Implementation of Cell Clustering in Cellular Automata

by

Roxane Adams

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at the University of Stellenbosch

Department of Mathematical Sciences,
Computer Science Division,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. L. van Zijl

March 2011

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2011/02/18

Abstract

Cellular Automata (CA) have become a popular vehicle to study complex dynamical behaviour of systems. CA can be used to model a wide variety of physical, biological, chemical and other systems. Such systems typically consist of subparts that change their state independently, based on the state of their immediate surroundings and some generally shared laws of change.

When the CA approach was used to solve the LEGO construction problem, the best solution was found when using a variant of CA allowing for the clustering of cells. The LEGO construction problem concerns the optimal layout of a set of LEGO bricks. The advantages found for using the CA method with clustering in this case are the ease of implementation, the significantly smaller memory usage to previously implemented methods, and its trivial extension to construct multicoloured LEGO sculptures which were previously too complex to construct.

In our research we propose to explore the definitions of clustering in CA and investigate the implementation and application of this method. We look at the ant sorting method described by Lumer and Faieta, and compare the implementation of this algorithm using regular CA as well as the clustering variation. The ant sorting model is a simple model, in which ants move randomly in space and pick up and deposit objects on the basis of local information.

Opsomming

Sellulêre Outomate (SO) het 'n populêre metode geword om die komplekse dinamiese gedrag van sisteme bestudeer. SO kan gebruik word om 'n groot verskeidenheid fisiese, biologiese, chemiese en ander tipe sisteme te modelleer. Sulke sisteme bestaan tipies uit subafdelings wat, gebaseer op die status van hulle omgewing en 'n paar algemene gedeelde reëls van verandering, hulle status onafhanklik verander.

Met die gebruik van die SO benadering om the LEGO konstruksieprobleem op te los, is die beste oplossing bereik deur gebruik te maak van 'n variant van SO, waar selle saamgroepeer kan word. Die LEGO konstruksieprobleem behels die optimale uitleg van 'n stel LEGO blokkies. In hierdie geval is die voordele van die SO met sel groepering die maklike implementasie, 'n beduidende kleiner geheuegebruik teenoor voorheen geïmplementeerde metodes, en die triviale uitbreiding daarvan om gekleurde LEGO beelde wat voorheen te kompleks was, te kan bou.

In ons ondersoek verken ons die definisies van selgroepering in SO en ondersoek die implementasie en toepassing van die metode. Ons kyk na die miersorteringsmetode beskryf deur Lumer en Faieta, en vergelyk die implementasie van hierdie algoritme deur gewone SO asook die groeperingsvariasie te gebruik. Die miersorteringsmodel is 'n eenvoudige model waarin miere lukraak in 'n omgewing beweeg en voorwerpe optel of neersit volgens plaaslike inligting.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations who have in some way contributed to this thesis:

- my supervisor Prof Lynette van Zijl who has been my mentor for the past few years and who has taught me much. For her patience, invaluable insight and motivation throughout this journey;
- my parents who have raised me and supported me throughout my school and university years;
- my friends and family for their moral support and encouragement when I needed it most;
- the National Research Foundation (NRF) for their financial support; and
- the Creator of all without whom none of this would be possible.

To
the God that created the sun
which shines even when
the clouds cover its light.

Contents

List of Figures

Chapter 1

Introduction

Cellular automata (CA) are automata that are particularly suited to model the complex dynamic behaviour of systems. As such, CA can be used to model a wide variety of physical, biological, chemical and even sociological phenomena. Given a CA model that represents a certain behaviour, it can be applied to many different real-world applications. For example, a model of swarming behaviour may be applied to investigate the behaviour of animal migration, or packet switching in communication networks.

CA were first defined by Von Neumann as a linear multidimensional grid of cells, where each cell contains an automaton. The basic model can be adapted in many different ways; for example, by changing the shape of the grid elements from rectangular to hexagonal [34]. One recent suggestion for a variation is that of so-called cell clustering, where the *size* of a cell is allowed to change as the model evolves [31]. In this thesis, we study cell clustering in CA.

Cell clustering has been shown by Smal [31] to successfully and efficiently solve some optimization problems. However, apart from its basic definition in [31], little theoretical work has been done on CA with cell clustering. Also, apart from the one application in [31], CA with cell clustering have not been investigated for their applicability to other applications, and have not been investigated as far as the practical efficiency of their implementation is concerned.

In this thesis, we therefore start out by giving more strict theoretical definitions for

CA with cell clustering. Then, we consider a well-known scientific model (that of so-called ant sorting). Given the standard implementation of the ant sorting model (ASM) with CA, a solution based on CA with cell clustering is implemented. The differences between these solutions in terms of the efficiency and effectiveness of the implementation are then investigated and discussed.

1.1 Thesis outline

An overview of CA, cell clustering and the ASM are given in Chapter 2. In Section 2.3.1, besides introducing Lumer and Faietas model (LF-model) for ant sorting, an example implementation for this model is described.

Chapter 3 looks at the definition for clustering and discusses how clustering affects the neighbourhood of CA.

In Chapter 4 we extend the LF-model to include cell clustering and modify the example described in Section 2.3.1 to apply to cell clustering. This implementation is then compared with the example implementation in Section 2.3.1.

Chapter 5 contains a set of experiments, which are described in detail and their outcomes analysed.

Finally, in Chapter 6 we present our final conclusions and mention possible future work.

Chapter 2

Literature Overview

CA can be used to model various systems, including physical, biological and chemical systems. Such systems typically consist of sub parts (*cells*) that change their state independently, based on the state of their immediate surroundings (*neighbourhood*) and some shared laws of change (*update rules*).

The basic CA model can be adapted in a variety of ways, including the variation where the *size* of a cell is allowed to change as the model evolves. We call this variation cell clustering.

This chapter contains a brief overview of CA, including a discussion on the implementation and application of CA in real world problems. It then introduces the variation of CA with cell clustering, before focusing on the ASM. The ASM is discussed in the context of its implementation with CA.

2.1 Cellular automata

A CA is defined as a discrete dynamical system[1] that consists of an infinite grid of cells. Each of these cells, c_i, represents an automaton and has a defined neighbourhood. A neighbourhood is defined to be a set of cells that is associated with c_i

[1] A discrete dynamical system is a system that describes how the state of a process changes over each discrete time step [35].

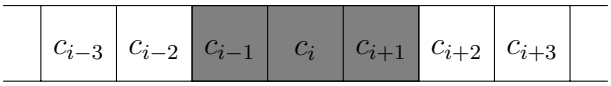


Figure 2.1: 1D cellular automaton $\mathcal{C}$ with an example neighbourhood $\{c_{i-1}, c_i, c_{i+1}\}$ for cell c_i.

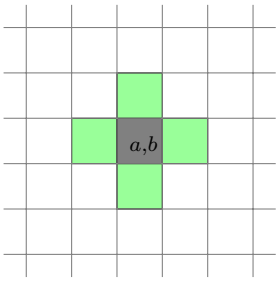
in some way. The cells in this system execute simultaneously, changing their state from one time step, t, to the next, $t + 1$, using a transition rule that is defined in terms of the neighbourhood of each cell.

Although CA are defined as having an infinite grid of cells, a finite grid is used when implemented on computers. This is necessitated by the fact that computers have limited memory. When a finite grid is used, the cells at the edges of the grid are left with incomplete neighbourhoods. The standard solution to overcome this obstacle is to use either null boundary or periodic boundary conditions. Under the null boundary condition the values of the missing cells in the incomplete neighbourhoods are ignored. The periodic boundary condition requires that the cells at the edges of the grid be considered adjacent. For example, in a two-dimensional CA (2D CA) with a grid size of $M \times N$, the cells in row 1 are adjacent to the cells in row M and the cells in column 1 are adjacent to the cells in column N.

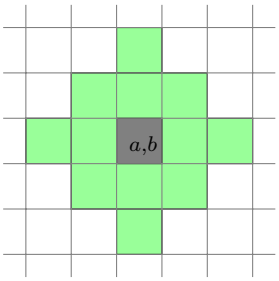
By definition, CA can have a multidimensional grid of cells. For most practical applications, however, only one-, two-, and three dimensional grids are used. The simplest one dimensional CA (1D CA) are called elementary CA and have been extensively researched by Wolfram [36]. Each cell in an elementary CA contains only two states, which are numbered 0 and 1. Elementary CA are also known as binary CA. While 1D CA are interesting to investigate and usually have complex resulting patterns, it is two dimensional and three dimensional CA (3D CA) that are typically used for more realistic applications of natural phenomena.

The higher the dimensions of the CA, the more complex the neighbourhoods that can be defined. For 1D CA the neighbourhood is usually defined as the current cell plus its two nearest neighbours. This is shown in Figure 2.1. Another neighbourhood example for 1D CA is to omit the current cell from the neighbourhood, leaving only the cells c_{i-1} and c_{i+1} in the neighbourhood.

The two most popularly used neighbourhoods in 2D CA are the Von Neumann and



(a) Von Neumann neighbourhood for range $r = 1$.



(b) Von Neumann neighbourhood for range $r = 2$.

Figure 2.2: The classic Von Neumann neighbourhood of 2D cellular automata for ranges $r = 1$ and $r = 2$.

Moore neighbourhoods. The Von Neumann neighbourhood is the diamond shaped grid of cells around a given cell, c_i. The formal definition of a Von Neumann neighbourhood with range r for a cell at position (a, b) can be described as [2]:

$$N^r_{(a,b)} = \{(i, j) \ : \ |i - a| + |j - b| \leq r\} .$$

The Von Neumann neighbourhood for ranges $r = 1$ and $r = 2$ is illustrated in Figure 2.2.

The Moore neighbourhood is a square shaped grid of cells around a given cell, c_i. The formal definition of a Moore neighbourhood with range r for a cell at position (a, b) can be described as [2]:

$$N^r_{(a,b)} = \{(i, j) \ : \ |i - a| \leq r, \ |j - b| \leq r\} .$$

The Moore neighbourhood for ranges $r = 1$ and $r = 2$ is illustrated in Figure 2.3.

Another example of a 2D neighbourhood is the Margolus neighbourhood. This neighbourhood is known as a partition scheme neighbourhood. It partitions time into even and odd time steps and the grid space into blocks of four cells. On every time step, the partitioned blocks are switched to alternate coordinates on the grid. This is illustrated in Figure 2.4.

Both the Von Neumann and Moore neighbourhoods can be extended into three dimensions to be used by 3D CA. The formal definition for the three-dimensional

(a) Moore neighbourhood for range $r = 1$.

(b) Moore neighbourhood for range $r = 2$.

Figure 2.3: The classic Moore neighbourhood of 2D cellular automata for ranges $r = 1$ and $r = 2$.

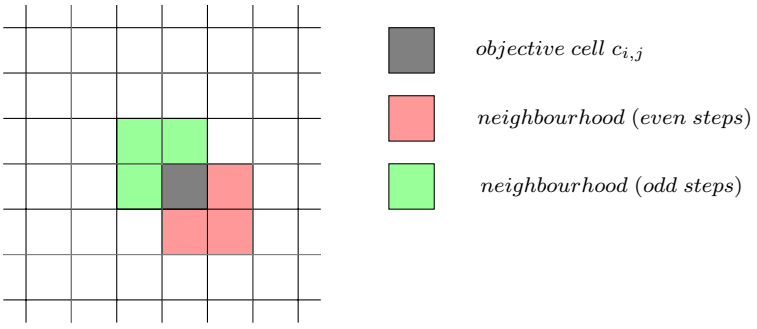


Figure 2.4: The 2D Margolus neighbourhood. The nearest four cells make one block, and the neighbourhood is the block including the cell itself. The boundaries of the blocks change with each step as shown above, alternating the neighbourhood with every time step.

Von Neumann neighbourhood is:

$$N^r_{(a,b,c)} = \{(i,j,k) \ : \ |i-a| + |j-b| + |k-c| \leq r\}\,,$$

and for the three-dimensional Moore neighbourhood:

$$N^r_{(a,b,c)} = \{(i,j,k) \ : \ |i-a| \leq r,\ |j-b| \leq r,\ |k-c| \leq r\}\,.$$

The Margolus neighbourhood can also be extended to three dimensions and is illustrated in Figure 2.5.

The transition function of a CA describes the transitions of the cells in the CA grid, from their current state at time step t to the next state at time step $t + 1$.

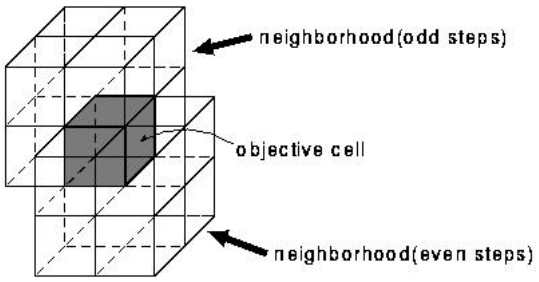


Figure 2.5: An extension of the 2D Margolus neighbourhood, shown in Figure 2.4, to three dimensions (taken from [4]).

Neighbourhood state:	111	110	101	100	011	010	001	000	
Next state:	0	1	0	1	1	0	1	0	(rule 90)
Next state:	1	0	0	1	0	1	1	0	(rule 150)

Figure 2.6: The truth table for rules 90 and 150 (taken from [10]).

This is usually described in terms of the cells in the neighbourhood of the current cell.

As an example of a transition function, consider a three-neighbourhood 1D CA:

$$c_i(t+1) = f[c_{i-1}(t), c_i(t), c_{i+1}(t)],$$

where f denotes the transition function of the CA. Here, the next state of cell c_i is defined as a function of the current states of the cells in the neighbourhood of the CA, in this case $c_{i-1}(t)$, $c_i(t)$ and $c_{i+1}(t)$.

There are 2^3 possible initial configurations for the three cells in the neighbourhood of an elementary CA. This can be mapped to a total of $2^{2^3}(256)$ different outcomes, each representing a rule for an elementary CA. Each of these mappings can be seen as depicting an eight digit binary number which can be translated to its corresponding decimal number. This number is used to identify the transition function rules of elementary CA.

Figure 2.6 shows two such rules. In the first row, the eight initial neighbourhood configurations are shown. The second and third rows show the next state mappings for two example CA rules. For the first example rule, the output 01011010 has a decimal equivalent of 90 and the output, 10010110, for the second rule has a decimal equivalent of 150. The combinatorial logic for these rules are given by:

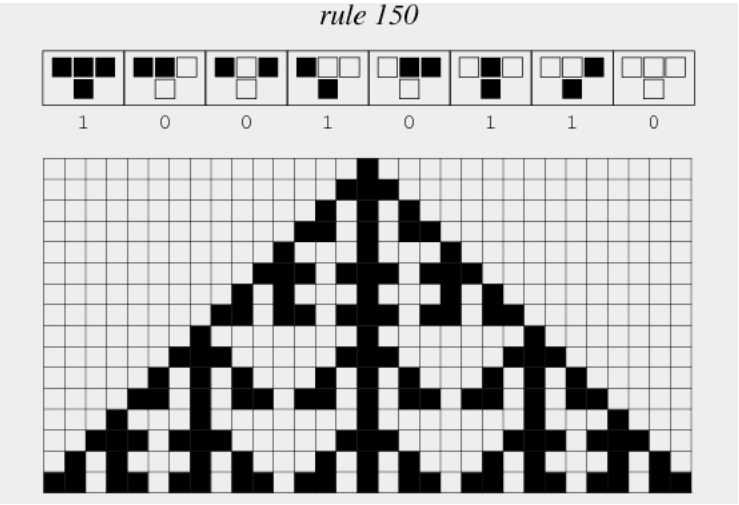


Figure 2.7: The elementary cellular automaton with rule $c_i(t+1) = c_{i-1}(t) \oplus c_i(t) \oplus c_{i+1}(t)$. The rule outcomes are encoded in the binary representation $150 = 10010110_2$. The illustration above shows 15 generations of a CA using this rule. Its initial configuration is a single black cell (representing a 1 in our analogy) (taken from [33]).

$$\begin{aligned} \texttt{rule}\ 90 \quad &: \quad c_i(t+1) = c_{i-1}(t) \oplus c_{i+1}(t) \\ \texttt{rule}\ 150 \quad &: \quad c_i(t+1) = c_{i-1}(t) \oplus c_i(t) \oplus c_{i+1}(t), \end{aligned}$$

where $\oplus$ denotes the XOR operation [10].

As an example of a 1D CA, consider rule 150. Suppose now that a CA $\mathcal{C}$ contains a set of cells $\{c_0, c_1, c_2, c_3, c_4, c_5, c_6\}$ and these cells are initialized at time step $t = 0$ with the values $\{0, 0, 0, 1, 0, 0, 0\}$ respectively. Then the value of $c_2(t = 1) = c_1 \oplus c_2 \oplus c_3 = 0 \oplus 0 \oplus 1 = 1$. Computing the value of each cell in the next time step $t = 1$ in a similar way, we get $\{0, 0, 1, 1, 1, 0, 0\}$ respectively for each cell from c_0 to c_6. Usually the evolution of a 1D CA is illustrated by drawing the initial state in the first row, and then the next generations on consecutive rows. Figure 2.7 shows the first 15 time steps of this example CA. Each black cell represents a 1 and each empty cell represents a 0.

