Incrementally Evolving a Dynamic Neural Network for Tactile-Olfactory Insect Navigation

Øyvin Halfdan Thuv

August 3, 2007

Contents

Abstract

This Masters thesis gives a thorough description of a study carried out in the Self-Organizing Systems group at the Norwegian University of Science and Technology. Much Artificial Intelligence research in the later years has moved towards increased use of representationless strategies such as simulated neural networks. One technique for creating such networks is to evolve them using simulated Darwinian evolution. This is a powerful technique, but it is often limited by the computer resources available.

One way to speed up evolution, is to focus the evolutionary search on a more narrow range of solutions. It is for example possible to favor evolution of a specific "species" by initializing the search with a specialized set of genes. The downside of doing this is of course that many other solutions (or "species") are disregarded so that good solutions in theory may be lost. It is therefore necessary to find focusing strategies that are generally applicable and (with a high probability) only disregards solutions that are considered unimportant.

Three different ways of focusing evolutionary search for cognitive behaviours are merged and evaluated in this thesis: On a macro level, incremental evolution is applied to partition the evolutionary search. On a micro level, specific properties of the chosen neural network model (Continuous-Time Recurrent Neural Networks) are exploited. The two properties are seeding initial populations with center-crossing neural networks and/or bifurcative neurons. The techniques are compared to standard, naive, evolutionary searches by applying them to the evolution of simulated neural networks for the walking and control of a six-legged mobile robot. A problem simple enough to be satisfactorily understood, but complex enough to be a challenge for a traditional evolutionary search.

Keywords

Neuro-Evolution. Incremental Evolution. Genetic Algorithms. Dynamic Systems Theory. Neural Networks. Adaptive Behaviour. CTRNN. Insect Navigation. Neuroethology. Stick Insect. Evolution. Machine learning.

Acronyms used

ADT Abstract Data Type

AEP Anterior Extreme Position

AI Artificial Intelligence

ANN Artificial Neural Network

ANNs Artificial Neural Networks

CL Common Lisp

CPG Central Pattern Generator

CPGs Central Pattern Generators

CTRNN Continuous-Time Recurrent Neural Network

CTRNNs Continuous-Time Recurrent Neural Networks

DNA Deoxyribonucleic Acid

DPE Dynamical Parameter Encoding

EA Evolutionary Algorithm

ESP Enforced Segregated Populations

GA Genetic Algorithm

GAs Genetic Algorithms

GP Genetic Programming

GOF-AI Good Old-Fashioned-AI

LC Locomotion Controller

NTNU Norwegian University of Science and Technology

ODE Open Dynamics Engine

PEP Posterior Extreme Position

RC Resistor Conductor

SANE Symbiotic Adaptive Neuro-Evolution

Chapter 1

Introduction

"A cat that sits on a hot stove won't sit on a hot stove again - or a cold one either."

- Mark Twain

1.1 The AI field

I recently learned from the news that robots had been deployed at the local[1] university hospital. Their task was to move goods like clothes and medical equipment between different hospital departments. A job previously done by humans. Disappointingly, though not surprisingly, the robots differed somewhat from the humanoid robots that are often found in science fiction literature. They had wheels, and not legs, for example. From their visual appearance it seemed that they were just another set of standard, specialized, industrial robots of the sort that are gradually taking over tasks to humdrum for humans to endure.

Moving objects around in a predictable, static environment has become a typical example of a robot task. Humans do not enjoy this kind of work, and programming robots to do these tasks is relatively easy. It also seems that there are few limits for how complex the tasks can be, as long as they

[1] St. Olavs hospital, Trondheim, Norway.

can be accurately and unambiguously described. Systems like the Kuala Lumpur driver less trams, is a typical example. The trams have been transporting people around Kuala Lumpur for a few years now. A complex, but predictable job. The trams run up and down the same track every day. Picking up passengers at predefined stops. Ensuring that the doors open after stopping, and keep them closed while the tram is moving. Only a limited number of exceptions from normal operation can happen. Perhaps the doors will not close for some reason, or some other train may have had a breakdown further down the track. Engineers have probably (hopefully!) planned explicitly for what the train should do if any such exceptions should occur.

Watching the report on the hospital robots, I soon found out that their tasks were actually a bit more complex than what I first thought. The robots did not run along predefined paths specially prepared for them. They had to drive around the existing hospital corridors along pretty much the same path the humans before them had trodden. This is much more difficult than driving up and down a rail road, because, in the corridor, the most unforeseen of events may happen at any time. People might get in the way, for example, and robots running down patients or doctors are not what we want in an hospital. There were also several robots driving around at the same time, all sharing the same floor, the same elevators and even the same tasks. They were also not allowed to end what they were currently doing just because an exception to the normal procedure occurred. That would probably fill the corridors with cold dinner plates and dirty laundry rather quickly. The hospital robots were therefore forced to re-plan all the time, finding new ways to solve their task if anything unusual should happen. This environment was clearly far more complex and dynamic than the environment that the driverless trams had to cope with.

It was clear that these machines had to do quite a bit of complex reasoning to be able to achieve what they did! But how is it possible for an engineer to come up with all possible situations that the robot may fall in to? Can new information be constantly added to it whenever the problem gets harder? If the hospital is partially rebuilt, would the robot cope with that? What if the robot was sent on a mission to mars, could the engineer possibly think of all the problems that may occur there? Is it possible to just keep adding more and more processing capabilities until the robot knows everything, ensuring that it is capable of always doing perfect planning? And, if it really could do perfect planning, would that be artificial *intelligence*?

Many definitions for Artificial Intelligence (AI) have been proposed. The New Oxford American Dictionary define it as "the theory and development of computer systems able to perform tasks that normally require human intelligence". The same dictionary defines intelligence as "the ability to acquire and apply knowledge and skills". Within AI-literature, definitions such as Alan Turnings "imitation game" (from the 1950ies) has been proposed. Turing postulates that a machine is intelligent if a human interrogator (communicating over some textual device) cannot distinguish a computers' answers from a humans' answers.[2] Other researchers, such as Beer [1990], define AI as the ability to adapt behaviours to some environment. Others again, such as Brooks [1986] have more humoristic definitions, such as " when nobody has any good idea of how to solve a particular sort of problem (e.g. playing chess) it is known as an AI problem.". A professor at a introductory AI-class here at Norwegian University of Science and Technology (NTNU) concluded that AI could perhaps be defined as "whatever AI-scientists are working at". Because definitions differ, and uncertainty isn't desirable in a masters thesis, I have settled on my own fixed definition of AI. It is not meant to be anything revolutionary, but should serve as a means of evaluating the performance of the applied methods in this thesis. It is a mixture of the definitions above, an it goes like this: *AI is a machines capability of acquiring knowledge and skill, and applying this knowledge and skill to new problems*. So it emphasizes an aspect of intelligence that can be observed in any animal from tiny insect, burnt cat or sapient human: The ability to *generalize* knowledge gained by experience so that the it can adapt to new problem situations.

Definition: AI

1.2 Navigation and minimally cognitive behaviour

Up until the 1980ies, most work in AI was focused around building AI systems that would store increasing amounts of information in some internal memory representation. The idea was that proper manipulation of these (symbolic) knowledge representations would eventually result in intelligent behaviour if only the information was accurate enough, and the procedure for storing it was efficient enough. An artificially intelligent robot could then for example apply logics to its internal world-representation, and deduce exactly what to do in a given situation using standard mathematical

[2] See Luger [2002] for a more complete discussion of the Turing-test.

deduction. During the 1980 some researchers begun to question this strategy of achieving intelligent behaviour. Most notable of the researchers were (arguably) Rodney A. Brooks. In his papers *A Robust Layered Control System For A Mobile Robot* and *Intelligence without representation* ([Brooks, 1986] and [Brooks, 1987]) he suggested a major paradigm change for AI. Brooks idea was that the complex behaviour that real intelligent creatures exhibit, does not necessarily result from complex internal *representations*. Complex behaviours may equally well be a result of interaction with a complex *environment*.

Terminology: layered approach, incremental, subsumption, sub symbolic

Brooks suggested that AI therefore should move away from pondering with different representations, and rather rely on layered interfaces to the real world. He claimed that the best path towards creating complete AI systems, would be to incrementally - part by part, that is - build representation-free systems in a bottom-up fashion. One should start with the simplest possible behaviours first, composing the lowermost, easy layer to begin with. For a walking robot, the bottom layer might be responsible for aimless walking. That should be an achievable task. Later on, when the walking behaviour was sufficiently developed and tested, new layers could be added in top of the existing one. The new layer could provide for example planning capabilities, that could enforce walking in some particular direction. Even more layers could be added on top of this layer, making the robot capable of avoiding obstacles, or communicating with other robots. By continuously adding new layers of more and more advanced behaviour, whilst continuously testing each layer extensively in the environment that it is to perform in, higher level intelligent behaviour should ultimately emerge. Brooks named this strategy the subsumption architecture, as higher layers *subsume* lower layers. The field in itself is often referred to as *sub symbolic* AI, because it goes "below" manipulation of symbolic representations.

It will not be claimed here that a fundamentalist cling to Brooks strategy is the only way of proceeding when building artificially intelligent computer systems. Perhaps such systems cannot even be created this way, because at least some representation is necessary. Nevertheless, much can surely be learned from parts of it, as the subsumption architecture does not suffer badly from the problem of ever growing internal representations. It has also been shown that animal-like behaviours in mobile robots (including navigation behaviours) can actually be made up from the interaction between (and subsumption of) several simpler behaviours (see e.g. Beer [1990], which is described in more detail in the next chapter). All this without much inter-

nal representation. Systems resulting from such strategies typically have the advantages of being small, clear and having a comprehensible configuration. They provide an easy way of splitting cognition into manageable parts, each part possessing *minimally cognitive behaviour*.

Terminology: agent, environment, task, reactive, deliberative

In AI-agents parlance - which will be much used throughout this thesis - a navigating mobile robot is referred to as an *agent* that perform navigation in an *environment*. Navigation is said to be the agents *task* in this case. An agent strictly following the paradigm of Brooks, is called a *reactive* agent, since all its actions are direct *re*-actions to sensory input. An agent following the Good Old-Fashioned-AI (GOF-AI) paradigm is referred to as *deliberative*, since its actions rely on the result of deliberation (on an internal representation), and only indirectly on sensory input. When creating navigation systems close to a reactive paradigm, it is to be expected that they require varying degrees of memories. This will depend on the degree of deliberation required for the navigation system to perform well. For a robot trying to navigate down a crowded corridor, it could for example be of great use for it to remember the placement of stationary obstacles. There is no reason for it to run into one of the hospital equipment more than once!

The amount of representation that a navigation system require, depend somewhat on the type of sensor used. Sensors range from simple gyroscope readers, through sonars, Doppler sensors and complete visual perception systems [Borenstein et al., 1996]. The most prominent of these is probably the last-mentioned one. For this reason it may be tempting to choose visual sensing when designing mobile robots. The advantages of vision can be easily found in nature where many animals use some form of visual sensing system. Typically, such systems allow them to perceive objects at a distance, and this undoubtedly allows for very proactive behaviour: A bird might catch sight of a predator very early, and then plan how to avoid it before *real* threat is encountered. As Borenstein et al. [1996, p. 207] point out, visual sensing "is potentially the most powerful source of information among all the sensors used on robots to date". Much would have been achieved if machines like the hospital robots could make use of some of these advantages.

Looking back to the subsumption architecture it is not apparent, however, that vision would be the ideal "bottom layer" of intelligent behaviour. Indeed, some AI systems based on visual perception have emerged in the later years. Autonomous vehicles, traffic monitoring systems and aerial navigation systems are some examples of real systems that do real work (see [Clark,

2005] for an overview). But do these systems provide a solid foundation for further scientific research? Do they represent proper "bottom-layers" for more complex AI-systems? Although they prove that artificial visual sensing have some value in real AI systems, this does not necessarily imply that other sensory systems should be disregarded in all circumstances. The cost of highly proactive behaviour is extended deliberation, implying that the above mentioned advantages of low system complexity, high clarity and comprehensibility will suffer. From the view of someone trying to understand the workings of navigation, the simple rather than the complex model is to be preferred. A simple model should be easier to create *and* to test extensively in the real world, because the number of variables in it is minimized. We should therefore be careful when choosing where to start the study of minimal cognitive behaviour.

1.3 The tactile sense

It should be evident from the previous section, that visual sensing is an example of a perception technique that is probably best avoided when studying navigating agents. It is expected to both require much representation, and to be very complex. Following the line of Brooks, this thesis suggest iterating through other perception techniques, and start with one where deliberation is sparse, and complexity is low. Preferably, the simplest one that is still complex enough to be interesting, should be chosen.

An obvious place to start the search for simple navigation systems in nature, would be in the simplest known animals. Insects, that is. If the "simple" behaviours of such animals can be understood (or at least modeled), they may be both applied to real-world AI-systems directly, and they may help understanding (or modeling) higher level behaviours successively. Insects came along rather early in the evolution of life on earth - about 450 million years ago [Brooks, 1987]. Assuming that simple life forms arrived before complex life forms, much about simple systems could be learned from the workings of insects. Despite their simplicity they still adapt remarkably well to complex and differing environments. If a cockroach looses a leg, for example, the walking gait is immediately adapted to the new body configuration, and it continues to walk using a new stable gait [Beer, 1990]. Many other examples of adaption are to bee seen every day. An insect flying around before landing on water or perhaps a wall, before effortlessly walking upside-down,

is one. Creating an artificial system exhibiting all these behaviours would be quite an achievement, even though insects are usually considered lower level animals.

As will be outlined in the subsequent chapters, one reason for success within such a wide range of behaviours, seems to be that elegant physical body layout is strongly intertwined with cognitive processes. Highly realistic simulations of six-legged insect walking, for example, can be achieved through the training of a simple ANN. Such techniques have been shown to yield a well-performing (in terms of robustness and similarity to real insects) reactive sensor-motor system in Cruse et al. [1998]. As some researchers have pointed out, there seem to be a connection between six-legged walking and the tactile sense ("touch") and proprioception ("body awareness") in at least some insects. Specifically the walking gait is related to sensors reacting to foot-position (Cruse et al. [1998] and Beer [1990]) and also between the movement pattern of active tactile sensors and leg position [Dürr et al., 2001]. The insects can therefore generate stable walking gaits using very simple cognitive functions, almost in a completely reactive manner.

Compared to other sensor techniques such as audition or vision, the tactile and proprioceptive senses may not seem very impressive at first glance, and this is perhaps one reason that comparatively little research have been made on them. According to Dürr et al. [2001] the tactile sense is probably the least studied of all insect senses. The success of artificial walking with reactive agents and the close relation between walking and tactile sensing, does however suggest that the tactile sense may be a better candidate for a "bottom layer" of navigational research than other senses. Its seemingly simple impression, simplicity, becomes it's largest advantage. Existing research has shown that (close to) reactive mobile robots using passive tactile probes can exhibit impressive cognitive features. By "subsuming" an artificial insects walking controller, Beer [1990] showed that navigational behaviours like turning, recoil and gradient following in search for simulated food-patches, were indeed possible.

1.4 Neural networks for robot control

Artificial Neural Network (ANN)s have become popular for controlling autonomous agents. ANNs are loosely based on real neural systems. The idea

is to use machines to simulate the processes that take place for example in the human brain or spinal chord. It is of course a bit ambitious trying to simulate a complete human brain, but the study of even small ANNs have shown that they can achieve impressive things using only very sparse amounts of computer power [McLeod et al., 1998]. They perform particularly well for problems where the task is to estimate a set of output values based on a set of uncertain input values. For robot control this means estimating proper motor force (output values) from sensory readings (input values). For a simple robot driving down a corridor, the inputs could be for example range sensors measuring the distance from surrounding walls. The robot must then find out how much force should be applied to for example right and left motors, to ensure crash-free driving down the corridor.

Terminology: neurons, synapses

Conceptually, an ANN can be seen as a number of separated, but interconnected processing units. The parallel to real neural networks, is that the processing units correspond to brain cells, or *neurons*. The connections between the processors correspond to real world *synapses*. Some of these processing units can be connected to robots sensors, and other processing units can be connected to motors. The interplay of all these units - by passing messages over the connections - can then result in behaviours like crash-free driving, if the correct neurons are interconnected by adequate connections.

There are two features of ANNs that make them especially interesting for the research in this thesis. They provide a solution to the two major challenges that were introduced in the previous sections: Building representation less AI system bottom-up (incrementally, that is), and handling unforeseen events. It is easy to see why this is true, by considering the hospital robots:

For the robot, there is always a potential risk that one of the range sensors provide erroneous data. Even worse, it may report slightly wrong values only part of the time. Mostly, the sensor is fine. This can be a problem, because the robot may seem fine during testing. Then, at the most unfortunate time, the sensor fails (perhaps due to inference from other robots) and the robot crashes into some critical hospital equipment. Because ANNs distribute processing across many neurons, the one, failing, sensor need not result in a complete system crash. It is the calculations of *all* processing units, and *all* sensors that result in an approximate set of outputs from the network. A short sensor disruption may not even be noticed at all on the *overall* results of the ANN. The result is a system that is robust to at least minor

perturbations.

The other problem, building AI systems bottom-up, is a more complex issue, and is a major focus of this thesis. A random set of neurons interconnected by random connections, does of course not result in a working robot. Actually, if connections have real-valued strengths, there are an infinite number of ways to interconnect any set of neurons. This makes it very difficult to design ANNs manually, even when starting with very small "bottom layer" ANNs. As the ANN grows larger it usually gets less and less apparent what purpose a single neuron or connections has. A human designer will in such cases only be able to create a rough approximation of network topology. Something like "Each sensor must have one sensor neuron, and each motor should have a motor neuron. In between these, there should be some other neurons...", is more realistic. Luckily, several training techniques have come into existence to remedy this problem. Some techniques are back-propagation, Hebbian learning and genetic algorithms. The last mentioned technique is the primary focus of this thesis.

1.5 Genetic algorithms and incremental evolution

Nature seem to have had success in creating neural networks like the human brain through evolution, and it would therefore be likely that something about creating artificial neural networks could be learned from that process. Genetic Algorithm (GA)s are loosely based on Darwinism, and provide a form of artificial evolution. The algorithm is conceptually similar to its real-world counterpart. In a GA, several (pseudo)random solutions to a problem make up *individuals* in a *population*. Each such individual carry a *genome*[3] that describe its solution to the problem. Two or more such genome can go through *recombination* to see if parts of their solutions - single *genes* - can be combined into a even better complete solution. Typically, the solutions that perform best with regard to a *fitness* measure are *selected* to be recombined. By continuously replacing poor quality solutions with new well-performing solutions, recombination moves the population from a set of random performing solutions to a set of better performing solutions.

Terminology:
individual,
population,
gene,
genome,
recombination,
fitness,
selection,
mutation,
local/global optima

There also exist *mutation* operators, making small changes to the existing

[3]Animal genome are Deoxyribonucleic Acid (DNA).

genome. Mutation provide a means to introduce new subproblem solutions that did not exist in the initial population. GAs are vulnerable to getting stuck in so called *local optima*, where performance of the population is getting very good, but is still not completely optimal. In a local optimum, all of the genome will be very similar, and the success of recombination will slowly begin to halt. This happens because the best performing individuals are all near the local optimum, but not very different from each other. Recombination of two similar individuals does not result in individuals very much different from the rest of the population, and change towards the better is therefore difficult. Mutation can help overcome this situation by making new recombinations sufficiently different from existing genome, by means of randomness. The population may then move forward towards better solution spaces that (hopefully) contain a *global optimum*.

It should be evident that GAs provide an interesting way of creating ANNs. Genes can correspond to network connection strengths, and other variables of the network. By testing each network description (individual or *genotype*) by creating a ANN (the corresponding *phenotype*) placed in a simulated robot, it should be possible (given lots of time) to evolve a proper neural network. For the evolution of cognitive behaviour in artificial life, the size of the neural networks to evolve may however be very large. Due to the time required by current computers to evaluate the fitness of a certain network, only smaller networks can be evolved. In nature, several individuals of a population are "evaluated" in complete parallel over several millions of years. This is opposed to the situation that the researcher finds her/himself in. Simulations are usually only partially in parallel, and computation power is (always to) sparse. The precision of the simulation environment is some discrete value - because computers are discrete - rendering the real-world quality of a evolved creature questionable. Not least, a researcher do not want to spend a few million years for a simulation, only to find out that "Murphys law" had been maintained. For these reasons, techniques to speed simulated evolution has become of great demand.

One strategy for evolving animal-like behaviour faster, is to cut the task at hand into smaller subtasks. This is much the same spirit as that of Brooks layered control architecture. For the evolution of legged walking, for example, typical requirements are that an individual is 1) capable of moving its legs, 2) that it is able coordinate this movement between several legs, and 3) that it has some Central Pattern Generator (CPG) that starts off movement of each leg at some interval. The probability that a population

of random individuals contain one or more outstanding individuals with *all* these behaviours at the same time is very low. Applying recombination on such a population is futile, since it is impossible do decide which creatures to select for recombination. Only by selecting parents among the really well-performing individuals (which requires knowing which individuals are well-performing), can recombination be effective. The probability that one of the *subtasks*, for example just moving the legs, exist in the population is of course much higher.

Incremental evolution (exemplified by Gomez and Miikkulainen [1997]) provide the necessary means for dividing a task such as walking or navigation into subtasks. Just like in Brooks layered architecture, simple behaviours may be evolved first, and then new behaviours may be added on top. Because inputs to neural networks can be outputs of other neural networks, it is very easy for one network to later "subsume" another. The most striking difference between incremental evolution is that any evolved subtask may be changed as new behaviours are evolved later. Old solutions may therefore adjust to being subsumed, even though they are "extensively tested in the real-world". In the following chapters, incremental evolution, together with other techniques will be described in detail, and applied to the problem of evolving navigation.

1.6 Summary

- AI can potentially enable robots to work reliably in unpredictable (dynamic) environments.
- This thesis follows the tradition of Brooks [1986]: AI is an emergent property of many (small) interacting modules.
- Artificial Neural Network (ANN)s are ideal for such modules, but they are not straight forward to design.
- Artificial Neural Network (ANN)s can be created with the help of Genetic Algorithm (GA)s, but GAs have problems with things like local optima, and are generally computationally expensive.
- This thesis provides experimental results from trying out strategies for helping GAs to evolve AI:

- Incrementally evolving complex behaviour from several minimally cognitive behaviours (bottom-up design).

- Choosing proper behaviours to evolve in this process (as simple as possible, but still complex enough to be challenging for a GA).

- Choosing proper level of detail for the representation for the ANN to evolve.

1.6.1 Further reading

For a summary of biological based approaches to robotics, see Beer et al. [1997].

Chapter 2

Background

As outlined in the introduction, this thesis looks into techniques for efficiently evolving higher level cognitive behaviour from many minimally cognitive behaviours using Genetic Algorithm (GA)s. Tactile-olfactory navigation is proposed as a proper start-off point for a bottom-up evolutionary process, leading to gradually more complex cognition.

Many researchers have already studied minimally cognitive behaviours, and the interplay of these. Much work has been done in manually creating ANNs or similar distributed controllers. Other work has been done on partial design of ANNs, that are later trained (e.g. using back-propagation) to carry out a specific task. Since about the mid 1990ies, major focus has been brought to the study of processes that can automate the design of distributed controllers. A brief outline of relevant work in these areas, and the major challenges of each, is given in this chapter.

2.1 Work on neural networks

2.1.1 Standard Artificial Neural Network (ANN)s

Real animals use neural networks for calculations: A brain is an aggregate of interconnected neural cells called neurons. Neural networks are robust

to perturbations such as random cell death, minor chemical interference or faulty sensor readings. Because the calculations of a complete network depend on many neuron cells, a single faulty neuron has only limited effect on the total result of the network. Neurons also work in parallel. This means that complex computations can be carried out very fast without any single neuron having negative effect on the overall speed of the system. Features like this, make ANNs attractive for the control of mobile robots.

A common way of simulating neural networks using computers, is to map each neuron into some small computational unit (such as an object and related methods in an object-oriented programming language). The unit usually holds information about two things: The neurons' current *activation level*, and the neurons' *synaptic connections* from other neurons.

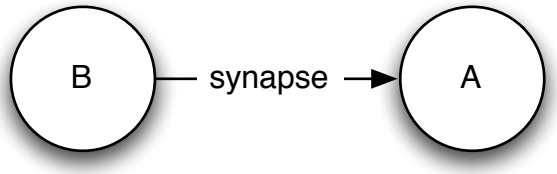
A neurons activation level is a measure of a neurons' state, or degree of "activity", and can be represented using a real value. The higher neuron activity, the higher the value is. This neurona-activity can in turn influence the activity of other neurons by sending messages over synaptic connections. Synaptic connections are often called synaptic *weights* in ANN literature (e.g. Beer [1995]) and the value of the weight decides how much one neuron can affect another. This is analogous to the strength of real synaptic connections. A very simple example ANN is shown conceptually in figure 2.1a, on page 15, where two neuron units, A and B, are connected using a synaptic weight. Here, the activation of unit A depend on the activation of unit B, since there exist a synaptic weight *from* A *to* B.

