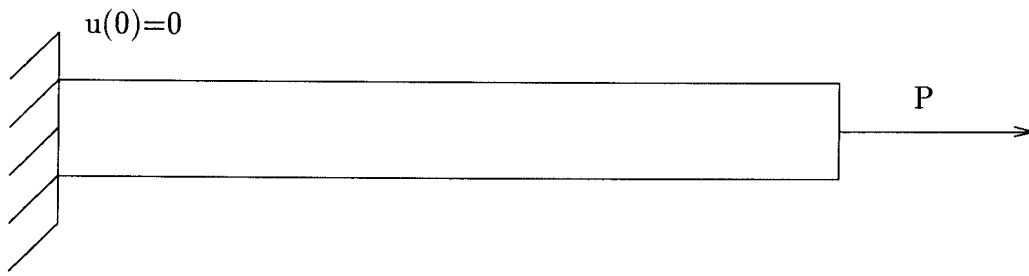


Figure 3.1: One-dimensional bar problem

indicates the number of bits used in memory. In computer programming languages, integers are usually limited to  $2^{(16-1)}$ , which is equal to 32768. One bit is subtracted as it is used internally to indicate the sign of the number. Since this number is easily handled by current programming languages, the maximum number of cell states is selected as 32768. This also results in a sufficiently accurate approximation in accordance with Section 2.7.

The cell rule used is the basic two-dimensional FDM equation as set out in Table A.1. By not using the information of the cell under consideration Laplace's equation (A.1) is satisfied and the cell rule is *totalistic*.

For this static problem, the solution will converge to a fixed state, which will place our CA in *class 2*. Thus the CA will reach the deformed state independently of its initial conditions. The number of cell states used will not influence the final state achieved but the evolution will indeed be different. For comparison, the initial cell state was set to 0 except at the natural boundary. This is performed for a different numbers of cells, and the result is shown in Figure 3.2.

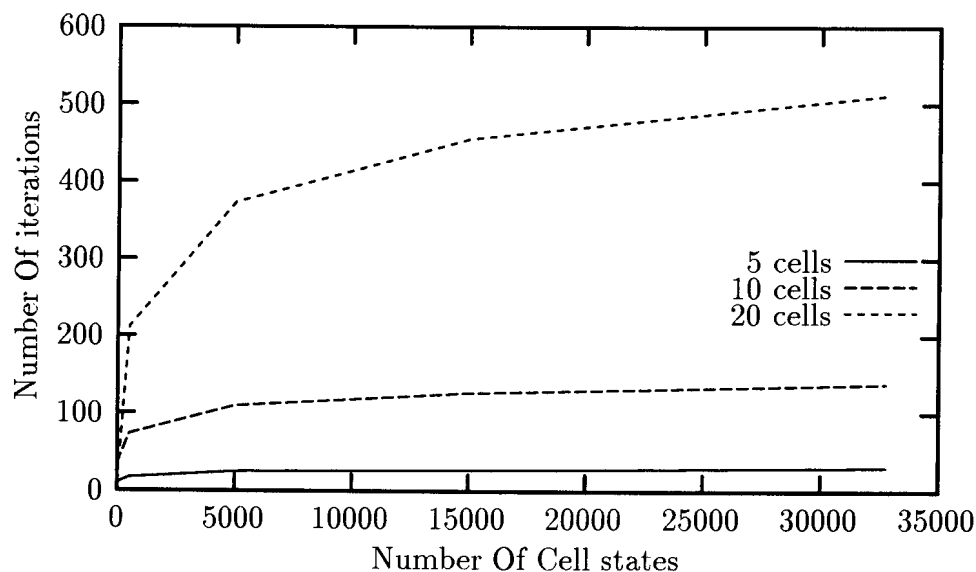


Figure 3.2: One-dimensional computational cost comparison

From Figure 3.2 it is clear that the cost of evaluation dramatically increases with the number of cells in the simulation. The number of cell states initially has a large influence on the



number of iterations (for few cell states), but as the number of cell states increase, the influence becomes small. This implies that the influence in processing time between a medium number of cell states and a full integer is not significant. The evolution of a CA with different numbers of cell states would, however, be different, although the model will reach the same final state.

From the comparisons performed above it is apparent that the combinations of variables that can influence the system are endless. One must instead attempt to logically identify combinations that are both practical and capable of delivering an extensive understanding of how a structural approximation would react on such parameters.

### 3.1.2 CA implementation

The values of interest in structural analysis are not directly prescribed by the boundary conditions (since the boundary conditions are typically prescribed in terms of displacement constraints, while the values of interest are displacements and stresses or strains). For CA to be applicable to general structural analysis, a method has to be found to “translate” the prescribed boundary conditions into a form that can be used by the CA.

Comparatively the BEM calculates values for displacements at the boundary of the structure and then calculates displacements at internal points in a separate step. FEM calculates all the displacements simultaneously and then calculates the stress at the integration points of each element in a separate step.

Since the stresses in a structure are directly related to the strains, it seems natural to only calculate only the displacements, then the stresses can be calculated from the strains. In BEM, only boundary conditions are approximated [38]. BEM can be used to calculate the displacements at the structures boundary and then this data could be transferred to the boundary of the CA simulation. BEM thus only calculates the data required for the CA simulation and the second step in the BEM is replaced by the CA.

## 3.2 Problem formulation

### 3.2.1 Problem descriptions

For a CA approximation to be universally acceptable, the approximation to the structural problem at hand must be able to handle various displacement and stress fields. This implies that more than one type of problem has to be considered. Figures 3.3 through 3.5 depict the problems considered in the remainder of this chapter.

The first problem depicted in Figure 3.3 is well suited to showing how the force applied at a point dissipates through a structure. Since the problem is square, it presents no problems for meshing or neighborhood definition.

For the second problem, we consider a similar problem to problem 1, obtained by simply modifying the boundary condition (the constraints, or *natural boundary conditions* as described in Appendix A). This problem is discussed by Hajela [39] and it is a classical problem

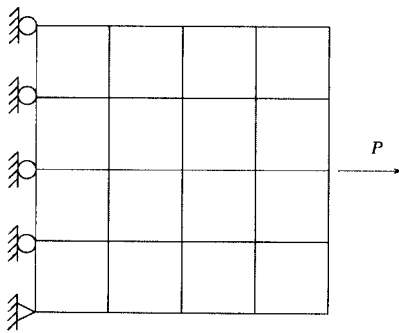


Figure 3.3: CA problem 1

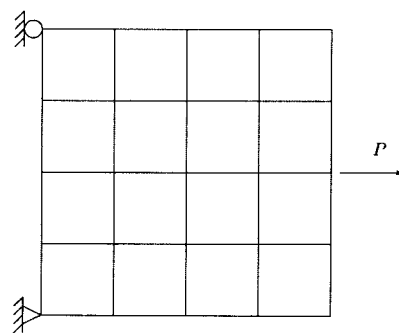


Figure 3.4: CA problem 2

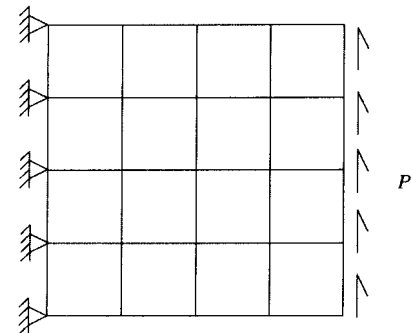


Figure 3.5: CA problem 3

used in topology optimization where the geometry of the structure is optimized. When a point load is applied in the numerical problem, the stresses at the point where the load is applied, tend towards infinity. It is physically impossible to apply a point load to a real structure.

The third problem is therefore subjected to a distributed load, which yields a complex stress field. However, the problem is still sufficiently simple for easy implementation. This problem is shown in Figure 3.5.

### 3.2.2 Neighborhood descriptions

For two-dimensional models, there are two computational FDM molecules frequently used as shown in Table A.1. These models are derived directly from the governing differential equations and are symmetrical. In CA terminology these computational molecules represent the cell rule, the difference being the discrete nature of the CA formulation, were the FDM method is continuous in nature.

For the first two-dimensional rule, no other combination of variables that are linearly independent from the original rule will result in convergence. This is because this rule is a simple averaging rule as implied by Laplace's equation A.1. Any other combination of variables will obviously result in a diverging rule.

The second molecule consists of a fixed neighborhood which is larger than the first one. Furthermore, if we presume that we are dealing with an isotropic<sup>1</sup> material, it would be logical to assume a symmetric cell rule similar to the one originally derived for FDM. This means that there are four possible variables that would form the cell rule. The four variables are depicted in Figure 3.6. No predictions can be made regarding which combination of variables would converge to a steady state, or even that if these variables have converged, that they will predict a correct solution.

Both of the neighborhoods used in FDM are capable of delivering good approximations. The definition of CA neighborhood however allows for the definition of any neighborhood to be used as long as it is uniform. To minimize the number of boundary cells required from an external source, (no more than one row of boundary cells), whilst maximizing the number of

<sup>1</sup>The same governing rule holds in all directions of the material.

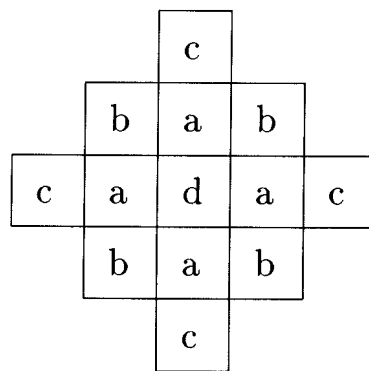


Figure 3.6: Extended neighborhood description

cells available for use, we can return to classical CA neighborhoods and conclude the need for a Moore neighborhood as defined in Section 2.3. The variables are defined in Figure 3.7.

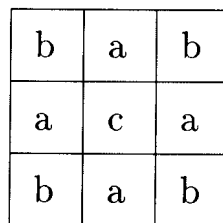


Figure 3.7: Moore neighborhood description

### 3.2.3 Analysis criteria

#### Introduction

As mentioned in structural analysis there is more than one value of interest to describe the state of a structure. These values are directly related to one another, e.g. the deformation in one direction is related to the deformation in the directions perpendicular to it by Poisson's ratio  $\nu$ . Deformation is caused by forces acting on the structure, this in turn induces stresses in the structure. When the stresses in the structure exceed the stress that the material is capable of handling the material fails and implicitly the structure also fails. But a structure might be considered as having failed when the deformations become large even if the stresses remain sufficiently low. (Such failures are considered as geometrically non-linear and beyond the scope of this investigation.)

To use CA as a computational mechanics method applicable to structural mechanics we need to isolate the values of interest in our simulation. These values should be sufficient to describe the deformation and failure of the structure as well as being capable of being competitive in terms of the computational time required.

### Displacement analysis

The values approximated in the FDM are displacements. Herein the first CA simulations are also performed using displacements. For an initial cell rule we use the first two-dimensional finite difference molecule shown in Table A.1, which makes use of only the four immediate neighbors. Because BEM calculates values all along the edge of the structure, the cell at the boundary could be given a fixed value for the CA simulation. The difference between this example and the work done by Hajela et al. [39] is that they presumed that fixed boundary cells (the boundary condition of the CA) only appeared at the points where the structural boundary conditions are applied. They assume free edges for the other cells. However, since the BEM calculates these points, in a combined method it would be senseable to use them even though the approach of Hajela is more general.

### Example problem: Separated displacement components

For the first attempt the two displacements are analysed separately; viz. the  $x$  displacements are calculated independently from the  $y$  displacements after the boundary conditions are obtained. The maximum cell state is set to a full integer for high precision and this value is prescribed as the maximum displacement encountered. This means that the difference in displacement between every cell state is directly proportional to the maximum displacement encountered at the boundary.

The boundary conditions of the structure are then imported from BEM. In order to benchmark how well the CA simulation performs, the problem is compared to the same problem solved using FEM as discussed in Section A.3.3. The FEM model is set up to be more refined by placing a FEM node at the center of each cell (where the value for the cell is calculated). In this manner, four 4-noded isoparametric FEM elements are used to model one cell. Using four FEM elements produced a more accurate solution than BEM, and the FEM solution is considered an ideal solution. With BEM as an input, CA could not achieve the same accuracy and for this reason, the BEM code was also used to calculate internal values. Both were then compared to the FEM analysis by the error function, as given below.

$$\epsilon = \sum |f_1 - f_2| \quad (3.1)$$

The calculations are performed separately for the  $x$  and the  $y$  components and the results are shown in Table 3.1.

Number of cells	BEM		CA	
	$x$	$y$	$x$	$y$
$5 \times 5$	4.7041E-2	1.3452E-1	1.3167E-1	1.4656E-1
$10 \times 10$	9.3842	1.5656	9.5645	1.77099

Table 3.1: Displacement error comparison of BEM and CA in comparison to FEM

A number of features are apparent from Table 3.1. Firstly, the CA simulation delivers similar results to BEM. This is a good indication since CA cannot be expected to deliver

better performance based on the same boundary calculations as BEM. In order to determine what difference the cell rule makes to the results, the size of the neighborhood is increased. There are no other possible cell rules to be used with this neighborhood that are able to converge. The fact that the two displacements are calculated separately means that the analysis is run twice. Since it is clear from Figure 3.2 that this would result in a dramatic increase in the processing time required as the number of cells increases, it is obviously not a viable option.

All attempts to combine the displacements into one cell by making use of the Poisson's relation failed, since none of the combinations constructed were able to correctly keep track of the direction of each component of the combined displacement, because an extension in one direction meant a narrowing in the other direction. This is too much information to be contained in one cell state while the physics of the problem at hand are also being captured.

### Stress analysis

A single criterion is obviously desirable that describes the behavior of the structure by one cell rule. Such a criterion (von Mises stress) was previously used by Hajela [39].

The von Mises stress is used as a criterion in determining the onset of failure in ductile materials [40]. The failure criterion states that the von Mises stress  $\sigma_{VM}$  should be less than the yield stress  $\sigma_Y$  of the material [40]. In the inequality form, the criterion may be stated as

$$\sigma_{VM} \leq \sigma_Y \quad (3.2)$$

where the von Mises stress  $\sigma_{VM}$  is given by

$$\sigma_{VM} = \sqrt{I_1^2 - 3I_2} \quad (3.3)$$

In (3.3),  $I_1$  and  $I_2$  are the first two invariants of the stress tensor [40].  $I_1$  and  $I_2$  are given by

$$\begin{aligned} I_1 &= \sigma_x + \sigma_y + \sigma_z \\ I_2 &= \sigma_x\sigma_y + \sigma_y\sigma_z + \sigma_z\sigma_x - \tau_{yz}^2 - \tau_{xz}^2 - \tau_{xy}^2 \end{aligned} \quad (3.4)$$

In terms of the principal stresses,  $\sigma_1$ ,  $\sigma_2$  and  $\sigma_3$ , the invariants can be written as

$$\begin{aligned} I_1 &= \sigma_1 + \sigma_1 + \sigma_3 \\ I_2 &= \sigma_1\sigma_2 + \sigma_2\sigma_3 + \sigma_3\sigma_1 \end{aligned} \quad (3.5)$$

Hence, (3.3) can be written as

$$\sigma_{VM} = \frac{1}{\sqrt{2}} \sqrt{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2} \quad (3.6)$$

The equation is simplified in the case of plane stress and plane strain. For plane stress we obtain

$$\begin{aligned} I_1 &= \sigma_x + \sigma_y \\ I_2 &= \sigma_x \sigma_y - \tau_{xy}^2 \end{aligned} \quad (3.7)$$

and for plane strain we obtain

$$\begin{aligned} I_1 &= \sigma_x + \sigma_y + \sigma_z \\ I_2 &= \sigma_x \sigma_y + \sigma_y \sigma_z + \sigma_z \sigma_x - \tau_{xy}^2 \end{aligned} \quad (3.8)$$

where  $\sigma_z = \nu(\sigma_x + \sigma_y)$ .

The von Mises criteria will always have a positive value. This value is an indication of how far the structure is from failure at any point within the structure. It is thus a sensible criterion to ascertain the overall performance of the structure.

## 3.3 Determination of the optimum cell rule

### 3.3.1 Motivation

Given generic macroscopic behavior, it is important for both theoretical and practical purposes to try and find the simplest microscopic dynamics which can reproduce the macroscopic behavior. One may, for example, seek the simplest cellular automaton rule which reproduces a particular form of continuum behavior<sup>2</sup>.

In the simplest CA, one considers rules which specify the new value of a single site in terms of the values of a neighborhood of sites around it at the previous time step. In most such rules, no additive quantities can be conserved. In addition, such rules are usually highly irreversible, so that they evolve towards attractors which contain only a subset of the possible states [7]. One needs to identify these rules that govern the physics of the problem. These rules would create macroscopic behavior common in all structural problems.

In problems where such rules are difficult to identify, or situations where the physics of the problem is not properly understood, alternative strategies must be implemented [39]. Hence

---

<sup>2</sup>“Simplest” can be defined for example, as requiring minimum storage space and minimum number of logical operations to implement [7].

one can consider finding minimal cellular automaton rules by various iterative and adaptive procedures [7]. This process of discovering “the laws of the universe” for a given problem is central to an ability to extend the approach to diverse analysis and design problems [39]. Before a rule can be extracted for a certain problem or set of problems one needs to define a criterion by which the performance of a certain rule can be evaluated. Once such a criterion is in place, an attempt can be made to identify the governing rule by an artificially intelligent selection process.

### 3.3.2 Cell rule comparison definition

From the description of the genetic algorithm (GA) in Appendix B.2 we define the best cell rule as the “fittest individual”. In order to determine the performance of our CA model, we need to define an *error* or *residual* with which all the points in the system can be combined to ascertain the “fitness” of the rule. For each cell in our discrete mesh we can define an error given by

$$\epsilon = f(x_k) - y_k \quad \text{for } 1 \leq k \leq N \quad (3.9)$$

Here,  $N$  is the total number of cells in the system,  $y_k$  is the real function value at point  $k$  and  $f(x_k)$  is the approximate value, in this case the value obtained by our CA simulation. We now have to combine all these small errors into a function value that can be used to describe the “fitness” of the rule. In classical curve fitting there are three criteria by which this can be done, as shown in (3.10) [41]. The CA simulation actually performs a three-dimensional curve fit, where the stress is seen as the function value dependent on the  $x$  and  $y$  coordinates.

$$\begin{aligned} \text{Maximum error:} \quad E_1 &= \max|\epsilon| \\ \text{Average error:} \quad E_2 &= \frac{1}{N} \sum_{k=1}^N |\epsilon| \\ \text{Root-Mean-Square(RMS) error:} \quad E_3 &= \left[ \frac{1}{N} \sum_{k=1}^N |\epsilon^2| \right]^{\frac{1}{2}} \end{aligned} \quad (3.10)$$

The RMS error is considered as the best criterion, since it penalizes the solution heavily for a single point with low accuracy, but uses all the values in the system to calculate the total error. We have deduced that the von Mises stress criterion is a good value to be approximated by the CA simulation, since it is described by one physical positive quantity, but it is sufficient to describe failure in linear problems.

With a discrete optimization problem, it is clear that in finding the best cell rule, traditional gradient-based approaches to numerical optimization would be unsuccessful. With all the numbers being discrete values, the optimization is more difficult than for continuous problems. If we consider that, for the extended von Neumann neighborhood each of the four optimization variables has 64 possible discrete values, this means that the total number of combinations are  $64^4 = 16777216$ . If we presume that one second were required per function evaluation (as a result of the rule, the problem either converged to a single solution or diverged until the maximum number of iterations is exceeded), evaluating all possibilities

would take 194.18 days. A smart searching (optimization) method therefore has to be used to extract the optimum cell rule. GAs make use of the survival of the fittest strategy found in nature to search the solution space of a function as described in Appendix B.2.

## 3.4 Problem analysis and results

### 3.4.1 Extended neighborhood

Consider Problem 1, shown in Figure 3.3. In accordance with Figure 3.6 (the extended Moore neighborhood), there are four variables to be used in the optimization methodology. We presume that the cell rule will involve some type of averaging to be able to reach a steady state prescribed by Poisson's equation. Presuming that these variables will have discrete integer values, similar to the FDM rule, we define a maximum and minimum value for each number. For the sake of computational efficiency we presume that each cell can have an influence of  $2^6 = 64$ . This means that the variables  $a, b$  and  $c$  can have values between  $-31$  and  $32$  and the variable  $d$  can vary between  $1$  and  $64$ . The rule for a cell with coordinates  $(i, j)$  is shown in

$$\begin{aligned} Cell_{ij}^{New} = & (a(Cell_{i+1,j} + Cell_{i-1,j} + Cell_{i,j+1} + Cell_{i,j-1}) \\ & - b(Cell_{i+1,j+1} + Cell_{i-1,j+1} + Cell_{i-1,j-1} + Cell_{i+1,j-1}) \\ & - c(Cell_{i+2,j} + Cell_{i-2,j} + Cell_{i,j+2} + Cell_{i,j-2}))/d \end{aligned} \quad (3.11)$$

This rule uses the value of the cell under consideration and is consequently considered *outer totalistic*. It is clear that not all possible combinations will converge. Rules that diverge are monitored, stopped and then penalized with a large error, to prevent propagation into future generations in the optimization process.

The GA search is performed on a small mesh size which makes the simulation run reasonably fast. The FEM analysis is now used to prescribe boundary conditions for the CA simulation. The rules obtained for each specific number of cell states are shown in Table 3.2.

Table 3.2 shows the computations performed with different variables. The first column describes the maximum amount of cell states used in the form  $2^n$ , with  $3 \leq n \leq 15$ , which uses the optimal amount of computer resources. Finally this is done with a high precision continuous 64 bit floating point variable for comparison with the traditional FDM rule. The second column shows the total RMS error for all the points in the system. The fact that the total error is divided by the amount of points used enables us to compare different mesh sizes. The cost is the amount of iterations taken for convergence of the whole system and the variables show the rules extracted in accordance with (3.11).

The RMS error decreases as more cell states are used and then starts increasing again. The lowest RMS error is obtained at 512 cell states. A very interesting result is seen with the rules obtained:

- The higher the number of cell states, the more often the same cell rule is obtained.



Cell states	Error RMS	Cost	a	b	c	d
8	0.43291552E-01	5	2	-15	0	56
16	0.27817578E-01	5	-2	-16	-4	62
32	0.15375927E-01	6	-1	-20	2	62
64	0.79718766E-02	8	0	-14	0	52
128	0.56412691E-02	8	2	-15	0	64
256	0.57616191E-02	10	8	-6	0	54
512	0.26902889E-02	17	16	-1	2	59
1024	0.31755326E-02	17	13	-2	1	55
2048	0.35064670E-02	21	16	-1	1	63
4096	0.40553210E-02	20	13	-2	1	55
8192	0.34612693E-02	23	14	-2	1	59
16384	0.35566437E-02	24	14	-2	1	59
32768	0.35830220E-02	25	14	-2	1	59
Cont.	0.36324311E-02	46	14	-2	1	59

Table 3.2: Optimized rules for  $8 \times 8$  mesh on Extended neighborhood FEM problem 1

- A problem with the continuous values used for accuracy, is that almost twice the number of iterations are required to obtain convergence on the same cell rule with closely the same error.

None of these rules correspond with the FDM equation derived and shown in Table A.1. Smaller numbers of cell states display higher errors than the larger numbers. The RMS errors obtained are compared in Figure 3.8.

The errors achieved with the optimized rules are lower than those achieved with the FDM rule. The FDM rule formulation uses continuous variables and it would be expected that the FDM rule would perform far better with more cell states than with lower numbers of cell states. However, the error remained almost constant and independent of the number of cell states used for large numbers of cell states.

Just as important as the error achieved, is the number of iterations required for the solution to converge. Bigger mesh sizes take longer to update their cell states with each iteration and since the influence of the boundary values propagates through the structure, require more iterations to converge. The cost in terms of the number of iterations required is shown in Figure 3.10.

The rules obtained converge in fewer iterations than the FDM rule, and are less dependent on the number of cell states used. This could mean a large reduction in computational cost for the rules used in large problems since each iteration requires a CA adjustment of each cell.

The RMS error indicates how close the approximation is to the real solution in terms of the curve fit. But, as for common structural analysis, we need to display the stresses in a form which is universally understandable in engineering. The von Mises stress plots for the

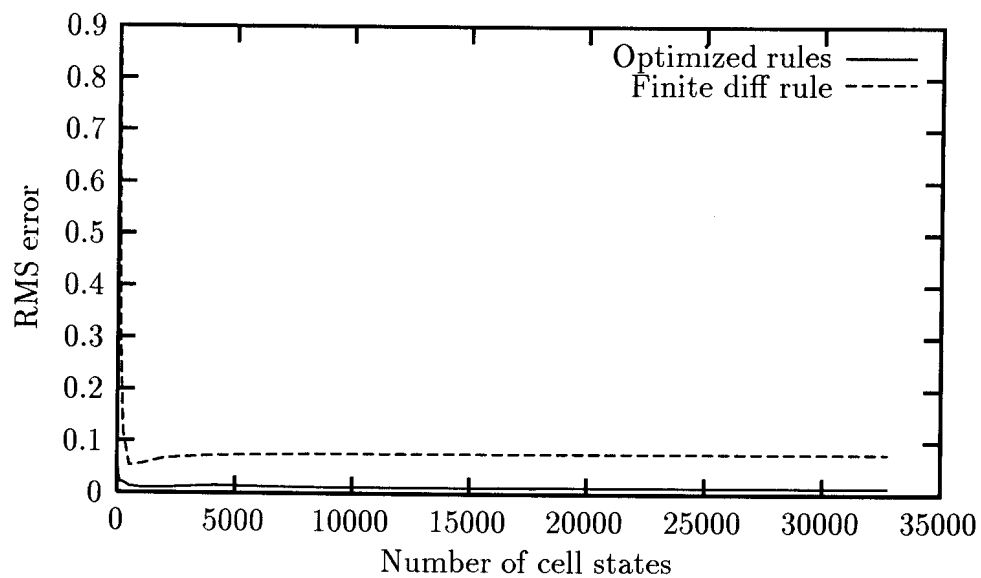


Figure 3.8: Extended neighborhood RMS error comparison

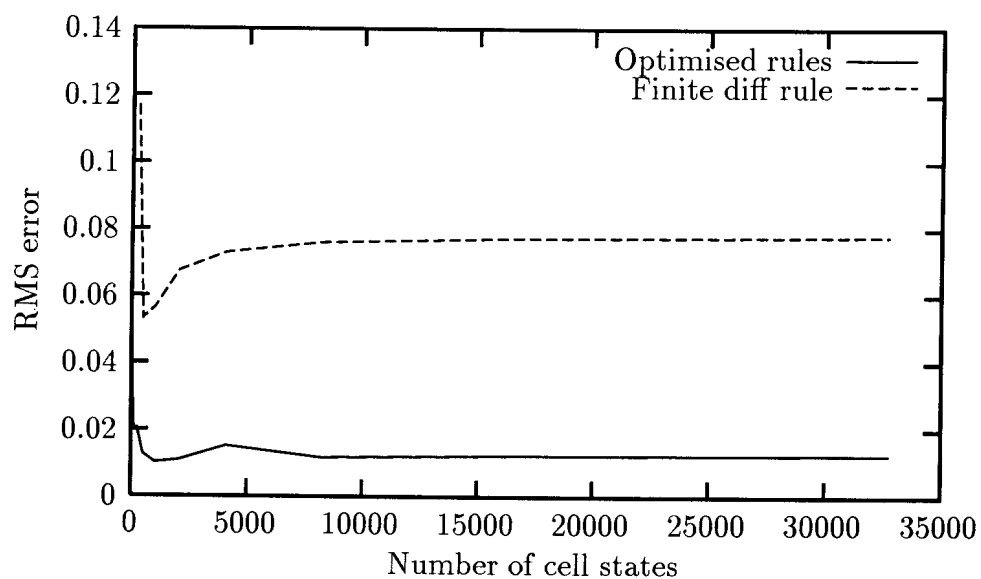


Figure 3.9: Extended neighborhood RMS error closer comparison

various stresses are given in Figures 3.11 to 3.16.

The results obtained seem promising since the resultant rules out-perform the FDM rules in all cases. The stress plots seem accurate for low numbers of cell states. Before these tendencies can be considered as a phenomenon, a few remarks should however, be made:

- Is it practical to use an extended neighborhood since the neighborhood requires that two rows of imported cells be used as the boundary conditions?
- If this good performance is indeed true, is it also true for other, more practical, prob-

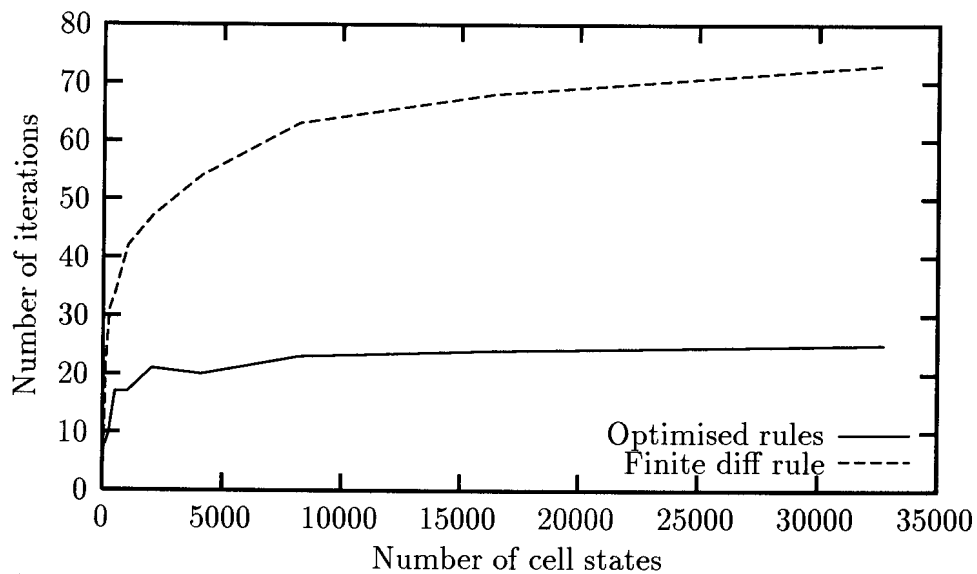


Figure 3.10: Extended neighborhood computational cost comparison

lems?

- It is clear that the extended neighborhood would involve more complications in larger mesh sizes. The number of cells used in each calculation requires many neighboring cells to be “looked up”. This procedure is known to be expensive in terms of computational cost in a conventional computer, despite the differences in instruction sets used by various computer implementation.
- In the foregoing example, the number of “free” cells in the simulation is relatively low.