In summary, a CA consists of a grid of cells, each containing one automaton. Each of these cells has a neighbourhood, defined in terms of its surrounding cells on the grid. The transition function of a CA describes the evolution of a cell in terms of the current states of the cells in its neighbourhood. There are many ways in which the traditional definition of a CA can be varied. Examples include

using a triangular or hexagonal [34] instead of a rectangular grid of cells and using probabilistic [5] instead of deterministic rules. A probabilistic rule is a rule that contains probabilities determining how the current state of a cell c_i would change from a certain time step t to time step $t+1$. In Section 2.2 we introduce a variation of CA where clustering of cells are allowed.

CA are traditionally implemented on sequential computers by copying the data structure used, to simulate the simultaneous update of each of the cells in the CA. Chapter 3 explores how the introduction of clustering in CA effects the implementation of CA. The next section discusses the different ways in which CA can be implemented.

2.1.1 Implementation

The CA concept has been researched and used in both hardware [1; 6; 13] and software models. CA simulations can be implemented using general purpose computers, specialized hardware [32], parallel computers [9] and even distributed systems [24; 30]. Even though specialized hardware can be finely tuned to maximize the efficiency of CA, it can be expensive and limits the grid size and number of states per cell of the CA [18].

The inherent parallelism of a CA simplifies its simulation on parallel computers. When implemented on parallel computers or distributed systems, an important consideration is the distribution of the cells of the CA amongst processors or different nodes. Communication is only needed between processors or nodes that are located in the neighbourhood of a given cell in the CA. This means that overhead for the passing of messages can be kept low.

When a CA is implemented on a sequential computer, the data structure used is copied and the new values entered into this copy at each time step. The current data structure is then switched with the copy, simulating the parallel update of the cells in the CA grid at a given time step. Sometimes, when implementing a CA, the finite automaton in each cell is augmented with a numerical value which has special meaning and changes according to the rules of the CA. For our example implementation, we use a sequential computer and each cell will contain a numer-

ical value that depicts a certain colour. This issue is discussed in more detail in Section 2.3.

CA have been used to model a variety of systems including complex natural systems like economic systems and insect colonies. These CA models have been used in various real world applications and the next section briefly mentions some of these applications.

2.1.2 Models and applications

CA models have been widely researched and implemented to simulate physical [20; 25], biological [3; 11; 16], chemical [12; 14; 17] and other types of systems [4; 7; 27–29]. Some examples where CA are used within these systems include traffic modelling (both road traffic and internet traffic), modelling of the distribution of disease, tumour growth, optimization problems, swarming behaviour, crowd behaviour and even music generation.

Some of the above-mentioned models have an inherent clustering behaviour and it is these types of models on which we want to focus. We already know that clustering has been successfully applied to the LEGO construction problem [31], which is essentially an optimization problem, and shown to be more efficient than other optimization methods not based on CA.

Another model that has a natural clustering behaviour, is insect-based clustering models. Here, ant-based clustering is the most widely used. These ant-based clustering and sorting algorithms have been applied to a range of different applications, for example intrusion detection, web usage mining, graph partitioning, bioinformatics, text mining, texture segmentation and packet clustering in router based networks [23].

We will use the ASM to investigate the implementation of clustering in CA. Ant sorting will be further discussed in Section 2.3. In the next section, a summary of the existing literature on CA with cell clustering is given.

2.2 Cellular automata with cell clustering

CA with cell clustering were originally proposed by Smal [31]. Usually a CA has a given number of cells, and this number stays constant throughout all time evolutions. A CA with cell clustering allows cells to merge or split during time steps, thus forming clusters of cells. This implies, if a cluster of cells is viewed as one cell, that the number of cells in the CA may vary in different time steps. Smal restricts his attention to 2D CA and defines a cluster as a set of adjacent cells (see Definition 2.2.2) with the adjacency of cells described in Definition 2.2.1.

Definition 2.2.1. *Let $\mathcal{C}$ be a 2D CA of size $N \times P$. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$. Two cells c_{ij} and c_{km} in $\mathcal{C}$ are adjacent if $|i-k| + |j-m| = 1$.*

Definition 2.2.2. *Let $\mathcal{C}$ be a 2D CA of size $N \times P$. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$. A cluster B in $\mathcal{C}$ is a set of cells from $\mathcal{C}$ such that any cell c_{ij} in B is adjacent to at least one other cell c_{km} in B.*

Smal also defines clusters to be disjoint if the underlying sets of cells which form the clusters are disjoint:

Definition 2.2.3. *Let $\mathcal{C}$ be a 2D CA of size $N \times P$. Let $0 \leq i \leq N-1$ and $0 \leq j \leq P-1$. Two clusters, A and B, in $\mathcal{C}$ are disjoint if there is no cell a_{ij} in A such that a_{ij} is also in B.*

Clusters are homogeneous, meaning that all the cells in the cluster contain the same status information. However, clusters can have different sizes. This affects the meaning of the neighbourhood for each cluster. Smal defines the Von Neumann neighbourhood of a cluster as a collection of clusters adjacent to a cluster A (the adjacency of clusters is described in Definition 2.2.4). From this description of the Von Neumann neighbourhood, it can be seen that a cluster may not necessarily have four neighbours, as in the traditional case (see Section 2.1, and Figure 2.2). The number of Von Neumann neighbours may vary between different clusters and a single cluster may have a different number of neighbours in different time steps.

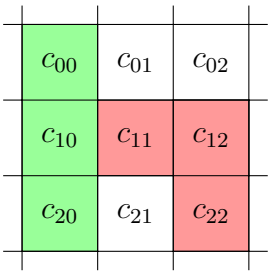


Figure 2.8: Cell clustering in a 2D CA.

Definition 2.2.4. *In a 2D CA of size $N \times P$ two clusters A and B are adjacent if there exists at least one cell a_{ij} in A and at least one cell b_{km} in B such that a_{ij} is adjacent to b_{km}, with $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$.*

Example 2.2.1. Consider the 2D CA in Figure 2.8. The two cells c_{00} and c_{10} are adjacent to each other while c_{00} and c_{11} are not adjacent. The set of cells $\{c_{00}, c_{10}\}$ can form a cluster, but the set $\{c_{00}, c_{11}\}$ can not. If we define the sets $\{c_{00}, c_{10}, c_{20}\}$ and $\{c_{11}, c_{12}, c_{22}\}$ to be two clusters, C_1 and C_2 respectively, then C_1 and C_2 are adjacent, since the cells c_{10} and c_{11} are adjacent. □

In Chapter 3 the definitions described in this section will be expanded. Some new definitions will also be added. The next section introduces the Lumer-Faieta model for ant sorting and describe an example implementation of this model, using traditional CA.

2.3 The ant sorting model

Swarm intelligence describes how a group of individuals indirectly collaborates to achieve complex dynamic behaviour. This collaboration happens without the presence of a central control point and is achieved through communication by modifying the environment. Social insects such as bees, wasps and ants are known to survive through this type of collective effort [21].

Several species of ants, for example, form piles of corpses, known as "cemeteries", or sort their larvae into piles according to their different life stages. Deneubourg *et al.* [15] proposed two models, one to account for the clustering and another for the

sorting behaviour of ants. The clustering model has been found to be more faithful in reproducing experimental observations, while the sorting model can be used in more diverse applications. The clustering model can be seen as a special case of the sorting model. In the sorting model, ants walk randomly on the grid, picking up objects that have a low concentration of similar objects in its neighbourhood and dropping these objects in locations with a higher concentration of similar objects.

Lumer and Faieta generalized the sorting model to apply to exploratory data analysis [26]. In our investigation we focus on this adapted sorting model by Lumer and Faieta. The Lumer and Faieta model (LF-model) is discussed in more detail in the following section.

2.3.1 The Lumer-Faieta model

In the LF-model each object has certain attributes that are used to describe it. The objects can be anything from simple one attribute objects to more complicated objects with k attributes. There are two important aspects to note in this model. The first is that a distance $d(o_i, o_j)$ between objects should be defined. The distance is used to measure the similarity of the objects on the grid through their attributes. The second is determining how this distance should be used to group similar objects in such a way that

- intra-cluster distances are minimized; that is, distances between objects with similar attributes should be small, and
- inter-cluster distances are maximized, which means that objects with different attributes should be separated [19].

Intuitively the model introduced by Lumer and Faieta is quite simple. Initially a number of objects are randomly placed on an empty grid, with each cell containing at most one object. Ants are then randomly placed on the grid and during every time step, ants are allowed to randomly move on the grid. Before each move, an ant will either pick up or drop an object, depending on the density of objects located in the neighbourhood. Algorithm 1 gives a high level description of the LF-model.

Algorithm 1: The Lumer-Faieta algorithm.

```
1  for every object o_i do
2      place o_i randomly on grid
3  for all ants do
4      place ants at randomly selected site
5  for t = 1 to t_max do
6      for all ants do
7          if (ant unladen) and (site occupied by object o_i) then
8              compute f(o_i) and p_pickup(o_i)
9              select random real number R between 0 and 1
10             if R ≤ p_pickup(o_i) then
11                 pick up object o_i
12         else if (ant carrying object o_i) and (site empty) then
13             compute f(o_i) and p_drop(o_i)
14             select random real number R between 0 and 1
15             if R ≤ p_drop(o_i) then
16                 drop object o_i
17         move to randomly selected neighboring site not occupied by another ant
```

Suppose an unladen ant (not carrying an object) is in a cell r that contains an object o_i. The ant picks up the object based on the local density function $f(o_i)$ of the object. Intuitively the local density is the number of objects with similar attributes to o_i in relation to objects with different attributes in the Moore neighbourhood of r. The local density function $f(o_i)$ is used to calculate the local density of a cell containing an object o_i. The function $f(o_i)$ is given below in Equation 2.3, together with a more detailed discussion.

When an unladen ant encounters a cell containing an object o_i, the probability to pick up this object, p_{pickup}, needs to be calculated. The probability of picking up o_i is defined by Equation 2.1. The probability is higher when the number of similar neighbours in the Moore neighbourhood of the cell is small and lower when more similar neighbours can be found:

$$p_{pickup} = \left(\frac{k_1}{k_1 + f(o_i)} \right)^2 . \tag{2.1}$$

In Equation 2.1, k_1 is a constant value and $f(o_i)$ denotes the local density function. Lumer and Faieta do not specify a value for k_1, but a constant value of 0.1 is used in the experiments by [8]. When $f(o_i) \ll k_1$, then p_{pickup} is close to 1 and when $f(o_i) \gg k_1$, p_{pickup} is close to 0.

When an ant carrying an object lands on a cell r that is empty, the probability to drop the object, p_{drop}, must be calculated. In the same way as the pick up probability, this probability is higher when more similar neighbours are found in the Moore neighbourhood of the cell and lower when fewer similar neighbours can be found. This is defined by Equation 2.2:

$$p_{drop} = \begin{cases} 2f(o_i) & \text{if } f(o_i) < k_2 \\ 1 & \text{otherwise} \end{cases} . \tag{2.2}$$

In Equation 2.2, k_2 is a constant value. Again, Lumer and Faieta does not specify a value for k_2, but a constant value of 0.15 is used in [8]. When $f(o_i) < k_2$, then p_{drop} is close to 0 and when $f(o_i) > k_2$, $p_{drop} = 1$.

An ant located at a cell r at position (a, b) can perceive a region of size s^2 (the Moore neighbourhood of r including r). If we assume that the ant is located at cell r at time step t and finds an object o_i or is currently carrying an object o_i, then the local density can be calculated using Equation 2.3:

$$f(o_i) = max \left\{ 0, \frac{1}{s^2} \sum_{o_j \in N_{(a,b)}} \left[1 - \frac{d(o_i, o_j)}{\alpha} \right] \right\} . \tag{2.3}$$

In this equation s^2 is the normalizing term and it introduces a density dependency in the density function $f(o_i)$. The sum includes all the objects in the neighbourhood of r and adds up all their similarity measures with object o_i. Each similarity measure is calculated by subtracting the dissimilarity measure, $d(o_i, o_j)$, between an object o_j and o_i, from 1, which is the value for maximum dissimilarity. The scaling factor α defines the scale for dissimilarity. It determines when two objects should be located next to each other. If α is too large, objects which do not belong in the same cluster would be placed together. On the other hand, if α is too small, objects with a relatively similar attribute space cannot be clustered together. Also if the value chosen for α is too small, $f(o_i)$ may become negative.

We can examine the behaviour of $f(o_i)$ by considering the extreme cases. If all s^2-1 cells around r are occupied by objects similar to o_i (that is, $\forall o_j \in \text{Neigh}_{(s\times s)-1}(r)$ with $d(o_i, o_j) = 0$), then $f(o_i) = 1$ and the pick up probability will be low. Assuming that α is chosen in such a way that dissimilarity between objects can be clearly measured, then when all $s^2 - 1$ cells around r contain objects that are maximally dissimilar to o_i (that is, $\forall o_j \in \text{Neigh}_{(s\times s)-1}(r)$ with $d(o_i, o_j) = d_{max}$), $f(o_i)$ would be small and p_{pickup} resultantly high. For example, if all $s^2 - 1$ cells around r are empty, then $f(o_i) = 0$ and $p_{pickup} = 1$.

In this section the LF-model was introduced and along with the Lumer-Faieta algorithm (LF-algorithm) explained in detail. This included the definition of the pick up probability, p_{pickup}, the drop probability p_{drop} and the local density function, $f(o_i)$. Next we give an example implementation of the LF-algorithm using CA.

2.3.2 Example implementation using CA

This section describes an example implementation of the LF-algorithm using CA. The LF-algorithm is based on the ability of ants to sort their larvae into piles. As the LF-Algorithm uses probabilistic rules its CA implementation can be classified as a probabilistic CA. In this example, ants randomly walk on a grid which contain a number of randomly placed coloured objects. If an ant is unladen, it can pick up a coloured object based on a calculated probability. Each ant is limited to carrying only one object at a time. If an ant is already carrying an object, it must first find a suitable cell to drop this object, also by using a calculated probability, before it may attempt to pick up a new object.

An implementation of the LF-model requires a definition for a measure of similarity for the coloured objects. For our implementation we use the five base colours: red, yellow, green, blue and magenta. These colours are mapped to the integer values 5, 15, 25, 35 and 45, respectively. The distance between two objects can then be defined to be $d(o_i, o_j) = |n_{c_i} - n_{c_j}|$, where n_{c_i} denotes the integer mapping of a colour i and n_{c_j} the integer mapping of a colour j. For example, if object o_1 has a colour with integer mapping 5, o_2 has a colour with integer mapping 35 and object

o_3 has a colour with integer mapping 15, the distance amongst these objects would be:

$d(o_1, o_2) = |o_1 - o_2| = 30,$
$d(o_1, o_3) = |o_1 - o_3| = 10,$ and
$d(o_2, o_3) = |o_2 - o_3| = 20.$

The LF-algorithm listed in Algorithm 1 (see page 14) was used for our implementation. During the initialization process, coloured objects and ants are randomly placed on the grid. Each object receives a random colour attribute. For each iteration, if an ant is not carrying an object, the pick up probability is calculated; otherwise the drop probability is calculated. Then, depending on which probability was calculated, an object is either picked up or dropped with this probability.

In this chapter CA as well as their implementation and application were discussed. CA with cell clustering was also introduced and will be discussed further in the following chapter. Finally the LF-method, which has a natural clustering behaviour, was introduced and its implementation using CA was discussed. In Chapter 4 we will extend this model to be used with CA with cell clustering and describe the implementation of the LF-method with cell clustering.

Chapter 3

Cellular Automata with Cell Clustering

In this chapter we define and describe CA with cell clustering. The definitions for clustering given in Section 2.2 are revised and new definitions added. This chapter also looks at some of the popular neighbourhoods and discusses how clustering affects the definition and layout of these neighbourhoods. Finally we consider the implementation of CA with cell clustering and compare it to traditional CA.

3.1 Clustering and adjacency of cells

In order to explore the properties of clusters and their neighbourhoods in more detail, we expand the definition of adjacency given in Section 2.2 to include different types of adjacency. The first type of adjacency, which is important for the definition of a cluster, is proper adjacency. In essence, we define two cells to be properly adjacent if they touch on a joint border.