Terminology: activation level, weights/synapses

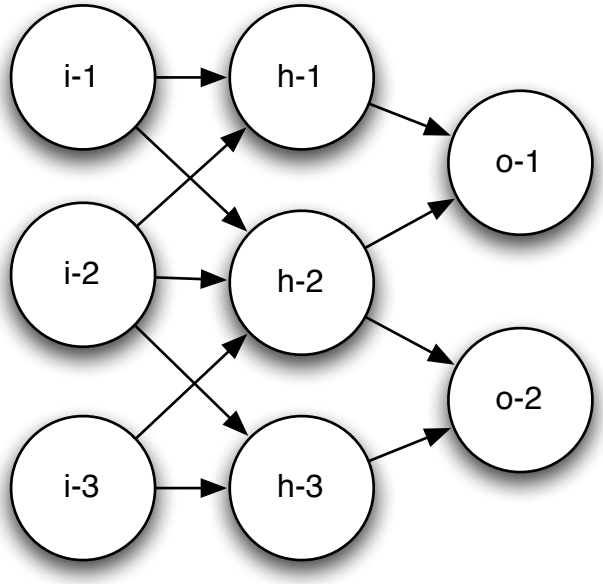
The simulation environment is usually discrete, so time is partitioned into small steps. The activity of each neuron unit is computed at every such time step, taking into account each neurons' incoming connections. The most common way of doing this calculation is to sum all the input connections, and normalizing the sum into some range (typically the range [0 1]). The value of every input connection is calculated by multiplying the weight value by the connected neurons' activation level, a process quite similar to the summing that goes on in real neurons.

In addition to be influcenced by other neurons, an external input such as a sensor reading can also affect a neurons activation. Neurons that depend on external input are often referred to *input neurons*. Corresponding *output neurons*, are those whose activation level can be interpreted as a network output. The network can therefore be considered a mathematical function,

Figure 2.1: Example networks.



(a) Simple network, where the activation of neuron A depend on neuron B.



(b) A more complex (feed-forward) network where the activation of the output neurons depend on the hidden layer, which again depend on the input layer.

mapping a set of input values to a set of output values.

A common way of arranging neurons in a network, is to connect them in a layered fashion. In such networks, computations are done in stages. One layer may pre-process sensor input before it is sent to the next layer[1] The layer of input neurons only have connections from sensors, and is called the input layer. Another neuron layer has synaptic connections from the input layer neurons, and therefore depend only indirectly on sensors. Several successive layers may be added this way, so that all neurons always connect to neurons in the previous layer. The last layer added, makes up the the output layer (an example network is shown in figure 2.1b, page 15). Multi layered networks can compute complex mappings between a large set of input and output values very fast, because of the simplicity of each neurons activation function, and the aforementioned advantage of parallel processing.

The layered topology of these networks, is also the basis for how activation is calculated in all the networks' neurons. The process is often referred to as spreading activation, and it involves starting to calculate the activation of the input neurons, and then spreading the resulting activation forward through each layer of the network. A typical spreading of activation goes on like this:

1. In figure 2.1b the activation of neurons $i-1$, $i-2$ and $i-3$ are calculated first. This is done by applying the sum of any external input to the neurons activation function. The sigmoid function (see figure 2.2) is often used as an activation function (e.g. Beer [1995]), always yielding an activation level in the range [0 1].

2. When all input neurons have been activated, the next layer (neurons $h-1$, $h-2$ and $h-3$) is considered. The activation calculation performed here is the same as for the input layer, except that external inputs do not exist. Instead, synaptic inputs from the input layer are used. The value of each synaptic input is calculated by multiplying connection strength with the connected input neurons activation.

3. When all neurons in the hidden layer have been activated, the same activation process is performed for the output neurons.

[1]This also happens in real life, for example in the human brain. Sensory input from the eyes retina is preprocessed by the visual lobe before sent further on. [McLeod et al., 1998].

4. The activation of the output neurons are finally read as the network output.

Networks that spread activation from inputs, always feeding activation forward towards the output neurons are called *feed forward* networks. Very large feed forward networks can be created without much hassle, and complex mappings between a set of inputs and outputs can be computed with relative ease.

A typical complete setup for an ANN controlling a simulated agent is shown in figure 2.3 (page 15). Sensors are connected to the networks input neurons, and output neurons connect to motors. A simulated agent cannot usually percieve all of its environment, and sensors are sensitive to environmental noise or may even fail. Because ANNs are relatively robust to uncertain or noisy inputs, they can still perform quite well under these circumstances. At each step in time the ANN can calculate an approximate set of motor forces depending on sensor input. Small errors in sensory readings does not matter much, because a single neuron play only a minor part in the complete calculation. The agents *reaction* to a given environment is therefore always a good estimate to the best possible action to perform.

2.1.2 Extensions to the standard ANN

While the simplest ANNs perform very well for reactive agents, they are not that suitable for deliberative agents. If neurons do not remember the activation from each time step, there is no way to keep state in them. This means that the output always depend directly on sensory readings. A common addition of such networks is therefore to allow neurons to have synaptic connections either to themselves, or back to previous layers. Such networks are called *recurrent* ANNs. By maintaining a state for each neuron activation, one network activation can then influence a later activation. This allows the agent to maintain state, and thereby allowing deliberative behaviour.

In addition to allowing recurrent connections inside ANNs, it may be interesting to make time a factor that influence the state of each neuron. In time dependent ANNs, a neuron may have to be activated several times before the activation level rises significantly. This can help an agent ignore sensory

Figure 2.2: The sigmoid activation function, $activation(t) = \frac{1}{1+e^{-t}}$, used by many ANNs). The sum of neuron inputs are applied to this function, and then mapped into a neuron activation. Here, the saturation toward 0 and 1 is exemplified by visualizing the inputs from -10 to 10 to the function.

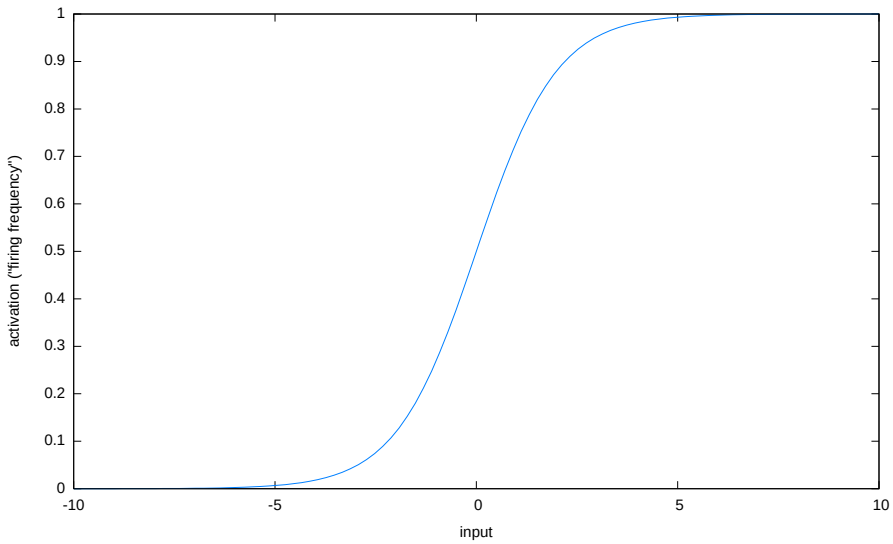
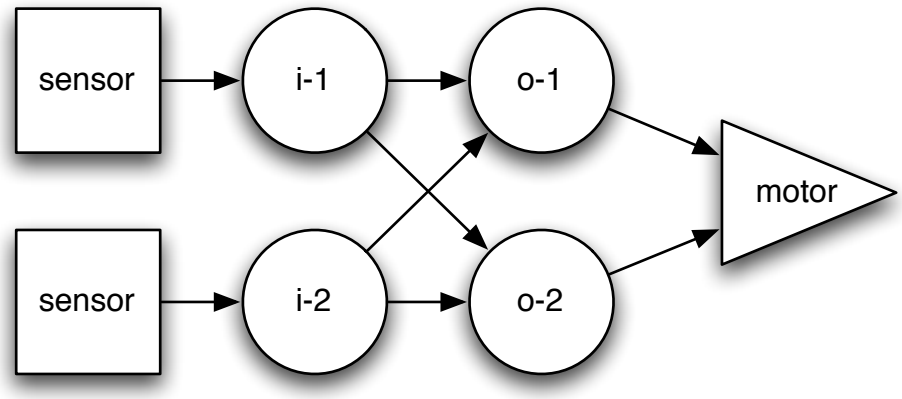


Figure 2.3: A complete sensor-motor setup. Motor force is a function of two sensory inputs.



input until it has been active for a prolonged amount of time, for example. It also allows for short term memories, since a memory (represented by a certain network state) may fade over time.

Continuous-Time Recurrent Neural Network (CTRNN)s are, as should be obvious from the name, capable of both recurrent connections and time-dependent calculations as described above. In a CTRNN each node of a network keep two values: 1) A bias (θ), and 2) a time-constant (τ). The bias may be regarded as a variable deciding how sensitive the neuron is to connections from other, neurons. The time-constant decides how fast (time relative) a neuron should change its state ($\dot{y}$). As described above, the state of the neuron is stored between every time the state of the network neurons are updated (every time step). At each update, the neuron state change for each time step is calculated. Given that each incoming connection (or weight factor) to a neuron is named w_{ji}, the equation governing the state of each neuron is:

$$\dot{y_i} = \frac{1}{\tau_i}(-y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_j + \theta_j) + I_i) \tag{2.1}$$

where the function σ is the standard logistic function from figure 2.2:

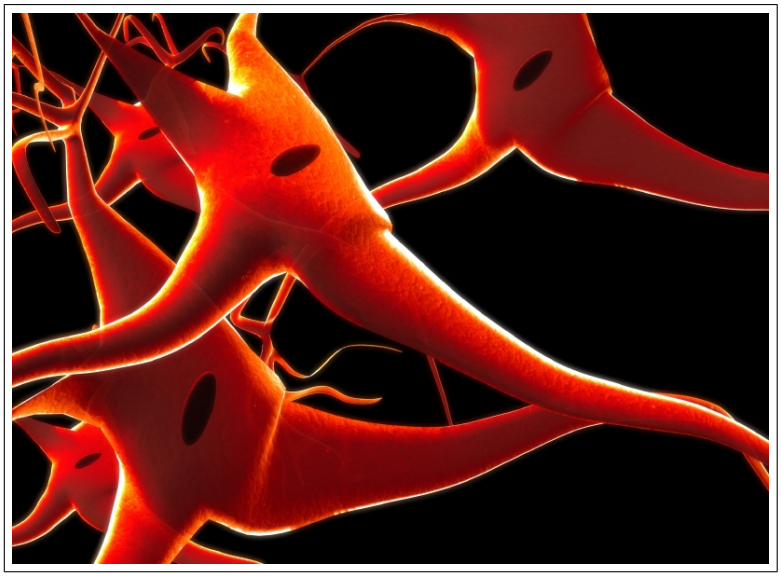
$$\sigma(t) = \frac{1}{1 + e^{-t}} \tag{2.2}$$

and I_i is an external input to the neuron, such as a sensor reading.

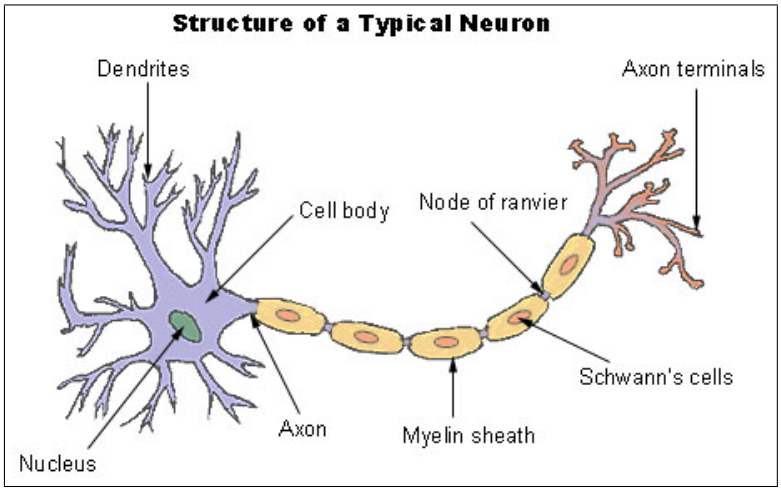
In the current implementation this equation is solved discretely using Euler's forward method for differential equations. This gives an approximately correct solution, and the resolution (correctness) of it can be adjusted by setting the integration time step to an adequate value (currently 10 milliseconds). The most crucial factor here, is that the integration time step is smaller than the time-constant of the neurons. Otherwise neuron state may change faster than the integration resolution will indicate, and unpredictable behaviour may result. Therefore, all time-constants are kept higher than the integration time step in this thesis.

If CTRNNs are allowed to be infinitely large and complex, they can in

Figure 2.4: Neurons. Real and simulated.



(a) Real neurons
(Picture taken from URL
http://serc.carleton.edu/images/cismi/neuroscience/neurons.jpg (last visited 25.04.2007). Following the Copyright on URL
http://serc.carleton.edu/serc/terms_of_use.html
(last visited 25.04.2007))



(b) Morphological characteristics of real neurons
(Picture taken from URL
http://training.seer.cancer.gov/module_anatomy/unit5_2_nerve_tissue.html
(last visited 25.04.2007))

theory (approximately) simulate any dynamic system. They are arguably the simplest such model available [Beer, 1995]. Due to their simplicity, such networks are relatively easy to evolve (as compared to e.g. the leaky-integrator model used in Beer [1990]) because the number of interdependent variables are minimized. Only two variables are to be evolved for each neuron, and only one variable for each connection. There is only one type of connection in this model, and the connection may be either inhibitory (negative) or excitatory (positive) or effect-free (zero).

Terminology: Dendrites firing-frequency, membrane potential,

The connections between neurons are referred to as synapses, just like for standard ANNs. Further, the list of such synaptic connections into a neuron are referred to as dendrites (connections points), and the activation of each neuron is referred to as firing-frequency. Finally the state of the neuron is referred to as membrane potential. This should clarify the possible biological interpretation of the model.

Except from the advantages of the CTRNN model as outlined above, a few other factors make such networks attractive for the implementation of insect nervous systems: CTRNNs can be used for associative memories [Beer, 1995]; a CTRNN is also similar enough to the leaky-integrator model used by Beer [1990] that much of those results can be used here more or less directly. Especially the locomotion controller is interesting here; much other research have been put into figuring out the dynamics of CTRNNs, meaning that other research to build upon is readily available.

2.1.3 Walknet - Training a manually designed ANN

General idea and design

Walknet [Cruse et al., 1998] is a computer system that simulates insect walking. The system was developed at the department of Biology, University of Bielefeld, Germany. As may be guessed from its origin, the system is not primary a study of how to develop an AI-device. Rather, its goal is to create a scientific tool for the study of insect walks. Nevertheless, the system is of great interest also for the study of cognitive behaviours, because it is in fact an autonomous, decentralized, adaptive (simulated) robot system capable of moving around in dynamic environments.

Walknet is a modular ANN, and each module is in itself an ANN. Every of the artificial insects six legs are controlled by one separate ANN. They are therefore distinguishable from the rest of the neural system. All the leg modules are interconnected, so they can interfere with each other and coordinate into stable walking gaits. There are some conceptual similarities to Brooks layered subsumption architecture, but there is not really any hierarchy of "subsuming" modules (they can rather be viewed as cooperative). Each leg module in Walknet consist in itself by smaller cooperative networks. There are for example one ANN responsible for controlling the leg during leg stance (providing support and propulsion for the insect), and another network responsible for swinging the leg forward. A third net (the selector net) is responsible for choosing whether the stance or the swing net should be in control of the leg motors. That is, whether a leg should be moved forwards or backwards.

Being a reactive sensor-motor system, Walknet decides what to do based on sensor input. To achieve the leg swing movement, for example, the swing net takes as inputs three coordinates representing the current leg position and three coordinates representing the desired leg position at the end of the swing. The output values that result from this represent three angular velocities. One for each of the legs three joints. The actual leg angles are continuously measured, and fed back into the net as the swing progresses. The selector net will eventually turn the swing net off (and the stance net on), based on a sensor that registers when the leg has reached its Anterior Extreme Position (AEP) (the position where the leg is maximally backwards). When six such legs are interconnected, the insect is capable of walking.

Results and possibilities

Walknet exhibits several interesting features seen from an AI-perspective. First of all it is an actual implementation of a working system. The simulated walking is hard to distinguish from the walking of a real insect[2] (unless you are I biologist, I suppose). What is even more interesting is that the system is robust to perturbations, such as legs crashing into obstacles for example. This allows the artificial insect to walk steadily even in highly

[2]Example videos are downloadable from the URL http://www.uni-bielefeld.de/biologie/Kybernetik/research/walk_results.html (last checked 03.04.2007).

dynamic environments or environments that were not presented to it during training. In other words: The system *generalizes* its *specific* knowledge (limited knowledge acquired during training) to fit new situations. This type of generalization is a key feature of an AI-system, when considering the initial definition of AI in the introduction.

Cruse et al. [1998] mention some minor drawbacks (minor from an AI-perspective, probably of larger concern for the biologists) of their system. Leg amputation does not result in gait transition (as observed in real insects by Graham 1977). Walknet also has some problems separating AEP sensing from obstacle sensing when swinging the legs forward, making some walks unrealistic. There are also some discussion of whether the ANNs used for simulation are similar enough to real neural networks that any comparison between the two is defensible. Real neurons do not simply sum their inputs, like the neurons in Walknet do. There is a lot more complexity in real life.

Of larger concern from the AI-perspective is that the network is created by training a hand coded network with empirical data, using back-propagation. That means that a set of empirical data is required for training the network (= adjusting the network weights) until it performs close (enough) to a real system. Such data is not always available. It would be better if the system was told *what* (not *how*) to do, and then learned the "how"-part by it self. Then we wouldn't need the empirical data. A side-effect of training an ANN with empirical data is also that the ANN may fit too much to the training data. This problem is known as overfitting, and may result in bad performance on any other environment than the training environment. Over fitting is a known drawback of supervised learning algorithms such as backpropagation.

2.1.4 Intelligence as Adaptive Behavior

General idea and design

In his book [Beer, 1990], Randall D. Beer suggests an alternative view of intelligence, exemplifying his claims using a simulated insect. His view is alternative with regard to what was at the time the prevalent AI methodology. He claims that the traditional view, copying the intelligent behaviour of humans, has been to heavily focused on deliberative reasoning. He therefore

suggest *Adaptive Behavior* as a more appropriate definition of intelligence. Beer argues that [Beer, 1990, p. 11]:

> *(...) it is adaptive behavior, the much broader ability to cope with the complex, dynamic, unpredictable world in which we live, that is, in fact, fundamental.*

Throughout his book, the artificial insect - *Periplaneta Computatrix* - is used to exemplify how intelligent behaviour can be realized though adaptive behaviour. P. Computatrix is a model of a real cockroach (Planeta Americana), and it is built up in a stepwise fashion. By manually designed dynamic neural networks, walking behaviour is implemented. The walking network is later extended (subsumed) by new neurons to make the insect capable of more complex behaviours such as turning, edge-following and recoil. Many behaviours which form the basis of navigation.

Neural model and the locomotion controller

Among the findings in Beer [1990], the P. Computatrix locomotion controller is of great interest for this thesis. The insect walks by means of a sensor-motor system that depend on two factors: 1) time and 2) tactile (or perhaps more correctly proprioceptive) sensor input. This is interesting because, as found by Dürr et al. [2001], there are temporary connections between the leg movements and antennal movements of real insects such as the Carausius Morosus.

The system that lies between the sensors and motors in Beers' insect, is a recurrent, time-dependent ANN, based on the leaky-integrator model. Just like for CTRNNs, the leaky-integrator model calculates neuron activations based on a membrane potential, not just sums of input. The membrane potential is adjusted every 5 msec, and is increased or decreased by some proportion to the sum of neuron inputs. Additionally, for each time step, a "leak-current" is subtracted from the membrane potential, so as to simulate the conductance of real cell membranes. The resulting ANN model, according to Beer, is modeled so that it "(..)strikes the proper balance between the complexity of biological nervous systems and the requirements and constraints of our simulation." Beer [1990][p.49]. By addition of so-called *intrinsic currents* (similar to self-weights of CTRNNs), he describes how the

neural model can be used to create neurons capable of pulse-behaviour. This pulse-behaviour is central to the P. Computatrix locomotion controller.

Based on Pearson ([Pearson, 1976a and Pearson et al., 1973] in Beer [1990]) the pacemaker is used to create a locomotion controller capable of statically stable hexapod walking. The ANN topology is shown in 2.5. The basic idea is as follows:

1. A singe pacemaker neuron (P in figure 2.5) generates rhythmic bursts.
2. The stance (determining propulsion force) and foot neuron (determining down pressure of leg) is usually active.
3. When the pacemaker fires, the stance and foot neurons are inhibited. Lifting the leg, and ensuring that it does not provide propulsion.
4. When the pacemaker fires, the swing neuron is exited, and therefore the leg is moved forward.
5. By the rhythmic nature of the pacemakers, the leg is capable of stepping rhythmically.
6. To properly time the transition between stance and swing phases, proprioceptive sensors may interfere with the stepping cycle:

   - When the leg is in its AEP, swing is inhibited and the pacemaker is reset. The foot is pressed down, and stance begins.

   - When the leg is in its Posterior Extreme Position (PEP), pacemaker is reset (exited to start a new cycle).

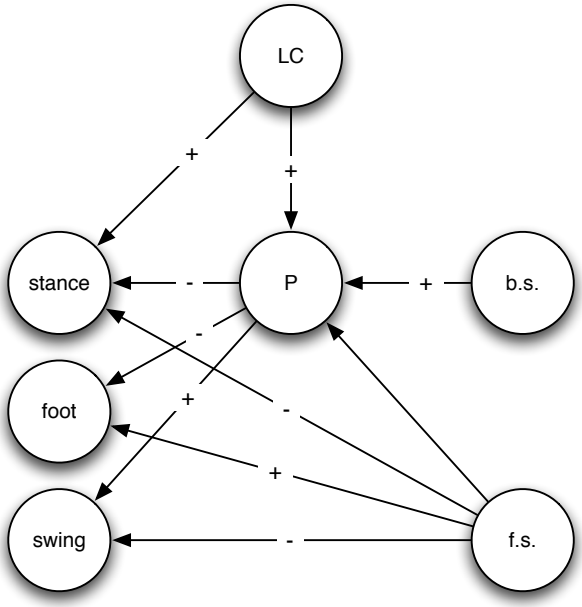
   - These tactile sensors fine-tune the centrally generated stepping rhythm.

To coordinate the stepping of all six legs, all pacemakers are interconnected by inhibitory connections, so that adjacent legs are discouraged from swinging at the same time. Thereby generating statically stable gaits.

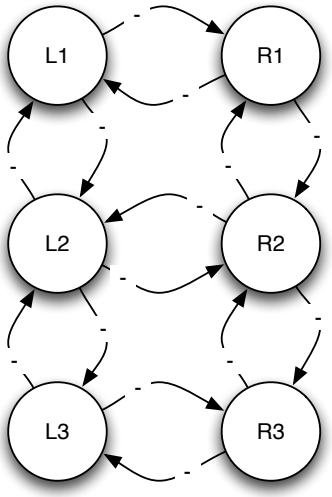
Results and possibilities

The locomotion controller reliably produces successful walking when embedded in the P. Computatrix body. Beer also stresses that the controller - in

Figure 2.5: The leg and locomotion controller of the P. Computatrix.



(a) The leg controller, each leg has one such net. 'b.s.' and 'f.s.' are backward and forward angle tactile/proprioceptive sensors. 'P' is a pacemaker cell, and 'LC' is a special controller neuron that is shared by all leg controllers.



(b) Inhibitory connections of all the leg controllers pacemakers.

addition to producing stable gaits - reproduces a wide range of insect like gaits. He claims that the design was never motivated by a desire to reproduce this range of features from insect locomotion. The similarity with a real P. Americana was therefore an unexpected result that he owes to the close attention to biological details.

From the view of evolving a navigation system, the locomotion controller also exhibit some other interesting behaviours. First of all it is an example of a working sensor-motor system. The stepping pattern is dependent on tactile/proprioceptive sensors, and it is a result of continuous interaction between body (sensor input) and the ANN. Neither would have any value without the other. Secondly, the controller is very robust to perturbations. As shown in the lesion studies of Beer [1990], the locomotion controller is robust to small perturbations such as artificial injection of current into neurons, removing some sensory neurons or removing some connections. Also (as will be used in this thesis) the removal of *all* connections from the LC-neuron did not have any effect on the tripod gait.

The robustness of the locomotion controller makes it ideal for inclusion into a navigation system, because it should be able to work reliably in a wide range of environments. Because it is implemented using a leaky-integrator ANN, it should also be possible to create a similar CTRNN with only limited effort. The fact that the controller is implemented using an ANN is also an advantage. It makes it easy to subsume it with higher-level behaviours by adding new synaptic connections.