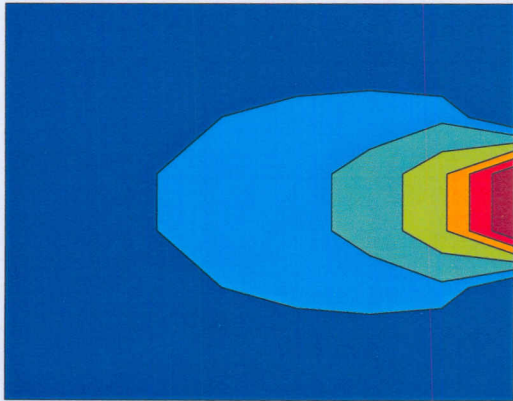


Figure 3.11: Problem 1 FEM  $8 \times 8$

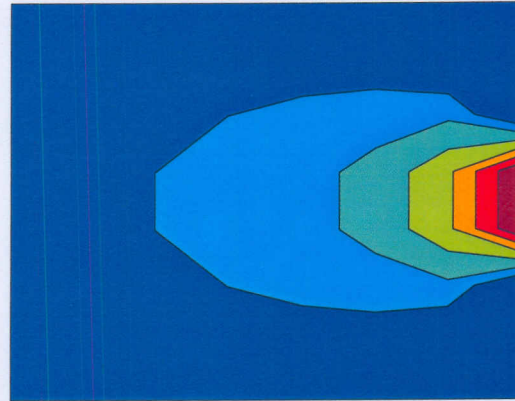


Figure 3.14: CA optimized 1024 cell states

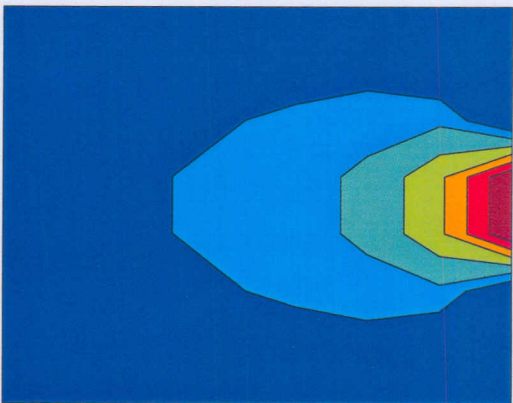


Figure 3.12: CA optimized 32 cell states

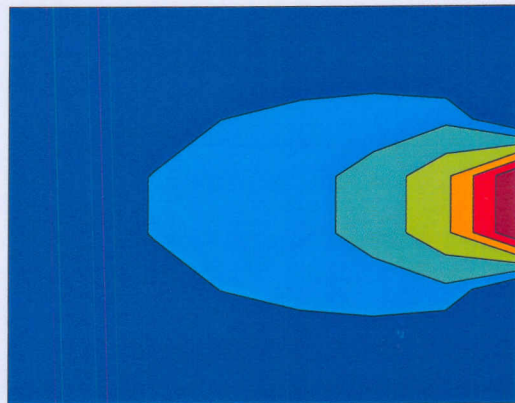


Figure 3.15: CA optimized 32768 cell states

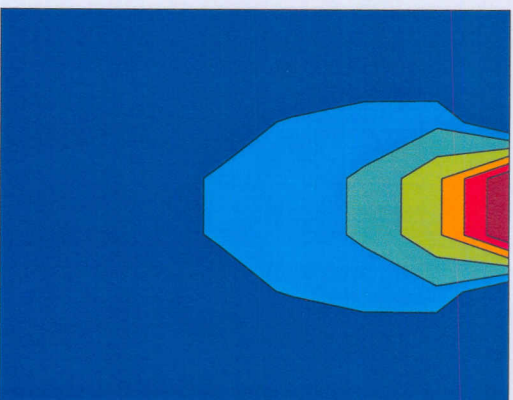


Figure 3.13: FDM 4 neighbors

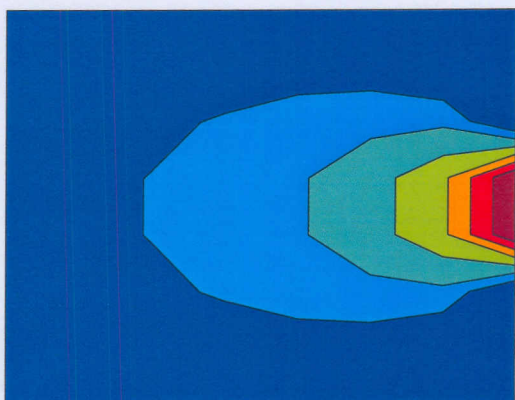


Figure 3.16: Extended FDM

### 3.4.2 Moore neighborhood

We shall now attempt to answer some of the questions that arose in Section 3.4.1. By reducing the neighborhood size to a Moore neighborhood, the cell rule for the cell at any point  $i, j$  can be written in terms of the variables

$$Cell_{ij}^{New} = (a(Cell_{i+1,j} + Cell_{i-1,j} + Cell_{i,j+1} + Cell_{i,j-1}) + b(Cell_{i+1,j+1} + Cell_{i-1,j+1} + Cell_{i-1,j-1} + Cell_{i+1,j-1}))/c \quad (3.12)$$

These rules are also considered *outer totalistic*. The number of variables is now reduced to three. This means that the number of possible combinations is  $64^3 = 262144$ , which is  $O(2)$  less than the extended neighborhood. Exploring all possible combinations on an AMD Athlon 800MHz requires less than 20 minutes and hence all solutions shown for this problem are the known optimum combinations for the three variables used. This time measurement is reliant on the assumption that the range of  $a$  and  $b$  may vary between  $-31$  and  $32$  and variable  $c$  has a range between  $1$  and  $64$ . The mesh size is also increased to  $16 \times 16 = 256$  cells. Either FEM or BEM can be used for the boundary input for this problem, but FEM is chosen as it is capable of obtaining more accurate solutions. Still considering Problem 1 (Figure 3.3) the best cell rules are now extracted, and depicted in Table 3.3.

Cell states	Error RMS	Cost	a	b	c
8	0.53015190E+00	3	-1	14	35
16	0.47712490E+00	4	3	8	34
32	0.42685911E+00	10	2	7	31
64	0.34310171E+00	41	1	15	58
128	0.30804555E+00	64	0	11	42
256	0.53125077E-01	75	9	2	43
512	0.61781108E-01	58	8	8	63
1024	0.93192644E-01	54	1	15	63
2048	0.11832671E+00	60	0	16	63
4096	0.13436320E+00	69	0	16	63
8192	0.14203794E+00	75	0	16	63
16384	0.14606122E+00	86	0	16	63
32768	0.14467440E+00	106	4	1	20

Table 3.3: Optimized rules for  $16 \times 16$  mesh on Moore neighborhood FEM problem 1

Table 3.3 shows the same tendencies as Table 3.2. The error decreases as the number of cell states increases and then increases again as even more cell states are used. The smallest error is obtained with 256 cell states. Again the cell rule seems to converge to a given rule for this problem. The RMS errors are greater than that obtained for the previous problem. This is to be expected, since the number of points used for the error calculation is increased

dramatically. The cost (in terms of the number of iterations required) for this problem increases dramatically when compared to the smaller mesh size. The rules obtained are totally different from those obtained in Table 3.2. This change is expected due to the change in neighborhood. Once again, most of the obtained CA rules are superior to the FDM rule. The fact that fewer cell states result in more accurate solutions is cause for investigation. The believed reason for this tendency is simplified and explained in Figure 3.17.

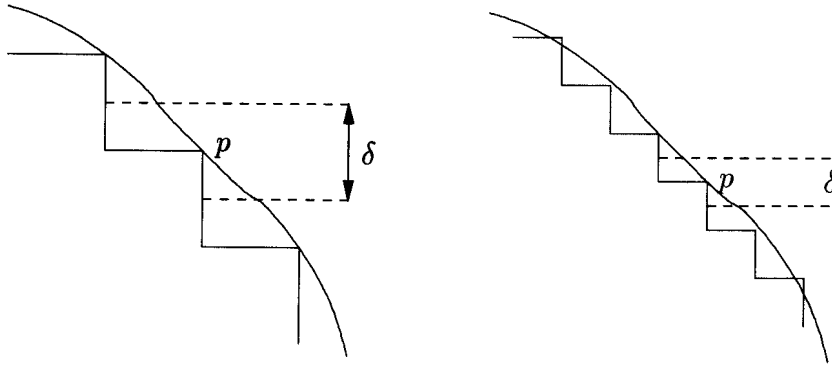


Figure 3.17: Cell state accuracy round off occurrence

As the solution is only being approximated at point  $p$ , the course grid has a bigger area which is seen as the exact value at that point. The smaller grid has a smaller area which is translated into the exact point. With a smaller number of cell states, the number of points which yields a solution close to the real values is greater in comparison to the total number of points.

The stress contours for selected results are plotted in Figures 3.18 to 3.23.

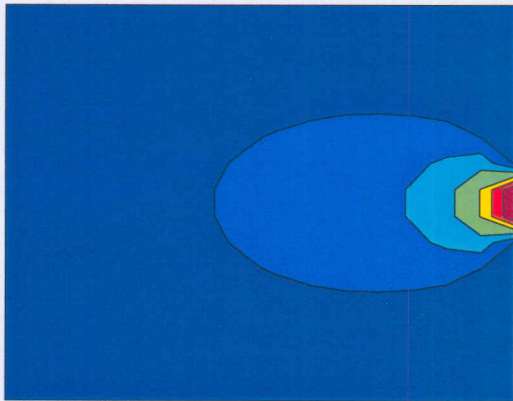


Figure 3.18: Problem 1 FEM  $16 \times 16$

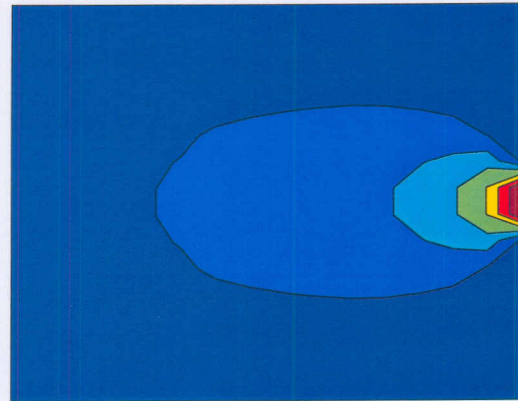


Figure 3.21: CA optimized 256 cell states

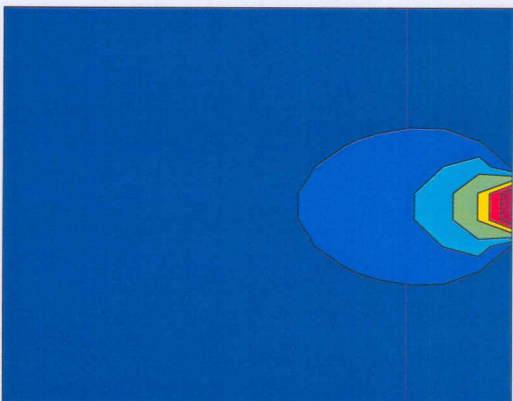


Figure 3.19: FDM 4 neighbors

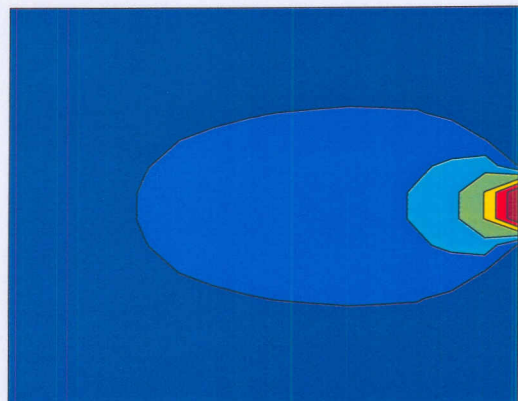


Figure 3.22: CA optimized 2048 cell states

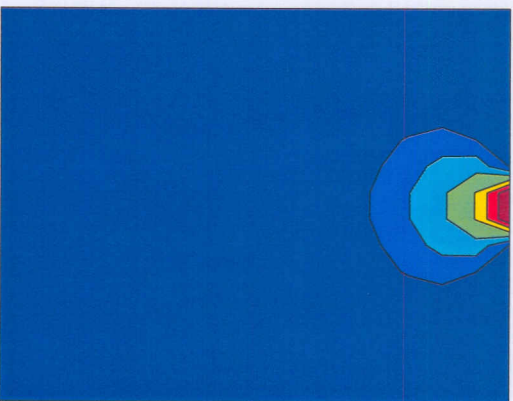


Figure 3.20: CA optimized 32 cell states

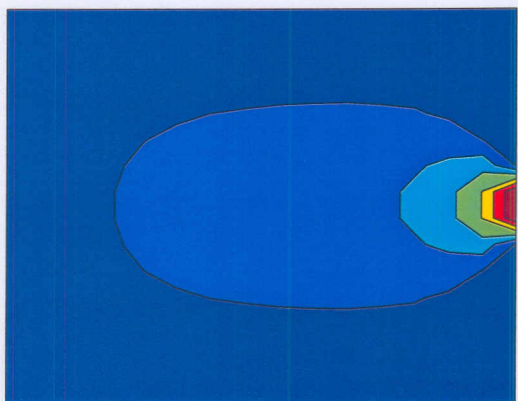


Figure 3.23: CA optimized 32768 cell states

**Problem 2**

To gain a better understanding of how a CA model approximates structural problems, the required rules for different stress fields are now investigated. The optimum rules for a Moore neighborhood are determined for the second problem investigated (see Figure 3.4). The cell rules shown in Table 3.4 are obtained using BEM.

Cell states	Error RMS	Cost	a	b	c
8	0.60469618E+00	2	6	1	22
16	0.55514977E+00	3	3	4	24
32	0.54489576E+00	5	14	1	57
64	0.50717726E+00	10	13	3	62
128	0.47369715E+00	12	4	1	20
256	0.38586495E+00	29	1	0	4
512	0.37508172E+00	44	8	-1	28
1024	0.36018374E+00	43	14	1	61
2048	0.35617211E+00	47	15	0	61
4096	0.35266954E+00	54	15	0	61
8192	0.35122104E+00	60	15	0	61
16384	0.35072195E+00	70	15	0	61
32768	0.35035057E+00	77	15	0	61

Table 3.4: Optimized rules for  $16 \times 16$  mesh on Moore neighborhood BEM problem 2

In this problem the optimum cell rule is obtained using the greatest number of cell states. An interesting fact is that the rule obtained is indeed the 4 neighbor FDM rule, since  $\frac{15}{61} = 4.066$ . The rules obtained with this problem are clearly different from those obtained in our first problem, which indicates that the change in stress field influences the best rule for the problem. The stress contours in this problem are shown in Figures 3.24 to 3.29.

It is clear that the stress contours from the resultant optimum rules do not yield the correct stress contours. The BEM solution shows a band which spreads the stress from the point where the force is applied to the corner points, resulting in a core in the center where no stress occurs, as shown in Figure 3.24. The CA rules are not capable of reproducing this phenomenon. Results from the refinement of the meshes for FEM and BEM are shown in Figures 3.30 to 3.33. By using a non-linear FEM package, the problem was found to be geometrically non-linear. The CA simulation is, however, capable of using these values and obtaining reasonably good results.



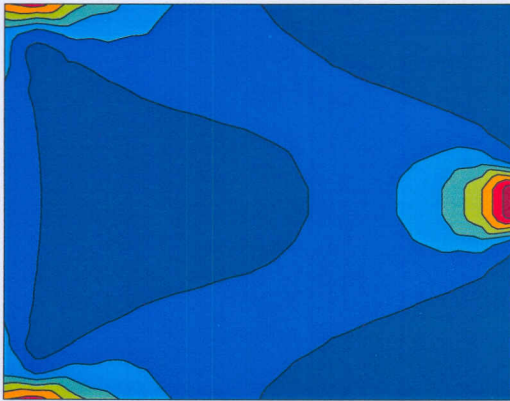


Figure 3.24: Problem 2 BEM  $16 \times 16$

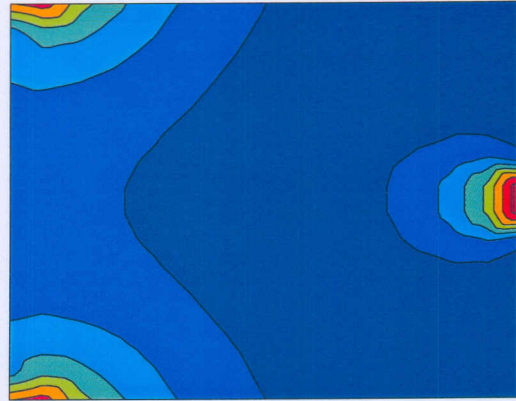


Figure 3.27: CA optimized 256 cell states

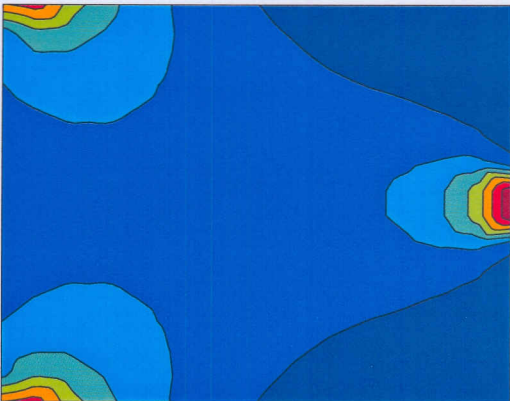


Figure 3.25: FDM 4 neighbor

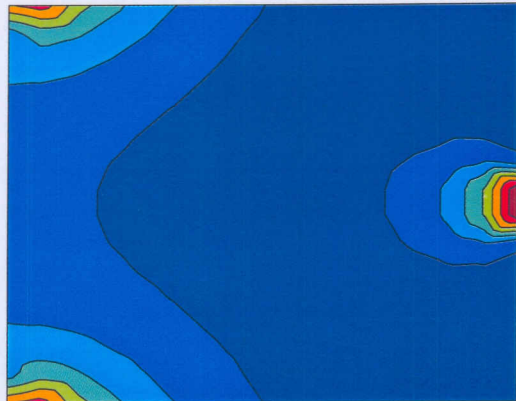


Figure 3.28: CA optimized 2048 cell states

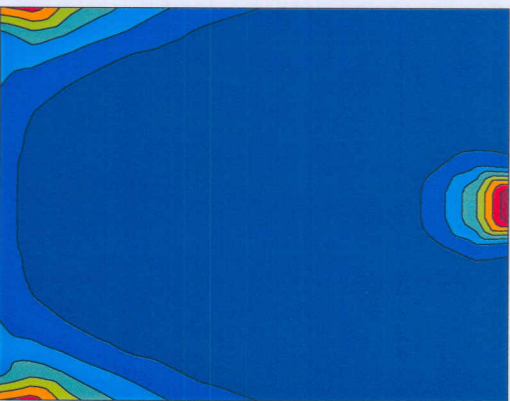


Figure 3.26: CA optimized 32 cell states

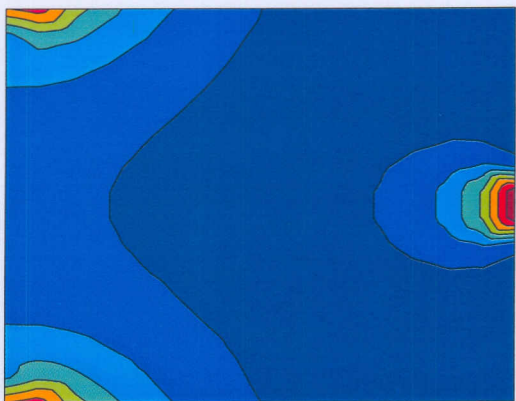


Figure 3.29: CA optimized 32768 cell states

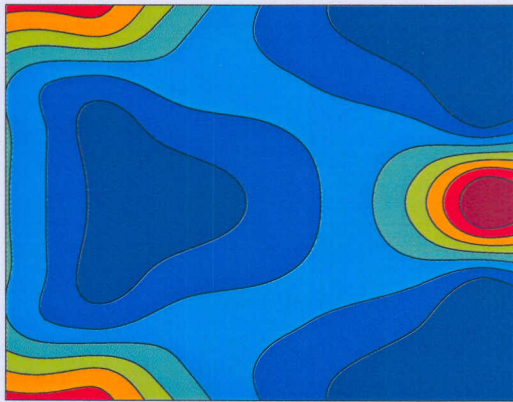


Figure 3.30: Problem 2 BEM  $8 \times 8$

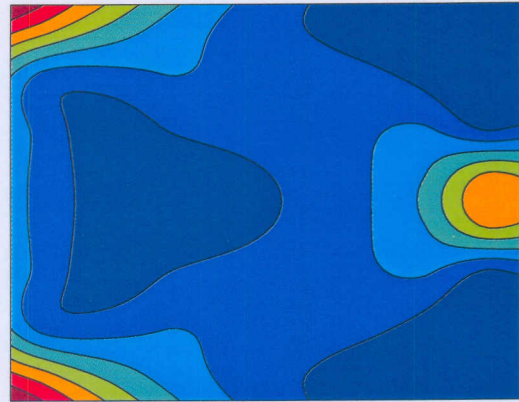


Figure 3.32: Problem 2 FEM  $8 \times 8$

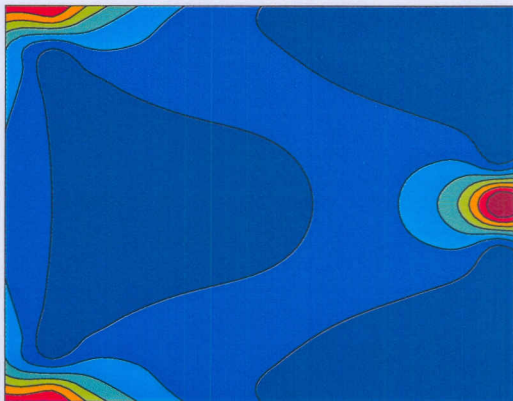


Figure 3.31: Problem 2 BEM  $16 \times 16$

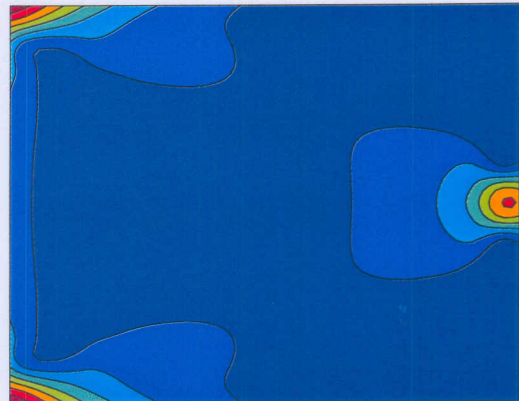


Figure 3.33: Problem 2 FEM  $16 \times 16$

### Bending problem

We now consider Problem 3 (Figure 3.5). It is possible to obtain an analytical solution to this problem [42]. However, the FEM and BEM codes are still used in this section. As opposed to Problem 2, this problem is linear, which allows for better comparison between different mesh sizes. Searching for the best cell values, we obtain the results given in Table 3.5.

Cell states	Error RMS	Cost	a	b	c
8	0.25105715E+00	11	1	3	14
16	0.11395269E+00	15	5	2	26
32	0.84983295E-01	17	11	2	50
64	0.78008803E-01	24	14	-3	43
128	0.84722215E-01	19	13	2	59
256	0.87620568E-01	21	12	1	52
512	0.77930804E-01	26	7	1	32
1024	0.75518361E-01	29	5	1	24
2048	0.74438666E-01	31	8	1	36
4096	0.74259859E-01	34	7	1	32
8192	0.74165211E-01	35	5	1	24
16384	0.74118742E-01	42	6	1	28
32768	0.74098354E-01	42	5	1	24

Table 3.5: Optimized rules for  $8 \times 8$  mesh on Moore neighborhood FEM problem 3

For this problem, the error decreases as the number of cell states decreases but the decrease is not dramatical. At 64 cell states, the problem is already very close to the minimum error. The error then increases slightly due to the rounding as described in section 3.4.2 and then decreases slowly.

The cell rules obtained appear to approximate the FDM 4 neighbor rule. The ratio is always very close to four. The rules obtained do, however, use some of the information from the corner cells. Selected plots for the rules obtained are shown in Figures 3.34 to 3.39. The best stress contours are obtained with 64 cell states.

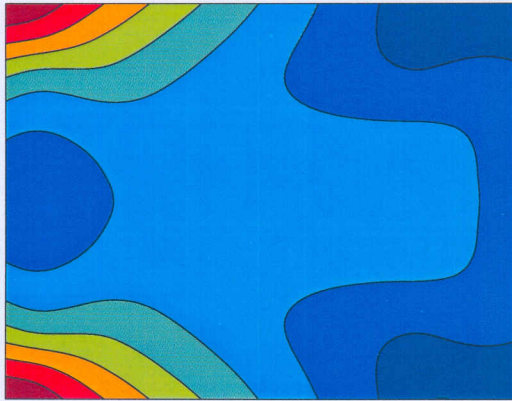


Figure 3.34: Problem 3 FEM  $8 \times 8$

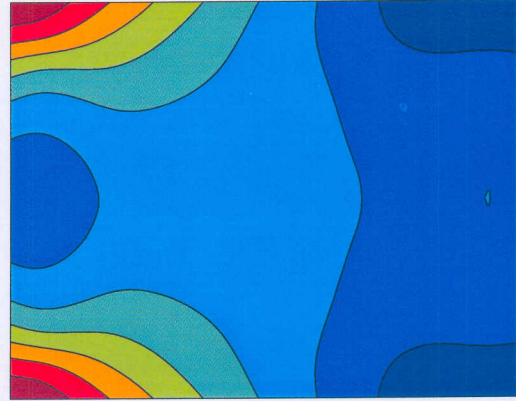


Figure 3.37: CA optimized 512 cell states

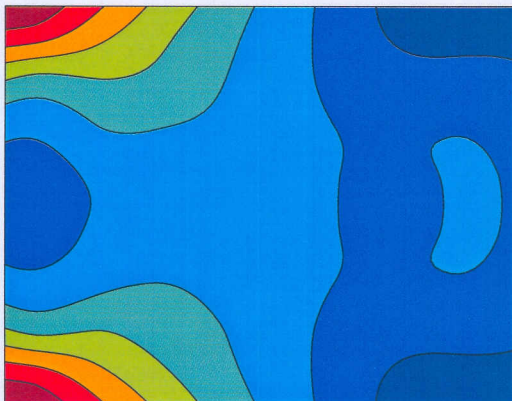


Figure 3.35: CA optimized 32 cell states



Figure 3.38: CA optimized 1024 cell states



Figure 3.36: CA optimized 64 cell states



Figure 3.39: CA optimized 32768 cell states

For continuity, we now increase the mesh size to  $16 \times 16$  for the same bending problem (Figure 3.5). The results are shown in Table 3.6.

Cell States	Error RMS	Cost	a	b	c
8	0.97218127E+00	12	21	-5	52
16	0.90144640E+00	17	13	-4	32
32	0.83100041E+00	30	23	-7	60
64	0.75792672E+00	33	23	-8	58
128	0.58255949E+00	76	25	-9	63
256	0.61141889E+00	64	25	-9	64
512	0.31104888E+00	138	25	-9	64
1024	0.17773177E+00	166	11	-4	28
2048	0.12366561E+00	235	11	-4	28
4096	0.10983362E+00	260	11	-4	28
8192	0.10741246E+00	296	11	-4	28
16384	0.10732287E+00	323	19	-7	48
32768	0.10771372E+00	354	11	-4	28

Table 3.6: Optimized rules for  $16 \times 16$  mesh on Moore neighborhood FEM Problem 3

Once again we find that the solution converges to one cell rule. This time the converged cell rule is not the same as the FDM rule and the error terms decrease as the number of cell states increases. The rules are completely different from those obtained in Table 3.5 although the stress fields are comparable. The stress contours for the selected rules are plotted in Figures 3.40 to 3.45. Note that, although the rules obtained are reasonable, none of them are capable of closely resembling the correct result.

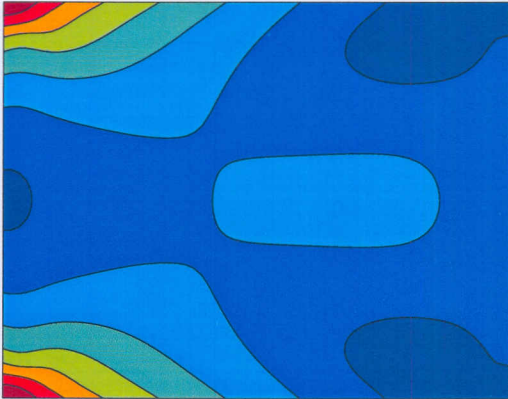


Figure 3.40: FEM  $16 \times 16$  stress plot

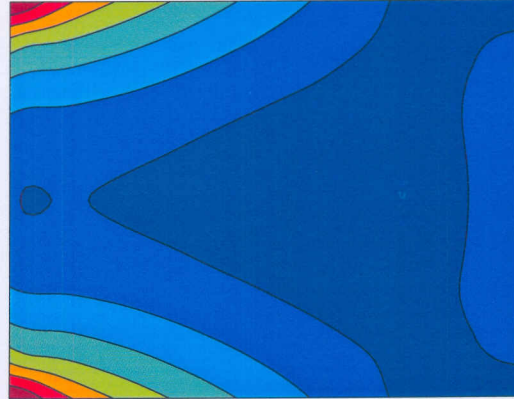


Figure 3.43: CA optimized 512 cell states

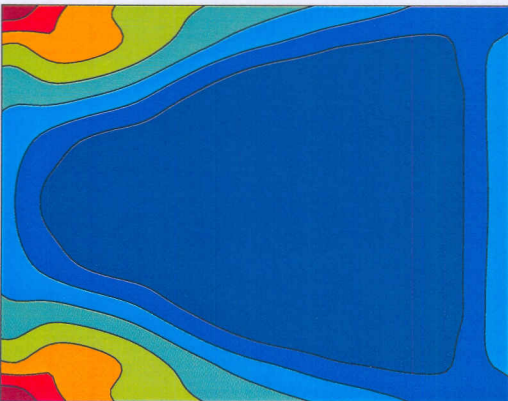


Figure 3.41: CA optimized 32 cell states



Figure 3.44: CA optimized 2048 cell states

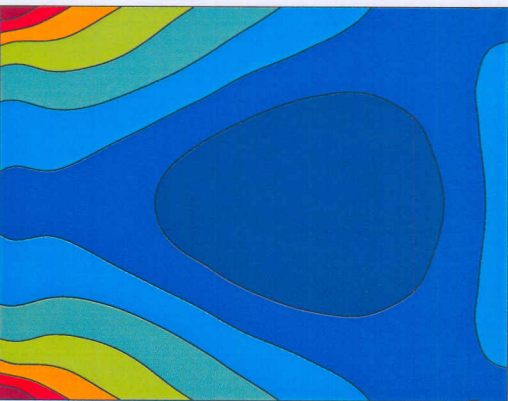


Figure 3.42: CA optimized 128 cell states

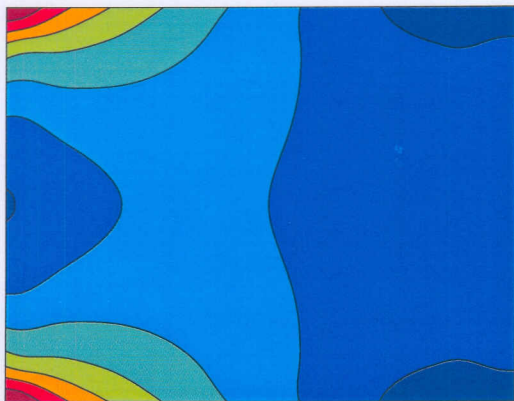


Figure 3.45: CA optimized 32768 cell states

We shall now investigate the influence of increasing the number of variables that influence each cell. Doubling the number of cell variables per cell implies that the range of variables  $a$  and  $b$  may vary between  $-63$  and  $64$  and variable  $c$  has a range between  $1$  and  $128$ . This increases the number of combinations. The rules obtained can be seen in Table 3.7.

Cell states	Error RMS	Cost	a	b	c
8	0.97218127E+00	12	21	-5	52
16	0.88905889E+00	18	27	-7	71
32	0.83100041E+00	30	23	-7	60
64	0.69464500E+00	46	44	-15	112
128	0.52526133E+00	85	47	-17	118
256	0.54318169E+00	118	53	-23	119
512	0.30518049E+00	132	48	-17	124
1024	0.17581473E+00	173	36	-13	92
2048	0.12366561E+00	235	11	-4	28
4096	0.10983362E+00	260	11	-4	28
8192	0.10730810E+00	285	47	-17	120
16384	0.10724750E+00	322	47	-17	120
32768	0.10768116E+00	358	35	-13	88

Table 3.7: Optimized rules for  $16 \times 16$  mesh for more cell states on a Moore neighborhood FEM Problem 3

We find that for four of the rules obtained (8, 32, 2048 and 4096 cell states) the increase in the number of cell variables had no influence in the rule obtained. For the other cell rules obtained, the reduction in the RMS error does not seem to be significant. The stress plots for selected rules can be seen in Figures 3.46 to 3.51.

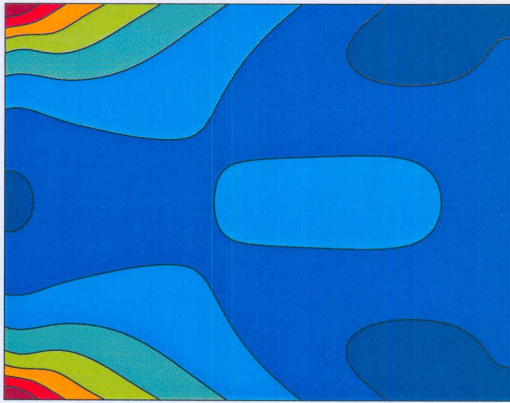


Figure 3.46: FEM  $16 \times 16$  stress plot

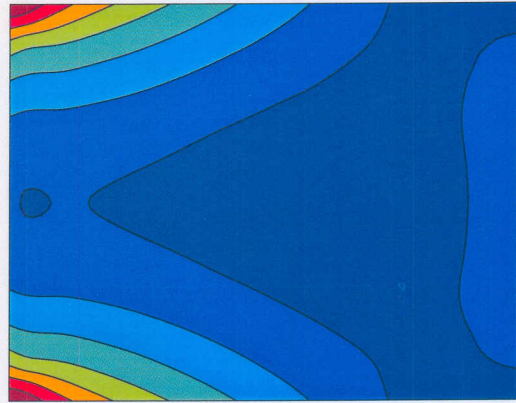


Figure 3.49: CA optimized 512 cell states

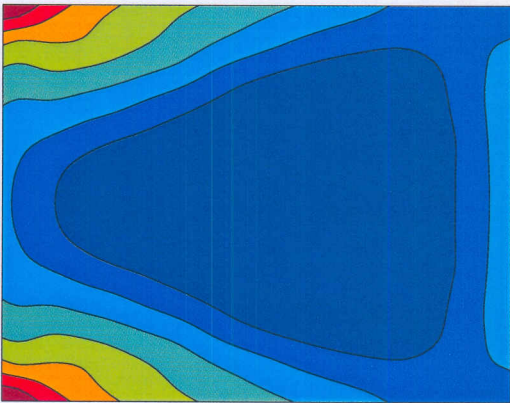


Figure 3.47: CA optimized 32 cell states



Figure 3.50: CA optimized 2048 cell states

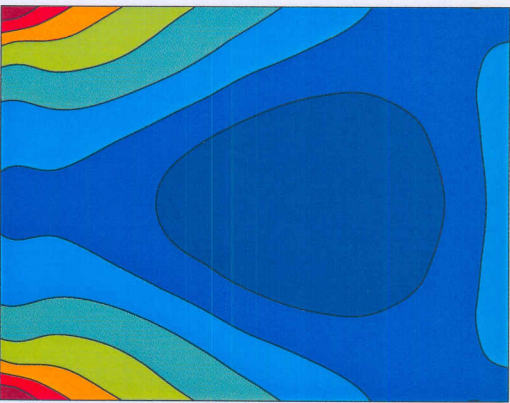


Figure 3.48: CA optimized 128 cell states



Figure 3.51: CA optimized 32768 cell states



### Asymmetric rule

We shall now attempt to extract the optimum cell rule using an asymmetric rule, where symmetry is not enforced, even though the problem studied is symmetric. Instead, we use asymmetry as a qualitative measure of the fitness of the rule derived. Owing to the large number of possibilities, the cell rule extraction is attempted with only 1024 cell states with the FEM  $8 \times 8$  model as learning basis. As a result the optimization problem becomes enormous and exploring all possibilities with all nine variables impossible. The GA was restarted a number of times and the best cell rules obtained were kept. The values of each cell position are shown in Figure 3.52. The best cell rules obtained for this problem are shown in Table 3.8.

f	b	h
c	p	d
e	a	g

Figure 3.52: Asymmetric neighborhood problem description

Rule number	RMS error	a	b	c	d	e	f	g	h	p
1	0.558182E-01	18	16	9	-1	-1	5	3	11	59
2	0.532882E-01	19	0	3	-14	-1	16	4	31	56
3	0.528102E-01	16	17	1	-14	0	13	7	21	59
4	0.474360E-01	30	-3	-18	5	4	32	-6	19	61
5	0.492669E-01	28	13	-1	18	2	24	-13	4	59
6	0.515053E-01	32	7	-1	20	-9	23	-17	11	63

Table 3.8: Optimized rules  $8 \times 8$  FEM problem 3 (asymmetric)

The rules deliver errors that are far lower than the best solution obtained for the symmetric cell rule. It is, however, very difficult to comprehend if they deliver any significant pattern in terms of how the rule develops. For this reason, the rule development is depicted in Figure 3.53.

