Never Too Old To Learn

On-line Evolution of Controllers
in Swarm- and Modular Robotics

Evert Haasdijk

Department of Computer Sciences
Faculty of Sciences, Vrije Universiteit

2012

Thesis Reading Committee:

prof.dr. L. Bull	Department of Computer Science, The University of the West of England, UK
prof.dr. D. Floreano	Laboratory of Intelligent Systems, Ecole Polytechnique Fédérale de Lausanne, Switzerland
dr. A. Visser	Informatics Institute, Faculty of Science, University of Amsterdam, The Netherlands
dr.ir. D. Thierens	Department of Information and Computing Sciences, Utrecht University, The Netherlands
prof.dr. J. Treur	Agent Systems Research Group, Faculty of Sciences, VU University Amsterdam, The Netherlands

SIKS Dissertation Series No. 2012-35
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

VRIJE UNIVERSITEIT

Never Too Old To Learn

On-line Evolution of Controllers in Swarm- and Modular Robotics

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op donderdag 1 november 2012 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Evert Willem Haasdijk

geboren te Voorburg

promotor: prof. dr. A.E. Eiben

For Willem and Keetje

Listen to the MUSTN'TS, child.
Listen to the DON'TS.
Listen to the SHOULDN'TS,
The IMPOSSIBLES, the WON'TS.
Listen to the NEVER HAVES,
Then listen close to me—
Anything can happen, child,
ANYTHING can be.

Shel Silverstein

Table of Contents

Acknowledgements

First of all I am, of course, beholden to my parents: they encouraged intellectual curiosity, were always ready with advice and encouragement and generally gave me a solid, trusted basis that knew I could always rely on. Thanks mum, thanks dad. I am very sad that mum is no longer with us to see this.

When I was young, probably about thirteen or fourteen years old, Dutch television aired the incomparable Carl Sagan's *Cosmos* series. I was allowed to stay up late one night each week to watch Carl Sagan take us on his cosmic voyage into science, into what we know, what we guess at and how we find things out. These were magical evenings; I would sit there with my dad watching carl Sagan's infectiously enthusiastic, yet at the same time serene presentation of this phantasmagoric illustration of man's curiosity and inventiveness. It may go too far to say that Carl Sagan is responsible for my interest in science, but he certainly helped kindle it.

At the end of my studies at the UvA, I was lucky enough to find a position as an intern at Cap Gemini Innovation where I worked closely with Rob Walker and David Barrow. Together with Rob, David and Colin Baker I later went on to run KiQ. During our years together we not only had loads of fun, but working with Rob and David has defined the first ten years of my career and taught me many things that have stood me in good stead over the years.

Over the last years, I have been fortunate to work with some very talented and intimidatingly smart people at the VU and our partners in the NEW TIES and SYMBRION projects. It has been a pleasure and an education to work with you all and I hope to see many of you in further collaborations. In particular my roommate Selmar Smit with whom I had so many discussions, coffees and beers: thanks. I'm glad we managed to co-author a chapter of both our theses.

Thank you, my other co-authors Arif Atta-ul-Qayyum, Nicolas Bredèche, Gusz Eiben, Pablo García-Sánchez, Robert Griffioen, Robert-Jan Huijsman, Giorgos Karafotias, Serge Kernbach, Abraham Prieto, Andrei Rusu, Martijn Schut, Paul Vogt, Berend Weel, Willem van Willigen and Alan Winfield.

And of course thank you, my friends and colleagues in the CI group with whom I haven't (yet) co-authored anything: Joeri Bekker, Nivea Ferreira, Vincent van der Goes, Eelco den Heijer, Mark Hoogendoorn, Rob Konijn, Zoli Szávik and Christian Tzolov.

I owe a great debt of gratitude to Gusz Eiben, not just for his support and guidance during the research that led to this thesis, but more importantly because he took a chance on me, first hiring me for what should have been a postdoc position, then for his efforts to secure my (provisional) assistant professorship when Martijn Schut left our group, even though I hadn't yet completed my thesis. I am looking forward to many more years of joint research, project meetings that run on into the night and feet-on-the-table discussions on a host of topics long after the meeting has ended, with the beer flowing freely.

Thank you, members of the reading committee. For your time, your attention and your feedback: Larry Bull, Dario Floreano, Arnoud Visser, Dirk Thierens and Jan Treur.

Thank you, my paranymphs: darling sister Suzan and dear friend Luuk, for being there. Always.

Thank you, Tom, for suggesting that I consider a position as PhD student when I was looking for a more challenging job than freelance programmer.

Thank you, reader, for taking the time to read this. I'm sure I've forgotten someone in the frantic effort of getting this thesis to the printer in time. I'm sorry. Remind me sometime and I'll buy you a beer to make up for it.

Much, if not all, of the work presented in this thesis was made possible by the European Union FET funding the NEW TIES project under grant agreement 003752 and FET's Proactive Initiative: Pervasive Adaptation funding the SYMBRION project under grant agreement 216342.

Last, but certainly not least: thank you Dorien, Willem and Keetje. For being there, for bearing with me and for making me a better person than I could possibly be by myself. I love you.

It Beats...as it Sweeps...as it Cleans

Gerald Page-Wood for the Hoover Company

1 Summary

The work described in this thesis was inspired by a vision of truly autonomous robots that can adapt their behaviour, possibly even their shape, to suit varying tasks and circumstances. Autonomy occurs at two levels: not only do the robots perform their tasks without external control, they also adapt their behaviour without referral to external oversight and so learn autonomously.

Such versatility is well beyond most autonomous robots that have been designed for particular tasks ("weld the body to the chassis") in well-defined environments. To be able to handle unexpected circumstances and tasks, robots must acquire the ability to learn appropriate behaviours and morphologies as they encounter these circumstances and are given these tasks. This ability to learn, to adapt, autonomously is the focus of our research.

Robots can be programmed to adapt individually, by themselves, without referral to any external overseers, but when there are multiple robots that try to tackle the same task it makes sense to seek strength in numbers by learning collectively. Robots can then combine their knowledge through evolution (evolutionary adaptation) or they can exchange knowledge through social interaction (social learning) to boost their individual learning processes. Thus, there are three possible ways in

which a collective of robots can implement adaptivity: individually, socially and evolutionarily.

The research described in this thesis was undertaken as part of the SYMBRION project, which envisages groups of dozens of robots that can link together to form and manipulate 'organisms', but that can also act separately, in 'swarm mode'. The robots must be able to learn, jointly as well as individually, to perform tasks in diverse environments: they must exhibit the adaptivity lacking in regular robotics.

We have looked closely at learning behaviour at the individual level. One of our fundamental choices was to rely on evolution as the main enabler of adaptivity. Therefore, we implement even individual learning through an evolutionary algorithm, encapsulating a population of evolving controllers in each individual robot. This may be somewhat confusing, since this mechanism does not implement evolution at the level of the robot collective. This becomes clear when we consider that there is no exchange of information among the robots, and robots learn in isolation just as they learn in the presence of their peers. Adding such exchange of information amounts to introducing 'proper' evolution, leading to a distributed evolutionary mechanism. Finally, we can combine these two mechanisms to obtain an implementation of social learning.

We have performed comparative analyses with algorithms that implement evolutionary adaptation by distributing evolution over the robots in the collective and hybrid algorithms that combine encapsulated and distributed evolution into a social learning approach.

We set out to answer three main research questions:

- Can we devise evolutionary algorithms that allow robots to learn to perform simple tasks autonomously?

- Which approach –encapsulated, distributed or hybrid– offers the best results?

- How does the performance of our algorithms depend on parameter settings?

We have developed and tested an encapsulated evolutionary algorithm called $(\mu + 1)$ ON-LINE that provides for individual learning in robots. This algorithm proved capable of adapting robot behaviour to perform a number of simple tasks like obstacle avoidance and patrolling, allowing the robots to learn to perform these tasks without the need for any external oversight.

We have also developed distributed and hybrid alternatives for $(\mu + 1)$ ON-LINE, and we have seen that it is in general beneficial to learn collectively, but that care should be taken when the task implies competition among the robots.

Extensive tests of the algorithms have shown –as expected– that their performance depends profoundly on the parameter settings. However, we found that there is no 'silver bullet' setting that works equally well across our experiments. This indicates a need for further research into parameter control schemes that will allow the algorithms to adjust their parameters according to the circumstances and the task.

Es ist nichts trauriger anzusehen als das unvermittelte Streben ins Unbedingte in dieser durchaus bedingten Welt

Johann Wolfgang von Goethe

2

Background and Contributions

Imagine a group of small, relatively simple, autonomous robots that collectively have to perform various complex tasks. To achieve their goals, the robots can move about individually, but more importantly, they can physically attach to each other to form and manipulate multi-robot organisms for tasks that an unconnected group of individual robots cannot cope with. Think, for instance, of scaling a wall or holding a relatively large object in place. One of the advantages of a swarm of simple robots is the increased robustness compared to complex monolithic systems: if a single robot fails, the swarm can carry on regardless because of redundant modules that can replace the failing robot. Another advantage is that the robots can reconfigure the organism to suit particular tasks and circumstances, something that large, complex and monolithic robots would find impossible.

Because of the inherent versatility of such robots, they seem ideal for autonomous deployment in unknown and dynamic environments: environments that we cannot fully describe at deployment time and where the circumstances under which the robots must operate as well as the tasks that they must tackle change over time. Autonomous deployment means that the robots are isolated from direct human control. To ensure robustness of the robot collective, control

must be distributed over the individual modules: if there were some central unit responsible for the whole, that would become the single point of failure, negating the possibility of robots autonomously replacing any failing member of the collective.

The level of flexibility that we envisage is very ambitious and calls for, among other things, many levels of adaptation. It requires adaptation of morphology: the robotic modules must be able to reconfigure the structure they form collectively to meet some challenge. For instance, scaling a wall requires a different, larger shape than negotiating a warren of narrow pathways. They must also be capable of learning new shapes to fit circumstances that we cannot foresee when we develop the system. All this also requires adaptivity of control: the controllers of the robot modules must be able to learn to perform well in circumstances that we do not know in advance, in configurations that will develop over time, to perform tasks that we cannot yet specify.

This vision has inspired the start, in 2008, of the EU-funded Symbiotic Evolutionary Robot Organisms (SYMBRION) project and its sister project Robotic Evolutionary Self-Programming and Self-Assembling Organisms (REPLICATOR). Particularly in the SYMBRION project, a substantial proportion of the researchers concentrate on researching and developing techniques that allow the robots to learn – to adapt their controllers and the shape of the organisms they form to various tasks and circumstances – in this context. As SYMBRION's full name implies, evolution is an essential aspect of this research: the ability to learn is provided through the use of evolutionary algorithms.

We can distinguish between the design and operational stages of a robot's life. Prior to deployment, at design time, we can then see evolutionary algorithms as tools for the design of robots and their controllers, but after deployment, during a robot's operational phase, evolutionary algorithms become tools that provide adaptivity. To differentiate between the use of evolution in these two phases, we call it *off-line* and *on-line*, respectively.

We claim that the ability to adapt autonomously is a condicio sine qua non for the successful implementation of collective robotic systems so that they may cope with a number of challenges:

Unforeseen environment The environment where the robots operate may not be fully known during the design process. Therefore, the robot controllers at the

time of deployment are only approximate solutions that need to be adapted to the environment as it is found at operational time.

Changing environment The environment may change to such an extent that the initial skill set of the robots is no longer adequate. Hence, controllers must adapt to changing situations.

Reality gap Even if the environment were known beforehand and constant during operational time, it is very likely that the design process uses approximations and simulations of real operational conditions. Hence, the robot controllers will have to be fine-tuned after deployment.

In their overview of tasks considered in evolutionary robotics, Nelson et al. make an eloquent case for the necessity of adaptation 2009:

> *"Advanced autonomous robots may someday be required to negotiate environments and situations that their designers had not anticipated. The future designers of these robots may not have adequate expertise to provide appropriate control algorithms in the case that an unforeseen situation is encountered in a remote environment in which a robot cannot be accessed. It is not always practical or even possible to define every aspect of an autonomous robot's environment, or to give a tractable dynamical systems-level description of the task the robot is to perform. The robot must have the ability to learn control without human supervision."*

In the vision professed above, such adaptation is necessarily on-line and without human intervention: a robot's controller changes on-the-fly, as it goes about its regular tasks.

To set the scene, assume that the individual robots in our collective have to learn to avoid obstacles while moving around, a common task in evolutionary robotics. They drive around a maze-like arena filled with obstacles. While driving around, they perform on-line, on-board evolution, replacing their controller at regular intervals to test controller configurations proposed by the on-board evolutionary algorithm. They measure their performance with every controller they test (e.g., do they (nearly) bump into obstacles, does the controller cause then to stop moving or does their average speed increase); these measurements guide the creation of new controllers to test. When they are done evaluating one controller, they

replace the controller and continue their wandering through the arena, measuring the performance of this new controller. The robots may communicate to exchange controllers so that they can pool their knowledge or they can adapt purely individually, without referring to others. As you can see, the adaptive process never ends: in fact, the robot's behaviour is an everlasting sequence of controller appraisals. The evolutionary process must make sure that the proposed controllers actually lead to the desired behaviour and that the robots perform their taks – obstacle avoidance, in this case. In our research, we have performed these kinds of experiments with simulated robots: the real robots we would use are being developed in the REPLICATOR project mentioned earlier are only now becoming available; we expect to continue this work with real robots over the coming years.

What distinguishes the research in this thesis from most other research into evolutionary robotics is the on-line nature of adaptation: this is a major departure from 'traditional' evolutionary robotics, where the controllers are developed off-line, and remain fixed once the robots are deployed actually to perform their tasks. On-line evolutionary robotics is in many ways radically different from regular evolutionary algorithms because it has to address a number of uncommon or even game-changing impositions. We consider these below and after that, table 2.1 provides a summary of these issues.

On-line, on-board evolution has to contend with limited processing power. Although the microprocessors one finds in small robots rapidly become better and better, there is and will always remain a lag between what is possible inside a robot and the processors available for the desktop or even supercomputers.