2.1.5 On the dynamics of small CTRNNs

General idea

In an effort to understand the dynamical properties of CTRNNs, Randall D. Beer in Beer [1995] did several numerical experiments and elementary analyzes of smaller CTRNNs. His work was specifically motivated by the increasing usage of CTRNNs for modeling nervous systems of autonomous agents. The paper provides extensive insight into many of the dynamical properties of CTRNNs, and one of particular concern in this thesis, is the study of what conditions that must be fulfilled for CTRNN neurons can have several equilibriums. An equilibrium is the "membrane potential(s)" a

neuron will fall into if all inputs are kept unchanged for a sufficient amount of time. Mathematically, an equilibrium point is reached when a transition from one time-step to another leads to no change in the membrane potential. That is when the equation for membrane change, shown in section 2.1.2, evaluates to 0:

$$0 = \dot{y}_i = \frac{1}{\tau_i}(-y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_j + \theta_j) + I_i) \tag{2.3}$$

Results and possibilities

The observation that a single CTRNN neuron can have two stable equilibriums (and additionally one unstable between these two) [Beer, 1995] is based on the fact that if a neuron has a strong enough self-connection, then even in the absence of any external input, it can still self stimulate enough to remain active. This behaviour is natural, if we consider a specific moment in time where a positive external input is removed from a neuron: Naturally, the neuron will not become active *after* that time if it was inactive before that time (there is no positive input present). If it indeed *was* active at that time, it can remain so because (self) stimuli now *is* present. It is shown in Beer [1995] that this condition can occur for neurons with a self connection $w > 4$.

Beers results are interesting for evolving navigation (and possibly many other behaviours) because the presence of more than one equilibrium point in some or all neurons of a network extends the possible steady states of the network as a whole. The network should therefore be expected to exhibit a wider range of dynamical features. Because CPGs can be created by neurons that continuously force each other into changing state [Chiel et al., 1999], they are of particular interest in this regard. When evolving CPGs it should be very likely that neurons capable of forcing each other in this manner would exist more frequently in a population of networks of bifurcative neurons, as each neuron is already capable of falling into several equilibriums. It would therefore be interesting to see the effect of enforcing bifurcative neurons in a GA for evolving CPGs.

2.2 Work on genetic algorithms

2.2.1 Center-crossing CTRNNs for the evolution of rhythmic behavior

General idea

In 2002, researchers Boonyanit Mathayomchan and Randall D. Beer set out to test another of the strategies for faster evolution that were suggested in Beer [1995]. Namely seeding initial GA populations with individuals corresponding to center-crossing recurrent neural networks. The dynamic properties of center-crossing CTRNNs are described in more detail in section 3.1.3. For now it should suffice to say that a center-crossing network is a network where all neurons are maximally sensitive to each other.

Networks that have the center-crossing property are in theory more likely to exhibit a wider range of dynamic properties (including oscillation) than random networks [Mathayomchan and Beer, 2002]. Rhythmic behaviour should therefore evolve faster from a population of such networks. To test this hypothesis, Mathayomchan and Beer used GAs seeded with center-crossing networks for the evolution of a fully interconnected five-neuron Central Pattern Generator (CPG) with a population of 100 individuals, each represented by a vector of 35 real numbers (each neuron has two internal parameters and five weights). The CPG controlled a one-legged creature, whose fitness was measured by the distance walked withing a given amount of simulation time. They then compared several evolutions of the CPG with different mutation probabilities ([0.05 - 3.0]%). For each mutation probability, they observed the performance with and without center-crossing networks in the initial population (all other things being kept equal).

Results and possibilities

Mathayomchan and Beer [2002] showed that evolutionary searches that were seeded with individuals corresponding to center-crossing networks, performed better than random searches for all mutation variances tested. The resulting CPGs in itself performed better, and at the same time evolution was faster. The difference was especially striking at small mutation vari-

ances, where GAs starting with center-crossing individuals performed about twice as good as an equal search initiated with random networks. On average (for 100 evolutions) the performance curve (best performance as a function of generation number) for center-crossing searches also started off higher than the curve for random searches. It also rose much faster, and always showed better performance than the curve for the random networks. From this, Mathayomchan and Beer [2002] read that center-crossing networks were generally better fit for evolving CPGs, and mutation only fine-tuned the walking behaviour for these evolutions. Random networks, on the other hand, were generally unfit, and required mutation to recover from poor initial conditions to perform successfully. This they further supported by a test, where they generated 10000 random center-crossing networks, and 10000 completely random networks: Of the center-crossing networks, 26.6% produced oscillations, whereas for the corresponding random networks, only 1.2% did.

Considering the importance of CPGs in lower level insect behaviours such as walking (e.g. Beer [1990]), it should be apparent that the use of center-crossing networks would be of great value for the evolution of navigation. Looking at the rationale for the increased performance - the wider range of dynamic properties in the initial population - it should in theory be possible to take advantage of center-crossing networks also for other GAs. As Mathayomchan and Beer [2002] note:

> *Since nothing about the center-crossing condition is specific to walking, it is likely that a similar improvement would be found on any oscillatory task. In fact, all other things being equal, our results suggest that seeding evolutionary searches with center-crossing networks may always be beneficial, since a wider range of dynamics is more likely to be easily accessible from a population of center-crossing networks than from a random population.*

It would therefore be interesting to test this strategy also on other parts of the search for navigation behaviours in general.

2.2.2 Incremental evolution of general complex behaviour

General idea

So, how are the advantages of the mentioned strategies for faster evolution best combined? Both the use of bifurcative neurons and the use of center-crossing networks suggest placing an initial population into a fruitful region of a genetic search space. A third strategy that also does this is incremental evolution. In a work done by Gomez and Miikkulainen [1997], the evolution of prey capture behaviour is studied using this technique.

Prey capture is an example of a very general behaviour, compared to for example rhythmic behaviour. According to Gomez and Miikkulainen [1997], simulated evolution of such complex behaviours are very difficult to evolve, because evolution often end up with mechanical strategies that are tied to specific environments. They claim that these strategies are ineffective, do not appear to be believable and generalize poorly to new environments. The reason for this, they claim, is that the general strategy is too difficult to evolve directly.

To support their claims, Gomez and Miikkulainen [1997] evolve prey capture behaviour in a stepwise fashion. The prey capture task consists of (at least) two agents, say A and B, where A tries to capture B, while B tries to escape. The agent to evolve here is agent A, and it is controlled by an ANN. Because agent A have limited sensing capabilities, B may (at least temporarily) go "out of sight" from A, so A must have some memory about where B went out of sight, and must figure out how to approach B again. The overall task is therefore non-trivial.

To solve the problem, Gomez and Miikkulainen [1997] separate the task into easier tasks as follows:

1. Capture of a stationary prey within (short) sensory range is evolved.
2. Capture of a stationary prey, moved a bit further away, is evolved. This is more difficult, because the agent will "die" if it cannot reach the prey (= "food") quickly enough.
3. 2 is repeated two more times, forcing the agent to go further and further.

4. Four more subtasks are evolved. They let the prey move, gradually faster for each task.

The ANN is represented using a set of chromosomes[3], and the resulting population after the first subtask is saved. Between each subtask, the necessary change to these chromosomes are stored instead of storing new chromosomes. This strategy is called delta-coding, and the string representing change from original chromosomes is called delta-chromosomes. All recombination and mutation is applied to the delta-chromosomes, and not to the original ones. This allows each evolution to explore the "neighborhood" of earlier evolved solutions. This also allows further evolutionary stages to continue unhindered of convergence in earlier tasks, because only delta-chromosomes have become converged.

Results and possibilities

Gomez and Miikkulainen [1997] successfully evolves prey capture behaviour using incremental evolution. They also show that a non-incremental approach for the exact same task, fails every time (for 10 runs). On average, the non-incremental approach improves slightly the 20 first (of 200) generations, but then gets trapped in a local optima where not even basic skills have been evolved.

Partitioning a task in this manner could be a very interesting strategy for a complex task such as navigation. As Gomez and Miikkulainen [1997] note, artificial life is particularly well suited for progressing through increasingly difficult tasks, because complex tasks are naturally separable into subtasks, as exemplified by Brooks [1986]. Navigation has already been proved to be separable for manual design of ANNs [Beer, 1990], so it should be possible also to do this in an evolutionary setting. It would be interesting if incremental evolution would be applicable for this. Intuitively, it should be.

[3]Note that they evolve chromosomes, not singe genes. This is because they use a form of SANE [Moriarty and Miikkulainen, 1996] called ESP [Moriarty and Miikkulainen, 1998] where individual neurons are evolved instead of complete networks.

2.3 Summary

- ANNs are commonly used to control AI-agents. They are easily layered into "Brooks-style" layers.
- Walknet and P. Computatrix are examples of such systems. Their designs are good pointers for how the problem of navigation can be split up into manageable parts.
- CTRNNs, time-dependent, recurrent ANNs, are capable of maintaining internal state and biological-like dynamics, thereby allowing short-term memory and other deliberative behaviours. They are therefore ideal for control of agents with complex behaviours.
- Genetic algorithms have been used to design such networks, but they are still slow and suffer from convergence towards local optima.
- To speed evolution of CTRNNs, at least two techniques have been proposed:

  - Incremental evolution.

  - Seeding initial populations with center-crossing networks.

  - Both techniques provide a means to initialize genetic search in an area of the search space where the desired behaviour is likely to evolve.
- Work is still required:

  - Does center-crossing networks provide any advantage for evolving other networks than CPGs?

  - Can evolutionary search benefit from other properties of CTRNNs, such as enforcing bifurcative neurons?

  - Incremental evolution should in theory be ideal for evolving this in a layered fashion. Does it provide an advantage over "head-on" evolution?

2.3.1 Further reading

Some research has influenced this thesis without finding its way into this background chapter. Particulary, some techniques for evolving adaptive behaviour is barely mentioned (e.g. Dynamical Parameter Encoding (DPE)

and delta-coding). For more information on issues, see Beer and Gallagher [1992], Gomez and Miikkulainen [1997]. Also, researchers Meyer and Kodjabachian have provided insight into evolution of insect controllers by means of a Genetic Programming (GP)-like technique. See Kodjabachian and Meyer [1998] and Kodjabachian et al. [1998] for more about this. Pavel Petrovic here at the Department of Computer Science, NTNU, has recently carried out a study that involves incremental evolution for the generation of finite-state automata Petrovic [2006].

There are other ways to create dynamic neural netwoks Beer [1990] use the leaky-integrator model (see appendix for example code) and Beer [1996] use a CTRNN model with a slightly different equation that the one used in this thesis.

Chapter 3

Design of the Genetic Algorithms

This chapter is divided into three separate sections. The *methods* used are described first. These include a description of the actual simulation setup used, and the mathematical and philosophical background for each evolutionary task.

The method section is followed by a description of the *results* obtained for carrying out the experiments, and possible interpretations of these results. For each evolutionary task, a comparative analysis is done where each evolutionary technique (center-crossing networks, bifurcative neurons and incremental evolution) is compared to traditional, random-seeded, evolutionary searches. For each of the tasks, two major questions are considered: 1) Does the evolutionary search technique provide a usable solution for the task? and 2) Does the technique provide a solution faster and/or better than other (traditional) techniques?

Finally a *discussion* of the theoretical limitations and possibilities of every technique is provided.

3.1 Methods

With the current state of computers, evolving a tactile-olfactory navigation system is a task that is probably too hard for one single GA to solve. It is simply too difficult to design a fitness measure that ensures that significant selection pressure is applied to the population to make it evolve in a desirable direction. Such a fitness measure would be required to encourage recombination of individuals that perform outstanding in a plethora of behaviours, all of these at the same time. Chances are that a random population do not contain many such outstanding individuals - if any at all. As a result, the GA may converge towards some sub-optimal maxima (perhaps equaling *one* of all the desired behaviours) and never find a globally optimal solution to the problem! But what if all individuals in the population were placed as close to a proper solution as possible before evolutionary search was started? What if the initial population consisted solely of individuals that had a high probability of making up a well-performing solution?

In this thesis several techniques for guiding evolution are considered. This section describes the scientific experiments that were carried out in order to collect empirical data on each of a set of chosen techniques. On a macro-level, **incremental evolution** was applied so that increasingly complex behaviours could be gradually evolved. On a micro-level, particular properties of the chosen representation model, CTRNNs, were exploited. Specifically, these micro-level techniques were seeding an initial population with **center-crossing neural networks** to a large extent made up from **bifurcative neurons**.

3.1.1 Incremental evolution

Evolving a CTRNN for controlling limb movement, memory and other navigational properties at the same time results in an enormous search space for a GA. Since computational power is a limited resource, it is necessary to find ways to properly partition the problem at hand into manageable parts. As outlined in the background (chapter 2) Gomez and Miikkulainen [1997] suggest *incremental evolution* as a way of partitioning the evolution of general complex behaviour into smaller, successive, tasks. Incremental evolution begins by finding solutions to at least one of these smaller parts first. When a solution to one of the parts is found, the problem to solve is

made more challenging, and a new evolutionary search is begun using the previous result as a start-off point. For the navigational robot example in chapter 1, learning to drive could be the first "easy" part. Learning how to avoid crashing into walls could be an appropriate subsequent task. The parallels to the subsumption architecture of Brooks [1986] should be fairly evident: Low-level behaviours are created before higher level behaviours; each module is tested extensively in "the real world" before a new level is allowed on top of it; many new layers can be added on top of this one again, etc.

Mathematically, incremental evolution can be considered a *set* of tasks T, where the solution of the task can be defined as the sum of the solutions of all N possible subtasks [Gomez and Miikkulainen, 1997]:

$$T = \sum_{i=1}^{N} t_i \tag{3.1}$$

where each such subtask, t_i can be partitioned to new subtasks, t_{t_i} (just like T it self is partitioned).

So how should the process of evolving a tactile-olfactory navigation system commence? T is of course evolving a complete navigation system, but what subtasks exist? Incremental evolution is applicable only to problems that can be decomposed into a sequence of increasingly complex subtasks, and the problem of identifying this sequence is still left to the researcher. Luckily, as Gomez and Miikkulainen [1997][p.15] notes:

> *(...) at least for Artificial Life and robot control, the task sequences are usually easy to come by, because the goal-task often subsume natural layers of behavior (...)*

Because this thesis looks into how evolution can be properly guided to evolve a specific task (and not so much into how a physical robot creature should be created), results from earlier research have been used to help identifying these "natural layers of behavior". Particularly the layering of ANNs have been inspired by other work in artificial life (Beer [1990], Cruse et al. [1998], Chiel et al. [1999]) , but also the physical layout of the simulated robot has been designed after earlier work in biology and artificial life (Ekeberg et al. [2004]). The choices for each layer of behaviour, each subtask, will be described in the following subsections.

t_1 Walking along a straight line

The first subtask chosen, was to learn how to walk along a straight line. Wandering is the lowest layer described in Brooks [1986], but that was not the only reason that walking was chosen as the first task to evolve here. The work of Beer [1990] also uses walking as a start-off point for further studies of higher level behaviours. Because Beer [1990] use a neuron model similar to CTRNNs (the leaky-integrator model) much of his work can be used in a corresponding CTRNN controller with only minimum of effort of mapping his neuron layout into a CTRNN network. Additionally, Cruse et al. [1998] has built a walking controller around some of the same principles as Beer [1990] - namely six inhibiting leg-modules that are capable of cooperating to create stable tripod gaits. Both the works of Beer [1990] and Cruse et al. [1998] are made up using a modular layered design, which can be easily subsumed by adding new layers on top. There is also another advantage for both of the architectures: They do walking along a **straight** line. Differing from the (aimless) walking controller of Brooks [1986], the quality of straight line walking is very easy to measure. Basically, the fitness of an agent that walks in a straight line is equal to the distance travelled from start to end. Long walks are good walks. It is not clear how to make a similar measure for an aimless agent.

The walking controller of Beer [1990] is based on separate leg modules that cooperate to generate stable walking gaits. It is known from biology that insects use Central Pattern Generators (CPGs) to generate pulses that drive other behaviours (see e.g. Beer [1990] and Bässler and Büschges [1998]), including the control of legs. The simulated leg controllers in Beer [1990] are based on such CPGs. Each leg controller is in itself controlled by an internal pacemaker neuron, that "drives" the stepping rhythm of the leg. Because the evolution of pacemakers is anything but an easy task (see for example Chiel et al. [1999]), the task of walking along a straight line, t_1, is split into two subtasks:

- t_{1_1}: Creating a pacemaker (CPG) for controlling individual legs.
- t_{1_2}: Coordinating pacemaker controlled legs into stable gaits.

So pacemaker behaviour is considered most "primitive", and coordination (which depend directly on pacemakers) is considered the subsequent task.

t_2 Control of walking into a specific direction

Considering the design of the leg controller (an ANN), it should be an easy task to subsume it into force walking into a specific direction. Any neuron can be subsumed by adding a sufficiently strong (positive or negative) synaptic connection to it, so by properly subsuming the right neurons, moving into a specific direction should be possible. Turning will then be the most basic *navigational* task provided by the robot.

Looking at the earlier works in creating walking controllers (Beer [1990] and Cruse et al. [1998]), there are two possibilities for controlling turning that have already been explored: Beer [1990] uses a force that provides lateral propulsion on the fore legs; Cruse et al. [1998] uses a more complex simulation model with a leg model that allows individual legs to pull the insect into the proper direction. A third, unexplored scheme, is to enforce differences in the stepping frequency between the two sides of the body. This scheme is suggested by Beer [1990][pp.109-110], but he does not implement it due to the inaccurate physical model of the P. Computatrix.

Because the simulation environment in this thesis is physically accurate [Klein, 2006], it should be possible to use a scheme pretty close to what would work in the real world. Therefore, walking in a specific direction is following the suggestion by Beer [1990] about enforcing different stepping rhythms between the two sides of the body. As a side note, the approach of Cruse et al. [1998] would require an even more complex neuron model (and physical model too), because it involves more joints (four joints on each leg, versus one joint on each leg for Beer [1990]). The turning model of Cruse et al. [1998] is therefore disregarded here.

The turning task is not further split into easier subtasks.

t_3 Avoiding obstacles

Just like for the robot in Brooks [1986], it should be possible to both force walking into a proper direction, and avoid obstacles at the same time, using different layers of behaviour that subsume each other. In Brooks [1986], avoiding contact with other objects is the bottom behaviour of the robot control system. For reasons described in the previous subsections, it is

not desirable for incremental evolution to start off with obstacle avoidance. It would make fitness measuring (and therefore applying proper selection pressure) difficult. To make up a sequence of tasks that allow for a easy measuring of fitness, obstacle avoidance is here evolved as the third subtask. Evolution should therefore be able to start off with a creature that walks pretty well toward some goal, but needs to adapt its walking to avoid obstacles, using tactile sensing.

It should be possible to evolve obstacle avoidance as a single task. It is however also possible to make the task easier/more difficult, for example by evolving avoidance of stationary obstacles before avoiding moving obstacles (as successfully done by Gomez and Miikkulainen [1997]).

3.1.2 Genotype representation

To make incremental evolution work, the resulting genome from one evolutionary search must be easy transferable into subsequent searches. This should make it possible to use the result of one evolutionary task as a start-off point for later more complex tasks. As outlined in section 2.2.2, there exist techniques such as delta-coding that are very suitable for e.g. Symbiotic Adaptive Neuro-Evolution (SANE) with Enforced Segregated Populations (ESP) [Moriarty and Miikkulainen, 1996] and [Moriarty and Miikkulainen, 1998]. In this thesis, a even simpler approach is taken to transferring genome: All genes are represented by 24-bit integers (equaling any value from 0 to 2^{24}, about 16.8 million values). The complete genome is therefore a vector of such 24-bit integers. This has the following advantages:

- When starting off a new evolution, results from earlier evolutions can be injected directly into the population. This is done by setting the first part of the new genome (vector) to the result from earlier evolution. To ensure population width, the values can be randomized (e.g. by adding/subtracting some small value from every gene) before insertion.
- Since all genes are equal, they can be mixed freely by recombination. There are no special considerations to take for different gene-types or similar. Simple one- or two-point crossover, or other well known recombination techniques, can be used.

- The same advantage applies to mutation. One, standard, mutation operator that works the same way for all genes.

It should also be noted that this simple genotype representation does not exclude any solutions. Other techniques, such as DPE (see a very relevant example in Beer and Gallagher [1992]) may remove half of the solutions after a predefined number of evolutions (e.g. start to focus when an optimum is approached). This can be done to increase precision for the remaining generations, but at the cost of discarding all the other solutions that are not focused on. Because of this last point, DPE has been disregarded here.

Recombination

Because of the advantages mentioned above, the recombination operator used here was two-point crossover for all the GAs. It is loosely based on the recombinations done with DNA in nature. It works like this:

Lets say two genes, A and B, are to be recombined into two offspring. We have:

$A : [0, 12, 3, 34, 643, 2, 232, 532]$ and $B : [13, 4, 23, 143, 23, 43, 243, 1]$.

Two random crossover points are chosen within the length of these genome. Say, 3 and 6. The offspring, C and D, are then produced by creating genome C from genes 0-2 of A, genes 3-5 of B, and finally genes 6-7 of A. The opposite procedure is used to form genome D. We therefore end up with:

$C : [0, 12, 3, 143, 23, 43, 232, 532]$ and $D : [13, 4, 23, 34, 643, 2, 243, 1]$.

Mutation

Mutation was applied by tampering with one gene. To exemplify, lets say we want to mutate gene D, above, before we insert D into the population. To do this, we pick a random number within the length of gene D, say 4. We then pick a random number within some range (e.g. -10000 to 10000), say 304, to subtract from gene 4. Unless the result would be outside the range

of a 24-bit integer, the calculation is performed, and the resulting genome is returned:

$D : [13, 4, 23, 34, (643 - 304), 2, 243, 1] \rightarrow [13, 4, 23, 34, 339, 2, 243, 1]$.

Genotype to phenotype conversion

In order to convert 24-bit genes into phenotype values, a standard set of equations were used. Genes could be mapped into three different values, either: 1) (Synaptic) Weights(w_{ij}), 2) Biases (θ) or 3) Time-constants (ϕ).

In any CTRNN, one continuously maximum-firing synaptic connection have the same effect on a neuron as the bias have (this can bee seen from equation 2.3, page 28). Therefore, one common function was used both for mapping weight and bias genome into real values:

$$\theta = -20 + \frac{gene}{\frac{2^{24}}{40}} \tag{3.2}$$

This, of course, resulted in phenotype values (weight/bias values) between -20 and 20. Because CTRNNs use sigmoid activation (that saturates towards 0 and 1 for -20 and 20), one maximally weak/strong connection is capable of completely inhibiting/exhibiting a neuron.

For the time-constants, a similar function was used:

$$\theta = -16 + \frac{gene}{\frac{2^{24}}{16}} + t \tag{3.3}$$

where t was the CTRNN integrator time step (0.01 sec).

Basically the same equation as for the weights, but converting the gene to a value in the range 0.010 to 16.01. Reasons for keeping the value above t is due to the correctness of the simulation results (see also section 2.1.2). A maximum of 16 has also been used in similar research earlier (CPG evolution in Mathayomchan and Beer [2002] use [0.5 10] for their two-neuron circuit) and should provide a sufficient range here.

3.1.3 Evolving a rhythmic CTRNN (t_{1_1})

Many animals use some form of pacemakers to drive other behaviours. Pacemakers create rhythmic bursts that can signal things like for example the contraction of an animals heart muscle. Insects use pacemaker circuits to create complex coordinated rhythmic behaviour, such as the leg stepping rhythm. In Beer [1990]s research, special pacemaker cells are used to coordinate a hexapod robots legs into stable gaits.

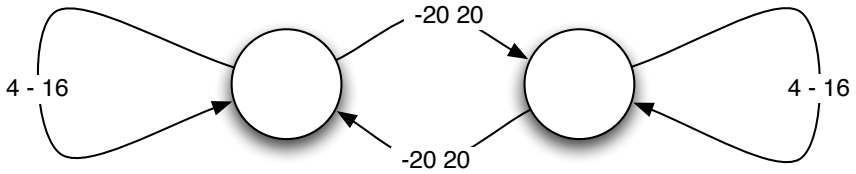
Pacemakers that control other neural networks are often called Central Pattern Generators (CPGs) in relevant literature. Because of the wide applicability of CPGs, much work have been put into using CTRNNs and GAs to create them. CTRNNs are in theory able to simulate any dynamical system (if sufficiently large, (Funahashi and Nakamura 1993) in [Chiel et al., 1999]), and CPGs should be no exception. A GA should therefore be able to evolve any CPG given an infinitely large population or infinite time and infinite network size.

Reality is however somewhat different from theoretical GAs with infinitely large populations and no time constraints. The probability that a random network population of moderate size (say 100 individuals) should contain one or more CTRNNs exhibiting pulse behaviour is of course rather small (actually very small, see Mathayomchan and Beer [2002] for a statistical analysis of this question), since each individual in the population could represent any of a very wide range of dynamical systems. Even a large population cannot be *guaranteed* to contain *any* genome close to one corresponding to a pulse network. As a result, evolution may have to run for a long time before any improvement towards desired behaviour is observed. If it happens at all.

Fortunately, several researchers have suggested ways to guide evolution into faster discovery of near-optimal CPGs. An approach similar to Mathayomchan and Beer [2002] have been used in this thesis. It takes advantage of so-called center-crossing CTRNNs to start evolutionary search in fruitful regions of the genetic search space. Also, some mathematical background from Beer [1995] and Chiel et al. [1999] have been used to further guide evolutionary search: Namely the theory of bifurcative neurons, ensuring that all neurons in the initial networks are capable of generating pulses - even before any evolutionary search is started.