The rules clearly follow no particular pattern. The RMS errors obtained are much lower than those obtained with the symmetric rule. The stress fields obtained are plotted in Figures 3.54 to 3.59. Even though the rules obtained are asymmetrical, the stress fields obtained are remarkably symmetrical. The symmetry can be used as a qualitative measure of the fitness of the rules, and the rules are superior to the rules found in Section 3.4.2.

5	16	11
9	59	-1
-1	18	3

Rule 1

16	0	31
3	56	-14
-1	19	4

Rule 2

13	17	21
1	59	-14
0	16	7

Rule 3

32	-3	19
-18	61	5
4	30	-6

Rule 4

24	13	4
-15	59	18
2	28	-13

Rule 5

23	7	11
-1	63	20
-9	32	-17

Rule 6

Figure 3.53: Asymmetric rule orientation for Problem 3 FEM  $8 \times 8$  1024 cell states



Figure 3.54: Asymmetric rule one

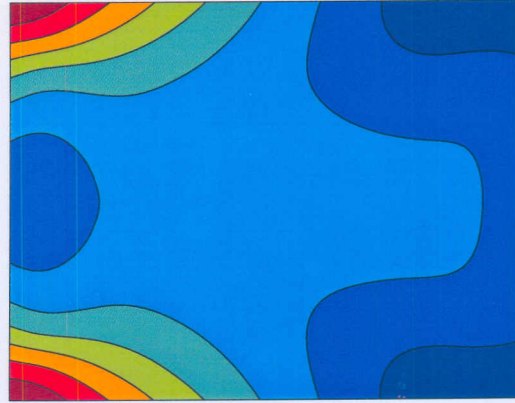


Figure 3.57: Asymmetric rule four



Figure 3.55: Asymmetric rule two

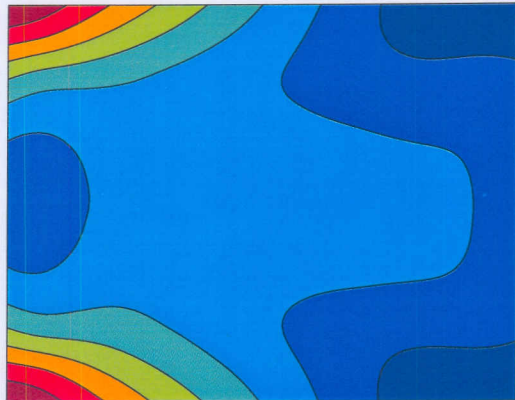


Figure 3.58: Asymmetric rule five

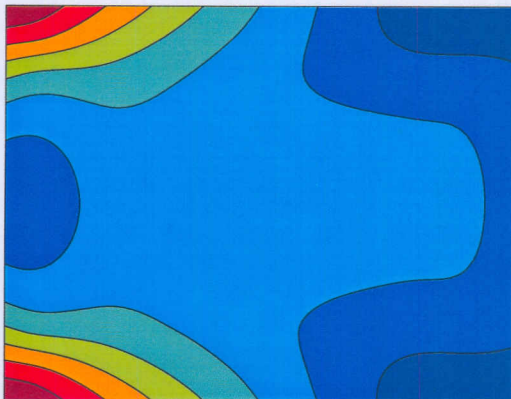


Figure 3.56: Asymmetric rule three



Figure 3.59: Asymmetric rule six

### 3.5 Cellular automata computational conclusions

The results obtained from using the CA simulations may be approached from different points of view. Each of the problems under investigation were capable of delivering a cell rule which is capable of approximating the state in which the structure found itself with relatively few cell states. Each of the rules obtained were more accurate than the corresponding finite difference equations. The use of a discrete number of cell states made the convergence criteria easy and allowed for a faster solution than continuous variables. By directly using the von Mises stress as the criteria to describe the state of the structure, we are able to develop a fast approximation ability which immediately provides enough information about the integrity of the structure. The ability to obtain the correct rule is of vital significance to the applicability of CA describing this natural system.

Although in each of the examples we were able to extract a cell rule in accordance with the neighborhood used that was able to surpass the performance of the FDM rules, not one of the rules were the same. Transferring the rules obtained from one problem to another yields very poor results. In fact, the rules obtained for the same problem where only the mesh was refined were not compatible at all. It is, however, clear that every problem has a cell rule which best described the stress field in that problem. This rule was usually obtained with a high number of cell states.

The lack of capability to obtain a universal cell rule indicates that the total formulation of the problem is not capable of capturing the governing aspects of the problems at hand. This failure in the CA simulation is not a failure of the CA itself but rather a failure in the formulation and approach for capturing the essence of the problem. With vast amounts of possible approaches, judging the CA capability using only one method would be unjustified. The results do indeed suggest that with more insight a cell rule construction might indeed be possible from the governing equation even if a complex criteria like the von Mises stress is used directly.

When CA rules and neighborhoods have developed far enough to allow for all classes of problems to be analyzed, the serious problem of obtaining the correct boundary values for CA simulations will require urgent attention. The current methods discussed in this thesis are more well known as numerical approximations. A number of other methods exist which can be used for this approximation. For instance, Victor Apanovitch [43] uses an external approximation which requires no mesh to obtain the solution. The approach of such methods lies with the mathematical formulation and assumptions made with such formulations. Underlying assumptions are already present in what are considered the governing equations in Appendix A. By using different assumptions in the understanding of the physics of the problem it might even be possible to formulate a mathematical model directed towards CA simulation.

# Chapter 4

## Parallel processing implementation

### 4.1 Introduction

#### 4.1.1 Parallel computing concepts

Parallel processing refers to the concept of speeding up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. A program being executed across  $N$  processors might execute  $N$  times faster than it would using a single processor [44]. The term parallelism denotes the possibility of executing several operations (instructions) simultaneously. Nowadays there are a considerable number of results on algorithms and parallel architectures, i.e. algorithms and architectures which make the parallel execution possible [45].

There is an intrinsic parallelism present in many computational problems. While it is sometimes clear that the solution of many problems can be easily obtained by simultaneously executing some stages of the solving process, often a simple conversion of sequential algorithms into parallel algorithms does not provide the maximum attainable parallelism for a given problem. In fact, it is not always possible to transform the best sequential algorithms into the best parallel algorithms, i.e. there is no trivial correspondence between sequential and parallel computing, and the existence of parallel computing environments, opens new horizons for scientific investigation [45].

The first parallel algorithms were developed at the beginning of the sixties, even though parallel architectures did not exist at that time. In that period, and in the following years, many researchers were fascinated by the challenge of solving problems assuming the existence of a parallel environment, without pondering on the applicability of their studies. This research, which later became important in practice, led to the first investigations into the characteristics and properties of parallel computing [45].

### 4.1.2 Technology

Technology has been growing very rapidly during recent years. Different types of parallel computers have been designed, starting from the vectorial computer, up to the multiprocessor and the distributed system. These new architectures belong to the very large class of parallel computers [45]. The design of parallel machines leads to a strong practical interest towards in the comprehension of all aspects of parallelism, with particular regard to the parallel solution of the main computational problems. Contemporary technologies make it very difficult to connect processors to one another completely and directly [46].

The advent of Very Large Scale Integration (VLSI) techniques has further augmented the interest towards parallelism from a different viewpoint: algorithms are designed in view of their hardware implementation. A very interesting feature of VLSI is its measure of “architectural complexity”, i.e. the circuit area. Another attractive characteristic of VLSI is the possibility of having a single circuit at reasonable costs and considerable number of computing elements cooperating for the execution of a given process. Finally, VLSI is also innovative from a theoretical viewpoint: we will see that in order to obtain an appropriate computing model from it, it is necessary to take into account the “geometric” structure of computations, in addition to their logical structure [45].

Recently there has been an increased interest in parallel processing in a Network of Workstations (NOW) as an alternative to Massive Parallel Processors (MPP) due to simplicity and cost effectiveness of such systems.

### 4.1.3 Classification of parallel machines

The terminology used in parallel computing lacks uniformity even when dealing with the basic issues. This is because a great number of ideas, concepts and methods differ both from a logical and computer architectural point of view when referring to the notion of parallelism. The characteristic which is common to all aspects of parallelism is the concept of concurrence, i.e. the simultaneous participation of entities in the quest towards a common goal [45].

Developing the notion of concurrency implies abandoning both the traditional machine model (Von Neumann computer) and the sequential nature of its algorithms. The problem then arises of characterizing new computing methodologies [45]. Already in 1966, some basic criteria were identified to classify parallel computing from a programmer’s point of view. Programming with a single instruction flow was called Single Instruction Multiple Data (SIMD), whereas programming with several instruction flows was called Multiple Instruction Multiple Data (MIMD). The SIMD and MIMD classes proved to be an important means of classification, since almost all parallel machines are actually included in them [45]. SIMD programming is also called synchronous parallel programming. There is a single program and all processors are constrained to execute the same instructions. The parallel flows which are automatically generated by an instruction on different data converge at the end of the instruction execution. In the case of MIMD parallelism, the programmer has the possibility of controlling many instruction (process) flows and many data flows. All processes can be made up of instructions belonging to different programs [45]. This high flexibility makes MIMD

parallelism more general than SIMD parallelism: a problem which is not characterized by a regular structure, but has a potential parallel exploitation, is appropriate for MIMD and not for SIMD. The greater flexibility must be paid for by the presence of problems which do not arise in the case of SIMD, in particular problems of data and process synchronization, as well as problems of allocation of processors to processes [45].

#### 4.1.4 Interconnecting structures and graphs

One of the main obstacles to the design of parallel computers resides in the difficulty of efficiently implementing hardware and/or software communication mechanisms among the different computing elements [45]. If all the elements are connected by a shared resource, this becomes the bottleneck of the system with consequently slower execution and worse performance. Hence, the interconnection problem should be addressed with the adoption of dedicated links.

In this framework, a parallel architecture can be seen as a set of processors connected by links for exchanging messages. In order to analyze the properties of interconnection structures, it is convenient to adopt the notation of a graph. In fact, it is possible to establish a correspondence between the nodes of a graph and computing elements, as well as between arcs and links. The use of this correspondence makes the study of the topological properties of the structure much easier. It follows that if the focus is on the geometry of the connections, a graph is a natural representation of the architecture [45].

#### 4.1.5 Parallel computing models and complexity measures

Computational complexity is a main field of computer science devoted to the determination of the minimal quantity of resources required to solve a computational problem. The investigation is usually carried out in two different ways:

- One method attempts to evaluate the quantity of resources necessary to solve a problem (complexity lower bound).
- The other method analyzes the way algorithms use their resources (complexity upper bound) [45].

If one wants to evaluate algorithm performance and compare it to the best possible performance, it is necessary to introduce computing models highlighting the fundamental resources and their relationships. It is then possible to define an optimum algorithm as the one making the minimum use of resources. The analyses carried out on the models give results that can be applied to the machines [45]. An apparent contradiction regarding the complexity of parallel computations is the fact that many computing models do not consider the costs due to data access and data exchange. Neglecting these costs is in certain cases an excessive simplification of reality. On the other hand, the lower complexity bounds obtained by neglecting these costs are of course still valid, when these costs are present [45]. It follows that the study and the analyses, even if performed on simplified models, give useful indications,

provided that a model expressing some of the particularities of parallel machines is used. For this reason, many complexity analyses are performed on models neglecting some costs [45]. One of the main objectives of computational complexity is to determine bounds to the cost required by computations, independently of technical details. Hence, it is necessary to define how to measure the performance of parallel algorithms independently of the specific architecture. We can introduce some of the main parallel computational models, (Boolean and arithmetic) circuits and parallel random access machines (PRAM), together with an “algorithmic” model, which is less formal than the preceding ones, but more useful to analyze the performance of algorithms. This parallel computing model has been widely used, especially for the early complexity analyses. The following rules describe the criteria to perform an analysis using such a model [45]:

- Any number of processors can be used at any time.
- Each processor can execute one (arithmetic or logic) operation in a unit of time.
- Data access has no cost.
- Communication among processors has no cost.

One of the most important aspects of computational models is their generality. The notion of a computational graph of a parallel algorithm is the unifying element of our approach [45]. In structural analysis, the above assumptions are reasonable if the cost of evaluating a function is orders of magnitude larger than the cost of communication. If the analyses is evaluated using FEM, the former is definitely true.

## 4.2 Parallel virtual machine, linux xpvm

### 4.2.1 Linux

*A quiet rebellion is under way in the world of computer operating systems and software. The rebels refer to themselves as the Open Source Movement. Their doctrine is called the “GNU Manifesto, ” a document available from the free software foundation in Boston and their standard is the Linux operating system [47].*

Linux is the free Unix written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers from across the Internet [47]. Linux aims towards POSIX compliance, and has all of the features one would expect of a modern, fully fledged Unix: true multitasking, virtual memory, shared libraries, demand loading, shared, copy-on-write executables, proper memory management and TCP/IP networking [48].

Linux runs mainly on 386/486/586-based PCs, using the hardware facilities of the 80386 processor family (TSS segments, and others) to implement these features [48]. Linux supports GCC, Emacs, the X Window System, all the standard Unix utilities, TCP/IP (including



SLIP and PPP), and all of the numerous programs that people have compiled or ported to it [48]. One might state that UNIX is like Linux. With the freely available source code of the kernel, every user trying to use this own OS (operating system) for a particular purpose is in fact a developer. This tendency causes the use and admiration for Linux to grow daily. Linux is an emerging technology that will affect us all [47].

### 4.2.2 Parallel virtual machine

Parallel Virtual Machine (PVM) is a software system that enables a collection of heterogeneous<sup>1</sup> computers to be used as a coherent and flexible concurrent computational resource [49]. The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations, that may be interconnected by a variety of networks, such as Ethernet or FDDI [49]. User programs may be written in C, C++ or FORTRAN and access PVM through library routines [49]. PVM is available as source code and can be installed on any operating system. The libraries handle the inter-communication and the user need only use these libraries for the sending and receiving of data. The programming can be split into two areas, the creation of the master program and slave programs. The master program controls each of the slave programs. Any number of slaves can be spawned on each machine and the results sent back to master after termination. One of the major advantages of PVM is that it uses an already available amount of resources to obtain the goal in a very effective manner. PVM is the most common NOW implementation in use today. The source code of PVM is granted under the GNU license which makes it distributable/available for everyone to use. In fact the PVM software comes as a default installation with most Linux distributions.

Below is a brief description of some aspects of PVM.

#### Heterogeneity

PVM supports heterogeneity at three levels:

- Applications, machines and networks. At the application level, sub-tasks can exploit the architecture best suited for them.
- At the machine level, computers with different data formats are supported, including serial, vector and parallel architectures.
- The virtual machine can be interconnected via different networks, at the network level.

Under PVM, a user-defined collection of computational resources can be dynamically configured to appear as one large distributed-memory computer, called a "virtual machine"

---

<sup>1</sup>Quality of being diverse and not comparable

## Computing model

PVM supports a straightforward message passing model. Using dedicated tools, one can automatically start up tasks on the virtual machine. A task, in this context, is a unit of computation, analogous to a UNIX process. PVM allows the tasks to communicate and synchronize with each other. By sending and receiving messages, multiple tasks of an application can co-operate to solve a problem in parallel. The model assumes that any task can send a message to any other PVM task, with no limit on the size or amount of the messages [50].

## Implementation

PVM is composed of two parts. The first is the library of PVM interface routines. These routines provide a set of primitives to perform invocation and termination of tasks, message transmission and reception, synchronization, broadcasts, mutual exclusion and shared memory. Application programs must be linked with this library to use PVM. The second part consists of supporting software that is executed on all the computers, and make up the virtual machine, called a “daemon”. These daemons interconnect with each other through the network. Each daemon is responsible for all the application components processes executing on its host. Thus, control is completely distributed, except for one master daemon.

Two crucial topics arise when discussing implementation issues: inter-process communications (IPC) and process control. These topics are discussed below [50].

## Inter-process communications

In PVM different daemons communicate via the network. PVM assumes existence of only unreliable, unsequenced, point-to-point data transfer facilities. Therefore, the required reliability as well as additional operations like broadcasts, are built into PVM, ontop the UDP protocol. For IPC, the data is routed via the daemons, e.g., when task A invokes a send operation, the data is transferred to the local daemon, which decodes the destination host and transfers the data to the destination daemon. This local daemon decodes the destination task and delivers the data. This protocol uses three data transfers, of which one is across the network. Alternatively, a direct-routing policy can be chosen (depending on available resources). In this policy, after the first communication instance between two tasks, the routing data is locally cached (at the task). Subsequent calls are performed directly according to this information. This way, the number of data transfers over the network is reduced to only one, over the network. Additional overheads are incurred by acknowledgment schemes and packing/unpacking operations [50].

## Process Control

Process control includes the policies and means by which PVM manages the assignment of tasks to processors and controls their execution. In PVM, the computational resources may be accessed by tasks using four different policies:

- A transparent-mode policy, in which sub-tasks are automatically assigned to available nodes.
- The architecture-dependent mode, in which the assignment policy of PVM is subject to specific architecture constraints.
- The machine-specific mode, in which a particular machine may be specified.
- A user's defined policy that can be "hooked" to PVM. Note that this last policy requires a good knowledge of the PVM internals.

PVM uses the transparent mode policy, by default. In this case, when a task initiation request is invoked, the local daemon determines a candidate pool of target nodes (from the nodes of the virtual machine), and selects the next node from this pool in a round-robin manner. The main implications of this policy are the inability of PVM to distinguish between machines of different speeds, and the fact that PVM ignores the load variations among the different nodes [50].

### Construction of a PVM cluster

In this study an 18 node Beowulf "Souper" computer is constructed using PCs that would otherwise have been disposed of. All of the computers used are installed with a Redhat 6.2 Linux distribution. A main server is set up to be in charge of all the major tasks. The server uses NFS (Network file system) to share the user home partitions and most of the user programs over the network. This means that each of the other nodes only needs a basic kernel to allow the rest of the file systems to be mounted over the network. Thus all changes in software only needs to be done on the server. Figure 4.1 shows the cluster constructed.

Installation and configuration of such a "Souper" computer cannot be seen as a trivial task. It is however, a necessary step to achieve a computational goal. The time spent constructing such a cluster in co-operation with the effort required for the parallel implementation of a program makes such implementations only viable if the same program can be re-used frequently for different problems. Once such a computing cluster has been installed, entirely new fields of exploration open up in computational mechanics. The gain in computational capabilities allows more traditional problems to be solved with higher accuracy. However, changes in software and programs require almost permanent administration. Details of the software implementation are best obtained by a study of the parallel programs depicted in Appendix C.

#### 4.2.3 Xpvm

XPVM is a graphical console and monitor for PVM. It provides a graphical interface to the PVM console commands and information, along with several animated views to monitor the execution of PVM programs. These views provide information about the interactions among tasks in a parallel PVM program, to assist in debugging and performance tuning [51].

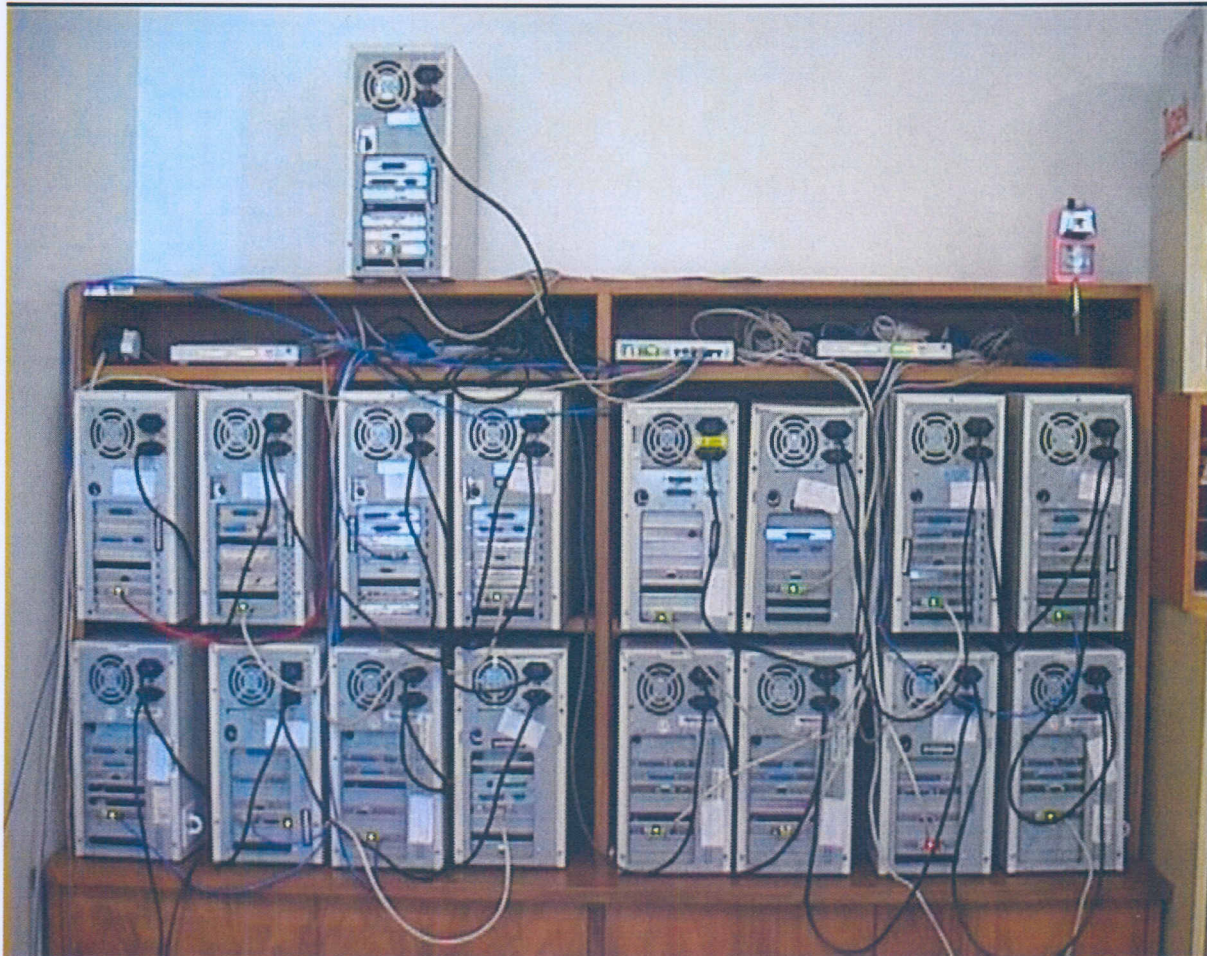


Figure 4.1: Photo of the 18 node Linux “Souper” Computer, named GOH, constructed in this study

To analyze a program using XPVM, a user need only compile his or her program using the PVM library, version 3.3 or later, which has been instrumented to capture tracing information at run-time. Then, any task spawned from XPVM will return trace event information, for analysis in real time, or for post-mortem playback from saved trace files [51]. Figure 4.2 shows a typical XPVM screenshot. However, care should be taken when using XPVM since the monitoring of the PVM program uses a large amount of computer resources which can slow down the execution of the program.

### **Benchmarking**

Benchmarking implies the measuring of the speed with which a computer system will execute a computing task, in a way that will allow comparison between different hardware or software combinations [52]. It does not involve user-friendliness, aesthetic or ergonomic considerations or any other subjective judgment [52].

Benchmarking is a tedious, repetitive task, and requires attention to details. Very often the



Figure 4.2: XPVM screen shot

results are not what one would expect, and are subject to interpretation (which actually may be the most important part of a benchmarking procedure) [52].

Benchmarking deals with facts and figures, not opinion or approximation [52]. The relative speed of computers can be determined by benchmarking different computer systems against one another for common tasks. When benchmarking programs, they should be run on the same system.

Benchmarking in a parallel environment becomes very intricate since none of the machines in the system can be exactly the same. Many more factors which can influence the speed of the program in a parallel environment than on a single machine and is thus very difficult to benchmark. The interpretation of the results becomes complicated and important. In parallel implementations important factors like data access cost, communication among processors and difference in speed of processors are very difficult too incorporate. Instead the standard idealizations of complexity measures allows the complexity upper and lower bounds to be determined with reasonable accuracy. Determination of complexity bounds gives a possible operational range rather than a specific performance feature for the implementation.

## 4.3 Parallelization of cellular automata

### 4.3.1 Parallelization method

#### Introduction

In this section we proceed to implement a CA program in parallel using PVM. The problem of parallel implementation is approached by first considering the new components required for a parallel implementation. We start with a simple parallel implementation. Doubtless superior implementations exist. At each stage of the parallel implementation our program is evaluated to ascertain possible performance increases obtainable in a PVM implementation.

We have already established that CA computations can become very expensive compared to traditional techniques as the size of the problem increases. Their discrete nature also allows an important analogy with digital computers: CA may be viewed as parallel-processing computers of simple construction [4]. Since each of the cells follow the same cell rule, the parallel architecture becomes a simple implementation of SIMD.

In Section 2.6 we saw that any dimension of CA can be mapped to a square lattice. Thus we only need to consider the implementation of a square computational lattice for a parallel implementation. We also introduced a novel internal computer representation in Section 2.10 which allows for easy interaction with currently available computing architectures.

CA seem to be ideally suited to parallel computations since every cell can be updated simultaneously. This, however, isn't an ideal implementation with conventional parallel computational architectures, since the updating of one cell will require information from more than one neighbor. Thus more data have to be exchanged than the number of computations performed per node. Implementing a parallel version using PVM implies that the data transfer would take place using network cards in each PC. Both 100Mb/s and 10Mb/s network cards will be used. The slower 10Mb/s card will create the biggest bottle neck in system, since the internal data transfer rate in each node (Bus speed) is much higher than the external transfer rate (network card). Additionally the calculation speed of each computer is much higher than the data transfer between its components. Thus, to create an effective parallel program, an attempt has to be made to find an optimum between the amount of data transferred and the actual computations performed. This complex optimum will depend on the number of nodes used in the PVM setup as well as speed difference between the nodes.

The methodology presented here explains how the computing takes place in terms of the physical position of the cell. The difference between this explanation and the internal representation described in Section 2.10 relates purely to indexing and is trivial to implement in such a setup. Since the goal of the implementation of the CA simulation is only to ascertain whether a parallel implementation using PVM would be able to deliver a performance increase, such a representation is excluded in this section.

### Parallel implementation data processing

Consider the simple two-dimensional lattice shown in Figure 4.3 with each cell given a number to indicate its position. We obtain the CA boundary conditions from an external method (FEM, BEM or known solutions) and by using initial cell states of zero we obtain an initial computational problem to be calculated with a parallel implementation.

The CA computational lattice can be divided into smaller sub-lattices in which each of the sub-lattices can be computed simultaneously. This means that the boundaries between the sub-lattices have to be exchanged to solve the initial problem. To prescribe boundary conditions for each sub-lattice, fixed boundaries have to be obtained from each neighboring sub-lattice. This results in a complicated setup as the boundaries “overlapping” (as seen in Figures 4.3, 4.4 and 4.5). By creating one “overlapping” boundary row we limit the neighborhood that can be used in our CA simulation to a Moore neighborhood. This does not create any accuracy concerns since the Moore neighborhood has already revealed acceptable accuracy (Section 3.4.2). This neighborhood is also an optimum for the amount of data exchanged.

We divide the lattice into sub-lattices, keeping the boundaries constant on each sub-lattice, to obtain the boundary values from their direct neighbors. If we presume a four sub-lattice split, as shown in Figure 4.4 we need to keep the boundaries of each sub-lattice fixed. Hence the only boundaries that are not updated are the original problem boundaries. This creates three “overlapping” boundaries for each sub-lattice where data has to be exchanged.

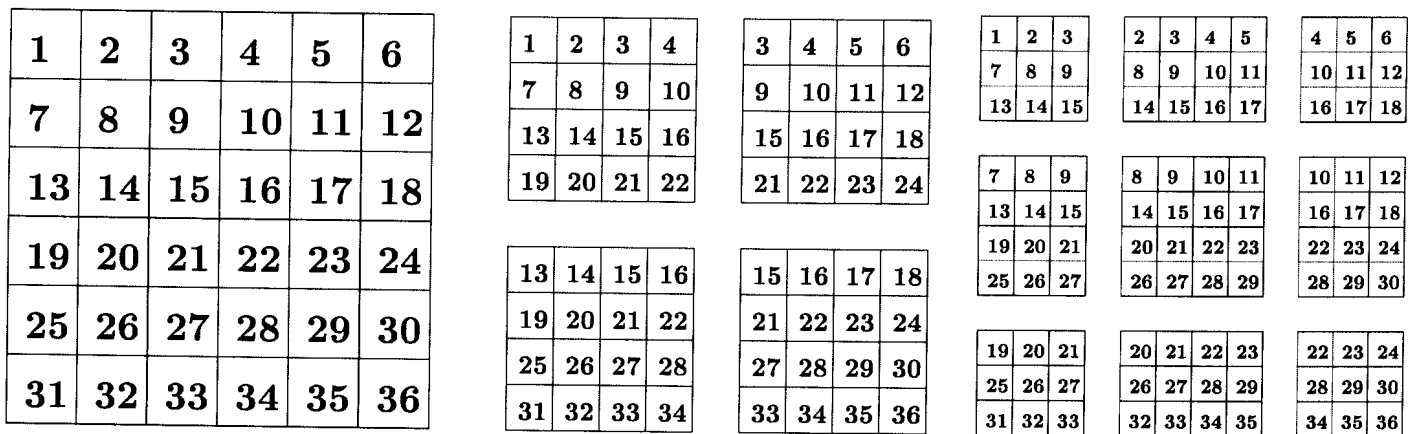


Figure 4.3: Original square lattice    Figure 4.4: Four sub-lattice split    Figure 4.5: Nine sub-lattice split

This simple principle can be expanded to more possibilities. We also consider the problem in Figure 4.3 divided into nine sub-lattices. In this case it is clearly not possible to divide the original problem into equally sized sub-lattices. It can, however, be set up as nine computational sub-lattices with different sizes. Figure 4.5 shows one such a possible setup. This example illustrates a few basic logical principles, namely:

- The more sub-lattices that are created, the more overlapping layers are created. This results in more data that have to be exchanged. It also causes the total amount of data in the computation to increase.

- The overlapping layers are not the same for each sub-lattice. The center lattice exchanges data with eight neighbors, the corner lattices exchange data with only three neighbors while the lattices on the sides connect to five neighbors. Obviously, each one connects to different neighbors.
- The different sized sub-problems have different calculation needs. To obtain artificial load balancing between the computing nodes, the more computational intensive problems should be sent to faster computing nodes in the cluster.

### 4.3.2 Sequential sub-lattice update method

It is clear from the basic description in Figure 4.5 that the implementation of the sub-lattices for computation is not a trivial task. This manipulation of the original data is generally referred to as pre-processing, while combining all the data after processing is referred to as post-processing. Two trains of thought exist regarding parallel computation. The one point of view is that pre- and post-processing of the data do not involve the actual computation and should therefore be neglected when compared to the speed of a single machine. A second viewpoint holds that, since the pre- and post-processing is only necessary when you want to do parallel computations, the cost must be reported when comparing speed.

In order to determine the difference between the two methods and to ensure that the pre- and post-processing is performed well, a program is first written to perform the pre- and post-processing on a single system. By doing this testing, the implementation is simplified, while simultaneously ascertaining the influence. It is also possible to see the influence the pre- and post-processing have on the CA evolution and the total computational time required.

