Explorations in Stochastic Diffusion Search: soft- and hardware implementations of biologically inspired Spiking Neuron Stochastic Diffusion Networks

Kris De Meyer [1]
Department of Cybernetics
University of Reading

KDM/JMB/2000-1

September 2000

[1]Email: krm@cyber.reading.ac.uk

Abstract

Stochastic Diffusion Search, a probabilistic pattern matching algorithm, has been applied to a variety of problems with great success, and its basic properties have been studied in detail. Although parallel in nature, most of the applications are implemented in software on serial machines. This report aims at giving an overview of possible modifications towards efficient implementation in parallel hardware and towards implementations as networks of biologically inspired Spiking Neurons, in both hard- and software.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The History of Stochastic Diffusion Search

Stochastic Diffusion Search (SDS), a parallel probabilistic pattern matching algorithm, was first proposed in [1] and [2]. It is capable of rapidly locating a specified pattern - or its best instantiation - in a noisy Search Space. The algorithm is based on *partial testing* for the presence of a target pattern at locations in the Search Space by a number of *independently* operating *Agents*. Agents exchange information about possible locations of the target pattern in the Search Space (*diffusion*).

The technique has been applied with great success to a variety of real-world problems. In [3], the Hybrid Stochastic Diffusion Network (HSDN), a combination of Stochastic Diffusion Search with N-tuple Weightless Neural Networks ([4]), was used to locate eye features in images of human faces, invariant of transformations within the Search Space, robust to noise distortions and partial occlusions of the object. [5] used the same combination for more general facial feature location, whereas [6] used it as a method for self-localisation of an autonomous wheelchair, extending the original algorithm to provide a faster solution in large Search Spaces (the Focused Stochastic Diffusion Network - FSDN).

The basic properties of Stochastic Diffusion Search are well understood. The time complexity of the algorithm - growing at most linearly with the Search Space size - is discussed in [7]. Convergence to the global-best solution is demonstrated in [8]. A full mathematical model is developed in [9]. Earlier attempts at modelling SDS can be found in [1], [3] and [5], but the results are limited.

Apart from being a fast and reliable pattern matching algorithm, SDS seems to present itself as a possible solution to many persisting problems in neurophilosophy. In [10], the HSDN is reviewed in terms of an alternative connectionism, based on communication between neurons, as opposed to the classical approach of viewing neurons as simple computational devices.
[11] introduces NESTOR (NEural STochastic diffusion search netwORk), a connectionist implementation of SDS based on recent findings in neurobiological research. [12] expands upon biological evidence for communication as a new metaphor for neuronal operation. In [13], the same network, but with a slightly different name (NESTER), is discussed in greater detail. Simulation results show behaviour qualitatively similar to SDS. Self-synchronisation of a large population of neurons in this network (now referred to as Spiking Neuron Stochastic Diffusion Network - SNSDN) is proposed as a mechanism of *attention* in [14].

Although parallel in nature, and thus easily implemented on parallel machines or directly in hardware, most of the above applications and connectionist models have only been implemented in software on serial machines. Only one attempt ([15]) has been made so far to implement a connectionist model (SNSDN) in parallel, on a multi-processor Silicon Graphics machine.

1.2 Where to Go ?

Given the present state of research in SDS, several possible routes await exploration:

- **Application to new domains**: research applying SDS to 3D object recognition and speech recognition is ongoing. Other areas might include Protein Matching and Data Mining. Although no pattern matching problem, SDS could be modified to perform Combinatorial Optimisation, based on partial evaluation and diffusion of promising solutions.
- **SDS and the brain**: the first results using SDS-like connectionist models seem to give rise to emergent properties which could be seen as answers to neurophilosophical problems like attention and the binding problem. However, more work is needed: the present model needs to be extended, refined and simulated with greater biological inspiration. Emerging properties in simulations, bearing resemblance to higher order cognitive functions, will enable us to judge if SDS-like processes are a mechanism used by the brain.

- **Parallel implementations**: the basic SDS algorithm lends itself perfectly to parallel implementations, and in that case, will take full advantage of its sub-linear time complexity. SDS for 3D object recognition, using such computationally heavy techniques as ray tracing or other methods of rendering, might benefit enormously from implementation on a multi-processor machine or on a network of single-processor CAD-stations. Another reason for the investigation of parallel platforms is to study the dynamic behaviour of the connectionist models. Real, continuous-time, asynchronous neurons can only be approximated on a digital serial machine. Possible improvements include multi-processor machines, implementation in Field Programmable Gate Arrays or even in analog hardware.

1.3 Overview

Chapter 2 will start with a detailed description of the basic properties of SDS. Efficient parallel implementations or biologically plausible neural network models need modifications of the Standard SDS algorithm; a roadmap of these modifications will be drawn and used to categorise different SDS architectures. This categorisation will serve as a reference of what has been investigated so far, and will therefore be slightly more complete than necessary for our purpose. Partial theoretical predictions and/or simulation results will be presented and compared with Standard SDS.

Chapter 3 will embark upon the connectionist models of SDS. Recent findings in neurobiology will be explored: an overview of possible mechanisms of information encoding in the brain (e.g. temporal encoding in Inter-Spike Intervals) and local computation in dendritic arbours will provide an indication of the plausibility of the Spiking Neuron Stochastic Diffusion Network. A full description of the model with simulation results will be given. Ideas for refinements and modifications of the model will be given at the end of the chapter.

Chapter 4 will present ideas about parallel hardware on which SNSDN could be implemented. One platform (CONEMA - COmmunicating NEurons MAchine), a multi-processor machine, based on Echelon's Neuron processor which is optimised for network communication, will be built in the next half year as part of another project; specifications will be given. Another possibility using Field Programmable Gate Arrays (FPGA) will be explored: neurons could be implemented in an array of self-timed logical blocks by programming the interconnections on the FPGA. A last possibility, analog (programmable) hardware, will be briefly touched.

1.4 Expected Contributions

A short paper on Lattice SDS (see Section 2.4.7 on page 25) and the design of CONEMA (see Section 4.1 on page 52) is under preparation, for submission to 'Electronic Letters'. Work on Unlabelled (see Section 2.4.4 on page 19), Asynchronous (see Section 2.4.6 on page 24) and Lattice SDS can be combined in a full paper for the 'Journal of Parallel Algorithms and Applications'.

Part of the work on SNSDN has already been published: [14] highlights one of the emerging properties of SNSDN: self-synchronisation as a method of attention (see Section 3.3 on page 47). A short paper about qualitative similarity between SDS and SNSDN has been submitted to 'Neural Networks', and a full paper is in preparation, targeting the special issue of 'Neural Networks' on Spiking Neurons. In the next two years, more publications will be presented to neural network conferences and journals.

A website on Stochastic Diffusion Search is under preparation. Part of this report, more specifically the overview of SDS and SNSDN, will be used on the website. The site can be accessed through the CIRG research page, or directly at `www.cyber.rdg.ac.uk/~krm/sdp.htm`.

Chapter 2

Stochastic Diffusion Search

Since its advent in 1989, several modifications of the original SDS algorithm have been used for different problems. In order to categorise all these architectures, and see what the differences are, the common properties will be listed first. Then it will be discussed how some properties can differ between applications. Several modifications of the original SDS algorithm, which have been simulated and for which (partial) theoretical results exist, will be explored.

2.1 Generic Stochastic Diffusion Processes

The classification of Stochastic Diffusion Search as a subclass of pattern matching algorithms can be found in [9]. From a different viewpoint, SDS could be regarded as a subclass of a generic Stochastic Diffusion Process (SDP). The common features of such an SDP would be:

1. a set of independently operating **Agents**.
2. *partial evaluation* of possible solutions; the **testing** phase.
3. *selective communication* between Agents of solutions promising to be global-best solutions; the **diffusion** phase.
4. the **stochastic** component: which parts of the solution are to be evaluated and how selective communication between Agents is established, is governed by a probabilistic process.

Only one SDP which is not an SDS has been tested so far, a process performing Combinatorial Optimisation. Although not really belonging in this overview, it will be briefly discussed, for the sake of completeness, at the end of this chapter.

One could argue that, given such a wide definition of SDP, many other existing algorithms, not featured in this review, would fall in the same class. This topic will be left aside for a future comparative study, and concentration will be on the SDP's performing pattern matching, SDS. Before discussing abstract properties, an analogy is presented which will make the following sections (hopefully) more understandable.

2.1.1 Ant Search Analogy

This analogy was first used in [10].

Consider the following example of hypothetical ant-like creatures searching for a good nutrient source in a dynamic environment. Each ant seeks to locate some food and return it to the nest. The colony as a whole seeks to maximise the rate of return of food or to minimise expenditure of energy.

With no a-priori information on the likely location of food, each searching ant will leave the nest and perform a random walk around the local terrain. If in the course of its explorations an ant finds some food, it returns to the nest a positive ant; otherwise it is labelled negative.

On its return to the nest, each positive ant simply tells the first searching ant it meets the location of its find. If the food source is good (i.e. it is temporally stable and bountiful), over a relatively small period of time the nest will allocate more and more of its resources (ants) to exploiting it. Whereas if the resource is poor, any positive ants that are initially attracted to it will sooner or later not find anything and revert back to being searching ants. Conversely since unsuccessful ants which meet on their return to the nest do not exchange resource-location information, they simply re-commence their random search.

Although meant as an analogy, many ant species do exhibit similar recruitment behaviour ([16]). A very recent publication in Nature ([17]) uses recruitment in a foraging task for a swarm of small mobile robots; results indicate that the strategy is superior to foraging without recruitment.

2.2 Basic properties of SDS algorithms

All the Stochastic Diffusion Search algorithms used so far have the following properties in common:

1. a set of **Agents** performing the search independently.

2. a target pattern, often called the **Model**. The target pattern is defined by a set of **micro-features**. The exact nature of the micro-features is irrelevant, as long as they provide sufficient characterisation of the Model.

3. a **Search Space**, consisting of micro-features of the same nature as the ones defining the Model.

4. Agents contain a **mapping** (location pointer) into the Search Space and have different **internal states** of operation. Agents in some states of operation will be more likely to point to an instantiation of the Model (a part of the Search Space which has many micro-features in common with the Model) than to background or noise, and will try to attract other Agents to explore the same location in the Search Space.

5. an **initialisation** phase

6. a **diffusion** phase, in which Agents can selectively communicate with other Agents. The goal of the diffusion phase is to generate new random Search Space locations to be searched, or to communicate promising locations to other Agents. The process depends on the Agent's own state and/or information from other Agents.

7. a **testing** phase, in which micro-features of the Model are compared with the corresponding micro-features of the Search Space, determined by an Agent's location pointer.

8. for practical purposes, a **termination** criterion is required.

9. the **stochastic** component: which micro-features of the Model are to be evaluated, which new locations in the Search Space will be investigated and how selective communication between Agents is established, is governed by a probabilistic process.

This class of algorithms will, in a probabilistic sense, **explore** the entire Search Space. They will also, as a result of the competition between Agents to attract other Agents, allocate resources in a *dynamical* way to regions of the Search Space having many micro-features in common with the Model - **exploitation** of the Search Space. The dynamical behaviour of a specific algorithm, and its balance between exploration and exploitation of the Search Space, is dependent on the details of its implementation. A detailed resource allocation analysis for some SDS algorithms has been performed in [9].

A dynamically changing, but stable (over some period of time) population of Agents evaluating a position in the Search Space ('converging onto'), will be interpreted as a solution to the pattern matching problem. Convergence of the Agents onto the global-best fit of the Model in the Search Space has been demonstrated for certain cases of Stochastic Diffusion Search ([8]).

2.3 Filling in the Details

2.3.1 The Timing Behaviour

Time it takes to update an Agent's state

Update of an Agent's state will usually consist of a test and/or a diffusion phase (in any order). During such an update, micro-features of Model and Search Space can be compared, and communication between Agents might take place. Under certain conditions, this update could be considered to occur instantaneously; whereas in other cases, as will be seen during discussion of the connectionist models using temporal encoding of information in Inter-Spike Intervals, this update process has a physical dimension of time. This changes the dynamics of the system, one iteration of one algorithm could constitute many iterations in another. Moreover, the temporal extent of the process could be used in directing the process itself (see Chapter 3). Most - if not all - of the other available alternatives can be combined with these two different modes of operation.

Mode of operation

Agents can operate synchronously or asynchronously. In synchronous mode, all Agents update their state at the same moment in time, or over the same period of time. At least two different modes of asynchronous operation exist:

1. At each iteration, one Agent updates its state at random. It is specifically suited for software simulations on digital serial computers, and is easier to analyse than other operational modes.

2. Each Agent has a certain probability distribution over time to update its state, independent from other Agents. Although it can be implemented in software, this asynchronous mode is specifically suited for hardware implementation. When the process is continuous in time, it can be seen as a limiting case of the first definition, since the chance that two Agents will update their states simultaneously is negligible.

However, even in discrete time, the first definition can be seen as an approximation to the second.

2.3.2 Internal States of Agents

Number of states

In all SDS algorithms used so far, Agents have two internal states, **active** and **inactive**. Active Agents are more likely to point to an instantiation of the Model in the Search Space than inactive Agents, and will try to attract other Agents to explore the same location in the Search Space.
The number of states is not limited to two though. More levels of activity (reflecting the likeliness to point to instantiations of the Model) could be possible; activity could even be a continuous variable.

Knowledge of internal states of other Agents

Two possibilities : Agents could have access to another Agent's internal state, or could have no explicit access. Both possibilities have been tested out and show slightly different dynamical behaviour. Examples will be given in the next section.

2.3.3 The Search Space

The nature of the micro-features

The symbolic representation of the problem will in general not be influencing the resource allocation of the Agents, other than through the number of micro-features (the alphabet size). What it does change is the computations during the testing phase, and how the Model is mapped onto a certain position in the Search Space by the Agent's location pointer. In this indirect way, the representation will influence total computation time, but not the convergence or equilibrium behaviour, which is general and feature-independent.