The network topology chosen here is a network that fulfills both the center-

Figure 3.1: Topology of the pacemaker circuit to evolve. One neuron can be considered an input/output(IO) neuron. The firing frequency of this neuron can be read as the pacemakers state. The other neuron is "hidden", and its function is to constantly force the IO neuron to change its firing frequency. The complete network can then produce rhythmic pulses, if the neuron and weight properties of both neurons are set properly.



crossing, and the bifurcation properties. The basic layout can be seen in figure 3.1. The following sections provide an elaboration on this design.

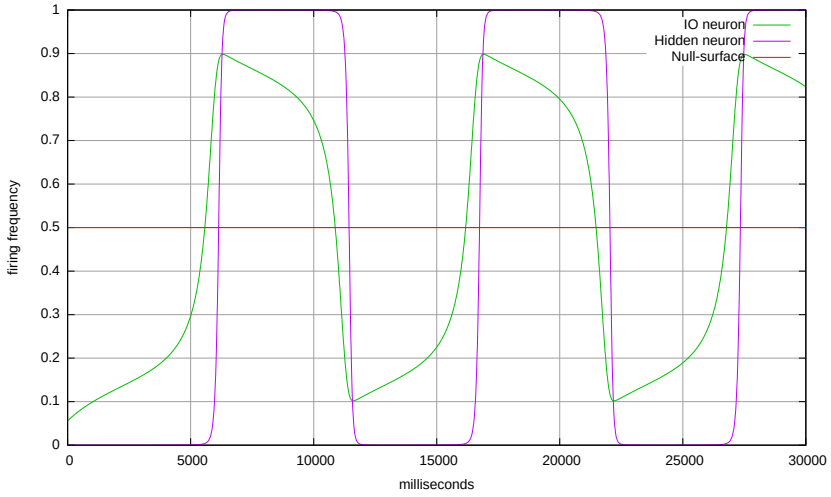
Population initialization

Center-crossing networks

Seeding an initial population with center-crossing CTRNNs have been proved to greatly improve both the frequency of pulse-circuits occurring in a population, and the speed that high fitness pulse-circuits are evolved [Mathayomchan and Beer, 2002]. A center-crossing CTRNN is a CTRNN where the null-surfaces of all neurons intersect at their exact centers of symmetry [Mathayomchan and Beer, 2002]. The null-surface of a neuron, is where the neuron bias and all synaptic inputs sum to 0. And - because CTRNNs use sigmoid activation - a neuron has a firing frequency of 0.5 at its null-surface ($\sigma(0) = 0.5$). Center-crossing networks therefore consist of neurons that on average have firing frequencies around this value. Example firing frequencies of the neurons in a "pseudo random" center-crossing network can be seen in figure 3.2.

As can be seen from the sigmoid activation function in figure 2.2, only small changes in synaptic inputs around the null-surface can lead to a very different neuron firing frequency. Changes within the range $[-5, 5]$ actually correspond to firing frequencies covering almost all of the neurons firing range. Outside this range, the change of net input has only sparse effect

Figure 3.2: Firing frequencies of the neurons in a pseudo-random neural network that satisfies the center-crossing requirement (topology in figure 3.1). Note that both neurons share the condition that they have firing frequencies centered around their null-surfaces (0.5).



on the firing frequency. Because center-crossing networks consist only of neurons whose activation functions are exactly centered around the range of net input the neuron receives, they can be viewed as *networks of maximally sensitive neurons*. It is to be expected that such networks exhibit a wider range of dynamical behaviours than random networks would. They should also be easier to interact with for other networks, because input/output neurons are also maximally sensitive. For these reasons, center-crossing networks *should* be more frequently fit to the fitness measure, than random networks.

Bifurcative neurons

In Beer [1990], the locomotion controller of the P. Computatrix insect is driven by special pacemaker neurons. These pacemaker neurons exhibited behaviour similar to real pacemakers that were described first by Kandell in 1976 (Kandell 1976 in Beer [1990]):

1. When a pacemaker cell is sufficiently hyper-polarized[1], it is silent.

[1] A hyper-polarization is a *decrease* in membrane potential, leading to a lower firing frequency.

2. When it is sufficiently depolarized[2] it fires continuously.

3. Between these two extremes it rhythmically produces a series of relatively fixed-duration bursts and the length of the interval between bursts is a continuous function of the injected current.

4. A transient depolarization which causes the cell to fire between bursts can reset the bursting rhythm.

5. A transient hyper-polarization which prematurely terminates a burst can also reset the bursting rhythm.

Since pacemaker neurons are central components of the locomotion controller of the P. Computatrix, it is desirable that the pacemaker evolved here have similar properties. This should ensure that the pacemaker evolved will be suitable for inclusion in the evolution of the rest of the locomotion controller. To increase the probability that such pacemakers were evolved, some theory from Beer [1995] was used:

According to Beer [1995], a CTRNN-neuron can have two different steady-state membrane potentials (two solutions to equation 3.5, page 48) if it has a self-connection with a weight > 4. Such neurons are called *bifurcative* neurons. Which of the two membrane potentials a bifurcative neuron will converge towards at some point in time, depend on the neuron potential it had before that point in time. The input from another neuron can also help pull the membrane potential of a bifurcative neuron into another steady-state membrane potential [Beer, 1995]. Other neurons can therefore provoke a state switch in another neuron by means of synaptic connections.

For a two neuron circuit like the one in figure 3.1, the pulse state can be read as the firing frequency of one of the neurons (e.g. one neuron serves as an output neuron). If this neuron is bifurcative, it should in theory be possible to affect its neuron state using a sufficiently positive/negative input into one or both of the neurons. This means that observations 1 and 2 of Kandell (1976) should be possible to fulfill.

It should also be possible for such a network to produce fixed duration bursts. This will happen to networks where the neurons continuously push or pull

[2] A depolarization is equal to an *increase* of membrane potential and indirectly an increase in firing frequency.

Figure 3.3: The equation controlling the state change of every CTRNN neuron. τ is the time constant of i, y_i is the current membrane potential, $w_j i$ is the weight of the synapse from neuron j, y_j is the current membrane potential of neuron j, θ_j is the bias of neuron j and finally I is a (constant) external input, and σ is the standard logistic activation function (sigmoid).

$$\dot{y_i} = \frac{1}{\tau_i}(-y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_j + \theta_j) + I_i) \tag{3.4}$$

each other into switching between their two possible states. If the initial population is seeded with center-crossing networks, both neurons should be maximally sensitive to change, and therefore also maximally easy to push or pull. Hopefully (probably), this combination of bifurcative and center-crossing seeding will lead to higher probabilities of pulse networks occurring in the initial population. If it does, then at least some of point 3 of Kandell (the length of the interval bursts depend much on the time-constants, and cannot be controlled this way) should be possible to fulfill.

A transient depolarization or hyper-polarization of either neurons in the network of bifurcative neurons should also be able to reset the bursting rhythm. If either of the neurons are hyper/depolarized for a short period of time, both neurons are ultimately affected, and the rhythmic behaviour should cease until both neurons again fall into one of their steady states, restarting the cycle. Points 4 and 5 of Kandell should therefore also be possible to achieve.

Putting it all together
As proved in Beer [1995] the center crossing condition occurs when the neuron biases of all neurons are set to half of the negative of the sum of input weights (see equation 3.5). This means that the bias exactly out-weight the sum of all synaptic inputs when the connected neurons have a firing frequency of 0.5. In other words we should have a network with neurons that on average are firing about 0.5. In yet other words, the network should consist of neurons that are in **maximally sensitive** states most of the time. It can be seen from equation 3.5 that either biases or weights must be adjusted to ensure center-crossing networks.

Figure 3.4: Equation for the bias of any neuron in a center-crossing network.

$$\theta = \frac{-\sum_{j=1}^{N} w_{ij}}{2} \tag{3.5}$$

Since at least self-weights are forced to be > 4 (because of the bifurcation requirement), it should be easier to adjust biases rather than weights. This strategy allows for setting all other networks properties (weights and time-constants) to random values. If the bias is afterward set according to equation 3.5, the null-surfaces of the neurons membrane potentials should be moved into intersection.

One bifurcative neuron should in theory be able to exhibit pulse behaviour solely through self stimulation (using one self connection). One neuron is not sufficient for the center-crossing condition to occur, however. For several null-surfaces to intersect as described above, at least two null-surfaces are required! Therefore, at least two neurons must be interconnected to form a center-crossing network. These are the reasons for suggesting a minimally complex network as shown in figure 3.1 (page 44).

To summarize, the resulting strategy for seeding the initial population was:

1. Create a set of random genome. Each genome is represented by a vector of length 8 (2 time-constants, 2 biases, 4 weights) with 24-bit unsigned bytes for each gene (equaling values in the range 0 to 2^{24} for each byte, see section 3.1.3 for details).
2. Setting the self weights to genes corresponding to a weight in the range $[4, 16]$ (ensuring bifurcation).
3. Setting all other genes corresponding to values in the range $[-20, 20]$.
4. Adjusting the bias gene so that a center-crossing network will result, using equation 3.5.
5. Additionally setting the time-constants to random bytes corresponding to a time-constant in the range $[0.44, 0.58]$. This is done to ensure that the neurons exhibit pulse behaviour close to the required pulse of about 1 second (the stepping frequency of a real Carausius Morosus [Dürr et al., 2001]).

The genome structure is visualized in figure 3.5

Genotype to phenotype conversion

The phenotype to be evolved consisted of a two-neuron CTRNN where each neuron had one self-connection, and one connection to the other neuron (see figure 3.1). In genotype → phenotype conversion, each byte in the genotype vector was converted into exactly one phenotype variable[3] . This of course meant that changing one of the genes in the vector could be seen as a direct change in one aspect of the corresponding phenotype.

Because biases below/above -20/20 are irrelevant (the sigmoid activation function saturates towards 0/1 for net inputs of these values), the 24-bit gene was scaled into a value between -20 and 20. Similarly, the time-constants were scaled into values in the range $[t, 16]$ where t corresponds to the integration time-step used by the Euler-method CTRNN integrator (see section 3.1.2 for more about the range chosen).

As for all other evolutions, the genotype *only* encoded the CTRNN. The physical model was hand-coded.

Figure 3.5: The genome layout. Each gene is a 24-bit unsigned byte, allowing for 2^{24} (approximately 16.8 million) different values. For the pacemaker, each gene corresponds to exactly one feature in the phenotype.

$\theta_{IO-neuron}$	$\tau_{IO-neuron}$	$\theta_{H-neuron}$	$\tau_{H-neuron}$
$W_{IO \to IO}$	$W_{H \to IO}$	$W_{H \to H}$	$W_{IO \to H}$

Fitness evaluation

The evolution of the rhythmic network differed a bit from the evolutions shown later with regard to fitness evaluation. A highly specialized creature was used for quantifying performance, and fitness was considered the length that this creature could walk using the evolved CTRNN. A snapshot of the pacemaker creature is shown in figure 3.6. It had four wheels (a bit like

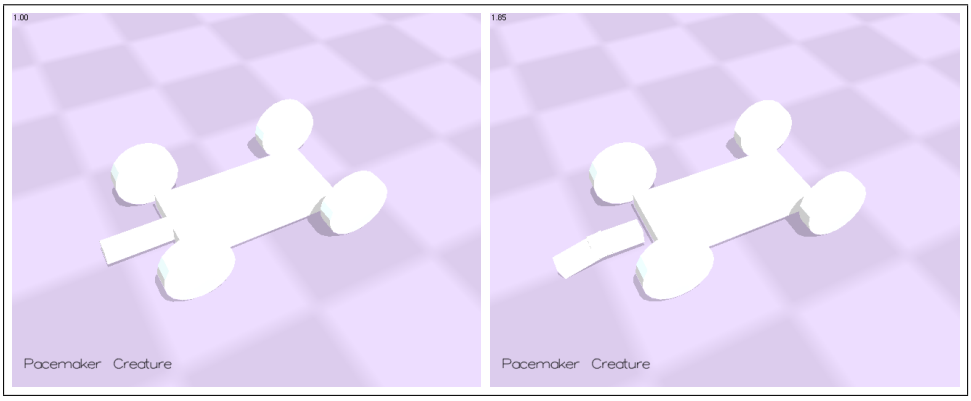
[3]This does not have to be the case. As will be shown i later evolutions, one gene may correspond to several features of the phenotype.

a car), but it also had one leg. The leg pointed straight out behind the creature, and could be pushed back and forth by applying force to it. To ensure that the leg touched the ground only when moving backwards, it also had one joint. The joint was bended with the same "muscle" that pushed the foot back. Applying force therefore led to a lowering of the leg, and then an extension (to provide propulsion). The joint was also straightened out when the leg was pulled forward, lifting the leg up when negative force was applied. This allowed for just one (positive/negative) force controlling foot extension/retraction: Activating the muscle lead to a lower-leg-and-push action, deactivating it lead to a lift-leg-and-retract action.

To ensure equal simulation settings for each fitness evaluation, a new creature was moved into a pre-defined starting position with the leg retracted. A CTRNN that corresponded to the genome to evaluate was then created, and connected to the creatures leg. The creature was then allowed to move for 8 seconds (800 CTRNN integrations). The maximum/minimum force that was allowed, was set so that foot extension would take a little less than one second at maximum force (that is: the time of maximum firing of the IO-neuron. An adequate value was found by manual experimenting with different forces applied). Foot retraction was a lot faster that foot extension. Far less than a second with the same force, because there was no friction. In sum a maximally efficient extension-retraction cycle took about one second. A maximally efficient creature would therefore have a pacemaker circuit that fired rhythmically with burst of about one second length. A frequency of about one second is desirable, because that is the stepping frequency of a real Carausius Morosus.

Because of the wide range of dynamics of the different individuals, a wide range of solutions to fast walking should exist in this setting. Some individuals may not move the creature at all. Others may move it by extending the leg once during the whole life-time of the creature. Others again may move it by taking several steps. The reason for evaluating the phenotype for as much as eight seconds, was to give the last mentioned type of individuals a competitive advantage, even if they do not perform exceptionally at first. For extremely short fitness evaluations (e.g. just one second), an individual that takes just one step would perform more or less equally as an individual capable of taking several exceptionally good one-second steps. For fitness evaluations of two seconds the multi-step creatures would start to outperform all other creatures. Using eight seconds, evolution would mostly be concerned about different multi-step creatures competing. Even creatures

Figure 3.6: The pacemaker creature with the leg at its two extreme positions (retracted/extended). The number at the top left corner is the simulation time in seconds.



with many weak steps would outperform all non-multi stepping creatures in such an environment.

The genetic algorithm

The genetic algorithm parameters are summarized in table 3.1.

3.1.4 Evolving a locomotion controller (t_{1_2})

Leg coordination for hexapod walking is not as easy as it may look. In addition to providing propulsion, the legs must be able to coordinate in stable gaits, so that the insect does not fall. Also, it should be possible not to do only forward walks, but also to move sideways or backwards. The system must also be robust to perturbations so that it can adopt to unpredictable real-world environments. The last point being the very philosophical basis for AI in this thesis.

Luckily, hexapod walking has been an active research area for some time, including connectionist models since the 1980-90s. Most notable of these are perhaps the pioneering works of Beer [1990] and the impressive work

Table 3.1: Parameters of the GA for evolving a rhythmic CTRNN

Function	**Description**
Fitness evaluation	Longest walk using a simulated, one-legged, creature.
Genotype representation	Vector of 8 24-bit unsigned bytes (one byte per gene).
Initialization	Random center-crossing CTRNNs with bifurcative neurons.
Mutation	Random add/subtract x (where $0 < x < 10000$) from gene.
Mutation probability	5% of the offspring.
Number of offspring	2
Parent selection	2 from best quartile of population (= 2/10 = 20% of best).
Population size	40 individuals
Recombination	Two-point crossover.
Recombination probability	100% of selected parents.
Survival selection	Replace worst 2 in population.
Termination condition	150 generations elapsed.

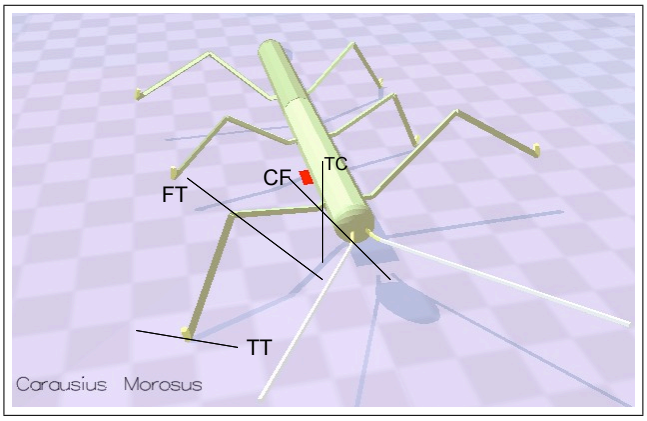
of Cruse et al. [1998]. Both which are described in more detail in chapter 2. The model used here is based on Beers work for a simulated Periplaneta Americana (American cockroach), which he have named *Periplaneta Computatrix* [Beer, 1990]. The insect here is simply called Quickbeam, because it through evolution will end up being the fastest of the slow, tree-like agents.

The physical model

The physical model resembles a Carausius Morosus, sometimes referred to as the laboratory stick insect, Indian stick insect, or just the "walking stick". There were several reasons for choosing the C. Morosus as a biological basis:

- The antennas have only two joints:
  This point is essential, because the fewer joints, the simpler the antenna is to control, and the smaller the search space for the GA is.

Figure 3.7: "Quickbeam", the physical model that the CTRNN is to control. It resembles a real Carausius Morosus. The axes that leg joint revolve around, are marked by black lines. TC is the thorax-coxa joint, CF is the femur-tibia joint, TT is the tibia-tarsus joint. Only the TC- and CF-joints were unlocked here.



- The joints are both hinge joints (as opposed to ball joints):
  They still seem to be complex enough for real Carausius Morosuses to live just fine. This basically boils down to the "as simple as possible" point in the introduction. Just like few joints are desirable, limiting the movement of each joint makes the physical layout even simpler. For a detailed description (measuring) of real Carausius Morosus antennal system, see Krause and Dürr [2004] and Dürr et al. [2001].

- Antennal (and leg) movement data are available:
  Actually, the C. Morosus is (by 2004) the *only* insect of which unrestrained three-dimensional antennal movement data is available, [Krause and Dürr, 2004]. This is potentially useful for evaluating simulation quality.

- C. Morosus actively move the antennas:
  This allow for studies of antennal control systems, and perhaps insight into the connection between stepping patterns and antennal movement.

The P. Computatrix of Beer [1990] wanders around in a two-dimensional environment with simplified Newtonian physics. In this thesis, a three-

dimensional simulation environment called Breve is used. The Breve environment have (approximately) realistic physical features [Klein, 2006], so there are a few things to consider before trying to use Beers' controller directly:

- Legs here have weight. In Beers environment they don't.
- Legs here have exact angles. Legs are either up or down in Beers environment.
- Inertia influence walking a lot. Beers environment is inertia-free.
- The walking gait of the C. Morosus is a lot slower than the gait(s) for the P. Computatrix.
- The body layout of the C. Morosus cannot be assumed to equal the body of the P. Computatrix (e.g. legs differ in length).

The task of the GA here was therefore to find proper weights, biases and time constants so that the P. Computatrix controller could make Quickbeam walk properly. To make this task as simple as possible, all but the absolutely necessary joints of the C. Morosus were locked (see figure 3.7). Also, the movement of the legs were limited so that stable gaits could be "easily" achieved. Note that the last constraint here may seem superfluous, or perhaps too limiting since a fitness function may easily penalize unstable gaits anyway. The strategy does however allow semi-stable gaits to exist in the population longer (e.g. not only "near-perfect" solutions will survive in the beginning), so that population diversity is easily maintained. Additionally, since the walking task is made "easy", a faster convergence towards an optimal solution is to be expected.

The exact layout of the physical model is drawn in figure 3.7 (page 53) . The weights and length of all limbs are set according to measures from [Ekeberg et al., 2004].

Population initialization

As already described in the method for evolving a highly fit CPG, seeding the initial population with center-crossing CTRNNs should theoretically

improve both the quality and the speed of evolving a CTRNN. A central question in this thesis is whether this theory can be applied also to other evolutionary tasks.

During pacemaker evolution, all neurons were initialized so that they would on average be maximally sensitive. In such networks, all neurons can easily affect other neurons. The network as a whole will therefore exhibit a very wide range of dynamics compared to random networks [Mathayomchan and Beer, 2002].

As can be seen from the network topology in figure 2.5 (page 26) it is not possible to create a true center-crossing network for the locomotion controller. The reason for this is that not all neurons have synaptic inputs (sensor neurons do not, for example) so some neurons will be unaffected by the rest of the network. Looking at the major point of center-crossing networks, however, the reasons for their success may not be *only* that all neurons are set up in a mutually advantageous interconnection. Single neurons may benefit from being **maximally sensitive** anyway. A network of such neurons will still yield a network of neurons that are more likely to give useful (strong) responses to change in input, and therefore exhibit a wider range of behaviours than random networks.

It would therefore be interesting to try and apply the bias adjustment (equation 3.5, page 48) at least to the neurons that *do* have incoming connections. The point of maximum sensitivity should be at the negative of medio of the sum of all inputs, just like for center-crossing neurons. In theory, this bias should then out weight all inputs (on average) always keeping the neurons' activity close to its null-surface.

To test this theory the bias of some neurons in the leg-controller network were adjusted according to equation 3.5. Just like for the pacemaker genome, the bias adjustment was done after all other genome values were set to random or pseudo random values. The pseudo random values were calculated in two ways:

1. An approximation to values of synaptic weights that were known from the leg controller of the P. Computatrix [Beer, 1990][pp. 178-180] was made. In other words: The genes were randomized to some value close to the values of Beer [1990] (see calculations below).

2. For the pacemaker part of the genome, the resulting genome from the best pacemaker evolution (task t_{1_1}) was injected. These values were also randomized a "neighborhood" of the earlier result.

The only neurons with synaptic inputs were motor neurons (M). The calculations (using equation 3.5) for the biases were (based on values from Beer [1990]) were as follows:

$$M_{foot_\theta} = \frac{-(-4+10)}{2} = -6/2 = 3 \tag{3.6}$$

$$M_{stance_\theta} = \frac{-(-10+10)}{2} = 0/2 = 0 \tag{3.7}$$

$$M_{swing_\theta} = \frac{-(10+-15)}{2} = 5/2 = 2.5 \tag{3.8}$$

Because the genotype representation consists of values between 0 and 2^{24}, and since these values are scaled into the range $[-20\ 20]$ in genotype $\rightarrow$ phenotype conversion, the resulting values to initialize the genes for these neurons must be a reverse mapping of the genotype $\rightarrow$ phenotype conversion. The calculations are therefore the exact opposite of equation 3.2 (page 42):

$$\begin{aligned} M_{foot_\theta} &= (Gene_{M_{foot_\theta}}) - 2^{24}\frac{40}{2^{24}}) + 20 \\ &\Longrightarrow \\ Gene_{M_{foot\theta}} &= \frac{M_{foot_\theta} - 20}{\frac{40}{2^{24}}} + 2^{24} \\ &= \frac{3-20}{\frac{40}{2^{24}}} + 2^{24} = \frac{48234496}{5} \\ &= 9646899 + \frac{1}{5} \rightarrow \mathbf{9646899} + e \end{aligned}$$

$$\begin{aligned} M_{stance_\theta} &= (Gene_{M_{stance_\theta}}) - 2^{24}\frac{40}{2^{24}}) + 20 \\ &\Longrightarrow \\ Gene_{M_{stance\theta}} &= \frac{0-20}{\frac{40}{2^{24}}} + 2^{24} \\ &= \mathbf{8388608} \end{aligned}$$

$$\begin{aligned} M_{swing_\theta} &= (Gene_{M_{swing_\theta}}) - 2^{24}\frac{40}{2^{24}}) + 20 \\ &\Longrightarrow \\ Gene_{M_{swing\theta}} &= \frac{5/2 - 20}{\frac{40}{2^{24}}} + 2^{24} \\ &= \mathbf{9437184} \end{aligned}$$

Where e is an additional factor to add, so that the insect actually applies enough force to stand when no other neurons are firing (found through trial and error with the breve simulation).

As stated above there are no synaptic inputs for sensor neurons. There is however one external current for each sensor: The sensory reading. By flipping this value between some positive value (sensor on) or the exact opposite negative value (sensor off) [4] the calculation of a proper bias can still be estimated. The external input can be considered equal to a incoming connection, and because this connection is equally strong and weak, the sum of inputs must be 0. Referring again to equation 3.5 the bias corresponding to the null surface of these neurons must always be:

$$S_\theta = \frac{-0}{2} = 0 \tag{3.9}$$

Or, no bias. The value for the gene should therefore be somewhere (randomized) around:

$$Gene_{S_\theta} == \frac{0 - 20}{\frac{40}{2^{24}}} + 2^{24} = \mathbf{8388608} \tag{3.10}$$

Genotype to phenotype conversion

The genotype to phenotype conversion for locomotion controller evolution was very much like the one for pacemaker evolution. The first genes of the genome were actually identical to the pacemaker genome, so the result from t_{1_1} could be directly injected into the locomotion controller genotype. The genome was however a bit longer (because there were many more neurons

[4] The external current of a sensory neuron is positive when sensory input is received. This causes the firing frequency to go towards 1. When no sensory input is received, the external current is set to a negative value of the same magnitude as the positive one, rather than setting it to 0. This ensures that the firing frequency saturates towards 0, rather than 0.5.

and synapses), and that it converted some of the genes into several features in the phenotype. This last point was due to the fact that it was known from Beer [1990] that all legs were created equal, and all sensor neurons shared the exact same properties. The genome structure is shown in figure 3.8.