The system used for the bench marking is an Intel PII 300MHz PC running RedHat Linux 6.2. The operating system comes with a utility that is very useful when doing these tests: by combining the binary with the time program through the command “time -v binary”, all the data regarding the process can be obtained. For example

```
Command being timed: "t"
User time (seconds): 19.84
System time (seconds): 0.04
Percent of CPU this job got: 96%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:20.64
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 0
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 130
Minor (reclaiming a frame) page faults: 232
Voluntary context switches: 0
Involuntary context switches: 0
```



```
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

In this case we are interested in the user time of the process. The system time reported is the time the process uses to communicate with other parts of the computer. The wall clock time is the total time elapsed since the beginning of the process has started. By using the user time on the same computer, it is easy to see whether or not the pre- and post-processing have any significant influence on the processing time. Table 4.1 shows the time comparisons for the setup used. Some interesting results are obtained. The table shows two columns for each number of cells, the task time in seconds and the number of iterations it takes for the solution to converge.

N sub-lattices	40000 cells		4096 cells		1024 cells		256 cells	
	time (s)	iterations	time (s)	iterations	time (s)	iterations	time (s)	iterations
1	880.39	16789	19.88	3937	1.52	1216	0.10	349
4	854.14	16771	20.36	3894	1.60	1182	0.14	328
9	851.18	16638	21.07	3839	1.74	1159	0.15	315
16	874.17	16770	22.25	3836	1.90	1145	0.18	309
25	877.23	16557	23.29	3836	2.05	1117	0.21	291

Table 4.1: Sequential block update method with 32768 cell states and finite difference rule(N, Number of sub-lattices)

We notice that as the number of sub-lattices increases, the number of iterations required for convergence decreases. To understand why this happens we need to return to Figure 4.5. The first sub-lattice to be updated will be the one in the top left corner. Once this sub-lattice has performed its update, the lattice to the right will start its update, but its left boundary will already contain an updated boundary, and so on. In this manner, with the exception of the first block, all the other blocks will receive either one, two or three updated boundaries which cause the solution to reach a stable state in fewer iterations. This is very similar to the FDM methods of successive approximation to increase convergence rate as discussed in Section A.2.3. Since we are only interested in the final value, it is advantageous to reach the solution in fewer steps.

Consider the example in which we used the greatest number of cells (40000). We note the interesting trend: as mentioned previously the processing time actually decreases slightly and then increases again as the number of sub-lattices increase. The minimum processing time is required at nine sub-lattices. This happens even though the number of iterations actually keeps on decreasing after nine sub-lattices. It can be expected that the processing time is linearly dependent on the amount of iterations as shown in Figure 3.2. This trend illustrates

how the increase in the total amount of data in the system increases the computational need in the parallel methodology.

This implementation is still extremely ineffective since it combines and splits the lattice after every iteration. But it is sufficient to prove that the effect of pre- and post-processing are negligible in CA computations.

### 4.3.3 Parallel lattice computation

To introduce real parallel processing we need to expand our implementation to use the PVM libraries to create different programs from our existing code. The first step in using CA in parallel processing is to implement a basic program which can calculate the correct result using separate tasks. The first iteration involves the data being split up and sent to the different processors (slaves) to perform one iteration. The data is then sent back to the master where the data is combined and then split up again. The slaves receive the new problem and perform the calculations. By doing the implementation in this manner we implement the sub-lattice boundary exchanges in a very easy way. This method proved to be very ineffective since the amount of data that has to be transferred between machines is too large for effective computations.

Since the slowest part of the PVM setup is the network connection, the optimum way of improving the performance of the system is to decrease the amount of data that has to be transferred, for a fixed amount of processing. The CA approximation for a linear elastostatic analysis will always have to converge to a fixed state placing the simulation in *class 2*. So a first attempt to improve the ratio is to allow each sub-lattice to converge on its own (local convergence) before boundary exchanges take place. This will reduce the total number of global iterations and thus the number of data exchanges. With the CA solution we are studying, we are only interested in the final state, and the evolution of our CA with local convergence will be different. We only require that this evolution yield the same answer at the final convergence. Table 4.2 indicates that even for a small problem, we obtain a speed-up factor of roughly 2 in this fashion.

Number of cells	Single iteration time(s)	Local convergence	
		Iterations	time(s)
$8 \times 8$	34.48	40	16.21
$16 \times 16$	117.81	130	37.81

Table 4.2: Local convergence speed improvement for 4 sub-lattices

Although the software created is capable of performing the parallel computation correctly, it is by no means optimized since the data is first sent to the master before being communicated to the correct slave. Although this helps to simplify programming, it does not give any performance gain in the parallel computation. In order to see if parallel computations with PVM would yield any significant performance gain, the parallel implementation has to be modified.

Only the necessary data has to be communicated between slaves and the communication has to be performed with direct slave communication. Implementing a method which would involve inter-slave communication, needs the calculation of the direct neighbors to and from which data has to be sent and received. Each sub-lattice has a maximum of eight neighboring blocks, four sides of which each has to send and receive an array of data and four corner points which is only a single value. Sending and receiving data between the master and the slave programs when one big chunk of data is sent for computation already requires complicated indexing. Allowing each slave to calculate how many different chunks of data had to be sent and received as well as the position of each data set causes complex problems. To solve this difficulty a flag is set for each data position. This flag is sent with the data so that the receiving slave knows where the amount of data is going to be received and where it had to be placed. This method saves greatly on data transfer since the data is sent directly to the sub-lattice that requires it and not first via the master program. The implementation of such a direct communication is by no means trivial. Although it has the potential to greatly improve performance, it complicates the coding enormously. In fact, the PVM portion of the program became the major part of the program.

Benchmarking on a single machine is already a very controversial issue. It cannot be seen as a simple measurement to compare the performance of a single machine with that of multiple machines with different hardware. All the machines in the cluster consist of different configurations. The only real concept that can be gained from parallel benchmarking is if it is possible to gain a speed improvement through parallel implementation of an algorithm (to gain perspective into the complexity upper bound). It is common in the determination of a systems complexity bounds of a system to use simplified models. It follows that the study and the analysis, even if performed on simplified models, gives useful indications, provided that a model expressing some of the particularities of parallel machines are used [45].

One way in which this can be done on our program in a very idealistic manner is to spawn more than one slave on the same machine and check the user time of each slave (the actual computation time for the slave) on the machine to determine an idealistic time that the process would take on equivalent machines with no external network traffic. The algorithm is tested on various problems and the results are shown in Table 4.3. The speed-up factor is defined as the time used to solve a single problem divided by the time of the multiple sub-lattice problem.

$8 \times 8$					
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor	
1	1	1.93	0.006	1.00	
4	21	1.93	0.172	0.04	
9	26	1.93	0.276	0.02	
16	32	1.93	0.463	0.01	
$16 \times 16$					
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor	
1	1	2.74	0.503	1.00	
4	35	2.74	0.296	1.70	
9	51	2.74	0.471	1.07	

16	58	2.74	0.856	0.59
<hr/>				
32 × 32				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	2.99	1.00	
4	51	2.99	0.519	1.92
9	67	2.99	0.689	1.45
16	78	2.99	1.173	0.85
<hr/>				
64 × 64				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	2.69	1.503	1.00
4	38	2.69	0.764	1.97
9	49	2.69	0.679	2.21
16	70	2.69	1.231	1.22
<hr/>				
128 × 128				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	2.21	8.641	1.00
4	29	2.21	2.543	3.40
9	43	2.21	1.096	7.88
16	33	2.21	0.773	11.18
<hr/>				
180 × 180				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	1.98	12.666	1.00
4	37	1.98	6.033	2.10
9	54	1.98	2.556	4.96
16	41	1.98	1.293	9.80
<hr/>				
256 × 256				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	1.74	73.418	1.00
4	48	1.74	18.898	3.89
9	58	1.74	6.893	10.65
16	56	1.74	2.965	24.76
<hr/>				
400 × 400				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	1.46	339.141	1.00
4	62	1.46	103.431	3.28
9	69	1.46	31.546	10.75
16	73	1.46	10.692	31.72
<hr/>				
450 × 450				
Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	1.39	286.200	1.00
4	66	1.39	107.253	2.67
9	73	1.39	33.712	8.49
16	73	1.46	16.191	17.68
<hr/>				
512 × 512				

Number of sub-lattices	Iterations global	RMS error	Time (s)	Speed-up factor
1	1	1.74	635.810	1.00
4	71	1.74	200.787	3.17
9	77	1.74	63.048	10.08
16	80	1.74	22.309	28.50

Table 4.3: Parallel computing speed performance of CA with PVM

Spawning all the slaves on the same machine, however, does not reflect the true nature of the problem since the network connection between machines which causes the biggest bottle neck in the whole system, is not accounted for. It does, however, give us insight into the approximate complexity upper bound and the evolution of the parallel implementation methodology. We notice that on smaller mesh sizes, there is no gain in parallel processing speed of the solution. We only start seeing a significant speed improvement at a mesh size of  $128 \times 128$ .

For more sub-lattices with bigger mesh sizes, the speed-up factor seems to become greater than the number of sub-lattices.

For a mesh of  $256 \times 256$  with 16 sub-lattices we obtain a speed improvement of 24. This seems impossible, supposing that we have a process that takes a certain time  $x$  on a single processor. It is clear that if we have  $n$  of the same processors the same number of computations cannot be performed in less than  $\frac{x}{n}$  of the time. In our definition of a speed-up factor for 16 processors we cannot obtain a speed-up factor larger than 16. But on each of our larger problems we obtain values greater than this limit. We should, however, be aware that we are not comparing the same problem. Each sub-lattice division of the problem has a different evolution, although their evolutions lead to the same final state.

Let us consider the factors that cause this phenomenon. The more sub-lattices we have, the more data has to be exchanged per global iteration. The number of global iterations also increases as the amount of sub-lattices increases, since the evolution of the CA is very dependent on the boundary conditions supplied to the problem. Subsequently each sub-lattice obtains boundary conditions at every global iteration, which causes the sub-problem to evolve in a very different way than it would in the single problem. The parallel implementation creates more complicated tendencies as seen in Figure 4.6.

In Figure 4.6 we see that in the first iteration two of the sub-lattices converge very quickly while the other two sub-lattices take much longer to converge. It is clear through the whole simulation that only two of the sub-lattices keep the other processes waiting. (One sub-lattice requires the largest amount of computations.) This process keeps all other processes waiting and thus determines the speed of the whole system. We note the difference in computational requirements for the different sub-lattices.

Running all the processes on the same machine implies that as soon as one slave has stopped processing, the other slaves take up a larger percentage of the central processing unit (CPU). The change in CPU usage causes a small error in the CPU time reported, since the program keeping track of the process only updates at discrete intervals. Investigating the total impact



Figure 4.6: Convergence of four sub-lattices (green=processing; white=waiting; red=data communication)

of this error is not possible and also unnecessary, because it only impacts on our already idealized system.

There is a clear pattern that can be observed: as the problem gets closer to convergence, the time between data transfers decreases. By measuring this time one would be able to determine if the problem is converging or diverging. The closer the problem gets to convergence the more frequently data transfer takes place, thereby reducing the computational efficiency of the problem.

## 4.4 Parallelization of a genetic algorithm

### 4.4.1 Introduction

Parallelization of a GA enables us to extract cell rules for larger mesh sizes of the CA. Since the function evaluations become very expensive for large mesh sizes, it allows more than one member of the population to be evaluated simultaneously and thus reduces the total time required for the optimization. The implementation of the GA has a further advantage in that the only data that has to be sent over the network is the cell rule and the function value. The fact that only a small amount of data has to be sent forth makes the implementation of the GA in PVM very attractive.

### 4.4.2 Implementation method

In the CA parallel program, the master is only used to perform the pre- and post-processing of the problem. The master program in the GA performs most of the GA calculations and the slaves are only involved in performing the function evaluations (CA). For big mesh sizes, the function evaluation becomes the most processor intensive part of the calculation. The master initializes each of the slaves (population member) with the whole problem and solution. From this point onwards, the GA (master) performs the usual calculations and sends only the rule to evaluate to each of the slaves. The slave uses the CA rule for the evolution of the problem and once it has converged, compares it to the prescribed solution

and calculates a RMS error which is returned back to the master. This process continues until the maximum number of generations is reached.

### Practical Implications

The nature of the problem that is being optimized is such that some CA rules will converge very quickly, some take long, while others may diverge. Since the optimization is performed using discrete variables, there is no way of predicting the outcome and the simulation must be performed. This means that the master program still has to wait for all slaves to finish before it can calculate the values for the next population.

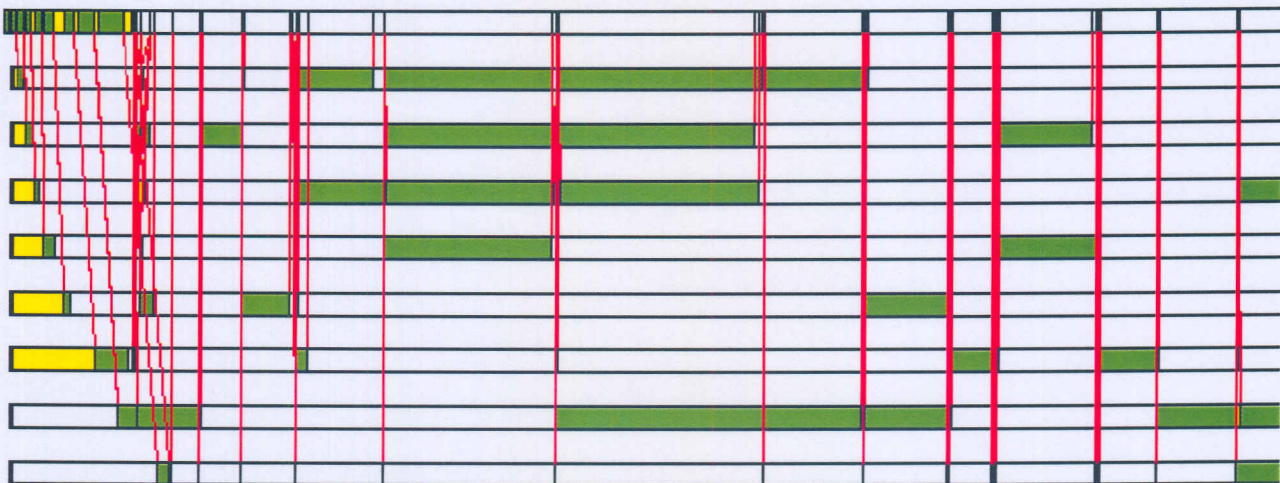


Figure 4.7: Calculation with nine population members (green=processing; white=waiting; red=data communication)

Figure 4.7 shows a typical performance for the parallel GA. Although it is clear that the data transfer is immediate, it is also clear that one or two processes keep the others waiting. To obtain an idea of what possible speed improvement can be obtained, think of the time it takes for the slowest machine on the PVM network to reach the maximum number of iterations that is required for divergence and multiply this time with the number of generations. The result indicates the longest time the PVM program could run. Since the values the GA assigns to the slaves to evaluate is based on random numbers, it is not possible to run a benchmark on the program as was done with the CA. It is, however, clear in finding optimum solutions that this method saves time. By trial and error it is found that the program delivers relatively good results when approximately three slaves are spawned on the same machine, thus setting the population size to three times the number of machines used. This works relatively well since it rarely happens that more than one slave on the same machine diverges and all the processors are used for almost the whole run.

In actual fact, by changing the number of population members the complexity bounds are adjusted. If we increase the population size we increase the amount of computations required per generation, thereby increasing the complexity lower bound. By not being able to

determine the complexity upper bound and just increasing the lower bound we increase the average of the operational area and thereby gain more performance from the entire system.

## 4.5 Computational conclusions

The basic principles of, and the gains that can be derived from, parallel processing are clear and transparent. However, the determination of optimum configurations in a parallel environment are not so simple. One may in fact consider this problem as complex and quite demanding. The existing technologies that allow for different parallel processing capabilities are diverse, and all subject to different limitations.

In this chapter, the possibilities of parallel processing were investigated using a simple, inexpensive setup, consisting of a cluster of personal computers (PCs) using freely available software. The free standing computers in the cluster contribute processing power to an infrastructure commonly known as a parallel virtual machine (PVM).

While the parallel implementations considered herein are not necessarily optimal, it suffices to demonstrate that a cluster of machines set up from obsolete resources allows for the calculation of solutions faster than is possible on unconnected, free standing resources. Since the cluster can be constructed with no software implementations whatsoever, it makes such a setup ideal for large computational problems when little financial resources are available.

For the parallel implementation of the CA considered in this chapter, a notable performance increase is demonstrated for both the sequential sub-lattice update and the parallel sub-lattice update methods, which are both a direct consequence of the inherent parallelism of CA. This creates the possibility of additional system performance increase by combining these two methods. This combination has the possibility of being quite flexible, since it would allow faster computers to process more than one sub-lattice, while waiting for slower computers to “catch up”. The coding involved in such a method is, however, complex and involves the continuous monitoring of each machine in the PVM to allow the master to determine how the calculation is to be distributed over all the available slaves.

It is demonstrated herein that the increase in computational efficiency when simulating CA in parallel does not necessarily yield a monotone increase in gain as the number of sub lattices increases. While this complicates the determination of the optimal division beforehand, it is nevertheless an attractive method to reduce the computational effort associated with machine learning in CA.

In addition, it is shown that parallel implementation of a GA allows for improved solutions in deriving the CA cell rule. Solutions are obtained in a shorter time span, even though the parallel configuration is once again not necessarily optimal.

The parallel GA implementation illustrates the possible benefits of parallel processing in numerical optimization. If the problem (viz. a function evaluation) is evaluated directly by each node, the data transfer between nodes becomes minimal. For computationally expensive function evaluations this results in a fully utilized cluster.



# Chapter 5

## Concluding Remarks

In this thesis, the suitability of using CA in structural analysis is investigated. In addition, simple approaches for the parallel execution of CA in structural analysis are implemented to evaluate the possible benefits of distributed computing for CA. Machine learning using a distributed genetic algorithm is used to determine the optimum cell rules for the CA, using finite element, boundary element and analytical approximations as the basis for the machine learning.

### 5.1 Summary of contributions

Firstly, the properties of CA are studied to develop a perspective on the possibilities and limitations of this idealization in structural analysis. The CA is then applied to some simple two-dimensional problems in structural analysis. An efficient formulation for the internal representation of the CA parameters is proposed, which provides for the future modeling of irregular geometries.

Machine learning, similar to the approach proposed by Hajela and Kim, is then used to derive the optimum rules for the CA. It is demonstrated that CA are capable of predicting reasonable approximations to problems in structural analysis, using few discrete elements. The computing cost requirement seems competitive with approximate methods which are currently popular, e.g. the finite element and boundary element method.

However, it is shown that the solutions obtained are not universally applicable, since they vary from problem to problem. Put differently: CA fail to capture the governing equations of structural mechanics, and optimal cell rules have to be determined for each problem studied. Nevertheless, the extraction of the “laws of nature” for a specific problem gives insight in how the CA parameters influence the obtained solutions.

In addition, it is demonstrated that the optimum cell rule is dependent on the number of cell states used in the system. Hence optimal CA cell rules cannot be transferred between meshes of different degrees of refinement for a given problem. Also, the number of cell states used was found to be of lower importance than the total number of cells in the system.

The Moore neighborhood with  $r = 1$  is proposed as the optimum cell representation in struc-

tural analysis, since it uses information from all the adjacent neighbors, without requiring an overly complicated boundary cell description. This neighborhood also allows for a simple parallel implementation, resulting in notable speed-up factors.

Furthermore, symmetric problems in structural analysis are analyzed using asymmetric rules in the machine learning process, where the symmetry of the solution found is used as a quantitative indication of the quality of the solution. It is demonstrated that the quality of the asymmetric rules is superior to the quality of symmetric rules, even for those problems that are symmetric in nature.