In one-dimensional string matching, Model and Search Space are strings, the individual characters of the string being the micro-features. The mapping (location) of the Model into the Search Space will then be a single integer, denoting the position of the Model in the Search Space. In the HSDN ([3]), two-dimensional feature recognition, micro-features are RAM cells, and the mapping of the Model into the Search Space consists of X and Y translation parameters, and a rotational parameter, thus constituting a search with 3 degrees of freedom. In 3D object recognition, the degrees of freedom would be 3 translational and 3 rotational parameters, while the test phase would typically be a rendering of (parts) of a 3D scene.

Strategies to limit the size of the Search Space

Convergence time (the time it takes Agents, evaluating an instantiation of the Model in the Search Space, to form a stable population) is dependent on the size of the Search Space. It would therefore be helpful, for very large Search Spaces, to limit the Search Space Size. In FSDN ([6]), a more and more detailed search in hierarchically organised search spaces speeds up the convergence process. No other such ‘heuristics’ have been tried so far.

Boundary conditions

Under certain conditions, partial matches to the Model at both edges of the Search Space could remain undetected. An example comes from one-dimensional string matching: consider the Model to be ’blah’, and the Search Space to be ’lahzzzblahzzzbla’. The best match to the Model starts at location 6 in the Search Space, but partial matches start at location 0 and location 13.
Several possibilities exist: one could completely disregard these partial matches, being only interested in the middle part of the Search Space. Other possibility: a number of Search Space locations (in this case, $size(Model)-1$) could be added at the beginning and the end of the Search Space, and filled with a dummy value which will always result in the Agent failing the test phase. Last possibility (and preferable to the previous) is to allow the location pointers of the Agents to point to non-existing locations in the Search Space. If micro-features at these non-existing locations are to be evaluated, then this would result in an Agent failing the test. For instance, if we want the partial match ’lah’, starting at position 0 of the Search Space to be detected, we will have to allow for negative integers as location pointers (-1), and make sure that in that case the testing of the first micro-feature of the model (the character ’b’) will result in the Agent failing the test.

2.3.4 The Order of Test and Diffusion

The actual order of test and diffusion phase is quite irrelevant. Initialisation sometimes determines that the algorithm should start with a test phase; in other cases, programming elegance makes starting with a diffusion phase preferable. Any difference in dynamical behaviour can hardly be noticed.

In other implementations, the sequence of test and diffusion phases is made probabilistic or dependent on the internal state.

2.3.5 Initialising

At the start of the algorithm, the internal state of the Agents will be initialised to inactive. If a-priori knowledge about the position of the Model in the Search Space is available, then this knowledge can be used to initialise the location pointers ([6]). The algorithm will then have to start with a test phase. If no a-priori knowledge is available, then the location pointers could be initialised to random values, and the algorithm starts with a test phase. Instead, the location pointers could remain undefined; the algorithm would then have to start in the diffusion phase.

2.3.6 Testing

During the testing phase, micro-features from the Model are compared with micro-features from the Search Space, and dependent on the outcome of the comparison, the activity level of the Agent is changed. Usually only 1 randomly chosen micro-feature of the Model will be compared to the corresponding micro-feature in the Search Space.

2.3.7 Diffusing

Two- or one-way communication

Communication between Agents can be a two- or a one-way process. An Agent could pick out another Agent at random, by sending a request to get its location pointer and/or internal state. Alternatively, Agents could just broadcast any information to a number of other Agents at certain moments in time, leaving it up to the receiving Agents to use the information or not.

Agents communicating during diffusion

In Standard SDS, only inactive Agents will change their states and/or location pointers during the diffusion phase. In Context-Free SDS and Context-Sensitive SDS([9]), active Agents will also react upon information from another Agent in the population, and possibly change their internal states. These two alternatives to the standard diffusion phase shift the balance from exploitation towards more exploration of the Search Space. Note that not all of these schemes are compatible with one-way communication implementations or without knowledge of the internal state of the other Agent.

Information exchange between other combinations of active/inactive Agents might result in different diffusion phases, but has not been tried out.

2.3.8 Terminating

For practical purposes, a termination criterium is needed. One possibility (used mainly when investigating the dynamic behaviour of a particular algorithm) is to run the algorithm for a fixed number of iterations. In practical implementations, the algorithm has to stop when it has located the Model in the Search Space. Testing for stability of a population evaluating a position in the Search Space, or testing for the stability of the overall activity, are two possibilities. More information can be found in [3], [8] and [9].

2.3.9 Probability Distributions

Micro-features

Any micro-feature to be evaluated during testing, may be chosen at random with an equal probability.

New Search Space locations

A new Search Space location, adopted by an Agent during diffusion, will usually by chosen at random with an equal probability. Some heuristical methods or a-priori information could be applied, like in [6].

Communication between Agents

In Standard SDS, Agents can communicate with all the other Agents; communication being governed by a uniform probability distribution. Such a probability distribution could also impose a neighbourhood structure on the Agents, limiting the number of Agents for communication. Self-selection, the fact that an Agent can pick out itself during diffusion, will also slightly influence the dynamical behaviour of the network.

Other steps which can be made probabilistic

The above probability distributions can also be changed in an indirect way. When the timing behaviour is not a passive property of the process, but is incorporated as an active part of the computational algorithm, then probability distributions will be dependent on the specific timing details. See the

SNSDN model, Chapter 3 for an example. Other steps can be made probabilistic, e.g. the internal state change after failing a test (as in 'the secret optimist', proposed in [5]); the chance of undergoing a diffusion phase (as in 'the hermit', also proposed in [5]). Most of these modifications will shift the balance between exploration and exploitation, and their usefulness depends on the particular search problem.

2.4 Overview of Implemented SDS Algorithms

In the previous sections, the basic common properties of SDS algorithms were covered, and a number of different ways of filling in the details were discussed. In this section, focus will be on the algorithms which have actually been implemented. A roadmap will be made up, moving the original Standard SDS to one which can, on the one hand, be implemented in hardware more efficiently, and on the other hand, is more biologically plausible. It will be assumed that all algorithms in this section update states of Agents instantaneously, and covering of specific issues related to temporal encoding will be postponed to the SNSDN model of Chapter 3. The search problem at hand will be one-dimensional string matching. Micro-features and locations of the Model in the Search Space can then be defined unambiguously; other search problems would introduce additional complications and divert us from the basic question, namely, how do the algorithms compare in terms of convergence time and resource allocation ?

2.4.1 Conventions

To make comparison easier, the naming conventions of [9] will be taken over:

- the number of Agents is denoted by N.
- let the Search Space size (measured as a number of possible mappings of objects) be M.
- p_m is the probability of locating the best fit of the Model in a uniformly random draw.
- p_d is the probability of locating a sub-optimal pattern (one sharing to some extent micro-features with the Model).
- the probability of a false positive test is p^+, and of a false negative test is p^-.

2.4.2 Standard Stochastic Diffusion Search

In Standard SDS, introduced in [1] and [2], and analysed in [3], [7], [8] and [9], Agents operate in synchrony. The algorithm involves several stages:

1. **Initialise Agents:** Agents are assigned random locations in the Search Space, and their state is initialised to *inactive*.
2. **Test:** A randomly selected character of the Model is compared to the character of the Search Space, obtained by taking the location pointer of the Agent and adding the offset of the character in the Model string. If the test succeeds, then the internal state is set to *active*; else, the state is set to *inactive*.
3. **Diffuse:** Every *inactive* Agent chooses another Agent at random. If the other Agent is *active*, then its location in the Search Space is copied. If the other Agent is *inactive*, a new, random location in the Search Space is adopted.
4. **Termination:** If the termination criterium is fulfilled, terminate. Else go back to Step 2.

It can easily be seen that Agents have access to the internal state of other Agents; communication is a two-way process. All the probability distributions used for randomising the process are uniformly distributed.

Since this is the oldest and best-understood form of Stochastic Diffusion Search, subsequent modifications will be compared with this algorithm, which will be referred to as Standard SDS.

Context-Free and Context-Sensitive SDS

In [9], two modifications of Standard SDS were introduced. The operation is similar to Standard SDS, apart from the diffusion phase for *active* Agents. Resource allocation analysis can be found in [9].

- **Context-Free diffusion phase:** *inactive* Agents perform the same operation as in Standard SDS. *Active* Agents choose another Agent at random. If the other Agent is also *active*, the Agent becomes *inactive* and a new random location in the Search Space is adopted. If the other Agent is *inactive*, then the Agent remains *active*.

- **Context-Sensitive diffusion phase:** *inactive* Agents perform the same operation as in Standard SDS. *Active* Agents choose another Agent at random. If the other Agent has the same location pointer, the Agent becomes *inactive* and a new random location in the Search Space is adopted. If the other Agent has a different location pointer, then the Agent remains *active*.

2.4.3 From Software Implementations to Hardware

In order to move from the standard algorithm to one that can be implemented in hardware and/or is more biologically plausible, the effects of following modifications need to be investigated:

1. The internal state of an Agent is not available to other Agents.
2. Asynchronous mode of operation.
3. Local connectivity of Agents. Agents can only communicate with other Agents in their neighbourhood.
4. A one-way communication protocol.

2.4.4 Unlabelled SDS

Unlabelled Stochastic Diffusion Search was an attempt to propose an algorithm as close as possible to Standard SDS, but without the internal states (activity label) of Agents accessible to other Agents. Operation is similar to Standard SDS, apart from the diffusion phase. During diffusion, each *inactive* Agent generates a new location in the Search Space at random, then chooses another Agent at random and copies the location pointer. If the chosen Agent was active, the location pointer is likely to be pointing to an instantiation of the Model; if the chosen Agent was inactive, then the Agent actually adopts a randomly generated search space location in the above described two-step process. An Agent does not need access to the activity state of the other Agent, while, on first sight, still sampling from the same probability distributions as in Standard SDS. The difference is very subtle though.

Suppose one (possibly imperfect) instantiation of the Model is present in the Search Space, with no additional noise in the Search Space. This is the case studied in Chapter 4 of [9] where $p_m = 1/M$ and $p_d = p^+ = 0$. Two statements can be proved:

1. The probability for a single inactive Agent to get the correct location value during a diffusion phase is the same for both Standard and Unlabelled SDS.
2. The probability distribution of the total number of inactive Agents which get the correct location value during a synchronous diffusion phase is *not* equal to the distribution for Standard SDS.

Probability for a single Agent of getting the correct location

For Standard SDS, the probability for an inactive Agent to get the correct location during diffusion, is given by ([9]):

$$p = \frac{a}{N} + (1 - \frac{a}{N})p_m$$

where a is the number of active Agents.

The first term gives the probability of picking out an active Agent during diffusion; the second term gives the probability of picking out an inactive Agent followed by the probability of 'guessing' the correct location.

For Unlabelled SDS, this probability looks somewhat different:

$$p = \sum_{k=0}^{N-a} \binom{N-a}{k} p_m^k (1 - p_m)^{(N-a-k)} \frac{a+k}{N}$$

Every term in this sum gives the probability that k out of $N - a$ inactive Agents 'guess' the correct location value during the first step of the diffusion phase, followed by the probability that one of the $a+k$ Agents with the correct location value will be picked out during the second step of the diffusion phase.

Looking very different at first sight, both probabilities are actually the same. The proof of this is rather long and will be summarised here in a few steps :

1. substitute $(1 - p_m)^{(N-a-k)}$ by its sum-of-powers form
2. re-order and re-write everything in ascending powers of p_m
3. (a) the zeroth order term of the sum is equal to $\frac{a}{N}$
   (b) the first order term of the sum is equal to $(1 - \frac{a}{N})$
   (c) all the higher order terms are zero

Probability distribution of getting the correct location

At the startup of Standard SDS, when all Agents are inactive, location values are chosen by N uniformly random draws of potential positions from the Search Space (in subsequent diffusion phases, this random draw is only performed by the inactive Agents that picked out another inactive Agent). The probability of getting the correct location k times in this procedure is given by:

$$p[X_{n+1} = k|X_n = 0] = \binom{N}{k} p_m^k (1 - p_m)^{N-k}$$

At the startup of Unlabelled SDS, or whenever no active Agents are biasing the process, the probability is given by:

$$p[X_{n+1} = k|X_n = 0] = \sum_{i=0}^{N} \binom{N}{i} p_m^i (1 - p_m)^{N-i} \binom{N}{k} \frac{i}{N}^k (1 - \frac{i}{N})^{N-k}$$

Every term in this sum gives the probability that i out of N Agents guess the correct location during the first step of the diffusion, followed by the probability that k out of N Agents pick up the correct location during the second step of the diffusion. That these two distributions are not the same can be easily seen for $k = 0$: the first term of the sum for Unlabelled SDS is equal to the transition probability $p[X_{n+1} = 0|X_n = 0]$ in the case of Standard SDS. This can be interpreted as follows: multiple paths to get $p[X_{n+1} = 0|X_n = 0]$ exist: 0 correct locations can be guessed in the first step of diffusion, with the obvious result that 0 correct locations will survive the second step. However, any number of correct locations, generated during the first step, can be lost in the second step, thus increasing the probability of having no correct locations at the end of the diffusion phase. On the other hand, the effect of guessing some correct locations in the first step can be enhanced in the second.

The general case of having k Agents with the correct location after the diffusion phase, given a Agents with the correct location before the diffusion phase, is governed by the probabilities:

$$p[X_{n+1} = k|X_n = a] = \binom{N-a}{k-a} p_1^{k-a} (1 - p_1)^{N-k}$$

for Standard SDS, with $p_1 = \frac{a}{N} + (1 - \frac{a}{N})p_m$; and by:

$$p[X_{n+1} = k|X_n = a] = \sum_{i=0}^{N-a} \binom{N-a}{i} p_m^i (1-p_m)^{N-a-i} \binom{N-a}{k-a} (\frac{i+a}{N})^{k-a} (1-\frac{i+a}{N})^{N-k}$$

for Unlabelled SDS. Since in a diffusion phase, Agents do not loose the correct location pointer, it is clear that $k \geq a$.