Figure 3.8: The genome layout for locomotion controller evolution.

<pacemaker genes>	<locomotion controller genes>

(a) The basic layout of the genome, after addition of the locomotion controller genes.

θ_{stance}	τ_{stance}	θ_{swing}	τ_{swing}	θ_{foot}	τ_{foot}	$\theta_{sensors}$	$\tau_{sensors}$

(b) The layout of the locomotion genes part of the genome. Each gene is a 24-bit unsigned byte. Note that some of the genes are shared; they each result in realization of several features in the phenotype. All legs are equal, and the sensor neurons inside each leg equal too. This narrows the search space substantially.

Fitness evaluation

Good performing creatures should walk using stable gaits. To reflect this requirement in the fitness evaluation, the following simulation setup was used:

At each fitness evaluation, a new Quickbeam body was initialized, and a new CTRNN (corresponding to the genome to test) was connected to the body. Right legs were placed in a forward direction, while the opposite was true for the left legs (so that every fitness evaluation started off equally). The creature was then allowed to try to walk on a flat surface for 2500 CTRNN integrations (25 seconds), and the distance from start to end of this walk was measured.

To add some extra selection pressure, an artificial force that could potentially push the creature backwards was applied whenever anything but the legs were touching the ground. Falling was therefore penalized quite hard, but still not hard enough to remove the creature if it was able to get up again and continue walking. The idea of this fitness scheme, was to quickly remove the creatures who just fell and never rose again, and still maintain a wide population of other solutions. Good walks with occasional falling could

Table 3.2: Parameters of the GA for evolving leg coordination.

Function	**Description**
Fitness evaluation	Longest walk for 500 neural network integrations.
Genotype representation	Vector of 24-bit unsigned bytes (one byte per gene).
Initialization	Pseudo-random genes. First genes were placed close to result from by pacemaker evolution. Other were adjusted according to the P. Computatrix controller. Others again were placed close to center-crossing networks.
Mutation	Random add/subtract x (where $0 < x < 100000$) from gene.
Mutation probability	5% of the offspring.
Number of offspring	2.
Parent selection	2 from best quartile of population (= 2/25 = 8% of best).
Population size	100 individuals
Recombination	Two-point crossover.
Survival selection	Replace worst 2 of population.
Termination condition	150 generations.

possible improve to become very good walks if properly recombined. There is also a possibility that non-falling solutions could improve using some genes from the falling-solutions.

The genetic algorithm

A summary of the different GA parameters can be found in table 3.2.

3.1.5 Evolving turning (t_2)

Turning is an essential behaviour for an agent to aim for a specific target in it's environment. A turning system has at least one sensor, and this

sensor must affect locomotion so that the insect moves properly as a function of the sensory input. Since other behaviours, such as obstacle avoidance ultimately depend on the quest for a specific target, it would be natural to evolve turning on top of straight locomotion, before other behaviours are considered. It is therefore the first real navigational task evolved here, and is numbered t_2 in the sequence of incremental steps.

In this thesis an olfactory sensor model will be used to aim at targets (see Kodjabachian and Meyer [1998] for results from evolving a similar (though based on GP) insect controllers using olfactory sensing). The real Carausius Morosus uses the flagellum of its antennas to sense pheromones, and a solution inspired on this will be used here. There are several ways to implement an apparatus for turning it self. As already mentioned in section 3.1.1 it is possible to achieve turning by interfering with the stepping frequency on each side of the insect (e.g. slowing down and/or increasing stepping frequency for one side of the body). It is not apparent how to do this, however, so a rough approximation is suggested here, and the rest is left to evolution.

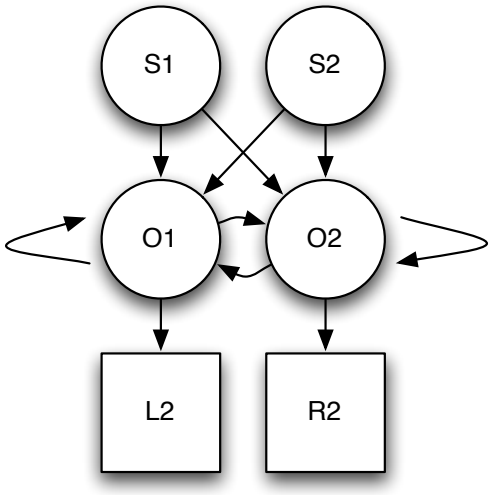
Population initialization and network structure

The task at hand was to make odor sensing affect the direction Quickbeam walked. From this requirement, the following network structure was suggested (with no other biological basis than that differing the stepping frequency is desirable):

Because there are two olfactory sensors, there must be two sensor neurons. If these sensors can affect the middle legs, then turning should be possible. This is true, because if the middle leg is affected (by a synaptic connection), then the pacemaker of that leg is indirectly affected, because front and back legs sensors will be tilted out of sync with the walking gait (Beer [1990][pp.109-110]). It may also be that the insect will learn to step backwards with one leg (if the synaptic connection is strongly negative) while the legs on the other side of the body is pushing the insect forward in the normal manner. That would definitively result in turning on the spot. Evolution may of course also end up with a totally different design, but that OK too of course, as long as i works.

So how should the connection between the sensors and the legs be? One possible solution is to consider the required direction to turn as a function of

Figure 3.9: A suggestion for a neural network controlling turning by subsuming the locomotion controller. S1 and S2 are the two sensory neurons. O1 and O2 are two "hidden" neurons keeping a temporal memory. L2 and R2 are the (stance neurons of) the left and middle legs locomotion controllers, respectively.



the difference between the sensor values of the two antennas. Two examples can be considered to see that this will hold: 1) If the sensors fire equally strong (the same amount of odor is present on both sensors), then the target must be either right in front of the insect, or right behind it. 2) If one sensor is firing stronger than the other, then the odor source must be on the side that is firing strongest. To allow evolution to make use of this, an additional layer of neurons were introduced, so that the odor sensing could change the state of two (possible bifurcative) neurons that again affect walking. If these neurons were bifurcative they could potentially adapt four stable states (left on and right off or right on and left off, etc.). The creature would then be able to keep a temporary memory (one memory per network state) about where it is/was walking, and it should therefore be robust to perturbations. The suggested network can be seen in figure 3.9.

The population was seeded with networks of this type, and the following gene adjustments were done to improve the evolutionary search:

1. Weights from output neurons ($O1$ and $O2$ in figure 3.9) to middle legs were initialized negative. This was done to encourage solutions where the middle leg is slowed down or even moved backwards.

2. Additional weights were added to ensure that the output neurons were bifurcative. They should therefore be able to keep state by flipping between several network states (four states: neuron $O1$ "on" and $O2$ "off", $O1$ "off" and $O2$ "on" etc.). These weights were initialized to some value > 4, ensuring that they were initially bifurcative.

3. All earlier evolved genome were set to values close to what had been evolved in t_1.

4. The network was initialized as center-crossing. Note that this included readjusting the earlier evolved bias of the leg-controller stance neuron, as it now had an additional connection on it (from either of the output, O, neurons).

Genotype to phenotype conversion

The genotype to phenotype conversion was similar to that of the two earlier evolutionary searches. The exception was that the genome was a bit longer (see figure 3.10).

Figure 3.10: The genome layout for turning evolution.

<pacemaker genes>	<locomotion controller genes>
<turning controller genes>	

(a) The basic layout of the genome, after addition of the turning controller genes.

θ_{S_1/S_2}	τ_{S_1/S_2}	θ_{O_1/O_2}	τ_{O_1/O_2}
$S_i \rightarrow O_i$	$S_i \rightarrow O_{+/-1}$	$O_i \rightarrow O_{i+/-1}$	$O_i \rightarrow L/R - leg$

(b) The turning controller part of the genome. Note that some genes are shared by several phenotype features.

Fitness evaluation

The task was to make the insect walk towards some odor source. To measure the fitness of this, the following simulation setup was used:

At each evaluation, a new Quickbeam was created and a new CTRNN corresponding to the genome was connected to the creature, just like for earlier evolutions. In addition to this, a food patch was introduced in the simulated world. Quickbeam could "smell" the odor from this patch using the flagellum. A mathematical function describing the distribution of"odor" from the food was set to be the simulated sensory reading of each of the antennas. The function was:

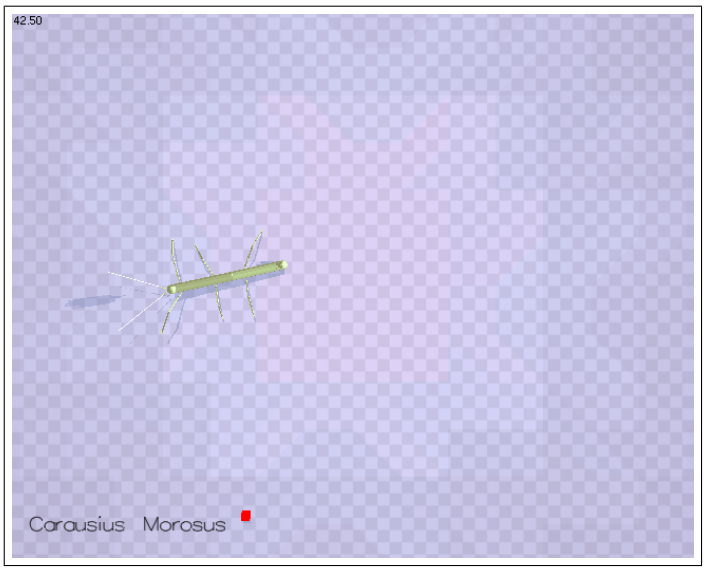
$$\frac{150 - A_{distance(target)}}{150} \cdot 5 \tag{3.11}$$

for each antenna (A). Considering a sensor range of about 300 (Breve measure of length) the sensor reading was therefore in the range -5 (for a distance of 300) to 5 (for a distance of 0). This allows for a high sensitivity, because the range is within the most sensitive range of a sigmoid activated neuron. A snapshot from the simulation is shown in figure 3.11.

Quickbeam was allowed to walk two times for 400 neural network integrations (four seconds) in this environment, and the decrease in distance to the food-patch was measured every time. The first walk was with the food-patch to the left, the second was with the food-patch to the right. This double-evaluation scheme was used to avoid the population to converge toward a local optima where all creatures are good at walking just to the left/right. This could happen if the initial position of the legs was abused, for example. The aggregated decrease in distance to the food was finally returned as the fitness of the creature.

Because the population was initialized with creatures that were close to walking in a straight line, there should exist many creatures that would do walking. Most should just wander somewhere, perhaps only marginally affected by the sensory readings, and will over time become extinct. The walks would however be expected to be quite different, because the dynamics of the (center-crossing and bifurcative) networks should in theory be wide. Because not all those who wander are lost, the search will over time begin to focus only on individuals that actually approach the food-patch.

Figure 3.11: A creature beginning to approach a simulated food-patch.



The genetic algorithm

A summary of the different GA parameters can be found in table 3.3.

3.1.6 Evolving obstacle avoidance (t_3)

In the real-world, it is not always possible to walk directly toward some goal. There may be obstacles that force an agent to at least temporarily walk in a different direction before the goal is reached. To allow Quickbeam to cope with such environments, a system for enforcing walking around obstacles is necessary.

Population initialization and network structure

To enable Quickbeam to extract tactile information from his environment, appropriate sensing devices are required. Just like for olfactory sensing, the system should be able to turn in specific directions based on sensory input,

Table 3.3: Parameters of the GA for evolving turning.

Function	**Description**
Fitness evaluation	Highest ability to approach a food-patch for 400 neural network integrations.
Genotype representation	Vector of 24-bit unsigned bytes (one byte per gene).
Initialization	Pseudo-random genes. First genes were placed close to result from by pacemaker evolution. Other were adjusted according to the P. Computatrix controller. Others again were placed close to center-crossing networks.
Mutation	Random add/subtract x (where $0 < x < 100000$) from gene.
Mutation probability	5% of the offspring.
Number of offspring	2.
Parent selection	2 from best quartile of population (= $2/25 = 8\%$ of best).
Population size	100 individuals
Recombination	Two-point crossover.
Survival selection	Replace worst 2 of population.
Termination condition	250 generations elapsed.

so a tactile sensor was added to each of the antennas. The sensors were then connected to two sensor neurons, so that antennal contact would lead to a increase in membrane potential of each sensor neuron. No contact would lead to a decrease in the potential. If the sensor input should have any effect on turning, it must in some way interfere with the legs. Because the insect is already composed of a set of subsuming modules (e.g. individual legs subsumed by locomotion, locomotion by olfactory turning) it should be straightforward to subsume the olfactory turning system too. This way it should be possible to refuse the olfactory turning system to turn in a specific direction when tactile sensing reveals that it is not actually possible to walk in that direction. A synaptic connection going from the tactile turning system into the output neurons of the olfactory system should be a sufficient minimum to achieve this.

To keep a temporal memory of tactile information, a set of interconnected hidden neurons were also added, analogous to what was done in the olfactory turning controller. These neurons should then together be able to take on several network states. The output of these hidden neurons were then fed into the output neurons of the olfactory turning controller to subsume it.

In addition to the network topology described above, another small addition to the network was made. Because the position of the front legs can tell something about the direction the insect is walking in, it can be interesting to include angle information from the fore legs. If the left leg is in a forward position, then the insect is (potentially) moving right, for example. Indeed, real C. Morosuses do at least have temporal connections between the fore legs and their antennas ([Krause and Dürr, 2004] and [Dürr et al., 2001]). For this reason, one connection from each of the forward angle sensors to every hidden neuron was added. The complete suggested network topology is shown in figure 3.13.

Genotype to phenotype conversion

The genotype to phenotype conversion was similar to that of earlier evolutions. The only difference was conversion of the newly added genes. The complete gene can be seen in figure 3.12.

Figure 3.12: The genome layout for evolution of obstacle avoidance.

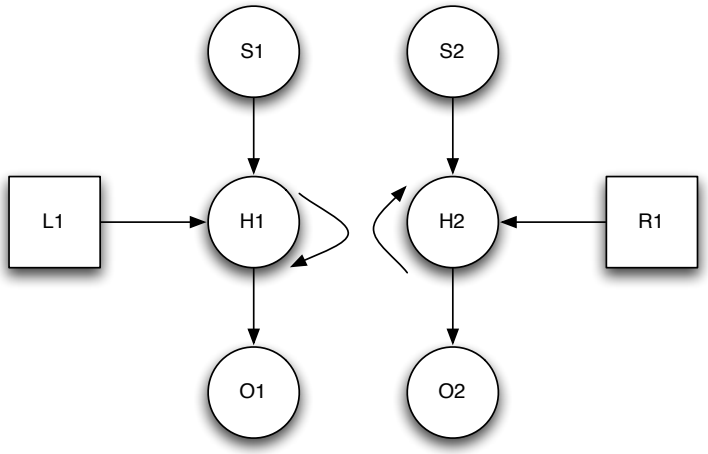
<pacemaker genes>	<locomotion controller genes>
<turning controller genes>	<obstacle avoidance genes>

(a) The basic layout of the genome, after addition of the obstacle avoidance genes.

θ_{S_1/S_2}	τ_{S_1/S_2}	θ_{H_1/H_2}	τ_{H_1/H_2}	
$S_i \rightarrow H_i$	$H_i \rightarrow O_i$	$L_1 \rightarrow H_1$	$R_1 \rightarrow H_2$	$H_i \rightarrow H_i$

(b) The obstacle avoidance controller part of the genome. Note that some genes are shared by several phenotype features.

Figure 3.13: A network for avoiding obstacles. S1 and S2 are tactile sensors. L1 and R1 are (forward angle sensors of) the front leg controllers. H1 and H2 are "hidden" neurons that merge sensory information and keep a temporal tactile memory. O1 and O2 are the output neurons of the olfactory turning system.

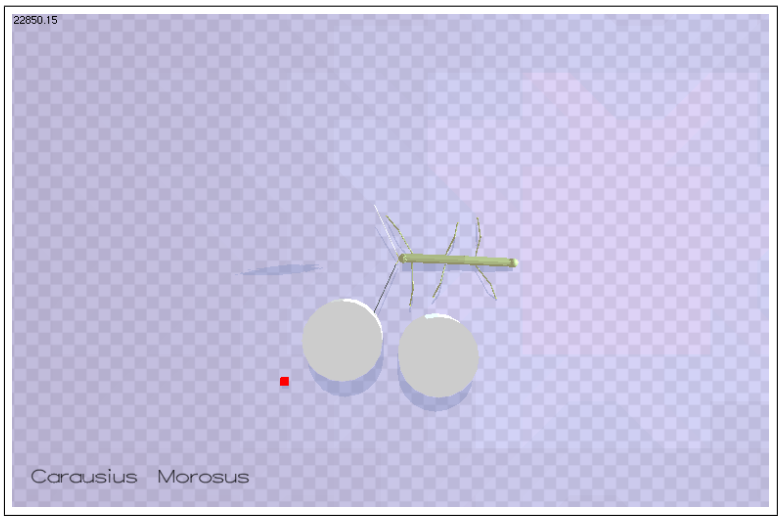


Fitness evaluation

To evaluate the fitness of the creature, the following simulation setup was used:

For each fitness evaluation a new Quickbeam was created and a CTRNN corresponding to the genome was created. In addition to the food-patch in the evolution of turning, two obstacles were added. The creature had to find the way around these. To encourage the evolution of temporal memory, the obstacles were placed next to each other, with the second obstacle slightly more in the way than the first one. Temporal memory is useful in this situation, because remembering to keep straight for a while after the first contact would be beneficial. The ideal path is to just slightly touch the first obstacle, then remember to keep straight on and then slightly touch the second obstacle. The environment did also penalize creatures *not* subsuming the olfactory turning controller quite hard. If the insect blindly followed odor scent, it would crash into the obstacles, and most probably get stuck between them.

Figure 3.14: Avoiding two obstacles.



It was further assumed that the creature was now capable of turning both right and left, and for this reason only one direction was tested for each evaluation. There was however a 50/50 chance for the wall to be on either the right or the left side of the creature. The returned fitness measure was the difference in start-off distance to the food-patch, and end distance to the food-patch. The simulation was run for 2500 CTRNN time-steps (25 seconds). A snapshot from the simulation can be seen in figure 3.14.

3.2 Results

As mentioned in the introduction of this chapter, there are basically two questions to answer when evaluating the results of the evolutions: 1) Does the evolutionary search technique provide a usable solution for the task? and 2) Does the technique provide a solution faster and/or better than other (traditional) techniques?

In an attempt to elaborate on these two questions a series of comparative analyses are presented here. Using naive search (random and/or non-incremental) as a basis for comparison, each of the techniques are evaluated in turn.

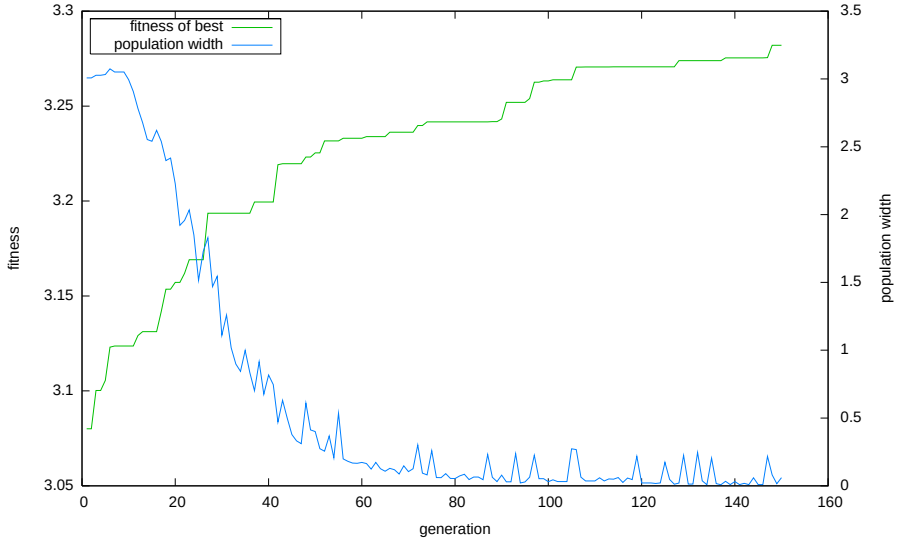
3.2.1 Evolving a rhythmic CTRNN (t_{1_1})

The solutions applicability to the task

The task of evolving a rhythmic CTRNN was to generate a CPG for each leg controller of the simulated C. Morosus insect. Because it was not desirable to do this evolution in the body of the C. Morosus directly, a surrogate referred to as the "pacemaker creature" was used. To provide reliable results, the GA was run 15 times using this creature, and on every of these runs, highly fit rhythmic CTRNNs were found, and the creatures managed to "walk" reliably using several steps with these networks. Even at the first generation, the populations' best individual walked using several steps, but the fitness also continued to increase toward generation 150. The average fitness improvement for all 15 runs is shown in figure 3.15.

Because a surrogate for the C. Morosus creature was used, it is difficult to evaluate the solutions applicability to the final task. The background from Beer [1990] does however provide a set of observations from real pacemakers (Kandell 1967 in Beer [1990]). These observations make up a very good basis for at least comparing this pacemaker to the pacemaker of Beer [1990]. The argument for this is simple: If the pacemaker evolved here exhibit the same properties as the single pacemaker cell of Beer [1990], then it should work similarly in a similar leg controller.

Figure 3.15: Fitness (Breve measure of length traveled) of best individual in population, averaged over 15 runs.



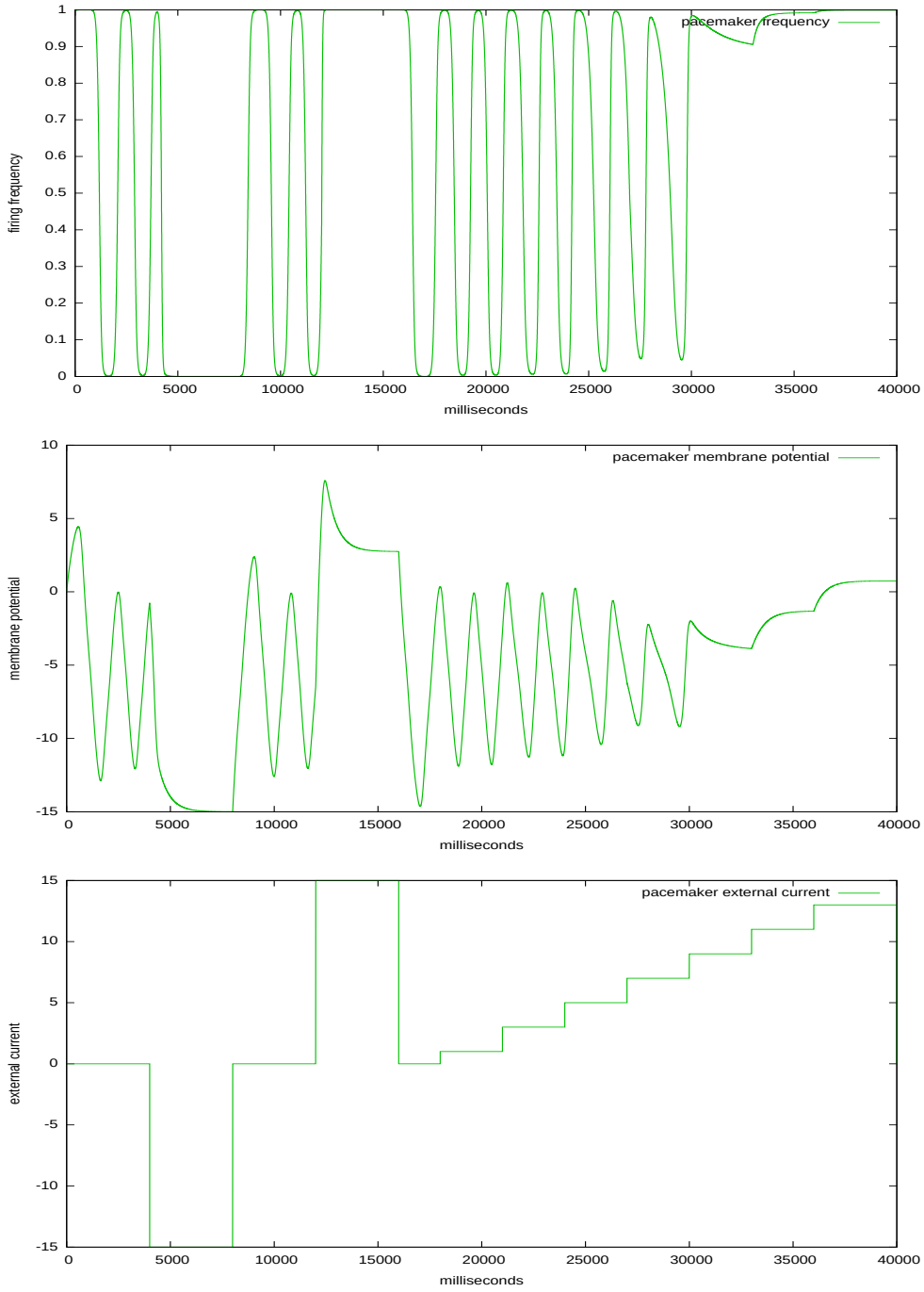
To study the existence of any of these properties, artificially injected currents were applied to the pacemaker. The purpose of these currents were to simulate synaptic inputs to the cell. The effect on the membrane potential and firing frequencies of the cells were then observed, and the resulting behaviour of the circuit can be seen figure 3.16. This behaviour can be interpreted as follows, based on Kandells observations (printed in *italic*):

* *A sufficiently hyper polarized cell is silent.*
This is certainly true. When injected with a current of -15, the cell is turned completely silent (milliseconds 4000 to 7000). A value of -15 is strong, but plausible, as the connections to the pacemaker have a weight in the range $[-20\ 20]$.