Definition 3.1.1. *Let $\mathcal{C}$ be a 2D CA of size $N \times P$. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$. Two cells c_{ij} and c_{km} in $\mathcal{C}$ are **properly** adjacent if $|i-k| + |j-m| = 1$.*

Two cells c_i and c_j are also said to be properly connected if they are properly adjacent. Following from this, we define a set of properly connected cells:

Definition 3.1.2. *A set of cells A is properly connected if there exists a path $c_i \to ... \to c_j$ between any two cells c_i and c_j in A, so that $c_k \to c_{k+1}$ if and only if c_k and c_{k+1} is properly connected.*

A cluster can now be defined as a set of properly connected cells:

Definition 3.1.3. *Let $\mathcal{C}$ be a 2D CA. A cluster B in $\mathcal{C}$ is a set of cells from $\mathcal{C}$ such that the set of cells is properly connected.*

Disjointed clusters are now defined as:

Definition 3.1.4. *Let $\mathcal{C}$ be a 2D CA. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$. Then two clusters A and B are disjoint if there is no cell a_{ij} in A such that a_{ij} is also in B.*

In the definitions that follow we assume that all clusters are disjoint. Having defined a cluster as a set of properly connected cells, we also want to define adjacency for clusters. The definition for two properly adjacent clusters follow:

Definition 3.1.5. *Two clusters A and B are properly adjacent if there exists at least one cell a_{ij} in A and at least one cell b_{km} in B such that a_{ij} is properly adjacent to b_{km}.*

Example 3.1.1. Consider the 3x3 CA given in Figure 3.1. Here the cell c_{22} is properly adjacent to cell c_{12}, but c_{22} is not properly adjacent to cell c_{11}. It follows that the set of cells $\{c_{11}, c_{22}\}$ does not form a cluster, but the set $\{c_{12}, c_{22}\}$ forms a cluster Z. The clusters X and Y are properly adjacent, since the cells c_{10} and c_{20} are properly adjacent. These two clusters are also disjoint. □

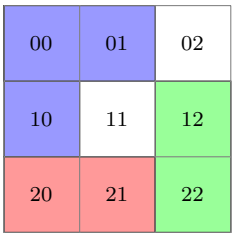


Figure 3.1: Cellular automaton $\mathcal{C}$ with cells $\{c_{00}, c_{01}, c_{02}, ..., c_{22}\}$, with clusters X, Y and Z where $X = \{c_{00}, c_{01}, c_{10}\}$, $Y = \{c_{20}, c_{21}\}$ and $Z = \{c_{12}, c_{22}\}$.

Different types of proper adjacency for cells can now be defined:

Definition 3.1.6. *Let $\mathcal{C}$ be a 2D CA. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$.*
*A cell c_{ij} is **right** adjacent to another cell c_{km} if $i - k = 0$ and $j - m = 1$.*
*A cell c_{ij} is **left** adjacent to another cell c_{km} if $i - k = 0$ and $j - m = -1$.*
*A cell c_{ij} is **top** adjacent to another cell c_{km} if $i - k = -1$ and $j - m = 0$.*
*A cell c_{ij} is **bottom** adjacent to another cell c_{km} if $i - k = 1$ and $j - m = 0$.*

Following this definition we also define the types of proper adjacency for clusters.

Definition 3.1.7. *Let $\mathcal{C}$ be a 2D CA. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$.*
*A cluster A is **right** adjacent to another cluster B if any cell $c_{ij} \in A$ is right adjacent to some cell $c_{km} \in B$.*
*A cluster A is **left** adjacent to another cluster B if any cell $c_{ij} \in A$ is left adjacent to some cell $c_{km} \in B$.*
*A cluster A is **top** adjacent to another cluster B if any cell $c_{ij} \in A$ is top adjacent to some cell $c_{km} \in B$.*
*A cluster A is **bottom** adjacent to another cluster B if any cell $c_{ij} \in A$ is bottom adjacent to some cell $c_{km} \in B$.*

Example 3.1.2. We continue with Figure 3.1 as an example. Here, the cell c_{11} is bottom (right, left, top) adjacent to c_{01} (c_{10}, c_{12}, c_{21}, respectively). Also cluster Y is left adjacent to cluster Z, cluster Z is right adjacent to cluster Y, cluster X is top adjacent to cluster Y and cluster Y is bottom adjacent to cluster X. □

In order to define neighbourhood configurations of clusters in the next section, we first need to define another type of adjacency called corner adjacency. The definitions follow below:

Definition 3.1.8. *Let $\mathcal{C}$ be a 2D CA. Let $0 \leq i, k \leq N-1$ and $0 \leq j, m \leq P-1$. Two cells c_{ij} and c_{km} in $\mathcal{C}$ are **corner** adjacent if both $|i-k| = 1$ and $|j-m| = 1$.*

Definition 3.1.9. *a) A cluster A is corner adjacent to another cluster B if any cell $c_{ij} \in A$ is corner adjacent to any cell $c_{km} \in B$.*
*b) A cluster A is **strictly corner** adjacent to another cluster B if any cell $c_{ij} \in A$ is corner adjacent to any cell $c_{km} \in B$ and A is not properly adjacent to B.*

Example 3.1.3. Consider Figure 3.1 again. Here, the cells c_{01} and c_{12} are corner adjacent, and therefore the clusters X and Z are corner adjacent. Clusters X and Z are also not properly adjacent and thus strictly corner adjacent. Clusters X and Y, however, are both properly adjacent and corner adjacent, which from the definition means that they are not strictly corner adjacent. □

In this section we defined the different types of adjacency for both cells and clusters. These definitions can now be used to define the neighbourhoods of CA. In the following section we discuss how clustering might affect the definition of two well known neighbourhoods of CA, namely the Von Neumann and Moore neighbourhoods.

The reader may note that all the definitions for adjacency and clustering above were given for the 2D case. The obvious question is whether these definitions can be given for other dimensions, and also, whether such definitions would have sensible practical interpretations. We briefly consider the issue.

In the 1D case, it is indeed possible to define clustering. However, the concept trivializes quickly, and we cannot immediately see many practical applications. Given a 1D CA with the cells $\{c_0, c_1, ..., c_n\}$, we get the following definitions for clustering:

- Two cells c_i and c_j are properly adjacent if $|i-j| = 1$.
- A cluster is a set of properly connected cells.
- Two clusters A and B are disjoint if no cell a_i in A is also in B.
- Two clusters A and B are properly adjacent if there exists a cell a_i in A and a cell b_j in B such that a_i is properly adjacent to b_j.

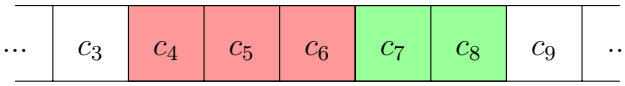


Figure 3.2: 1D CA with cells $\{c_0, c_1, ..., c_n\}$, with clusters X and Y where $X = \{c_4, c_5, c_6\}$ and $Y = \{c_7, c_8\}$.

- A cell c_i is right adjacent to another cell c_j if $i - j = 1$ and left adjacent if $i - j = -1$. From this follows the right and left adjacency of clusters.

In context of 1D CA it does not make sense to define top, bottom and corner adjacency of cells.

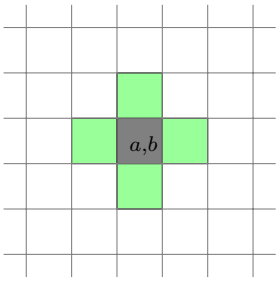
Example 3.1.4. In Figure 3.2 the cells c_7 and c_8 are properly adjacent to each other and form the cluster Y. The clusters X and Y are properly adjacent to each other and also disjointed. The cell c_6 is left adjacent to c_7 and c_7 is right adjacent to c_6. From this follows that the cluster X is left adjacent to Y and Y right adjacent to X. □

The 3D case for clustering is more interesting, and generalizes neatly from the 2D case. Note that there are three types of adjacencies for the 3D case. We believe that 3D applications which display an inherent clustering behaviour would benefit from a clustering implementation. One of the main issues would be whether an efficient implementation is still possible.

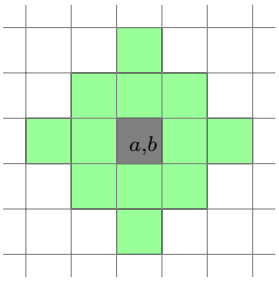
As the main focus of this thesis is practical applications and implementation issues of clustering, we do not attempt a generalization of clustering to n dimensions here. However, we do not foresee any mathematical difficulties in such a definition.

3.2 Neighbourhoods

In the case of CA without cell clustering the different types of neighbourhoods (such as Von Neumann or Moore) can be defined by the type of adjacency of the neighbouring cells. See, for example, Figures 3.3 and 3.4. In the case of CA with cell clustering, we follow similar principles, and define neighbourhoods in terms of *cluster* adjacency.

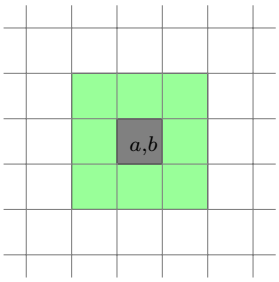


(a) Von Neumann neighbourhood for range $r = 1$.

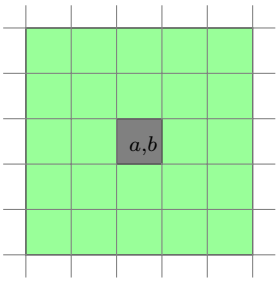


(b) Von Neumann neighbourhood for range $r = 2$.

Figure 3.3: The classic Von Neumann neighbourhood of 2D cellular automata for ranges $r = 1$ and $r = 2$.



(a) Moore neighbourhood for range $r = 1$.



(b) Moore neighbourhood for range $r = 2$.

Figure 3.4: The classic Moore neighbourhood of 2D cellular automata for ranges $r = 1$ and $r = 2$.

3.2.1 The Von Neumann neighbourhood of a cluster

We recall the definition of a Von Neumann neighbourhood in CA:

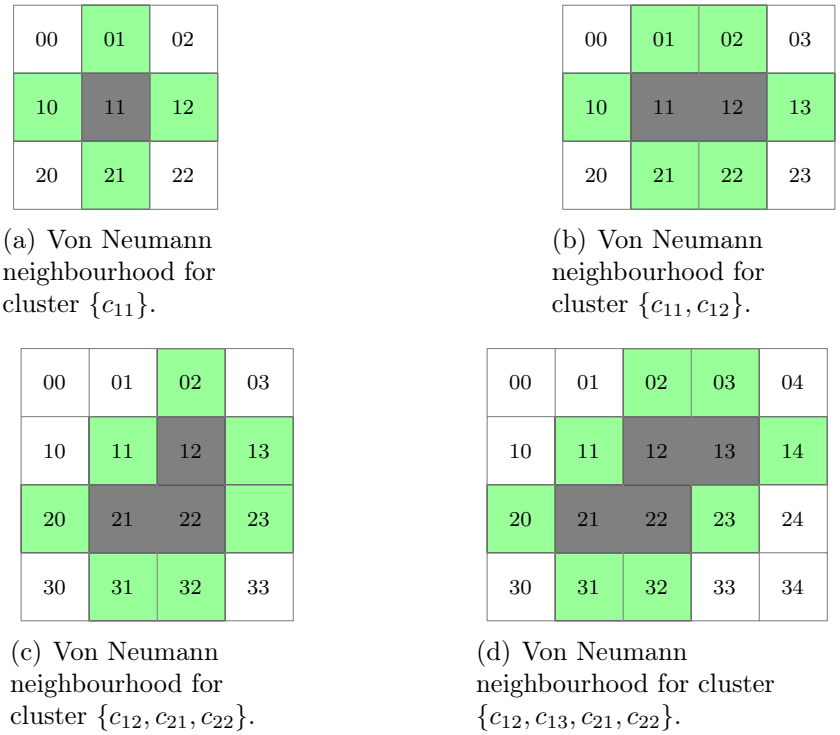
$$N^r_{(a,b)} = \{(i,j) \ : \ |i-a| + |j-b| \leq r\}\,.$$

Similarly, we want to define the Von Neumann neighbourhood of a cluster A for a distance r. Intuitively, for distance $r = 1$, we consider every cell c on the border of A, and if there is a cluster with a cell c' that is in the Von Neumann neighbourhood of c, then the cluster containing c' is in the Von Neumann neighbourhood of A. For $r > 1$, the idea generalises accordingly. Hence, the Von Neumann neighbourhood of a cluster A is defined as all the clusters B such that B has at least one cell c_{ij}

with the property $|i-a|+|j-b| \leq r$ and $c_{ab} \in A$ is properly adjacent to at least one cell not in A:

$$N_A^r = \{\ B :\ \exists\, c_{ij} \in B \text{ such that } |i-a|+|j-b| \leq r \text{ and } \exists\, c_{km}\ \notin A \text{ and } c_{ab}\ \in A \text{ such that } |a-k|=1 \text{ and } |b-m|=0 \text{ or } |a-k|=0 \text{ and } |b-m|=1\}. \tag{3.1}$$

See Figure 3.5 for an illustration of the Von Neumann neighbourhood of different clusters with range $r = 1$. Figure 3.5(a) shows the Von Neumann neighbourhood for a single cell, in comparison to Figures 3.5(b), 3.5(c) and 3.5(d), which show the Von Neumann neighbourhood for different clusters.



(a) Von Neumann neighbourhood for cluster $\{c_{11}\}$.

(b) Von Neumann neighbourhood for cluster $\{c_{11}, c_{12}\}$.

(c) Von Neumann neighbourhood for cluster $\{c_{12}, c_{21}, c_{22}\}$.

(d) Von Neumann neighbourhood for cluster $\{c_{12}, c_{13}, c_{21}, c_{22}\}$.

Figure 3.5: Examples of the Von Neumann neighbourhood for different clusters.

Note that Equation 3.1 implies that a cluster does not necessarily have four neighbours in its Von Neumann neighbourhood. Also, the number of Von Neumann neighbours may vary between different clusters. In addition, a single cluster may have a different number of neighbours in different time steps. For example, consider

again Figure 3.5(b) above. The maximum number of clusters that can be in the Von Neumann neighbourhood of the cluster $X = \{c_{11}, c_{12}\}$ is six, assuming that the cells $c_{01},\ c_{02},\ c_{13},\ c_{22},\ c_{21}$ and c_{10} are all in different clusters. If, however, some of these mentioned cells are in the same cluster, the total number of clusters in the Von Neumann neighbourhood of X is less than six. For example, if $\{c_{01}, c_{02}\}$ forms one cluster and $\{c_{21}, c_{22}\}$ forms another cluster, then the number of clusters in the Von Neumann neighbourhood for X are four. Note that in this particular case, the number of neighbours in the Von Neumann neighbourhood of $\{c_{11}, c_{12}\}$ cannot be less than four. It is possible that at any time step t the cells $c_{00},\ c_{01},\ c_{02}$ and c_{10} could merge to form one cluster and subsequently the cluster $\{c_{11}, c_{12}\}$ would have less than four neighbours. It is also possible that a cluster A could be surrounded by another cluster B, which means that A would have only one neighbour in its neighbourhood.

The same concept holds for the clusters in Figures 3.5(c) and 3.5(d), where the maximum number of clusters are seven and eight respectively. More neighbourhood properties are further explored in Section 3.2.3.

3.2.2 The Moore neighbourhood of a cluster

We recall the definition of a Moore neighbourhood in CA:

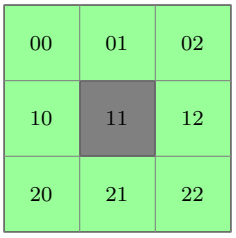
$$N^r_{(a,b)} = \{(i,j)\ :\ |i-a| \leq r,\ |j-b| \leq r\}\,.$$

Similarly, we want to define the Moore neighbourhood of a cluster A for a distance r. Intuitively, for distance $r = 1$, we consider every cell c on the border of A, and if there is a cluster with a cell c' that is in the Moore neighbourhood of c, then the cluster containing c' is in the Moore neighbourhood of A. For $r > 1$, the idea generalises accordingly. Hence, the Moore neighbourhood of a cluster A is defined as all the clusters B such that B has at least one cell c_{ij} with the properties $|i-a| \leq r$ and $|j-b| \leq r$ where $c_{ab} \in A$ is properly adjacent to at least one cell

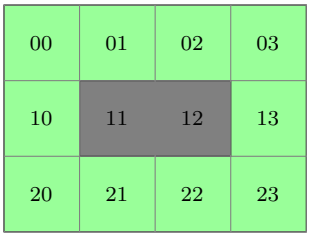
not in A:

$$N_A^r = \{\ B :\ \exists\ c_{ij} \in B \text{ such that } |i-a| \leq r \text{ and } |j-b| \leq r,\ \text{and } \exists\ c_{km}\ \notin A \text{ and } c_{ab}\ \in A \text{ such that } |a-k| = 1 \text{ and } |b-m| = 0 \text{ or } |a-k| = 0 \text{ and } |b-m| = 1\}. \tag{3.2}$$

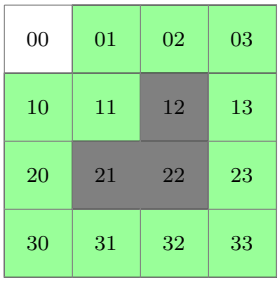
See Figure 3.6 for an illustration of the Moore neighbourhood of different clusters with range $r = 1$. Figure 3.6(a) shows the Moore neighbourhood for a single cell, in comparison to Figures 3.6(b), 3.6(c) and 3.6(d), which show the Moore neighbourhood for different clusters.



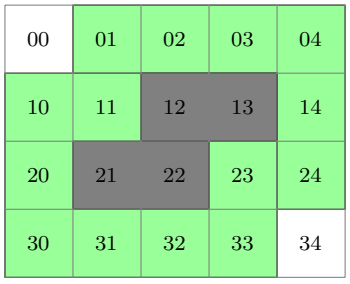
(a) Moore neighbourhood for cluster $\{c_{11}\}$.