A parallel computing infrastructure was constructed, by combining 18 obsolete Pentium computers in a single cluster. The software used in assembling the cluster is in the public domain, and is available free of charge.

Finally, exploiting the inherent parallelism of CA, it is shown that distributed computing greatly improves the efficiency of the CA simulation, even though the speed-up factor is not necessarily proportional to the number of sub lattices used.

## 5.2 Conclusions

While CA are recent additions to the ‘tools’ used in structural analysis, increased use of CA as distributed computing becomes more widely available is envisaged, even though the CA rules are at this stage not transferable between different problems or even between meshes of varying refinement for a given problem. However, CA remain attractive due to their capability to reveal complex behavior and their ease of interaction on modern computing systems.

CA allow for computations in a discrete system to be performed in parallel in a natural way. By performing computations in parallel on different machines, the speed of execution of a simulation can be improved dramatically. This provides for the continual increase in the maximum size of a structure that can be solved. The implementation on distributed systems is not ideal. However, as advanced technology becomes available that allows thousands of processing units in a single computational device, CA will become increasingly suited to parallel computations.

One can even visualize computers designed specifically for CA simulations, based on RISC-like (Reduced Instruction Set Computing) processors. Such “computers” will exhibit great performance gains since they will only require a small instruction set on the CPU itself. Using a small instruction set (the cell rule in CA) will result in extremely fast computers.

## 5.3 Directions for future work

The following points deserve attention in studies following on this work:

1. Since prediction of good cell rules for CA in structural analysis remains the key issue, increased efficiency and effectiveness in the cell rule learning process is desirable.

2. The use of artificial intelligence in deriving CA cell rules, e.g. using neural networks, is of interest.
3. The efficiency of the simple approach to parallel implementation used herein should be increased.
4. A study of the effect of increased dimensionality on CA simulations is of interest.
5. Finally, CA have been proved to be capable of simulating biological growth. When designing cell rules which allow for material growth in areas of high stress, and conversely, the removal of material in regions of low stress, CA seem a natural candidate for applications in topology optimization.

# Bibliography

- [1] S. Wolfram. Cellular automaton supercomputing. *High-Speed Computing: Scientific Applications and Algorithm Design*, 1988.
- [2] S. Wolfram. Approaches to complexity in engineering. *Physica D*, 22:385–399, October 1986.
- [3] S. Wolfram. Cellular automata as models of complexity. *Nature*, pages 419–424, October 1984.
- [4] S. Wolfram. Cellular automata. *Los Alamos Science*, 9:2–21, Fall 1983.
- [5] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644, July 1983.
- [6] P. M. Juhl. *The Boundary Element Method for Sound Field Calculations*. PhD thesis, Technical University of Denmark, 1993.
- [7] S. Wolfram. Minimal cellular automata approximations to continuum systems. *Originally Presented at Cellular Automata '86*, June 1986.
- [8] A. Sameh. Computational science and engineering. *ACM Computing surveys*, 28(4):810–817, 1996.
- [9] G. Spezzano and D. Talia. Programming cellular automata algorithms on parallel computers. *Future generation computing systems*, 16:203–216, 1999.
- [10] J. von Neumann. *Theory of self reproducing automata*. University of Illinois Press, Champaign. IL, 1966.
- [11] H. Gutowitz. Frequently asked questions about cellular automata. <http://alife.santafe.edu/alife/topics/cas/ca-faq/ca-faq.html>, June 1996.
- [12] J. R. Weimar. Simulations with cellular automata. <http://www.tu-bs.de/institute/WiR/weimar/ZAscript/ZAscript.html>, June 1996.
- [13] M. Gardner. The fantastic combinations of john conway’s new solitaire game “life”. *Scientific American*, pages 120–123, October 1970.
- [14] M. Gardner. On cellular automata , self-reproduction, the garden of eden and the game “life”. *Scientific American*, (224(2)):112–117, February 1971.

- [15] Kari Eloranta. Is it alive or is it a cellular automaton?
- [16] E. R. Berlekamp, J. H. Conway, and R. K. Guy. Winning ways for your mathematical plays, 1982.
- [17] J. C. P. Miller. Periodic forests of stunted trees. *Phlos Trans. R. Soc. London*, A 266:63, 1970.
- [18] J. C. P. Miller. Periodic forests of stunted trees. *Phlos Trans. R. Soc. London*, A 293:48, 1980.
- [19] H. G. ApSimon. Periodic forests whose largest clearings are size 3. *Proc. R. Soc. London*, A 266:113, 1970.
- [20] H. G. ApSimon. Periodic forests whose largest clearings are size  $\geq 4$ . *Proc. R. Soc. London*, A 319:399, 1970.
- [21] C. Sutton. Forests and numbers and thinking backwards. *New Science*, 90:209, 1981.
- [22] M. Mitchell. An introduction to genetic algorithms. MIT Press, 1996.
- [23] A. Scatten. Cellular automata.  
<http://www.ifs.tuwien.ac.at/ascatt/info/ca/ca.html>, October 1999.
- [24] R. Kapral. Discrete models for chemically reacting systems. *Journal of mathematical chemistry*, 6:113–163, 1991.
- [25] R. Kapral. Chemical waves and coupled map lattices. *Theory and Applications of Coupled Map Lattices*, pages 135–168, 1993.
- [26] S. Wolfram. Univerality and complexity in cellular automata. *Physica D*, pages 1–35, January 1984.
- [27] F. S. Beckman. Mathematical foundations of programming, 1980.
- [28] G. D. Smith, R. F. Churchhouse, and A. Tayler. *Numerical Solutions of Partial Differential Equations: Finite Diffrence Methods*. Oxford University Press, 1985.
- [29] H. Abelson and A. A. diSessa. Turtle geometry : The computer as a medium for exploring mathematics. *MIT Press*, 1981.
- [30] S. Wolfram, O. Martin, and A. M. Odlyzko. Algebraic properties of cellular automata. *Communications in Mathematical Physics*, 93:219–258, March 1984.
- [31] T. Toffoli. Cellular automata mechanics. Technical report, University of Michigan, 1977.
- [32] E. Fredkin and T. Toffoli. Conservative logic. *International journal of theoretical physics*, 21:219–253, 1982.

- [33] T. Toffoli. Cellular automata as an alternative to (rather than approximation of) differential equation in modeling physics. *Physica D*, 10:117–127, 1984.
- [34] J. Hardy, O. De Pazzis, and Y. Pomeau. Molecular dynamics of classical lattice gas. *Physical review A*, 13:1949–1960, 1976.
- [35] U. Frish, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier-stokes equation. *Physical review letters*, 56:1505–1508, 1986.
- [36] Vichniac G. Simulating physics with cellular automata. *Physica D*, 10:96–115, 1984.
- [37] C. Bennett and G. Grinstein. Role of irreversibility in stabilizing complex and nonergodic behavior in locally interacting discrete systems. *Physical review letters*, 55:657–660, 1985.
- [38] P. Hunter and A. Pullan. Fem/bem notes. Department of engineering Science. The University of Auckland New Zealand.
- [39] P. Hajela and B. Kim. Ga based learning in cellular automata models for structural analysis. In *Proc. Third World Congress of Structural and Multidisciplinary Optimization*, Niagara Falls, N.Y., U.S.A., May 1999. Paper no. 13-GAM1-2.
- [40] R. C. Tirupathi and D. B. Askok. *Introduction to Finite Elements in Engineering*. Prentice Hall, 1997.
- [41] J. H. Mathews. *Numerical Methods for Mathematicians, science and engineering*. Prentice Hall International editions, 1992.
- [42] J. M. Gere and S. P. Timoshenko. *Mechanics of Materials*. Chapman Hall, 1991.
- [43] V. Apanovitch. *Procision: A technical overview*. PhD thesis.
- [44] H. Dietz. Linux parallel processing howto. January 1998.
- [45] B. Codenotti and M. Leoncini. *Introduction to Parallel Processing*. Addison-Wesley Publishing Company, 1993.
- [46] W. Handler. Multiprocessor arrays: Topology, efficiency and fault-tolerance. *Lecture Notes in Computer Science*, 1988.
- [47] M. J. Rocchetti. The quiet rebellion. *Mechanical Engineering*, 122(9):86–88, September 2000.
- [48] M. Welsh. Linux frequently asked questions with answers. Linux-FAQ.
- [49] J. J. Dongarra, G. E. Fagg, G. A. Geist, J. A. Kohl, R. J. Manchek, P. Mucci, P. M. Papadopoulos, S. L. Scott, and V. S. Sunderam. Pvm version 3.4: Parallel virtual machine system. University of Tennessee, Knoxville TN Oak Ridge National Laboratory, Oak Ridge TN Emory University, Atlanta GA.

- [50] A. Barak, A. Braverman, A. Gilderman, and O. Laden. Performance of pvm with mosix preemptive process migration scheme. In *7th annual Israeli conference on computer systems and software engineering*, June 1996.
- [51] J. Kohl. Xpvm. XPVM Readme.
- [52] A.D. Balsa. Linux benchmarking howto. August 1997.
- [53] S. P. Timoshenko and J. N. Goodier. *Theory of Elasticity*. McGRAW-HILL Engineering Mechanics Series, 1970.
- [54] C.A. Brebbia and J. Dominguez. *Boundary Elements An Introductory Course*. Computational Mechanics Publication, 1989.
- [55] Wilson E. Fem class notes.
- [56] K. J. Bathe. *Finite Element Procedures*. Prentice Hall, 1996.
- [57] Y. W. Kwon and H. Bang. *The Finite Element Method using Matlab*. CRC press, 1997.
- [58] S. L. Crouch and A. M. Starfield. *Boundary Element Methods In Solid Mechanics*. George Allen & Uwin (Publishers), 1983.
- [59] B. Gucun. Bem for structural analysis.  
<http://cavity.ce.utexas.edu/kinnas/papers/bem/node7.html>, 1997.
- [60] C. A. Brebbia, J. C. F. Telles, and L. C. Wrobel. Boundary element techniques. *Springer-Verlag Berlin Heidelberg New York*, 1984.
- [61] J. Holland. Adaption in natural and artificial systems. Ann Arbor.
- [62] C.R. Houck, J.A. Joines, and M.G. Kay. A genetic algorithm for function optimization:aa matlab implementation. Technical report, North Carolina State University.
- [63] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, MA, 1989.

# Appendix A

## Established structural approximation methods

### A.1 Governing equations

The theory of elasticity is clearly set out in the book by Timoshenko [53]. The basic equations are summarized here for the sake of completeness and basic comprehension.

Consider the partial differential equations

$$\nabla^2 u = 0 \quad (\text{A.1})$$

This is known as Laplace's equation. If we only consider the two-dimensional case, we obtain

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = b \quad (\text{A.2})$$

(A.2) is a special form of (2.7). If  $b$  is not equal to zero, it is known as Poisson's equation which governs many types of engineering problems such as seepage and aquifer analysis, heat conduction, diffusion processes, torsion, fluid motion and others [54]. This equation is generally associated with equilibrium or steady state problems and is known as an elliptic partial differential equation as discussed in [28].

We can define a basic coordinate system  $(x_1, x_2, x_3)$  that corresponds to displacements  $(u_1, u_2, u_3)$ . We also define our main stresses  $\sigma_{ij}$  on a body in the same directions and the surface tractions  $(p_1, p_2, p_3)$ . This is shown in Figure A.1.

For the stresses of the body volume in Figure A.1 we have the equilibrium equations

$$\frac{\partial \sigma_{ij}}{\partial x_j} + b_i = 0 \quad (\text{A.3})$$

Where the  $b_i$  term represents the body force per unit volume (internal). We consider  $(n_1, n_2, n_3)$  as the direction cosines of the outward normal vector  $\vec{n}$  as indicated in Figure A.1. We can now relate the surface tractions  $p_i$  to the stresses with the relationship



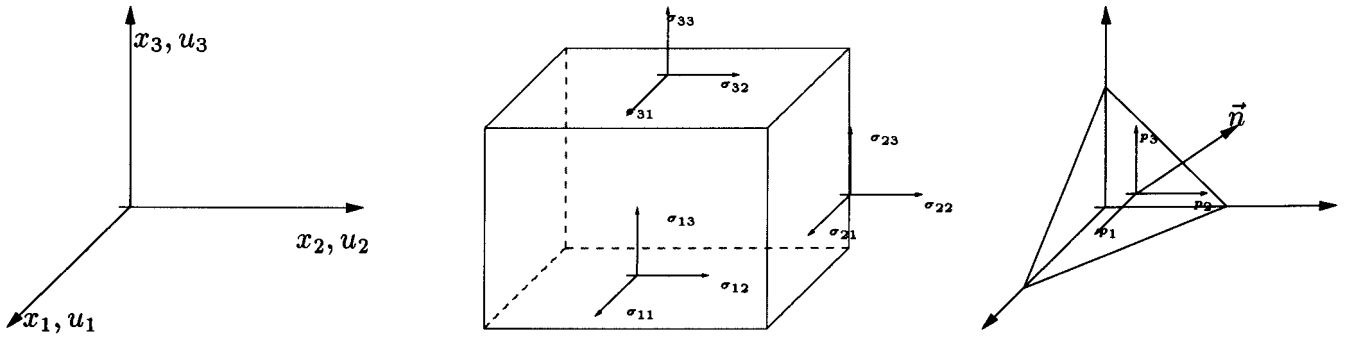


Figure A.1: The main coordinates, displacements, stresses and tractions in three dimensions

$$p_i = \sigma_{ij}n_j \quad (\text{A.4})$$

The deformations of the boundary are a function of the displacements which have the components  $(u_1, u_2, u_3)$  at every point on the boundary. The strain is a description of the deformation per unit length; thus we can write the strains in terms of the deformations.

$$\epsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (\text{A.5})$$

The strain-displacement equations are also called the kinematic equations of deformation equations; they yield the strain field given the displacement field. The constitutive equations connect the stress and strain fields. For linear elasticity however, a considerable simplification occurs because the relation becomes algebraic, linear and homogeneous. Thus for this case the stress-strain relations may be written in component notation as

$$\sigma_{ij} = E_{ijkl}\epsilon_{kl} \quad (\text{A.6})$$

The  $E_{ijkl}$  module of elasticity in this case does not presume the same mechanical properties of the material in all directions. They are components of a fourth order tensor  $\mathbf{E}$  called the elasticity tensor. The elasticity module generally satisfies the following symmetries

$$E_{ijkl} = E_{jikl} = E_{ijlk} \quad (\text{A.7})$$

This relation reduces the number from  $3^4 = 81$  constants to  $6^2 = 36$  constants. Furthermore, it is a requirement that the elastic energy that is stored must be positive and if the body admits strain energy<sup>1</sup> the elastic module must satisfy the additional symmetries

$$E_{ijkl} = E_{klij} \quad (\text{A.8})$$

This reduces the number of independent constants to 21. Further symmetries occur if the material is isotropic and the module of elasticity can be expressed in terms of Young's modulus  $E$  and Poisson's ratio  $\nu$ .

<sup>1</sup>The material is not only elastic but hyperelastic.

The states of stresses and strains in a body are related through the constitutive relationship for the material as shown in [54] generally they are PDEs. This is shown to be

$$\sigma_{ij} = \frac{E}{1+v} \left[ \frac{v}{1-2v} \delta_{ij} \epsilon_{kk} + \epsilon_{ij} \right] \quad (\text{A.9})$$

where  $v$  is Poisson's ratio,  $E$  the Modulus of elasticity and  $\delta_{ij}$  is Kronecher delta ( $\delta_{ij} = 1$  when  $i = j$  and 0 if  $i \neq j$ ).

### A.1.1 One dimension

The only possibility in one dimension is a stress  $\sigma$  along the direction  $x$  and its corresponding strain  $\epsilon$ . The stress strain relationship reduces to

$$\sigma = E\epsilon \quad (\text{A.10})$$

### A.1.2 Two dimensions

#### Plane stress

A thin planar body subjected to in-plane loading on its edge surface is said to be in plane stress [40]. The state of stress is then specified by only  $\sigma_x, \sigma_y, \tau_{xy}$  [53]. It may be assumed that these three components are independent of  $z$  and thus they do not vary through the thickness [53].

#### Plane strain

Plane strain occurs when a body of uniform cross section is subjected to transverse loading along its length [40]. This simplification usually occurs when the dimension of the body in the  $z$  direction is very large [53]. Here  $\epsilon_z, \gamma_{yz}$  and  $\gamma_{xz}$  are taken to be zero [40].

### A.1.3 Mechanics of the boundary value problem

Consider an elastic bar of length  $l$ , constant cross sectional area  $A$ , Young's modulus  $E$  and a temperature coefficient  $\alpha$ . A point load  $P$  is applied to the end of the bar and an arbitrarily distributed load is applied to the length of the bar Figure A.2. The problem is purely one-dimensional and will have only one Euclidean coordinate, namely  $x_1 \in [0, l]$  [55]. An infinitesimal section of the bar is now chosen. The equilibrium is shown schematically in Figure A.3.

By physical reasoning, the state of equilibrium can be written as

$$\begin{aligned} -F + (F + dF) + \rho dx_1 &= 0 \\ \rho + \frac{dF}{dx_1} &= 0 \end{aligned}$$

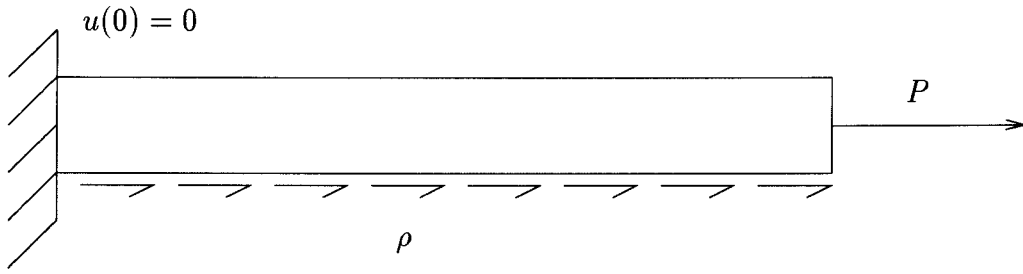


Figure A.2: One-dimensional bar problem

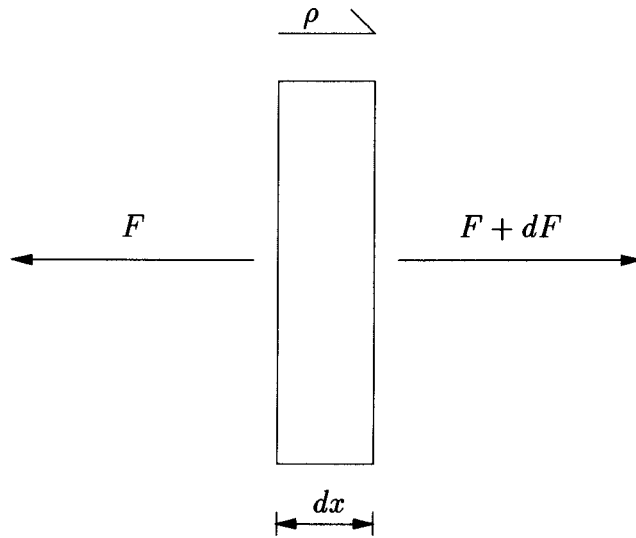


Figure A.3: Equilibrium of an infinitesimal bar element

$$\rho + F_{,1} = 0 \quad (\text{A.11})$$

The strain-displacement relationship for this problem can be written as

$$\epsilon_{11} = \frac{du}{dx_1} = u_{,1} \quad (\text{A.12})$$

The constitutive law (material law) for this problem is obtained as

$$\begin{aligned} \sigma &= E(\epsilon_{11} - \alpha \Delta T) \\ F &= A\sigma = AE(\epsilon_{11} - \alpha \Delta T) \end{aligned} \quad (\text{A.13})$$

The boundary conditions that are applied to the problem are described by

$$u(0) = 0 \quad (\text{A.14})$$

$$F(x_1 = l) = P \quad (\text{A.15})$$

(A.14) is the so-called *essential boundary condition*.

(A.15) is the so-called *natural boundary condition*. It is important to make this distinction since their implementation into finite element equations are different. To define the variational form of the equilibrium equations, we need to characterize two classes of functions [55].

1. The candidate or trial solutions to  $u$ 
  - a) These solutions must satisfy the essential boundary conditions [55].
  - b) They must also satisfy  $\int_0^l u_{,\alpha} u_{,\alpha} dx_1 < \infty$ ;  $\|\alpha\| \leq m$ . (We say  $u \in H^m$  where  $H^m$  is a *Sobolev space* [55].) In this case  $m = 1$  [55].
2. Weighting functions, test functions or variation functions,  $U$  [55].
  - a) We require  $U$  to satisfy the homogeneous part of the essential boundary conditions [55].  $U \in H_0^m$ , where the 0 subscript refers to the *homogeneous part* [55].

The boundary value problem as defined above is known as the strong form of the boundary value problem. In the finite element displacement method, the equilibrium equation can be written in its variational form as (A.16) where  $U$  are the weighting functions

$$\int_0^l U(\rho + F_{,1}) dx = 0 \quad (\text{A.16})$$

(A.16) corresponds to weighting the error or residual over the domain with respect to a weighting function  $U$  which may be arbitrary. The solution of this form is identical to the solution of the equilibrium equation if  $U(x_1)$  is allowed to be arbitrary with the constraint that  $U$  is required to satisfy the homogeneous part of the essential boundary conditions (Point 2a above) [55].

### The weak form of the boundary value problem

Integration by parts of (A.16), to reduce the order of the derivatives of  $u$

$$\int_0^l U(\rho + F_{,1}) dx_1 = FU|_0^l - \int_0^l FU_{,1} dx_1 + \int_0^l U\rho dx_1 = 0 \quad (\text{A.17})$$

Now we need to satisfy exactly

$$\left. \begin{array}{l} \text{Strain-displacement} \\ \text{Constitutive law} \\ \text{Boundary conditions: } F(l) = P \end{array} \right\} F = AEu_{,1} - AE\alpha\Delta T \quad (\text{A.18})$$

This is called the *variational equation* which is equivalent to the equation of virtual work. The solution is the weak or generalized solution.  $U(x_i)$  is the same as the virtual displacement and is sometimes also written in the form  $\delta u(x_1)$ . The weak form is also called the

variational boundary value problem. The above formulation leads to a *symmetric* form and to a weakening of the smoothness requirements on  $u$  since we are only working with the first derivative [55].

### The Galerkin method: discretization

Our first step in the direction of a solution method is to construct finite dimensional approximations to the trial solutions  $u$  and the weighting functions  $U$ .  $u^h$  is denoted as a member of the collection of functions used to approximate  $u$  and  $U^h$  as a member of the functions used to approximate  $U$ . In fact  $u^h$  is an approximate solution to the weak form [55].

To approximate the trial functions we use a set of basis functions  $\phi^k$ ,  $k = 1 \dots N$  and we obtain

$$u^h = \phi^k q^k \quad \text{sum on } k \quad (\text{A.19})$$

(Note:  $k$  is not a power.) The trial function is therefore made up of the accumulation of chosen basis functions multiplied by unknown coefficients,  $q^k$ ,  $k = 1 \dots N$  [55].

There are various ways in which the weighting function can be chosen, but we will use the *Bubnov-Galerkin* method which assumes the same basis functions as are used for the trial solutions [55]. Hence we obtain

$$U^h = \phi^k Q^k \quad (\text{A.20})$$

We can also write both equations in matrix form

$$u^h = \underbrace{N}_{1 \times N} \underbrace{q}_{N \times 1} \quad U^h = \underbrace{N}_{1 \times N} \underbrace{Q}_{N \times 1} \quad (\text{A.21})$$

These are substituted into the weak form to give an approximate solution in

$$\int_0^l U_{,1}^h A E u_{,1}^h dx_1 = \int_0^l U^h p(x_1) dx_1 + \int_0^l A E \alpha \Delta T U_{,1}^h dx_1 \quad (\text{A.22})$$

(A.22) is sometimes referred to as the Galerkin equation. Approximate methods of the type considered are examples of so-called weighted residual methods. We are in fact weighting the residual of the equilibrium equations  $\rho + F_{,1}$ . By substitution of  $U^h$  and  $u^h$ , (A.22) becomes (A.23) [55].

$$Q^T \int_0^l N_{,1}^T A E N_{,1} dx_1 = Q^T N_l^T P + Q^T \int_0^l N^T \rho(x_1) dx_1 + Q^T \int_0^l N_{,1}^T A E \alpha \Delta T dx_1 \quad (\text{A.23})$$

Since  $U^h$  can be arbitrary,  $Q$  can also be arbitrary and (A.23) becomes (A.24),

$$\int_0^l N_{,1}^T A E N_{,1} dx_1 q = N_l^T P + \int_0^l N^T \rho dx_1 + \int_0^l N_{,1}^T A E \alpha \Delta T dx_1 \quad (\text{A.24})$$

or

$$\left[ \int_0^l N_{,1}^T A E N_{,1} dx_1 \right] q = N_l^T P + \int_0^l N^T \rho dx_1 + \int_0^l N_{,1}^T A E \alpha \Delta T dx_1 \quad (\text{A.25})$$

or

$$\mathbf{K} \mathbf{q} = \mathbf{R}_f + \mathbf{R}_p + \mathbf{R}_t \quad (\text{A.26})$$

$\mathbf{K}$  designates the stiffness matrix ( $N \times N$ )

$\mathbf{q}$  designates the unknown displacement coefficients ( $N \times 1$ )

$\mathbf{R}_f$  designates the generalized or equivalent point loads ( $N \times 1$ )

$\mathbf{R}_p$  designates the generalized or equivalent distributed loads ( $N \times 1$ )

$\mathbf{R}_t$  designates the generalized or equivalent thermal loads ( $N \times 1$ )

The *Bubnov-Galerkin method* leads to symmetry of the stiffness matrix and the requirement of only one set of basis functions [55]. The only task remaining to find  $u^h$  is the solution of the generalized equilibrium equations of (A.26).

## A.2 Finite difference method

### A.2.1 Introduction

The finite difference method (FDM) is a very general way of solving PDEs for all types of problems. Smith [28] deals with this in great detail. Since the structural problems under consideration are governed by the Laplace equations, (A.2), we will only consider finite difference methods for solving this PDE. It is valuable to be familiar with this approach because such knowledge will reinforce our understanding of finite element procedures [56]. The approach is also very similar to approaches that could be used for CA.

### A.2.2 Methodology

If we divide a function  $y(x)$  into equally distant spaces  $\delta$  so that we obtain discrete values  $y_1, y_2, y_3 \dots$  for  $x = 0, x = \delta, x = 2\delta \dots$ , we can approximate the first derivatives of  $y(x)$  at corresponding points

$$\left( \frac{dy}{dx} \right)_{x=0} \approx \frac{y_1 - y_0}{\delta} \quad \left( \frac{dy}{dx} \right)_{x=\delta} \approx \frac{y_2 - y_1}{\delta} \quad (\text{A.27})$$

In the same manner we can obtain the second derivatives as shown

$$\left( \frac{d^2y}{dx^2} \right)_{x=\delta} \approx \left( \frac{dy}{dx} \right)_{x=\delta} - \left( \frac{dy}{dx} \right)_{x=0} = \frac{y_2 - 2y_1 + y_0}{\delta^2} \quad (\text{A.28})$$

Suppose we have a smooth function  $f(x, y)$  we can use approximate solutions to partial derivatives similar to (A.27) and (A.28) for the points specified in Figure A.4.