Populations in on-line evolution are small, and this is due to two reasons. Firstly, there are the limitations of robot memory size: often measured in kilobytes for small robots where computers measure their memory in gigabytes. The advent of cheap replaceable memory, e.g. on SD-cards makes this less of a worry than it used to be (although the recently developed kilobot[i] still sports only 32 kilobytes of programmable memory). More fundamentally, a large population would necessitate a long episode of testing the initial population, entailing very poor performance for a fairly long period with all its attendant risks.

In typical applications of evolutionary algorithms, the be-all and end-all is a champion individual that is as good as possible: the best performing individual at termination has to be as close to optimal as we can get. The performance of

[i] `http://www.k-team.com/mobile-robotics-products/kilobot`

the remainder of the population is, in the end, of no consequence as they will be discarded when the champion is deployed; their only reason for existence is to guide evolution's search process to the pinnacle that is the best individual in the population. Things are very different for on-line evolution: controllers evolve as the robots go about their tasks and so the robots' actual performance is determined by the quality of *all* controllers that they evaluate, not that of any single individual controller alone. When a robot evaluates poor controllers, that robot's actual performance will be inadequate, no matter how good the best known individuals as archived in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution (even if it is not the best) and search prudently: it must display an acceptable level of performance throughout the continuing search. To put it bluntly, the evolutionary algorithm can't afford too many bad guesses – they lead to unacceptable task performance and may even cause damage to the robot.

These real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably and there is no oversight, human or otherwise, to control these conditions. Different controllers will be evaluated under different circumstances: any controller's evaluation will start wherever and in whatever state the previous evaluation left the robot. The very dissimilar evaluation conditions caused by one –possibly very poor– individual setting the scene for the evaluation of another individual result in very noisy fitness assessments. As Nordin and Banzhaf (1997) note:

> *"Each individual is thus tested against a different real-time situation leading to a unique fitness case. This results in 'unfair' comparison where individuals have to navigate in situations with very different possible outcomes."*

A traditional evolutionary process is centrally orchestrated because the selection of parents and survivors is performed in a single loop where the fitness of every individual is known. Such a scheme would violate our premise of distributed control: even if the robots' regular controllers operate in a distributed manner, if their adaptation were centralised, that central authority would become the single point of failure. Therefore, the evolutionary process must run autonomously across the robot collective and there is no central authority that decides which robot controllers reproduce and which ones are replaced and there is no omniscient presence who knows (let alone determines) the fitness values of all individuals. Consequently, the robots gauge their own (and each other's) fitness themselves

and it is they themselves who autonomously decide (based on their fitness information) when to mate and with whom. If the algorithm is distributed across multiple robots, this introduces the notion of a neighbourhood from which robots can select partners. This neighbourhood can be physical and consist of other robots that happen to be within communication range or it can be in terms of a social network across the population using long-distance communication. The robot's fitness must be evaluated *in vivo*: the quality of any given controller is determined by actually using that controller for some time.

It is well known that the performance of evolutionary algorithms depends in large part on their parameter settings, even to the extent that optimal parameter values for an evolutionary algorithm often differ from problem to problem (Eiben et al., 1999b). The common practice of parameter tuning (seeking good parameter values through trial runs) before deployment is not an option for on-line evolution, because the robots have to adapt to operational circumstances – without human intervention – that are unknown beforehand. Such unknown, possibly dynamic circumstances imply that the evolutionary mechanism through which the controllers adapt must be somehow (re)tuned to be equal to these (new) circumstances. Hands-free adaptation therefore requires evolutionary algorithms that are either capable of calibrating themselves on the fly (known as *parameter control*) or that use robust parameter settings that work well under (almost) all circumstances.

To reduce the number of costly evaluations, traditional evolutionary computation can turn to surrogate models: a candidate solution's quality is estimated by a surrogate model that approximates the expensive calculations or trials needed for definite assessment (Ratle, 1998). This estimate may then be used, for instance, to determine which candidate solutions are then evaluated properly (and expensively). Section 6.5 describes such a procedure as used in the Bonesa (Smit and Eiben, 2011) parameter tuning approach. For off-line evolutionary robotics, simulators can provide surrogates, but for on-line evolution on the kind of platform we have in mind, the computational requirements of simulators is beyond what the robot's processor can handle. If sufficient computational power were available, however, this may become a feasible option to reduce the risk of evaluating poor controllers in real time. In the case of self-reconfiguring robots where the exact morphology of the robot may not be known at run time, a self-modelling approach may be beneficial. Bongard et al. (2006) describe how robots can calibrate a simple simulator to reflect their body plan. Such a self-model would greatly boost

the applicability of on-board simulation as a surrogate model for self-reconfiguring robots.

On-line, on-board evolutionary robotics	**Off-line evolutionary robotics**	**Evolutionary computing**
Limited processing power	Virtually unlimited processing power	Virtually unlimited processing power
Limited population size	Virtually unlimited population size	Virtually unlimited population size
Requires good progress with few evaluations	Only end result matters; rate of progress is immaterial	Only end result matters; rate of progress is immaterial
Poor individual entails low task performance, may break robot	Poor individual wastes time and resources	Poor individual wastes time and resources
Requires parameter control	Requires parameter tuning or control	Requires parameter tuning or control
Implicit noise	Implicit noise	Noise possible
No control over initial conditions for an evaluation	Control over initial conditions for an evaluation	Control over initial conditions for an evaluation
No surrogate model	Surrogate model (simulation)	Surrogate model possible
Local, de-centralised selection	Global, centralised selection	Global, centralised selection

Table 2.1 – Summary of technical challenges for on-line, on-board evolutionary robotics compared to off-line evolutionary robotics and 'regular' evolutionary computing.

This thesis describes, for the most part, work undertaken at the Computational Intelligence group at the VU University Amsterdam towards the SYMBRION endeavour. The SYMBRION project can be seen as tackling two major challenges: on the one hand, that of autonomously (re-)forming multi-robotic organisms and on the other hand that of on-line, on-board adaptation through evolution. The research described in this thesis focusses on the latter challenge: the development and study of on-line, on-board evolutionary algorithms to allow robots to learn how to tackle specific tasks. However, section 7.4 in particular shows a shift in fo-

cus towards the organism aspect that reflects the progress of our research beyond the scope of this thesis. We will get back to this shift in emphasis in chapter 9. Other techniques, for instance reinforcement learning, Hebbian learning or Learning Classifier Systems (also evolutionary) could also be considered as providers of on-line adaptivity. These techniques are outside the purview of our research, however, and are therefore not treated in this thesis.

The aim of our research, then, can be phrased as developing novel evolutionary algorithms that meet the challenges outlined above, considering the following questions:

- First and foremost: can on-line evolution provide the ability to learn as required and provide consistent task performance?
- How can we tackle the peculiar demands that on-line evolution poses?
- Which parameters of the algorithms we investigate have the most influence on the quality of control?

The remainder of this thesis describes our efforts to answer these questions, bundling papers in which we investigate the pitfalls and possibilities of on-line evolutionary robotics. These papers have been published elsewhere or are under review. As a result, each chapter, and in the case of chapters 5 and 7 each section, can be read separately. Consequently, there is a certain repetitivity; the vision underlying this research as professed above, for instance, is re-stated in every chapter, just as the description of algorithms and experiments pop up in more than one place, to put it mildly.

Chapter 3 introduces a conceptual framework for positioning *Population-based Adaptive Systems* in general, not limited to robotic systems. The framework differentiates between three techniques that can provide adaptivity to collective systems: individual learning, social learning and evolution. The chapter also provides some case-studies concerning the interplay between these three approaches to adaptivity.

Chapter 4 reviews related work in on-line evolutionary robotics, introducing a classification into *encapsulated, distributed* and *hybrid* approaches to on-line evolution. It also catalogues issues that on-line evolutionary robotics has to contend with that are not commonly considered in evolutionary computing research and proposes a research agenda to bring this field forward. This chapter provides the

best introduction into the vision that underlies our work and highlights the issues that on-line evolution of robot controllers poses.

Chapter 5 contains a collection of shorter (conference) publications that describe stages of development and initial analysis of the $(\mu + 1)$ ON-LINE algorithm for on-line evolutionary robotics. It details our efforts to tackle the issues raised in chapter 4.

Chapter 6 provides an in-depth analysis of $(\mu + 1)$ ON-LINE. Apart from the analysis itself, this chapter proposes a methodology and tool – Bonesa – for scientific testing of stochastic adaptive algorithms.

Like chapter 5, chapter 7 combines a number of conference papers. These, however, concern themselves with implementations of distributed and hybrid on-line evolution, where the subject of chapter 6 takes the encapsulated approach.

Chapter 8 provides a vision, a manifesto if you will, of where artificial evolution – not just of robots – could go in the coming decades.

Finally, chapter 9 draws overall conclusions, returning to the research questions stated above. It also suggests future research to follow up on the findings in this thesis.

This thesis is based on the following papers:

Evert Haasdijk, A.E. Eiben, Alan F.T. Winfield (2011). Individual, Social and Evolutionary Adaptation in Collective Systems. Chapter 13 in S. Kernbach*Handbook of Collective Robotics - Fundamentals and Challenges*, Pan Stanford Publishing, Singapore.

Evert Haasdijk, A.E. Eiben. Look Ma, No Hands! – An Overview of On-Line, On-Board Evolutionary Robotics. Submitted to *Swarm and Evolutionary Computation*, Elsevier.

Nicolas Bredeche, Evert Haasdijk and A.E. Eiben (2009). On-line, On-board Evolution of Robot Controllers. In Pierre Collet et al., *Artificial Evolution, 9th International Conference, Evolution Artificielle, EA, 2009, Strasbourg, France, October 26-28, 2009*, Pages 110–121, Springer-Verlag, Berlin / Heidelberg.

Evert Haasdijk, A.E. Eiben and Giorgos Karafotias (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Pages 1–7, IEEE Press, Piscataway, NY.

Evert Haasdijk and Arif Atta-ul-Qayyum and A.E. Eiben (2011). Racing to Improve On-line, On-board Evolutionary Robotics.In Natalio Krasnogor et al., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Pages 187–194, ACM, NY.

Evert Haasdijk, S.K. Smit and A.E. Eiben. Exploratory Analysis of an On-line Evolutionary Algorithm in Simulated Robots. To appear in *Evolutionary Intelligence*, Springer-Verlag, Berlin / Heidelberg.

Giorgos Karafotias, Evert Haasdijk and A.E. Eiben (2011). An Algorithm for Distributed On-line, On-board Evolutionary Robotics. In Natalio Krasnogor et al., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Pages 171–178, ACM, NY.

Robert-Jan Huijsman, Evert Haasdijk and A.E. Eiben (2011). An On-line On-board Distributed Algorithm for Evolutionary Robotics. In Jin-Kao Hao et al., *Artificial Evolution, 10th International Conference, Evolution Artificielle, EA, 2011, Angers, France, October 24-26, 2011*, Pages 119–131, Springer-Verlag, Berlin / Heidelberg.

P. García-Sánchez, A. E. Eiben, E. Haasdijk, B. Weel and J.J. Merelo (2012). Testing diversity-enhancing migration policies for hybrid on-line evolution of robot controllers. In Di Chio et al., *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, Pages 52–62, , Springer-Verlag, Berlin / Heidelberg.

Berend Weel and Evert Haasdijk and A.E. Eiben (2012). The Emergence of Multi-Robot Organisms using On-line On-board Evolution. In Di Chio et al., *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, Pages 124–134, , Springer-Verlag, Berlin / Heidelberg..

A.E. Eiben, S. Kernbach and Evert Haasdijk (2012). Embodied Artificial Evolution – Artificial Evolutionary Systems in the 21st Century. To appear in *Evolutionary Intelligence*, Springer-Verlag, Berlin / Heidelberg.

Without deviation from the norm, progress is not possible

Frank Zappa

3 More Than the Sum of its Parts

Individual, Social and Evolutionary Adaptation in Collective Systems

This chapter focusses on adaptivity as a pivotal enabler of future robotic systems. It is *the* fundamental premise of our vision that future robots will have to be capable of autonomous adaptation, that is, able to change their control systems without human intervention.

To define adaptation –"learning control without human supervision," in the words of Nelson et al. (2009) – clearly, consider a robot's controller as a process that maps inputs, read from the robot's sensors and internal states, to outputs, typically actuator and state settings. Adaptation is then defined as any changes to this mapping process, including the setting of its parameters.

This chapter appeared as part of:

Evert Haasdijk, A.E. Eiben, Alan F.T. Winfield (2011). Individual, Social and Evolutionary Adaptation in Collective Systems. Chapter 13 in S. Kernbach *Handbook of Collective Robotics - Fundamentals and Challenges*, Pan Stanford Publishing, Singapore.

According to this definition, changing the output threshold on some artificial neural net controller constitutes adaptation because the mapping from in- to outputs changes, but varying outputs due to some internal state does not, because the mapping remains the same, even though behaviour changes. Adaptation is necessarily on-line and without human intervention: the robot controller changes on-the-fly, as it goes about its tasks. We can distinguish two stages in the robot life-cycle: design time and operational time, separated by deployment. In these terms, adaptivity amounts to changing robot controllers autonomously during operational time. There are various optimisation and design techniques based on adaptive systems, e.g., evolutionary algorithms, particle swarm optimisation, neural networks, etc., that can outperform traditional methods. Such techniques can be used, and often are to great effect, during design time to find (near-)optimal robot controllers. However, these adaptive techniques fall outside of the scope of this chapter if the controllers remain static after deployment.

The vision that underlies this chapter is that adaptivity is a necessary feature in collective robotic systems to cope with a number of fundamental challenges:

1. *Unforeseen environment* The environment where the robots operate may not be fully known during the design process. Therefore, the robot controllers at the time of deployment are only approximate solutions that need to be adapted to the real requirements during operational time.

2. *Changing environment* The environment may change to such an extent that the given skill set of the robots is not adequate anymore. In a robot collective this environment might include the robots' social environment. Hence, controllers must adapt to the new situation.

3. *Reality gap* Even if the environment is known in advance and is not changing during operational time, it is very likely that the design process is based on approximations and simulations of the real operational conditions. Hence, the robot controllers have to be fine-tuned after deployment.