(b) Moore neighbourhood for cluster $\{c_{11},c_{12}\}$.



(c) Moore neighbourhood for cluster $\{c_{12},c_{21},c_{22}\}$.



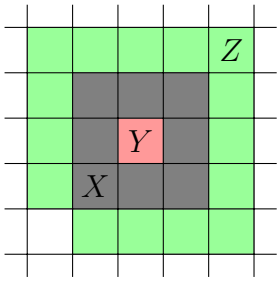
(d) Moore neighbourhood for cluster $\{c_{12},c_{13},c_{21},c_{22}\}$.

Figure 3.6: Examples of the Moore neighbourhood for different clusters.

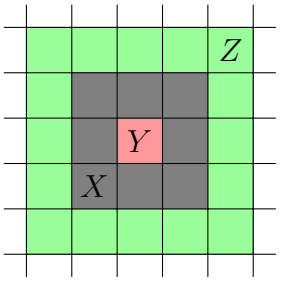
As in the previous examples for the Von Neumann neighbourhood, the case of the number of clusters in the Moore neighbourhood are generally the same. In the case of the Moore neighbourhood, however, the maximum number of clusters in the Moore neighbourhood for Figures 3.6(b), 3.6(c) and 3.6(d) are 10, 12 and 14 respectively. It is interesting to note in the Moore neighbourhood, that the

minimum number of neighbours is often one, in the case where all cells adjacent to the cluster in question form a single cluster. For example, in Figure 3.6(a), if $\{c_{00}, ..., c_{22}\}$ is a single cluster, then c_{11} has only one neighbour.

Clearly, the minimum and maximum number of neighbours in both the Moore and Von Neumann neighbourhoods of a cluster X is directly related to the geometrical form of X. For example, a doughnut-shaped cluster must have a minimum of two neighbours (see Figure 3.7). Note that in the Von Neumann example the cell in the bottom left corner is not in the cluster Z. This is the case depicting the minimum number of cells in the Von Neumann neighbourhood that would make up a neighbourhood size of two. We leave a detailed analysis of the issue of geometrical form for future work.



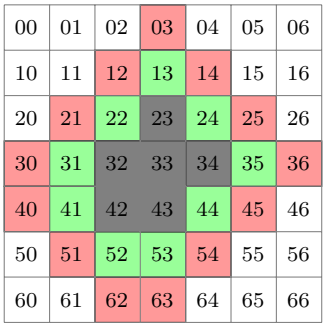
(a) Doughnut shaped cluster X with the two clusters Y and Z making up its Von Neumann neighbourhood.



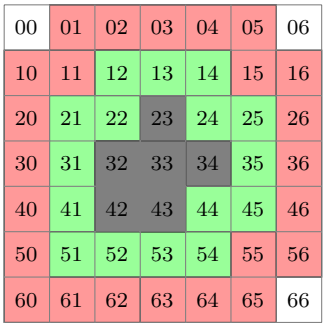
(b) Doughnut shaped cluster X with the two clusters Y and Z making up its Moore neighbourhood.

Figure 3.7: Example neighbourhoods for a doughnut shaped cluster.

So far all the neighbourhood examples for clusters were for neighbourhoods with range $r = 1$. We now consider neighbourhoods with a range of $r > 1$. Note that, for traditional CA, r determines the range of the neighbourhood for a single cell. However, for CA with cell clustering a cluster could contain more than one cell and the point from which the range r should be measured is not as straightforward. In the definitions for the Von Neumann and Moore neighbourhoods (see Equations 3.1 and 3.2) we specify that the range is measured from the border cells of a cluster (that is, all the cells c_{ab} in the cluster A that is properly adjacent to at least one cell not in A).



(a) A range $r = 2$ Von Neumann neighbourhood example for cluster $X = \{c_{23}, c_{32}, c_{33}, c_{34}, c_{42}, c_{43}\}$.



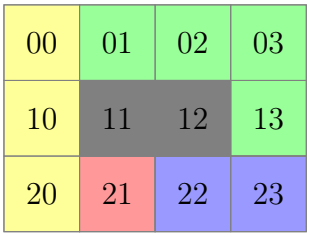
(b) A range $r = 2$ Moore neighbourhood example for cluster $X = \{c_{23}, c_{32}, c_{33}, c_{34}, c_{42}, c_{43}\}$.

Figure 3.8: Neighbourhood examples with $r = 2$.

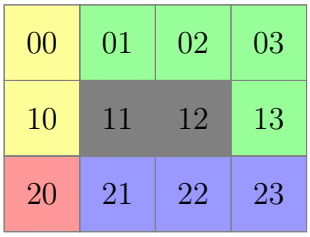
Example 3.2.1. As an example, we consider the Von Neumann neighbourhood with range $r = 2$ given in Figure 3.8(a). The cells $\{c_{23}, c_{32}, c_{34}, c_{42}, c_{43}\}$ are all border cells of cluster X, as each one is properly adjacent to at least one cell not in X. The cell c_{33}, however, is not a border cell as it is properly adjacent to the cells $\{c_{23}, c_{32}, c_{34}, c_{43}\}$, which is also in X. The cluster $\{c_{03}\}$ is in the Von Neumann neighbourhood of X as there exists a border cell c_{23} in X so that the condition given in Equation 3.1 is met. Note, however, that the cell c_{33} in X can never satisfy the condition for any cluster in the Von Neumann neighbourhood. This also applies to any range $r > 2$ and for the Moore neighbourhood. An example Moore neighbourhood with range $r = 2$ is given in Figure 3.8(b). □

3.2.3 Neighbourhood properties

Consider Figure 3.9(a). Then, according to Definition 3.1.5, clusters A, B, C and D are all properly adjacent to cluster E and therefore form the Von Neumann neighbourhood (Equation 3.1) of cluster E. However, it is interesting to note that from Equation 3.2 follows that clusters A, B, C and D also form the Moore neighbourhood of cluster E. Hence, in CA with cell clustering, the situation may occur that the Von Neumann neighbourhood and Moore neighbourhood of a cluster is the same set of clusters.



(a) Cluster E surrounded by the clusters A=$\{c_{00},c_{10},c_{20}\}$, B=$\{c_{01},c_{02},c_{03},c_{13}\}$, C=$\{c_{22},c_{23}\}$ and D=$\{c_{21}\}$.



(b) Cluster E surrounded by the clusters A=$\{c_{00},c_{10}\}$, B=$\{c_{01},c_{02},c_{03},c_{13}\}$, C=$\{c_{21},c_{22},\ c_{23}\}$ and D=$\{c_{20}\}$.

Figure 3.9: Neighbourhood examples for the cluster $E = \{c_{11}, c_{12}\}$.

On the other hand, Figure 3.9(b) shows an example where the Von Neumann neighbourhood and the Moore neighbourhood differ. All the clusters surrounding cluster E is in the Moore neighbourhood, but cluster D is not in the Von Neumann neighbourhood (as it is not properly adjacent to E).

With the neighbourhood configurations of CA with cell clustering varying over time, the question arises how this might change the definition of transition rules for CA. This subject is discussed in the next section.

3.3 Transition function rules

In Chapter 2 we mentioned that the transition function is defined in terms of the neighbourhood of the CA. In the case of CA with cell clustering, the neighbourhood may vary for different clusters and for different time steps. Hence, it could potentially be difficult to define a transition function for CA with cell clustering.

For example for traditional CA, where a grid has null boundary conditions the neighbourhoods for the cells on the boundaries are incomplete. The standard solution to this problem is to ignore the missing neighbourhood cells (for example, setting the values of the missing cells to zero). For clustering of course we can get complex neighbourhoods, but that is also the case with traditional CA if such a complex neighbourhood is defined.

The main difference between a neighbourhood for traditional CA and that of CA with cell clustering is that the neighbourhood for CA with cell clustering can change with each time step. Though clusters are defined in this chapter to be any size and shape, this can be limited depending on the implementation for example as was done in [31].

To summarize, there is no fixed recipe for how a transition function should look. As with traditional CA, the transition function depends on the specific model. The next section looks at the implementation of CA with cell clustering.

3.4 Implementation

In this section we briefly note some general issues concerning the implementation of CA with cell clustering. These issues will be applicable in most applications of CA with clustering.

With traditional CA, a single data structure is used to store the information for the grid of cells. During a transition from one time step to another this structure is usually copied to simulate the parallel transition from one time step to another. For clustering an extra data structure is needed to keep track of the clusters on the grid.

Parallelisation for the clustering implementation is not as straightforward as it is for traditional CA. Firstly, the extra data structure is accessed and updated by all the cells in the CA. Hence, if different cells execute on different nodes, the information about the clusters would need to be globally available and will potentially need to be sent to many different cells during each time step. Clearly this will be more inefficient than in the case of traditional CA without clustering. Another issue is the allocation of processors to a cluster. Depending on the implementation and the specific definition of clusters, issues that would need to be considered include the division of clusters between processors and how cluster formation should be handled. A detailed analysis of these issues is to be considered in future work.

The next chapter looks at the implementation of the LF-algorithm using CA with cell clustering. In it we discuss the modification of the LF-model and some of the

obstacles encountered during the implementation.

Chapter 4

Implementing the LF-algorithm using Cell Clustering

This chapter describes an example implementation of CA with cell clustering. The LF-model described in Chapter 2 is modified and applied to the example implementation where coloured objects are sorted into clusters, as described in Section 2.3.2. The goal of this chapter is to explore the implementation possibilities of CA with cell clustering within a specific application and thereby gain insight into CA with cell clustering in general. Some of the obstacles of the implementation are discussed along with possible solutions.

4.1 Modifying the LF-model for clustering

The modifications to the LF-model are kept as minimal as possible. The modification of this model is not explicitly to improve the model, but to add the clustering concept to it. The LF-model was specifically chosen for our purposes, because it exhibits an inherent clustering behaviour. In this section we describe the minimal modifications necessary to allow for explicit clustering.

In this modified LF-model similar objects are grouped together to form clusters on the grid. A cluster is defined to be a group of similar objects, where the set of

objects is properly connected. The similarity of objects are defined by the distance $d(o_i, o_j)$ as described in Section 2.3.2, page 16.

Instead of allowing ants to only pick up single objects on the grid, ants can now pick up clusters of objects. Each ant can carry at most one cluster, regardless of the size or shape of the cluster. The clusters are dropped near clusters with similar attributes, potentially forming a new larger cluster.

The modified algorithm (Algorithm 2) is given below. The reader may compare it with the original LF-algorithm (Algorithm 1 on page 14). Note that lines 11 and 16 in Algorithm 1 is now replaced with *pick up* ***cluster*** c_j and *drop* ***cluster*** c_j respectively (lines 12 and 17 in Algorithm 2). In addition, for initialization purposes, the clusters on the grid are calculated after objects have been randomly placed on the grid (see line 5). Note that an object is allocated to a cluster if it has attributes similar to the attributes of this cluster and it is in the Von Neumann neighbourhood of at least one of the objects in the cluster. Clusters are homogeneous and the attributes of a cluster is the same as the attributes of the objects in it.

The ants perform a random walk on the grid to find a cluster of similar objects. If an ant is unladen (carries no cluster), it picks up a cluster with the calculated probability; else, it drops the cluster with the calculated drop probability.

It is important to notice that the calculations for pick up and drop probabilities used in Equations 2.1 and 2.2 on page 14 do not take any clustering information into consideration. Our initial implementation uses these equations directly to calculate the pick up and drop probabilities, based on cells and not clusters. Possible variations of the equations include variations which are based on clusters and not only single cells. For example, the size of a cluster is used to refine the pick up or drop probabilities. These variations are to be considered for future work.

The next section describes the implementation of the modified LF-algorithm using CA with cell clustering. This includes data structures used and challenges encountered during the implementation process.

Algorithm 2: The modified Lumer-Faieta algorithm for clustering.

```
1  for every object o_i do
2      place o_i randomly on grid
3  for all ants do
4      place ants at randomly selected site
5  calculate clusters
6  for t = 1 to t_max do
7      for all ants do
8          if (ant unladen) and (site occupied by object o_i in cluster c_j) then
9              compute f(o_i) and p_pickup(o_i)
10             select random real number R between 0 and 1
11             if R ≤ p_pickup(o_j) then
12                 pick up cluster c_j
13         else if (ant carrying cluster c_j) and (site empty) then
14             compute f(o_i) and p_drop(o_j)
15             select random real number R between 0 and 1
16             if R ≤ p_drop(o_j) then
17                 drop cluster c_j
18         move to randomly selected neighbouring site not occupied by another ant
```

4.2 Implementation

In the modified LF-algorithm, objects are randomly placed on an empty grid. These objects are grouped into clusters according to their location and similarity. Ants are then allowed to randomly move on the grid, picking up or dropping clusters depending on the density of objects located in the neighbourhood. If a cluster is dropped in the neighbourhood of similar clusters then these clusters are merged to form a new, larger cluster.

Each cluster is assigned a unique identifying value and stored using a map data type [22]. This is to allow for easy insertion and retrieval of the desired clusters. Each cell on the grid stores a value that indicates to which cluster it currently belongs, if any. These values on the grid ensure that an ant can easily pick up an entire cluster by using the reference value stored in the cell to access the map containing the cluster (see Figure 4.1 for an illustration of this). Dropping a cluster

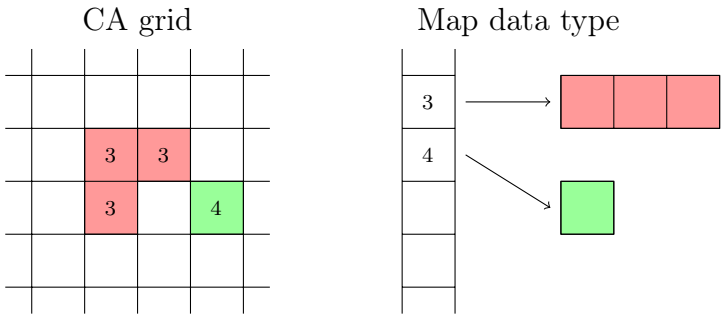


Figure 4.1: Example of storing clusters in a map data type. Note that if an ant lands on a cell in the grid, it can access the entire cluster the cell belongs to through the reference value stored in the CA grid.

is, however, not so straightforward and will subsequently be discussed in more detail.

4.2.1 Dropping clusters

When an ant has to drop a cluster, there are a number of different questions that may arise. Firstly, is the shape of the cluster important and should it be preserved? If so, then a suitable space would have to be found into which this cluster can fit. This can become quite complicated, given that in our implementation a cluster can be any size or shape and there is no way to predict this shape at any given time. A more detailed investigation into the problem of determining if there is a sensible way to take shapes of clusters into account when dropping clusters, or taking sizes of clusters into account when calculating pickup probabilities, would be interesting. For our implementation, however, we generally ignored the shape of the cluster when dropping it, and just preserve the size of the cluster to be dropped. We did implement one simple layout drop method as an example, namely the diamond drop method (see Section 4.2.1.3). Note that a cluster is dropped cell by cell, for each of the cells in the cluster.

The only constraint we put on the dropping of a cluster is that each object in the cluster should be dropped in a cell that is in the neighbourhood of similar objects. When adding this constraint, there are still a number of ways in which a cluster can be dropped. We describe three possible methods below.

4.2.1.1 The Von Neumann drop method

The first two drop methods, namely the Von Neumann and Moore drop methods, use the same basic algorithm, which is shown in Algorithm 3. For each object in the cluster, a suitable position should be found before it can be dropped. If there is no suitable position, the ant simply moves on with the remainder of the cluster to find space elsewhere on the grid[1].

Algorithm 3: Basic drop method.

```
1 p ← find suitable neighbour position
2 while p ≠ null and c_i not empty do
3 |   drop object at p
4 |   p ← find suitable neighbour position
```

As implied by its name the Von Neumann method finds a suitable position by considering the empty cells in the Von Neumann neighbourhood of the current position p. In Algorithm 4 the empty cell with the highest object density is chosen as the next suitable position to drop an object from the cluster carried by an ant.

Algorithm 4: Finding next suitable position for Von Neumann drop algorithm.

```
1  /* Find neighbour in Von Neumann neighbourhood of p with the highest object
   density f(o_i), where o_i is the next object to be dropped.*/
2  max ← 0 //The highest object density
3  maxP ← null //The position of the neighbour with the highest object density
4  for each cell in Von Neumann neighbourhood of p do
5  |   if cell is empty then
6  |   |   s ← f(o_i)
7  |   |   if max < s then
8  |   |   |   max ← s
9  |   |   |   maxP ← position of cell
10 return maxP
```

[1] In effect, this means that existing clusters can be split into multiple smaller clusters. Splitting was also explicitly used by Smal [31], in the cases where his layout algorithm failed to find a suitable object placement.