* *A sufficiently depolarized cell fires continuously.*
This is certainly true. When injected with a current of 15, the cell fires continuously (milliseconds 13000 to 16000).

* *Between these two extremes, it rhythmically produces a series of relatively fixed duration bursts, and the length of the interval between bursts is a continuous function of the injected current.*
The first part is certainly true. The cell(s) successfully produces fixed du-

Figure 3.16: Behaviour of the pacemaker circuit (IO-neuron reading), depending on injected current into the IO-neuron.



ration bursts. The other part of the question is debatable. There does not seem to be any *continuous* dependency between injected current and burst frequency. At least not when current is injected only into the IO-neuron (and not into the hidden neuron as well) (milliseconds 16000 to 40000).

* *A transient depolarization which causes the cell to fire between bursts can reset the bursting rhythm.*
This is true. After a 4 second depolarization that makes the pacemaker fire prematurely (milliseconds 11000 to 15000), the network falls back into a new rhythm.

* *A transient hyper polarization which prematurely terminates a burst can also reset the bursting rhythm.*
This is true also (milliseconds 4000 to 8000).

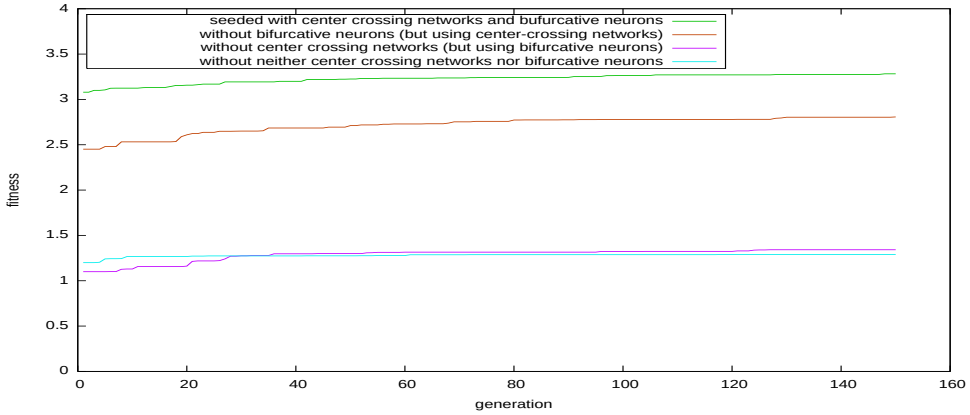
It therefore seems that to some extent, the GA provide a usable solution to the task. The results from the evolution of the locomotion controller (section 3.2.2) will show if this functionality is actually sufficient for coordinating legs into stable gaits.

Comparative analysis

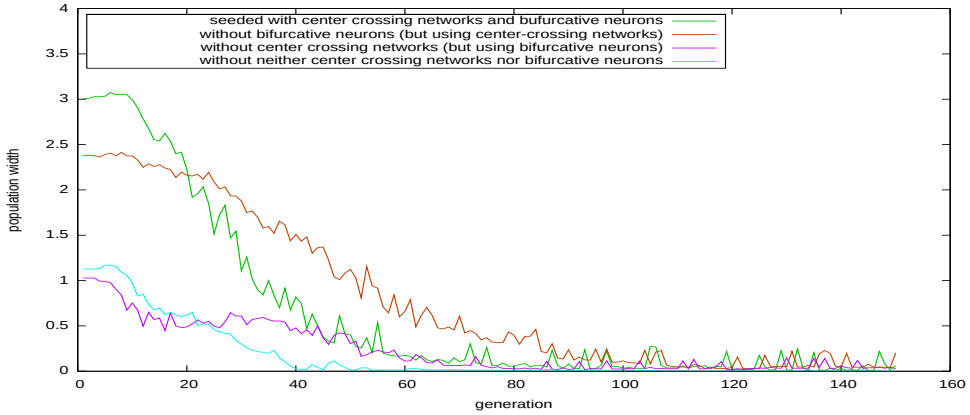
The second major question to ask, is if the results really are a consequence of the seeding techniques applied. Do they really lead to faster evolution of a CPG? Is the quality of it any better than for solutions from other techniques? To approach an answer for either of these questions, it can be interesting to compare the average quality of different techniques, using a random (non-seeded) evolutionary search as a common reference. To collect data for comparison, additionally 15 evolutionary searches without each of the techniques were run. So, all other things being equal, 15 runs without center-crossing networks were run, 15 runs without bifurcative neurons were run, and finally 15 runs without neither of the techniques were run. The results are shown in figure 3.17 and table 3.4.

As can be read from the graphs, the results of seeding the population with center-crossing networks are very promising. The fitness of the best CTRNN is more than twice the best of the random CTRNNs. This is true even after the first generation, so the GA has arrived at better results faster too. It also seems that center-crossing networks really help for maintaining pop-

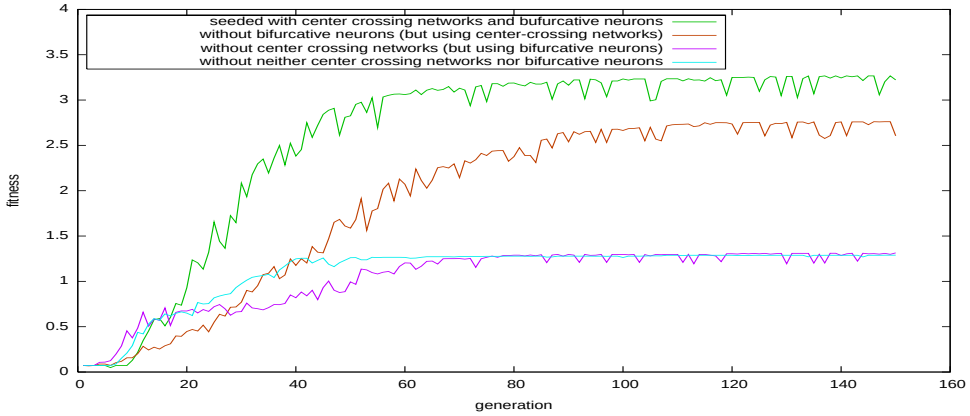
Figure 3.17: Averages after 15 runs.



(a) Best individual per generation.



(b) Difference between best and worst individual per generation (population width).



(c) Worst individual per generation.

Table 3.4: Best pacemaker circuit after 1 and 150 generations (G) using different search techniques. Values are in mean (μ), standard deviation (σ). Additionally the standard deviations percentage of the mean (relative standard deviation / coefficient of variation, C_v) is included for easier comparability.

	Center-Crossing				**Non-Center-Crossing**			
	Bifurcative		**Non-Bifurc.**		**Bifurcative**		**Non-Bifurc.**	
G	1	150	1	150	1	150	1	150
μ	3.081	3.282	2.451	2.807	1.100	1.342	1.200	1.293
σ	0.300	0.219	0.407	0.251	0.347	0.583	0.353	0.354
C_v	9.725%	6.661%	16.61%	8.930%	31.52%	43.47%	29.39%	27.40%

ulation width. From figure 3.17a it can be seen that the center-crossing searches generally continue to introduce better solutions for a larger number of generations than the other searches. It can also be seen from 3.17c that the center-crossing searches continue to introduce *worse* solutions. The random/bifurcative-only solutions seem to not introduce much new at all, which is a result of convergence into a local optimas. This is also seen from the extremely high standard deviation for the last generation of these searches. The results should come as no surprise, as they correspond well to what has been found on earlier research of center-crossing CTRNNs (see Mathayomchan and Beer [2002]).

Another, perhaps even more interesting result (with more novelty value), is that populations with **bifurcative neurons outperform non-bifurcative neurons qualitatively**. As stated in section 3.1.3, it was expected that a population with enforced bifurcative neurons should have a "head start" of other neurons because they were already capable of maintaining two possible states to flip between. It was somewhat surprising that bifurcative neurons were still ahead of non-bifurcative neurons after 150 generations. After all, neither the bifurcative condition nor the center-crossing condition were enforced throughout the search, the neurons were only placed in a estimated fruitful region of the search space. In theory the center-crossing networks should also be able to reach this region of the search space, though perhaps a bit slower. The center-crossing-only search does however not appear to approach the same performance as the search with both center-crossing *and* bifurcative neurons.

The increased performance when seeding with bifurcative neurons could be due to the more complex dynamic properties of the networks (several steady states per neuron), which would result in higher diversity (figure 3.17b), and therefore allow recombination to make better progress (avoid local optimums, see section 1.5). However, this does not explain the much better fitness already after generation 1 (two of the searches, 3 and 7, already found an optimum at generation 1). So there must be other factors that make a difference too.

A more elaborative explanation is that a rhythmic CTRNN with this topology *must* have at least one bifurcative neuron to exhibit pulse behaviour. As outlined in section 3.1.3 (page 43), the working of such a rhythmic CTRNN can then at least be simply explained. This assumption also allows for some other plausible interpretations of the graphs:

- If at least one bifurcative neuron is required, then providing several such neurons in the initial population should be highly beneficial. If these neurons additionally are maximally sensitive to change, the neurons can easily cooperate into pacemaker behaviour. This explains why the seeded searches are initially good, but shows only slight improvement for the remaining generations (the neurons are already placed in or near an optimum).

- If at least one bifurcative neuron is required, then a genetic search *not* seeded with bifurcative neurons will have to find one through recombination and/or mutation. This explains why the search seeded with center-crossing networks (but without bifurcative neurons) perform quite well, but uses longer time to improve. It also explains the slower convergence of the population as a whole for non-bifurcative populations.

- It is natural that populations seeded with bifurcative neurons do not perform well unless they are combined with center-crossing networks. Bifurcative neurons alone cannot easily realize their true potential unless the bias of each neuron is in a range where the neuron respond to input. Non-responding neurons will simply never be able to flip between states. This explains why only some of the bifurcative-only searches (table 3.4) reach a proper result, while most of the searches perform poorly. The result is a very high standard deviation from the average result.

As a summary, the genetic search seeded with both center-crossing networks *and* bifurcative neurons have a high probability of resulting in a population that represent a wide range of good solutions to a pacemaker CTRNN. It is natural that this leads to good fitness values in a genetic search that is focused on the evolution of a pacemaker. As can be seen from table 3.4, this combination not only produces the best results on average. It also provides results most steadily of all searches (it has a high probability of converging toward a good solution).

3.2.2 Evolving a locomotion controller (t_{1_2})

The solutions applicability to the task

The task of this evolution was to find proper weights, biases and time-constants to make the locomotion controller of Beer [1990] work for a simulated Carausius Morosus. A GA was run 15 times on this problem, and every one of these searches succeeded in making the insect walk reliably. The resulting fitnesses can be seen in figure 3.18 and table 3.5.

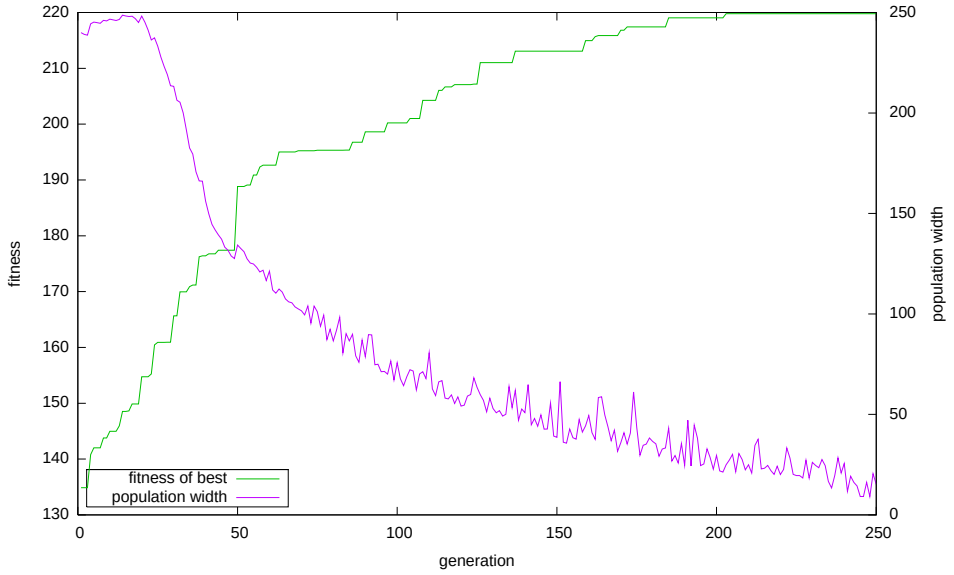
Comparative analysis

Table 3.5: Fitness of best after 1 and 250 generations (G) for populations with and without individuals with bias adjusted to the center-crossing equation. Values are the mean (μ), standard deviation (σ) and relative standard deviations (C_v) for 10 runs.

	With c.c. bias		**Without c.c. bias**	
G	1	250	1	250
μ	134.9	219.8	18.58	40.60
σ	17.61	28.32	9.828	26.96
C_v	13.06%	12.88%	52.90%	66.41%

The network topology of the locomotion controller was copied from the controller of Beer [1990]. For reasons explained before (section 3.1.4, page 51) the technique of enforcing bifurcative neurons did not apply to this evolution. It is also difficult to enforce center-crossing networks, when many of

Figure 3.18: Fitness of best creatures when evolving the locomotion controller. Averaged over 15-runs.

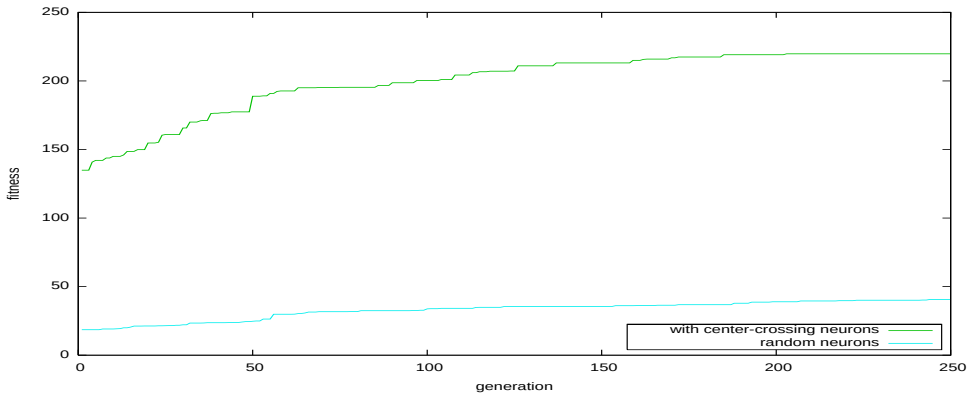


the neurons are sensor neurons (which are not influenced by other neurons). The two techniques left to be inspected were therefore the use of incremental evolution and adjustment of bias (as if we *had* center-crossing networks, see section 3.1.4) for some of the neurons. Incremental evolution is best compared together with the other incremental tasks, and will be considered in section 3.2.5. The only interesting result left here is therefore about the enforced bias.

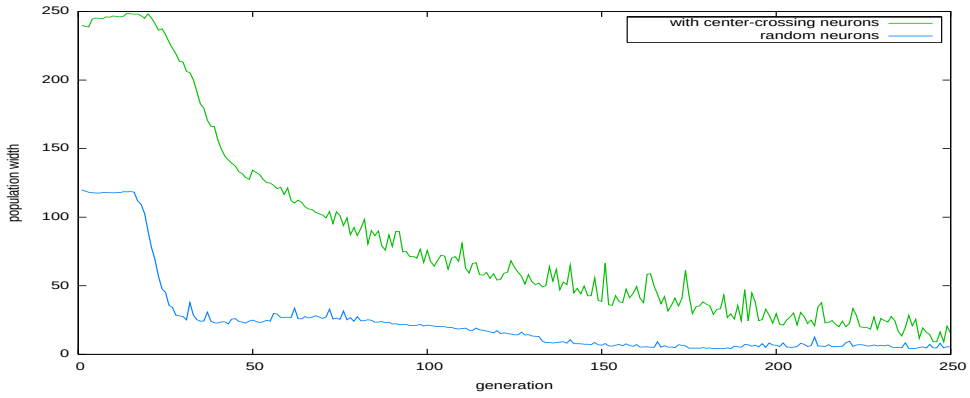
To find differences between evolutions with and without enforced center-crossing-like bias, the same experimental technique was applied as for the pacemaker evolution. 15 evolutionary searches were run with the bias adjusted according to equation 3.5 (page 48), and 15 evolutionary searches with random biases were run. For all of these runs, the pacemaker part of the genome was randomized in the neighborhood of the earlier evolved result. In sum, anything but the bias adjustment was kept equal for the two different types of evolution.

From figure 3.19 we can see that, just like for pacemaker evolution, bias adjustment according to the center-crossing equation (3.5) leads to a search

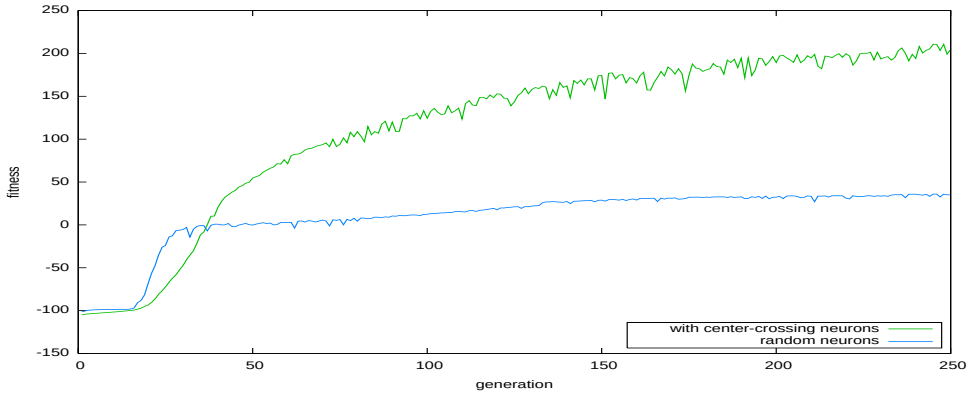
Figure 3.19: Comparing the evolution of a leg controller. Averaged over 15 runs, with and without adjusting the bias according to the center-crossing equation.



(a) Best individual in population.



(b) Population width.



(c) Worst individual in population.

that significantly outperforms populations with random biases. Even after the first generation, the population with adjusted biases have individuals performing about three times better than the best individual in a random population does after 250 generations. For generation 250 of the search with adjusted biases, the performance has reached almost five times the fitness of the random search.

For reasons described earlier (chapter 1), it should in general be beneficial for a genetic search to maintain a diverse population. It may sound counter-intuitive that a specific enforcement (of bias) actually helps maintain diversity, but the averages of figure 3.19b shows that this holds for the evolutionary search done here, just like it did for the pacemaker evolution. Already in generation one the population with adjusted biases has more than twice the width of the random population. It also maintains a higher width throughout the whole search. Why does this happen?

Looking at figure 3.19 hints at an answer. From graphs a and c, it can be seen that populations with adjusted bias have a steady removal of bad-performing individuals.[5] The random search, on the other hand, seem to follow a "platform progression" where improvement is done in stages. From this it seems that **random genome tend to converge towards local optima**. This is also supported in table 3.5 where it can be observed that random searches end up in a very wide range of fitnesses (very large ratio of standard deviation), while seeded searches generally end up in a more narrow range (low ratio of standard deviation) of (better) results.

An explanation of this effect, is that random searches at times find network parameters (biases) that makes one (or a few) neurons capable of exhibiting interesting (improved) behaviours. The search then quickly approaches a CTRNN where this situation is maximally abused (an optimum). No further progress is then made until the same happens again for some other neuron. Given the 5% mutation rate, and the very low chance of actually *properly* mutating the correct gene (there are only 4 biases, but 25 genes total), it should be quite evident that chances are small that any further improvement will happen within 150 generations.

[5]Remember that replacement is done by swapping the worst two individuals in the population with the new offspring of the best quartile of the population. This replacement scheme is the same both for evolution with and without adjusted biases, of course. The performance of the worst individual therefore points to how diverse solutions recombination creates for each generation.

Table 3.6: Performance of best individuals. Mean (μ), standard deviation (σ) and relative standard deviation (C_v) for 10 runs.

	Center-Crossing				**Non-Center-Crossing**			
	Bifurcative		**Non-bifurc.**		**Bifurcative**		**Non-bifurc.**	
	G. 1	G. 250	G. 1	G. 250	G. 1	G. 250	G. 1	G. 250
μ	74.25	122.8	71.93	121.6	61.39	92.73	42.98	84.87
σ	22.81	24.84	14.64	16.95	29.09	26.00	9.562	30.18
C_v	30.71%	20.23%	20.34%	13.94%	47.38%	28.04%	22.25%	35.56%

For the searches with biases adjusted to the center-crossing equation, the story is quite different. All such searches already have neurons that are maximally sensitive. The evolutionary search is therefore mainly focused on moving biases (just enough) *away* from this situation to have an optimum. It is natural that this progresses in a more steady manner, because the biases of all neurons are moved at away from sensitivity at the same time. Such a search will not suddenly jump into better fit regions and drop population width. The searches therefore generally end up in a narrow range of much better-performing results.

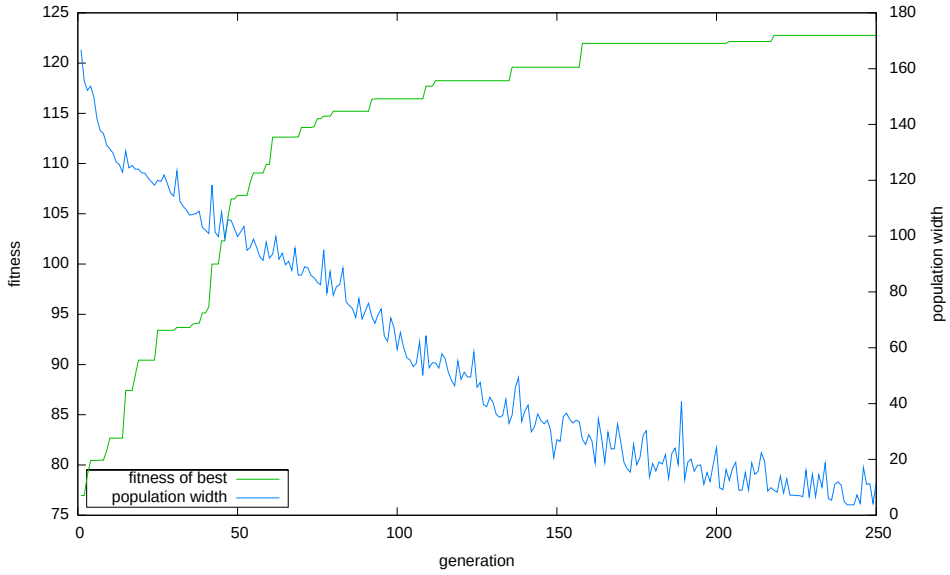
3.2.3 Evolving turning (t_2)

The solutions applicability to the task

The task for evolving turning was to learn to move towards a simulated food patch. A GA was run 10 times on this task, and for every of these runs evolution succeeded in finding CTRNN properties that made the insect walk significantly closer to the food patch. The fitnesses did however differ some for each run. A graph for the fitness of the best performing individuals, and a statistical sum up of the 10-runs, are shown in figure 3.20 and table 3.6 (page 81).

The solution seems applicable to the task, but why did evolution end up with exactly this solution? As outlined in section 3.1.5, the general idea was to have a network of two neurons that could force each other into different states depending on the difference between the two sensory readings (right/left). This way the network should be able to switch between inhibit-

Figure 3.20: Performance of best individuals averaged for 10 evolutionary searches.



ing the right/left middle leg, and thereby enforce the insect to turn or just walk straight forward. To inspect whether this was what was really going on in the network, the following mini-study was done, based on the best genome from any of the runs (run 2):

- Two artificial "probes" were connected to the right and left hidden neurons in the insects turning lobe, so that the neurons firing frequencies could be read.