The above probability distributions are quite different from each other for small Search Space sizes (high p_m) and for a small number of active Agents. For larger Search Spaces, the difference is less pronounced. The effect of the number of active Agents can be seen in Figure 2.1. The two probability distributions for 4 different numbers of a are plotted.

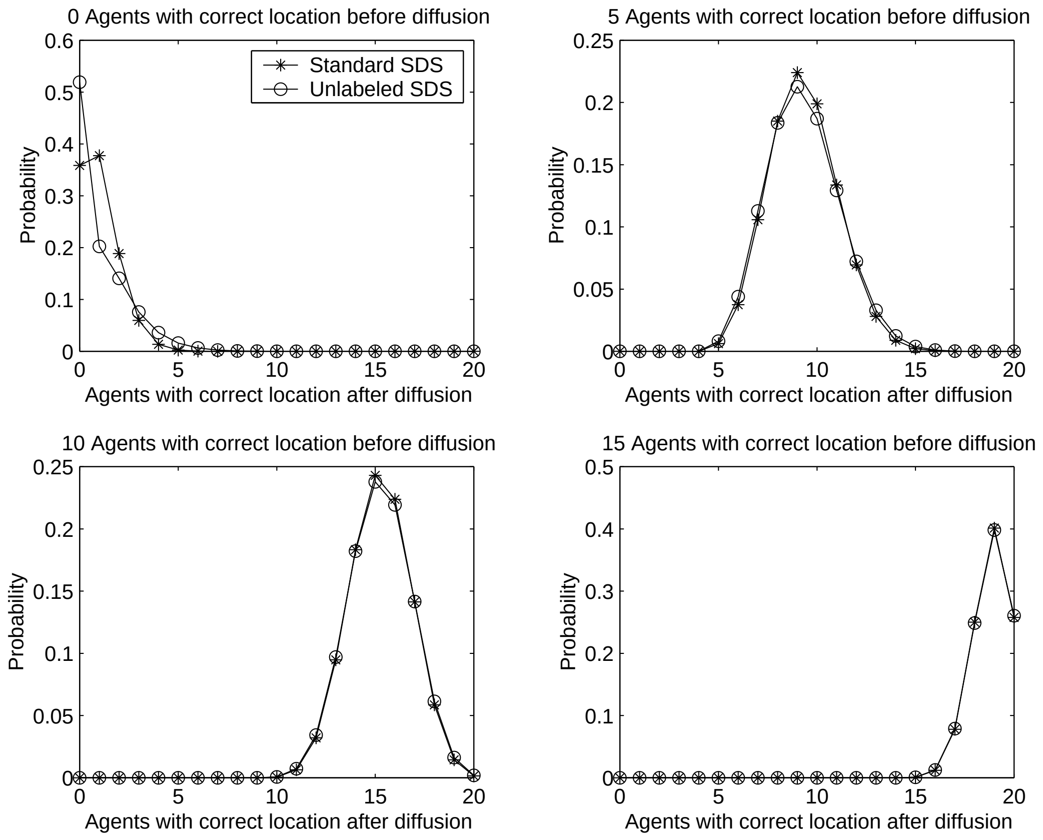

Figure 2.1: Probabilities of having k number of Agents with the correct location pointer *after* diffusion, given a certain number *before* diffusion. There are 20 Agents in total, and the number of possible locations in the Search Space is 20. With 0 Agents before diffusion, the probabilities for Standard and Unlabelled SDS differ considerably. The difference becomes smaller for 5 and 10 Agents, and can hardly be noticed for 15 Agents.

Comparison of Standard and Unlabelled SDS

The difference discussed above forms an explanation for the result in Figure 2.2. Convergence is somewhat slower for Unlabelled SDS, but the equilibrium populations are equal in size, since a high number of active Agents diminishes the difference in the diffusion phase.

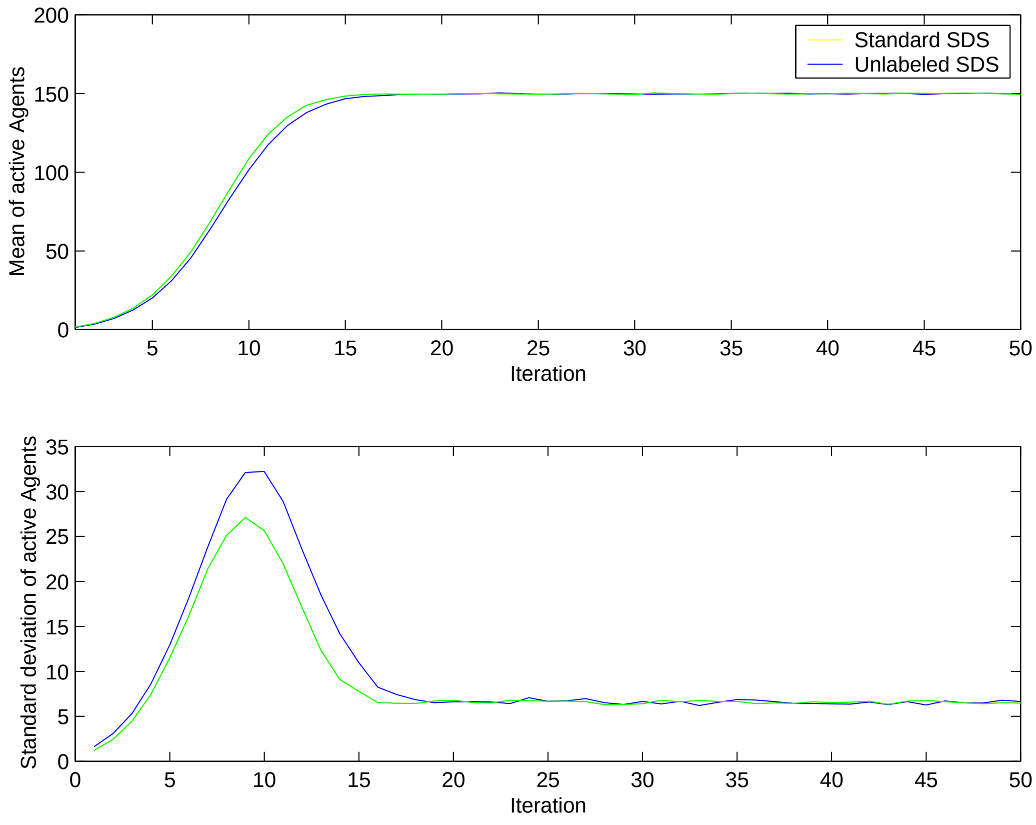

Figure 2.2: Mean convergence behaviour of 500 runs of Standard and Unlabelled SDS. Each run lasted 50 iterations. $N = 200$, $M = 104$ and $p^- = 0.2$. Standard SDS converges slightly faster to an equilibrium state, and its standard deviation over the 500 runs is smaller during the convergence period.

Conclusion

Standard and Unlabelled SDS differ only in their diffusion phases. The altered mechanism of Unlabelled SDS makes knowledge of the internal state of another Agent unnecessary, while keeping quantitative difference to a minimum. Unlabelled SDS has slightly lower convergence speed, but the equilibrium population is similar to Standard SDS. A Context-Free Unlabelled SDS is impossible by definition, but Context-Sensitive SDS could be implemented.

2.4.5 Spiking Neuron SDS

Another SDS algorithm in which Agents do not need knowledge of the internal state of other Agents, is the search algorithm used in NESTER ([13]). Its operation is asynchronous and state updates have a physical dimension of time. Communication is a one-way process and timing of neuronal spikes is used as a randomisation method. Labels active and inactive are used in a slightly different way than in Standard SDS. An in-depth description will be given in section 3.2 on page 43.

2.4.6 Asynchronous SDS

An intermediate step on the Road to Freedom is the study of asynchronous SDS. The first version of asynchronous (see definition on page 12) Standard SDS was simulated. If the random distribution for selecting the Agent is uniform, then results for synchronous and asynchronous SDS will be exactly the same, apart from the time scale on which they operate. In synchronous Standard SDS, N Agents update their state in one iteration, whereas in asynchronous (Standard) SDS, only one Agent updates its state in an iteration. N iterations of asynchronous SDS will then constitute, in a probabilistic sense, one iteration of synchronous SDS.
Several reasons exist for taking this step: the process can be modelled by a very simple Markov chain; it is actually a one-dimensional random walk, and in some cases, an exact formulation of the expected value of the convergence time becomes possible. Besides this, the process also gives an indication of the behaviour of the second mode of asynchronous operation ('hardware asynchronous'), which can be considered as its continuous time equivalent. It is also very easy to consider the effect on the convergence time of the possibility of self-selection during diffusion.

Markov chain model for asynchronous Standard SDS

The one step evolution of an Agent is quite similar to the one step evolution of an Agent in synchronous SDS. Concentrating on the case where one perfect instantiation of the Model exists in the Search Space, and without additional noise ($p^- = 0$, $p^+ = 0$ and $p_d = 0$) the process becomes a one-dimensional, one-directional random walk (total activity of the Agents can only increase, and by not more than one unit per iteration). The transition probability of the active Agent population is given by:

$$p[X_{n+1} = a+1|X_n = a] = \frac{N-a}{N}(\frac{a}{N} + (1-\frac{a}{N})p_m)$$

with the possibility of self-selection of an Agent during diffusion, and by:

$$p[X_{n+1} = a+1|X_n = a] = \frac{N-a}{N}(\frac{a}{N-1} + (1-\frac{a}{N-1})p_m)$$

without self-selection.

It has been demonstrated in [3] and [8] that if $p^- = 0$, all Agents in synchronous Standard SDS will converge upon the perfect instantiation of the Model with probability 1. The same can be proven for the asynchronous version; moreover, it becomes very easy to compute the expected value of the number of iterations required for all Agents to become active (convergence time TC):

$$E[TC] = \sum_{a=0}^{N-1} \frac{N^2}{(N-a)(a+(N-a)p_m)} \tag{2.1}$$

$$E[TC] = \sum_{a=0}^{N-1} \frac{N(N-1)}{(N-a)(a+(N-1-a)p_m)} \tag{2.2}$$

with and without self-selection respectively. The difference between both choices is not very large, except for a small number of Agents. It will be shown later that in case of a restricted neighbourhood for Agent communication (Lattice SDS, see next section), self-selection slows down convergence considerably.

Conclusion

Moving from synchronous to 'first definition' asynchronous mode of operation does not have an effect on the resource allocation of SDS. It can be expected that a move to continuous time asynchrony retains the properties, as it is a limiting case of the discrete time asynchronous mode. The effect of self-selection on convergence speed is negligible, except for a small number of Agents.

2.4.7 Lattice SDS

In Lattice SDS, the probability distribution determining communication between Agents defines a neighbourhood structure over the set of Agents. Analysis of this kind of process as a Markov chain is hard: the process is not determined, as in the previous cases, by one or two integers denoting the number of active Agents, but by the exact location of active and inactive Agents. The number of states of the Markov chain is given by 2^N, which makes exact formulation of transition probabilities impossible for anything but a very small number of Agents N. Many of these states are equivalent, so aggregation of states could be performed. However, determining which states belong to which class, and how many distinct classes exist, is non-trivial.

For resource allocation analysis of the equilibrium population, the Ising model - or a formulation as the more general class of Markov Random Fields - might prove to be a valuable tool. For large numbers of Agents, continuous diffusion processes could give an approximation for convergence speed. Later on in this section, an approximative method to determine convergence time for a small number of Agents, based on graph theory, will be presented.

The main reason for taking the step from global to local interconnections is implementation in hardware. For instance, CONEMA (see section 4.1) will consist of 64 processors, each with a maximum of six interconnections to neighbouring processors. Attention will therefore be focused on a small number of Agents and a small number of connections, with the main goal of predicting how such a machine, performing SDS, will compare to Standard SDS.

The following analysis will consider the asynchronous operation of Lattice SDS. This in an attempt to simplify the analysis. Results should apply to a synchronous mode of operation as well, as explained in previous section.

Convergence time will be studied in the case of one perfect instantiation in the Search Space, and no additional noise ($p^- = 0$, $p^+ = 0$ and $p_d = 0$). Some experimental results concerning resource allocation (with $p^- \neq 0$) will be presented, but no theoretical analysis has been performed so far.

Lattices under scrutiny

Lattices with 16, 32 and 64 Agents are investigated. Most of the layouts are nearest-neighbour lattices: Agents are only connected to the nearest neighbours. Boundary conditions are periodic and self-selection is not allowed. The links are bi-directional: they allow two-way communication.

Other layouts with 16 Agents are tested as well. One is the proposed PCB layout for CONEMA. The second is the (probably) the layout with minimal inter-agent distance, for 16 Agents with 4 neighbours each. Links are bi-directional. A schematic of the layouts can be seen in Figure 2.3.