4.2.1.2 The Moore drop method

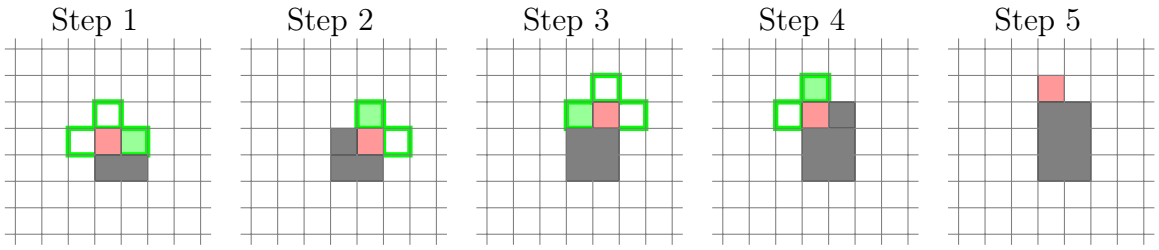
The Moore drop method is similar to the Von Neumann drop method, with the only difference being that the Moore neighbourhood of the current position p is considered. The empty cell with the highest object density is chosen as the next suitable position to drop an object from the cluster.

As an aside, we note that both the Von Neumann and the Moore drop methods tend to form rectangularly shaped clusters (see Figure 4.4). This situation occurs as a result of the way each drop position for an object is determined. When calculating the position of the new drop site, the neighbour with the highest object density is chosen. If there are two or more such neighbours, then the first of these neighbours is selected (note that the neighbours are traversed in the order North, East, South and West). This order of selection plays a role in the formation of the rectangularly shaped clusters. Figure 4.2(a) shows a step by step example of the process of dropping a cluster of size five for the Von Neumann drop method by traversing the Von Neumann neighbourhood in the order given above. As an example of how the order changes the shape of a dropped cluster, we give another example in Figure 4.2(b). Here the neighbours are traversed in the order West, South, East, North. The principle is generally similar for the Moore drop method, except that there is now a maximum of eight possible positions for each object to be dropped at. Also, the neighbours are traversed in the order North, Northeast, East, Southeast, South, Southwest and West.

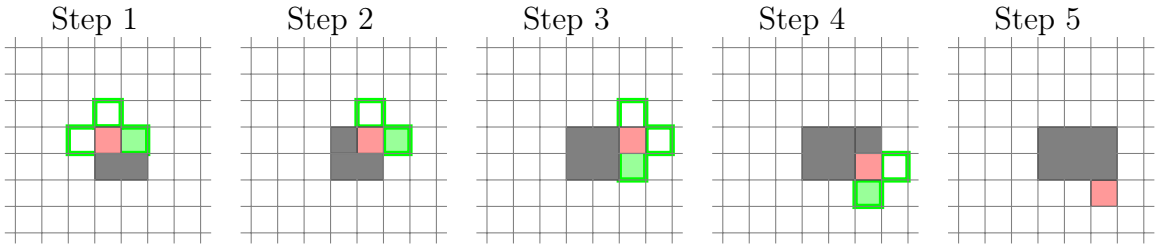
4.2.1.3 The diamond drop method

In contrast to the Von Neumann and Moore drop methods, where the goal is to find a drop position with the highest object density, the diamond method drops a cluster in a diamond-like shape. Algorithm 5 describes the diamond drop method.

Consider an ant at position p that has to drop a cluster. For every position p where an object o_j in the cluster c_i is dropped, the cells in the Von Neumann neighbourhood of p are added at the end of a queue. These cells then become potential drop sites for the objects in c_i. For every next object to be dropped, a cell is removed from the front of the queue. If this cell is empty, then o_j is dropped



(a) If two or more neighbours have the maximum object density, then the first suitable neighbour is chosen.



(b) If two or more neighbours have the maximum object density, then the last suitable neighbour is chosen.

Figure 4.2: Two step by step examples of dropping a cluster of size five using the Von Neumann drop method. The gray shaded cells are the objects in the current cluster, the red shaded cell is the most recently dropped object and the green shaded cell is the next position where an object will be dropped. The green bordered cells show the empty cells in the Von Neumann neighbourhood.

at the position p_f of the cell. p_f then becomes p and the process is repeated until either all objects in c_i is dropped or the queue is empty (that is, the whole cluster has been dropped or no suitable drop position is found). In the latter case, the ant continues with the remaining part of the cluster to find another drop site. Figure 4.3 shows an example of the diamond drop method.

An example output for these three different drop methods is shown in Figure 4.4 for a grid size of 100x100.

4.2.2 Empty grid deadlock

Lumer and Faieta specify in their model that the dimensions of the grid must exceed the number of objects on the grid by roughly an order of magnitude. In addition, the number of objects must also exceed the number of ants by at least an order of

Algorithm 5: Diamond drop method.

```
1  queue ← new Queue()
2  p ← position of ant
3  while p ≠ null and c_i not empty do
4      aP ← positions of all objects in Von Neumann neighbourhood of o_j
       queue.addAll(aP)
5      drop object o_j at p
6      /*find next empty position*/
7      while queue not empty do
8          p ← queue.dequeue() //remove position at head of queue
9          if cell at p is empty then
10             aP ← positions of all objects in Von Neumann neighbourhood of p
11             queue.addAll(aP)
12             break
```

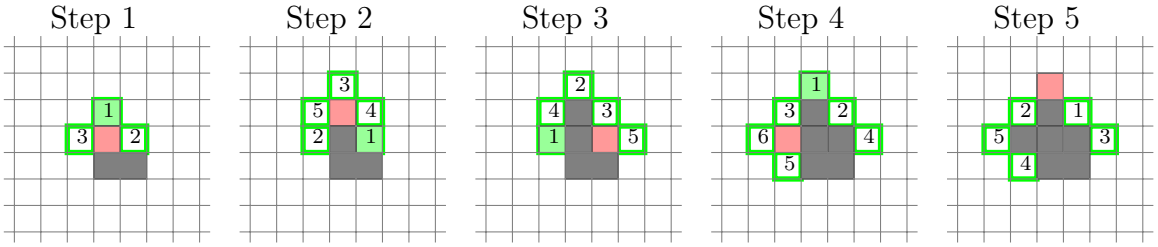
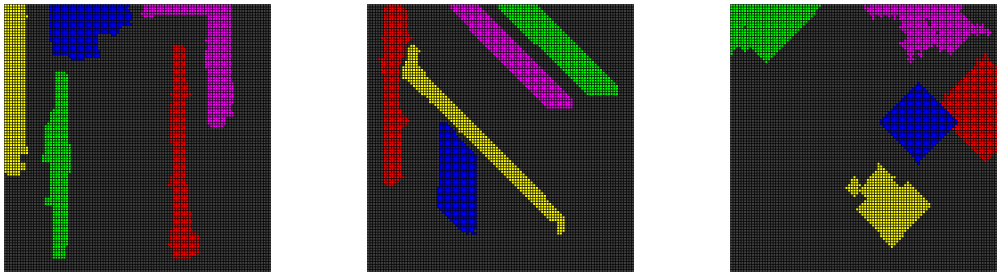


Figure 4.3: A step by step example of dropping a cluster of size five using the diamond drop method. The gray shaded cells are the objects in the current cluster, the red shaded cell is the most recently dropped object and the green shaded cell is the next position where an object will be dropped. Also, the numbers indicate the current position a cell is in the queue. The green bordered cells show the cells currently in the queue.

magnitude [26].

In CA with cell clustering, of course, the number of objects on the grid changes constantly. Hence, it is not possible to comply to the second condition above when implementing clusters. As clusters form, the situation invariably arises where the number of ants exceed the number of objects. Eventually, the grid is left empty of clusters, which causes the drop probability to become zero, and the simulation ends up in a deadlock situation.

We solved the deadlock situation by prohibiting an ant to pick up a cluster if it is



(a) The Von Neumann drop method.

(b) The Moore drop method.

(c) The diamond drop method.

Figure 4.4: Output for three different drop methods.

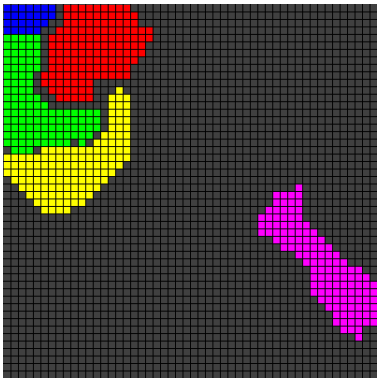
the only cluster on the grid. Note that the extra data structure storing the clusters on the grid allows for easily checking this global condition. If there is more than one data type (in this context, more than one colour on the grid), then least one cluster of each type should be left on the grid.

Another possible solution would be to keep track of the ratio of the number of ants and the clusters on the grid. This ratio could then be used in the calculation of the pick up probabilities. Also the size of a cluster could be taken into account making it far more likely that an ant would pick up small clusters and leave larger clusters on the grid.

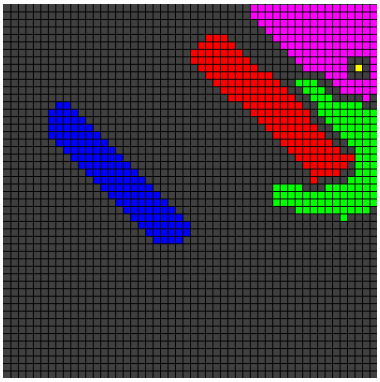
4.2.3 Surrounded cluster deadlock

In our simulation we have come across the situation where a cluster c_i with colour attribute x is surrounded by one or more clusters with attributes that differs from x. See Figure 4.5 for a few examples of this occurrence. If c_i happens to be the only cluster with the colour x then the simulation ends up in a deadlock situation. Every ant carrying a cluster with the colour x can now never find a suitable drop site. In turn, this prevents the simulation from sorting all the objects into their respective clusters.

A possible solution to this problem is to add a counter that keeps track of the number of time steps that no clusters are dropped. If a certain threshold value is reached, then all the ants drop the clusters they are carrying at their current



(a) In this picture the green cluster is surrounded by the blue, red and yellow clusters. Also, the blue cluster is surrounded by the green and red clusters.



(b) In this picture the yellow cluster is surrounded by the magenta cluster.

Figure 4.5: Two examples of the occurrence of where a cluster is surrounded by another. A grid size of 50x50 is used.

position or a nearby available position. A position is defined to be available if it does not contain any object. Thus an ant would move around on the grid searching for these available positions to drop an object. We used this solution in our implementation with a threshold value of $x = w * h$, where w is the number of rows and h the number of columns of the grid.

The next chapter describes the experiments conducted to evaluate the efficiency of CA with clustering, as compared to CA without clustering. These experiments include an investigation into the empty grid deadlock situation described in Section 4.2.2.

Chapter 5

Experiments and Evaluation

In this chapter we evaluate the effects of using CA with clustering in the implementation of a given model, as opposed to an implementation without clustering. To this end, we chose a representative model (the ant sorting model) which displays an inherent clustering behaviour [1].

We consider two aspects in our evaluation: (1) the efficiency of the implementation, in terms of both memory usage and runtime, and (2) the success of the final goal of the CA model – in this case, how well do the ants sort with the clustering implementation as compared to sorting without the clustering implementation.

In the next section we first consider the issue of deadlock, which was discussed in Section 4.2.2 on page 38.

5.1 Empty grid deadlock

As clusters merge, the situation arises where all the clusters on the grid are picked up and subsequently the grid is left empty. In this section we investigate this deadlock situation to see under which conditions it occurs and also how often it

[1] For future work, it could perhaps be interesting to choose a model without a natural clustering behaviour, in order to show that implementations based on clustering do not perform well if no inherent clustering is present in the model. We seriously doubt that any advantage will be gained in models not displaying inherent clustering.

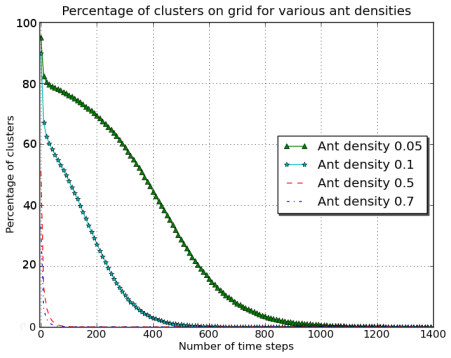
occurs. To determine if there are any parameters for which the deadlock would not occur, experiments were ran for varying ant and object densities.

The following experiments were set up using a grid size of 100×100. For each experiment the simulation executed for 1500 time steps. Because of the random nature of the simulation, we calculate the average percentage of clusters on the grid and the average percentage of clusters carried by ants for each time step over 100 simulations.

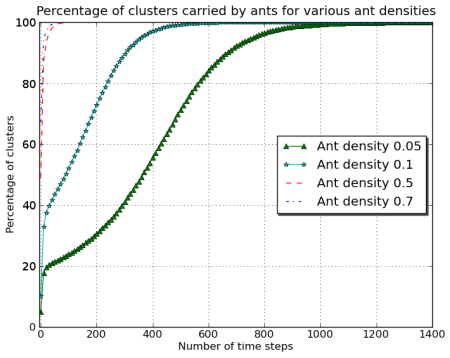
For the first experiment, we consider the effect of different ant densities on the deadlock state. An object density of 0.25 was used. In other words, the grid was initialized so that a quarter of the grid was filled with coloured chips. Figure 5.1 shows how the density of ants affect the number of time steps it takes before all the clusters on the grid are picked up. As expected, a higher ant density implies that all the clusters are picked up quicker than with a low density of ants. For example, with an ant density of 0.7, all clusters are already picked up by time step 100. We also see, however, that regardless of the density of ants the deadlock state is still reached in a relatively short timespan, leaving the grid empty of any clusters and defeating the purpose of sorting the objects on the grid. These results are the same for different grid sizes. Note, however, that for smaller grids, for example a grid of size 10×10, another kind of deadlock might occur where each ant is carrying a cluster and the clusters on the grid is in such a configuration that no ant can drop their respective clusters. For example, if every ant is carrying a yellow cluster and there are no more yellow clusters on the grid.

The effect of different object densities on the time it takes for the deadlock to occur is shown in Figure 5.2. A constant ant density of 0.1 was used. As with different ant densities, we see here that regardless of the density of objects the deadlock is still reached in a relatively small number of time steps.

From the results shown in this section we see that the deadlock occurs regardless of the initial ant or object densities. As was mentioned in Chapter 4.2.2 (see page 38), the reason is that objects are clustered together to form a new larger object, hence reducing the number of objects on the grid. This means that the second condition of Lumer and Faieta, namely, the number of objects must exceed the number of ants by at least an order of magnitude, cannot be met (see page 38). Our solution

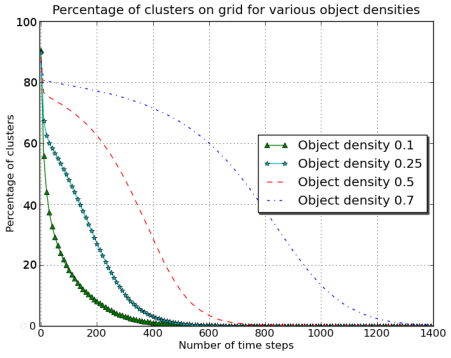


(a) Percentage of clusters on grid.

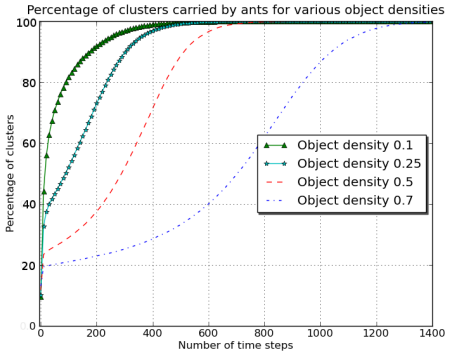


(b) Percentage of clusters carried by ants.

Figure 5.1: Percentage of clusters for varying ant densities.



(a) Percentage of clusters on grid.



(b) Percentage of clusters carried by ants.

Figure 5.2: Percentage of clusters for different object densities.

to this problem was to specify that an ant can only pick up a cluster if there is at least one other cluster on the grid with the same attributes. This is, however, not the most efficient solution as it needs to keep track of the number of clusters that is on the grid for each attribute. Another possible solution might be to calculate the pick up probability in such a way that the size of a cluster is taken into account. In this way the heavier clusters (clusters consisting of a large number of objects) are less likely to be picked up. This, in turn, might help with the deadlock situation as even though the merging of objects decreases the number of objects on the grid, these objects would also have a smaller risk of being picked up. A more detailed investigation of the deadlock issue could be interesting to investigate.

Having addressed the problem of deadlock in the clustering implementation, we now go on to evaluate the clustering implementation. In the next section we investigate the performance of our implementation in terms of its running time.

5.2 Time complexity

The time complexity for each time step in a CA is usually $O(n^2)$, where n^2 is the number of cells in a square grid with size $n \times n$. This is also true for clustering when every cell in the $n \times n$ grid is seen as an individual cluster. For the example model we compare the sorting times of both the traditional CA and the CA with clustering implementations by looking at the number of time steps required for objects to be sorted.

For the traditional CA implementation of the LF-algorithm it seems that the objects are rarely sorted in such a way that an optimal clustering solution is found. Also, it usually takes a large number of time steps before a relatively acceptable solution is found. We measure the time by counting the number of time steps it takes for the objects to be sorted on the grid. For all the experiments that follow a grid size of 50×50, an ant density of 0.05, an object density of 0.25 and the 5 base colours (as described in Section 3, page 18) is used.