In this chapter we elaborate on the notion of adaptation and place adaptive systems into one conceptual framework, called *Population-based Adaptive Systems* (PAS). The notion of PAS serves as the unifying concept and the name PAS as an umbrella term. Within this framework we further distinguish different types of adaptation. One of the fundamental distinctions we make is based on differentiating learning and evolution. In turn, this is based on distinguishing phenotypes

and genotypes regarding robot controllers (Eiben et al., 2010a). Simply put, this distinction means that:

- We perceive the controllers with all their structural and procedural complexity as phenotypes.
- We introduce a (typically structurally simpler) representation of the controllers as genotypes.
- We define a mapping from genotypes to phenotypes, that might be a simple mapping or a highly complex transformation.

For example, a robot controller may consist of two artificial neural nets and a decision tree, where the decision tree specifies which ANN will be invoked to produce the robot's response in a given situation. This decision tree can be as simple as calling neural net 1 when the environment is lit and calling neural net 2 when the environment is dark. This complex controller, i.e., phenotype consisting of a decision tree and two artificial neural nets, could be represented by a simple genotype of two vectors, showing the weights of the hidden layer in neural net 1, respectively neural net 2. A technical distinction between learning and evolution is now straightforward if we postulate that learning acts at the phenotypic level, while evolution only affects the genotypes.

This chapter is structured as follows: section 3.1 establishes a framework that identifies three main forms of adaptation (evolution, individual learning, and social learning) in the context of population-based adaptive systems, ranging from artificial life systems to robot swarms. Section 3.2 presents a case study carried out in a system where individual learning and evolution are combined in such a way that they can directly influence each other, rather than acting independently on the agent/robot population. We demonstrate that in such a system learning – that optimises for the benefit of the individual – can effectively kill the population by ignoring the group level benefits of reproduction. In section 3.3 we investigate social learning as a mechanism to disseminate 'knowledge nuggets' –bits of adapted controller– in a population of agents/robots. Thus we show how the results of individual learning efforts (that would normally disappear if the individual dies) can be kept. In other words, here we demonstrate how social learning can facilitate the emergence of a knowledge reservoir in a population. While the experiments reported in these sections were conducted in an artificial life setting, the conclu-

sions are just as pertinent to robot swarms that implement combinations of these forms of adaptation. Finally, section 3.4 concludes the chapter.

3.1 Population-based Adaptive Systems

We coin the phrase "*Population-based Adaptive Systems*" (PAS) to label systems such as robot swarms or artificial life systems that have adaptive behaviour at agent and/or population level. Such systems can be characterised by two essential features:

- A group of basic units (agents or robots) that can perform actions, e.g., computation, communication, interaction, etc. By acting, these units exhibit behaviour – individual behaviour at unit level, as well as collective behaviour at the group level.
- The ability to adapt at individual and/or group level. If the exhibited behaviour is generated through behavioural rules[i] inside the units, then adaptation implies that these rules change. For instance, a change can take place inside an existing unit by replacing an existing rule by a new one, or a change can take place on population level by creating a new unit with a new set of rules.

There is a large variety of PASs with quite different examples. For instance, a peer-to-peer computer system where each node (peer) is able to improve its workings through experience, a genetic algorithm seeking an optimal solution to the travelling salesman problem, a group of learning robots collectively gathering red rocks on Mars, or a simulation of socio-economic processes by means of adaptive agent society. Such systems have received increasing interest over recent years with an increasing number of related papers. However, the lack of a common underlying framework of terminology means that the presentation of related problems and solutions shows a large (application dependent) variation. This forms an obstacle for identifying similar concepts, problems, solutions, etc. over various publications and implies the risk that individual researchers reinvent the wheel. A common conceptual framework describing a large class of PASs forms a helpful stepping stone towards futher developments in the area.

[i] We do not necessarily mean a set of `IF-THEN` rules, but any representation, including, for instance, neural nets, decision trees, etc.

We introduce the notion of Population-based Adaptive Systems and identify related concepts and research issues in this section. We focus our study on a class of PAS where adaptation occurs through three fundamental adaptation mechanisms: evolution, individual learning and social learning.

In the remainder of this section, we present a conceptual framework that captures a wide class of adaptive systems and identify research issues of general relevance in PAS.

3.1.1 Three Tiers of Adaptation

We use an agent-based metaphor, where the group of basic units is perceived as a population of agents (be they software agents or robots) whose behaviour is controlled by themselves — subject to environmental constraints, of course. That is, we assume that each agent has a controller that takes observations regarding the environment and the agent's internal state as input and generates actions to be executed by the agent as output. Furthermore, we assume that two levels of change can occur:

1. Changes at agent level: the controllers of the agents can change;

2. Changes at population level: it is possible to delete existing and to add new agents. In common parlance, this amounts to birth and death in the system.

As mentioned above, we see adaptation as change of controllers in a population of agents and distinguish three fundamentally different adaptation mechanisms. Denoting the set of all possible controllers by C, we can perceive adaptation mechanisms in PASs as search algorithms traversing the space C in a volume oriented manner – maintaining a population of controllers $P = \{c_1, \ldots, c_n\} \subset C$ simultaneously. Adaptation or learning then amounts to taking search steps, moving from the presently given set P of controllers to a new set P' and we distinguish adaptation on agent level (cf. property 1) and adaptation on population level (cf. property 2). We will call these *lifetime learning* and *evolution*, respectively. Furthermore, we make an additional distinction between two types of lifetime learning. In *individual learning*, an agent adapts its controller through a purely internal procedure, not through some oracle or other agents. If agents adapt their controllers by communicating controller information to each other and incorporating (good) pieces of

knowledge from each other, we speak of *social learning*. Figure 3.1 illustrates this taxonomy and the corresponding terminology.

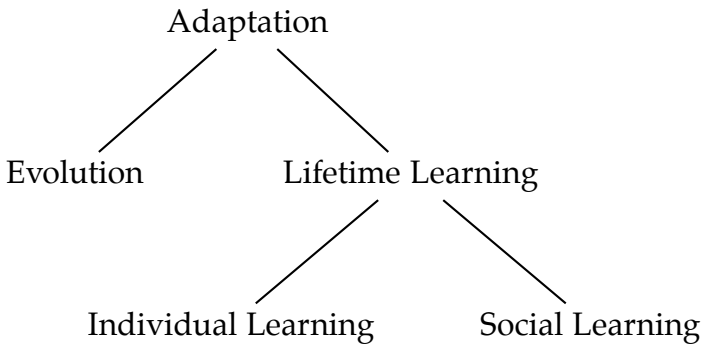


Figure 3.1 – Taxonomy of adaptation mechanisms in PASs

To delineate this framework, consider a few examples. (1) A genetic algorithm solving the Travelling Salesman Problem has birth and death, but the agents (individuals, candidate solutions) do not have a controller because they are not supposed to *do* anything other than producing offspring. Reproduction, moreover, is not actively controlled by the individuals themselves. Rather, they undergo it, arranged by an "oracle"–the outer loop of the evolutionary algorithm procedure. Thus, in this example we have no controllers and changes occur only at population level. (2) In embodied evolution as introduced by Ficici et al. (1999), the robots broadcast (possibly mutated) genes at a rate proportional to their fitness (measured as the number of batteries collected). Robots also resist "infection" at a similar rate. A good individual, collecting many batteries by virtue of its superior controller will infect many others before being replaced (i.e., infected) itself. If we see infection as death and immediate replacement, the robots in such a system do not adapt individually and changes occur at population level. (3) As a third example consider a single Web-agent serving a single user by selecting news items every morning using some given set of rules that are continuously improved through reinforcement learning. Here, the agent does have a controller (the rule set) that can change, but the population is a singleton and there is no death – no changes at population level. (4) Finally, consider the AEGIS artificial life system (Buresch et al., 2005; Eiben et al., 1999a), where a population of agents exists in an artificial habitat. The agents can move, eat, mate, fight, etc. as determined by their controllers and they undergo adaptation of their body characteristics (by evolution from generation to generation) and their controllers (by evolution from generation

to generation or by learning during lifetime). In this system, we have controllers and changes occur at both individual and population levels.

As an example of a PAS with adaptation through evolution as well as individual and social learning, we consider the NEW TIES system[ii], which we will describe before we elaborate on the three adaptation mechanisms (section 3.1.3) and their interactions (section 3.1.4) and research challenges these raise. Note, that NEW TIES serves as an example only and that, although we describe many design choices that were made for this particular system, the interactions between adaptation mechanisms that we describe are not specific to this example implementation and mostly do not depend on the design choices described.

3.1.2 The Environment and the Agents

The NEW TIES system provides a simulation platform in which a cultural society develops through evolution, individual learning and social learning of autonomous agents (Gilbert et al., 2006). The artificial, virtually embodied agents that make up this artificial society live in a grid world containing objects such as food sources (plants), tokens, places and building bricks.

In this world, time passes in discrete steps. Every time-step, the agents receive stimuli regarding objects (including agents) that they see or carry, messages from other agents that they hear and their internal state (e.g., their own energy level). The agents process these stimuli to select actions such as move or turn, pick up or put down objects, eat, communicate or interact otherwise with other agents (e.g., mating, or giving or taking objects to/from other agents). To process these inputs and arrive at a decision about which action to take, the agents use their individual controllers.

The project models agents anthropomorphically, thereby imposing strict autonomy, (virtual) embodiment and situatedness. This limits our options when designing agent interactions (e.g., agents cannot communicate unless they are within each other's vicinity), perception (e.g., they cannot see inside each other's heads) and learning mechanisms (e.g., no supervised learning).

Agents have to husband their energy: performing the selected action, even if that amounts to inactively surviving a time-step, costs energy. Should an agent run out of energy, it dies. To gain energy, an agent must eat food (plants). Other

[ii] New and Emerging World models Through Individual, Evolutionary and Social learning (NEW TIES), EU FP6 Project, `http://www.new-ties.org`

than that, agents die when they reach a certain maximum age. There is no other selection mechanism: as long as an agent lives, it can act—and therefore, engage in mating or social learning. To gain energy, an agent must eat food (plants). The laws of nature governing the environment determine the preconditions and the results of actions, e.g., specifying the amount of energy a plant yields when eaten and the costs of movement, the maximum lifetime for agents, or a minimum age and energy level at which agents can mate. Agents decide on their actions using a controller. In other words, the controller is the decision making unit inside an agent that maps inputs, i.e., perceptions of the agent regarding the world and its own internal state, to outputs, i.e., the agent's action.

3.1.2.1 Decision making and agent controllers

At every time-step, the agent processes the incoming information and describes the situation it finds itself in in terms of concepts. Then, based on this description, the agents decides on an action to perform.

Categorisation and conceptualisation To reduce the dimensionality of the observation space (the raw data where attributes are the elementary attributes of all possible entities in the world), a process of categorisation and conceptualisation map it onto another space, where the attributes are the so called concepts. Raw data is aggregated in two steps. First, it is aggregated to form categories that are then further aggregated to concepts. The incoming information is processed by categorising the raw data-bundle of features. Each feature concerning objects in the world, like color or shape, can be regarded as an axis in the features space; a category is defined by a range of possible values within the whole range of a feature. For example, for the feature `colour` everything between $1, \ldots, 10$ could be considered `green`. Concepts are (multi-dimensional) entities composed from (one-dimensional) categories. For instance, plants could be the `green` and `triangular` objects while agents could be `pink` and `circular`. Concepts are stored in an agent's ontology and are used to provide a characterisation of a given situation on a higher level than the original raw data.

Decision making The agents' controllers are implemented as special kinds of decision trees, *decision Q-trees* (DQTs). The 'Q' refers to the fact that they can be

adapted through Q-learning (Sutton and Barto, 1998), the NEW TIES implementation of individual learning. With crossover and mutation operators inspired by those used in genetic programming (Koza, 1992), these trees can also be adapted through evolution when two agents mate to create offspring.

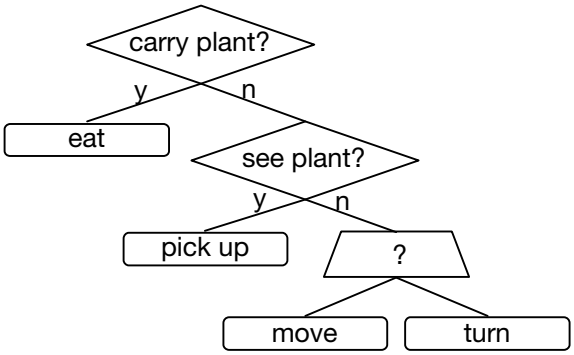


Figure 3.2 – A simplified example of a decision Q-tree (DQT).

DQTs consist of test, bias and action nodes (Fig. 3.2; depicted as lozenges, trapezoids and rounded rectangles, respectively).

A **test** node evaluates a Boolean query based on concepts known to the agent, e.g., "Is there a plant ahead?" or "Is there an agent nearby?", and depending on the answer (Yes or No) the tree is further traversed through either of the two child nodes. Thus, a full path from the root to a leaf (an action to be performed) node forms a conjunction of statements that together provide a partial situation description in terms of the agents' concepts.

To traverse a bias node, the agent probabilistically selects one of multiple branches for further traversal – each of these branches has a bias that determines the likelihood of it being selected. These biases are determined genetically through evolution and onto-genetically through individual and social learning.

The leaves of the DQT are action nodes that select an action. Action nodes, like bias nodes, are probabilistic: the actual action is stochastically chosen according to a weight distribution over all possible actions. The available actions are simple actions – such as move, turn-left or turn-right –, unary – such as eat(x) or hit(y) –, and binary actions such as give(a,o). The arguments for the higher arity actions are implied by the tests that were traversed to select an action –e.g., testing for visible agents implicitly selects all agents in sight– and can be any object, but if, e.g., an agent attempts to eat a non-food item, this action will fail in the world.

3.1.3 Adaptation Mechanisms

As outlined in section 3.1, we envision adaptation as the change of controllers in a population of agents. In NEW TIES, this amounts to changing DQTs. In this subsection we discuss how the general trinity of adaptation is instantiated in NEW TIES. To begin with, we note that all three adaptation mechanisms work in the same search space – that of all possible DQTs.