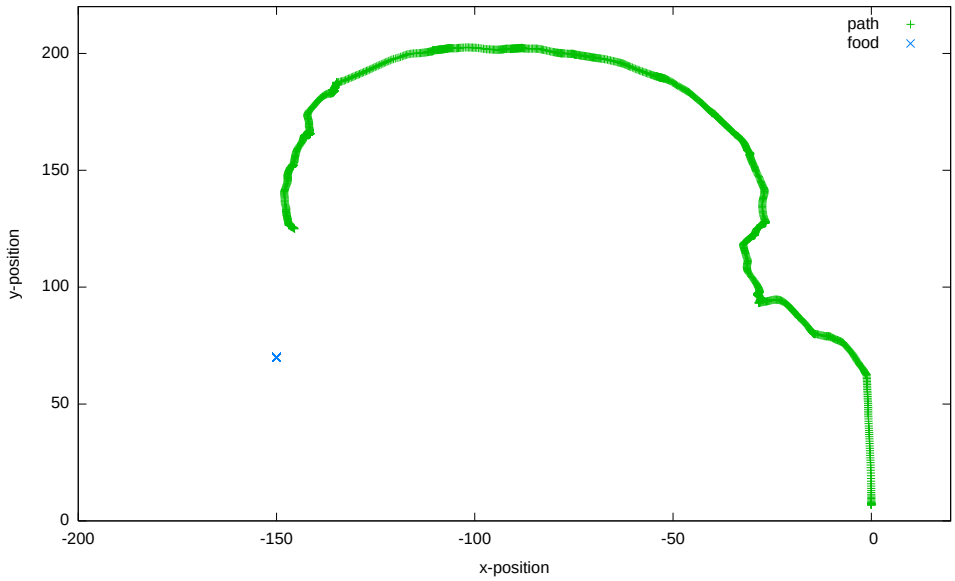
- A slightly adjusted fitness evaluation run was set up: The creature was allowed to walk for 70 seconds, and during that time the following happened:

  - At time step 0, a food patch was put to the *left* of the insect.

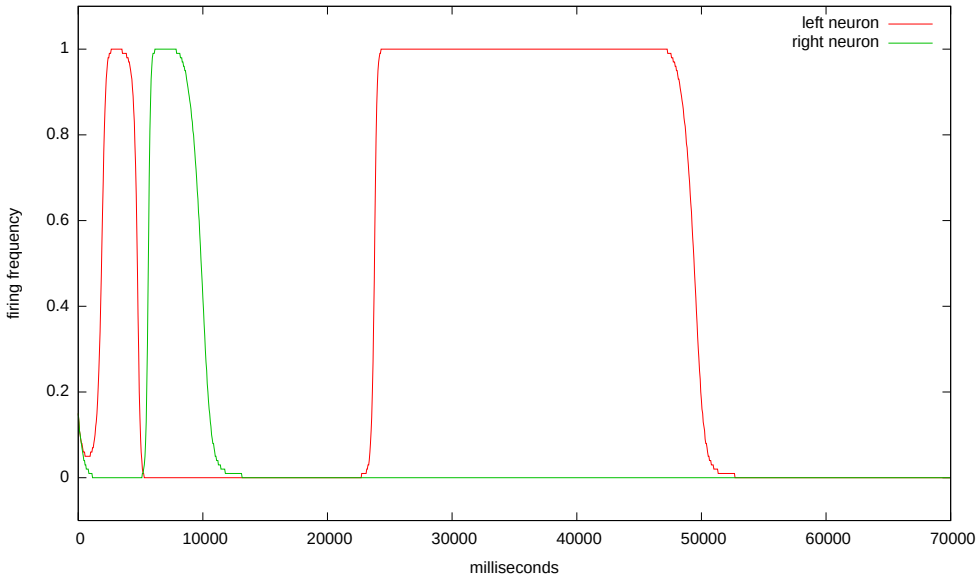
  - After 2.50 seconds, the food patch was moved to the *right* of the insect.

  - After additionally 3.75 seconds, the food patch was moved back to the *left* again.

Figure 3.21: Studying the behaviour of the turning controllers' subsumption of the locomotion controller.



(a) The path chosen.



(b) The firing frequency of the neurons in the turning lobe.

- Between seconds 14 and 15 the patch was moved swiftly (1/2 second) right → left → right to simulate a sensor disruption.

- For every 0.5 seconds the position of the insect was recorded (figure 3.21a).

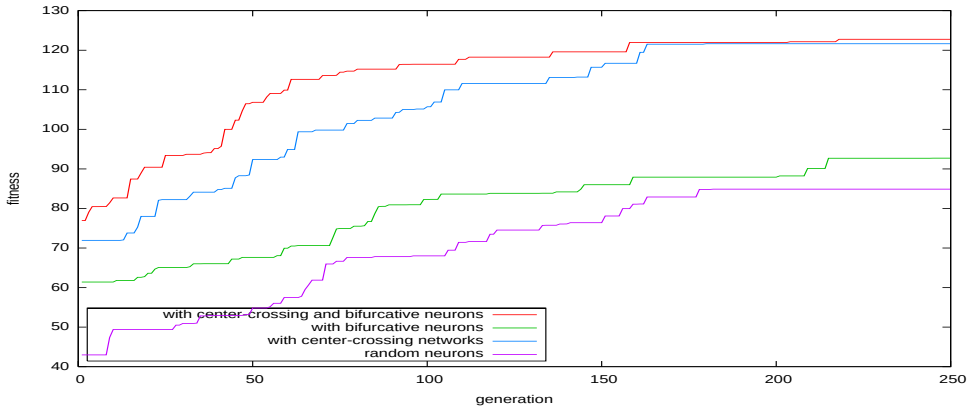
The resulting behaviour of the insect can be seen in figure 3.21. At initialization, both neurons for a short time seem to adopt a non-firing state. The insect soon start to fire strongly on the left neuron though, because of the strong left sensor reading. The insect then starts to move left. After 2.5 seconds the food patch is moved to the right and **after some short period of time** the insect drops the commitment to move left, and shortly thereafter starts to excite the neuron that will force the locomotion controller to walk right instead. The food is however moved back to the left again, before movement to the right is really begun. The insect then moves neither right nor left for nearly 10 seconds. This most probably happens because both neurons fire quite strong now (the insect is nearer the food), so neither of the neurons strongly suppress the other. After some time the left neuron takes over though, and the insect starts moving left until the food-patch is right in front of it (about second 45). The sensory readings are then more or less equal for a while, so both neurons eventually adopt their non-firing state, and the locomotion controller can move straight forward without being subsumed.

As can be seen from figure 3.21b, the short "sensor disruption" have no visible effect on the left hidden neuron of the turning lobe. Walking is not altered, so the net seems quite robust to at least minor disruption.

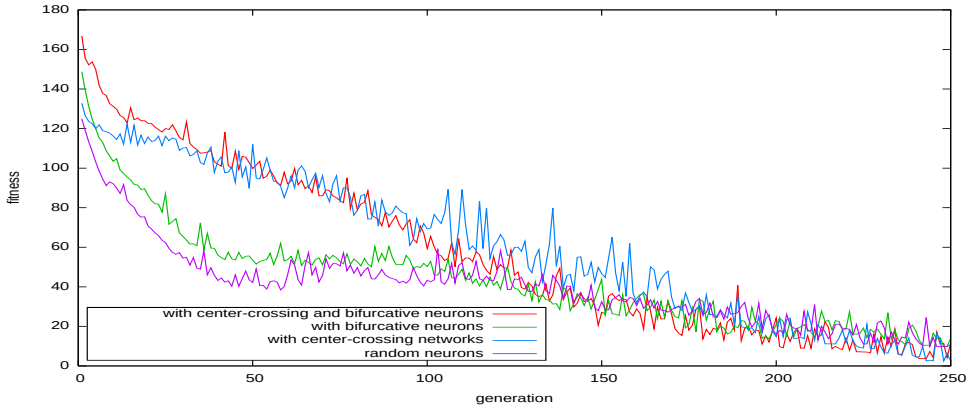
Comparative analysis

It is very interesting to study the evolution of turning when populations are initially seeded with bifurcative neurons and center-crossing networks. Looking at the comparison between the best performing individuals per generation (figure 3.22), it bears a clear resemblance to the evolution of the CPG from section 3.2.1. The best performance is when bifurcative neurons are combined with center-crossing networks. Next to this is the center-crossing-only search, and then the bifurcative-only search. Random search perform worst.

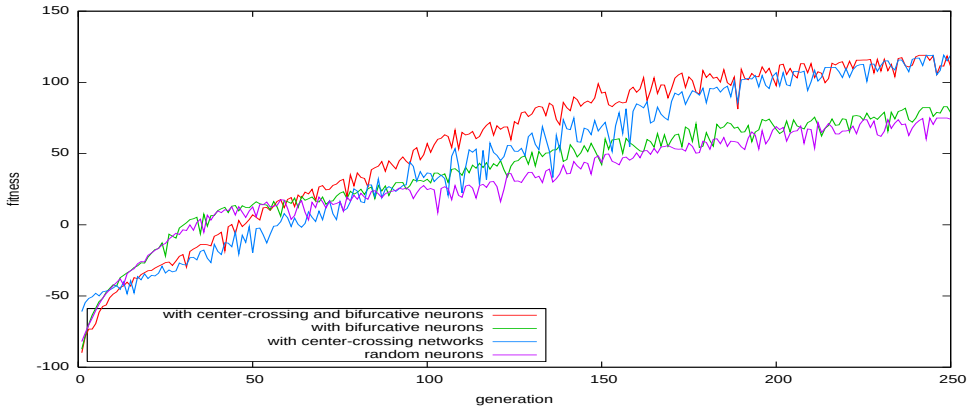
Figure 3.22: Comparing the evolution of a turning controller. 10 runs average.



(a) Best individual in population.



(b) Width of populations.



(c) Worst individual in population.

From figure 3.22 is is again quite clear that evolutions that are not seeded with center-crossing networks remove bad individuals faster than the best are improved. Just like for the earlier evolutions, random search proceed in stages, where the GA converge toward "platforms" of locally optimal solutions. These optima may differ a lot in quality, and the result from this can be seen in table 3.6 where the ratio of standard deviation is very high for such searches. They actually end up in a wider range of results than they start with. This happens because the searches converge toward local optimums, and the fitnesses of these optima vary a lot from search to search. The evolutionary searches seeded with **center-crossing networks seem to avoid local optima**, and the worst individual is removed about equally fast as the best is improved. The fully seeded search (bifurcative neurons *and* center-crossing networks) can therefore continue to improve, even though it starts off with solutions that perform about equally well as the best random solutions. As a result, these searches end up in a narrow range (low ratio of standard deviation) of well-performing solutions.

3.2.4 Evolving obstacle avoidance (t_3)

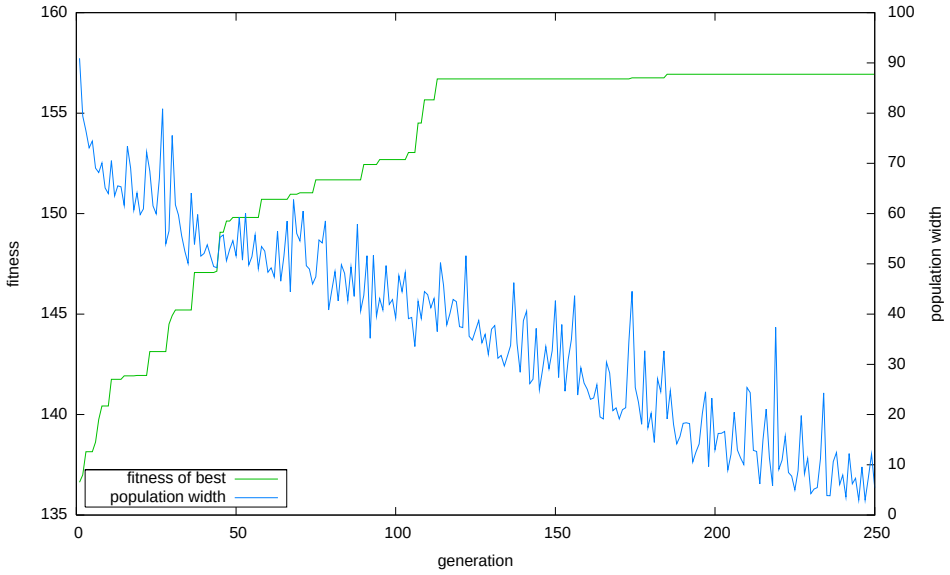
The solutions' applicability to the task

The task for evolving obstacle avoidance, was to approach a simulated food patch at the same time as avoiding obstacles that could be sensed using tactile probes (antennas). An evoutionary search was run 10 times, and throughout all of these runs, the insect improved its walk so that it could get closer to the food.

Comparative analysis

The evolution of obstacle avoidance was quite similar to the evolution of turning. For space and time considerations, the GA was therefore only run 10 times using both center-crossing networks and bifurcative neurons. The result of which can be seen in figure 3.24(b). As can be seen in the table of the figure this leads to searches that in general starts off within a wider range of results than it ends with (it ends with a lower ratio of standard deviation). This is analogous with the other evolutions using these two

Figure 3.23: Best performing individuals for 10 evolutionary searches.



(a) Best performing individual in population, averaged over 10 runs.

	Best in generation 1	**Best in generation 250**
μ	136.6	156.9
σ	8.747	7.208
C_v	6.402%	4.594%

(b) Performance of best individuals. Mean (μ), standard deviation (σ) and ratio of standard deviation from mean (C_v). Averaged over 10 runs.

techniques. The comparison of incremental versus non-incremental evolution of obstacle avoidance is described in section 3.2.5.

3.2.5 A comparative analysis of incremental evolution

The results from evolutions that makes use of incremental evolution has already been described. Each step successfully generated working neural networks. Gomez and Miikkulainen [1997] have already shown that partitioning complex behaviour in a similar manner can make a difference between a good result, and no result at all. Is that true in this thesis also? Does incremental evolution actually make a difference? Does incremental

Table 3.7: Results from evolution of different behaviours after the last generation of each evolution (250 generations for incremental locomotion, 400 generations for non-incremental locomotion, 250 generations for incremental turning, 650 generations for non-incremental turning, 250 generations for incremental obstacle avoidance, 900 generations for non-incremental obstacle avoidance). Values are mean (μ) with standard deviation (σ), and relative standard deviation (C_v).

	Locomotion		**Turning**		**Obstacle avoidance**	
	Inc.	**Non-inc.**	**Inc.**	**Non-inc.**	**Inc.**	**Non-inc.**
μ	219.8	200.0	122.8	70.35%	156.9	116.1
σ	28.32	30.22	24.84	54.05	7.208	12.30
C_v	12.88%	15.11%	20.23%	76.83%	4.594%	10.59%

evolution result in faster discovery of well-performing solutions?

In an attempt to answer these questions, an approach similar to the comparisons of the other techniques was chosen. Leaving everything[6] but the seeding with earlier genome untouched, a series of non-incremental evolutions were run. Three steps were taken to obtain results for comparison:

1. Evolving a leg controller without seeding it with the result of pacemaker evolution genome.

2. Evolving turning without seeding it with the result of pacemaker + leg controller evolution genome.

3. Evolving obstacle avoidance without seeding it with the result of pacemaker + leg controller + turning evolution genome.

A summary of results can be found in table 3.7. Plausible interpretations of these results follow in the next sections.

[6]There are exceptions to "everything". Sometimes with regard to evaluation time, sometimes with regard to population size. The exceptions are described explicitly.

Locomotion

The result of incremental/non-incremental evolution of locomotion is shown in figure 3.24a (page 90). The incremental approach to evolving locomotion started off seeded with genes that were taken from the best pacemaker evolutions. To make up for the 150 evolutions elapsed for evolving the pacemaker, the non-incremental search was allowed a head start of 150 generations. This is reflected in the figure, where no data are presented for the incremental approach before generation 150.

With the exception of (not) seeding the population with genes from t_{1_1}, the two approaches were kept as similar as possible. There are however some obvious exceptions to this. The non-incremental approach have to evolve a pacemaker *at the same time as evolving the rest of the leg*, while most of this job is already done for the incremental approach. A question therefore arises: Is it possible to evolve a pacemaker in this setting?

The fitness measure of pacemaker evolution was designed specifically for evolving pacemakers. On the other hand, the pacemaker was meant to be an integral part of a leg controller! It is therefore difficult to state one of the measures as clearly better than the other. 150 generations of evolution in the exact environment the pacemaker is supposed to work in, can be just as good as 150 generations in a surrogate creature.

There was also one quantifiable difference that is known to affect quality of evolution in general: Pacemaker evolution had a population size of 40 individuals. The locomotion controller population had a size of 100. It is generally advantageous to have a larger population, because it increases the likeliness of having good genome available. A larger population does however take longer to converge, because it takes more generations to replace bad genome. Considering the elitist selection scheme used, this should have minimum effect here though. Bad performing individuals are never recombined anyway (only the best quartile is). It is therefore *probable* that the non-incremental approach actually had a slight advantage in this regard.

As can bee seen from figure 3.24 and table 3.7, the incremental searches quickly catch up with the non-incremental searches. The results are also better on average, and the searches end up at these good values more steadily (12.88% versus 15.11% ratio of standard deviation) than the non-incremental searches. Even though it is clear from figure 3.24 that incremental evolution

outperforms non-incremental evolution on average, it should be noted that both strategies evolve into working locomotion controllers.

Olfactory turning

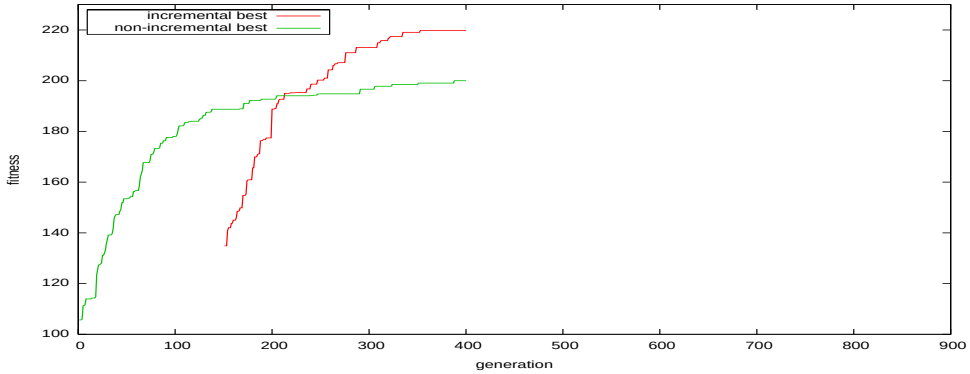
The result of incremental/non-incremental evolution of turning is shown in figure 3.24b (page 90). Just like for the evolution of locomotion, there were a minimum of differences between incremental and non-incremental evolution. Actually there are no differences except the seeding with earlier evolved genome, and no head-start in the incremental one. The two should therefore be easily comparable.

As can be read from the graph in figure 3.24b, the incremental approach outperforms the non-incremental approach by a factor of about two after the last generation. For generation 401, where the incremental approach starts, the best individual in the incremental population is already better than that of the best non-incremental one. It should be noted that the incremental evolutions also generally ended up in a more narrow range of results. There ratio of standard deviation is about 20% of the mean for the incremental approach, whereas the non-incremental approach has a standard deviation ratio of 77% (!) of the mean. In practice this means that **it is neccesary to run several non-incremental GAs to ensure that at least one GA reaches a usable result**.

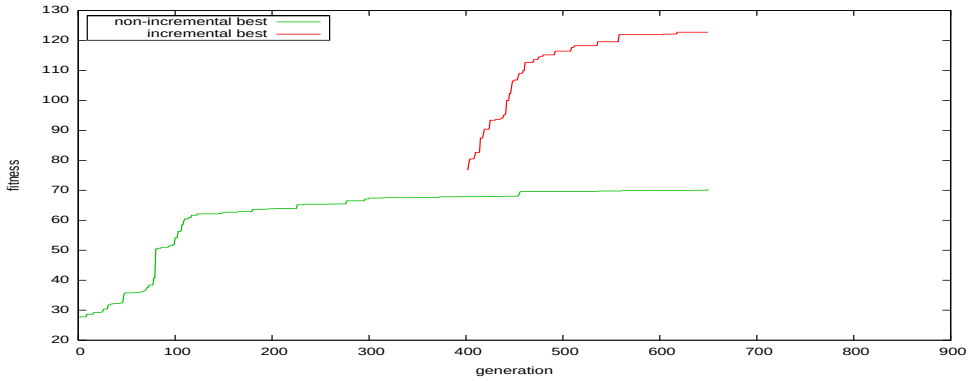
It should be noted that even though the graph show improvement also for the non-incremental approach, the resulting genome does not correspond to individuals exhibiting normally expected behaviours. In general, the evolved genome correspond to individuals that "twitches" toward the food-patch by exploiting some specific property of the environment (e.g. inertia of limbs). Some individuals do movements similar to walking, but they are not stable walks (perhaps with one exception, where the walk was quite stable, but that individual walked in the wrong direction). It is natural that these searches end up in very different results, as can be read from the high standard deviation of these searches.

It is of course perfectly fine that evolution finds solutions that differ from what is expected from the designers point of view. After all, finding "unimaginable" solutions is perhaps the most prevalent advantage of GAs. There is however a potential problem with this when evolving more complex be-

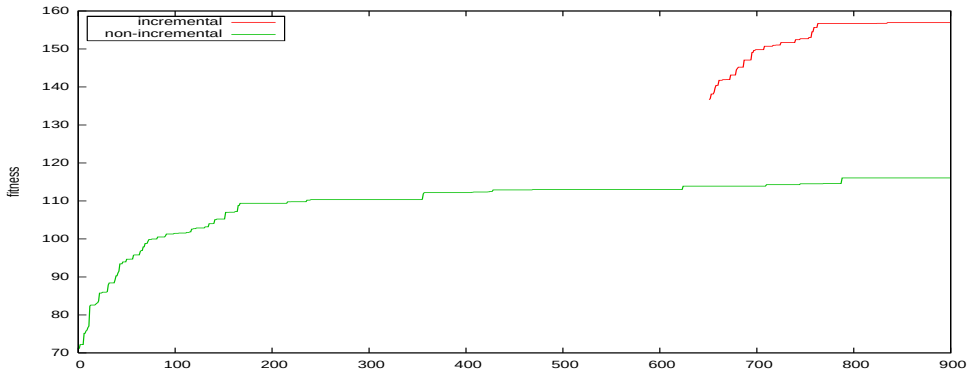
Figure 3.24: Comparing non-incremental evolutions to incremental ones.



(a) Evolution of locomotion. 15-runs averages. The non-incremental approach has a head-start of 150 generations to make up for pacemaker evolution.



(b) Evolution of turning using olfactory sensing. 10-runs averages. The non-incremental approach has a head-start of 400 generations to make up for pacemaker (150) and locomotion controller (250) evolution.



(c) Evolving tactile obstacle avoidance. 10-runs averages. The non-incremental approach has a head-start of 650 generations to make up for evolution of pacemaker (150), locomotion (250), and olfactory turning (250).

haviours. As Gomez and Miikkulainen [1997] point out, evolution of too complex behaviour may end up in discovering mechanical strategies that are unable to generalize to new environments. If evolution discovers such solutions, further steps in the incremental build sequence up may be difficult to take, because the discovered behaviour only performs well in the exact fitness setting that it was evolved in. In other words: Specific solutions to a task is desirable, but it is even better if it can be as general as possible.

The solutions found by incrementally evolving the olfactory turning system, is however not exploiting the physical properties of the exact environment that fitness evolution is carried out in. Rather, strategies that exploit the rest of *the neural system* is found. It is natural that such strategies are discovered in an incremental search, because all individuals are initially exhibiting (or close to exhibiting) rhythmic, coordinated, stepping behaviour, and other minimally cognitive behaviours. These behaviours are therefore already available for the evolutionary search to exploit.

Obstacle avoidance

Similarly as for olfactory turning, the incremental evolution of obstacle avoidance outperforms the non-incremental approach. The results can be seen in table 3.7, and they are also visualized in figure 3.24c (page 90). As can be seen from the graphs, the difference between incremental and non-incremental evolution shows a clear resemblance to the two other incremental approaches. For the first step (locomotion), the incremental search "catches up" and eventually outperforms the non-incremental search. For the second step (turning) the incremental search starts off as good as the non-incremental ends. Finally, for the third (obstacle avoidance) step, the incremental step starts off much better than the non-incremental step ever gets. Evolution then brings the difference in performance even further away from the non-incremental approach.

Also, just like for locomotion and turning evolution, the incremental evolutions seem not only to provide much better results. It also provide these good results in a more stable fashion. The ratio of standard deviation is 4.594% versus 10.59% for this task. As for the other evolutions, a simple yet plausible explanation exist: It is almost impossible to evolve any complex behaviour (such as obstacle avoidance) before other *dependent* minimally

cognitive behaviours (e.g. walking) are already evolved. There are at least two reasons that the minimally cognitive behaviours do not emerge. First of all the number of behaviours increase, and the fitness measure must reflect (and reward) all these behaviours. Designing such a fitness measure is difficult, and the fitness measure used here is at least not sufficient. Secondly, the search space gets larger and larger for every extension of the genome. The population size is however not increased in pace with this (not increased at all here), and therefore the probability of having an initial population with *any creature at all* that exhibit all these behaviours, decreases.