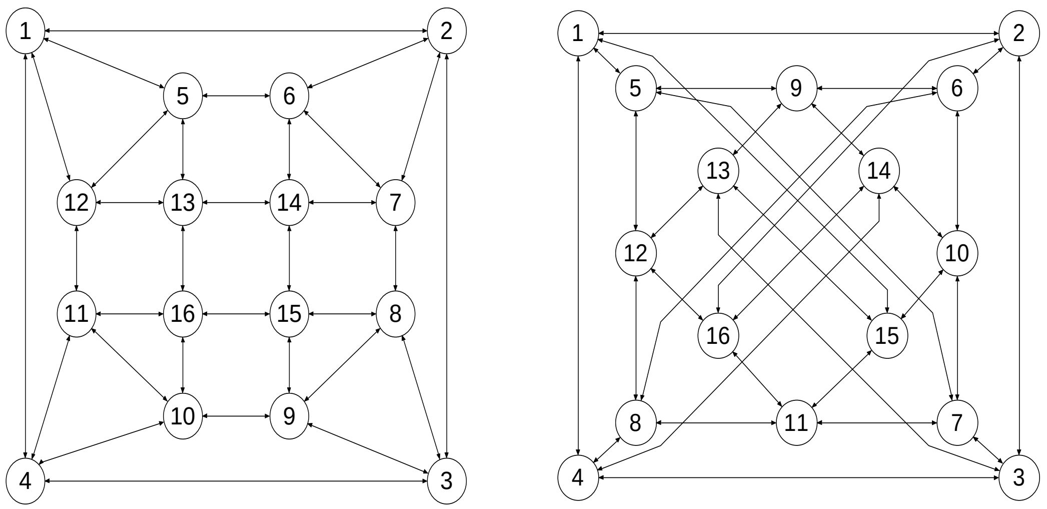

Figure 2.3: Two special lattices with 16 Agents each. Each Agent has four connections. Left is the proposed layout for the PCB's of CONEMA: it is easier to implement than the genuine nearest-neighbour layout, since no connection lines are crossing. Right is the (probably) optimal layout for 16 Agents with 4 neighbours each.

Time to hit and diffusion time

[1] introduced the terms 'time to hit' (TH) and 'diffusion time' (TD) in the analysis of convergence time of Standard SDS. TH is the time it takes for at least one Agent of the entire population to 'guess' the correct location in the Search Space. TD is the time it takes for one correct location to spread across the population of Agents. Although this analysis gives a crude approximation for the convergence time, and a better model has been developed (see [7]), two reasons exist for revising this method:

1. In case of a large Search Space and a small number of Agents N, $TH \gg TD$. Since in that case, the number of timesteps between two hits will be much larger than the diffusion time, there will be, in the average convergence case, on hit which will spread across the entire population

before another hit occurs; convergence time can then be approximated as:

$$E[TC] \approx E[TH] + E[TD]$$

And it forms an upper limit for $E[TC]$ when the Search Space size is smaller than or comparable to the number of Agents.

2. The expected number of iterations for TH, in the asynchronous case, is simply the Search Space size:

$$E[TH] = M$$

Connectivity is irrelevant until the first hit. To simplify analysis, TH will be left out, and analysis and experiments will concern the case where one random Agent is initialised with the correct location, and $p_m = 0$ for all other Agents.

Solved cases of $E[TD]$

For some lattices, the Markov chain for the pure diffusion process can be constructed, and the expected number of iterations before the initially present correct location has spread across the network, can be computed.

1. Every Agent in the lattice is connected with every other Agent. This is the pure diffusion process for asynchronous SDS described in the previous section, and the expected value for TD can be obtained by putting $p_m = 0$ in Equation 2.1 or 2.2. In case self-selection is not allowed, this results in:

$$E[TD] = \sum_{a=1}^{N-1} \frac{N(N-1)}{(N-a)a}$$

2. A one-dimensional lattice, where each Agent has only one neighbour (e.g. the left one) where it can get the correct location from. Boundary conditions are periodic ('ring topology'). The expected values for TD, with and without self-selection, are given by:

$$E[TD] = 2N(N-1)$$

$$E[TD] = N(N-1)$$

3. A one-dimensional lattice, where each Agent has two neighbours, one to the left and one to the right, and with periodic boundary conditions. Expected values are, with and without self-selection:

$$E[TD] = \frac{3}{2}N(N-1) \tag{2.3}$$

$$E[TD] = N(N-1) \tag{2.4}$$

Diffusion time for other lattices is not easily predicted. Although the number of allowed transitions is considerably lower than in the general convergence case, the number of states for the Markov chain is again 2^N. Some of the states have much higher probabilities to be visited than other states, but determining these states is not straightforward in Markov chain theory.

Some interesting conclusions can be drawn from the solved models: firstly, self-selection has a strong negative effect when the number of neighbours is small. Secondly, the linear lattices seem to be forming the worst-case scenario, with a time-complexity $O(N^2)$ in function of the lattice size, on a serial machine. The full-interconnected lattice is the optimal case, with a sub-quadratic time complexity on a serial machine ($O(N \log_{1.59} N)$).

Approximate method based on graph theory

A method, based on graph theory, will allow to predict an upper limit for $E[TD]$, given the lattice connections. Before explaining the method, some basic terminology from graph and network theory is defined ([18]):

Definition 1 *A **graph** G consists of a set of elements (**vertices**), and a list of* unordered *pairs of these elements (**edges**). A directed graph (**digraph**) has a set of vertices, and a set of* ordered *pairs of vertices (**arcs**). A **subgraph** of G has as vertices a subset of the vertices of G, and as edges a subset of the edges of G. Two graphs G and H are **isomorphic** if H can be obtained from G by relabelling the vertices. A **network** is a graph where each edge has a cost or distance associated to it. The number of edges **incident** with a vertex is called the **degree** of the vertex. If every vertex in the graph has the same degree r, then the graph is **regular of degree r**. A **path** between vertices x and y is a list of edges or arcs leading from vertex x to vertex y, visiting every edge (or arc) and every vertex not more than once.*

It is clear that a lattice of SDS Agents forms a graph, with Agents as vertices and connections as edges, if all the connections between Agents allow bi-directional communication. It forms a digraph if connections are one-way.

Furthermore, define the **distance** between vertices as the minimum path length between vertices, given that the path length between vertices which are connected by an edge, is 1.

The diffusion process can be reformulated as a particle travelling from one vertex to another on the graph. Given a starting vertex, it is the maximum distance in the graph to other vertices which will influence the time of the journey. Not only the maximum distance in the graph, but also the number of paths connecting the starting vertex with the most distant vertices is important, as it increases the chance of getting the particle from the starting vertex to the most distant vertices.
As an approximation, the paths with length greater than the distance between the vertices will be discarded. This corresponds to discarding the improbable states in the Markov chain; which was not straightforward in the Markov chain itself, but becomes an easy task in the graph. The following algorithm will provide an upper limit for the diffusion time TD, since, as a byproduct, it will only consider the most probable states of the Markov chain:

1. Determine all the different classes of vertices. Vertices belong to the same class if the subgraphs, containing all the shortest paths between the vertex and its most distant colleagues, are isomorphic.

2. Take a starting vertex from each class of vertices. Take as a subgraph H from the original lattice G, the digraph containing all the shortest paths between the starting vertex and the vertex which is most distant; if there is more than one most distant vertex, repeat the process for all different classes of most distant vertices.

3. Transform the digraph H to a network by labelling each arc with a travel time. The travel time of an arc is the expected number of iterations for an Agent X to become active, given the Agents from which arcs from H arrive in X , are active.

4. Compute the travel times for all shortest paths. Average over all paths.

5. Average over all different classes of most distant vertices.

6. Average over all different classes of starting vertices.

Example1 The network, derived from an 8 by 1 linear lattice, with two nearest neighbours and with periodic boundary conditions, is shown in Figure 2.4. All vertices in the network belong to the same class, i.e. the maximum distance in the graph is the same starting from each vertex , and subgraphs connecting vertices which are most distant are all isomorphic to each other. When choosing vertex 1 as the starting point, the most distant vertex is vertex 5. Two paths connect vertices 1 and 5: $(1, 2, 3, 4, 5)$ and $(1, 8, 7, 6, 5)$. Arc $(1, 2)$ has a travel time $2N$ associated to it; this travel time is the expected number of iterations it will take for Agent 2 to become active, given Agent 1 is active. Arcs (4,5) and (6,5) have an associated travel time N; this is the expected number of iterations for Agent 5 to become active, given Agent 4 and Agent 6 are active at iteration 0. The total travel time for both paths is $7N$; this is the value predicted by Equation 2.4. The approximate method gives the exact prediction, because the shortest paths connecting vertices 1 and 5, are the *only* paths in this case; there are no non-shortest paths in the one-dimensional lattice.

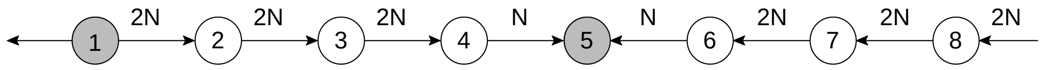

Figure 2.4: Network containing all shortest paths from vertex 1 to vertex 5. The network is derived from an 8 by 1 linear lattice, with two nearest neighbours, and with periodic boundary conditions.

Example2 The network in Figure 2.5 is derived from an 8 by 2 lattice; each vertex has three nearest neighbours and boundary conditions are periodic. All vertices belong to the same class. The largest distance in the network is 5, for instance from vertex 1 to vertex 13. There are 10 paths from vertex 1 to vertex 13. Arc (1,2) has a travel time of $3N$, because there is only 1 arc arriving in vertex 2. This is the expected number of iterations it takes for Agent 2 to become active, given Agent 1 is active at iteration 0. Vertex 12 has two arriving arcs, (4,12) and (11,12); they are labelled with a travel time of $\frac{3N}{2}$, because that is the expected time for Agent 12 to become active, given Agents 4 and 11 are active at time 0.
Four paths exist which have a travel time of $8.5N$, two paths have a travel time of $10N$ and the last four paths have a travel time of $11.5N$. Averaging over all these paths gives a travel time of $10N$, which gives $E[TD] = 160$. The experimental value from Table 2.1 is 149 iterations, which is very close to the predicted value. The difference is caused by the contribution of the

non-shortest paths in the lattice, e.g. (1,2,3,4,5,6,14,13), which are not allowed in the network approximation (because of the direction of arc (6,5)). Discarding these paths is similar to discarding the less probable transitions in the Markov chain.

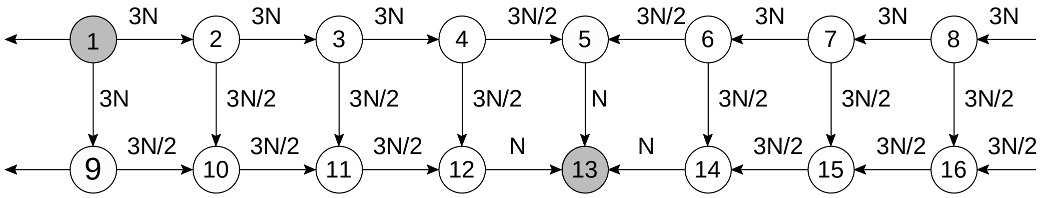

Figure 2.5: Network containing all shortest paths from vertex 1 to vertex 13. The network is derived from an 8 by 2 lattice, with three nearest neighbours, and with periodic boundary conditions.

More examples In a four by four lattice with four nearest neighbours each and periodic boundary conditions, all the vertices belong to the same class. The maximum distance in the lattice is 4, and there are 24 paths leading from a starting vertex to the most distant one. Each path has a travel time of $8.33N$, which gives a predicted TD of 133 - as opposed to the experimental value of 117.
The lattice in Figure 2.3a has a maximum distance of 4 starting from all vertices. However, there are two different classes of vertices: vertices 1 to 4 and 13 to 16 have 16 shortest paths between most distant nodes; vertices 5 to 12 have 10 shortest paths. The predicted value for TD is 155; experiments give $E[TD] = 127$. The difference here is quite large; the reason is the existence of quite probable non-shortest paths; e.g. paths of length 5 exist, which is not the case for the 4 nearest-neighbour model, where, apart from the shortest paths, only paths of length 6 and more exist.
The layout in Figure 2.3b is probably the optimal case for a lattice of 16 Agents with 4 neighbours each. In the optimal lattice, each Agent is at a distance 1 from 4 other Agents, and at a distance 2 from all other 11 Agents. Whether such a lattice could be constructed is highly doubtful (the proof of lattice optimality could not be found in a reasonable amount of time, but a strong suspicion exists after failing to develop a construction method for such a lattice; anyway, the question is only of theoretical interest); the lattice in Figure 2.3b comes very close to these requirements: only Agents 9 to 12 have one maximum distance of 3 (e.g. from Agent 9 to Agent 11). All other

Agents are at most distance 2 away from all other Agents. The predicted value for TD is 130, with an experimental value of 115.

Conclusion on predicting E[TD] A method, based on elementary graph-theoretical concepts gives an upper limit for $E[TD]$. The method provides an easy, indirect way of discarding improbable states in the corresponding Markov chain. Moreover, the method seems to suggest that two parameters could be used to compare lattices with equal numbers of Agents: a first parameter is, the maximum distance in the lattice, given a starting vertex, averaged over all starting vertices (since the probability that the initial hit occurs in an Agent is uniform for all Agents). A second parameter is the number of shortest paths from one Agent to the most distant Agents, averaged over all number of distant Agents and all starting Agents. For optimal performance, the first parameter needs to be minimised, while maximising the second. Comparison of these two parameters, calculated for different lattices, allows a ranking of lattices to be made up, without specifically predicting $E[TD]$ or simulating the lattices. The method will be used during construction of CONEMA, where optimal performance and practicability of wiring the machine both need consideration.

Experimental results for TD

Experimental resource allocation results

Some experiments on resource allocation of Lattice SDS are reported in Table 2.2. The same conclusions as in previous section can be drawn: the shorter the distance from one Agent to other Agents, the better a stable population can sustain itself. If the Model is only slightly distorted, then results are similar to Standard SDS, and scale well with Search Space size. However, Lattice SDS does not seem to scale very well with high distortion and large Search Space size.