5.2.1 Comparing the average sorting performance of the clustering and traditional CA implementations

In the first experiment we calculate the average sorting performance of both the clustering and traditional CA implementations. Each test case is executed for 10000 time steps. The total number of clusters on the grid and those currently being carried by ants are added and the average taken over 100 test cases.

The results of this experiment are shown in Figure 5.3. From these results we see that the clustering implementation has sorted the data into under 100 clusters in less than 1000 time steps on average, while for the traditional CA implementation the total number of clusters are still above 100 clusters by 10000 time steps.

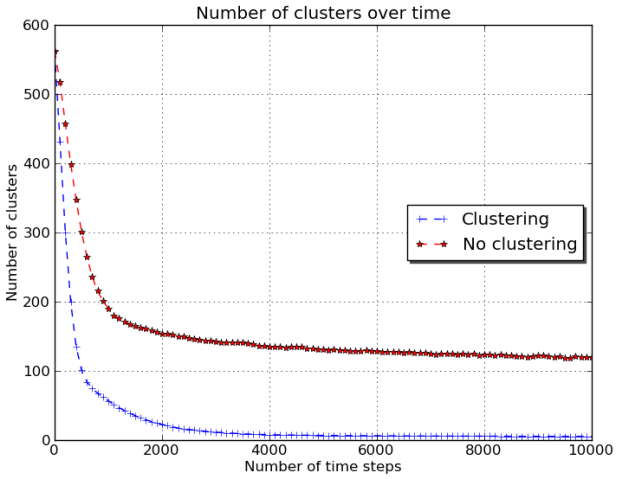


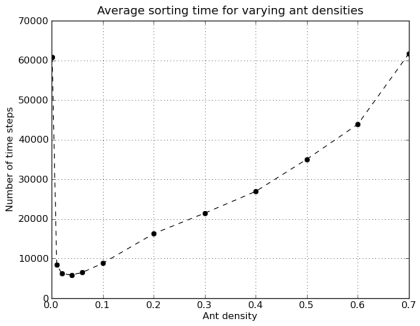
Figure 5.3: Number of clusters on grid for the clustering and traditional CA implementations.

Also, the clustering implementation already approaches a solution by 4000 time steps, while the traditional CA implementation has not reached a solution or an approximation of a solution by 10000 time steps. Note that here an approximate solution would be one where the total number of clusters are close to the value of the number of different attributes of the objects on the grid, in our case five.

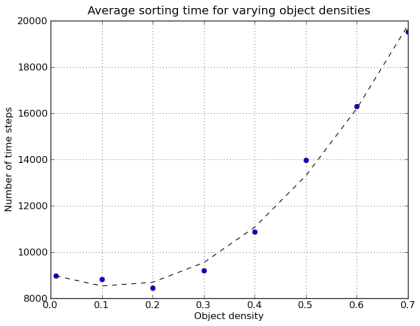
The above experiment shows that the clustering implementation performs better than the traditional CA implementation in terms of the sorting time of the model. The question now arises whether the same results would be achieved with different parameters. In the following experiments we investigate this by varying the ant density, object density, grid size and number of colours on the grid.

5.2.2 Effect of ant density on the sorting time

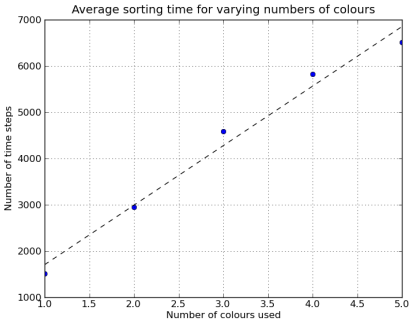
In the simulation of the LF-model the number of ants on the grid are important as it is only through their interaction with the objects on the grid that these objects can be sorted. Intuitively, it seems that if there are too few ants on the grid the sorting process will take longer as the ants will have to work harder to move the objects. Also, if there are too many ants on the grid this might slow down the sorting as the available space to traverse on the grid is smaller and the ants could



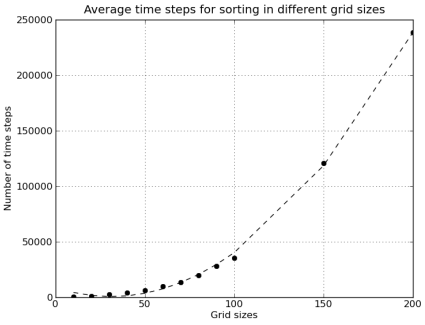
(a) Average number of time steps to sort clusters for varying ant densities.



(b) Average number of time steps to sort clusters for varying object densities.



(c) Average number of time steps to sort clusters by varying the number of attributes.



(d) Average number of time steps to sort clusters for different grid sizes.

Figure 5.4: Effect of the different parameters on the sorting time of the clustering implementation.

get into each others way. In this section, we calculate the average number of time steps it takes to sort the objects on the grid for various ant densities to see how the ant density effects the sorting time.

From Figure 5.4(a) we see that if the ant density is chosen to be too low, the sorting time increases and if the ant density is too high the sorting time is also high. The long sorting time with a higher ant density can be attributed to two factors. Firstly, with a higher ant density the moving space of an ant is less and it takes longer for the ant to traverse the grid. Secondly, with more ants on the grid there is a higher chance that more clusters would be picked up and less left on the grid. This makes it more difficult for an ant to find clusters with similar objects on the grid.

Lumer and Faieta give the two constraints that the number of objects on the grid must be at least an order of magnitude smaller than the size of the grid, and the number of ants on the grid must be at least an order of magnitude smaller than that of the number of objects on the grid. This means that, roughly speaking, the number of ants on the grid must be at least two orders of magnitude smaller than the grid size. It is interesting to note that the best results in terms of time are found between the densities 0.02 and 0.06 which gives a number of ants that roughly complies to Lumer and Faieta's constraint.

5.2.3 Effect of the size of the grid on the sorting time

We now calculate the effect of the size of the grid on the sorting time of the clustering implementation. As the grid size increases, it seems reasonable to assume that the sorting time will also increase. The reason is that there is a larger space that the ants must traverse to find similar clusters on the grid. In Figure 5.4(d) we can see that this is indeed the case and that the increase is, in fact, quadratic. The quadratic increase is linked to the squared grid size of $n \times n$.

5.2.4 Effect of object density and number of different attributes on the sorting time

As it is the objects that have to be sorted on the grid, it follows that their density would have an effect on the sorting time. From Figure 5.4(b), which shows the number of clusters formed for different time steps with varying object densities, we see that this is indeed the case. We see that the sorting time for densities below 0.4 is under 10000 time steps, which is relatively short. Again, this is in line with the constraint given by Lumer and Faieta as all the densities below 0.4 form a number of objects which is at least one order of magnitude less than that of the grid size.

Also, note that the number of different attributes on the grid affects the initial configuration of the objects placed on the grid. For example, if all objects has the same attribute and the object density is high there would be few, large clusters on

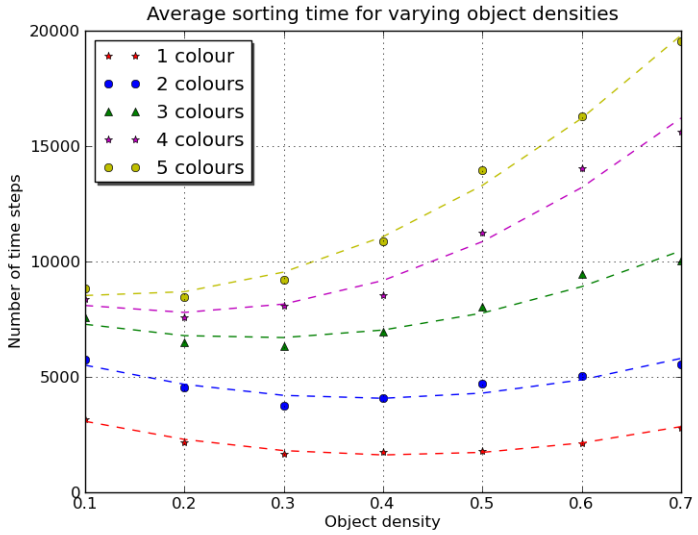


Figure 5.5: Effect of different object densities on the sorting time of the clustering implementation.

the grid. From Figure 5.5 we see that the slope of increase for object densities is steeper for a larger number of differing attributes on the grid.

From Figure 5.4(c), which shows the effect of the number of attributes (in our case the number of colours) on the sorting time, we see that the sorting time increases linearly with the number of attributes. As the number of attributes increase, the difficulty of finding similar objects on the grid also increases. However, this increase is linear as the increase in attributes does not affect the number of ants or the size of the grid, which, as was shown in the previous sections has a significant influence on the sorting time.

In summary, the above experiments show that the grid size and the ant density have a significant effect on the sorting time of the objects. This is because the grid size defines the space in which the ants move around to find suitable sites to drop objects. The number of ants also affects the sorting time, and the best performance is found between an ant density of 0.02 and 0.06. An increase in the object density and an increase in the number of colours on the grid also cause the sorting time to increase but the increase in sorting time is smaller than that of a higher ant density or larger grid size.

Because of time constraints we have not done the above experiments on the traditional CA implementation, as the simulation for a grid size of 50×50, for example, have not reached a solution yet by 100000 time steps. We, however, predict the results to be the same in terms of the increase of sorting time and sorting difficulty as the various parameters is increased. Note that, for the traditional CA implementation, if the number of ants is more than the number of objects on the grid a deadlock situation similar to the one described in Section 4.2.2 will occur. In the next section we look at the space complexity of the clustering implementation.

5.3 Space complexity

In general the space used by CA is $O(n^2)$ where n^2 is the size of the grid. As CA with cell clustering uses an extra data structure, it follows that it uses more space than traditional CA. This space usage depends on the data structure used and the specific implementation of the CA. In the following section we discuss the space complexity for the clustering implementation.

In our clustering example, a map data type was used to store the clustering information. The $n \times n$ CA grid is then used to store references to the clusters in the map. The total number of objects in the map is constant (note that we count all the objects whether they are in the same cluster or not). Thus we have an extra data structure with a constant size of $O(d)$, where d is the number of objects to be sorted and $d < n^2$.

It is, however, also possible (maybe even more likely) that in a CA with clustering implementation all the cells in the CA will be divided up into clusters. This means that the map data type would store a number of objects which is the same as the number of cells on the grid. If the grid is of size $n \times n$, the total number of cells would be n^2 and thus the size of the clustering data structure would also be n^2.

In conclusion, the CA with clustering increases the space requirement of the implementation by n^2 in the worst case situation. Adding the size of the CA grid and the map data type we get a worst case space complexity of $n^2 + n^2 = 2n^2$ which is still of $O(n^2)$.

In this section we briefly discussed the space complexity of both traditional CA and CA with cell clustering. The next section describes a way to compare the performance of the implementation of the traditional CA and the clustering methods in terms of their sorting results.

5.4 Performance of clustering

In the LF-model the goal is to sort data objects in such a way that objects with similar attributes will be placed closer to each other and objects with different attributes will be separated. Thus, the goal is to minimize the distance between objects with similar attributes and separate them from objects with different attributes.

In our specific implementation this translates to sorting the coloured objects in such a way that eventually these objects will form one cluster for each colour. Therefore, we are interested in the intra-cluster distance (that is, the distance between two clusters with similar attributes) as well as the inter-cluster distance (that is, the distance between two clusters with different attributes). Our measure of success is then to minimize the intra-cluster distance and maximize the inter-cluster distance.

Visually, the results of the clustering implementation seems promising. However, we need a mathematical measure to compare the clustering results with those of the traditional CA implementation. There are a variety of possible ways to compare these results, with the simplest being a distance measure. In this section, we therefore compare the average intra-cluster distance of both the clustering and traditional CA implementations. The average inter-cluster distance is also measured and compared.

5.4.1 The distance between two clusters

In the following experiments the distance between two clusters A and B are defined to be the smallest Euclidean distance found between the set of objects in A and

B. The Euclidean distance between two objects a_i and b_j at positions (a_{i_x}, a_{i_y}) and (b_{j_x}, b_{j_y}) are calculated as follows [19]:

$$d(a_i, b_j) = \sqrt{(a_{i_x} - b_{j_x})^2 + (a_{i_y} - b_{j_y})^2} \tag{5.1}$$

If $A = \{a_1, a_2, ..., a_m\}$ and $B = \{b_1, b_2, ..., b_n\}$, then we define the distance matrix for two clusters as follows:

$$DM_{(A,B)} = \begin{pmatrix} d(a_1, b_1) & d(a_1, b_2) & \cdots & d(a_1, b_n) \\ d(a_2, b_1) & d(a_2, b_2) & \cdots & d(a_2, b_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(a_m, b_1) & d(a_m, b_2) & \cdots & d(a_m, b_n) \end{pmatrix}. \tag{5.2}$$

The distance matrix $DM_{(A,B)}$ contains the Euclidean distances between the sets of objects in the Cartesian product $A \times B$. The distance λ between two clusters is calculated by finding the smallest value in the distance matrix $DM_{(A,B)}$:

$$\lambda_{(A,B)} = min(DM_{(A,B)}) \tag{5.3}$$

5.4.2 The average distance between clusters in a given set

The average distance $\lambda'_{(\mathcal{X},\mathcal{Y})}$ between two sets of clusters $\mathcal{X}$ and $\mathcal{Y}$ shows, on average, how far the clusters in set $\mathcal{X}$ are from the clusters in set $\mathcal{Y}$. To calculate $\lambda'_{(\mathcal{X},\mathcal{Y})}$, the matrix $D_{(\mathcal{X}\times\mathcal{Y})}$ is needed.

Suppose $\mathcal{X} = \{X_1, X_2, ..., X_m\}$ and $\mathcal{Y} = \{Y_1, Y_2, ..., Y_n\}$. Then $D_{(\mathcal{X}\times\mathcal{Y})}$ for these two sets is obtained by calculating the distance $\lambda_{(X_i,Y_j)}$ for all pairs of clusters in the cartesian product $\mathcal{X} \times \mathcal{Y}$:

$$D_{(\mathcal{X}\times\mathcal{Y})} = \begin{pmatrix} \lambda_{(X_1,Y_1)} & \lambda_{(X_1,Y_2)} & \cdots & \lambda_{(X_1,Y_n)} \\ \lambda_{(X_2,Y_1)} & \lambda_{(X_2,Y_2)} & \cdots & \lambda_{(X_2,Y_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{(X_m,Y_1)} & \lambda_{(X_n,Y_2)} & \cdots & \lambda_{(X_m,Y_n)} \end{pmatrix}. \tag{5.4}$$

Using the resulting matrix from Equation 5.4, the average distance between $\mathcal{X}$ and $\mathcal{Y}$ are then defined as the mean of the entries in $D_{(\mathcal{X}\times\mathcal{Y})}$:

$$\lambda'_{(\mathcal{X},\mathcal{Y})} = mean(D_{(\mathcal{X}\times\mathcal{Y})}). \tag{5.5}$$

In the next section we use the average distance described above to calculate the inter- and intra-cluster distances of clusters to compare the sorting of the traditional CA implementation with that of the clustering implementation.

5.4.3 Experiments and results

For the following experiments a grid size of 50×50, an object density of 0.25 and an ant density of 0.05 have been used. The same initial object and ant configurations have been used for both the traditional CA and the clustering implementation.

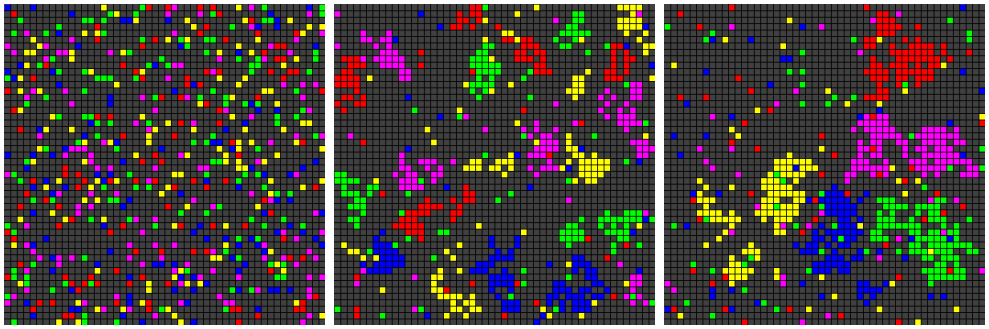
Figure 5.6 shows the configuration of the objects on the grid for the time steps 0, 10000 and 100000. We can see how the clusters form with the progression of time. For the clustering implementation the objects have already been sorted into their respective clusters by time step 10000 (see Figure 5.6(e)).

5.4.3.1 Average intra-cluster distance

For this experiment we calculate the average intra-cluster distance for the five different colours used in our implementation.

Given a set of clusters $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ with similar attributes, the matrix $D_{(\mathcal{S}\times\mathcal{S})}$ for all pairs of clusters in the cartesian product $\mathcal{S} \times \mathcal{S}$ can be calculated. The average distance $\phi_{(S_i,\mathcal{S})}$ of each cluster $S_i \in \mathcal{S}$ from all other clusters in the set $\mathcal{S}$ can then be calculated. The average intra-cluster distance for a set $\mathcal{S}$ would then be $\lambda'_{(\mathcal{S},\mathcal{S})}$.