3.1.3.1 Evolution

NEW TIES deliberately adopts a non-Lamarckian notion of evolution (Lamarck, 1809), so inheritable material cannot change during an agent's lifetime. This means that an agent created with a controller c seeds its descendants by exactly this controller c, regardless of any changes brought by lifetime learning.

The two pillars of evolution are selection and variation; variation is realised by straightforward tree-crossover and tree-mutation operators, much as in genetic programming. Viewing adaptation as search through the space of controllers, one elementary search step in this context amounts to combining two existing controllers c_1 and c_2 into a new one c_3.

It is an essential aspect of this system that selection is not based on some objective function to be maximised (Menczer and Belew, 1996; Mitchell and Forrest, 1994). Survivor selection is strongly environmental: agents die if they run out of energy or reach the maximum age. As for parent selection, an agent can decide any time to mate (subject to some constraints). If the controller chooses to mate, the agent selects itself as a would-be parent. To procreate, it needs to find and "convince" another agent: it sends a special message, a mate proposal. Only if the other agent accepts this mate proposal do the two agents become actual parents and produce a child. To make the child viable, each parent donates a portion of its current energy, consequently incurring a cost.

The NEW TIES evolutionary system differs from usual evolutionary algorithms in a number of essential aspects.

1. Fitness is not an a priori utility measure that determines the number of offspring. One could say there is no notion of fitness at all, or rather, that in NEW TIES fitness is a secondary, observable measure determined by the number of offspring rather than vice versa – a truly Darwinian notion.

2. Reproduction is not orchestrated by some central authority. Individuals autonomously and asynchronously decide to mate.

3. Reproduction is detached from survivor selection. Newly produced individuals can be added to the population without removing old ones. Likewise, an individual can die without being replaced by a new one. As a side-effect, there is no clear definition of a generation here.

These properties have two prominent consequences. Firstly, in the absence of an explicit objective function the selection probabilities (that embody the system bias for quality) must be based on indirect quality indicators. In general, the age and the energy level of agents can be used here: an agent that survives for a long period and/or has accumulated much energy must be well adapted, hence worthy of being reproduced. In this respect, PAS of this kind are closer to natural selection than, for instance, Genetic Algorithms where selection probabilities are calculated from an objective function.

The second effect is that points 2 and 3 imply a kind of reproduction –"*natural reproduction*"– where the population size inherently changes over time. Users of such systems face a tough challenge concerning the calibration of the system to avoid unlimited population growth (explosion) or complete extinction (implosion). In a particular system, such as NEW TIES or AEGIS, ad hoc solutions can work, based on balancing energy supply (number of plants, energy of plants, reproduction rate of plants) and energy consumption (costs of actions). From a general evolutionary point of view, population size can be controlled by tuning the selection mechanisms. For instance, the parameters specifying the minimum age or energy required for mating. At the moment, there are no general guidelines or design heuristics available to cope with this problem.

3.1.3.2 Individual Learning

A newborn agent, and with it individual learning, starts with the controller that is provided by (one of) its parents. The most appropriate learning type for individual learning is reward based: supervised learning is difficult, because agents can be in an environment of which the most optimal (set of) action(s) is unknown. Unsupervised learning is inappropriate, because information present in the environment is wasted if not used as feedback for learning.

NEW TIES implements reward based individual learning as reinforcement learning (Kaelbling et al., 1996; Sutton and Barto, 1998). Reinforcement learning can change the DQT by policy change. An agent's *policy* is –in the context of reinforcement learning– represented by its DQ tree. Any path in the DQT leading to an action is a result of the policy. Policies can be altered by changing the values of the edges that change the likelihood of taking a specific path. NEW TIES uses SARSA, one of the variants of Q-learning (Sutton and Barto, 1998).

In NEW TIES, the reward is usually based on energy, but other types of 'currency', e.g. something based on emotions or some mix of simpler currencies, are possible. The currency must in any case be accessible to the agent, or the agent would not able to use it for computing rewards. Such a mixture is probably needed for the problem described in section 3.1.4.1, where agents would unlearn to reproduce if reward is only based on energy – this is investigated in detail in section 3.2.

An important challenge for reinforcement learning is that the state-space created by the perceptual input is huge. To illustrate, the state-space for the visual field is $\#typeOfObjects^{\#gridcellsVisualField}$. Given that NEW TIES has at least 3 types of objects and that the visual field is 50 grid cells, it is obvious that the state space is very large, maybe intractably so. Moreover, the state space is further extended by non-visual perceptual input of auditory, internal and reproductive stimuli. To cope with the size of this state-space, it is not partitioned by the input stimuli, but by the tests in the test nodes of the DQT. The tests in a test-node test for certain concepts, for instance `green agent`. This divides the state space in agents that are green and all other coloured objects. The test-node uses the input, only testing for particular aspects of the environment.

3.1.3.3 Social Learning

Many studies have focussed on social learning with approaches including imitation, copying behaviour as well as using socially provided corrective feedback (Dautenhahn and Nehaniv, 2002; Acerbi et al., 2008). In NEW TIES, by contrast, agents communicate explicitly and social learning entails an agent modifying its controller by incorporating a piece of knowledge it receives from another agent. Social learning requires at least two agents a_1 and a_2 with controllers c_1 and c_2;

one search step amounts to changing c_1 into c_1' (assuming that a_1 learns from a_2), where c_1' is some combination of c_1 and c_2.[iii]

Agents communicating in this manner implies a multi-faceted set of features and parameters that govern issues such as (social) networks of knowledge exchange, levels of trust and relative merit of knowledge, etc. In general, they concern:

– when and with whom to exchange knowledge;

– the selection of knowledge to send or elicit;

– when and how to accept offered knowledge.

Obviously, a general consideration when designing these features is including a bias for quality. In other words, at least some of the choices involved in importing a "knowledge nugget" from another agent must favor learning from a better agent. Similar to introducing a bias for quality in evolution (cf. section 3.1.3.1), the age and the energy level of agents can be used as quality indicators here. Apart from any specific quality-driven social learning scenario, there is always qualitative pressure as described in section 3.1.3.1: agents with poor controllers die sooner and therefore cannot participate in social learning exchanges ("teach") as often as agents with good controllers.

Note, that communication introduces a "social dimension"; an overlay network in technical terms. The properties of this network depend on the given implementation, but in general, the network changes over time. In NEW TIES, this is realised by a protocol similar to gossiping in peer-to-peer systems. Every agent maintains a (fixed length) list of acquaintances – agents it has seen and talked to before. This list is updated with new observations (encounters with other agents) using a FIFO policy. The construction and maintenance of this social network can also be influenced by quality indicators of peers.

A knowledge nugget in our system is represented by a sub-DQT (extracted from the sender's controller). In the current implementation, this sub-DQT is included in the tree of the receiving agent by inserting –at some appropriate location in the DQT– a bias node that has two children: the foreign sub-DQT and the al-

[iii] Remember the non-Lamarckian nature of NEW TIES' reproduction: these controller changes do not affect the genetic material (which in effect is a copy of the initial controller with which an agent is created).

ready existing native sub-DQT. These alternatives are weighted by newly defined biases based on the ratio between the sender and recipient's age and energy levels.

Section 3.3.2 provides a more detailed description of the social learning mechanism in NEW TIES.

3.1.4 Relationships Between Adaptation Mechanisms

To position evolution, individual learning, and social learning it is helpful to consider them from the knowledge transfer perspective, where knowledge is seen as (good) pieces in the agent controllers. From this point of view, knowledge is transferred vertically by evolution, down along the line of descendants. (Recall the note from section 3.1.3.1 that we do not have a clear notion of generations here, because agents residing on different levels of the family tree can live at the same time in the same population.) On the other hand, individual learning is a sink: in the absence of social learning, individually accumulated knowledge simply disappears when the agent carrying it dies. Social learning can alleviate this, since it amounts to horizontal knowledge transfer, passing knowledge nuggets within the current population. In this respect, social learning makes the population into a knowledge reservoir, reducing (at least potentially) the risk that knowledge must be rediscovered over and over again.

3.1.4.1 Evolutionary and Lifetime Learning

A marked distinction between evolution and lifetime learning is that evolutionary operators do not change the controllers of agents during their lifetime, while lifetime learning operators obviously do. If evolution were the only adaptation mechanism, agents would die with the controller they were born with. Hence, evolution does not take place on an individual, but strictly on a population level. From this perspective, the death of an agent represents a contribution to the evolution process, because the population adapts with each death.[iv] This is particularly not the case for individual learning, where the death of an agent terminally destroys the results of the learning process.

In our example, evolution also differs from lifetime learning in the entity that initiates a learning step: individual learning and social learning steps are initiated without the influence of the agent's controller – by an oracle, or subconsciously,

[iv] Supposedly changing for the better, cf. survival of the fittest.

if you will. This is not the case for evolution search steps, because the agent has to decide itself to reproduce by sending or accepting a mate proposal. As a compelling consequence, agents can unlearn reproduction through lifetime learning because the individual reward for mating is negative: it costs energy without any mitigating personal benefit. To counteract such tendencies, one can introduce some specific reward for mating (orgasm), make mating a subconscious process or take population-level benefits into account in lifetime learning.[v] Section 3.2 investigates this consequence in detail.

Memetic algorithm research has pointed out positive interactions between evolution and lifetime learning, by showing that combinations of evolution and individual learning are particularly beneficial (Krasnogor, 2002). An interesting and promising interaction between evolution and lifetime learning is described by Best (1999). This study finds that lifetime learning decreases the need for evolution to get it spot-on: the chance of finding the optimal solution is much greater with lifetime learning and evolution combined.

3.1.4.2 Individual and Social Learning

As noted above, the non-Lamarckian nature of evolution in NEW TIES entails that knowledge that an agent acquires through individual learning cannot affect inheritable material, and therefore is lost when that agent dies. By proliferating knowledge over the population of agents, social learning preserves such knowledge pieces that would otherwise disappear. Thus, social learning turns the population into a reservoir of (individually acquired) knowledge.

A system that combines individual learning and social learning can be thought of as having division of labour: individual learning generates novel knowledge nuggets and social learning disseminates these. Social learning can also be seen as an accelerator making the system more efficient. Think, for instance, of agent a_1 learning x, agent a_2 learning y and a_1 and a_2 telling x and y to each other, rather than having to learn these knowledge pieces themselves. In general, efficiency improves if the costs of, respectively, time needed for and learning through communication, are lower for the agents than the costs/time of acquiring knowledge individually – an assumption that holds in a great many systems. As a net effect, combining social learning and individual learning allows agents to possess knowl-

[v] Taking a learning step in both individual learning and social learning could also be made into a conscious action, in which case similar considerations would apply.

edge regarding situations they never encountered themselves, acquired at greater speed and at lower costs. Such constellations have been shown to outperform either adaptation mechanism by itself, e.g. by Bull et al. (2007).

Section 3.3 investigates this interaction in detail.

3.1.4.3 Individual and Social Learning as Evolution

Recall from section 3.1.3.3 that knowledge nuggets are sub-DQTs. Incorporating such sub-DQTs into an agent's controller amounts to an operation similar to crossover in GP. Similarly, one can see an analogy between a learning step in individual learning and a GP mutation operator: both turn some controller c into c'. From this perspective it is quite natural to see the combination of individual learning and social learning as an evolutionary process. Similar observations were reported by e.g. Bull et al. (2007), Smith et al. (2000) and Richerson and Boyd (2005).

The selection components for this evolutionary system consist of the mechanisms regulating when two agents engage in sending/receiving knowledge pieces (parent selection)[vi] and the policies to accept and incorporate a received piece of knowledge (survivor selection).

It should be noted that this constitutes an evolutionary process quite different from the one described in section 3.1.3.1. The most visible difference lies in the replacement strategies: in the lifetime learning-based evolutionary process, reproduction and survivor selection *are* coupled: a new controller, whether made by mutation or crossover, immediately replaces an existing one: its parent and the population size remains unaffected. Another difference is that here, a new controller is created by either crossover (social learning step) *or* mutation (individual learning step), while in evolution this happens by crossover *and* mutation (which occurs sequentially in the reproduction procedure). Furthermore, we should note that here we *do* have an explicit fitness measure, used in at least some parts of the system. For the parent selection component this is not necessarily the case; an agent can perform a mutation (do an individual learning step) regardless of the quality of its present controller c – making c the parent of the new c' – and the same holds for an agent a_1 deciding to talk to a_2 – selecting their controllers c_1 and c_2 as would-be parents. We can distinguish two cases of survivor selection: in the case of mutation (an individual learning step), survivor selection does not involve fitness either: the old c (the controller being improved by individual learning) is

[vi] Combined with Darwinian survival of the fittest as described in 3.1.3.1

simply deselected and replaced by c' (the improved controller). However, if a new controller is created by crossover (an social learning step), a utility function is used to determine the relative merit of the received knowledge when integrating it with the already known c_1 to create the new c_1'. This utility is related to the relative ages and energy levels of the two agents involved.

Considering individual learning and social learning in this light raises two prominent research questions. First, how does existing evolutionary computing knowledge, e.g., regarding variation, selection and their balance, translate into these contexts? Second, how do the two evolutionary processes, genetic evolution on the one hand, social learning and individual learning on the other, interact in one system?

3.1.5 Discussion

Most of the technical details we introduced are merely illustrative in the sense that they do not restrict the generality of our discussions. Using trees to represent agent controllers is one such detail. Our line of thought about variation operators in evolution and merge operators in social learning can be repeated for other data structures as well. A similar argument holds for the categorisation and conceptualisation mechanism to pre-process sensory input of the agents; the general point here is dimensionality reduction. This is critical when using reinforcement learning algorithms, because they scale very badly with the size of the state-space, but this aspect is likely to occur in many systems.