Conclusions on Lattice SDS

Lattice SDS removes the need for full connectivity between Agents. Simulation experiments seem to suggest that for 4 to 6 connections in a lattice of 16 to 64 Agents, performance will be acceptable, compared to Standard SDS. An approximative method to compute the diffusion time of correct locations was developed. Based on this method, parameters can be extracted from the lattice layout, which will allow a qualitative ranking of lattices to be constructed, so that optimal lattice layouts can be easily found. No attempt to

Agents	Layout	Neighbours	P1	P2	Predicted	Simulated
16	Full	15	1	1	100	100
16	Optimal	4	2.25	1.82	130	115
16	$4 \times 4 \times 1$	4	4	24	133	117
16	PCB	4	4	13	155	127
16	$8 \times 2 \times 1$	3	5	10	160	149
16	$16 \times 1 \times 1$	2	8	2	240	242
32	Full	31	1	1	250	250
32	$4 \times 4 \times 2$	5	5	120	365	300
32	$8 \times 4 \times 1$	4	6	60	420	362
32	$16 \times 2 \times 1$	3	9	18	608	560
32	$32 \times 1 \times 1$	2	16	2	992	992
64	Full	63	1	1	596	598
64	$4 \times 4 \times 4$	6	6	720	941	716
64	$8 \times 4 \times 2$	5	7	420	1051	836
64	$8 \times 8 \times 1$	4	8	280	1094	900
64	$16 \times 2 \times 2$	4	10	180	1459	1234
64	$16 \times 4 \times 1$	4	10	180	1459	1231
64	$32 \times 2 \times 1$	3	17	34	2368	2165
64	$64 \times 1 \times 1$	2	32	2	4032	4023

Table 2.1: Diffusion time for lattices with 16, 32 and 64 Agents. 'Full' layout is asynchronous Standard SDS without self-selection; 'Optimal' is the layout from Figure 2.3b and 'PCB' the layout from Figure 2.3a; all the other lattices are nearest-neighbour layouts with periodic boundary conditions. Each Agent in a given lattice has the same number of neighbours. P1 is the average, over all Agents, of the longest distance to other Agents in the lattice. P2 is the number of shortest paths existing between two most distant Agents, averaged over all combinations of most distant Agents. The predicted TD for the 'Full' layout is obtained from the Markov chain model for asynchronous SDS; TD for other lattices has been predicted using the network approximation. The simulated TD is the average number of iterations over 5000 runs, starting with 1 randomly chosen active Agent and until all Agents in the lattice are active.

model resource allocation of Lattice SDS has been made, but the same parameters seem to give a qualitative indication of robustness. The Ising model - or more generally, Markov Random Fields - looks like a natural candidate for modelling resource allocation. In general, results from this section seem to suggest that, if implemented in hardware with limited interconnections,

Agents	Layout	Neighbours	Active Agents (%)					
			$p^- = 0.1$			$p^- = 0.3$		
			100	300	500	100	300	500
16	Full	15	88.7	89.0	88.8	51.4	49.4	42.2
16	Optimal	4	88.2	88.8	88.5	46.8	40.7	34.9
16	$4 \times 4 \times 1$	4	88.5	88.8	88.6	47.3	39.5	35.6
16	PCB	4	88.6	88.5	88.6	44.3	38.5	33.4
16	$8 \times 2 \times 1$	3	88.2	88.1	88.2	40.1	26.9	23.4
16	$16 \times 1 \times 1$	2	87.1	87.4	87.4	25.2	16.3	13.3
32	Full	31	89.2	88.9	88.9	55.2	55.8	56.8
32	$4 \times 4 \times 2$	5	88.6	88.5	89.0	51.6	51.4	50.1
32	$8 \times 4 \times 1$	4	88.6	88.5	88.2	48.0	47.3	45.7
32	$16 \times 2 \times 1$	3	88.4	88.3	88.3	40.4	33.1	32.3
32	$32 \times 1 \times 1$	2	87.6	87.6	86.9	23.8	14.9	8.6
64	Full	63	88.7	88.6	89.0	57.0	56.1	56.5
64	$4 \times 4 \times 4$	6	88.6	89.0	88.6	53.8	52.5	53.6
64	$8 \times 4 \times 2$	5	88.5	88.3	88.5	50.2	51.6	50.0
64	$8 \times 8 \times 1$	4	88.5	88.6	88.4	50.0	49.0	46.0
64	$16 \times 2 \times 2$	4	88.6	88.5	88.4	49.3	46.6	46.4
64	$16 \times 4 \times 1$	4	88.6	88.5	88.8	49.5	46.7	47.6
64	$32 \times 2 \times 1$	3	88.4	88.2	88.2	39.2	37.5	28.7
64	$64 \times 1 \times 1$	2	87.5	87.8	87.3	23.5	12.4	7.8

Table 2.2: Resource allocation for lattices with 16, 32 and 64 Agents. 'Full' layout is asynchronous Standard SDS without self-selection; 'Optimal' is the layout from Figure 2.3b and 'PCB' the layout from Figure 2.3a; all the other lattices are nearest-neighbour layouts with periodic boundary conditions. Each Agent in a given lattice has the same number of neighbours. Experiments have 1 imperfect instantiation of the Model in the Search Space (once with $p^- = 0.1$ and once with $p^- = 0.3$) and were conducted for three different Search Space sizes ($M = 100$, $M = 300$ and $M = 500$). The resulting values are the average percentages of active Agents, over 128000 iterations in steady-state.

SDS will still be a fast and robust pattern matching algorithm.

2.4.8 One- or Two-Way Communication

The last issue raised in Section 2.4.3 was one-way communication. Most, if not all, of the discussed algorithms can use both ways of communication:

an Agent can try to set up a communication link with one of its neighbours to get the required information; or if Agents send out the needed information at certain moments, an Agent can just sit and wait till the required information comes in. It is more a question of which option will provide the heaviest network traffic, waiting times etc., and it is difficult to simulate this in software. From the biological viewpoint, it is logical to move to one-way communication, since hand-shake protocols seem quite improbable in neurons.

2.4.9 Conclusions

The effect of relaxing properties of Standard SDS, which are considered to be biologically implausible or difficult to implement in hardware, has been investigated. Results show that Agents can operate asynchronously, without knowledge of the internal state of other Agents, and with limited connections to other Agents, and still perform a fast and robust search. Some of these relaxations are already used in the present version of the connectionist model of SDS, SNSDN, which will be further discussed in Chapter 3.

2.5 Combinatorial Optimisation

Some ideas on using Stochastic Diffusion Processes for Combinatorial Optimisation are presented. The work is far from conclusive, but initial results are promising. The proposed technique was used on the Weighted Matching Problem (WMP) and the Travelling Salesman Problem (TSP). WMP is a polynomial-time problem, and results will not be discussed here. TSP is an NP-complete problem, and therefore much more interesting; it is also one of the most popular optimisation problems, and several fast and effective methods exist, some of them in ideas quite similar to the proposed method. The main question will be whether SDP can provide a valuable contribution to the field. In order to answer this question, much more research is needed.

2.5.1 Travelling Salesman

The TSP is an old and well-known Combinatorial Optimisation problem. The goal is, given a list of cities and distances between them, to minimise the length of the tour which visits every city just once. The problem is NP-complete, which means that the time it takes to solve the problem exactly, grows exponentially with the number of cities N. An introduction to the TSP can be found in [19]. An overview of algorithms based on *local* heuristic optimisation techniques can be found in [20]. Among the algorithms discussed are 2-OPT, 3-OPT and the Lin-Kernighan algorithm, which seems to be the heuristic champion so far. 2-OPT operates by picking out two edges in the tour, deleting them and then reconnecting them in the other possible way. The edges are not picked out at random, but in such a way that an optimal improvement is being made. An interesting approach, based on the same ideas as SDP, is presented in [21]; it was originally proposed in [22] and termed *Memetic Algorithms* or *Genetic Local Search*. Agents go through periods of independent, local optimisation, interspersed with periods of interaction with other Agents. The details of the algorithm are quite different though: local optimisation is based on Simulated Annealing and interaction is based on Genetic Algorithms. A set of benchmark problems for TSP is available, TSPLIB ([23]). It contains various interesting and hard-to-solve instances of TSP. Results presented here will be based on instances from TSPLIB, for which, in most cases, the optimal solution is known.

2.5.2 The Algorithm

The algorithm consists of four steps:

1. **Initialise Agents:** The internal state of the Agents is set to inactive. Each Agent is initialised with a randomly generated tour visiting all cities. The micro-features are the distances between two cities which are neighbours in the tour.

2. **Test:** A random micro-feature (i,j), the distance between city i and city j, is chosen from the Agent's tour. Another Agent is chosen at random for comparison. In that Agent, the micro-feature (i,k) is determined by starting city i from the first micro-feature. The end-city k determines a micro-feature (k,l) in the original Agent. The following test is performed:

$$d(i,j) + d(k,l) \leq d(i,k) + d(j,l) \tag{2.5}$$

   with $d()$ an arbitrary distance measure, determined by the problem at hand. If Equation 2.5 is true, then the Agent's state is set to active, otherwise it is set to inactive. Activity means then that the chosen micro-feature is better than the corresponding micro-feature in the other Agent.

3. **Diffuse:** Active Agents do nothing in this phase. Inactive Agents' behaviour is dependent on the internal state of the Agent chosen for

comparison during the Test phase; if an inactive Agent lost the comparison from an Active Agent, it will copy the micro-feature (i,k) to its own tour. If an inactive Agent lost the comparison from another inactive Agent, it will swap the micro-feature (i,j) for a randomly chosen new one.

4. **Termination:** The algorithm ends when all Agents are active during a certain number of iterations. 100% activity means it is unlikely that any more improvements will be made using this method.

Agents operate synchronously. It should be clear that nowhere in this algorithm, the total length of the tour needs to be computed. Only at the end, when the algorithm has terminated, need the lengths to be calculated and the optimal tour selected.

2.5.3 Results

A typical result is presented in Figure 2.6. It optimises **hk48** of TSPLIB. The problem has a symmetric distance matrix, but distances are *non-euclidean*, i.e. they do not fulfill $d(i,j) \leq d(i,k) + d(k,j)$. The optimal tour has length 11461. This particular run returns as best tour 11607, which is 1.27% from the optimal tour. Note that tour lengths during convergence have only been computed for inspection purposes.

2.5.4 Conclusion

Problems like **hk48** are merely toy problems. Some instances in TSPLIB consist of up to 80000 cities, and euclidean problems of millions of cities have been reported in the literature. Certain algorithms, like variants of Lin-Kernighan, provide solutions within a few percents of the optimal solution in a matter of minutes. More testing needs to be performed on the present instance of SDP optimisation, and modifications need to be investigated. The algorithm as it is described here, seems to be nothing more than a stochastic variant of 2-OPT. Also, the requirement for each Agent to have a copy of a complete tour, might prove to be unacceptable for large numbers of cities. Apart from these objections, first results indicate an interesting opportunity for an extension of SDS to Combinatorial Optimisation.

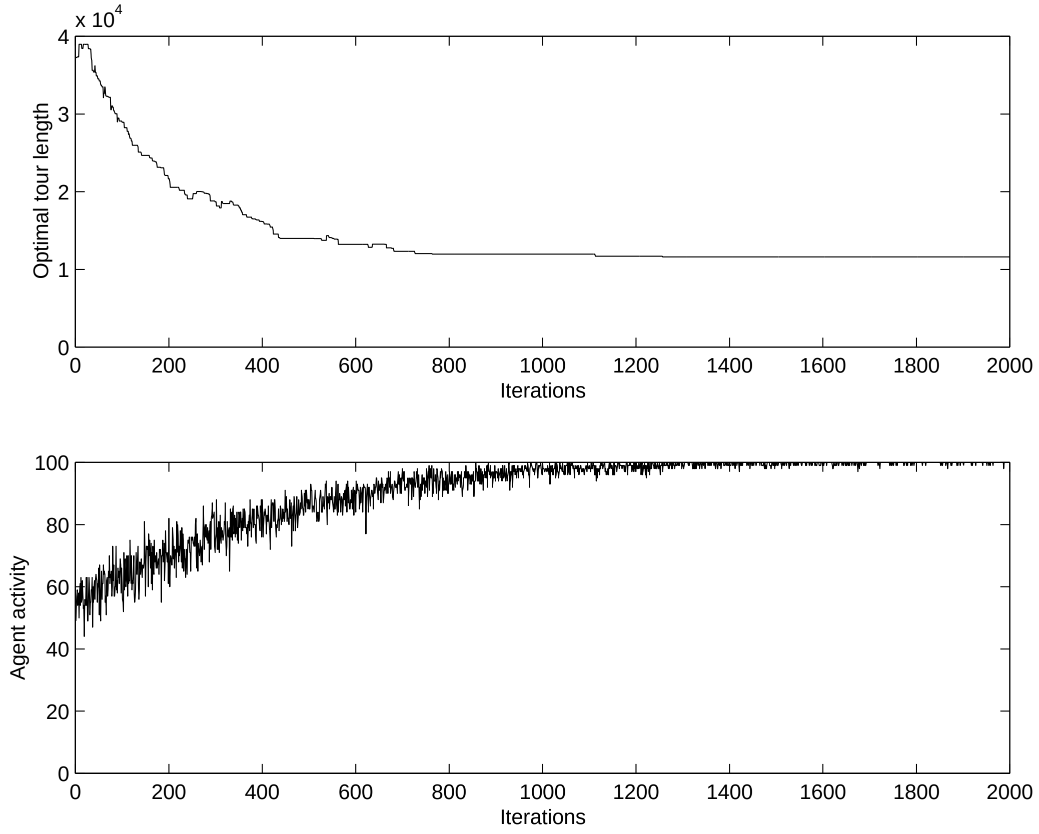

Figure 2.6: A typical run for a population of Agents performing optimisation on the TSP. The best tour in the population is depicted in the upper graph; the lower graph gives the Agent activity. As activity increases, optimisation slows down. The population is considered to be converged when activity is at a maximum. The problem at hand was **hk48** from TSPLIB. 100 Agents were allowed to optimise over 2000 iterations.