$$\begin{aligned} \frac{\partial f}{\partial x} &\approx \frac{f_1 - f_0}{\delta} & \frac{\partial f}{\partial y} &\approx \frac{f_2 - f_0}{\delta} \\ \frac{\partial^2 f}{\partial x^2} &\approx \frac{f_1 - 2f_0 + f_3}{\delta^2} & \frac{\partial^2 f}{\partial y^2} &\approx \frac{f_1 - 2f_0 + f_4}{\delta^2} \end{aligned} \quad (\text{A.29})$$

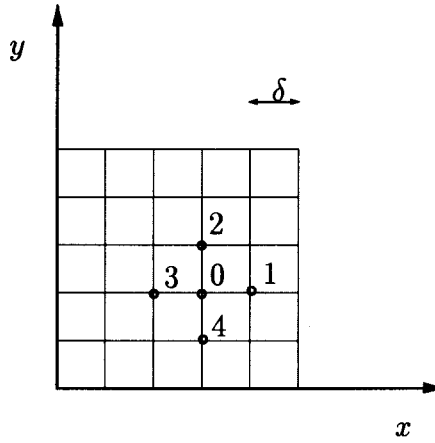


Figure A.4: Finite difference two-dimensional grid

To gain better insight into how this can be used, we can consider the shear stresses within a long uniform cylinder in torsion. The governing PDE is shown to reduce to

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = -2\mu\theta \quad (\text{A.30})$$

$\phi$  is the stress function which must be constant along the boundary of the cross section,  $\theta$  is the angle of twist per unit length of the bar, and  $\mu$  is the modulus of shear [53]. Using (A.29), we can transform (A.30) into a finite difference equation

$$\frac{1}{\delta^2}(\phi_1 + \phi_2 + \phi_3 + \phi_4 - 4\phi_0) = -2\mu\theta \quad (\text{A.31})$$

In this way every torsional problem reduces to a set of numerical values of the stress function  $\phi$  which satisfy (A.31) at every nodal point within the boundary of the cross section and become constant along the boundary [53]. As the simplest example, a bar of square cross section  $a \times a$  will be considered and a square net with a mesh side  $\delta = \frac{1}{4}a$  will be used as shown in Figure A.5.

From symmetry we can conclude that it is sufficient to consider only one eighth of the cross section shown in Figure A.5 [53]. If we determine values for the stress function  $\phi$  at points  $\alpha$ ,  $\beta$  and  $\gamma$  we shall know  $\phi$  at all nodal points on the net within the boundary [53]. We can now rewrite (A.31) as (A.32) by using symmetry.

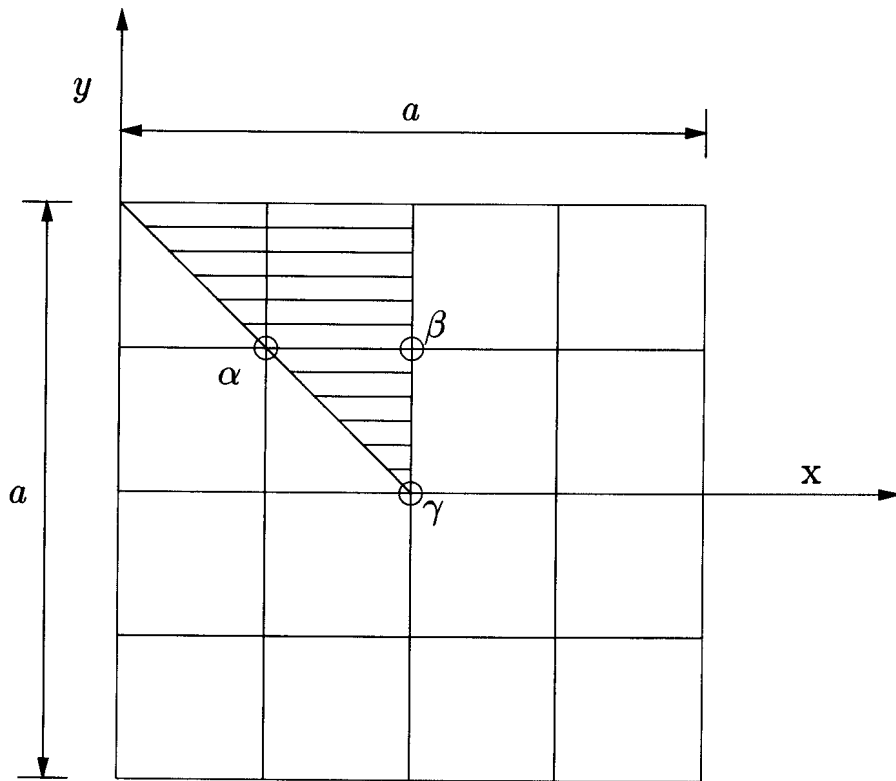


Figure A.5: Finite difference torsion bar example cross section

$$\begin{aligned}
 2\beta - 4\alpha &= -2\mu\theta\delta^2 \\
 2\alpha + \gamma - 4\beta &= -2\mu\theta\delta^2 \\
 4\beta - 4\gamma &= -2\mu\theta\delta^2
 \end{aligned}
 \tag{A.32}$$

(A.32) can now be solved to obtain  $\alpha = 1.375\mu\theta\delta^2$ ,  $\beta = 1.750\mu\theta\delta^2$  and  $\gamma = 2.250\mu\theta\delta^2$  [53]. The required stress function is thus determined by these numerical values at all nodal points within the boundary and by zero values at the boundary [53].

Methods of solution belong essentially to either a class of direct methods or a class of iterative methods [28]. Direct methods solve the system of equations in a known number of arithmetic operations and errors in the solution arise entirely from rounding errors introduced during computation [28]. These direct methods are elimination methods of which the best known examples are the Gaussian elimination method and the triangular decomposition method which factorizes the matrix  $\mathbf{A}$  of the equation  $\mathbf{Ax} = \mathbf{B}$  into  $\mathbf{A} = \mathbf{LU}$ , where  $\mathbf{L}$  and  $\mathbf{U}$  are upper and lower triangular matrices respectively [28]. In the latter method, once the decomposition has been determined, the solution is calculated from  $\mathbf{LUx} = \mathbf{B}$  by setting  $\mathbf{Ux} = \mathbf{y}$  and then solving  $\mathbf{Ly} = \mathbf{b}$  for  $\mathbf{y}$  by forward substitution and  $\mathbf{Ux} = \mathbf{y}$  for  $\mathbf{x}$  by back substitution [28]. With both methods it is usually necessary to employ partial pivoting with scaling to control the growth of rounding errors [28].



### A.2.3 Methods of successive approximations

To increase the accuracy of the solution, it is necessary to refine the mesh used. But the number of equations that must be solved becomes larger and larger. The solution can be greatly simplified by using methods of successive approximations [53]. An iterative method for solving equations is one in which a first approximation is used to calculate a second approximation which is in turn used to calculate a third and so on. The iterative procedure is said to be convergent when the differences between the exact solution and the successive approximation tend to zero as the number of iterations increase. In general, an exact solution is never obtained in a finite number of steps, but this does not matter. What is important is that the successive approximation converges very rapidly to values that are correct to a specific accuracy. One would consider using iterative methods when a direct method requires faster computer storage space than is available and the matrix coefficients are sparse but well conditioned. This situation often arises with the difference equations approximating elliptic boundary value problems [28], as is the case with Laplace's equation (A.1) governing elastostatic problems.

To illustrate this, we return to the two-dimensional case with Poisson's equation, (A.2). The corresponding finite difference equation is given by.

$$\frac{1}{4}(\phi_1 + \phi_2 + \phi_3 + \phi_4) = \phi_0 \quad (\text{A.33})$$

(A.33) implies that the approximate value of the function  $\phi$  at the nodal point 0 of the square net is equal to the average value of the function at the four adjacent nodal points. We suppose that for a simple system we obtain the system of equations in (A.34) where  $a_{ii} \neq 0$  for  $i = 1 \dots 4$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4 \end{aligned} \quad (\text{A.34})$$

Rewriting (A.34) in the form of the unknowns gives

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - a_{14}x_4) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - a_{24}x_4) \\ x_3 &= \frac{1}{a_{33}}(b_3 - a_{31}x_1 - a_{32}x_2 - a_{34}x_4) \\ x_4 &= \frac{1}{a_{44}}(b_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3) \end{aligned} \quad (\text{A.35})$$

### Jacobi method

We denote our first approximation to  $x_i$  by  $x_i^{(1)}$  and the second by  $x_i^{(2)}$  etc., and assume that  $n$  of them have been calculated, i.e.  $x_i^{(n)}$  is known for  $i = 1 \dots 4$ . The Jacobi iterative method expresses the  $(n + 1)$ th iterative values exclusively in terms of the  $n$ th iterative values and the iterations corresponding to (A.35) are defined by (A.36).

$$\begin{aligned}
 x_1^{(n+1)} &= \frac{1}{a_{11}} \left( b_1 - a_{12}x_2^{(n)} - a_{13}x_3^{(n)} - a_{14}x_4^{(n)} \right) \\
 x_2^{(n+1)} &= \frac{1}{a_{22}} \left( b_2 - a_{21}x_1^{(n)} - a_{23}x_3^{(n)} - a_{24}x_4^{(n)} \right) \\
 x_3^{(n+1)} &= \frac{1}{a_{33}} \left( b_3 - a_{31}x_1^{(n)} - a_{32}x_2^{(n)} - a_{34}x_4^{(n)} \right) \\
 x_4^{(n+1)} &= \frac{1}{a_{44}} \left( b_4 - a_{41}x_1^{(n)} - a_{42}x_2^{(n)} - a_{43}x_3^{(n)} \right)
 \end{aligned} \tag{A.36}$$

### Gauss-Seidel method

In this method the  $(n + 1)$ th iterative values are used as soon as they become available. Iterations corresponding to (A.35) are defined by (A.37)

$$\begin{aligned}
 x_1^{(n+1)} &= \frac{1}{a_{11}} \left( b_1 - a_{12}x_2^{(n)} - a_{13}x_3^{(n)} - a_{14}x_4^{(n)} \right) \\
 x_2^{(n+1)} &= \frac{1}{a_{22}} \left( b_2 - a_{21}x_1^{(n+1)} - a_{23}x_3^{(n)} - a_{24}x_4^{(n)} \right) \\
 x_3^{(n+1)} &= \frac{1}{a_{33}} \left( b_3 - a_{31}x_1^{(n+1)} - a_{32}x_2^{(n+1)} - a_{34}x_4^{(n)} \right) \\
 x_4^{(n+1)} &= \frac{1}{a_{44}} \left( b_4 - a_{41}x_1^{(n+1)} - a_{42}x_2^{(n+1)} - a_{43}x_3^{(n+1)} \right)
 \end{aligned} \tag{A.37}$$

This increases the rate at which the solution converges, which means the number of iterations required to obtain a solution is reduced.

### Successive over-relaxation method

If  $x_i^{(n)}$  is added to and subtracted from the right side of the Gauss-Seidel equation, (A.37) is rewritten as

$$\begin{aligned}
 x_1^{(n+1)} &= x_1^{(n)} + \left[ \frac{1}{a_{11}} \left( b_1 - a_{11}x_1^{(n)} - a_{12}x_2^{(n)} - a_{13}x_3^{(n)} - a_{14}x_4^{(n)} \right) \right] \\
 x_2^{(n+1)} &= x_2^{(n)} + \left[ \frac{1}{a_{22}} \left( b_2 - a_{21}x_1^{(n+1)} - a_{22}x_2^{(n)} - a_{23}x_3^{(n)} - a_{24}x_4^{(n)} \right) \right]
 \end{aligned}$$

$$\begin{aligned}
 x_3^{(n+1)} &= x_3^{(n)} + \left[ \frac{1}{a_{33}} \left( b_3 - a_{31}x_1^{(n+1)} - a_{32}x_2^{(n+1)} - a_{33}x_3^{(n)} - a_{34}x_4^{(n)} \right) \right] \\
 x_4^{(n+1)} &= x_4^{(n)} + \left[ \frac{1}{a_{44}} \left( b_4 - a_{41}x_1^{(n+1)} - a_{42}x_2^{(n+1)} - a_{43}x_3^{(n+1)} - a_{44}x_4^{(n)} \right) \right]
 \end{aligned} \tag{A.38}$$

It is seen that the expressions in the square brackets are the corrections or changes made to  $x_i^{(n)}$ ,  $i = 1 \dots 4$ , by one Gauss-Seidel iteration [28]. If successive corrections are all one-signed as they usually are for the approximating difference equation of elliptic problems, it would be reasonable to expect convergence to be accelerated if each equation of (A.38) were given a larger correction term than is defined by (A.38) [28]. This idea leads to the successive over-relaxation (SOR) which is defined by (A.39).

$$\begin{aligned}
 x_1^{(n+1)} &= x_1^{(n)} + \left[ \frac{\omega}{a_{11}} \left( b_1 - a_{11}x_1^{(n)} - a_{12}x_2^{(n)} - a_{13}x_3^{(n)} - a_{14}x_4^{(n)} \right) \right] \\
 x_2^{(n+1)} &= x_2^{(n)} + \left[ \frac{\omega}{a_{22}} \left( b_2 - a_{21}x_1^{(n+1)} - a_{22}x_2^{(n)} - a_{23}x_3^{(n)} - a_{24}x_4^{(n)} \right) \right] \\
 x_3^{(n+1)} &= x_3^{(n)} + \left[ \frac{\omega}{a_{33}} \left( b_3 - a_{31}x_1^{(n+1)} - a_{32}x_2^{(n+1)} - a_{33}x_3^{(n)} - a_{34}x_4^{(n)} \right) \right] \\
 x_4^{(n+1)} &= x_4^{(n)} + \left[ \frac{\omega}{a_{44}} \left( b_4 - a_{41}x_1^{(n+1)} - a_{42}x_2^{(n+1)} - a_{43}x_3^{(n+1)} - a_{44}x_4^{(n)} \right) \right]
 \end{aligned} \tag{A.39}$$

The factor  $\omega$ , called the acceleration parameter or relaxation factor, generally lies in the range  $1 < \omega < 2$  [28]. The determination of the optimum value of  $\omega$  for maximum rate of convergence is discussed in [28] but is beyond the scope of this research.

#### A.2.4 Practicality for structural analysis

In the problem of the torsion bar in Figure A.5 the boundary conditions are specified directly in the form of the stress function. This is, however, not possible with most problems since the boundary conditions applied are specified displacements (*essential boundary conditions*) and forces applied (*natural boundary conditions*). For a general problem it is possible to obtain either displacements or stresses where the boundary conditions are applied. This fact makes finite differences inapplicable in common problems. Table A.1 summarizes some widely used finite difference approximations, also called finite difference stencils or molecules [56]. These stencils are derived from the partial derivatives to be used for successive approximations.

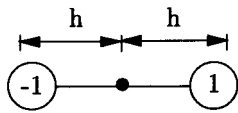
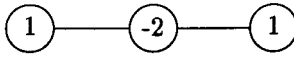

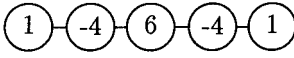
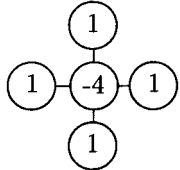
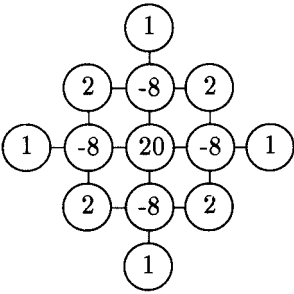
Differentiation	Finite difference approximation	Molecules
$\frac{dw}{dx} _i$	$\frac{w_{i+1}-w_{i-1}}{2h}$	
$\frac{d^2w}{dx^2} _i$	$\frac{w_{i+1}-2w_i+w_{i-1}}{h^2}$	
$\frac{d^3w}{dx^3} _i$	$\frac{w_{i+2}-2w_{i+1}+2w_{i-1}-w_{i-2}}{2h^3}$	
$\frac{d^4w}{dx^4} _i$	$\frac{w_{i+2}-4w_{i+1}+6w_i-4w_{i-1}+w_{i-2}}{h^4}$	
$\nabla^2 w _{i,j}$	$\frac{-4w_{i,j}+w_{i+1,j}+w_{i,j+1}+w_{i-1,j}+w_{i,j-1}}{h^2}$	
$\nabla^4 w _{i,j}$	$\frac{[20w_{i,j} - 8(w_{i+1,j} + w_{i-1,j} + w_{i,j+1} + w_{i,j-1}) + 2(w_{i+1,j+1} + w_{i-1,j+1} + w_{i+1,j-1} + w_{i-1,j-1}) + w_{i+2,j} + w_{i-2,j} + w_{i,j+2} + w_{i,j-2}]/h^4}$	

Table A.1: Established finite difference computational molecules

## A.3 Finite element method

### A.3.1 Introduction

In a 1941 mathematics lecture, published in 1943, Courant suggested piece-wise polynomial interpolation over triangular sub-regions as a way to obtain approximate numerical solutions. He recognized this approach as a Rayleigh-Ritz solution of a variation problem. This is the finite element method (FEM) as we know it today. Courant's work was forgotten until engineers had independently developed it [55].

None of the preceding work was practical at the time because there were no computers to perform the calculations. By 1953 engineers were writing stiffness equations in matrix notation and solving the equations with digital computers. The classical paper *Stiffness and deflection analysis of complex structures* by Turner, Clough and Topp appeared in 1956. With this paper and others, explosive development of FEM in engineering began. The name "finite element" was coined in 1960. By 1963 the method was recognized as rigorously sound, and it became a respected area of study for academics. As late as 1967, engineers and mathematicians worked with finite elements in apparent ignorance of one another.

Currently, finite elements are the most generally used method in structural analysis. FEM has the capability of dealing with both linear and non-linear problems and can be constructed using various types of formulations.

There are various ways of introducing the FEM. It can be introduced rather physically, as is done with the displacement formulation as found in most introductory courses. The approach works well for teaching basics but is somewhat superficial and does not allow the reader to think of new possibilities, an attitude that has to be developed in further analyses. One can also proceed along purely mathematical lines, the approach taken by mathematicians. For the engineer, a purely mathematical course should serve to broaden his or her knowledge base and not as an introduction to finite element analysis [55].

The vast amount of finite element formulations available designed for specific tasks makes it impossible to describe a full definition of finite elements in this document. Instead, the approach will be to simply explain enough of the method to allow us to analyse the problems with which the CA analysis will be compared. The formulation will adopt a mathematical point of view but with practical implementation since it is an existing method that is most often used to solve the governing equation in structural analyses. This also allows for comparison to boundary element methods which consist of a more complicated formulation and implementation. Finite elements are probably by far the most used and the most well-known method in structural analyses and should thus be considered as an extremely suitable standard for comparison.

### A.3.2 Methodology

#### One dimension

In the FEM domain of a bar as discussed in section A.1.3, the bar is divided into intervals and the displacements field is assumed to be constructed from piece-wise linear functions. Some degree of continuity is imposed at the boundaries between the pieces or elements. The degree of continuity in the field variable must be the same as one order lower than its differential order in the weak form [55].

If we use the same notation as in section A.1.3, we can divide the bar into  $n - 1$  intervals.

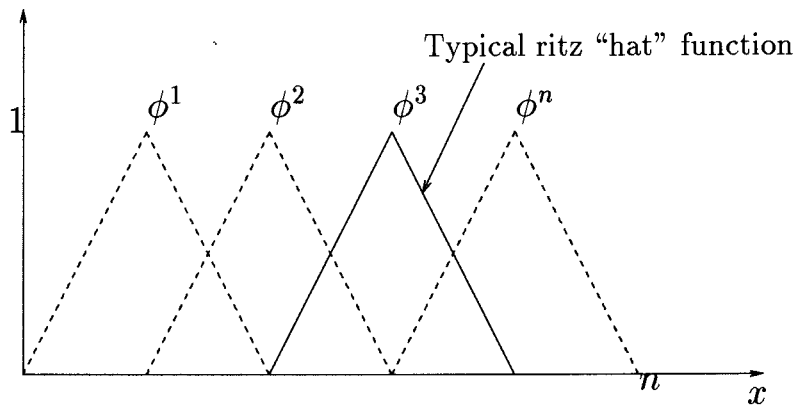


Figure A.6: Typical ritz function used in the analysis of one-dimensional bar problem

A piece-wise linear function can be made up as the linear combination of the Ritz functions  $\phi$  shown in Figure A.7. The  $\phi^k$  shown in Figure A.6 are functions that equal 1 at a particular node  $k$  and vanish at all others [55].

$$u = \phi^k q^k \quad k = 1, 2, 3 \dots n \quad (\text{A.40})$$

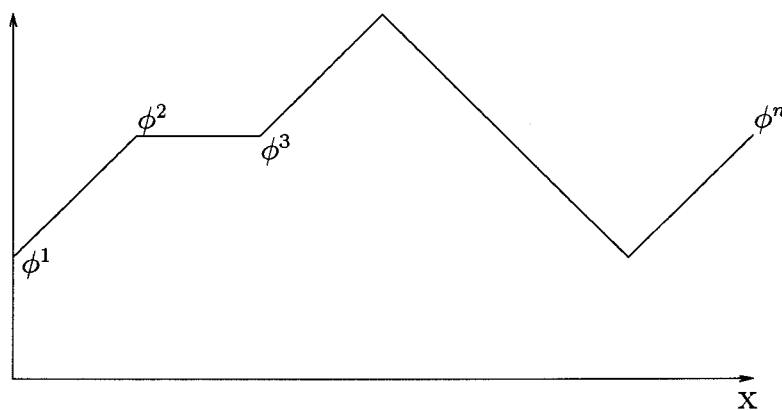


Figure A.7: Linear combination of the ritz functions

We notice that  $\phi^k$  forms a local basis for the displacement variable as it is identically zero except in elements adjoining node  $k$ . Only the adjacent elements will be coupled. The

piece-wise linear finite element functions are the most widely used finite element functions for one-dimensional models [55].

We now consider the one-dimensional bar problem shown in Figure A.2. We consider the solution to this problem to be obtained using two finite elements as shown in Figure A.8.

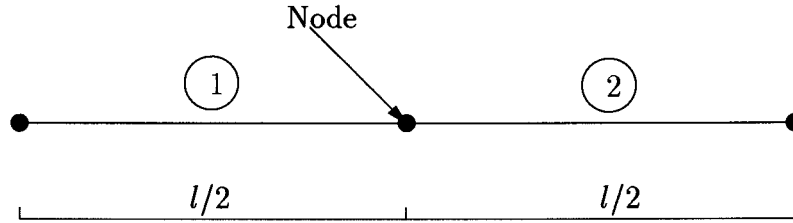


Figure A.8: Simple bar FEM mesh

Since the bar is constrained at the left hand side, the first shape function  $\phi^1$  will not influence the solution since the displacement is specified to be zero in that point. The second  $\phi^2$  and third  $\phi^3$  shape functions are defined by

$$\begin{aligned} \phi^2 &= \begin{cases} \frac{2x}{l} & 0 < x < \frac{l}{2} \\ 2 - \frac{2x}{l} & \frac{l}{2} < x < l \end{cases} \\ \phi^3 &= \begin{cases} \frac{2x}{l} - 1 & \frac{l}{2} < x < l \\ 0 & 0 < x < \frac{l}{2} \end{cases} \end{aligned} \quad (\text{A.41})$$

To solve the displacement  $u$  we can rewrite (A.40) as

$$u = \phi^2 q^2 + \phi^3 q^3 = [\phi^2 \phi^3] \begin{Bmatrix} q^2 \\ q^3 \end{Bmatrix} \quad (\text{A.42})$$

The strains at the represented nodes are defined by

$$\epsilon_{11} = u_{,1} = [\phi_{,1}^2 \phi_{,1}^3] \begin{Bmatrix} q^2 \\ q^3 \end{Bmatrix} \quad (\text{A.43})$$

To obtain the strain, we need to find the partial derivatives of our shape function as defined by (A.41). The derivatives are obtained as

$$\begin{aligned} \phi_{,1}^2 &= \begin{cases} \frac{2}{l} & 0 < x < \frac{l}{2} \\ -\frac{2}{l} & \frac{l}{2} < x < l \end{cases} \\ \phi_{,1}^3 &= \begin{cases} \frac{2}{l} & \frac{l}{2} < x < l \\ 0 & 0 < x < \frac{l}{2} \end{cases} \end{aligned} \quad (\text{A.44})$$

We use this known function in the weak form of our governing equation, (A.25). The evaluation of the left hand side of (A.26) can be shown to be

$$\begin{aligned}
& AE \int_0^l \left\{ \begin{matrix} \phi_{,1}^2 \\ \phi_{,1}^3 \end{matrix} \right\} [\phi_{,1}^2, \phi_{,1}^3] dx \\
& = AE \left[ \begin{array}{c} \left( \int_0^{\frac{l}{2}} \frac{2}{l} \frac{2}{l} dx + \int_{\frac{l}{2}}^l -\frac{2}{l} - \frac{2}{l} dx \right) \quad \left( \int_0^{\frac{l}{2}} \frac{2}{l} 0 + \int_{\frac{l}{2}}^l -\frac{2}{l} \frac{2}{l} dx \right) \\ \text{Element 1} \quad \text{Element 2} \\ \left( \int_0^{\frac{l}{2}} \frac{2}{l} 0 dx + \int_{\frac{l}{2}}^l -\frac{2}{l} \frac{2}{l} dx \right) \quad \left( \int_0^{\frac{l}{2}} 0 0 dx + \int_{\frac{l}{2}}^l \frac{2}{l} \frac{2}{l} dx \right) \\ \text{Element 1} \quad \text{Element 2} \end{array} \right] \\
& = AE \left[ \begin{array}{cc} \left( \frac{4}{l^2} \frac{l}{2} + \frac{4}{l^2} \frac{l}{2} \right) & \left( 0 - \frac{4}{l^2} \frac{l}{2} \right) \\ \left( 0 - \frac{4}{l^2} \frac{l}{2} \right) & \left( 0 + \frac{4}{l^2} \frac{l}{2} \right) \end{array} \right] \\
& = \frac{AE}{l} \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \tag{A.45}
\end{aligned}$$

Now we can consider the solution to the distributed load as set out in (A.26) as

$$\begin{aligned}
& \rho \int_0^l \left\{ \begin{matrix} \phi^2 \\ \phi^3 \end{matrix} \right\} dx \\
& = \rho \left[ \underbrace{\int_0^{\frac{l}{2}} \left\{ \begin{matrix} \frac{2x}{l} \\ 0 \end{matrix} \right\} dx}_{\text{Element 1}} + \underbrace{\int_{\frac{l}{2}}^l \left\{ \begin{matrix} 2 - \frac{2x}{l} \\ \frac{2x}{l} - 1 \end{matrix} \right\} dx}_{\text{Element 2}} \right] \\
& = \rho \left[ \begin{array}{c} \frac{x^2}{l} \Big|_0^{\frac{l}{2}} + \frac{2x - \frac{x^2}{l}}{l} \Big|_{\frac{l}{2}}^l \\ 0 \end{array} \right] \\
& = \rho \left[ \begin{array}{c} \frac{l}{4} + \frac{l - \frac{3l}{4}}{4} \\ 0 \end{array} \right] \\
& = \rho \left[ \begin{array}{c} \frac{l}{4} \\ \frac{l}{4} \end{array} \right] \tag{A.46}
\end{aligned}$$

The nodal load  $P$  is only applied at the third node. It can be added to the right hand side as

$$P \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{A.47}$$

We can now write out our solution which is a combination of (A.45), (A.46) and (A.47) as



$$\frac{AE}{l} \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \begin{Bmatrix} q^2 \\ q^3 \end{Bmatrix} = \rho \begin{bmatrix} \frac{l}{2} \\ \frac{l}{4} \end{bmatrix} + P \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (\text{A.48})$$

With  $\rho$  and  $P$  known, the only unknowns to solve in this equation are the displacements  $q^2$  and  $q^3$ . Once the displacements are known, we use (A.10) and (A.43) to calculate the stress as

$$\sigma = E[\phi_{,1}^2, \phi_{,1}^3] \begin{Bmatrix} q^2 \\ q^3 \end{Bmatrix} \quad (\text{A.49})$$

### Two dimensions

We will now consider a two-dimensional problem under plane stress as described in section A.1.3 and its governing equations described by (A.3), (A.5) and (A.9). In Figure A.9 we define a rectangular element used to model the membrane.

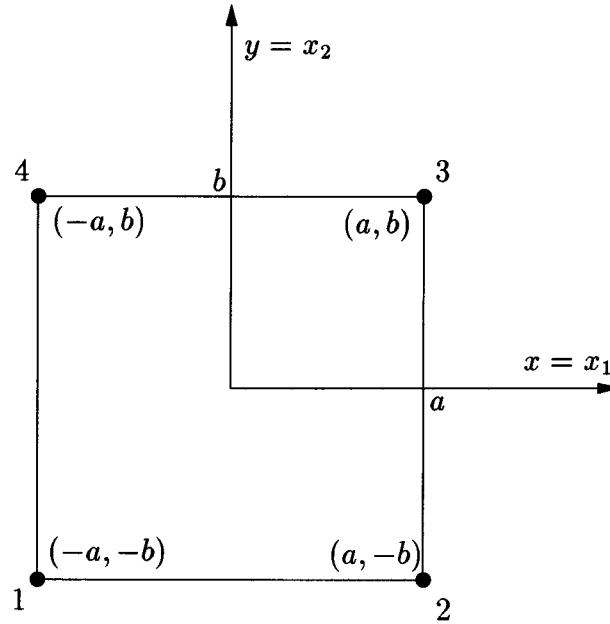


Figure A.9: Simple membrane FEM element

The displacements of each node can be approximated in a similar fashion as with our one-dimensional element. We can therefore rewrite (A.40) into a form where  $i$  denotes the node under consideration as

$$u_i = \phi^k q_i^k \quad k = 1, \dots, 4 \quad (\text{A.50})$$

A node is defined at each corner and similar to the ritz hat function in our one-dimensional problem, we want a shape function  $\phi$  such that it has a value of 1 at the node under consideration and zero at all other nodes. These functions should be linearly independent and

provide continuity of displacement with reference to Figure A.9. The four shape functions can be written as

$$\begin{aligned}
 \phi^1 &= \frac{1}{4} \left(1 - \frac{x_1}{a}\right) \left(1 - \frac{x_2}{a}\right) \\
 \phi^2 &= \frac{1}{4} \left(1 + \frac{x_1}{a}\right) \left(1 - \frac{x_2}{a}\right) \\
 \phi^3 &= \frac{1}{4} \left(1 + \frac{x_1}{a}\right) \left(1 + \frac{x_2}{a}\right) \\
 \phi^4 &= \frac{1}{4} \left(1 - \frac{x_1}{a}\right) \left(1 + \frac{x_2}{a}\right)
 \end{aligned} \tag{A.51}$$

Since different finite elements have different shapes, we need to transform the  $x_1$  and  $x_2$  axis to normalized coordinates with axes  $\zeta_1$  and  $\zeta_2$  where our element is always square in these normalized coordinates. In order to determine the shape functions in these normalized coordinates the values of  $a$  and  $b$  in A.9 are set equal to one, resulting in the following shape functions.

$$\begin{aligned}
 \phi^1 &= \frac{1}{4} (1 - \zeta_1) (1 - \zeta_2) \\
 \phi^2 &= \frac{1}{4} (1 + \zeta_1) (1 - \zeta_2) \\
 \phi^3 &= \frac{1}{4} (1 + \zeta_1) (1 + \zeta_2) \\
 \phi^4 &= \frac{1}{4} (1 - \zeta_1) (1 + \zeta_2)
 \end{aligned} \tag{A.52}$$

Suppose that the finite elements occupy a domain  $\Omega$ , and are surrounded by a boundary  $\Gamma$ . Suppose too that a distributed load  $p$  is applied along the boundary between nodes 1 and 2. We know that the displacements are described by (A.50). We define our finite element formulation from (A.25) as

$$\underbrace{\int_{\Omega} \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega}_\text{Stiffness} q = \underbrace{\int_{\Gamma} \mathbf{N}^T p^* d\Gamma}_\text{Consistent Nodal Loads} \tag{A.53}$$

$\mathbf{C}$  is the constitutive matrix and derived from basic continuum mechanics.  $\mathbf{B}$  contains derivatives of the shape functions matrix which are contained in  $\mathbf{N}$ . As with the one-dimensional case, we will consider the left and right hand side of (A.53) separately. The left hand side is expressed in (A.54) for a two-dimensional membrane element with a thickness  $t$ .

$$k^m = \int_A \mathbf{B}^T \mathbf{C} \mathbf{B} dx_1 dx_2 t \tag{A.54}$$

In (A.54) we reduced our domain  $\Omega$  to an area  $A$  with our two axis  $(x_1 \ x_2)$  as defined in Figure A.9. We can now translate the entire equation to our normalized axis as set out with (A.52) rewriting (A.54) with  $dx_1 = ad\zeta_1$ ,  $dx_2 = bd\zeta_2$ . This yields

$$k^m = \int_{-1}^1 \int_{-1}^1 \mathbf{B}^T \mathbf{C} \mathbf{B} d\zeta_1 d\zeta_2 abt \quad (\text{A.55})$$

We can now move the constants  $a$ ,  $b$  and  $t$  in (A.55) out of the integral and use numerical integration to solve the problem as