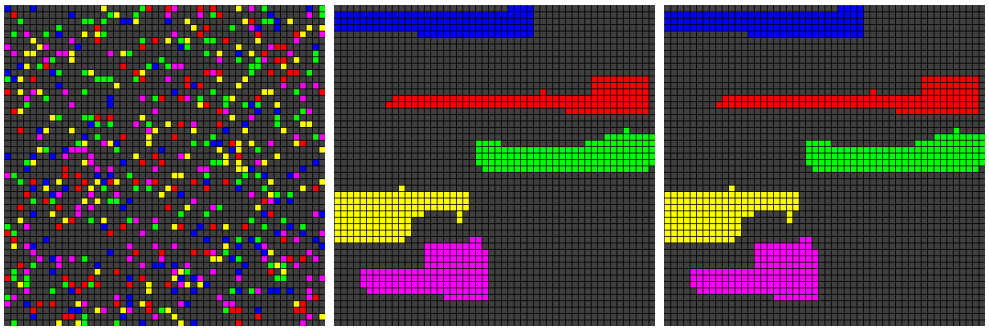
In Figure 5.7 and Figure 5.8 a comparison between the traditional CA and clustering implementations are shown. As was mentioned earlier, the goal is for intra-cluster distances to be small. This means that similar clusters should be packed close together, ideally forming one large cluster. In the figures we can see that for the



(a) Traditional CA implementation at time step 0.

(b) Traditional CA implementation at time step 10000.

(c) Traditional CA implementation at time step 100000.



(d) Clustering method at time step 0.

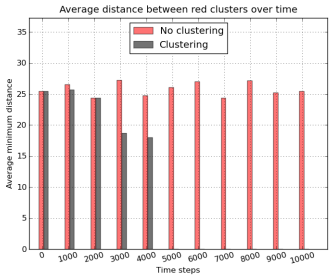
(e) Clustering method at time step 10000.

(f) Clustering method at time step 100000.

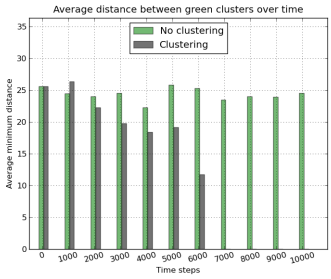
Figure 5.6: Configuration of objects on grid during different time steps.

clustering implementation the intra-cluster distance for all of the coloured clusters is already equal to zero by ten thousand time steps (see Figure 5.7). This indicates that all objects on the grid have been sorted into their respective clusters, as can be confirmed by the grid configuration shown in Figures 5.6(e) and 5.6(f).

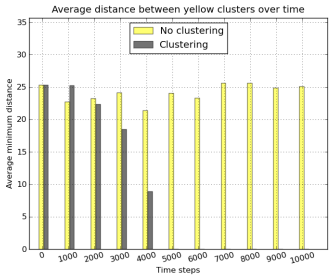
From the results in Figures 5.7 and 5.8 we see that the average intra-cluster distance for the traditional CA implementation decreases very slowly and by time step 100000 it is still very close to the starting distance. If we look at the images in Figure 5.6, it is clear that the objects are being sorted into larger clusters. Note, that there is also smaller clusters scattered between these larger clusters. This is because we let all the ants drop the current objects they are carrying, before we measure the clusters on the grid. It is these scattered objects that cause the results shown in Figures 5.7 and 5.8. We investigate this issue further in Section 5.4.3.3.



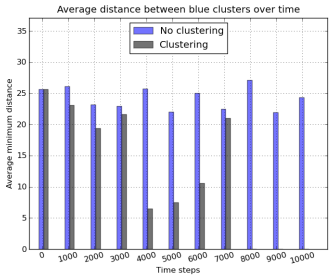
(a) Average intra-cluster distance for red coloured clusters over time.



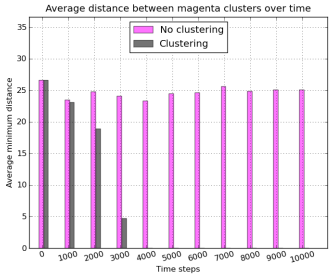
(b) Average intra-cluster distance for green coloured clusters over time.



(c) Average intra-cluster distance for yellow coloured clusters over time.



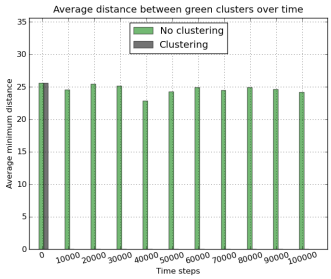
(d) Average intra-cluster distance for blue coloured clusters over time.



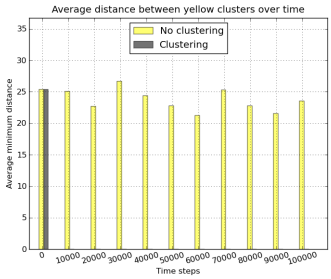
(e) Average intra-cluster distance for magenta coloured clusters over time.

Figure 5.7: Average intra-cluster distances. Time steps 0 to 10000, with 1000 step intervals.

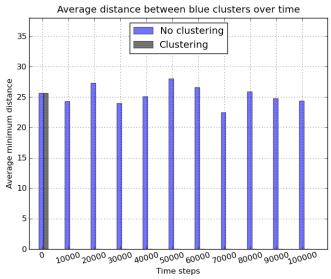
(a) Average intra-cluster distance for red coloured clusters over time.



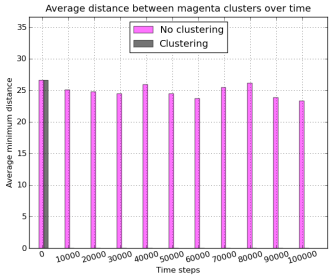
(b) Average intra-cluster distance for coloured clusters over time.



(c) Average intra-cluster distance for yellow coloured clusters over time.



(d) Average intra-cluster distance for blue coloured clusters over time.



(e) Average intra-cluster distance for magenta coloured clusters over time.

Figure 5.8: Average intra-cluster distances. Time steps 0 to 100000, with 10000 step intervals.

5.4.3.2 Average inter-cluster distance

In this section we calculate the average inter-cluster distances for the five different colours used in our implementation.

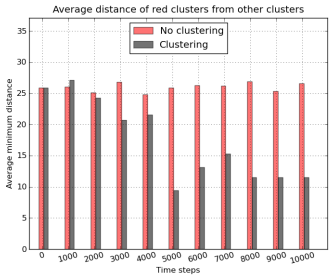
Given two sets of clusters $\mathcal{X} = \{X_1, X_2, ..., X_n\}$ and $\mathcal{Y} = \{Y_1, Y_2, ..., Y_n\}$ the average distance between these two sets would be $\lambda'_{(\mathcal{X},\mathcal{Y})}$. For our specific experiment the set $\mathcal{X}$ is the set of clusters with similar attributes and the set $\mathcal{Y}$ is the set of all the other clusters on the grid.

From the results shown in Figures 5.9 and 5.10 we see that for the clustering implementation the resulting inter-cluster distances are smaller than those of the traditional CA. The noise in the traditional CA implementation, however, influences these results and thus we investigate this influence in the next section. It is also interesting to note that from Figure 5.10 we see that the distance stabilizes by 20000 time steps. By that time, the clustering implementation has finished sorting and no more changes are made to the resulting cluster configurations.

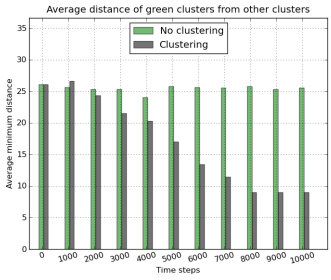
5.4.3.3 Influence of noise on clustering measurements

Before each clustering measurement is taken, the ants first drop all the objects they are carrying. Because these objects are then randomly scattered on the grid, it has a significant influence on the results of the calculations described in the previous sections. For the clustering implementation this does not really matter, because the objects are quickly sorted in such a way that only a few clusters are both being carried and on the grid. However, the dropped objects have a substantial effect on the results of the clustering calculations for the traditional CA implementation. The difference between the clustering implementation and the traditional CA implementation, with noise, can be clearly seen in Figure 5.6. In this section we compare the results of the traditional CA implementation with noise and the traditional CA without noise.

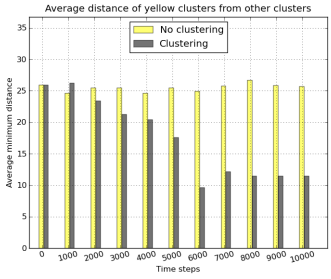
Figure 5.11 shows the results of the traditional CA implementation without noise. Note that by 10000 time steps, the objects are clustered into smaller coloured objects as opposed to the clustering implementation where the objects are already



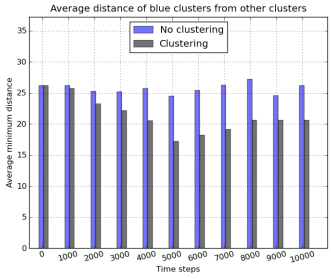
(a) Average distance between red coloured clusters over time.



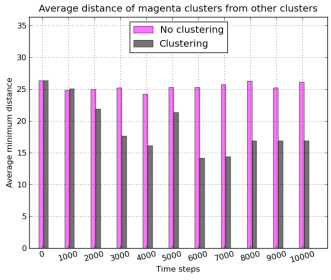
(b) Average distance between green coloured clusters over time.



(c) Average distance between yellow coloured clusters over time.



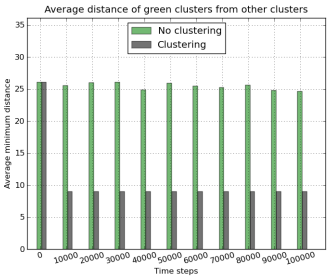
(d) Average distance between blue coloured clusters over time.



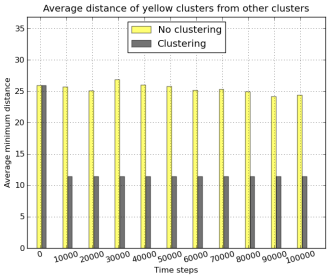
(e) Average distance between magenta coloured clusters over time.

Figure 5.9: Average distance between different coloured clusters. Time steps 0 to 10000, with 1000 step intervals.

(a) Average distance between red coloured clusters and other clusters over time.

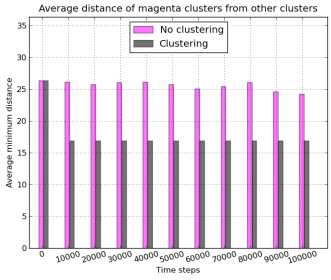


(b) Average distance between green coloured clusters and other clusters over time.



(c) Average distance between yellow coloured clusters and other clusters over time.

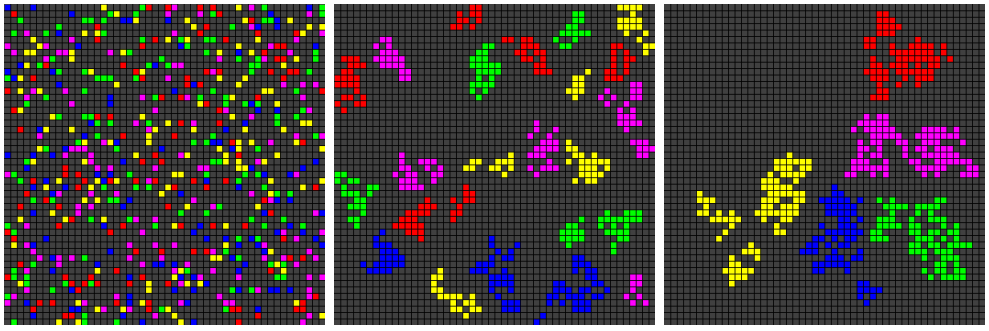
(d) Average distance between blue coloured clusters and other clusters over time.



(e) Average distance between magenta coloured clusters and other clusters over time.

Figure 5.10: Average distance between different coloured clusters. Time steps 0 to 100000, with 10000 step intervals.

clustered into their respective clusters (see Figure 5.6(e) on page 54). By 100000 time steps we see that bigger clusters have emerged.



(a) Traditional CA implementation at time step 0.

(b) Traditional CA implementation at time step 10000.

(c) Traditional CA implementation at time step 100000.

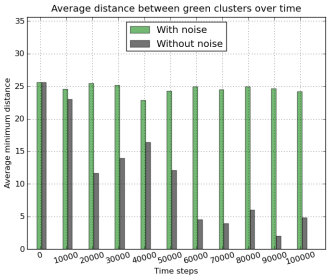
Figure 5.11: Configuration of objects on grid during different time steps for the traditional CA implementation with and without noise.

In Figure 5.12 we compare the intra-cluster distances for the traditional CA implementation with noise and without noise. We see that the distances, when the noise is removed, decrease as the same coloured objects are placed in clusters closer to each other. However, note that in Figure 5.12(e) at 80000 time steps the intra-cluster distance is 0, which means that there is only one cluster of that colour on the grid, but at 90000 time steps the intra-cluster distance is higher. This means that even when the traditional CA implementation finds a solution, the possibility exists that a cluster can be destroyed by the ants.

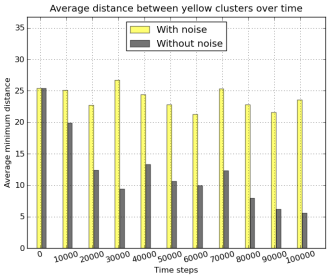
From Figure 5.13 where, we compare the inter-cluster distances for the traditional CA implementation with noise and without noise, we see that there is little difference between the two. Note that for both the traditional CA and clustering implementations the different coloured clusters are separated.

From the results in this section we see that the clustering implementation performs better in sorting the objects in such a way that the intra-cluster distances is small. It is, in fact, such that the intra-cluster distances are zero for all clusters. When ignoring the noise in the traditional CA implementation, we also see that the intra-cluster distances is small, but even after 100000 time steps it is still larger than that

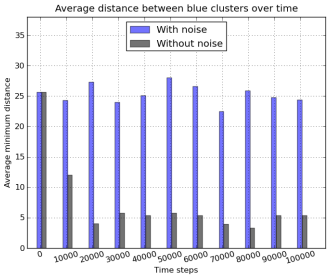
(a) Average intra-cluster distance for red coloured clusters.



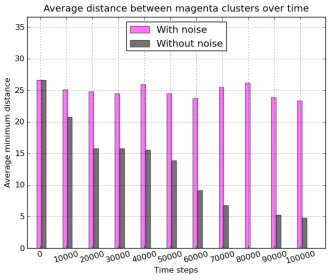
(b) Average intra-cluster distance for green coloured clusters.



(c) Average intra-cluster distance for yellow coloured clusters.



(d) Average intra-cluster distance for blue coloured clusters.

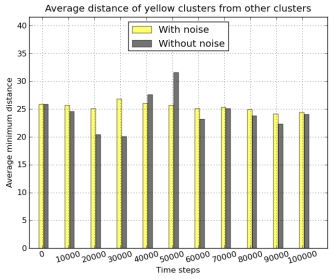


(e) Average intra-cluster distance for magenta coloured clusters.

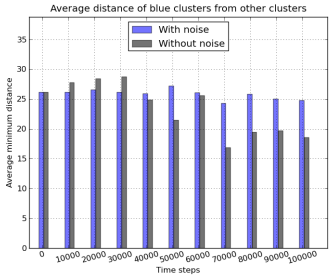
Figure 5.12: Average intra-cluster distances for the traditional CA implementation with and without noise. Time steps 0 to 100000, with 10000 step intervals.

(a) Average distance between red coloured clusters and other clusters over time.

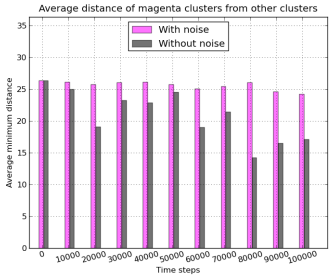
(b) Average distance between green coloured clusters and other clusters over time.



(c) Average distance between yellow coloured clusters and other clusters over time.



(d) Average distance between blue coloured clusters and other clusters over time.



(e) Average distance between magenta coloured clusters and other clusters over time.

Figure 5.13: Average inter-cluster distances for the traditional CA implementation with and without noise. Time steps 0 to 100000, with 10000 step intervals.

of the clustering implementation. Also, for the traditional CA implementation the possibility exists that the clustering solution can be destroyed by the ants, whereas for the clustering implementation it is not possible. In terms of the inter-cluster distances, both implementations have a favourable outcome, as the clusters are clearly separated.

5.5 Miscellaneous

In this section we discuss some of the interesting problems that we have encountered during our implementation of the modified LF-model.

5.5.1 Surrounded cluster deadlock

The first interesting occurrence, which was described in Section 4.2.3 (page 40), is the surrounded cluster deadlock. Summarized, this is the situation where a cluster c_i with colour attribute x is surrounded by one or more clusters with attributes that differs from x, while there are no other clusters of colour x on the grid. Our solution is to keep track of the number of time steps that no clusters have been dropped on the grid and if a certain threshold value is reached, all the ants drop the clusters they are currently carrying.