Chapter 3

Connectionist Models of SDS

The Nineties were proclaimed the 'Decade of the Brain' in the United States. With that decade now officially closed, one might ask whether it has brought the revolution that was expected 10 years ago. Improved imaging and recording techniques and anatomical studies have led to a detailed mapping of functions to specific areas of the brain. Cellular and Molecular Neuroscience have seen a real explosion of different cell types and neurotransmitter systems. However, are we really any closer now ? Many questions remain unanswered: the basic mechanisms of memory and information encoding and decoding are still largely a mystery. Higher up the ladder, a firm link between mind and brain is far from established: little is known about how activation of neurons in this or that area, using such and such neurotransmitter, leads to an internal representation, awareness of that representation, and eventually awareness of self. Not only is research into consciousness in a "preparadigmatic" stage, there is not even a consensus on *what* has to be explained ([24]). This confusion is even apparent in the popular scientific literature: books, ranging from the optimistic "we-are-almost-there" tone of voice (e.g. [25]) to the "it-will-never-happen" doom scenarios (e.g. [26]), are being published at an astonishing rate; undoubtedly, there is a huge market for both kinds.

Almost every introduction or layman's text on Neuroscience starts with the statement that the human brain is the most complicated structure in the known universe. If the past decade has taught us anything at all, then it should be that we (again) underestimated the problem. Several long-standing 'dogmas' are being questioned by new empirical data. For instance, it was long thought that no neuron regeneration takes place after birth. However, recent research ([27]) showed that new neurons are generated in the brain stem and can replace dead neurons in certain areas of the neocortex. Another problem forms the neural code: the prevailing opinion was that neurons encode information in their mean firing rate, but research on reaction times and spike reproducibility suggest this cannot be the full story. Dendrites, which were supposed to be passive cables, transporting the incoming signals to the soma where the computation occurs, turn out to be highly complex structures with active membrane properties, capable of selectively filtering out single spikes. These - and more - discoveries make it difficult to sustain the McCullogh-Pitts ([28]) model of a neuron as a simple computational device. A more rigourous critique on this subject is developed in [12].

A concise overview of biological evidence for different information encoding schemes in the brain and dendritic processing capabilities will be presented in a first section. The Spiking Neuron Stochastic Diffusion Network (SNSDN), inspired by the evidence, will then be described in detail. Next, attention modelling with SNSDN will be discussed. To conclude the chapter, some directions for further research will be proposed.

3.1 Biological Neurons

3.1.1 Neural Codes

Information encoding in the brain is the subject of an intense debate. The classical viewpoint of information encoded in mean firing rates of single neurons, is being questioned for a number of reasons. For instance, the time window over which averaging takes place has to be fairly large because of high variability observed in recordings of single *in vivo* neurons, when presented with the same stimulus; this is in contradiction with the timescale of certain processes in the brain, some of them occurring on a timescale of less than 100ms, much less than averaging over a time window for a single neuron would allow ([29]). A second problem is the irregularity of spike timing itself. [30] demonstrates that a simple integrate-and-fire model, receiving irregular spike trains as input, can not maintain the same irregularity in its output. They propose that information is carried by the exact arrival times of spikes and that neurons act as coincidence detectors of presynaptic events.

That the timing of spikes can carry information about the temporal structure of the stimulus in certain areas of the brain, is widely acknowledged; that it is a mechanism used in the cortex, and can carry information about other stimulus characteristics such as spatial form, is not so easily accepted ([31]). The above problem of spike irregularity can be explained differently: it can be caused by a balance between excitatory and inhibitory inputs ([32]); or by synchronicity in the input signal ([33]). Whatever the explanation, both approaches move away from time-averaging over single neurons, to averaging over a population of neurons.

Some researchers claim that certain results are better explained by time encoding. In [34], recordings in the visual cortex of the macaque monkey suggest that neurons are well equipped to decode stimulus-related information on the basis of relative spike timing and interspike interval (ISI) duration.

For other researches, the question of time or rate encoding is an irrelevant one. The dividing line between pulse codes and rate codes is not always clear ([29]), and leads them to the conclusion that the question is merely one of timescales ([31]). This conclusion sidesteps an interesting possibility: time encoding has a higher information capacity than rate encoding, a capacity which can not only be used to send the same sort of information faster, but allows *multiplexing* of information. It has been argued in [10] that multivariate information encoding could solve some of the problems in classical connectionism.

3.1.2 Dendrites

Dendrites have long been considered to be passive cables. Even when reports of dendritic action potentials surfaced in the 1960s, the idea of active dendrites was slow to spread ([35]). It was not until the advent of new techniques that voltage-gated Ca^{2+}, Na^{+} and K^{+} channels were discovered in several types of neurons ([36],[37]). As a result of these active membrane properties, dendrites can react in a much more complex way to the input than just transporting it to the axon hillock. In [38], four types of information flow are identified:

1. information transfer from dendrites to the axon hillock.
2. local integrative sites within the tree
3. information flow from the axon hillock back into the dendrites
4. multiple active dendritic sites for initiating information flow

It is suggested (e.g. in [39]) that these properties allow neurons to selectively react upon single EPSPs, as opposed to the classical view where all EPSPs are integrated in the soma. According to [35], coincidence of back-propagating spikes and EPSPs produces long-term potentiation, a cellular mechanism that may underlie learning.

Not only active membrane properties, but also morphology of dendritic trees are thought to influence neuronal behaviour. A review can be found in ([40]).

Although a conceptual framework for the significance of all this new information is lacking at present, it is clear that new levels of complexity are being added. In the next section, some of the new ideas will be (implicitly) incorporated into a neural network, performing an SDS-like search.

3.2 Details of SNSDN

3.2.1 High Level Description of the Network

The network consists of three sets of neurons. **Memory Neurons** encode the micro-features of the Model, one for every micro-feature; **Receptor Neurons** encode the micro-features of the Search Space, which is also called the **Retina**. The search is performed by **Matching Neurons**, neuronal equivalents of SDS Agents. In this particular exposition, the network performs one-dimensional string matching; micro-features are therefore single characters.

Memory and Receptor Neurons are fully connected to all Matching Neurons; Matching Neurons are also connected to all other Matching Neurons. It is assumed that Matching Neurons have knowledge about the source of a signal, i.e. they know whether a spike is generated by a Memory, Receptor or other Matching Neuron.

All neurons communicate via spike trains; information is encoded in the time between spikes, the interspike intervals (ISI). Memory and Receptor Neurons encode *two types of information*: the first ISI of a spike train encodes *where* a micro-feature (character) is located in Model or Search Space, the second ISI encodes *what* the micro-feature is. Matching Neurons communicate only one type of information to other Matching Neurons: a location pointer into the Search Space.

Matching Neurons have two different internal states, labelled active and inactive; but the meaning of these states is slightly different from Standard SDS. Inactive Matching Neurons do not have a location pointer into the Search Space; as a result, they can not undergo test phases. They wait for incoming spikes; if the first spike comes from another Matching Neuron, then they decode the information from its spike train, store the location pointer and change the internal state to active. If the first spike comes from a Memory or Receptor Neuron, then the *where* value is decoded from its spike train; the Matching Neuron then waits for a spike from a Receptor or Memory Neuron respectively. It adds the two *where* values, obtaining a valid location pointer into the Search Space which is then encoded in a spike train and send out to other Matching Neurons. The mechanism of generating new random location pointers is thus similar to the one used in Unlabelled SDS. Active Matching Neurons wait for a first Memory spike to arrive; its *where* value determines the micro-feature which will be tested. Together with the location pointer of the Matching Neuron, it also determines the corresponding Receptor Neuron. When both the *what* value from Memory and Receptor Neuron have been extracted from their respective spike trains, they are compared; if the test fails (the micro-features are not the same), then the Matching Neuron switches its state to inactive. If the test succeeds, then the location pointer of the Matching Neuron is encoded in a spike train and broadcasted to other Matching Neurons. A population of Matching Neurons evaluating the same location in the Search Space results in a dominant spike train, defining this location.

In terms of the classification of SDS algorithms given in Chapter 2, the state updates have a physical dimension of time. Every neuron operates in its own time, but the probability distributions of updating are dependent on the behaviour of other neurons. Therefore, it is difficult to call the network asynchronous as defined in section 2.3.1. Although the exact meaning of internal states active and inactive is different from Standard SDS, their interpretation remains: activity over long periods of time means likeliness to point to the correct instantiation. No explicit knowledge of internal states of other Matching Neurons is necessary, and communication is one-way. Matching Neurons are fully interconnected with all other Matching Neurons, but work on Lattice SDS suggests this condition can be relaxed. The stochastic component in the process is governed by a set of random **refractory periods** and **transmission delay** parameters, which will be discussed in the next subsection.

With respect to the section on new neurobiological knowledge (3.1), it can be seen that neurons use a time encoding scheme, multiplexing of information, and that local dendritic processing is implicitly present in the mechanism of filtering out spikes from a single source.

3.2.2 Randomisation Process

Standard SDS operates by picking micro-features, other Agents and new Search Space locations out at random. Chances that a biological neuron can operate in the same way are minimal. Randomisation in the network therefore is accomplished through a set of parameters, mimicking refractory periods and transmission delays. After every spike train, a Memory or Retina Neuron waits a random number of timesteps before it starts sending a new spike train. In Matching Neurons, sending of spike trains and receiving of spikes (for the next update) can occur simultaneously. Separate refractory parameters for both processes exist. In addition to the refractory parameters, every neuron-to-neuron contact (synapse) can have a separate delay parameter; this simulates axon and synapse transmission delays. As will be seen in the following section, the balance between exploration and exploitation is very sensitive to these parameter settings.

3.2.3 Comparison with Standard SDS

Within certain ranges of the randomising parameters, behaviour of Standard SDS and SNSDN is qualitatively similar. This can be seen in Figure 3.1. In this experiment, only random transmission delays are used. Since the transmission delay is set for every synapse independently, complete decoupling of spiking events is assured. The randomisation processes will therefore resemble the uniform random distributions from SDS. The difference between the two is mainly explained by the timing behaviour: several hundreds of timesteps in SNSDN make up one iteration in SDS. The timescale of SNSDN is compared to the timescale of SDS by counting the number of test-phases that occurred during the entire run.

When no transmission delay parameters are set, or the random refractory intervals are large compared to the transmission delays, then spiking events are not completely uncoupled. Suppose a group of Matching Neurons is waiting for a spike from a Memory Neuron; if one Neuron fires a few timesteps before other Memory Neurons fire, then this neuron will be picked out by all waiting Matching Neurons. They will all start testing the same micro-feature; in case it is a faulty micro-feature, they will all switch to inactive. The result is that populations are less stable, as can be seen in Figure 3.2. The same experiment was performed as in Figure 3.1; the only difference was that refractory parameters were set as well. The graph shows more an all-or-nothing behaviour: populations can reach higher values, but are not equally stable.

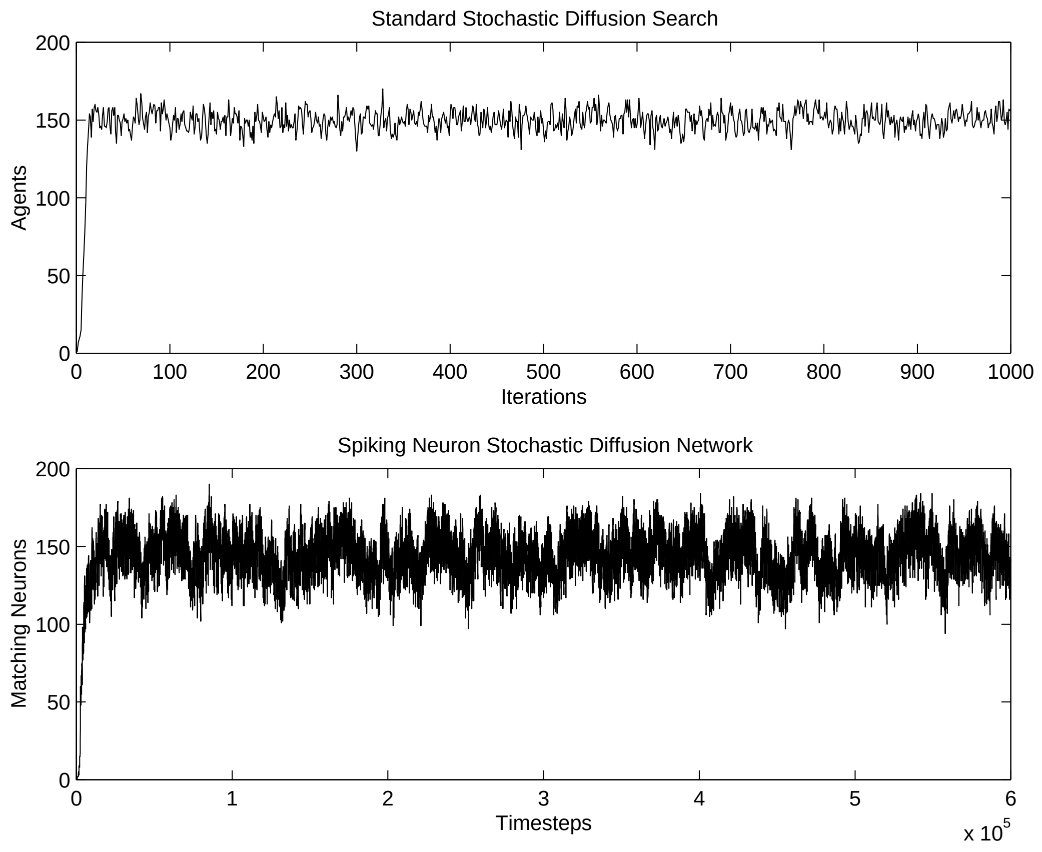

Figure 3.1: a) Time evolution of the cluster of Agents evaluating the correct location in the Search Space. b) Time evolution of the number of active Matching Neurons, emitting the spike train encoding the correct location. $N = 200$, $M = 200$ and $p^{-} = 0.2$. For SNSDN, only transmission delay parameters are set. SNSDN ran for 60000 timesteps, in which approximately the same number of test phases occurred as in the 1000 synchronous iterations of SDS.