$$k^m = abt \sum_{i=1}^m \sum_{j=1}^n \mathbf{B}^T(\zeta_i, \zeta_j) \mathbf{C} \mathbf{B}(\zeta_i, \zeta_j) w_i w_j \quad (\text{A.56})$$

The only aspect of (A.56) that has not been dealt with is the strain-displacement operator  $\mathbf{B}$ . Consider the displacements of the first two nodes only since this is sufficient to describe this operator. We can write the displacements of the nodes as a function of the shape functions equation. By doing this (A.40) becomes (A.52) in matrix form as

$$\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{bmatrix} \phi^1 & 0 & \phi_1 & 0 & \phi_1 & 0 \\ 0 & \phi_2 & 0 & \phi_2 & 0 & \phi_2 \end{bmatrix} \begin{Bmatrix} q_1^1 \\ q_2^1 \\ q_1^2 \\ q_2^2 \\ q_1^3 \\ q_2^3 \\ q_1^4 \\ q_2^4 \end{Bmatrix} \quad (\text{A.57})$$

Strain  $\epsilon$  if difined as the derivative of our displacement equation, (A.12). Thus, we can write the strain as

$$\epsilon = \mathbf{L}u = \underbrace{\mathbf{L} \mathbf{N}}_{\mathbf{B}} q = \begin{Bmatrix} \frac{\partial}{\partial x_1} & 0 \\ 0 & \frac{\partial}{\partial x_2} \\ \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} \end{Bmatrix} \begin{bmatrix} \phi^1 & 0 & \phi^2 & 0 & \phi^3 & 0 & \phi^4 & 0 \\ 0 & \phi^1 & 0 & \phi^2 & 0 & \phi^3 & 0 & \phi^4 \end{bmatrix} q \quad (\text{A.58})$$

The strain-displacement operator  $\mathbf{B}$  can be written as

$$\mathbf{B} = \begin{bmatrix} \phi_{,1}^1 & 0 & \phi_{,1}^2 & 0 & \phi_{,1}^3 & 0 & \phi_{,1}^4 & 0 \\ 0 & \phi_{,2}^2 & 0 & \phi_{,2}^2 & 0 & \phi_{,2}^3 & 0 & \phi_{,2}^4 \\ \phi_{,2}^1 & \phi_{,1}^1 & \phi_{,2}^2 & \phi_{,1}^2 & \phi_{,2}^3 & \phi_{,1}^3 & \phi_{,2}^4 & \phi_{,1}^4 \end{bmatrix} \quad (\text{A.59})$$

Where  $\phi_{,1} = \frac{\partial \phi}{\partial x_1}$  and  $\phi_{,2} = \frac{\partial \phi}{\partial x_2}$ , the strain-displacement operator  $\mathbf{B}$  can be written in the normalized coordinates by making use of the relation

$$\frac{\partial \phi}{\partial \zeta_\alpha} = \frac{\partial \phi}{\partial x_i} \frac{\partial x_i}{\partial \zeta_\alpha} \quad (\text{A.60})$$

In a more general form, (A.60) can be written as

$$\begin{aligned}
 \phi_{,\alpha} &= \phi_{,i} \mathbf{J}_{\alpha i} \\
 \text{or} \quad \phi_{,\alpha} &= \mathbf{J} \phi_{,i} \\
 \text{where} \quad \mathbf{J} &= \begin{bmatrix} \frac{\partial x_1}{\partial \zeta_1} & \frac{\partial x_2}{\partial \zeta_1} \\ \frac{\partial x_1}{\partial \zeta_2} & \frac{\partial x_2}{\partial \zeta_2} \end{bmatrix}
 \end{aligned} \tag{A.61}$$

$\mathbf{J}$  is known as the Jacobian matrix. For the rectangular element in Figure A.9 it is obvious that  $\frac{\partial x_1}{\partial \zeta_1} = a$  and  $\frac{\partial x_2}{\partial \zeta_2} = b$ .

The edge tractions can be calculated in accordance with (A.53) in the normalized coordinates as

$$\underbrace{R^m}_{\text{Consistent nodal load}} = \int_{-1}^1 \left\{ \begin{array}{l} \phi^1(\zeta_1, \zeta_2) \\ \phi^2(\zeta_1, \zeta_2) \end{array} \right\} p^* d\zeta_1 d\zeta_2 \tag{A.62}$$

(A.62) results in equivalent nodal loads for the distributed force applied. If the force had any other form besides that of a fixed traction value  $p^*$ , the traction would have to be written in the form of the generalized coordinates  $p^*(\zeta_1, \zeta_2)$ . It is, however, clear that the integral being evaluated in (A.62) is dependent on the form functions. This will give rise to the following system of equations:

$$\begin{aligned}
 \mathbf{Kq} &= \mathbf{R} \\
 \text{or in a more general numerical form} \\
 \mathbf{AX} &= \mathbf{F}
 \end{aligned} \tag{A.63}$$

Thus the finite element method creates a system of equations that solves the displacements at the nodes under consideration. With the displacements known, the strains are known for the problem and from the stress-strain relationship, (A.9) the stresses can be calculated. The strains are the derivatives of the displacements and are thus less accurate than the displacements [55].

In the previous example only one element was discussed. For more than one element, the principles remain the same, the only difference being that most nodes except for those on the boundary will form part of more than one element. Each element is set up in the normalized coordinates and the stiffness is translated to the global coordinates. The stiffness component of each element must be added together to form the global stiffness matrix. The size of the system of equations under consideration is dependent on the number of nodes in the system  $m$ , the number of degrees of freedom per node  $n$  and the number of nodes restrained by the *essential boundary conditions*  $e$ . Thus the total number  $N$  in the system to be solved can be calculated as

$$N = (m \times n) - e \tag{A.64}$$

It is clear that to better approximate the governing equations, the finite elements used should decrease in size resulting in more elements. One way in which the accuracy of each element is improved is by using more nodes per element. Thus far we have discussed isoparametric formulation where it is assumed that the interpolation function (shape function) is of the same order as the assumed displacement function. To be able to use higher order elements with such functions, the same principles apply. The shape function must be equal to one in the node under consideration and be zero in all other nodes. If we consider one nine-noded element (the equivalent of four connected four-noded elements) in the normalized local coordinates  $(\zeta_1, \zeta_2)$ , the element is still square with a maximum of one and a minimum of minus one. This occurs if we consider the first point at  $(1, 1)$  and its corresponding shape function can be set up by logic. If the line exists where  $\zeta_1 = -1$ , the function is required to be zero so a term  $\times(\zeta_1 + 1)$  is added to the equation. This will make all the terms on the line zero for the form function. The next line can be drawn at  $\zeta_1 = 0$  (middle nodes on  $\zeta_1$ ). For this a term  $\times\zeta_1$  is added. Now on the  $\zeta_2$  there are two more that must be set equal to zero where  $\zeta_2 = -1$  and where  $\zeta_2 = 0$ . In order to do this,  $\times(\zeta_2 + 1)\zeta_2$  is added. Now the first form function

$$\phi = \frac{a}{b}(\zeta_1 + 1)\zeta_1(\zeta_2 + 1)\zeta_2 \quad (\text{A.65})$$

By substituting the values of  $\zeta_1 = 1$  and  $\zeta_2 = 1$  into (A.65) which must be equal to one at this point, the fraction  $\frac{a}{b} = \frac{1}{4}$  is obtained. The remaining form functions can be constructed in the same manner.

### A.3.3 Implementation

Thus far, the implementation of the problem to be solved has been considered using analytical solutions. We use computer programs to solve the finite element equations. Kwon [57] offers a good explanation on how to become acquainted with the implementation. He uses Matlab as the programming language which makes the code easy to change and use. The code provided will be used to calculate the displacements of the single element problem shown in Figure A.10. This problem cannot be considered to be truly representative, but it has to deliver a constant stress in the direction of the applied load.

Using modified code from [57], the displacements as shown in Table A.2 and then the stresses as shown in Table A.3 were calculated.

Node	Displacement x	Displacement y
1	-1.0069e-016	-9.6507e-017
2	5.0000e-001	8.4198e-017
3	5.0000e-001	-1.5000e-001
4	0	-1.5000e-001

Table A.2: Displacements calculated by FEM (single membrane element)

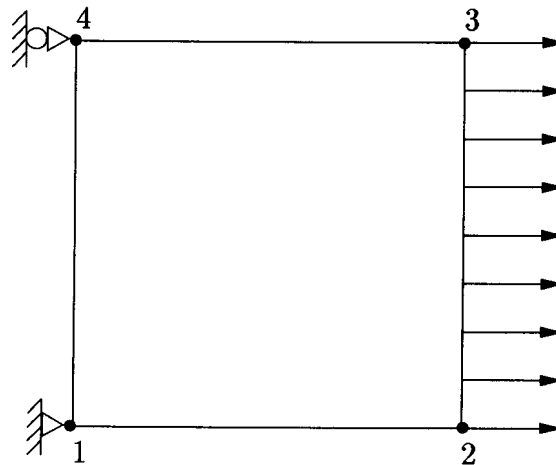


Figure A.10: FEM element patch test

Node	$\sigma_x$	$\sigma_y$	$\tau_{xy}$
1	5.0000e+005	1.1642e-010	9.0739e-011
2	5.0000e+005	5.8208e-011	6.9389e-011
3	5.0000e+005	5.8208e-011	6.4051e-011
4	5.0000e+005	0	4.2701e-011

Table A.3: Stresses Calculated by FEM (single membrane element)

## A.4 Boundary element method

### A.4.1 Introduction

Boundary element methods (BEM) are approximations made only on the boundary or surface of a domain. The approximation is based on a solution of a boundary integral equation. Many important engineering problems can be reduced to mathematical models that belong to a class of problems known as **boundary value problems** [58]. It is easy to understand that for most geometries, boundary elements which only use a mesh on the surface will require less elements than the same structure meshed with finite elements. The method also entails that the dimension of the elements are always one dimension less than the dimension of the defined problem. This entails that for a three-dimensional problem only two-dimensional elements are required to mesh the surface. The discretization is even simpler when using discontinuous elements [54]. The use of elements which sometimes do not meet at corners and are consequently discontinuous in terms of their variables are possible [54]. However, BEM can only be applied if the fundamental solution to the governing partial differential is known. BEM entails two steps in the solving of the problem. The solution is first calculated on the boundary and then internal points are calculated in two separate steps.

### A.4.2 Basic principles

Before BEM can be applied to a problem, a fundamental solution must be obtained. This section will handle some basic concepts which will allow us to obtain a fundamental solution later. The fundamental solution to Poisson's equation, (A.2) in two dimensions is known to be

$$u(p, q) = \frac{1}{2\pi} \ln\left(\frac{1}{r}\right) \quad (\text{A.66})$$

In (A.66),  $p$  can be seen as the source point and  $q$  the observation point. Each of these two points consists of two Cartesian coordinates which can be used to calculate the radius  $r$  in (A.66). Differentiating (A.66) with respect to  $x$  at point  $q$  we obtain

$$\frac{\partial u}{\partial x} = \frac{x_0 - x}{2\pi r} \quad (\text{A.67})$$

Similarly, an expression can be obtained with respect to  $y$ . An approximate solution  $u$  of the form in (A.68) is now considered:

$$u = \alpha_1 \phi_1 + \alpha_2 \phi_2 + \alpha_3 \phi_3 + \dots \quad (\text{A.68})$$

$\alpha_i$  are unknown coefficients and the  $\phi_i$ s are a set of linearly independent functions which are known. In general engineering problems,  $\alpha_i$ 's are considered as nodal values as they have a clear physical meaning. This is the manner of implementation for finite elements, finite differences and boundary elements. Introducing the approximation of  $u$  into the governing differential equation, it is clear that except for the case where enough coefficients and functions are present in (A.68), the approximation produces an error or residual function [54]. The residual function can be defined as

$$R = \nabla^2 u - b \quad (\text{A.69})$$

With approximations, errors also occur in the boundary conditions. The numerical methods used in engineering try to reduce these errors to a minimum by applying different techniques. This reduction is carried out by forcing the errors to be zero at certain points, regions or in a mean sense [54]. This operation can generally be interpreted as *distributing* these errors [54]. Forcing these errors to be zero is generally carried out by weighted residual techniques. If we presume that we have a domain  $\Omega$  surrounded by a boundary  $\Gamma$ , one can introduce the idea of multiplying (A.2) with an arbitrary weight function  $\omega$  to obtain

$$\int_{\Omega} (\nabla^2 u - b) \omega d\Omega = 0 \quad (\text{A.70})$$

The integration of (A.70) is done by integration of parts. This can be seen in [54]. The result is (A.71) which is known as Green's theorem. Although this theorem is in many cases the starting point for many engineering applications, including boundary element formulations, it is much more illuminating to use the concept of distribution as it illustrates the degree

of continuity required of the function and the importance of the accurate treatment of the boundary conditions [54]:

$$\int_{\Omega} [(\nabla^2 u)w - (\nabla^2 w)u] d\Omega = \int_{\Gamma} \left( \frac{\partial u}{\partial n} w - u \frac{\partial w}{\partial n} \right) d\Gamma \quad (\text{A.71})$$

The boundary  $\Gamma$  can now be divided into two segments  $\Gamma_1$  and  $\Gamma_2$  such that  $\Gamma = \Gamma_1 + \Gamma_2$  which surround the domain  $\Omega$ . We furthermore define a unit vector  $n$  so that it has the properties that it exists on every point on the boundary and at each point it is perpendicular to the boundary. We also define the boundary conditions that must be satisfied.

$$\begin{aligned} u &= \bar{u} & \text{on } \Gamma_1 \\ q &= \frac{\partial u}{\partial n} = \bar{q} & \text{on } \Gamma_2 \end{aligned} \quad (\text{A.72})$$

Rewriting (A.71) to separate the boundary and using the boundary conditions (A.72) we obtain

$$\int_{\Omega} [(\nabla^2 u - b)w] d\Omega - \int_{\Gamma_2} (q - \bar{q})w d\Gamma + \int_{\Gamma_1} (u - \bar{u}) \frac{\partial w}{\partial n} d\Gamma = 0 \quad (\text{A.73})$$

This equation shows that one is trying to satisfy a differential equation in the domain  $\Omega$  plus two types of boundary conditions, the *essential boundary conditions*  $u = \bar{u}$  on  $\Gamma_1$  plus the *natural boundary conditions*  $q = \bar{q}$  on  $\Gamma_2$  [54]. (A.74) defines the residual functions (A.69) on the segmented boundary.

$$\begin{aligned} R_1 &= u - \bar{u} & \text{on } \Gamma_1 \\ R_2 &= q - \bar{q} & \text{on } \Gamma_2 \end{aligned} \quad (\text{A.74})$$

If we write (A.73) in terms of our residual functions we obtain

$$\int_{\Omega} R w d\Omega - \int_{\Gamma_2} R_2 w d\Gamma + \int_{\Gamma_1} R_1 \frac{\partial w}{\partial n} d\Gamma = 0 \quad (\text{A.75})$$

Considering a special case of this function where the function  $u$  exactly defines the *essential boundary conditions*,  $u = \bar{u}$  on  $\Gamma_1$ , which means  $R_1 = 0$ , (A.71) becomes

$$\int_{\Omega} R w d\Omega = \int_{\Gamma} R_2 w d\Gamma \quad (\text{A.76})$$

Again using integration by parts, (A.77) is obtained.

$$\int_{\Omega} (\nabla^2 w) u d\Omega = - \int_{\Gamma_2} \bar{q} w d\Gamma - \int_{\Gamma_1} q w d\Gamma + \int_{\Gamma_1} \bar{u} \frac{\partial w}{\partial n} d\Gamma + \int_{\Gamma_2} u \frac{\partial w}{\partial n} d\Gamma \quad (\text{A.77})$$

(A.77) can be seen as the starting point for boundary element formulation of the Laplace equation. This equation is also seen as the weak formulation on which it is presumed that the *essential boundary conditions* have been satisfied.



### A.4.3 Fundamental solution

Now we will attempt to find a fundamental solution by making use of the principles described in section A.4.2. If (A.9) and (A.5) are substituted into (A.3), the result obtained is the Navier equation in terms of the displacements.

$$\left(\frac{1}{1-2\nu}\right)u_{j,jl} + u_{l,jj} + \frac{1}{\mu}b_l = 0 \quad (\text{A.78})$$

$\mu$  is the shear modulus which is usually denoted by  $G$  but in this case  $G$  will be used for Galerkin's vector.

$$\mu = \frac{E}{2(1+\nu)} \quad (\text{A.79})$$

Kelvin's solution of (A.78) when a unit contributed load applied at a point  $i$  in the direction of the unit vector  $e_i$  is given by

$$b_l = \Delta^i e_l \quad (\text{A.80})$$

An easy way of computing the fundamental solution is by using the representation of the displacements in terms of Galerkin's vector [54]. Galerkin's method implies that the same weight function as the interpolating function is used. Therefore one assumes a vector  $G$  from which the displacement components may be obtained as [54].

$$u_j = G_{j,mm} - \frac{1}{2(1-\nu)}G_{m,jm} \quad (\text{A.81})$$

Substitution of (A.80) and (A.81) into (A.78) gives

$$\nabla^2(\nabla^2 G_l) + \frac{1}{\mu}\Delta^i e_l = 0 \quad (\text{A.82})$$

Through a series of substitutions and integrations shown in [54] we can obtain an expression for  $G$

$$G = \frac{1}{8\pi\mu}r^2 \ln\left(\frac{1}{r}\right) \quad (\text{A.83})$$

This equation is valid for three dimensions. If we take each load as independent, one can write

$$G_{lk} = G\delta_{lk} \quad (\text{A.84})$$

$G_{lk}$  is the  $k$  component of Galerkin's vector at any point when a unit load is applied at ' $i$ ' in the  $l$  direction [54]. If we now return to our equation obtained from Galerkin's vector

(A.81) and we presume at any point that we have  $u_{lk}^*$  as the displacement at any point in the  $k$  direction when a unit load is applied at  $i$  in the  $l$  direction, we obtain

$$u_{lk}^* = G_{lk,mm} - \frac{1}{2(1-v)} G_{lm,km} \quad (\text{A.85})$$

By substituting (A.83) and (A.84) into (A.85) and simplifying it for two dimensions (A.86) is obtained.

$$u_{lk}^* = \frac{1}{8\pi\mu(1-v)} \left( (4v-3)(\ln(r))\delta_{lk} + \frac{r_k r_l}{r^2} \right) \quad (\text{A.86})$$

Stresses at any point can be written using the strain-displacement relation (A.5) and the stress-strain equation (A.9). By rearranging these equations they can be expressed as

$$\sigma_{kj}^* = S_{lkj}^* e_l \quad (\text{A.87})$$

Where we need to obtain  $S_{lkj}^*$  for (A.86) this is shown clearly in [54]. The tractions of surface forces on the  $\Gamma$  boundary with

$$p_k^* = p_{ik}^* e_i \quad (\text{A.88})$$

Now we can write the traction components  $p_{ik}^*$  in two dimensions as

$$p_{ik}^* = -\frac{1}{4\pi(1-v)r} \left[ \frac{r_l n_l + r_k n_k}{r} \left[ (1-2v)\delta_{lk} + 2\frac{r_k r_l}{r^2} \right] + (1-2v) \left( n_l \frac{r_k}{r} - n_k \frac{r_l}{r} \right) \right] \quad (\text{A.89})$$

To better explain the results obtained from (A.89) and (A.86), we will now consider a unit load placed at point  $P$  along the 1 2 direction as shown in Figure A.11 from [59].

We now find that (A.86) and (A.89) can be written as matrices shown in (A.90) and (A.91).

$$[u^*] = \begin{bmatrix} u_{11}^* & u_{12}^* \\ u_{21}^* & u_{22}^* \end{bmatrix} \quad (\text{A.90})$$

$$[p^*] = \begin{bmatrix} p_{11}^* & p_{12}^* \\ p_{21}^* & p_{22}^* \end{bmatrix} \quad (\text{A.91})$$

In (A.90) and (A.91),  $u_{11}^*$ ,  $u_{12}^*$  are the deformations and  $p_{11}^*$ ,  $p_{12}^*$  are the tractions. In this case, a unit-concentrated load along direction 1 is placed at any point  $P$  inside or on the surface of the two-dimensional body [59].

#### A.4.4 Boundary integral formulation

We now consider that one needs to minimize the errors involved in the numerical approximation of the governing equations in elastostatics (A.3). We know that we have to satisfy the

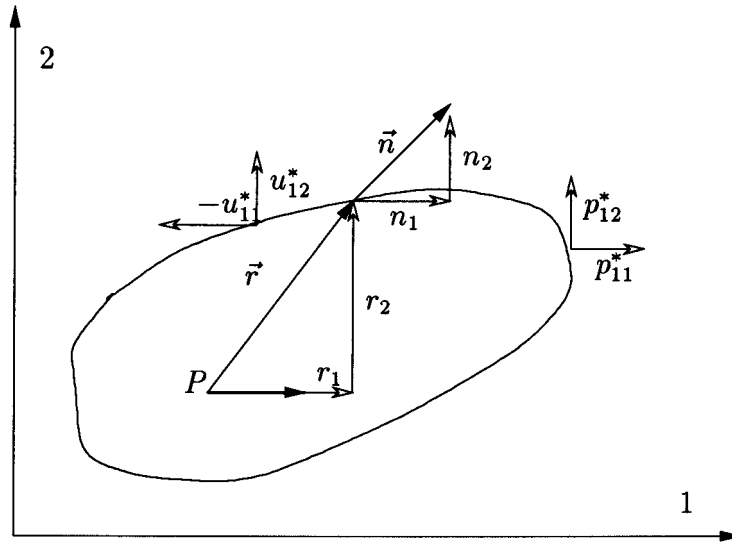


Figure A.11: The fundamental solution in two dimensions with unit force along direction 1

prescribed displacements (*Essential boundary conditions*) and tractions (*Natural boundary conditions*). If we rewrite (A.70) in terms of our governing equation, (A.3), we obtain

$$\int_{\Omega} (\sigma_{kj,j} + b_k) u_k^* d\Omega = 0 \quad (\text{A.92})$$

In this equation we define our weight function as a displacement type function  $u_k^*$ . We now use integration by parts similar to section A.4.2 (details shown in [54]) to obtain

$$u_l^i + \int_{\Gamma} p_{ik}^* u_k d\Gamma = \int_{\Gamma} u_{ik}^* p_k d\Gamma + \int_{\Omega} u_{ik}^* b_k d\Omega \quad (\text{A.93})$$

This equation is known as Somigliana's identity and gives the value of the displacements at any internal points in terms of the boundary values  $u_k$  and  $p_k$ , the forces throughout the domain and the known fundamental solution [54]. (A.93) is valid for any particular point 'i' where forces are applied [54]. However, for the solution to a boundary element problem we are interested in the values at the boundary  $\Gamma$  and not over the whole domain  $\Omega$ . A singularity occurs when solving the integral equation (A.93) at the boundary. The solution to this problem is obtained by defining the boundary  $\Gamma$  in the sense of Cauchy Principal values  $c_{ik}^i$ . This is described in detail in [54] and (A.94) results.

$$c_{ik}^i u_l^i + \int_{\Gamma} p_{ik}^* u_k d\Gamma = \int_{\Gamma} u_{ik}^* p_k d\Gamma + \int_{\Omega} u_{ik}^* b_k d\Omega \quad (\text{A.94})$$

In two dimensions we can define  $c_{ik}^i$  as (A.95) when it is inside  $\Omega$ .

$$[c] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{A.95})$$

When  $c_{ik}^i$  is on the boundary and the boundary  $\Gamma$  is smooth, we obtain (A.96).

$$[c] = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \quad (\text{A.96})$$

When  $c_{ik}^i$  is not smooth, this matrix becomes very difficult to calculate. In two dimensions we will later define the general expression in terms of the angles of the elements.

#### A.4.5 Boundary element formulation

To be able to obtain a numerical solution we must discretize the boundary into a series of elements over which displacements and tractions are calculated at specific nodal points. We begin by defining the  $u$  and  $p$  functions which apply over each element ' $j$ ' as (A.97) and (A.98) respectively.

$$u = \Phi u^j \quad (\text{A.97})$$

$$p = \Phi p^j \quad (\text{A.98})$$

The values of the matrices  $u^j$  and  $p^j$  have a dimension of  $[N \times 1]$  where  $N$  is equal to the number of nodes per element  $n$ , multiplied by  $m$  degrees of freedom per node. The matrix  $\Phi$  then has the dimension of  $[m \times N]$ .  $NE$  elements now exist into which the boundary  $\Gamma$  is divided. We can now write (A.94) in terms of a discretized system using (A.90), (A.91), (A.97) and (A.98) into (A.99)

$$c^i u^i + \sum_{j=1}^{NE} \left( \int_{\Gamma_j} p^* \Phi d\Gamma \right) u^j = \sum_{j=1}^{NE} \left( \int_{\Gamma_j} u^* \Phi d\Gamma \right) p^j + \sum_{s=1}^M \left( \int_{\Omega_s} u^* b d\Omega \right) \quad (\text{A.99})$$

We notice that our body force term  $b$  on the domain  $\Omega$  is divided into  $M$  internal cells over which the boundary force integrals are computed [54]. These body force terms can usually be avoided by taking the body force integrals to the boundary [54]. We can now also use the transformation of coordinates exactly similar to that used in finite elements. For the internal domain we write

$$d\Omega = \|J\| d\zeta_1 d\zeta_2 \quad (\text{A.100})$$

And similarly for the boundary we obtain

$$d\Gamma = \|G\| d\zeta_1 \quad (\text{A.101})$$

$J$  is the full Jacobian matrix in two dimensions.  $G$  is the reduced Jacobian matrix.

$$\|G\| = \frac{d\Gamma}{d\zeta} = \left( \left[ \frac{dx_1}{d\zeta} \right] \left[ \frac{dx_2}{d\zeta} \right] \right)^{\frac{1}{2}} \quad (\text{A.102})$$

With (A.101) and (A.100), we can write (A.99) as (A.103) by making use of numerical integration.

$$c^i u^i + \sum_{j=1}^{NE} \left( \sum_{k=1}^l w_k (p^* \Phi)_k \|G\| \right) u^j = \sum_{j=1}^{NE} \left( \sum_{k=1}^l w_k (u^* \Phi)_k \|G\| \right) p^j + \sum_{s=1}^M \left( \sum_{p=1}^r w_p (u^* b^*)_p \|J\| \right) \quad (\text{A.103})$$

$l$  is the number of integration points on the surface elements and  $w_k$  is the weight at those points [54]. Once (A.103) has been numerically integrated to correspond to a particular node ' $i$ ' it can be written as

$$c^i u^i + \sum_{j=1}^N \hat{H}^{ij} u^j = \sum_{j=1}^N G^{ij} p^j + \sum_{s=1}^M B^{is} \quad (\text{A.104})$$

$N$  is the number of nodes,  $u^j$  and  $p^j$  are the displacements and tractions at node ' $j$ ' [54].  $\hat{H}^{ij}$ ,  $G^{ij}$  and  $B^{is}$  are the matrices obtained after numerically integrating (A.103). We can now simplify the left hand side of (A.104) by considering

$$\begin{aligned} H^{ij} &= \hat{H}^{ij} & \text{if } i \neq j \\ H^{ij} &= \hat{H}^{ij} + c^i & \text{if } i = j \end{aligned} \quad (\text{A.105})$$

If we consider the contribution of all the ' $i$ ' nodes we obtain a global system of equations in

$$\mathbf{H}\mathbf{U} = \mathbf{G}\mathbf{P} + \mathbf{B} \quad (\text{A.106})$$