The main contribution of this chapter is the definition of a system where three different adaptation mechanisms – genetic evolution, individual learning, and social learning – can work simultaneously, yet clearly distinctly. The separation of the learning mechanisms is based on a differentiation between inheritable and learnable agent characteristics.[vii] Designating agent characteristics as inheritable or learnable is one of the major design decision when implementing PASs. Inheritable properties can undergo evolution through appropriate variation operators and environmental selection, learnable properties can undergo lifetime learning through individual and social learning. By the clear separation between evolution, individual learning, and social learning, particular adaptation mechanisms can be switched on and off independently, thus allowing research on their effects sepa-

[vii] In the system we described, these are the same, but our considerations are still valid if this is not the case.

rately or in various combinations. This allows us to gain insight in their mutual effects on each other and on the adapting population. Research in this area offers great benefits by the high potential of "fully powered" adaptive systems. In general:

- Social learning can act as an accelerator for individual learning in each agent and can preserve the individually discovered knowledge nuggets for the population that would otherwise be lost after the death of the individual that learned them;
- The combination of individual learning and social learning can be seen as an evolutionary system, creating an opportunity to use existing knowledge in evolutionary computing when designing such combined systems.

The specific choices concerning evolution in NEW TIES are reflected in our treatment of evolution. In particular, we focus on systems with natural reproduction, cf. section 3.1.3.1. In many applications, e.g., ALife, social simulations, peer-to-peer systems, this is the obvious choice of reproduction scenario, so we can safely state that the subset of PAS with natural reproduction is large and interesting. Considering such systems we observed that:

- In an evolutionary process relying on natural reproduction, population size is inherently volatile. This creates a tough challenge for designers and users of such systems: to design (selection) mechanisms that prevent explosion and implosion of the population;
- While, in general, combining lifetime learning and evolution is a powerful combination (cf. memetic algorithms), in PAS with natural reproduction lifetime learning can counteract evolution by unlearning mating.

In many instances of PAS, one is mainly interested in emergent phenomena, particularly in emerging behaviour and emerging structures, such as the controllers of the agents (world models) or the social network generated by social learning, or the emergence of 'traditions' in the socially learned behaviours across the population. It is characteristic that the experimenters can influence system properties only indirectly, via the adaptation mechanisms. Given some demanding world where agents only survive if they adapt to the particular challenges of that world, the experimenter's task is to engineer an appropriate mix of the

adaptation mechanisms so that these mechanisms will generate the desired emergent behaviours and structures. It is this aspect that makes understanding the trichotomy of evolution, individual learning, and social learning crucial to apply them successfully in any PAS, be it NEW TIES or a robot swarm.

3.2 Learning Benefits Evolution

This section considers the interplay between two of the three levels of adaptation introduced in section 3.1, namely evolution and individual learning. Combinations of evolution and learning have been investigated before (Belew and Mitchell, 1996), cf. the hundred years of the Baldwin effect (Turney et al., 1996). Prominent related work can be found within memetic algorithms, or hybrid evolutionary algorithms (Moscato, 1999; Krasnogor, 2002), evolutionary robotics (Nolfi and Floreano, 1999; Ijspeert et al., 1998) and ALife (Todd and Miller, 1990; Belew et al., 1990; Munroe and Cangelosi, 2002; Curran and O'Riordan, 2006; Buresch et al., 2005).

As described in section 3.1.3.1, the combination of features in NEW TIES implies that the population size can change, even to extinction. This property is typically absent in related work[viii], even some work that claims to model natural systems (Ruppin, 2002), although it is evident that in nature populations can and do die out. Past research by Hinton and Nowlan (1996); Mayley (1996); Munroe and Cangelosi (2002); Nolfi and Floreano (1999) has focussed on the costs and benefits of learning in evolution and on identifying factors that influence this relationship (Mayley, 1996; Nolfi and Floreano, 1999). This section continues research in this direction, but specifically in the context of a changing population size.

Remember that in NEW TIES, the evolutionary mechanism is under the control of the agents, because it is the agents themselves who decide if and when to create offspring. This means that the development of agent controllers (through evolution and/or learning) can lead to intensively reproducing agents or just the opposite: the evolutionary mechanism itself is subject to changes over time.

Evolution and individual learning act in a common search space: that of the set of all possible agent controllers. Hence, an agent can be born with controller C, created by some evolutionary operators applied to its parents' controllers, and can change C into C', C'', etc., during its lifetime through individual learning. Evolu-

[viii] Research on predator-prey phenomena is not usually concerned with combinations of evolution and learning.

tion is non-Lamarckian: when this agent reproduces, only its original controller C is used for creating a child, any individually learned modifications in C' etc. are disregarded as inheritable material. As noted in section 3.1.3.2, individual learning is implemented as reinforcement learning. In essence, reinforcement learning changes the controller by regulating agent preferences for actions based on a reward system. It is important to note that reinforcement learning can strengthen or weaken preferences for any agent action, including the mating action required for offspring creation. Thus, it is possible that individual learning unlearns reproduction and effectively counteracts evolution.

The questions we have to ask ourselves, then, are these:

1. What is the effect of adding individual learning through reinforcement learning?
   - On the viability of the population?
   - On the performance of the population?
   - On the evolutionary engine?
2. How does this depend on the rewards used by reinforcement learning? In particular:
   - When rewards are energy-based.
   - When rewards are hard-wired by the user.

3.2.1 The Experiments

As noted above, the system is not meant to set the agents any specific task other than to win the struggle for life. The environment can, of course, be set up to challenge the agents in specific ways. The agents then have to deal with these challenges in order to survive and prosper. In other words, an experimental set-up in NEW TIES represents a particular challenge or learning task that agents must solve through adaptation.

In the experiments we describe here, the environment is set up so that agents can only survive if they successfully tackle the well-known poisonous food problem (Cangelosi and Parisi, 1998; Nolfi and Parisi, 1995; Todd and Miller, 1990). The agents find themselves in an environment where there are two types of plants, both of which can be picked up and eaten. One type is nutritious and yields an

energy increase, the other type is poisonous and eating them actually drains energy. Agents can choose not to, but they can distinguish between the two types of plant. They do not, however, know *a priori* that one kind – let alone which kind – is poisonous. Because agents *must* eat to replenish their energy level as mentioned above, they have to learn to disregard poisonous food if they are to survive.

In these experiments, the world is a 200×200 grid, initialised with 500 agents, 8,000 edible plants and 10,000 poisonous plants. There is a maximum to the number of agents: agents are unable to reproduce when this limit is reached, but it may be exceeded through the concurrent creation of a number of new-born agents. Agents and both types of plants are randomly distributed over the grid. We call our atomic time step a day and 365 days a year; the minimum mating age for agents 1,000 days: i.e., they cannot successfully reproduce for the first 1,000 days of their lives. The maximum age for agents is 7,300 days (7.3 times the minimum mating age), when they reach this age they die, whatever their energy level. The experiments run for 30,000 days. Initially, agents are assigned a random age between zero and one year. The initial controller of all agents is the same; in this controller some behaviours are pre-wired[ix], like looking for food. However, the behaviour for eating the correct type of food is not present. This can be acquired by changing the tree structure and/or tuning the biases of bias nodes and action bias nodes, although the probability that the latter succeeds is small in the tree-structure of the initial controller. Evolution (without sub-tree mutation) and individual learning are the only active adaptation mechanisms; social learning is turned off.

3.2.1.1 Measurements

To answer the research questions we must measure the viability and performance of the population and provide insights into the evolutionary engine.

To measure the viability, we use the population size. A run is considered successful if the population size did not reach zero during that run.

[ix] Pre-wired is not the same as hard-wired: pre-wired controllers can be modified by the adaptation mechanisms.

As a behavioural performance measure we use a function based on the ratio of the different types of food the population eats:

$$g(t) = \frac{\sum_{t-1}^{t} eat_h}{\sum_{t-1}^{t} eat_p + \sum_{t-1}^{t} eat_h} \tag{3.1}$$

Where $\sum_{t-1}^{t} eat_h$ and $\sum_{t-1}^{t} eat_p$ are the number of wholesome and poisonous plants eaten by the population between $t-1$ and t.

Additionally, we measure the total and average energy of the population and the total and average age.

To measure the performance of the evolutionary system we monitor the average number of mate-agreements of the population.

3.2.2 Experiment I

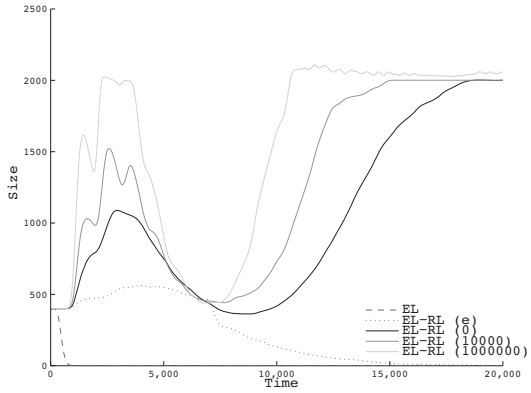
In the first experiment, poisonous plants drain twice the energy that an edible plant yields. We ran two sets of experiments; one where individual learning was either turned off or used only energy-based rewards and one where we introduced a specific reward for reproduction. The results are summarised in figure 3.3.

3.2.2.1 Evolution only and Evolution-reinforcement learning combination with energy based rewards

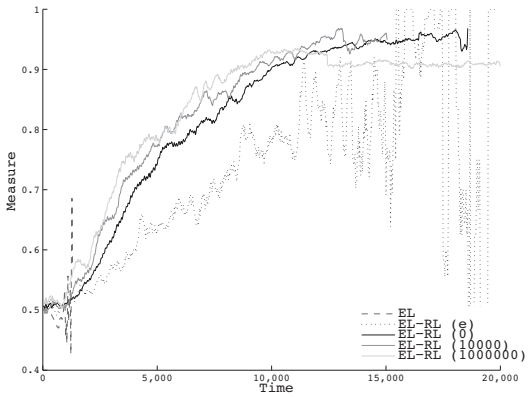
Figure 3.3(a) shows clearly that evolution only (indicated by the dashed line "EL") survives for approximately 1,000 time steps and thus does not yield a viable population. Adding energy based reinforcement learning to evolution markedly improves viability, as can be seen in figure 3.3(a) (the dotted line "EL-RL (e)"). In the long run, however, this is not a viable solution, because after 15,000 time-steps the population is as good as extinct.

So, the combination with energy based reinforcement learning is thus unable to make a population viable. This might be because reinforcement learning is unlearning reproduction, since it costs energy and therefore produces negative rewards. The rewards for other actions, except the eat action, are also negative, but usually not as bad as reproduction, because that costs a third of the agent's energy. The "EL-RL (e)" curve in figure 3.3(c) proves that reward based reinforcement learning is unlearning reproduction, because the total number of mate-agreements steadily decreases. Moreover, figure 3.3(d) indicates that agents do not reproduce

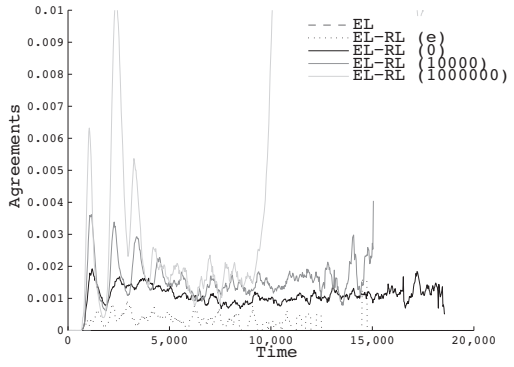
(a) Population Sizes



(b) g Measure



(c) Average Number of Mate Agreements per agent per time-step



(d) Average Age

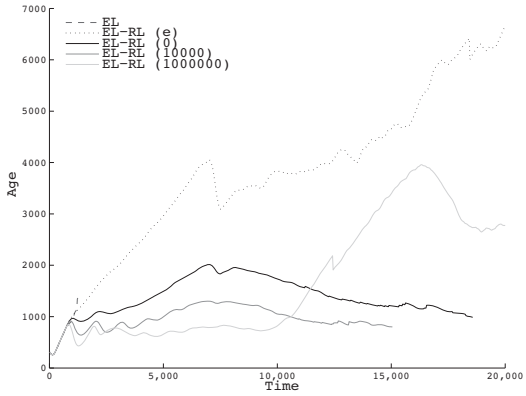


Figure 3.3 – Results for experiment I; graphs compare results for evolution by itself (EL), combined with reinforcement learning based on energy (EL-RL (e)) and combined with reinforcement learning with different "orgasm" levels (EL-RL (0;10,000;1,000,000)).

enough to sustain the population: agents reproduce once every 3,000 time-steps, while the average age in the population is only 1,000.

There are two reasons why agents reproduce at all in the face of the negative reward. Firstly, they have to try to reproduce at least once to learn its negative effects. Secondly, during exploration agents can still choose the mate-agreement action, even if they unlearned it. The periodic behaviour of the curve is a side-effect of the setting of the ages of the initial population and the minimal reproduction age.

3.2.2.2 Combination of evolution and reinforcement learning with a hardwired reward

The results in the previous subsection suggest that reproduction is unlearned or becomes so rare that the agent population is unable to sustain itself. To test the explanation that this is due to agents receiving negative rewards for reproduction, we introduce a special reward for reproduction. Its only role is to make mating actions attractive, so it can be regarded as a kind of pleasure or orgasm. We ran experiments with three levels of reward: 0, 10,000 and 1,000,000.

The most striking result is that a hardwired positive reward renders the population viable. Note, that even a reward of zero works because all other actions except eating yield a negative reward.

In terms of population performance, the results show that higher rewards for reproduction result in better performance. For instance, in the g measure graphs (figure 3.3(b)), the curve for a reproduction reward of 1,000,000 increases much more steeply than for a value of 10,000, indicating that the population learns to avoid poisonous food very quickly.

The intensity of the evolutionary engine is measured by the number of mate-agreements, displayed in figure 3.3(c). The general trend is that the higher the mate-reward the higher the number of mate-agreements.

Note that in all different simulations, including that of evolution alone, the g measure is similar for the first 1,000 time steps. This means that the combination of reinforcement learning and evolution is unable to learn the task during this period, implying that individual learning somehow keeps agents alive that would die in the case of evolution alone. To find out how agents were able to survive, we analysed the results by tracking the agents' actions. This analysis showed that agents often choose to do nothing (the NULL action). Agents thus learn to save

their energy. This suggests a *hiding* effect: individual learning preserves agents with a non-optimal strategy (Mayley, 1996).