Other general interpretations

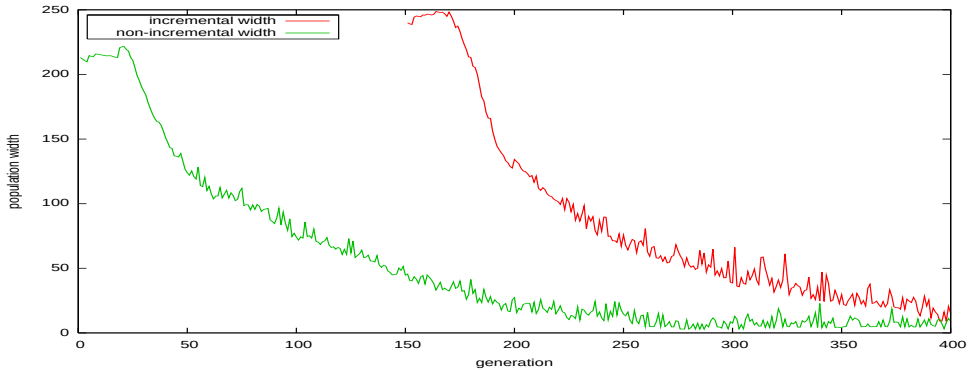
Like for the other two techniques - seeding with bifurcative and/or center-crossing networks - the enforcement of some genes leads to higher population width (see figure 3.25, page 94). This may sound somewhat surprising, because it is counterintuitive that an enforcement into one area of the population leads to higher diversity! One explanation for this phenomenon is that the incrementally evolved population has some very few individuals that perform much better than the average population, but this cannot be the whole story. Looking at figure 3.24, the best performing individual has an average fitness of about 80 for the evolution of turning, while the population width is about 170 for the non-incremental evolution. This means that the incremental evolution starts off with both better *and* worse individuals than the non-incremental evolution.

While this is indeed an interesting phenomenon, it does have a plausible explanation. For non-incremental evolution, the simulations are generally focused on a "dead" bug. It is hardly twitching at all, and certainly not moving very far. This is not very surprising, because chances are small that any individual in such a population should be able to walk. Whenever one or more creatures actually *do* walk, the population will quickly converge toward an optimization of this behaviour, because it is an easy way of increasing fitness. For the incremental approach, simulations are generally focused on moving in the right direction. Even at start time, most individuals manage to move at least a little bit. Moving in the wrong direction is however hardly penalized, and therefore these creatures may actually perform - according to the fitness measure - worse than a "half-dead bug" just lying there. As a result, such a population will often have individuals that are both better

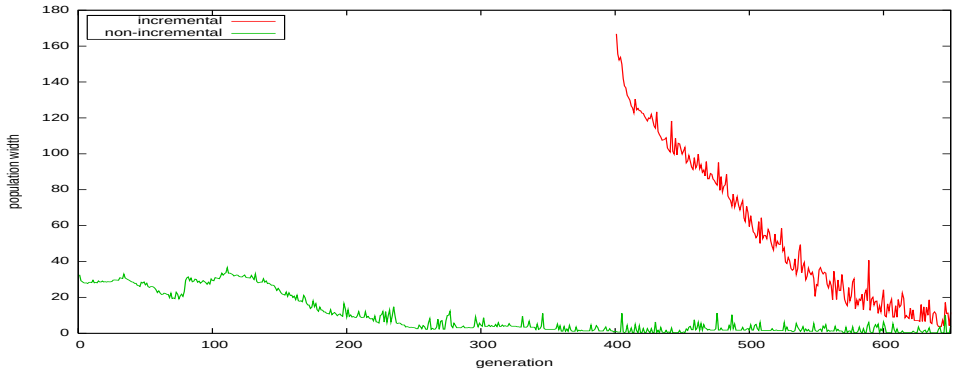
and worse than a random population.

This only stresses how difficult it is to create a fitness measure for complex behaviour. The measure works well for evolving turning if the creatures are close to walking, but it does not work well where several behaviours are to be evolved at the same time. It simply does not apply enough selection pressure to such a population, and it is difficult to imagine a measure that would do so.

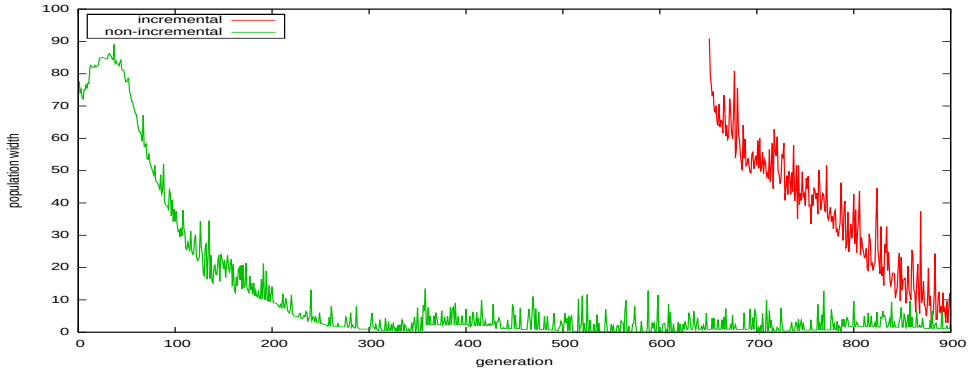
Figure 3.25: Comparing the width of non-incremental evolution to incremental ones.



(a) Evolution of locomotion. 15-runs averages. The non-incremental approach has a head-start of 150 generations to make up for pacemaker evolution.



(b) Evolution of olfactory turning. 10-runs averages. The non-incremental approach has a head-start of 400 generations to make up for pacemaker (150) and locomotion controller (250) evolution.



(c) Evolution of tactile turning. 10-runs averages. The non-incremental approach has a head-start of 650 generations to make up for pacemaker (150), locomotion controller (250) and olfactory turning (250) evolution.

3.3 Discussion

3.3.1 On focus

The techniques suggested in this thesis are all about focus. GAs are often associated with needle in a haystack problems, and following this analogy, a focusing of evolutionary search can be compared to something like "look through the bottom of the stack". Usually a good suggestion, but not always so (it would not be smart when searching for a light needle made out of bone, for example).

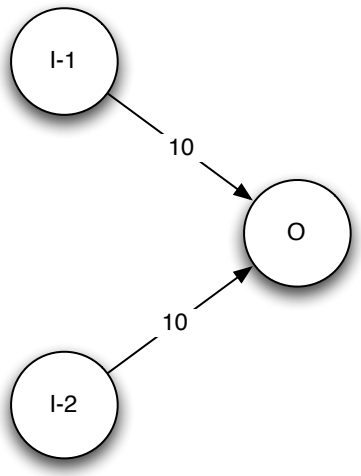
Anyone who has ever worked on machine learning, will know this problem as the problem of choosing proper bias: If you search for something in one direction, you are not searching in all the other directions. At the same time, if you do not add any bias, you will not be able to find anything! The focus, and therefore the search bias, is very strong in this thesis. In most of the experiments, very large portions of the genome was placed into a very narrow portion of the search space by copying results from earlier incremental steps. Center-crossing networks, and also networks with bifurcative neurons make up a *very* narrow portion of this space again. And, probably "worst" of all, all the networks were hand-coded into what was believed to be good approximations, loosely based on biological observations and what is believed to be common sense. It should be apparent from this, that there are an *infinite* number of solutions that have not been explored.

In this chapter I will therefore endeavour to evaluate each of the search techniques, and elaborate on whether or not the applied bias is a can be considered generally applicable. That is: Will this kind of focusing apply to other problems as well?

3.3.2 When not to use center-crossing networks

There is a possibility that the use of center-crossing networks in some circumstances can (at least theoretically) leads to slower evolution. This may happen in a CTRNN where one or more neurons are supposed to be very resilient to change. One example of such a network could be a logical AND-network as shown in figure 3.26 where the output neuron is supposed to fire

Figure 3.26: An example network for representing logical AND. Initially adjusting the bias of the output neurons according to the center-crossing equation will potentially mislead (slow down) an evolutionary search.



only when both two input neurons fire strongly at the same time, and not to fire when only one or none of the input neurons are firing.

If the bias of the output neuron in figure 3.26 is adjusted according to the center-crossing equation (equation 3.5), the bias would be:

$$\begin{aligned} \theta &= \frac{-\sum_{j=1}^{N} w_{ij}}{2} \\ &= \frac{-(10+10)}{2} \\ &= -10 \end{aligned}$$

Given a situation where neuron $I-1$ is firing strongly (1.0), and neuron $I-2$ is firing weakly (0.1) then the output neuron would indicate that the AND-condition is fulfilled (firing > 0.5), which is most probably (depending on the interpretation of the resulting firing frequency of course) an incorrect result. A more proper bias should be close to -20 (requiring that both input neurons fire strongly), which is the double of the bias given by the center-crossing equation.

Initially seeding a population with knowingly incorrect solutions is intu-

itively not a good idea. It is theoretically possible for a GA to overcome this situation, but it is hard to see how evolution could benefit from it.

3.3.3 Seeding with bifurcative neurons increases complexity

For a neuron to be bifurcative, it is necessary for it to be able to self stimulate (see section 3.1.3). This self stimulation is done using a self connection, and a neuron is bifurcative when this connection has a weight > 4 [Beer, 1995].

For all evolutions with bifurcative neurons in this thesis, the effect of enforcing bifurcative neurons was compared to other evolutions using a very naive approach: Either evolving networks where self-weights are > 4 or evolving networks with random self-weights.

This approach is naive, because there is sparse proof provided that self-weights are actually necessary at all. Removing the weights completely would shorten the genome by at least one gene (if the gene is shared by all self-weights) or maximally by the number of self-weights (if all self-weight have individual genes). Shorter genome are desirable, because it allows recombination to explore a larger number of solutions faster. There is therefore one unanswered question left after the experiments in this thesis: Would perhaps evolution without self-weights be faster than evolution with self-weights that allow bifurcation?

Time did not allow for an experimental study of this, but a short elaboration on the topology of the evolved pacemaker should still provide some insight. It is difficult to see that a pacemaker is actually possible to construct without self weights (see section 3.1.3), so the alternative would, at least for the pacemaker, be to add more neurons. Lets consider the simplest possible solution: Adding one neuron. For one neuron to have any effect on the circuit, it would need at least one incoming synaptic connection, and one output connection. As a result, at least four (2 connections + 1 bias + 1 time-constant) would be required (unless these uses shared genes, but that goes for self-connections too).

It may therefore seem that, at least for the evolution done here, the addition of self-weights > 4 is a relatively cheap way of extending the dynamical properties of the CTRNN. It may very well be that this is true for many other network evolutions as well.

Chapter 4

Conclusions

As stated in the introduction, there are many benefits of using Artificial Neural Networks (ANNs) for cognitive tasks like robot control. Some of the mentioned advantages were robustness, ability to generalize and adapt to real-world (unpredictable) environments, modularity and thereby the possibility of distributing processing and maintaining robustness just like real animals do.

The major question that arises from claims like these is of course "how are such networks created?". Several researchers have tried to make ANNs for simulating cognitive behaviours. Among the mentioned successful approaches was the work of Cruse et al. [1998] that focused on creating realistic insect walks. Also, the work of Beer [1990] made up a detailed study of a biologically realistic neuron setup for an artificial insect. Both of these approaches included manually designed ANNs and Cruse et al. [1998] additionally trained the network using a supervised training technique (backpropagation).

Much contemporary work on creating ANNs (and other controllers) for minimally cognitive behaviour has turned toward automated design by means of Genetic Algorithms (GAs) - or simulated Darwinian evolution. GAs are powerful, and two important examples of work in this area was described. One was the work of Gomez and Miikkulainen [1997] that evolved the complete topology of a network for complex general behaviour (prey capture). Another was the work of Mathayomchan and Beer [2002] that evolved net-

work weights in a network for rhythmic behaviour (a key feature of many biological neural systems). Though the two examples had quite different approaches to evolving cognitive behaviours, they both addressed two major shortcomings of GAs that were emphasized in the introduction: *GAs are computationally expensive, and they tend to converge toward local optima.*

In this thesis I have tried to pursue this research on GAs further. This has been done by applying a conglomeration of contemporary strategies to a sufficiently complex (but still manageable) problem: Insect navigation.

A nervous system for tactile-olfactory navigation was sketched, and synaptic weights and other neural parameters for it were then evolved. Throughout the process I followed a two-layered approach. On a macro level I used incremental evolution much like Gomez and Miikkulainen [1997]. This allowed me to successfully evolve "easy" tasks and evaluate these before continuing with more complex tasks. On a micro level I used the mathematical theory of Beer [1995] (bifurcative neurons). I also used and extended the technique of Mathayomchan and Beer [2002] (center-crossing networks) to speed up and improve each evolutionary subtask.

4.1 Center-crossing CTRNNs

Seeding initial populations with center-crossing networks was suggested by Mathayomchan and Beer [2002]. A center-crossing network is a Continuous-Time Recurrent Neural Network (CTRNN) where all neurons have activation functions that are centered around the net input each neuron receives. The network is therefore made up from maximally sensitive neurons. For an interconnected network (all neurons are connected to each other) this implies that there exist an equilibrium point where all neurons fire one half of their maximum firing frequency. Mathayomchan and Beer [2002] showed through experiment that an evolutionary search for highly-fit oscillatory CTRNNs was significantly improved by seeding the initial population with center-crossing networks. They suggested that this technique may be beneficial for any search for CTRNN weights, because the range of dynamics in such a network are wider than in a random network.

In this thesis, I have investigated the use of center-crossing networks for the evolution of several new network topologies. Extending the work of Math-

ayomchan and Beer [2002] these networks now also included non-oscillatory networks and networks that were not fully interconnected.

Through experimentation and comparative analysis (against random networks) I first verified that evolving a pacemaker greatly benefited from being seeded with center-crossing networks. This was done through the evolution of a simple two-neuron pacemaker circuit. The results showed similar improvement as found in the (only slightly different) approach of Mathayomchan and Beer [2002].

I further showed that evolution of networks that were not fully interconnected still benefited from parts of the theory presented in Mathayomchan and Beer [2002]. It is not always desirable to have completely interconnected networks. In this thesis, for example, a network topology based on Beer [1990] was used for controlling locomotion. The network in Beer [1990] was not fully interconnected, and it seemed exaggerated to add more connections just to fill the center-crossing requirement. I therefore extended the earlier work of Mathayomchan and Beer [2002] and set out to test if some of the underlying causes of the success of seeding with center-crossing networks might still be used in a network that was *not* completely interconnected.

Networks that are not fully interconnected are of course not capable of maintaining any *equilibrium* like fully interconnected networks do. The network can therefore not enjoy the complete range of extended dynamics that Mathayomchan and Beer [2002] originally argue for. Despite this, it is still possible to adjust the neurons biases of any network *as if* we had a center-crossing network. As a result, all neurons are still **maximally sensitive**. By evolving weights for a locomotion controller based on Beer [1990], I showed that seeding the network with networks created this way had a significant effect on evolution. The evolved controller performed much better, and the results when running several searches more frequently ended up with good solutions to the problem.

The technique of seeding initial populations with center-crossing networks was also applied to the evolution of a interconnected, non-oscillatory network. Similarly as for the other two searches, this evolution also greatly benefited from the technique. The resulting network performed significantly better with regard to the fitness measure. Statistical evidence also showed that the search also more frequently ended up with good solutions, compared to random searches.

As a general rule, the seeded searches converged steadily toward a good solution and avoided getting trapped in local optimums. The equal random searches tended to improve in stages where the search "jumped" from optima to optima and eventually stagnated in some local optimum, resulting in highly different (sub-optimal) results from evolution to evolution. The results here therefore support the suggestion of Mathayomchan and Beer [2002] that *seeding evolutionary searches with center-crossing networks may always be beneficial*. It also seemed that setting an initial bias just to make neurons maximally sensitive is beneficial, even for networks that are not fully interconnected.

4.2 Bifurcative neurons

When single neurons in a CTRNN have a self-connection with a weight factor > 4, the neuron is capable of maintaining several steady-states [Beer, 1995]. I therefore suggested to seed initial populations with networks made up from neurons with self-weights > 4 to further improve the evolution of at least oscillatory networks. The idea was that an oscillatory network would benefit from having individual neurons that could flip between two states. This should also be a cheap way way of extending the dynamic behaviour of the network compared to adding additional neurons.

To measure the effect of seeding initial populations with bifurcative neurons, a set of experiments were run. Using the same task (fitness measure), evolution with and without bifurcative neurons were run. Additionally, some evolutions were seeded with center-crossing networks. From the experiments with the oscillatory task (pacemaker evolution) it was clear that *the populations seeded with bifurcative neurons reached better solutions faster than searches without bifurcative neurons*. The difference was however most significant when the networks were center-crossing at the same time as they contained bifurcative neurons.

I further showed that the evolution of a navigational turning controller based on olfactory sensing highly benefited from the seeding with bifurcative neurons. For the network evolved here, a solution comprising a temporal memory was evolved. It was shown through experiment that even this very simple circuit was capable of maintaining a "commitment" toward different goals (directions) by adopting different network states. The evolution of

such states was facilitated by some of the neurons being capable of taking on more than one steady-state.

An overall observation from all evolutions was that none of them was made slower or worse performing by setting self-weights to values larger than 4 (fulfilling the theoretical bifurcation requirement [Beer, 1995]). To the contrary, most evolutions were made faster and/or better. I therefore suggested that actually adding self-connections to neurons that have no self-connections may be advantageous for many evolutions, even if it increases complexity: Addition of self-connections is a simple way of leveraging network dynamics with only minimal increase of genome complexity. I think that further studies of this effect could be very interesting.

4.3 Incremental evolution

Throughout this thesis I have demonstrated that it is possible to evolve at least some complex behaviours incrementally. In contrast, I showed that for the example task an equal, non-incremental, approach was slower and produced solutions that performed worse. Incremental evolution also facilitated acquaintance and thorough evaluation of parts of the complex cognitive task of navigation, since the task was evolved in a step-wise fashion.

The task of evolving navigation was partitioned into three subtasks (walking, turning and obstacle avoidance). The first of these tasks was also partitioned further into two even smaller tasks (a pacemaker and locomotion). By comparing incremental evolution of these behaviours with non-incremental counterparts, it was shown that it was possible to evolve higher level behaviours (turning and obstacle avoidance) when a non-incremental approach was unsuccessful. I think that this way of partitioning the problem made it easier to present it in an comprehensible manner.

For the three experiments carried out here, I further showed that the more complex the behaviour was (e.g. higher up in the subsumption hierarchy, and longer genome), the more the incremental approach outperformed the non-incremental approach.

Chapter 5

Appendix

5.1 Source code excerpts

For the complete source code, please see the attached Steve (*.tz) and Common Lisp (CL) (*.lisp) source code files. A few code excerpts are included here. They are of potential interest for anyone working with CTRNNs. For people unfamiliar with CTRNNs (but familiar with CL) they may provide insight into the workings of CTRNNs.

5.1.1 CTRNN bifurcation points

```
(defun bifurcation-points (self-weight bias)
 "Return the two bifurcation points of a neuron."
  (mapcar
   #'(lambda (plus/minus)
       (realpart
        (- (* (funcall plus/minus 2)
              (log (/ (+ (sqrt self-weight) (sqrt (- self-weight 4))) 2)))
           (/ (funcall plus/minus
                       self-weight
                       (sqrt (* self-weight (- self-weight 4))))
              2)
           bias)))
   (list #'+ #'-)))
```

5.1.2 Euler "leaky-integrator"

```
(defmethod update-membrane-potential ((n neuron))
  "Find and set membr. potential of n using Euler's integration method."
  (incf ; Increase and set potential by estimating change for *time-step*
   (neuron-membrane-potential n)
```

```
(* *time-step* ; Euler time step
   (/ (+ (loop ; Add sum of synaptic currents
           for s across (neuron-dendrites n)
           sum (* (synapse-strength s)
                  (neuron-snapshot-firing-frequency (synapse-from-neuron s))))
         (loop ; Add sum of intrinsic currents
           for i across (neuron-intrinsic-currents n)
           sum i)
         (neuron-external-current n) ; Add external current
         (- ; - Subtract the leak current
           (* (neuron-membrane-potential n) (neuron-membrane-conductance n))))
      (neuron-membrane-capacitance n))))) ; Div. by capacitance = pot. change
```

5.1.3 Euler CTRNN integrator

```
(defmethod update-membrane-potential! ((neuron neuron))
  "Calculate and modify neuron current membrane potential."
  (unless (> (neuron-membrane-potential neuron) ; Restrain from being enormous.
             (neuron-maximum-membrane-potential neuron)) ; may cause overflow
    (incf
     (neuron-membrane-potential neuron)
     (* *timestep* ; The integration timestep
        (* (/ 1 (neuron-time-constant neuron)) ; 1/tau
           (+ (- (neuron-membrane-potential neuron)) ; Neg. of the curr. pot.
              (neuron-external-current neuron) ; Ext. curr.
              (loop for synapse in (neuron-dendrites neuron) ; Synaptic curr.
                    sum (* (synapse-strength synapse)
                           (neuron-snapshot-firing-frequency
                            (synapse-from-neuron synapse))))))))))


(defmethod update-membrane-potential! :around ((neuron motor-neuron))
  "Call the associated motor function. Return energy consumed."
  (call-next-method) ; Update membrane potential first
  (funcall (motor-neuron-motor-function neuron) (firing-frequency neuron)))


(defmethod update-membrane-potential! :before ((neuron sensor-neuron))
  "Read the connected sensor before updating sensor neuron mem. pot."
  (funcall (sensor-neuron-sensor-function neuron)))
```

5.1.4 CTRNN firing frequency

```
(defmethod firing-frequency ((neuron neuron))
  "Return firing frequency based on membrane potential."
  (/ 1 (+ 1 (exp (- (+ (neuron-membrane-potential neuron)
                       (neuron-bias neuron)))))))
```

Bibliography

Randall D. Beer. *Intelligence as adaptive behavior: an experiment in computational neuroethology.* Academic Press Professional, Inc., San Diego, CA, USA, 1990. ISBN 0-12-084730-2.

Randall D. Beer. On the dynamics of small continous-time recurrent neural networks. *Adaptive Behavior*, 3:469–509, 1995.

Randall D. Beer. Toward the evolution of dynamical neural networks for minimally cognitive behavior. 1996. URL `http://vorlon.case.edu/ beer/Papers/SAB96.pdf`.

Randall D. Beer and John C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1:91–122, 1992.

Randall D. Beer, Roger D. Quinn, Hillel J. Chiel, and Roy E. Ritzmann. Biologically inspired approaches to robotics. what can we learn from insects? *Communications of the ACM*, 40:31–38, 1997.

Dr. Johann Borenstein, Commander H. R. Everett, and Dr. Liqiang Fleng. *Navigating Mobile Robots: Sensors and Techniques.* A. K. Peters, Ltd., Wellesley, MA, 1996.

Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2 No. 1:14–23, 1986.

Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1987.

Ulrich Bässler and Ansgar Büschges. Pattern generation for stick insect walking movements - multisensory control of a locomotor program. *Elsevier, Brain research reviews*, 27:65–88, 1998.

Hillel J. Chiel, Randall D. Beer, and John C. Gallagher. Evolution and analysis of model cpgs for walking i. dynamical modules. *Journal of Computational Neuroscience*, 7:99–118, 1999.

Adrian F. Clark. Applications of machine vision. 2005. URL `http://www.bmva.ac.uk/apps/index.html`.

Holk Cruse, Thomas Kindermann, Michael Schumm, Jefferey Dean, and Josef Schmitz. Walknet - a biologically inspired network to control six-legged walking. *Neural Networks*, 11:1435–1447, 1998.

Volker Dürr, Yvonne König, and Rolf Kittmann. The antennal motor system of the stick insect carausius morosus: anatomy and antennal movement pattern during walking. *Journal of Computational Physology*, 187:131–144, 2001.

Orjan Ekeberg, Marcus Blümel, and Ansgar Büschges. Dynamic simulation of insect walking. *Arthropod Structure and Development*, 33:287–300, 2004.

Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

John Klein. Breve: a 3d environment for the simulation of decentralized systems and artificial life. 2006.

Jerome Kodjabachian and Jean-Arcady Meyer. Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, 9:796–812, 1998.

Jérôme Kodjabachian, Christophe Corne, and Jean-Arcady Meyer. Evolution of a robust obstacle-avoidance behavior in khepera: A comparison of incremental and direct strategies. 1998.

Andre F. Krause and Volker Dürr. Tactile efficiency of insect antennae with two hinge joints. *Biological Cybernetics*, 91:168–181, 2004.

George F. Luger. *Artificial Intelligence 4th edition.* Pearson Eucation Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England, 2002. ISBN 0-201-64866-0.

Boonyanit Mathayomchan and Randall D. Beer. Center-crossing recurrent neural networks for the evolution of rythmic behavior. *Neural Computation*, 14:2043–2051, 2002.

Peter McLeod, Kim Plunkett, and Edmund T. Rolls. *Introduction to Connectionist Modelling of Cognitive Processes*. Oxford University Press, Great Clarendron Street, Oxford OX2 6DP, 1998. ISBN 0-19-852426-9.

David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

David E. Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5, 1998.

Pavel Petrovic. Comparing finite state automata representation with gp-trees. *IDI Technical Report*, 4, 2006. ISSN 1503-416X.