A formal analysis of balance between exploration and exploitation, of cluster stability and cluster size in relation to randomisation parameters, has not been carried out. One experiment does show the sensitive dependence of the exploration/exploitation balance on the randomisation parameters. Eight experiments were conducted, in which was counted how many times an inactive neuron picked out another inactive neuron during diffusion (and thus effectively generated a new random location pointer into the Search Space - exploration), and how many times an active neuron was chosen - exploitation of the correct location. The balance between the two gives an indication

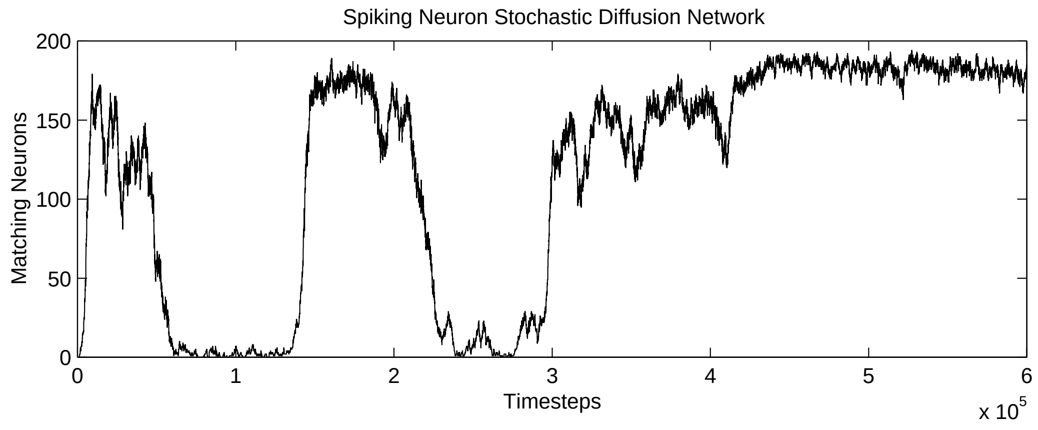

Figure 3.2: Time evolution of the number of active Matching Neurons, emitting the spike train encoding the correct location. $N = 200$, $M = 200$ and $p^{-} = 0.2$. In addition to transmission delays, also refractory periods are used. This results in higher, but less stable populations.

of the resource allocation for that particular set of parameters. It can be seen in Table 3.1 that the balance shifts considerably from experiment to experiment, although the difference in randomisation parameters is small from experiment to experiment. As a reference, the same values for SDS are given.

3.3 Attention

One of the main reasons to implement SDS as a neural network is that it seems to present itself as a model for *attention.* The question of selective attention has been an important area of research in psychology for a long time; however, it has remained controversial whether there are separate mechanisms subserving attention.

It is argued in [41] that brain activity in many cortical areas can be selectively amplified or suppressed, as a function of the attentional set. *Attentional amplification* is just that what SNSDN seems to do: given the Memory that has to be attended to, the corresponding target on the Retina is detected and its stimulus is amplified in the dominant spike train of a population of Matching Neurons, and this in a *dynamical* way: a moving pattern, a new pattern that has to be attended to, different patterns between which switching of attention occurs, all result in a different amplified signal, by the same set of neurons.

Exp.	Inactive	Active	Balance	Iterations	Mean Pop.
SDS	2122	4672	31%/69%	300	78%
1	411	6822	6%/94%	320	85%
2	644	6257	9%/91%	320	84%
3	1860	5895	24%/76%	300	80%
4	1885	6138	24%/76%	310	80%
5	1978	5899	25%/75%	310	80%
6	1808	5487	25%/75%	310	83%
7	2505	4077	38%/62%	335	85%
8	660	2483	21%/79%	330	94%

Table 3.1: The balance between exploration and exploitation as a function of randomisation parameters. The experiment, with $N = 100$, $M = 95$ and $p^- = \frac{1}{6}$ was conducted with different sets of randomisation parameters. Of the 8 different randomisation parameters, only 1 was changed from experiment to experiment. 'Inactive' gives the number of times an inactive neuron was picked out during diffusion; 'Active' the number of times an active neuron was chosen. 'Balance' gives the ratio between previous two columns. Iterations gives the equivalent number of Standard SDS iterations, and Mean Pop. the average population with the correct location. Note that in SNSDN, the number of diffusion phases is not equal to the number of test phases, and that a higher Mean Pop. value means that less diffusion phases take place, and thus less exploration.

The problem of attention is also related to the *binding problem*. Different binding problems exist ([42]): one of them is how different properties of objects (e.g. shape and colour) are bound to the objects they characterise. Attention seems to be underlying binding ([42]), since, for binding to occur, attentional amplification is necessary. It is simply impossible for the brain to combine all different features of one kind to features of another kind, and then select the correct combination.
A number of possible binding mechanisms have been envisaged; one receiving much attention nowadays is synchronised firing of populations of cells ([42],[43]).

For a certain range of randomisation parameters in SNSDN, the dominant spike train of the Matching Neurons, evaluating the correct location in the Search Space, will be loosely synchronised. This synchronisation results in a dominant frequency in the frequency spectrum of the Matching Neuron spikes. The dominant frequency is dependent on the location pointer encoded in the spike train: if a different target is attended to, then the dominant frequency will change. Figure 3.3 shows how attention is shifted from one pattern to the second. In Figure 3.4, the Matching Neuron Spikes are shown during various stages of attention, together with the power spectra. The different spiking behaviour during the three different stages of attention can be clearly distinguished: first, Pattern 1 is attended; during the shift from the first to the second pattern, there is no synchronisation, no dominant spike train so no dominant frequency. When the second pattern is discovered, a new dominant frequency emerges from the network.

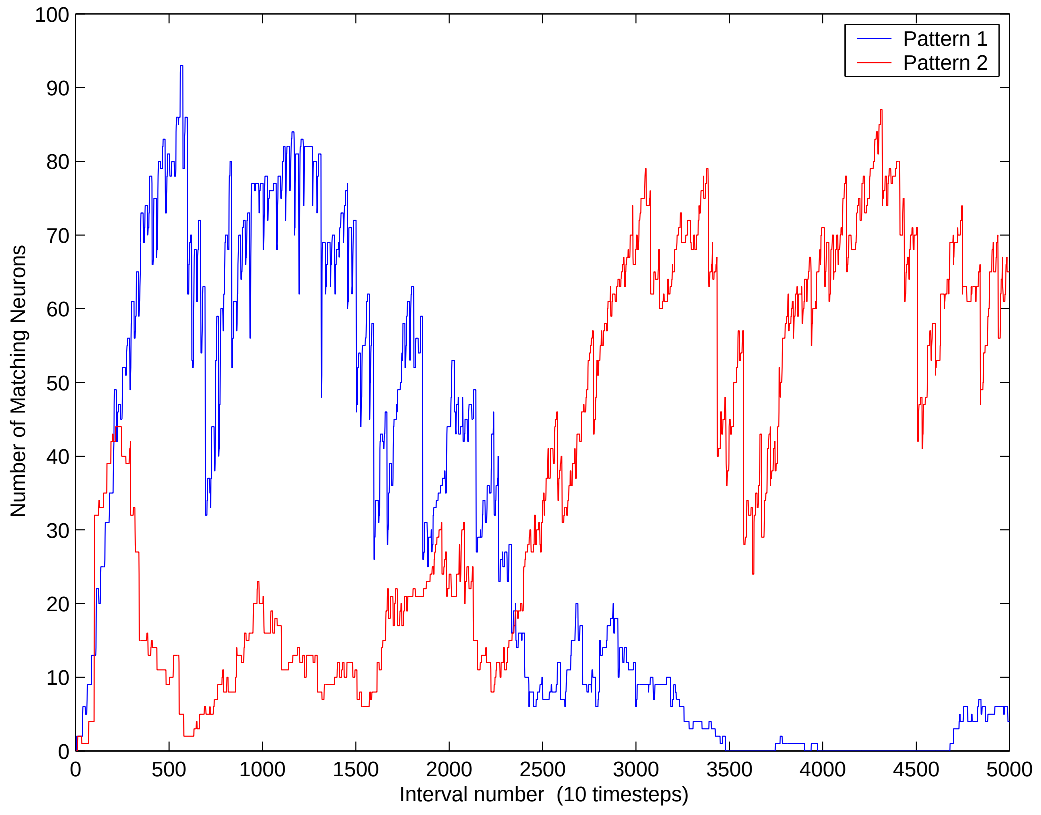

Figure 3.3: Time evolution of Matching Neurons, evaluating the two imperfect instantiations of the Memory on the Retina. $N = 100$, $M = 95$ and $p^{-} = \frac{1}{6}$.

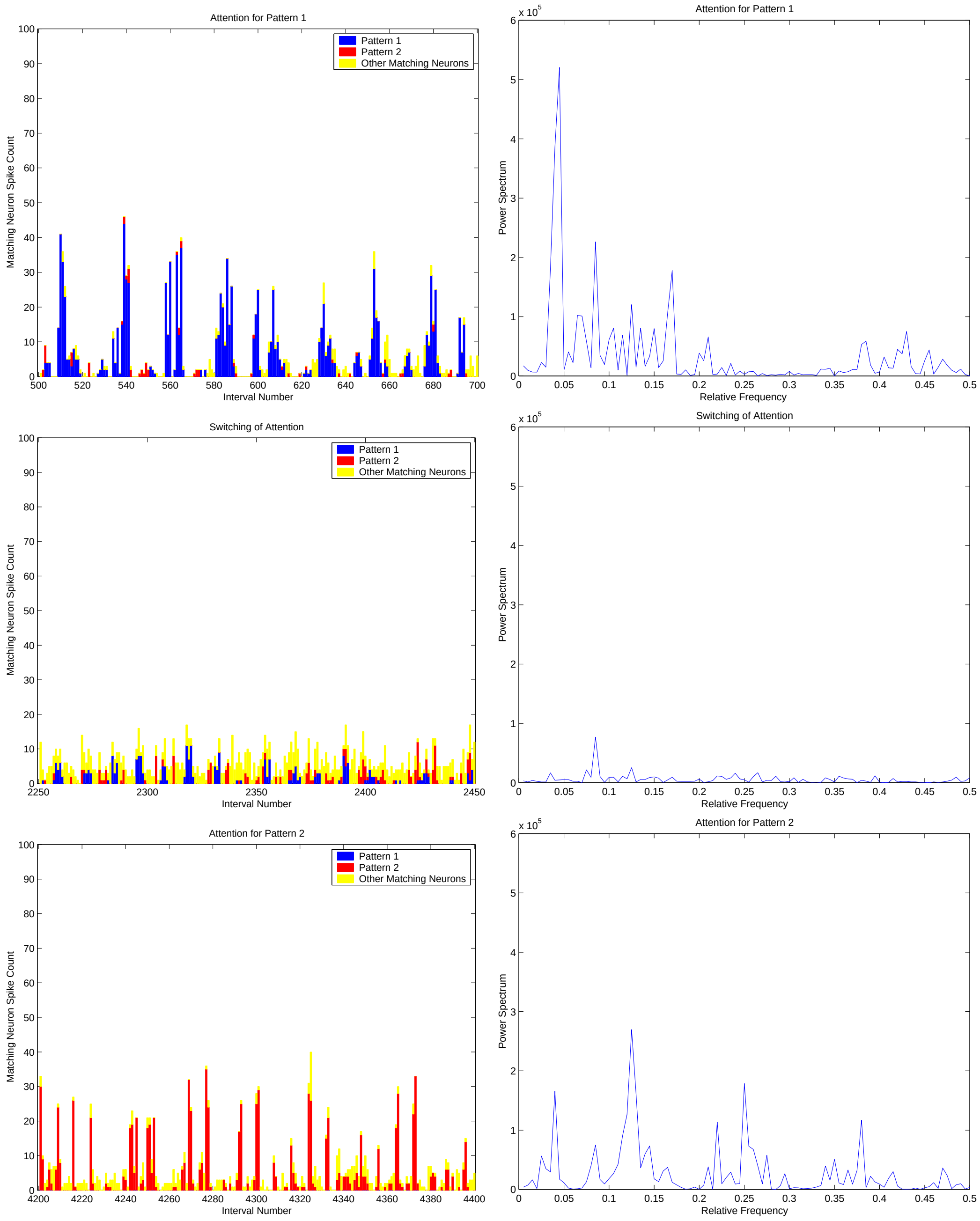

Figure 3.4: On the left side, spikes of Matching Neurons during several stages of attention, each for 200 intervals (2000 timesteps); on the right, their respective power spectra. The three different stages of attention can be clearly seen in both time and frequency domain

.

3.4 Further Research

The SNSDN model gives rise to interesting properties, but is simulated on a rather high level; it would be very interesting to develop it further. First step to be implemented is local connectivity. Work on Lattice SDS has shown that this should pose no problems. The main reason for local connectivity is implementation in hardware: the number of connections will always be the limiting factor in that case, as will be seen in Chapter 4. Another reason is biological plausibility: although small groups of neurons could be completely interconnected, it would be strange to assume this for large groups of neurons.

Although spatial dimensions of the neurons are simulated by transmission delays, neurons in SNSDN do not have a 3D morphology. Simple distance measures between neurons could form a start of giving the neurons a spatial extension; this would affect the transmission delays, which would be partly random (mimicking variability in synaptic transmission), and partly dependent on distance between neurons. Higher levels of detail could bring a more exact dendritic morphology into account, but this would increase complexity of the simulation much more. Together with spatial dimensions, dendritic active membrane properties could be incorporated as well; this leads to so called compartmental models (see Chapter 1 in [29]). Their complexity is too high to simulate with a large number of neurons, but small numbers could be used to see whether the current level of SNSDN operation can be obtained using lower level dendritic membrane properties.

A last interesting suggestion has to do with the output signal of the Matching Neurons; currently, the attentional amplification is not used anywhere. Suppose the Retina is dynamical, can slide over the Search Space, and has a coarser sampling of the Search Space at its borders than in the centre. The output signal of the Matching Neurons could be used by another set of Motor Neurons to steer the Retina so that the desired pattern falls in the centre of vision, or to track a moving pattern.