3.2.3 Experiment II

To test whether a hiding effect occurs as suggested above, we change the environment so that evolution alone can make the population viable. The only change from the previous experiments is that the levels of nutrition and poisonousness have been set so that a poisonous plant drains an equal amount of energy that an edible plant yields. We run experiments with evolution only and with both evolution and reinforcement learning. The average results over 10 runs are shown in figure 3.4.

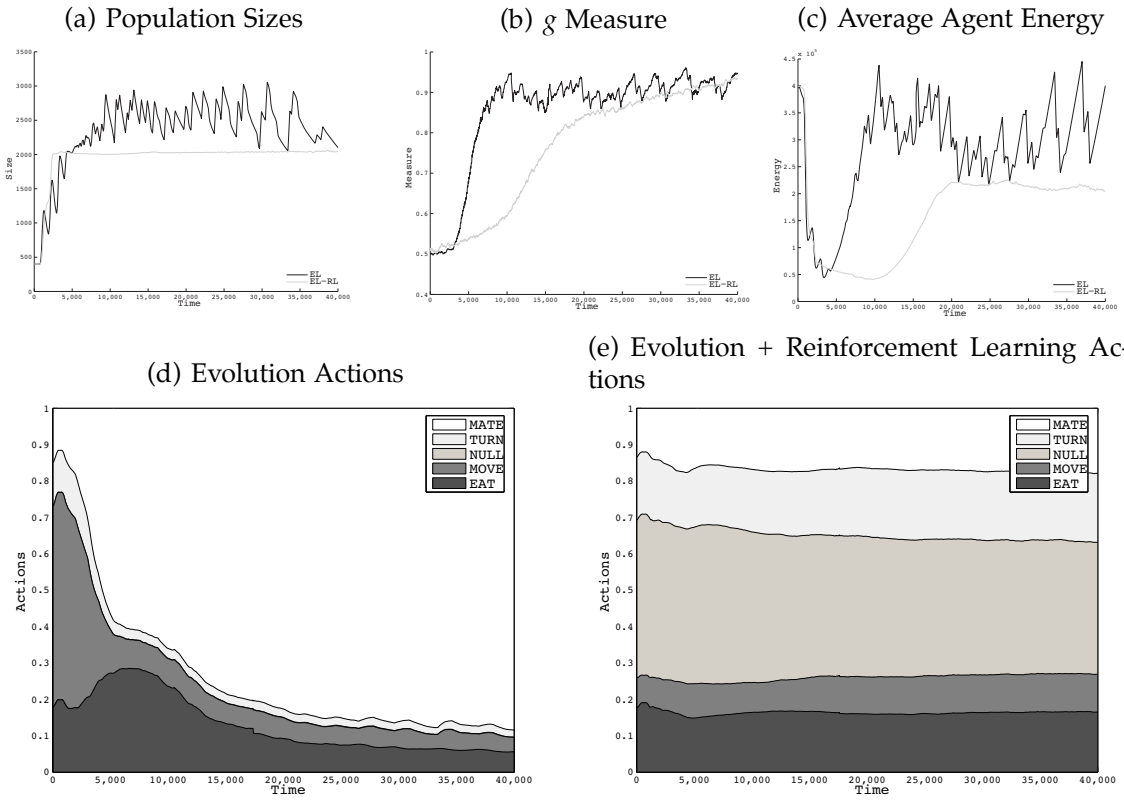


Figure 3.4 – Results for experiment II; graphs (a), (b) and (c) compare results for evolution by itself (EL) and combined with reinforcement learning (EL-RL).

Both the combination of evolution and reinforcement learning and evolution by itself yield viable populations in this set-up. The soft cap on population size causes

some boundary effects such as the fluctuating population size and the decrease of energy levels at some point.

The main result, however, is that there clearly is a hiding effect: the combination of evolution and reinforcement learning hides the ill-adapted nature of non-optimal agents. The population with only evolution very rapidly learns to eat only wholesome plants (figure 3.4(b)) and therefore accumulates much more much energy than the combination of evolution and reinforcement learning (figure 3.4(c)).

With evolution only, the agents accumulate enormous reserves of energy so that they can get by without any food; this changes the evolutionary pressure from eating (and preferring edible plants) to reproduction: we see the evolution of agents that only perform actions involved with reproduction.

Figure 3.4(e) clearly shows that the average number of mate-agreements is much lower with the combination of evolution and reinforcement learning than with only evolution; reinforcement learning apparently hinders evolution. The difference in number of mate-agreements already appears within the first 5,000 time steps, while there us no appreciable difference in g value or population size to explain the difference. One possible explanation is that the combination of evolution and reinforcement learning creates another type of agent that doesn't reproduce as often because while evolution is mainly focussed on reproduction, the agents with reinforcement learning try to balance between both eating and reproduction in order to maximise their rewards.

3.2.4 Discussion

Over the years there has been a fair amount of research into combinations of learning and evolution, in particular regarding their costs, benefits (Hinton and Nowlan, 1996; Mayley, 1996; Munroe and Cangelosi, 2002; Nolfi and Floreano, 1999) and factors that influence this relationship (Mayley, 1996; Nolfi and Floreano, 1999). We now (re)consider these issues in a context where agents:

- decide autonomously if and when they reproduce (*natural reproduction*, implying a dynamic population size);
- can adapt their controllers to unlearn the mating action.

Our experiments show that in such systems learning can counteract evolution. To be concrete: with a straightforward reward system based on energy, reinforce-

ment learning will cause the agents to lose interest in mating because of the high individual costs. Hereby the group benefits (maintaining the evolving population) are lost. This effect can be counteracted by introducing a specific reward for the mating action that gives positive feedback to the agents, regardless of the related energy costs. One could of course argue that this trick is known in nature, commonly called an orgasm. All in all, this indicates that we must consider the reward for reproduction as another factor that influences the effect of learning on evolution in addition to the list proposed by Mayley (1996).

In terms of the viability and performance of the population, our experiments show that learning can quite literally be a matter of life and death. In our first scenario, evolution by itself was not powerful enough to sustain the population. Adding reinforcement learning changed this, yielding populations that survive and prosper until the end of the simulations. Simply put: learning keeps the population alive. It can do so by creating controllers that minimise energy expenditure, a non-optimal behaviour, in the sense that such agents do not learn to eat the correct plant type. This is one of the costs of learning: learning causes a clear hiding effect because it allows non-optimal controllers to survive. By contrast, evolution by itself optimises by harshly cutting out the bad agents, but always with the risk that there is no population left. In a system allowing a changing population size this can be lethal.

Further research could show whether there is an optimal value for the reproduction reward (i.e., the extent of "pleasure" during mating). A good value would not frustrate evolution and still make a population viable when needed. One possibility is to make this value self-adaptive by adding it to the genome, allowing evolution to tune itself.

3.3 Social Learning as Enabler of a Knowledge Reservoir

As mentioned in section 3.1.3, agents in the NEW TIES PAS decide autonomically on the actions they perform by means of a controller that is inherited (for the initial population: generated) at birth. They implement evolution and reinforcement learning for individual learning. Through evolution, only the inherited controller is passed on (i.e. *non-Lamarckian* evolution (Lamarck, 1809)): agents do not inherit

knowledge (modifications to the controller) that their parents may have gained through experience; they can only recombine the controllers that their parents had at birth (with some mutation added). This means that, without some additional method of spreading knowledge through the population of agents, everything an agent learns through experience (i.e., through individual learning) will be lost when that agent dies.

This is where social learning comes into play: with social learning in place, anything an agent learns during its lifetime can be taught to other agents, so that this knowledge does not necessarily die with the agent that originally discovered it. With agents exchanging knowledge pieces –bits of adapted controller– through social learning, the population as a whole effectively becomes a knowledge repository – although not a randomly accessible one for individual agents – for individually discovered adaptations. Obviously, social learning can also speed up the learning process at the population level as found in e.g., (Acerbi and Nolfi, 2007; Denaro and Parisi, 1996; Bull et al., 2007)

Social learning can only play this role if it can effectively disseminate individually acquired knowledge pieces. The question, then, that we seek to answer is the following:

> *Is social learning an efficient mechanism to spread knowledge pieces through the population, thus creating a knowledge repository for individually acquired knowledge?*

In nature, social learning can be achieved through a host of mechanisms ranging from imitation to social guidance in individual learning (Acerbi and Nolfi, 2007). Here, we consider the case where social learning consists of agents actively suggesting behavioural rules (knowledge pieces) for the consideration of other agents in a peer-to-peer fashion. The recipients of these knowledge pieces then choose whether or not to integrate them into their own set of rules. The fact that all agents participate in social learning on an equal footing implies an inherent parallelism in the spreading of knowledge pieces: all agents that have acquired a knowledge piece can simultaneously share it with other agents, who can then share it in turn, and so on.

Cultural algorithms employ belief spaces (Reynolds, 1999), which can be seen as explicit knowledge repositories that the individuals build collectively. In the research presented in this subsection, however, knowledge repositories are formed

implicitly by the population and any individual agent can use only that part of the repository it embodies. It has been shown that social learning through imitation (sometimes called 'cultural evolution') can be beneficial by decreasing the learning time for individuals, particularly in cases where the required behavioural rules are difficult to acquire (Acerbi and Nolfi, 2007; Denaro and Parisi, 1996). Such implementations of social learning typically focus on a limited number of 'experienced' individuals instructing uninitiated individuals one by one and thus do not exploit the inherently parallel ink-stain effect present in the peer-to-peer knowledge exchange that we envisage. Similarly, in ensembles of learning classifier systems, social learning – termed 'rule-sharing' – has been shown to boost the learning speed (Bull et al., 2007) of the ensemble. Comparing such ensembles with a population of interacting, mortal agents is tenuous, however: the constituent parts of the ensembles are not considered separately, only the performance of the ensemble's aggregated behaviour is taken into account.

3.3.1 Energy and Agent Quality

As mentioned in section 3.1 and contrary to typical evolutionary algorithm or evolutionary robotics applications (Eiben and Smith, 2008; Nolfi and Floreano, 2000), the PAS we use as an example lacks a crisp optimisation criterion as well as a concrete task to be performed optimally. The agents survive whatever the environment throws at them or they do not—that's all there is. This also entails that there is no measure of fitness in this system: the only selection mechanism is –truly Darwinian– the struggle for life in the environment: environmental selection.

To gauge their relative quality, agents can, however, be compared in terms of their perceivable attributes such as age or energy level. Crucially, such comparisons cannot be performed by some central selection mechanism – as would be the case in traditional evolutionary algorithms – but by the individual agents themselves when they *autonomously* decide to mate, engage in social learning, or otherwise interact with another agent.

3.3.2 Social Learning in Detail

Social learning is implemented in a push model, where teachers volunteer knowledge pieces that the students then may accept. Alternatively, social learning can be implemented in a pull model, where agents request knowledge from other agents.

A combined model, where agents advertise that they believe that they have useful knowledge to share and other agents can then request that knowledge (similar to the plumage concept in (Smith et al., 2000)) could be implemented as well. Social learning as described here uses a measure of relative quality $R(a, b)$ (described below) that compares agents a and b in terms of energy and age, but could have used, for example, a reputation-based measure just as well.

Generally, this subsection describes the implementation of social learning within NEW TIES—alternative design choices could be made and implemented at every level described here. As mentioned above, however, some options are infeasible because of the anthropomorphic nature of agents in these experiments. For instance, agents have to be within range ('earshot,' if you will) to be able to communicate and hence engage in social learning.

Social learning is implemented in the following sequence for every agent at every time-step:

1. An agent chooses to initiate sending ('teaching') probabilistically ($p = 0.2$).

2. If it decides to send, the agent describes the trace through its DQT that led to the current action (e.g., "I'm moving because there is no food to pick up").

3. Of all the agents in range, the teacher then selects the one with the lowest energy as the 'student'.

4. When an agent receives a knowledge piece, it stochastically chooses to integrate ($p = 0.2$) or disregard it.

5. When an agent s incorporates a DQT path P it received from an agent t, agent s selects the most similar path P' in its own DQT according to the following criteria:

   (a) Percentage of matching tests

   (b) The number of tests P but not in P'

   (c) The number of tests in P' but not in P

   If the percentage of matching elements in P is 100%, the bias for the action that P results in is multiplied with the relative quality $R(t, s)$ (see below). Otherwise, the agent engages in a kind of dialectic: it inserts a bias node at the first point of divergence between P and P'. The remainder of P' is

inserted as one option at that node, a sub-tree corresponding to the non-matching entries in P is inserted as the alternative. The biases for the options are set proportionally to the relative quality $R(t, s)$. Figure 3.5 illustrates this procedure.

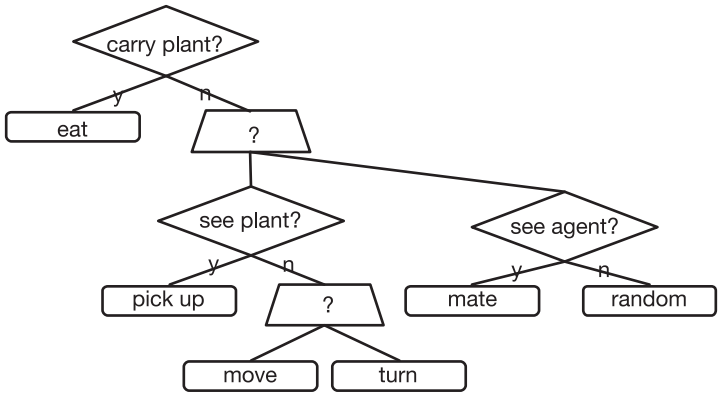


Figure 3.5 – The result of integrating the path `[not carry plant; see agent]` ⇒ `mate` into the DQT from Fig. 3.2.