The vectors  $\mathbf{U}$  and  $\mathbf{P}$  represent all the values of the displacements and tractions before applying the boundary conditions [54]. These conditions can be introduced by rearranging the columns in  $\mathbf{H}$  and  $\mathbf{G}$  and passing all the unknowns to a vector  $\mathbf{X}$  on the left hand side [54]. This gives the final system of equations

$$\mathbf{A}\mathbf{X} = \mathbf{F} \quad (\text{A.107})$$

We also note that the  $\mathbf{B}$  vector has been incorporated into  $\mathbf{F}$ . By solving the linear system of equations all the unknown displacements and tractions are obtained.

#### A.4.6 Implementation

We can now consider a boundary  $\Gamma$  divided into elements and at first we supply each element with a single node in the middle of the element. It follows that we have a smooth boundary and we can use (A.96) in (A.104) to obtain a system of equations to solve the unknowns. It is clear in Figure A.12 that this results in a constant element formulation causing large discontinuities between elements since it assumes a single solution is valid for the entire element.

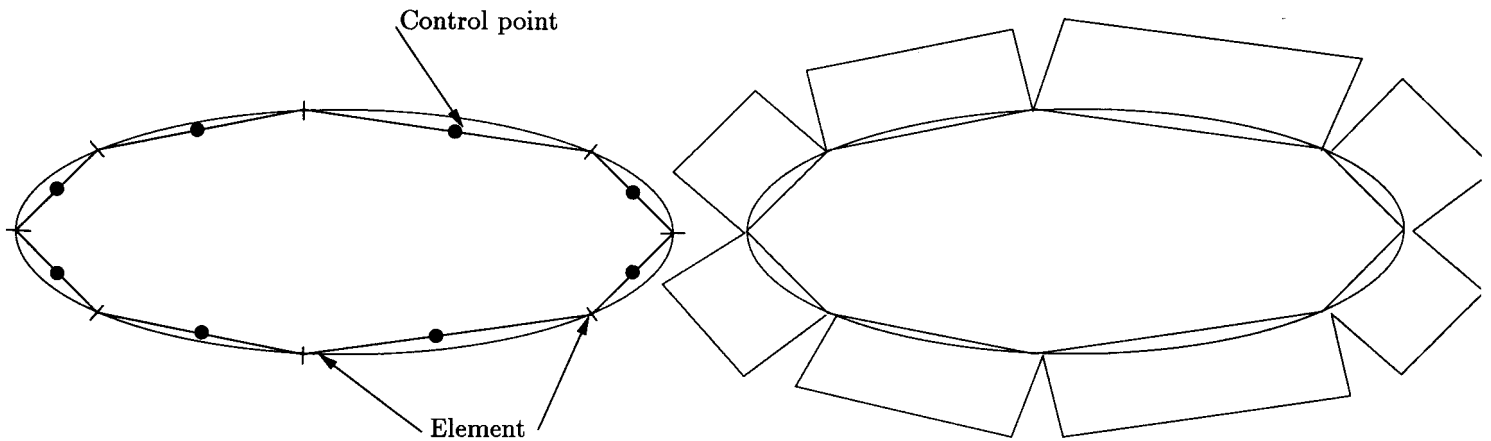


Figure A.12: Constant element solution BEM

Various methods such as “Saw tooth” correction exist to improve the solution of the constant element formulation. But in general it is a well-known fact that low order BEM perform very poorly in structural analysis. To improve this situation we have to put nodes at the edge of elements which in turn entails that we have to obtain a new  $[c]$  matrix for the boundary which is not smooth. To obtain this matrix we consider the limit of the fundamental solution of tractions in (A.108). This is more clearly illustrated in Figure A.13

$$[I] = \lim_{\epsilon \rightarrow 0} \left( \int_{\Gamma_\epsilon} p^* d\Gamma \right) \quad (\text{A.108})$$

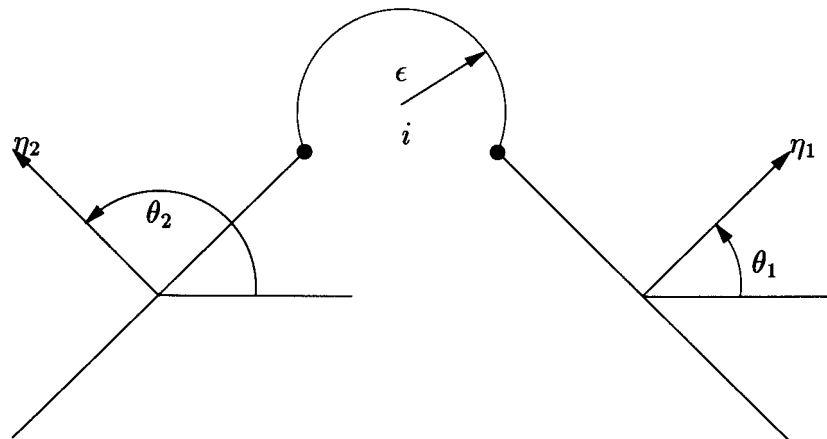


Figure A.13: Corner point

(A.109) gives the equation to describe the two-dimensional case. If we use the conventions described in Figure A.13 we obtain the limit

$$[I] = D \begin{bmatrix} 4(1 - \nu)(\pi + \theta_2 - \theta_1) + \sin 2\theta_1 - \sin 2\theta_2 & \cos 2\theta_2 - \cos 2\theta_1 \\ \cos 2\theta_2 - \cos 2\theta_1 & 4(1 - \nu)(\pi + \theta_2 - \theta_1) + \sin 2\theta_2 - \sin 2\theta_1 \end{bmatrix} \quad (\text{A.109})$$

$$D = \frac{-1}{8\pi(1-\nu)} \quad (\text{A.110})$$

We can now calculate the new  $[c]$  matrix

$$c_{lk} = \delta_{lk} + I_{lk} \quad (\text{A.111})$$

This now allows us to use elements with nodes on the end of each element. But in order to stimulate traction discontinuities over the boundary, the concept of double nodes can be employed [60]. This means that we have two nodes with exactly the same coordinates connected to two different elements. If we have one node at each end of the element we obtain linear elements but, however, this causes problems with curved geometries. It would thus be better to employ an element with two nodes at its corners and one in the middle. For these quadratic elements we can write (A.97) and (A.98) as (A.112) and (A.113) respectively

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \phi_1 & 0 & \phi_1 & 0 & \phi_1 & 0 \\ 0 & \phi_2 & 0 & \phi_2 & 0 & \phi_2 \end{bmatrix} \begin{pmatrix} u_1^1 \\ u_2^1 \\ u_1^2 \\ u_2^2 \\ u_1^3 \\ u_2^3 \end{pmatrix} = \Phi \mathbf{u}^j \quad (\text{A.112})$$

$$p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} \phi_1 & 0 & \phi_1 & 0 & \phi_1 & 0 \\ 0 & \phi_2 & 0 & \phi_2 & 0 & \phi_2 \end{bmatrix} \begin{pmatrix} p_1^1 \\ p_2^1 \\ p_1^2 \\ p_2^2 \\ p_1^3 \\ p_2^3 \end{pmatrix} = \Phi \mathbf{p}^j \quad (\text{A.113})$$

The  $\phi_i$ s are the quadratic interpolation functions as set out in (A.114) in the local coordinate system.

$$\begin{aligned} \phi_1 &= \frac{1}{2}\zeta(\zeta - 1) \\ \phi_2 &= (1 - \zeta^2) \\ \phi_3 &= \frac{1}{2}\zeta(\zeta + 1) \end{aligned} \quad (\text{A.114})$$

By using the same interpolation functions  $\phi_i$ s as we used in (A.112) and (A.113) for the transformation of the coordinates of the nodes we obtain

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \phi_1 & 0 & \phi_1 & 0 & \phi_1 & 0 \\ 0 & \phi_2 & 0 & \phi_2 & 0 & \phi_2 \end{bmatrix} \begin{pmatrix} x_1^1 \\ x_2^1 \\ x_1^2 \\ x_2^2 \\ x_1^3 \\ x_2^3 \end{pmatrix} = \Phi \mathbf{x}^j \quad (\text{A.115})$$

We can now consider a simple example of a square geometry with four elements, one on each side and three nodes per element, when we obtain the setup as shown in Figure A.14.

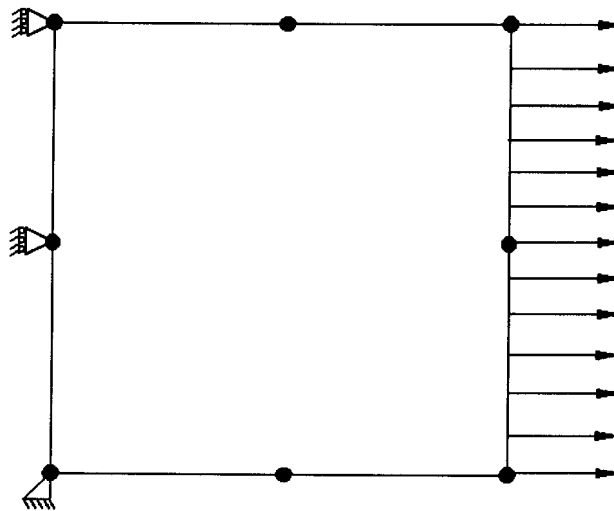


Figure A.14: Simple boundary element problem

Treatment of the boundary conditions are shown in Figure A.15. This clearly shows how the double node concept is employed to satisfy traction discontinuities over the boundary.

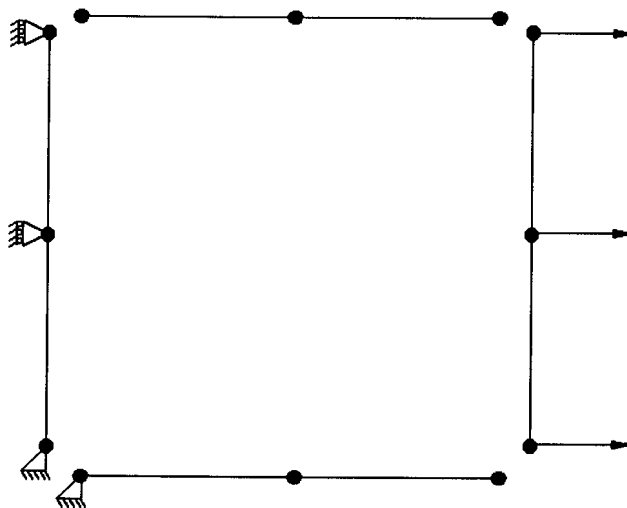


Figure A.15: Applying boundary conditions



We note that the distributed load is treated by only using the value of the force at the point at which it is applied since we are only interested in the tractions applied. This can be very useful when considering a complex distributed load as seen in Figure A.16. This problem would involve quite a complicated integration to be applied to a finite element. The boundary element method allows us to use only the immediate value at the node. This, however, entails that enough nodes have to be placed on the side with the complex load to describe the load accurately.

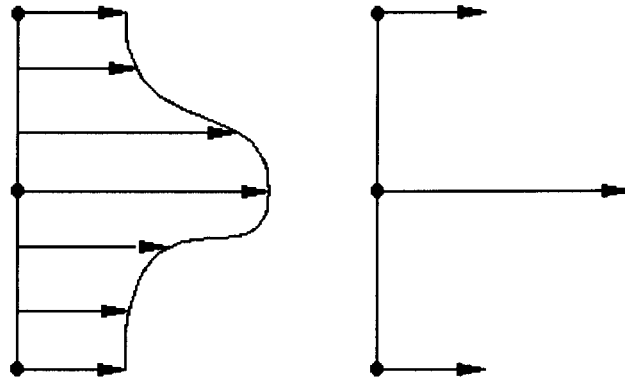


Figure A.16: Applying complex load

When solving the problem in Figure A.14, a modulus of elasticity  $E = 1e6$ , Poisson's ratio  $\nu = 0.3$  and a distributed load of 500000 was used. For this setup, the BEM should be able to represent a constant deformation and stress Figure A.17 shows the solution to the problem. The values used in this problem are not realistic but it does enable us to see if the correct results were obtained. The plot shows the original geometry as well as the deformed geometry.

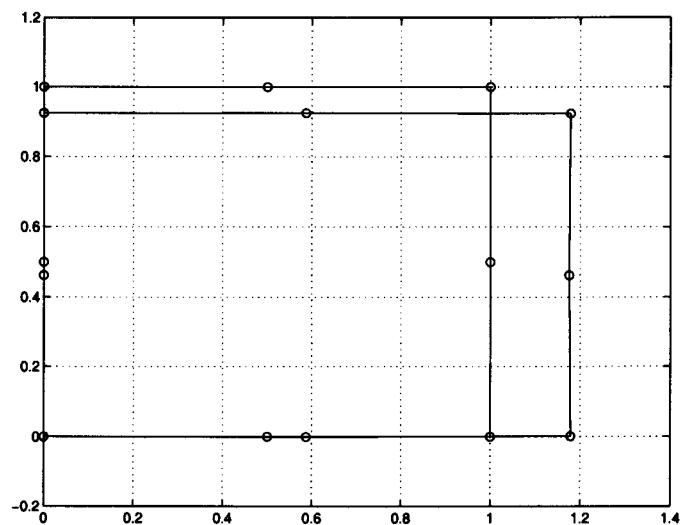


Figure A.17: Constant deformation of a simple geometry

In order to see that the answer obtained has converged, the mesh has to be refined. In Figure

A.18 the solution is plotted for a refined mesh which shows the same deformation as Figure A.17.

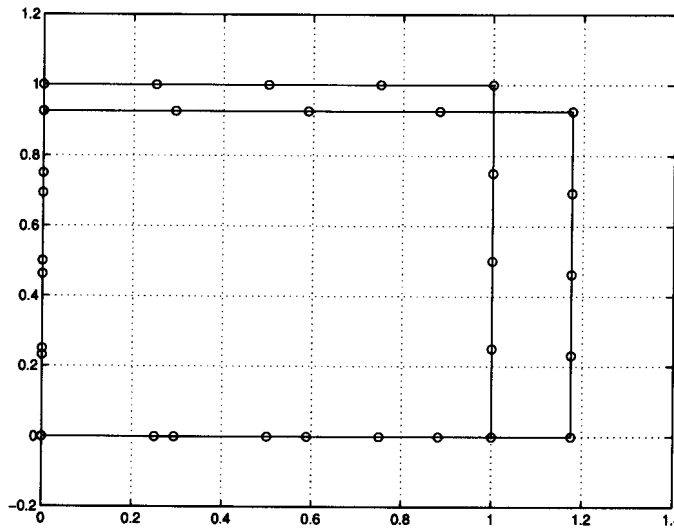


Figure A.18: Constant deformation of a refined mesh

The problem considered should also show a constant stress over the entire geometry. BEM solves the tractions and displacements on the boundary and then solves the internal displacements and stresses at the specified internal points by making use of Somigliana's identity (A.93). Table A.4 gives the internal values calculated by BEM using four internal points. This is done using the mesh in Figure A.17.

Coordinates				
$x$	$y$	$\sigma_x$	$\tau_{xy}$	$\sigma_y$
0.2500000E+00	0.2500000E+00	0.4871852E+06	-0.3897214E+04	0.3686987E+04
0.7500000E+00	0.2500000E+00	0.5055677E+06	0.1617945E+04	0.2063524E+04
0.2500000E+00	0.7500000E+00	0.4879818E+06	0.3100705E+04	0.2811925E+04
0.7500000E+00	0.7500000E+00	0.5063640E+06	-0.8215335E+03	0.1188476E+04

Table A.4: Internal stresses (BEM)

Clearly the stress representation is not very accurate for the internal points since we know that for this problem the stress in the x direction has to be constant over the whole surface. Table A.5 gives the stress at the same internal points calculated using the refined mesh.

The x stresses in Table A.5 are clearly more accurate than for the single elements. But this result is still poor in comparison with the results obtained in finite elements. We know that in our boundary element formulation A.103, the errors are only minimized on the boundary itself. The prediction of the displacements on the internal points are shown in Table A.6.

The displacement in the x direction is definitely more accurate than the stresses calculated at the same internal point. The tractions on displacements on the boundary are, however, calculated very accurately.

Coordinates				
$x$	$y$	$\sigma_x$	$\tau_{xy}$	$\sigma_y$
0.2500000E+00	0.2500000E+00	0.4994989E+06	-0.5405129E+04	0.3858706E+04
0.7500000E+00	0.2500000E+00	0.5002098E+06	-0.9422518E+03	-0.1184704E+04
0.2500000E+00	0.7500000E+00	0.4994980E+06	0.5405965E+04	0.3859719E+04
0.7500000E+00	0.7500000E+00	0.5002088E+06	0.9414868E+03	-0.1183872E+04

Table A.5: Internal stresses refined mesh (BEM)

Coordinates			
$x$	$y$	Displacement $x$	Displacement $y$
0.2500000E+00	0.2500000E+00	0.4550087E-01	-0.1889137E-01
0.7500000E+00	0.2500000E+00	0.1323639E+00	-0.1938061E-01
0.2500000E+00	0.7500000E+00	0.4549616E-01	-0.5655904E-01
0.7500000E+00	0.7500000E+00	0.1323686E+00	-0.5606983E-01

Table A.6: Internal displacements refined mesh (BEM)

## A.5 Finite element method and boundary element method comparison

The boundary element formulation is mathematically complicated and its implementation just as tedious. BEM and FEM have major differences. In a comparison it can be seen as either an advantage or a disadvantage. These differences will be put out more clearly in this section [38].

- 1 FEM: An entire domain mesh is required.

BEM: A mesh of the boundary only is required.

Comment: Because of the reduction in size of the mesh, one often hear people saying that the problem size has been reduced by one dimension. This is one of the major advantages of BEM - construction of meshes for complicated objects, particularly in 3D, is a very time-consuming exercise.

- 2 FEM: Entire domain solution is calculated as part of the solution.

BEM: Solution on the boundary is calculated first, and then the solution at domain points (if required) is found as a separate step.

Comment: There are many problems where the details of interest occur on the boundary, or are localized to a particular part of the domain, and hence an entire domain solution is not required.

- 3 FEM: Reactions on the boundary are typically less accurate than the dependent variables.

BEM: Both  $u$  and  $q$  are of the same accuracy.

4 FEM: Differential equation is being approximated.

BEM: Only boundary conditions are being approximated.

Comment: The use of the Green-Gauss theorem and a fundamental solution in the formulation means that the BEM involves no approximations of the differential Equation in the domain - only in its approximations of the boundary conditions.

5 FEM: Sparse symmetric matrix is generated.

BEM: Fully populated asymmetric matrices generated.

Comment: The matrices are generally of different sizes due to the differences in size of the domain mesh compared to the surface mesh. There are problems where either method can give rise to the smaller system and quickest solution - it depends partly on the volume to surface ratio. For problems involving infinite or semi-infinite domains, BEM is to be favored.

6 FEM: Element integrals are easy to evaluate.

BEM: Integrals are more difficult to evaluate, and some contain integrands that become singular.

Comment: BEM integrals are far more difficult to evaluate. Also the integrals that are the most difficult (those containing singular integrands) have a significant effect on the accuracy of the solution, so these integrals need to be evaluated accurately.

7 FEM: Is widely applicable, and handles non-linear problems well.

BEM: Cannot handle all linear problems.

Comment: A fundamental solution must be found (or at least an approximate one) before the BEM can be applied. There are many linear problems (e.g. virtually any non-homogeneous equation) for which fundamental solutions are not known. There are certain areas in which the BEM is clearly superior, but it can be rather restrictive in its applicability.

8 FEM: Is relatively easy to implement.

BEM: Is much more difficult to implement.

Comment: The need to evaluate integrals involving singular integrands makes the BEM at least an order of magnitude more difficult to implement than a corresponding finite element procedure.

# Appendix B

## Numerical optimization

### B.1 Optimization formulation

#### B.1.1 Mathematical definition

Mathematical optimization is the process by which we attempt to minimize a problem of the general mathematical form

$$\text{Minimize } f(x), x = (x_1, x_2, \dots, x_n) \in R^n \quad (\text{B.1})$$

Subject to the constraints

$$g_j(x) \leq 0 \quad j = 1, 2, \dots, m \quad (\text{B.2})$$

$$h_j(x) = 0 \quad j = 1, 2, \dots, r \quad (\text{B.3})$$

where the design variables are  $x_1, x_2, \dots, x_n$   
the objective function  $f(x)$   
the inequality constraints  $g_j(x)$  and  
the equality constraints  $h_j(x)$

The vector  $x$  that solves the problem (B.1) is denoted by  $x^*$ . This is called the optimal solution and  $f(x^*)$  the optimal function value.

#### B.1.2 Solution methodology

When the objective function  $f(x)$  does not exist in terms of a simple analytical function determining the optimal solution  $x^*$  becomes a more complicated procedure. It may not be possible to differentiate the equation and determine the minimum of all the turning points. In engineering, the objective function usually has to be obtained via a simulation. This is a timely process and thus it is not practical to evaluate all functions to determine which

solution is the optimum solution. Instead numerical optimization consists of techniques which allows us to find the optimum solution  $x^*$  by evaluating few functions. The algorithms can be divided into different classes and each class is based on certain principles. Since each problem is different, some algorithms will perform better than others on different problems.

## B.2 Genetic algorithm

### B.2.1 Introduction

Genetic algorithms (GAs) make use of the survival of the fittest strategy found in nature to search the solution space of a function. The principle of the survival of the fittest results in the fittest individuals of any population reproducing and surviving to the next generation. These species are well adapted to their environment. Inferior individuals also have the possibility of surviving [61, 62].

Many optimization algorithms are limited to convex regular functions. Many functions are, however, multi-modal, discontinuous and non-differentiable. These functions have been optimized, using stochastic search techniques. The stochastic techniques do not make use of traditional search techniques such as gradients, Hessians, linearity and continuity. Decision rules or stochastic sampling is used instead to determine the next sampled point. GAs can therefore be used to solve functions that do not possess properties such as continuity, differentiability and satisfaction of the Lipschitz Condition [62].

GAs require the determination of six issues: solution representation (representation of design variables), selection, genetic operators that make up the reproductive function (including crossover and mutation), the creation of the initial population, termination criteria and the evaluation function. Each of these issues will be discussed [62].

### B.2.2 Solution representation

A chromosome representation function determines how the problem is structured in the GA and the genetic operators that are used. The function is required to describe each individual in the population of interest. An initial design population is made up of a sequence of genes. Each chromosome represents a design variable. The design vectors in the design population are improved in subsequent generations by means of the selection, crossover and mutation operators [61, 62].

### B.2.3 Selection function

A probabilistic selection operation is performed based upon an individual's fitness. The operation is such that fitter individuals have a greater chance of being selected to form the mating pool. Individuals may be selected more than once and all individuals in the pool have a chance of being selected [61, 62].

A common selection method uses the following approach:

1. Assign a probability of selection  $P_j$  to each individual  $j$  based on its fitness value.
2. Compare the series of random numbers  $N$  that is generated against the cumulative probability of the population  $C_i = \sum_{j=1}^i P_j$ .
3. Select the appropriate individual  $i$  and copy to the new population if  $C_{i-1} < U(0, 1) \leq C_i$ .

There are a number of different methods of selection determining how probabilities are assigned to individuals: roulette wheel, tournament selection and ranking selection.

### Roulette wheel

The roulette wheel selection is also known as the expected value selection [63]. The objective function is converted from a maximization to a minimization problem by multiplying the objective function with -1. A constant must be added to functions with negative values as function values must be positive. The probability  $P_i$  for each design is calculated as follows:

$$P_i = \frac{F_i}{\sum_{j=1}^{Popsize} F_j} \quad i = 1, 2, 3, \dots \quad (\text{B.4})$$

where  $F_i$  denotes the fitness of individual  $i$ .

### Tournament selection

Tournament selection selects  $j$  individuals randomly and inserts the best  $j$  into the new population. The procedure is repeated until  $N$  individuals have been selected. Tournament selection simulates the process whereby individuals in the population compete for mating rights [61, 62].

### Ranking methods

In ranking method, after ranking in descending order, the relative member's fitness  $P_i$  is expressed as the relative fitness of each member  $i$ .

$$P_i = \frac{t_i}{\sum_{i=1}^e t_i} \quad (\text{B.5})$$

where

$$t_i = 2 \cdot \frac{(e + 1 - i)^c}{(e^2 + e)} \quad (\text{B.6})$$

$c$  is any value between 1 and 10 and  $e$  is the population size. The cumulative probability space  $g_j$  is constructed using the relative fitness  $p_i$  [61, 62].

Normalized geometric ranking defines  $P_i$  as:

$$P_i = q'(1 - q)^{r-1} \quad (\text{B.7})$$

where  $q$  is the probability of selecting the best individual,  $r$  is the rank of the individual where 1 is the best,  $P$  is the population size and

$$q' = \frac{q}{1 - (1 - q)^P} \quad (\text{B.8})$$

### B.2.4 Genetic operators

Genetic Operators are used to create new solutions based on solutions that already exists in the population. Crossover and mutation are the two basic types of operators. Crossover produces a new individual from two individuals. Crossover allows selected individuals to trade characteristics. Mutation alters one individual in the production of a new solution. Mutation protects against complete loss of genetic diversity by randomly changing characteristics of the design variable [61, 62].

Binary mutation and simple crossover are defined for  $\bar{X}$  and  $\bar{Y}$  binary. Binary mutation flips each bit in every individual in the population with probability  $p_m$  as shown in (B.9) [62].

$$x'_i = \begin{cases} 1 - x_i & \text{if } U(0, 1) < p_m \\ x_i & \text{otherwise} \end{cases} \quad (\text{B.9})$$

Simple crossover generates a random number  $r$  from a uniform distribution from 1 to  $m$  and creates two new individuals  $\bar{X}'$  and  $\bar{Y}'$  according to (B.10) and (B.11) [62].

$$x'_i = \begin{cases} x_i & \text{if } i < r \\ y_i & \text{otherwise} \end{cases} \quad (\text{B.10})$$

$$y'_i = \begin{cases} x_i & \text{if } i < r \\ y_i & \text{otherwise} \end{cases} \quad (\text{B.11})$$

Uniform mutation, non-uniform mutation, multi-non-uniform mutation, boundary mutation, simple crossover, arithmetic crossover and heuristic crossover are defined for  $X$  and  $Y$  real. Uniform mutation randomly selects one variable  $j$  and sets it equal to a uniform random number  $U(a_i, b_i)$  as shown by (B.12) [62].

$$x'_i = \begin{cases} U(a_i, b_i) & \text{if } i = j \\ x_i & \text{otherwise} \end{cases} \quad (\text{B.12})$$

Boundary mutation randomly selects one variable  $j$  and sets it equal to either its lower or upper bound where  $r = U(0, 1)$  shown by (B.13) [62].

$$x'_i = \begin{cases} a_i & \text{if } i = j, r < 0.5 \\ b_i & \text{if } i = j, r \geq 0.5 \\ x_i & \text{otherwise} \end{cases} \quad (\text{B.13})$$

Non-uniform mutation randomly selects one variable  $j$  and sets it equal to a non-uniform random number as seen in (B.14) and (B.15) [62].



$$x'_i = \begin{cases} x_i + (b_i - x_i)f(G) & \text{if } r_1 < 0.5 \\ x_i - (x_i + a_i)f(G) & \text{if } r_1 \geq 0.5 \\ x_i & \text{otherwise} \end{cases} \quad (\text{B.14})$$

where

$$f(G) = \left( r_2 \left( 1 - \frac{G}{G_{max}} \right) \right)^b \quad (\text{B.15})$$

$r_1, r_2 =$  a uniform random number between  $(0, 1)$

$G =$  current generation

$G_{max} =$  the maximum number of generations

$b =$  a shape parameter

The multi-non-uniform mutation operator applies the non-uniform operator to all of the variables in the parent  $\bar{X}$  [62].

Real-valued crossover is the same as the binary simple crossover shown in (B.10) and (B.11). Arithmetic crossover results in two complimentary linear combinations of the parents, where  $r = U(0, 1)$  [62].

$$\bar{X}' = r\bar{X} + (1 - r)\bar{Y} \quad (\text{B.16})$$

$$\bar{Y}' = (1 - r)\bar{X} + r\bar{Y} \quad (\text{B.17})$$

Heuristic crossover makes use of the fitness of the individuals to produce a linear extrapolation of the two individuals.  $\bar{X}'$  is created where  $r = U(0, 1)$  and  $\bar{X}$  has a better fitness than  $\bar{Y}$ . If  $\bar{X}'$  is unfeasible, a new random number  $r$  must be generated and a new solution created. After  $t$  failures, the children should be made to equal the parents and the procedure should stop [62].

$$\bar{X}' = \bar{X} + r(\bar{X} - \bar{Y}) \quad (\text{B.18})$$

$$\bar{Y}' = \bar{X} \quad (\text{B.19})$$

$$\text{feasibility} = \begin{cases} 1 & \text{if } x'_i \geq a_i, x'_i \leq b_i \quad \forall i \\ 0 & \text{otherwise} \end{cases} \quad (\text{B.20})$$

### B.2.5 Initialization function

An initial population must be supplied to the GA. A randomly generated solution is commonly chosen. GAs can, however, improve on existing solutions and the beginning population can therefore consist of a mixture of randomly generated individuals seeded with potentially good solutions [62].

### **B.2.6 Termination criteria**

The GA moves from one generation to the next generation and parents are selected and reproduced until the criterion for termination has been met. This criterion is often specified as a maximum number of generations. Termination can also be chosen as population convergence, when the sum of the deviations among individuals becomes smaller than a certain specification. Other criteria may be lack of improvement in the best solution over a specified number of generations or the achievement of some arbitrarily “acceptable” threshold [62].

### **B.2.7 Objective function**

Many forms of evaluation functions may be used in a GA. The minimal requirement of the function is that it maps the population into a partially ordered set. Stochastic decision rules make for the evaluation function being independent of the GA [62].

# Appendix C

## Flow diagrams for selected programs

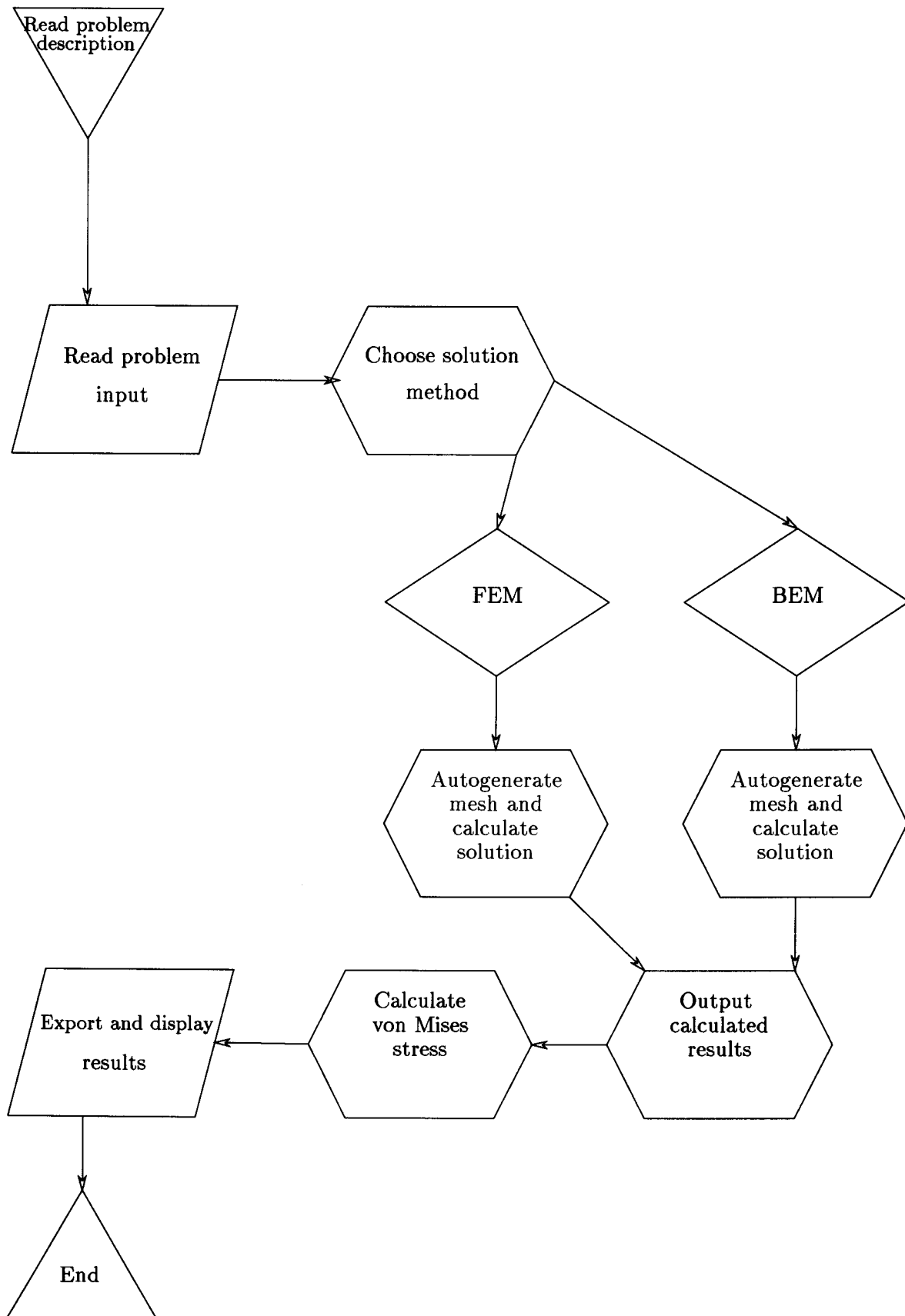


Figure C.1: Boundary cell calculation

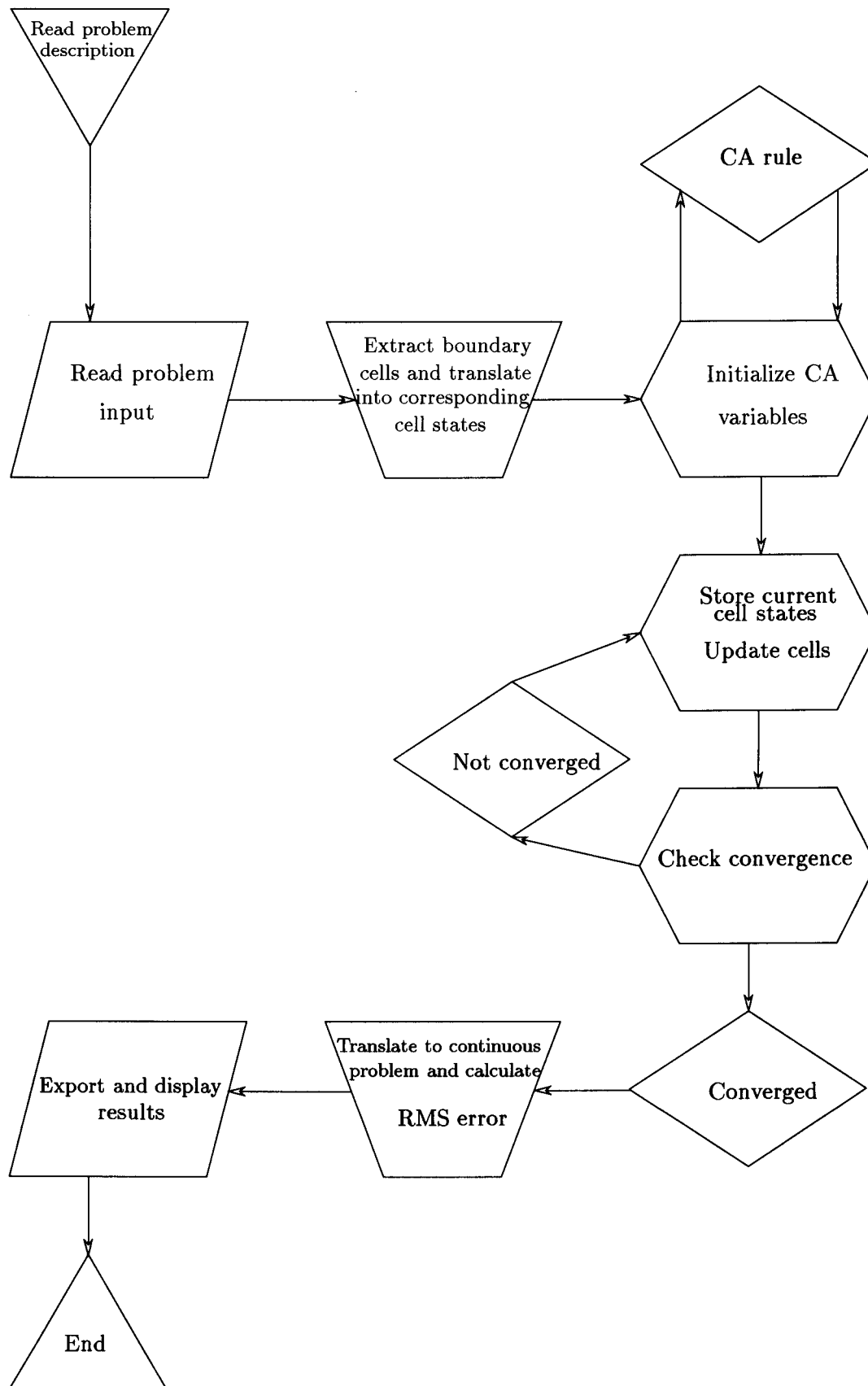


Figure C.2: CA simulation main

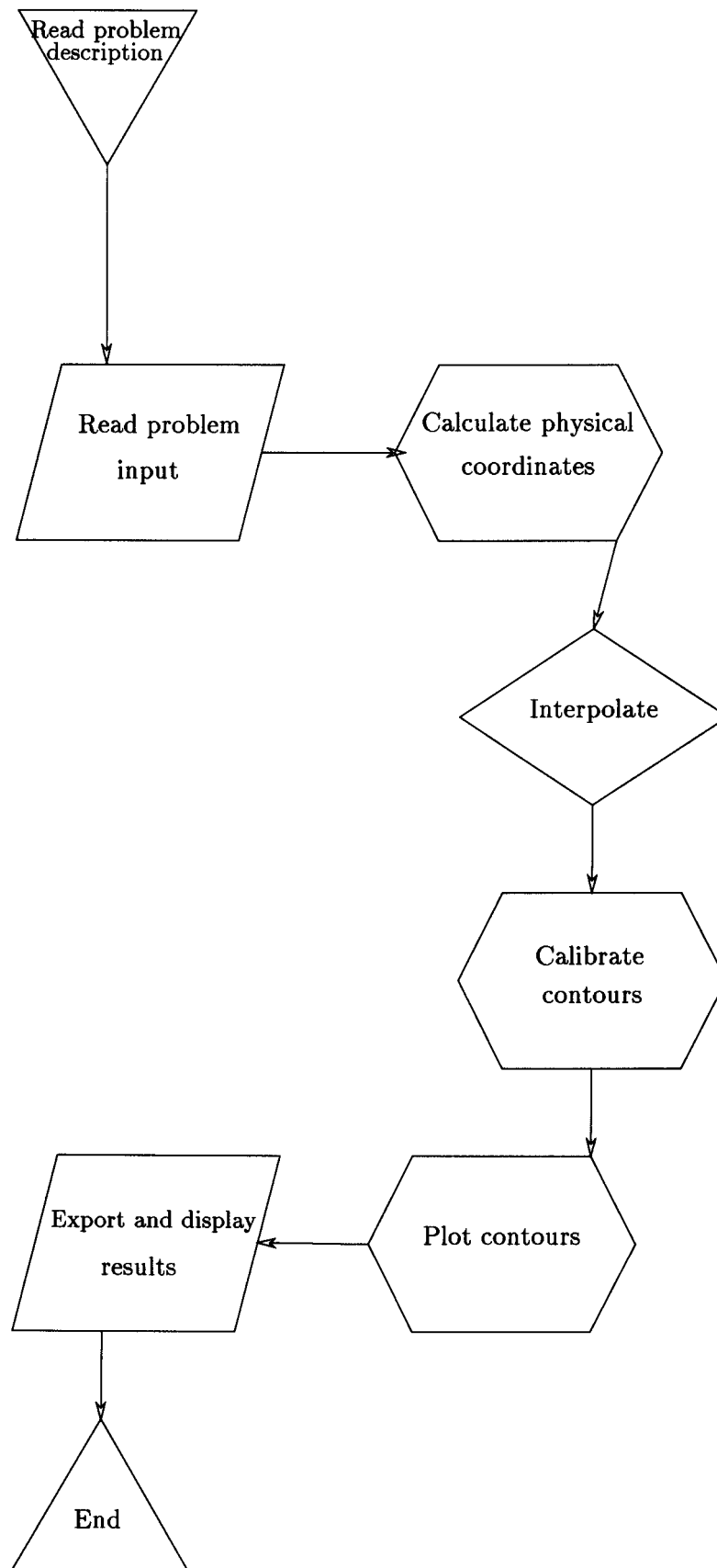


Figure C.3: Stress contour plot

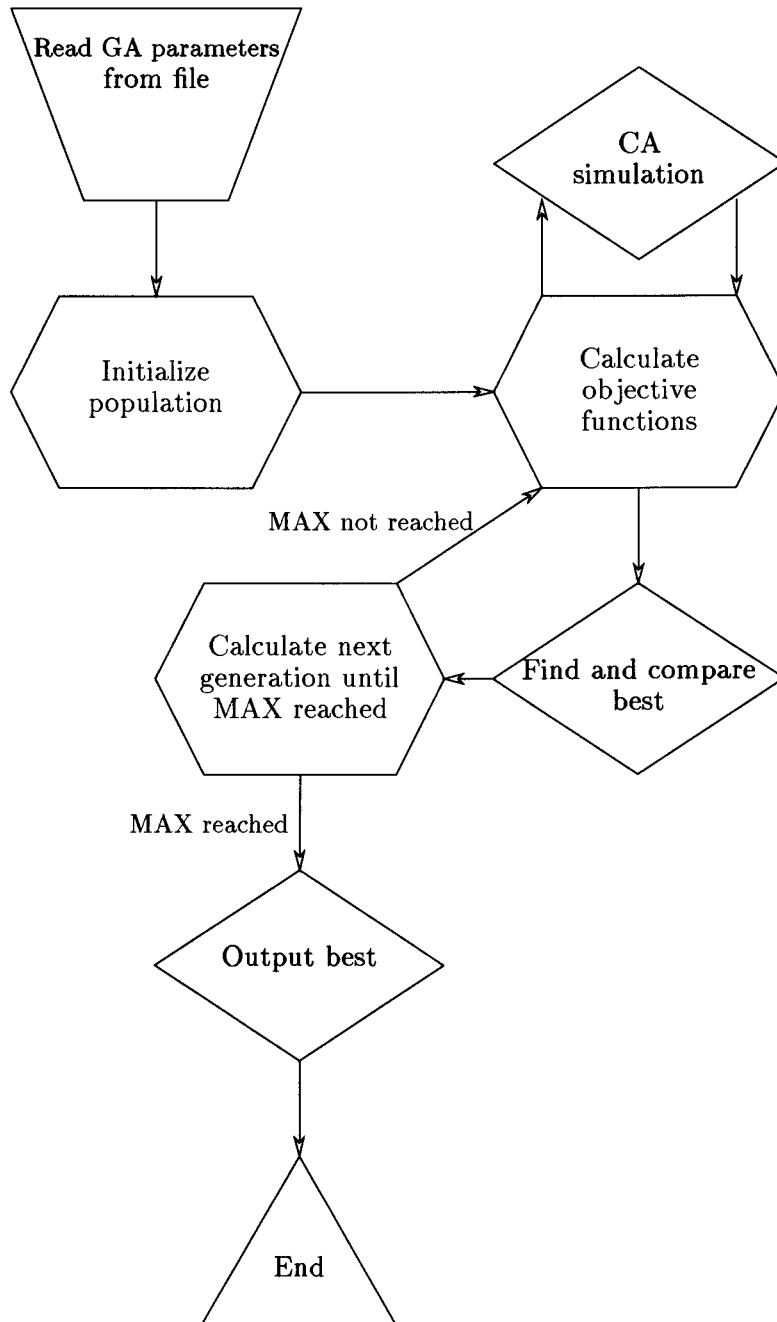


Figure C.4: Genetic algorithm

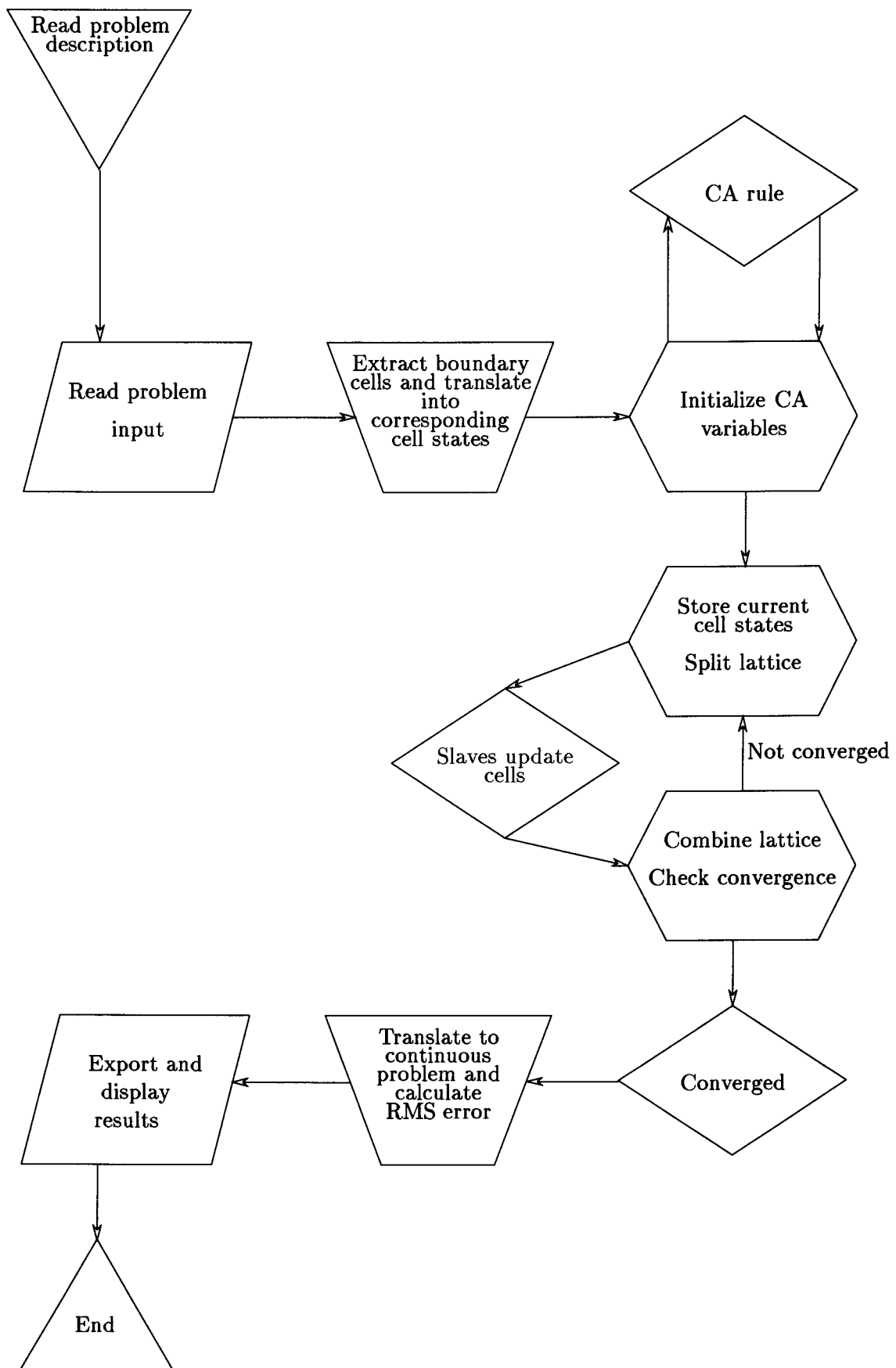


Figure C.5: CA parallel implementation number one



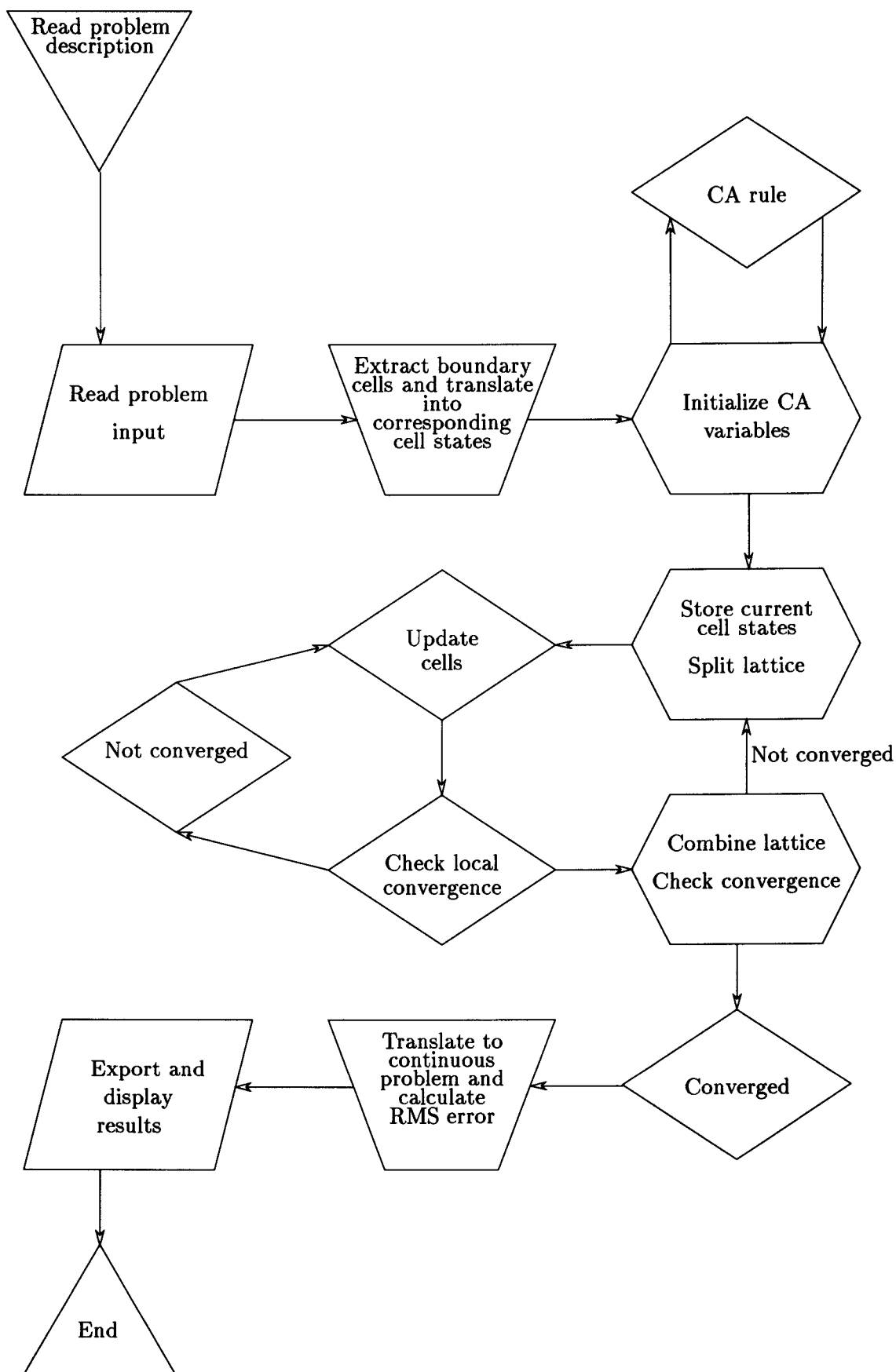


Figure C.6: CA parallel implementation number two

APPENDIX C. FLOW DIAGRAMS FOR SELECTED PROGRAMS

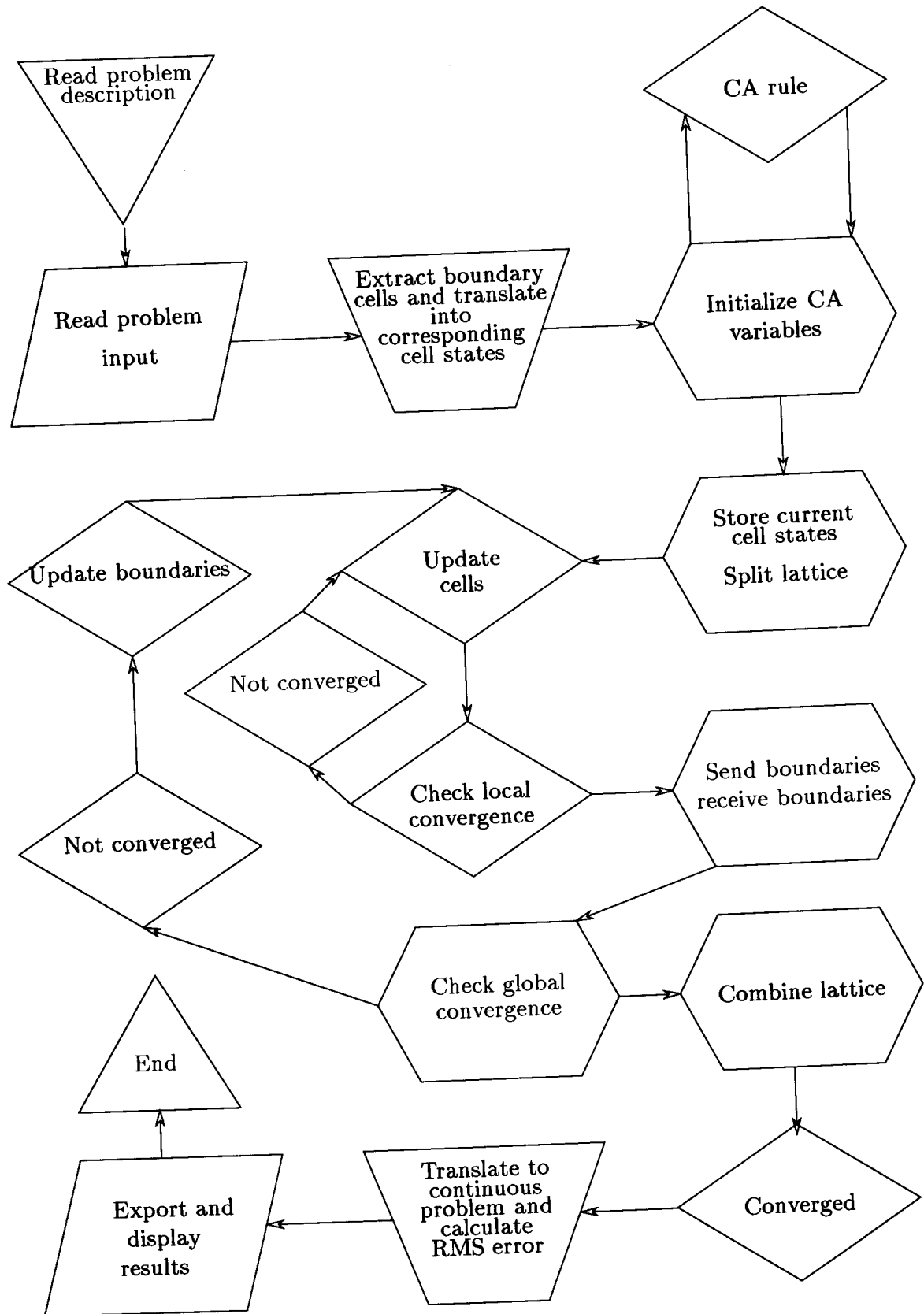


Figure C.7: CA parallel implementation number three

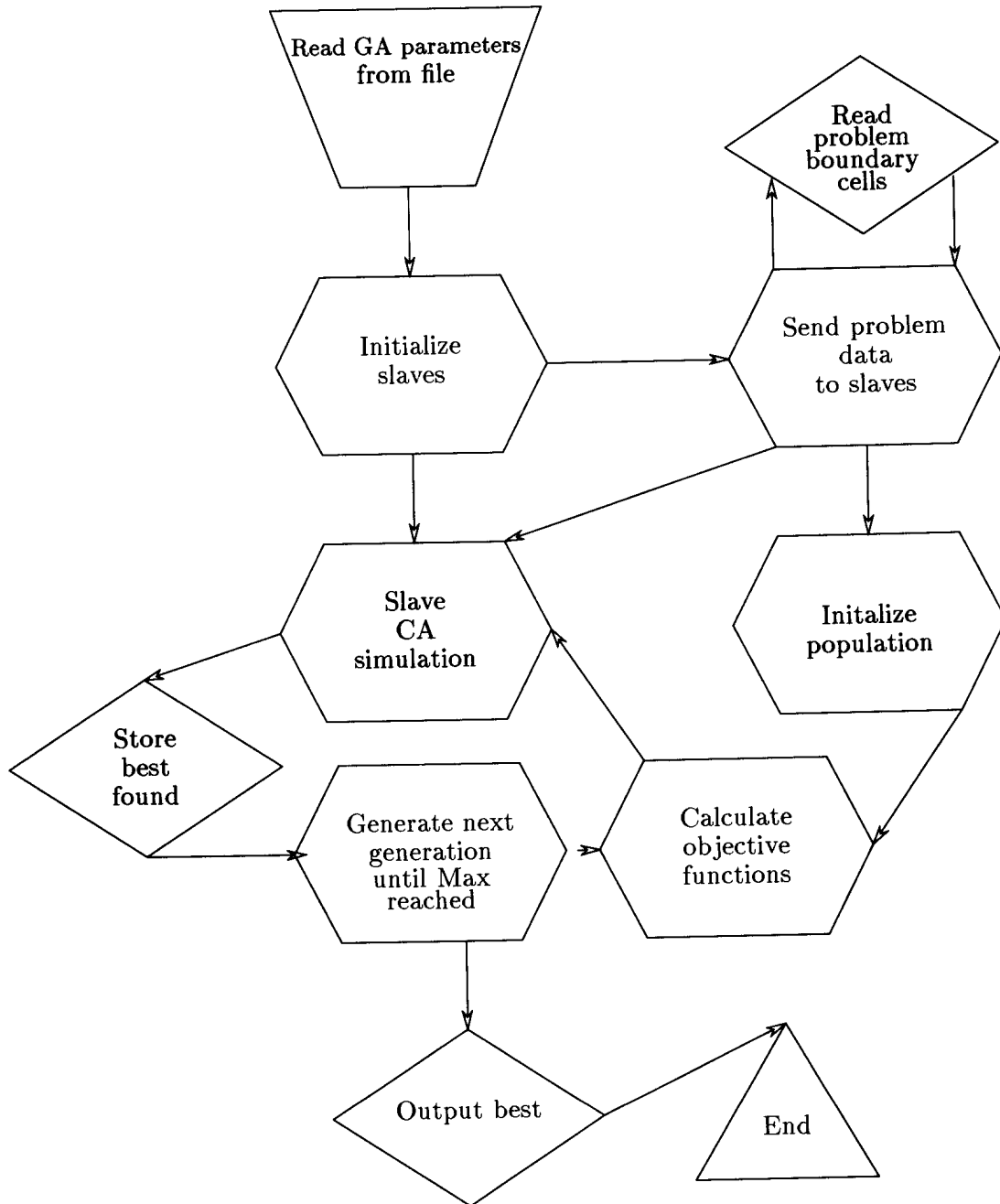


Figure C.8: GA parallel implementation