Chapter 4

Hardware Implementations

A first attempt at parallel implementation of SNSDN ([15]) was not very successful; an SGI Genesis machine with 12 processors was much slower than a normal PC. The main reasons are the overhead of the communications protocol (such a machine is made for heavy computations and relatively little communication between processors) and the constant workload caused by other users. It also proved to be difficult to program the communication protocol.

A lot of work on parallel implementations of Spiking Neurons has been done by other researchers (see Chapter 3 and 5-9 of [29]). Implementations range from detailed neuromorphic models in Analog VLSI to digital simulations on parallel computers. The main problem all pure hardware implementations seem to suffer from is limited connectivity; it seems to be very difficult to get the same number of connections as in the brain. Another indication that neuronal operation is better described in terms of communication !

Analog VLSI is unfortunately out of range, because it is too costly. Three other possibilities will be discussed, of which the first will definitely be implemented in the coming months, as part of a fourth year undergraduate project. The other two will need more exploration before they can be attempted.

4.1 The Communicating Neurons Machine

The 'Neuron' chip from Echelon is a small processor specifically designed for communication. It has all the necessary built-in communication and control functions for the LonTalk network protocol, a protocol which has very little overhead compared to e.g. an Ethernet protocol. The advantages of using this chip are abundant: since all the communications machinery is on board the chip, design of a multiprocessor machine will be relatively easy. The operating system takes care of all the communication, which is conducted by two dedicated CPU's on the chip. A third CPU takes care of the user application. This setup makes the chip easy to program as well. Disadvantages of the chip are its limited resources: 3kB of EEPROM have to host the user application, while 4kB or RAM can be used for data. The chips only run at 10 or 20Mhz. So it will not be possible to use this machine for computation-intensive applications. However, since SNSDN is mainly based on communication, the Neuron chip seems to be the ideal choice.

CONEMA, the Communicating Neurons Machine, will use 64 Neuron processors on 4 PCB's with 16 processors each. All the processors will be connected to each other via a serial bus. This makes implementation of Standard SDS possible on the machine. The problem with using the serial bus for studying asynchronous SNSDN, is that it will not have the dynamical behaviour that it should have: if a Matching Neuron has to wait until it finds a free slot on the bus to put its spike train on, then the exact timing of the spike train will be lost. However, the chip has 11 I/O pins, which can be set as input or output by the user. This allows 5 bi-directional links to neighbouring chips. A certain level of unpredictability on these lines will be caused by encoding and decoding of the spikes by the dedicated communication CPU's, but timing behaviour should be more controllable than when using the serial bus.

Since Memory and Receptor Neurons need full connectivity to Matching Neurons, they will either have to be simulated indirectly in memory or over the serial bus by some of the processors.

The proposed layout of the PCB's can be seen in Figure 2.3. Four connections to neighbours are on-board. The advantage of this layout is that no lines connecting the chips directly, are crossing. The design will therefore become feasible on a two-layer PCB, with one layer accommodating the serial bus, power, clock and ground lines, and the other layer the direct interconnections between processors. Two of the remaining I/O lines will be used to make connections to chips on other PCB's.

Since the available on-chip memory is too small to record the Matching Neuron's behaviour during a run, some other inspection method will need to be used. The serial bus could be used for this, but it is not certain at present whether its bandwidth will be sufficient, especially if it will also be used for Memory and Receptor Neurons. Alternatively, the last I/O line could be used for inspection purposes. All lines could be assembled on a special-purpose designed board, where information about the exact occurrence of spikes can be collected and eventually transmitted into the memory of the host PC. Whether its design is feasible and worth the effort will have to be decided later on.

4.2 Programmable Digital Hardware

Although custom made VLSI chips are not a possibility, another interesting option for direct implementation in hardware exists: programmable or reconfigurable hardware. Of all the architectures available, Field Programmable Gate Arrays (FPGA's) offer the most desirable properties. They consist of small logic blocks, which can be programmed to implement a set of functions, and of interconnection lines which can be configured to connect logic blocks into more complicated logic functions. FPGA's offer high logic capacity, i.e. the amount of digital logic that can be mapped into a single chip. Some of them are also highly reconfigurable, even during program execution. *Dynamic hardware*, as it is called, offers interesting perspectives on network adaptivity, neural plasticity and learning processes. A slightly outdated but still useful starting point on FPGA's and other programmable logic can be found in [44].

FPGA's are meant to work as synchronous logic; but since they are reconfigurable, implementation of asynchronous circuits is not impossible. [45] proposes an architecture called STACC (Self-Timed Array of Configurable Cells), which replaces the global clock of an FPGA with an array of timing cells that provide local self-timed control to a region of logic blocks. This architecture could prove very useful for the implementation of the self-timed Spiking Neurons.

Whether FPGA's can provide a good platform for SNSDN implementation is not certain at present. FPGA's have a high logic capacity, often an equivalent of many tens of thousands of logic gates. However, no work has been done on rethinking SNSDN in terms of its basic logic operations. An estimation of the number of neurons that can be implemented on an FPGA is therefore not possible. As is the case in most direct hardware implementations of neural networks, the limiting factor will most likely be the connectivity of the network, since a limited number of interconnection lines between logic blocks is available on chip.

The implications of reconfigurable hardware on computation in general are only starting to surface. With the advent of high-level programming languages (like Handel-C, a language very much like C), and chips which are *in situ* reconfigurable, the future of computation might look completely different: the ease of software design combined with the speed of dedicated hardware.

4.3 Programmable Analog Hardware

Very recently, a new form of reconfigurable hardware appeared in the electronics world: Field Programmable Analog Devices (FPAD) or Totally Reconfigurable Analog Devices (TRAC), as they are called by their manufacturers ([46]), offer blocks of circuitry which can be configured to perform analog signal processing. What their capacities are in terms of neurons is even less clear than for FPGA's.

4.4 Conclusion

Of the three parallel implementations discussed in this chapter, the first is being constructed. Once the hardware for this machine is finished, SNSDN will be implemented and provide us with insight in the dynamical behaviour of SNSDN.

Bibliography

[1] J.M. Bishop, (1989). *Anarchic Techniques for Pattern Classification.* PhD. Thesis, Chapter 5, University of Reading.

[2] J.M. Bishop, (1989). *Stochastic Searching Networks.* Proc. 1st IEE Conf. on Artifical Neural Networks, pp 329-331, London.

[3] J.M. Bishop, P. Torr (1992). *The Stochastic Search Network.* In R. Linggard, D.J. Myers, C. Nightingale (eds.), Neural Networks for Images, Speech and Natural Language. New York, Chapman & Hall.

[4] I. Aleksander, T.J. Stonham (1979). *Guide to Pattern Recognition using Random Access Memories.* Computers & Digital Techniques, 2(1), pp. 29-40.

[5] E. Grech-Cini, (1995). *Locating Facial Features.* PhD Thesis, University of Reading.

[6] P.D. Beattie, J.M. Bishop, (1998). *Self-Localisation in the 'Senario' Autonomous Wheelchair.* Journal of Intelligent and Robotic Systems 22, pp 255-267, Kluwer Academic Publishers.

[7] S.J. Nasuto, J.M. Bishop, S. Lauria (1998). *Time Complexity of Stochastic Diffusion Search.* Neural Computation '98, Vienna, Austria.

[8] S.J. Nasuto, J.M. Bishop, (1999). *Convergence Analysis of Stochastic Diffusion Search.* Journal of Parallel Algorithms and Applications 14, pp 89-107.

[9] S.J. Nasuto, (1999). *Resource Allocation Analysis of the Stochastic Diffusion Search.* PhD Thesis, University of Reading.

[10] J.M. Bishop, S.N. Nasuto, (1999). *Communicating Neurons - an Alternative Connectionism.* Proc. WNNW99, York.

[11] S.J. Nasuto, J.M. Bishop (1998). *Neural Stochastic Diffusion Search Network - a Theoretical Solution to the Binding Problem.* Proc. ASSC2, Bremen.

[12] S.J. Nasuto, K. Dautenhahn, J.M. Bishop, (1999). *Communication as an Emergent Methaphor for Neuronal Operation.* Lecture Notes in Artificial Intelligence, 1562:365-380, Springer.

[13] S.J. Nasuto, J.M. Bishop, K. De Meyer, (submitted). *Emergent Computation Based on Inter-Spike Interval Coding and Active Communication.* Neural Networks.

[14] K. De Meyer, J.M. Bishop, S.J. Nasuto, (2000). *Attention through Self-Synchronisation in the Spiking Neuron Stochastic Diffusion Network.* In Proc. ASSC4, Brussels. Consciousness and Cognition, 9-2 p.S81, Academic Press.

[15] T. Morey, K. De Meyer, S.J. Nasuto, J.M. Bishop, (2000). *Implementation of the Spiking Neuron Stochastic Diffusion Network on Parallel Hardware.* In Proc. ASSC4, Brussels. Consciousness and Cognition, 9-2 p.S97, Academic Press.

[16] B. Hölldobler, E.O. Wilson, (1990). *The Ants.* Springer Verlag, Berlin.

[17] M.J.B. Krieger, J.B. Billeter, L. Keller, (2000). *Ant-Like Task Allocation and Recruitment in Cooperative Robots.* Nature 406, 992-995.

[18] A. Dolan, J. Aldous, (1993). *Networks and Algorithms*, Chapter 2. John Wiley & Sons.

[19] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley & Sons, New York.

[20] D.S. Johnson, L.A. McGeoch, (1996). *The Traveling Salesman Problem: a Case Study in Local Optimization.* In E.H.L. Aerts, J.K. Lenstra, *Local Search in Combinatorial Optimization.* Wiley & Sons, New York.

[21] P. Moscato, M.G. Norman, (1992). *A 'Memetic' Approach for the Traveling Salesman Problem: Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems.* In M. Valero, E. Onate, M. Jane, J.L. Larriba, B.Suarez, *Parallel Computing and Transputer Applications.* IOS Press, Amsterdam, pp. 187-194.

[22] P. Moscato, (1989) *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms.* Caltech Concurrent Computation Program, C3P Report 826.

[23] `http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/`

[24] T. Metzinger, A. Engel, (2000). *Constraining Consciousness: Towards a Systematic Catalogue of Explananda.* In Proc. ASSC4, Brussels. Consciousness and Cognition, 9-2 p.S10, Academic Press.

[25] R. Carter, (1999). *Mapping the Mind.* Seven Dials, London.

[26] J. Horgan, (1999). *The undiscovered mind : how the human brain defies replication, medication and explanation.* Free Press, New York.

[27] S.S. Magavi, B.R. Leavitt, J.D. Macklis, (2000). *Induction of neurogenesis in the neocortex of adult mice.* Nature 405, 951-955.

[28] W.S. McCulloch, W. Pitts, (1943). *A logical calculus immanent in nervous activity.* Bulletin of Mathematical Biophysics 5, 115-133.

[29] W. Maass, C.M. Bishop, (1999). *Pulsed Neural Networks.* MIT Press, London.

[30] W.R. Softky, C. Koch, (1993). *The Highly Irregular Firing of Cortical Cells is Inconsistent with Temporal Integration of Random EPSPs.* Journal of Neuroscience 13, 334-350.

[31] R.C. deCharms, A.M. Zador, (2000). *Neural Representation and the Cortical Code.* Annual Review of Neuroscience 23, 613-647.

[32] M.N. Shadlen, W.T. Newsome, (1998). *The Variable Discharge of Cortical Neurons: Implications for Connectivity, Computation and Information Coding.* Journal of Neuroscience 18(10), 3870-3896.

[33] C.F. Stevens, A.M. Zador, (1998). *Input Synchrony and the Irregular Firing of Cortical Neurons.* Nature Neuroscience 1(3), 210-217.

[34] D.S. Reich, F. Mechler, K.P. Purpura, J.D. Victor, (2000). *Interspike Intervals, Receptive Fields and Information Encoding in Primary Visual Cortex.* Journal of Neuroscience 20(5), 1964-1974.

[35] R. Yuste, (1997). *Dendritic Shock Absorbers.* Nature 387, 851-852.

[36] D. Johnston, J.C. Magee, C.M. Colbert, B.R. Christie, (1996). *Active Properties of Neuronal Dendrites.* Annual Review Neuroscience 19, 165-186.

[37] D.A. Hoffman, J.C. Magee, C.M. Colbert, D. Johnston, (1997). *K^+ Channel Regulation of Signal Propagation in Dendrites of Hippocampal Pyramidal Neurons.* Nature 387, 869-875.

[38] G.M. Shepherd, (1999). *Information Processing in Dendrites.* In M.J. Zigmond, F.E. Bloom, S.C. Landis, J.L. Roberts, L.R. Squire, *Fundamental Neuroscience.* Academic Press.

[39] H. Barlow, (1996). *Intraneural Information Processing, Directional Selectivity and Memory for Spatio-Temporal Sequences.* Network: Computation in Neural Systems 7, 251-259.

[40] G.A. Ascoli, (1999). *Progress and Perspectives in Computational Neuroanatomy.* Anatomical Record 257(6), 195-207.

[41] M.I. Posner, S. Dehaene, (2000). *Attentional Networks.* In M.S. Gazzaniga, *Cognitive Science - a Reader.* Blackwell Publishers, Oxford.

[42] A. Treisman, (1996). *The Binding Problem.* Current Opinion in Neurobiology 6, 171-178.

[43] C. Tallon-Baudry (2000). *Oscillatory Synchrony as a Signature for the Unity of Visual Experience.* In Proc. ASSC4, Brussels. Consciousness and Cognition, 9-2 p.S25, Academic Press.

[44] S. Brown, J. Rose, (1996). *Architecture of FPGAs and CPLDs: A Tutorial.* IEEE Design and Test of Computers 13(2), 42-57.

[45] R. Payne, (1997). *Self-Timed Field Programmable Gate Array Architectures.* PhD Thesis, University of Edinburgh.

[46] `www.fac.co.uk` and `www.latticesemi.com`