As described above, this social learning implementation requires some measure of (relative) quality for agents to be able to assess the merit of received knowledge pieces when incorporating those pieces. To that end, an agent a can determine the relative quality $R(a, b)$ of another agent b from their relative ages A_a and A_b and energy levels E_a and E_b, respectively:

$$R(a, b) = 0.5 \cdot \left(\frac{A_a}{A_a + A_b} + \frac{E_a}{E_a + E_b}\right)$$

This measure ranges from 0, where agent b devastatingly outperforms agent a to 1, where the converse is true. If the agents have the same energy and are equally old, $R(a, b)$ equals 0.5. Note that this measure does not constitute an optimisation criterion as typically used in evolutionary algorithms: it does – without specifying any goal – allow for the comparison of the success of adaptation of individuals.

Social Learning as an Evolutionary Algorithm

(Smith et al., 2000) already showed that an agent-based knowledge exchange mechanism similar to social learning constitutes an evolutionary algorithm. Moreover, as pointed out in (Eiben and Smith, 2008), an evolutionary algorithm requires:

– Selection as a force to push quality;

– Variation operators to create the necessary diversity and thereby create novelty.

This implementation of social learning achieves the former of these at various levels. Firstly, ill-adapted individuals tend to die relatively quickly, and hence cannot further distribute their knowledge, while well-adapted individuals tend to survive and have ample opportunities to distribute their knowledge. The second level is that of student selection mentioned above: when an agent has to choose between potential recipients of a knowledge piece, it selects the one with the lowest energy. Finally, the integration mechanism uses the relative quality $R(a, b)$ to set the bias for already known or newly received knowledge.

Variation is provided by the knowledge integration mechanism, which can be seen as a guided adaptation of crossover such as commonly used in genetic programming. Although this suffices, individual learning and social learning dovetail very nicely in this respect (as well as because of the benefit that we expect from social learning providing a knowledge repository for individual learning): individual learning then plays the part of a mutation-like variation mechanism.

3.3.3 Experimental Set-up

In this section, we –or rather,the agents– revisit the poisonous food challenge described in section 3.2.1 where agents have to learn to avoid poisonous food and eat only healthy food.

To measure the efficacy of social learning as a mechanism for the proliferation of knowledge pieces through a population (i.e., for the population as a whole to adapt from individually learnt adaptations), we ran a series of experiments where the population consists of two kinds of agents: knowers and students. The knowers have pre-built controllers that allow them to tackle the poisonous food problem. The students have a partially randomly constructed controller—they know how to pick and eat plants (regardless of their being poisonous or not) and how to mate, but the rest of their DQT is constructed randomly. A varying proportion of the agents with pre-built controllers can send, but not receive social learning messages ('teachers'), while students both send and receive social learning messages. The remaining knowers do not engage in social learning in any way; they are only there to ensure that the environment contains the same amount of agents eating away at the wholesome plants across the experiments, so that the results are comparable.

Another difference between students and knowers is that the former can mate to produce offspring where the latter cannot. Note, that this does not – in these particular experiments – constitute evolution: there is no variation operator because it does not entail recombination, but cloning of either parent. Therefore, there is no evolution at play to disturb our measurements. Neither kind of agent can perform individual learning in these experiments.

This set-up serves as an idealised exemplar of a population where some agents – represented by the teachers – have discovered, through individual learning or otherwise, a particularly useful bit of knowledge: to eat only wholesome plants. Note, that these teachers play quite a different role from the 'experienced individuals' employed by (Acerbi and Nolfi, 2007; Denaro and Parisi, 1996): from the students' point of view, they are no different from any other agent they encounter. We ran the experiment with varying numbers of teachers to compare the rate at with which the population of students learns to differentiate between nutritious and poisonous food.

In our experiments, the agents can move in a 200×200 grid. There are initially 250 students and 100 knowers, of which 0, 1, 5 or 50 individuals are teachers. Agents can live well beyond the length of the experiments, so agents can only die of lack of energy. Each experiment was repeated 20 times. Poisonous plants drain 1.5 times the energy that wholesome plants yield, the environment is initialised with 16,000 plants of each type. Plants regrow practically immediately (within 2 time-steps), even if they've been picked, similar to food in SugarScape (Epstein and Axtell, 1996). Thus, there is always food (and poison) available and the ratio poisonous/wholesome plants remains more or less at the initial value of 0.5.

To quantify behaviour, we use the g measure introduced in Eq. 3.1 – the ratio between wholesome and poisonous plants eaten. We also employ a structural measure that actually detects the presence of the required knowledge. There are, of course, many different strategies that allow the agents to eat only wholesome plants—e.g., "only pick up wholesome plants and eat anything you carry", or "drop any poisonous plant and eat anything you still carry". In these experiments, however, we know exactly which knowledge piece we expect to find because it is the relevant trace through the handcrafted knowers' DQT: it's `[carry wholesome plant]` ⇒ `eat`. This allows us to identify, during a run, those students that have incorporated this rule by inspecting their DQTs. Thus, we can measure the incidence among the students of the appropriate knowledge piece.

Note, that the measurements we present here were taken only over the population of students.

3.3.4 Results

Figure 3.6 shows the development over time of $g(t)$ – averaged over 20 runs – for the students with 0, 1, 5 and 50 teachers. For reasons of legibility we omitted error bars; the 4 curves do differ considerably, although the standard deviation for 0 and 1 teacher is large, due to the fact that in many of these simulations, the students didn't eat at all.

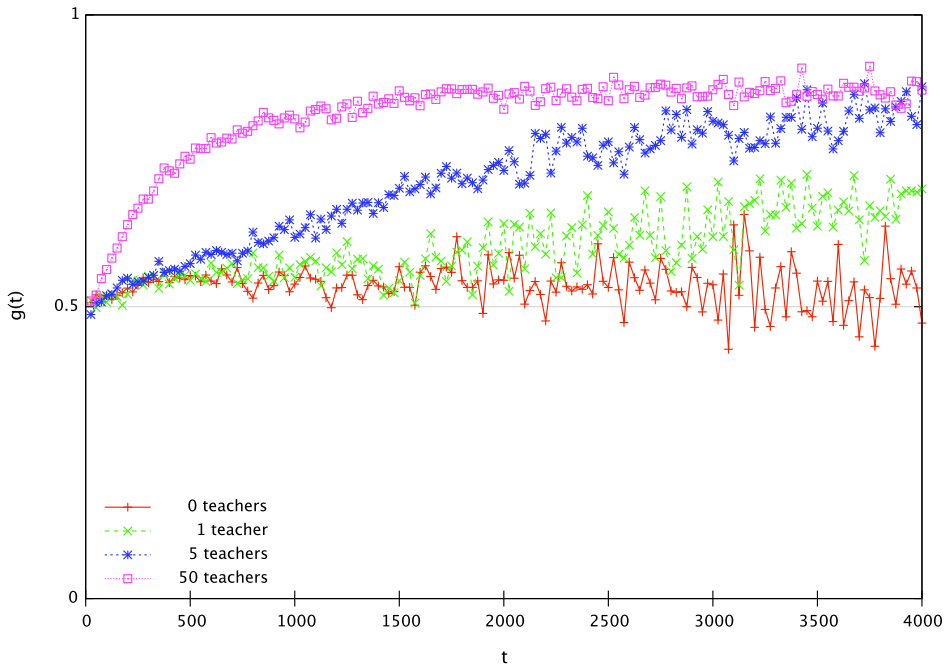


Figure 3.6 – Development over time of $g(t)$ –the ratio between wholesome and poisonous plants eaten– for the student population for different numbers of teachers.

As can be seen, $g(t)$ remains level just above 0.5 for 0 teachers – there is no learning at all – the slight improvement over fully random behaviour is due to environmental selection: agents that eat too much poisonous food simply die at a faster rate than agents that do not or less so, leaving a slightly better set of surviving agents. In the case with a single teacher, the performance of the students increases substantially: even from so small a seed, a knowledge repository can grow. For 5 and 50 teachers, the population behaviour improves rapidly until $g(t)$ reaches a plateau between 0.8 and 0.9—there is no substantial difference between

these experiments after that point. This seems to imply that in both cases the population of students becomes saturated – at least at a behavioural level – with the appropriate knowledge piece.

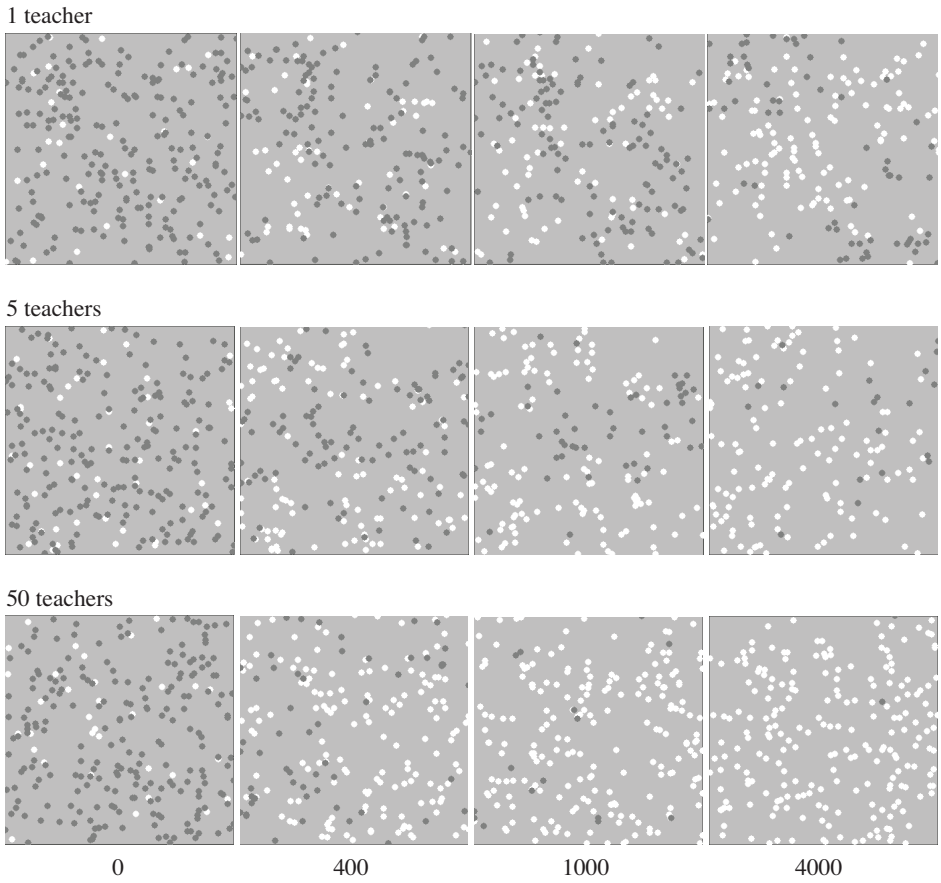


Figure 3.7 – Spread of knowledge pieces over the students for typical runs with 1, 5 and 50 teachers at timesteps 0, 400, 1000 and 4000.

Figure 3.7 shows a series of maps of the world displaying the incidence of the required knowledge piece (`[carry wholesome plant]` ⇒ `eat`) geographically. The three sequences of maps show the spread of knowledge over time for typical runs with 1, 5 and 50 teachers respectively. Students that contain the required knowledge show white, those that don't show dark grey. Teachers and knowers are not shown. Note the logarithmic time-scale.

Again, it is plain that, even with a single teacher to initiate dissemination, the decisive knowledge is spread through a significant part of the population—the population as a whole stores the knowledge effectively and robustly. As could be

expected, the knowledge becomes even more widespread for the experiments with 5 and 50 teachers.

While we have seen the behaviour for the student population reach similar levels for the experiments with 5 and 50 teachers, this is not the case for the incidence of the expected knowledge piece. With 50 teachers, practically all students have obtained this knowledge piece after 4000 time-steps, but in the 5 teachers case, a portion of the students remains unaware of this information at that time. Similarly, there is no appreciable difference between $g(t)$ at time-step 1000 and at time-step 4000 for the 50 teachers experiments, but there is a marked difference in incidence of the required knowledge piece. From this we can conclude that, after a certain level of prevalence has been achieved, further proliferation of the knowledge piece has no perceivable effect on population behaviour in terms of $g(t)$.

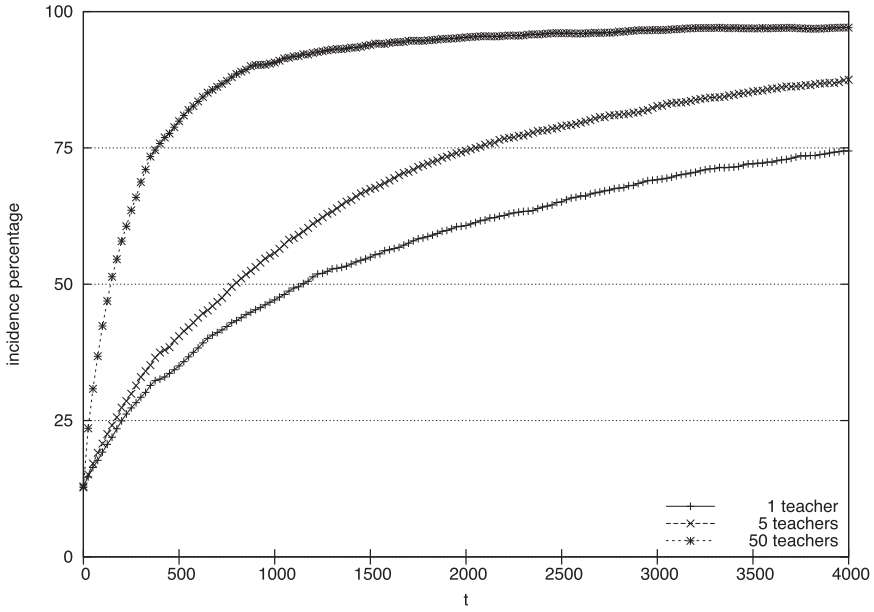


Figure 3.8 – Development over time of the percentage of students with the crucial knowledge piece.

Figure 3.8 shows how the percentage of students that have learned the requisite knowledge develops over time with 1, 5 and 50 teachers, respectively, averaged over 20 runs. Because the students spread the knowledge they receive, incidence grows almost exponentially as can be seen from the graph.