In the following experiment we investigate how frequently the surrounded cluster deadlock occurs for the three different drop methods described in Section 4.2.1 on page 35. For this experiment we used a grid size of 50×50, ant density of 0.05 and object density of 0.25. We calculate the percentage of times the deadlock occurs over 500 test cases.

From Figure 5.14 we see that for the diamond drop method the deadlock occurs less frequently than in the other drop methods. This can be attributed to the fact that the diamond drop only takes local object density into account when calculating if a cluster should be dropped and then, subsequently, drops the rest of the objects in the cluster around the first dropped object regardless of the density of other

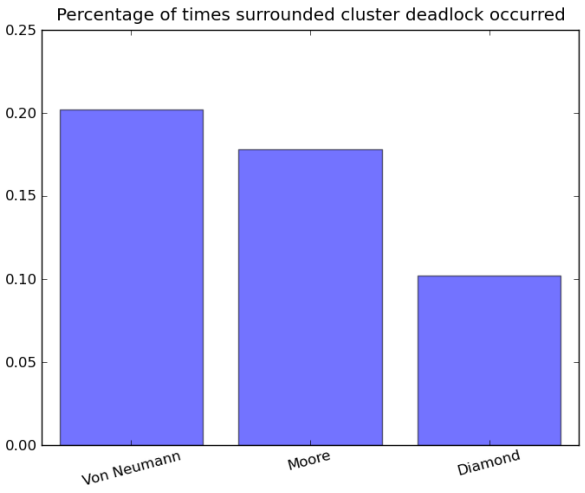


Figure 5.14: Percentage of times surrounded cluster deadlock occurred for different drop methods.

objects in the neighbourhood. In general the surrounded cluster deadlock occurs less than 20 percent of the time.

5.5.2 Dropping of clusters

Another problem that we came across in our implementation is, of course, the way a cluster should be dropped. This issue is discussed in detail in Section 4.2.1 on page 35. In our implementation we have ignored the shape of the cluster when picked up and only preserved its size. We described three methods that can be used to determine how a cluster should be dropped, of which one is concerned with dropping the cluster in a certain shape, namely the diamond drop method. In this section we compare these three drop methods in terms of their average sorting times.

For the following experiment a grid size of 50×50, an ant density of 0.05 and an object density of 0.25 was used. Figure 5.15 shows that the average sorting time for all three drop methods are relatively close to each other. From these results we can see that for the clustering implementation of the LF-algorithm, none of these three drop methods have any significant advantage over each other in terms of sorting time.

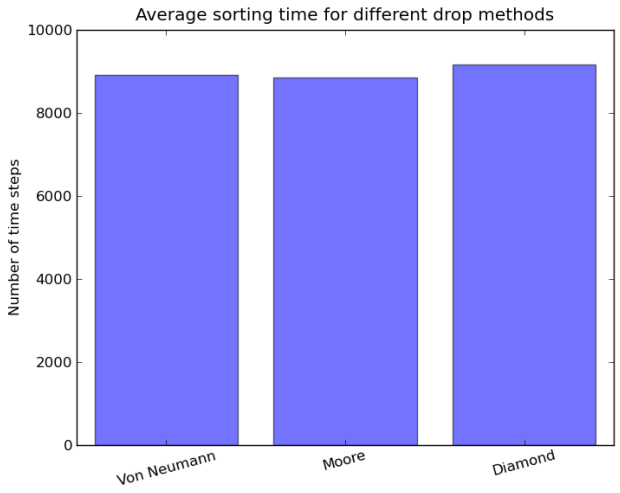
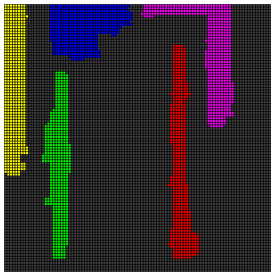
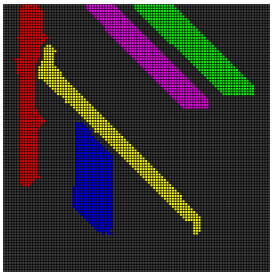


Figure 5.15: The average sorting time for different drop methods.

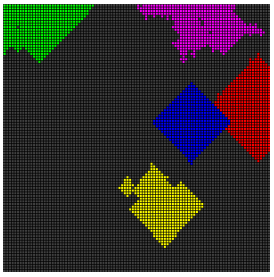
An interesting difference to note for these three drop methods, is their inter-cluster distances. We present Figure 5.16 as an example. As can be seen, the clusters in the Von Neumann and Moore drop results are clearly separated with their inter-cluster distances greater than zero, while the results for the diamond drop method shows the red and blue clusters touching (that is, their inter-cluster distance is zero). This is because while dropping a cluster the diamond drop method does not use the local density function to determine the next available location for each object to be dropped. This is not necessarily a disadvantage as packing clusters tightly might be used in layout problems.



(a) The Von Neumann drop method.



(b) The Moore drop method.



(c) The diamond drop method.

Figure 5.16: Output for three different drop methods.

In this chapter, we have compared the modified LF-model with the traditional CA implementation of the LF-model in terms of runtime, space usage and the quality of the clusters formed on the grid. For the sorting time and cluster quality, the clustering implementation shows better performance than in the traditional CA implementation. It is, however, important to notice that the quality of clusters is open to interpretation, and depends on the implementation and the goal of such an implementation. The clustering implementation uses more space than that of the traditional CA. This extra space usage also differs for different implementations. However, the extra space needed would normally not be larger than the grid size of the CA.

Chapter 6

Conclusion

The goal of this thesis was to investigate the concept of clustering in CA and explore the implementation of cell clustering in CA through an example model. The LF-model was used as an example because it seemed intuitive to apply CA with cell clustering to natural clustering problems. In this chapter we give a summary of the main issues discussed in this thesis and mention possible future work.

6.1 Cellular automata and clustering

The defining difference between traditional CA and CA with cell clustering is that in CA with cell clustering cells are allowed to cluster together to form a larger *'cell'*. This means that in cell clustering the number of cells in the grid is not constant and can change with every time step.

Another notable difference between traditional CA and CA with cell clustering is their neighbourhoods. The size and shape of the neighbourhood in traditional CA stays constant throughout the simulation of the model, while this is not the case for CA with cell clustering. Also, the total number of neighbours is constant for traditional CA, whereas for CA with cell clustering, the total number of neighbours change with the clusters on the grid.

Implementationwise, CA with cell clustering need an extra data structure to store the clustering information and parallelisation is not as straightforward as for traditional CA.

6.2 Implementation of clustering

Through implementing clustering in an example model, it was possible to compare the implementation of clustering with that of the traditional CA implementation. It was shown that an existing model showing inherent clustering can be modified to apply to CA with cell clustering. The difficulty of the modification process would depend on the chosen model. Because of the inherent clustering in the LF-model it is relatively easy to modify the model in such a way that the CA with cell clustering concept could be incorporated.

In the modified LF-model we have chosen not to change the calculation of pick up and drop probabilities to take in account clustering information. This, however, might not be the best course of action when implementing clustering as, for our specific implementation, the size of a cluster is important in terms of the fact that it covers more space on the grid which would make it easier for an ant to find it when searching for a place to drop similar clusters. Thus, it would be better to calculate the probabilities in such a way that smaller clusters would have a higher probability of being picked up while larger clusters are not as easily picked up.

In Chapter 5 we compared the implementation of the modified LF-model to the traditional CA implementation. We have shown that even with only the most basic modifications to the LF-model that, in terms of the sorting time and cluster quality, the clustering implementation shows a better performance than the traditional CA implementation. On the other hand, in terms of memory usage the traditional CA implementation uses less space than the clustering implementation.

6.3 Future work

As the idea of CA with cell clustering is a relatively new one, there are still a lot of interesting issues that can be researched. This includes the generalization of clustering to n dimensions, a more detailed investigation into the neighbourhoods and transition rules of CA with cell clustering and the effect of clusters with holes in them on these definitions.

For our implementation example we adapted the LF-model to add cell clustering to it. A number of modifications have been made to the LF-model to speed up the searching process of ants and to improve the clustering results [19]. We only modified the original LF-model and not these improved versions, but it would be interesting to see if these improved models could be adapted to clustering, and if this in turn would further improve these models. Some of the improvements added to the LF-model include using different pick up and drop probabilities, ants having different moving speeds and giving ants a short-term memory.

We have already mentioned that the clustering version of the LF-model might be improved if the pick up and drop probabilities are calculated in such a way that clustering information could be taken into account. Also, if an ant were to have a short-term memory it might help the ant to decide in which direction to move to find similar clusters to the one it is currently carrying, thus reducing the time it would otherwise have taken to find such a position.

In conclusion, adding clustering to CA opens a whole new avenue for interesting research possibilities. The results obtained from our example implementation show promising results despite the fact that we have not purposefully tried to optimize the implementation. We believe that clustering would be fairly easy to apply to models that have an inherent clustering behaviour and that it could improve these models. It would be interesting to investigate more models such as the LF-model and see if adding clustering could improve the model.

Bibliography

[1] Al-Anzi, F.S.: Efficient Cellular Automata Algorithms for Planar Graph and VLSI Layout Homotopic Compaction. *International Journal of Computing and Information Sciences*, vol. 1, no. 1, pp. 1–17, 2003.

[2] Amani, J., Rezaie Moghaddam, F. and Rezaie Moghaddam, T.: Solution of Two Dimensional Quasi-Harmonic Equations with CA Approach. *World Academy of Science, Engineering and Technology*, vol. 59, pp. 132–137, November 2009.

[3] Apte, A., Bonchev, D., Cain, J. and Fong, S.: Cellular Automata Simulation of Topological Effects on the Dynamics of Feed-Forward Motifs. *Journal of Biological Engineering*, vol. 2, no. 1, 2008.

[4] Arata, H., Saito, T., Takai, N.K. and Takai, Y.: Virtual Clay: A Deformation Model by Three-Dimensional Cellular Automata. *Computer Graphics and Geometry*, vol. 2, no. 1, pp. 50–74, 2000.

[5] Billings, S.A. and Yang, Y.: Identification of Probabilistic Cellular Automata. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 33, no. 2, pp. 225–236, April 2003.

[6] Blair, E.P. and Lent, C.S.: Quantum-Dot Cellular Automata: An Architecture for Molecular Computing. In: *Proceedings of International Conference on Simulation of Semiconductor Processes and Devices, SISPAD 2003*, pp. 14–18, September 2003.

[7] Botha, L. and Van Zijl, L.: Feature Extraction for Image Pattern Matching with Cellular Automata. In: *Proceedings of the Prague Stringology Conference 2009*, pp. 3–14. Czech Technical University in Prague, Czech Republic, 2009.

[8] Bonabeau, E., Dorigo, M. and Theraulaz, G.: *Swarm Intelligence, From Natural to Artificial Systems*. Oxford University Press, Madison Avenue, New York, 1999.

[9] Caraco, T., Maniatty, W.A. and Szymanski, B.K.: Parallel Computing with Generalized Cellular Automata. Tech. Rep., Department of Computer Science, Rensselaer Polytechnic Institute, 1998.
Available at: `http://www.cs.rpi.edu/~szymansk/papers/pdcp.v1.pdf`

[10] Chaudhuri, P.P., Chowdhury, D.R., Nandi, S. and Chattopadhyay, S.: *Additive Cellular Automata: Theory and Applications*, vol. 1. IEEE Computer Society Press, Los Alamitos, California, 1997.

[11] Chen, Q., Recknagel, F. and Qu, S.: Cellular Automata Based Simulation of Random Versus Selective Harvesting Strategies in Predator-Prey Systems. *Ecological Informatics*, vol. 3, no. 3, pp. 252–258, 2008.

[12] Cheng, C., Kier, L.B. and Seybold, P.G.: Modeling Chemical Systems using Cellular Automata. In: *Modeling Nature*, pp. 1–8. Springer Netherlands, 2005.

[13] Das, S.: *Theory and Applications of Nonlinear Cellular Automata in VLSI Design.* Ph.D. thesis, Bengal Engineering and Science University, June 2006.
Available at: `http://it.becs.ac.in/content/sukanta_das/Thesis.pdf`

[14] Dehkordy, P.K., HosseynPeyravi, M., Kyoomarsi, F. and Torkestani, J.A.: Using Chemical Cellular Automata in Simulation of Chemical Materials. In: *Proceedings of 5th ACIS International Conference on Software Engineering Research, Management Applications, SERA 2007*, pp. 769–773, August 2007.

[15] Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C. and Chretien, L.: The Dynamics of Collective Sorting: Robot-Like Ant and Ant-Like Robot. In: *Proceedings of the First Conference on Simulation of Adaptive Behaviour: From Animals to Animats*, pp. 356–365. MIT Press, Cambridge, MA, 1991.

[16] Deutsch, A. and Dormann, S.: *Cellular Automaton Modeling of Biological Pattern Formation.* Modeling and Simulation in Science, Engineering and Technology. Birkhauser, Boston, 2005.

[17] Di Gregorio, S., Di Maio, F.P. and Lignola, P.G.: Cellular Automata Simulation of Coal Combustion. *Physical Chemistry Chemical Physics*, vol. 2, no. 1, pp. 83–39, 2000.

[18] Di Gregorio, S., Rongo, R., Spataro, W., Spezzano, G. and Talia, D.: High Performance Scientific Computing by a Parallel Cellular Environment. *Future Generation Computer Systems*, vol. 12, no. 5, pp. 357–369, 1997.

[19] Engelbrecht, A.P.: *Fundamentals of Computational Swarm Intelligence.* John Wiley & Sons, Ltd, 2006

[20] Fowdur, S.C. and Rughooputh, S.D.D.V.: Traffic Cellular Automata Simulation of a Congested Round-About in Mauritius. *International Journal of Modern Physics C*, vol. 20, pp. 459–468, 2009.

[21] Freitas, A.A., Lopes, H.S. and Parpinelli, R.S.: Data Mining with an Ant Colony Optimization Algorithm. *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 321–332, August 2002.

[22] Goodrich, M.T. and Tamassia, R.: *Algorithm Design: Foundations, Analysis, and Internet Examples.* John Wiley & Sons, Inc., 2002.

[23] Handl, J. and Meyer, B.: Ant-Based and Swarm-Based Clustering. *Swarm Intelligence*, vol. 1, no. 2, pp. 95–113, 2007.

[24] Kopetz, R.: *Extension of a Parallel Library for Cellular Computing on a Cluster of PCs and on Computer Grids.* Ph.D. thesis, University Politechnic of Bucharest, Bucharest, June 2004.
Available at: `http://www.metz.supelec.fr/~ersidp/Publication/OnLineFiles/kopetz_04.pdf`

[25] Leonski, W. and Walczak, M.: A Cavity with Two-Level Atoms and Cellular Automata Simulations. *Fortschritte der Physik*, vol. 51, no. 2-3, pp. 186–189, 2003.

[26] Lumer, E. and Faieta, B.: Diversity and Adaptation in Populations of Clustering Ants. In: *Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats*, pp. 499–508. MIT Press, Cambridge, MA, USA, 1994..

[27] Mange, D., Petraglio, E., Stauffer, A. and Tempesti, G.: Bio-Inspired Design of Computer Hardware by Self-Replicating Cellular Automata. *Ecological Informatics*, vol. 15, no. 7, pp. 125–147, 2006.

[28] Popovici, A. and Popovici, D.: Cellular Automata in Image Processing. In: *Proceedings of the 15th International Symposium on Mathematical Theory of Networks and Systems*, August 2002.

[29] Rosin, P.: Training Cellular Automata for Image Processing. *IEEE Transactions on Image Processing*, vol. 15, no. 7, pp. 2076–2087, 2006.

[30] Saldana, R.P. and Yu, W.E.S.: Cellular Autmata Explorations on a Beowulf Cluster Computer. In: *Proceedings of Cellular Automata 2001.* Yokohama National University, 2001.

[31] Smal, E.: *Automated Brick Sculpture Construction.* Master's thesis, University of Stellenbosch, August 2008.
Available at: `http://www.cs.sun.ac.za/~lynette/publications/ESmal.pdf`

[32] Toffoli, T.: CAM: A High-Performance Cellular-Automaton Machine. *Physica D: Nonlinear Phenomena*, vol. 10, no. 1-2, pp. 195–204, 1984.

[33] Weisstein, E.: Rule 150. From MathWorld-A Wolfram Web Resource, 2009.
Available at: `http://mathworld.wolfram.com/Rule150.html`

[34] Weisstein, E.: Cellular Automaton. From MathWorld - A Wolfram Web Resource, 2009.
Available at: `http://mathworld.wolfram.com/CellularAutomaton.html`

[35] Weisstein, E.: Dynamical System. From MathWorld-A Wolfram Web Resource, 2010.
Available at: `http://mathworld.wolfram.com/DynamicalSystem.html`

[36] Wolfram, S.: *Cellular Automata and Complexity: Collected Papers.* Addison-Wesley Publishing Company, Reading, MA, 1994.