Note, that at time-step 0, a portion of the population does contain the knowledge as part of the randomly initialised tree while $g(t)$ for the runs without any teachers doesn't increase over time. This can be explained by the context in which

the knowledge piece may be present (i.e., as sub-clause in a more complex, possibly nonsensical rule) and by the fact that the action node's weights (as described in section 3.1.2) aren't sufficiently biased towards actually selecting the `eat` action.

3.3.5 Discussion

We asked ourselves the questions of whether social learning can provide a successful mechanism to spread knowledge pieces over a population, and is there a minimum requirement to enable the population to create a knowledge repository for otherwise volatile individually acquired knowledge.

The results of the poisonous food experiments clearly show that social learning does provide an efficient mechanism for the dissemination of knowledge pieces through a population of agents. Even from a single agent, the knowledge can spread over the majority of the population like an ink-stain on tissue paper. Within the framework of PAS in general and the implementation in NEW TIES in particular, this means that social learning is capable of allowing the population to form a knowledge repository for individually acquired knowledge so that such knowledge doesn't necessarily expire with the agent that discovered it.

3.4 Conclusion

We began the chapter by introducing a framework for adaptation in population-based adaptive systems (PAS), positioning and relating evolution, individual and social learning.

The examples in this chapter illustrate that there are many ways to set up adaptive behaviour in a PAS, be it individually or collectively. The experiments show that evolution, individual and social learning all provide powerful mechanisms for initiating and spreading adaptation. Combining mechanisms may further enhance the population's performance, as indicated by the experiment in section 3.3. There, we saw that social learning can provide an excellent method to share individually acquired adaptations among the population, allowing the whole population to benefit from an individual's experience and preventing valuable knowledge from being lost when an individual –be it a software agent or a robot– ceases to function.

We saw that social learning can constitute (part of) an evolutionary adaptive system. The 'telepathic' version described in section 3.3 can co-operate with in-

dividual learning; individual learning provides variation, while social learning implements recombination. Selection is done environmentally (poorly adapted individuals disappear) and/or in social learning.

Indiscriminate combination of adaptive mechanisms, however, carries a danger: it may lead to the emergence of unwanted interactions as shown in section 3.2. There, we saw that individual learning can counteract as well as promote evolution depending on the rewards the learning is based on: a case in point that illustrates how the goals of the adaptation mechanisms have to be in tune with each other. Therefore, careful consideration should be given to the interactions between the adaptive mechanisms when designing a PAS: one cannot simply design the mechanisms in splendid isolation.

Investigating these interactions, then, is one of the challenges that future PAS research will have to address to understand how adaptive mechanisms can be combined to enable truly autonomous robots, robots that can indeed learn control without human supervision.

If I have seen further it is only by standing on the shoulders of giants

Sir Isaac Newton

4
Look Ma, No Hands!

An Overview of On-line, On-board Evolutionary Robotics

Typical implementations of evolutionary robotics optimise robot controllers prior to proper deployment in the real world. This contrasts with the use of evolution for *on-line* adaptation where the robots autonomously optimise their own controllers while they go about their tasks.

We survey research that does employ on-line evolution to endow the robots with continuous and autonomous adaptivity. We distinguish three schemes in which on-line evolution can be implemented and classify existing work along these lines. We establish common issues in on-line evolutionary robotics and identify research in mainstream evolutionary computing that may provide solutions and suggest directions for future research.

This chapter was submitted as:

Evert Haasdijk, A.E. Eiben. Look Ma, No Hands! – An Overview of On-Line, On-Board Evolutionary Robotics. Submitted to *Swarm and Evolutionary Computation*, Elsevier.

4.1 Introduction

Imagine a collective of autonomous robots that find themselves in a dynamic environment that they (or rather, their designers) didn't know to expect. Obviously, they cannot rely on pre-defined behaviour determined by fixed controllers to cope robustly with the unknown challenges in such a scenario. Rather, the robots have to learn to cope with their environment and any changes in it, just as they have to learn to react to changes in their own bodies, for instance as a result of hardware failure or reconfiguration. An essential capability, therefore, of such robots is the ability to adapt their controllers – to learn – in the face of challenges they encounter in a hands-free manner, without supervision — human or otherwise.

This review is inspired by the vision of robots one day being able to adapt like this as so eloquently articulated by Nelson et al. (2009):

> *"Advanced autonomous robots may someday be required to negotiate environments and situations that their designers had not anticipated. The future designers of these robots may not have adequate expertise to provide appropriate control algorithms in the case that an unforeseen situation is encountered in a remote environment in which a robot cannot be accessed. It is not always practical or even possible to define every aspect of an autonomous robot's environment, or to give a tractable dynamical systems-level description of the task the robot is to perform. The robot must have the ability to learn control without human supervision."*

There are many approaches besides evolutionary algorithms that can provide on-line adaptation, and there has been substantial research into areas such as reinforcement learning. For the purposes of this overiew, however, we focus on evolutionary approaches and a comparison with these alternatives is outside the purview of this review.

We can distinguish between the design and operational stages of a robot's life. Prior to deployment, at design time, we can then see evolutionary algorithms as tools for the design of robots and their controllers, but after deployment, during a robot's operational phase, evolutionary algorithms become tools that provide adaptivity. To differentiate between the use of evolution in these two phases, we call it *off-line* and *on-line*, respectively.

We claim that the advanced robots in our vision must have the ability to adapt on-line in an autonomous manner so that they may cope with a number of challenges:

Unforeseen environment The environment where the robots operate may not be fully known during the design process. Therefore, the robot controllers at the time of deployment are only approximate solutions that need to be adapted to the environment as it is found at operational time.

Changing environment The environment may change to such an extent that the initial skill set of the robots is no longer adequate. Hence, controllers must adapt to changing situations.

Reality gap Even if the environment were known beforehand and constant during operational time, it is very likely that the design process uses approximations and simulations of real operational conditions. Hence, the robot controllers will have to be fine-tuned after deployment.

To provide such levels of autonomous adaptivity, an evolutionary algorithm must run on-board, without any external master computer to execute evolutionary operators or evaluations. Crucially, the robots' controllers change on the fly, as they go about their proper tasks: adaptation occurs *on-line*, during the operational period of the robots.

The overwhelming majority of evolutionary robotics research to date, however, involves *off-line* adaptation during the design stage preceding the operational period. Additionally, the evolutionary algorithm that develops the robot controllers actually doesn't run on the robots themselves at all, but it runs on a separate computer that centrally maintains the population of robot controllers (or rather, their genetic encodings) and performs selection and genetic operators. The robots themselves (be it real or simulated) only come into play to evaluate candidate controllers. Once the controllers are deployed on the actual robots to perform their tasks for real, the evolutionary process is terminated and the controllers are modified no more — at least not through evolution. Nolfi and Floreano (2000) provide ample illustration of this approach to evolutionary robotics. In these cases, evolution does not provide the on-line adaptive capabilities that we envisage.

Essentially, this off-line approach to evolutionary robotics is no different from other fields of evolutionary computation as we have known them since the 1960s

such as Evolution Strategies, Genetic Algorithms, Genetic Programming, Differential Evolution, to name but a few. Traditional evolutionary robotics and mainstream evolutionary algorithms share the centralised management of evolutionary operators, where would-be parents do not select mates and produce offspring autonomously, but are being selected by an 'oracle' (the outer loop of the main evolutionary algorithm) and undergo variation operators passively.

For the on-line adaptive capabilities without supervision as we envisage, this is not an appropriate scheme. For that, the adaptation mechanism – the evolutionary algorithm – must operate in a decentralised, autonomous fashion.

In section 4.2 we argue that this departure from centrally controlled evolution fundamentally changes the rules of the game and introduces specific issues that are not commonly considered in off-line approaches to evolutionary robotics. We also present a classification to categorise on-line, on-board evolution implementations based on how they delegate control of the evolutionary process.

Sections 4.3 to 4.5 present a survey of relevant research that considers decentralised autonomous approaches to evolution. We focus on evolution towards specific tasks with well-defined fitness function, for instance, tasks such as avoiding obstacles (Nordin and Banzhaf, 1997) or collecting objects (Watson et al., 2002). We do not consider machine learning approaches that do not employ evolutionary computation, for instance reinforcement learning.

We conclude the survey by suggesting directions for future research in section 4.7.

4.2 On-line, On-board Evolution

In the previous section we identified two critical requirements of employing evolutionary algorithms to provide adaptivity as we envisage: it is on-line and it is autonomous.

Instead of a pre-deployment phase where evolution is free to try anything as long as the end result meets the requirements, evolution now takes place *post*-deployment, implying less control over and higher stakes during the evaluations. Lower control because the designer of the system has no control over, possibly not even knowledge of the exact circumstances in which subsequent evaluations of controllers take place. Higher stakes because evaluating a poor controller doesn't

just waste development time, it also affects system performance directly. We will investigate the consequences of this requirement in section 4.6.

The requirement of an autonomous evolutionary process entails a departure from traditional evolutionary algorithms in two points in particular:

- There is no central authority external to the robots that decides which robot controllers reproduce and which ones are replaced;
- There is no omniscient presence who knows (let alone determines) the fitness values of all individuals.

Consequently, the robots gauge their own (and each other's) fitness themselves and it is they themselves who autonomously decide (based on their fitness information) when to mate and with whom. As noted by Eiben et al. (2007):

> *[T]he key element here is the locally executable selection. Crossover and mutation never involve many individuals, but selection in evolutionary algorithms usually requires a comparison among all individuals in the population.*

In other words, the operators that are specific for decentralised evolution (as opposed to a traditional, centrally controlled evolutionary algorithm) are the selection operators: mate selection instead of parent selection and a local survivor selection instead of a global replacement strategy. The variation operators (mutation and crossover) need not be specifically designed for on-line evolution, but should only match the given genetic representation. Note that these, too, should be executed locally, though.

There are examples of evolutionary algorithms that do away with external selection and modification of the individuals. Such algorithms have a very different look-and-feel than a run-of-the-mill genetic algorithm solving, say, the travelling salesman problem, but they are clearly evolutionary. Intuitively, they are a different kind of evolutionary system, where the essence is not optimisation in some abstract search space, but a population of actively reproducing agents that undergo selection in some (possibly virtual) environment.

Implementations of on-line, on-board evolutionary robotics can be distinguished by how control of the evolutionary algorithm (particularly of the selection operators, as noted above) is devolved away from a central authority and onto the robots themselves. This can be achieved by encapsulating the complete evolutionary algorithm in the robot, by distributing the selection operators over a collective of robots or by hybridising these two approaches:

The encapsulated approach Each robot has an evolutionary algorithm implemented on-board, maintaining a population of genotypes inside itself. The robots run these (possibly different) evolutionary algorithms locally and perform the fitness evaluations autonomously. This is typically done in a time-sharing system, where one member of the inner population is activated (i.e., decoded into a controller) at a time and is used for a while to gather feedback on its quality. Here, the iterative improvement of controllers is the result of the evolutionary algorithms running in parallel on the individual robots.

The distributed approach Each robot has a single genotype and is controlled by the corresponding phenotype. Robots can reproduce autonomously and asynchronously and create offspring controllers by recombination and/or mutation. Here, the iterative improvement of controllers is the result of the evolutionary process that emerges from the exchange of genetic information among the robots;

The hybrid approach The two previous approaches can be combined, resulting in a set-up akin to an island model as used in parallel genetic algorithms. In such a combined system, each robot contains a complete evolutionary algorithm on board, just as in the encapsulated case, but in addition exchanges genetic information with other robots as in the distributed case. Thus, each robot is an island, and the iterative improvement of controllers is the result of the evolutionary process that emerges from intra-island variation (i.e., within the population of the enclosed evolutionary algorithm in one robot) and inter-island migration (between two or more robots).

4.3 The Encapsulated Approach

Evolutionary algorithms that embrace the encapsulated approach may not seem materially different from 'regular' evolutionary algorithms because the whole process runs on a single robot, just as it would run inside a single computer. We will see, however, that there are particularities of on-line evolution; these will be elaborated on in section 4.6.

One practicality of (especially encapsulated) on-line evolution is *time-sharing*; obviously, only a single controller can be active at any one time on a single robot. Therefore, the encapsulated algorithm can only evaluate a single individual (which

defines a single controller) at a time. To evaluate multiple individuals (as any evolutionary process must) the algorithm has to try them in sequence, one after the other: a controller runs for a certain amount of time and the task performance over that period determines the controller's fitness. Then, the next controller is loaded and gets its chance to control the robot while fitness is measured, and so on. As a consequence, there can be no guarantee that two individuals are evaluated in the same or even similar circumstances; we will return to this in section 4.6.

To our knowledge, the oldest implementation of on-line evolution of robot controllers is that of Nordin and Banzhaf (1997). Nordin and Banzhaf implemented a genetic programming system to evolve machine code for a Khepera's micro-controller. They successfully evolved controllers for obstacle avoidance and for obstacle following, with the robot learning the first task in 40 to 60 minutes and the latter in around 30 minutes.

The initial implementation, for which the results are reported, used an external computer to run the actual evolutionary algorithm (and is therefore off-board, but still on-line), but the algorithm was subsequently ported to run on the Khepera's own micro-controller. No results of this on-board version are noted, but with battery life ranging from 40 to 60 minutes, successful evolution of controllers for both tasks is feasible.

Floreano et al. (2002) describe a fully on-board implementation of an algorithm to evolve spiking neural networks for obstacle avoidance in very limited hardware (an Alice robot with a PIC16F628 processor). While the paper focusses on the practicalities of implementing an evolving spiking neural net in the cramped conditions of a small robot's microprocessor, the evolutionary algorithm can serve as an illustration of two characteristics that are common in almost all encapsulated and hybrid implementations of on-line, on-board evolutionary algorithms. Firstly, the algorithm implements steady-state evolution: an individual is tested and possibly replaces an individual in the current population so that parents and offspring may exist side-by-side. In this case, the new individual replaces the worst in the population if it performs better. Secondly, the population on a single robot (consisting of only 6 individuals) is tiny compared to what is common in evolutionary computation; after all, it had to fit inside the robot's on-board memory.

Walker et al. (2006) show one of the benefits of on-line, on-board evolution in addition to hands-free adaptivity: it can help overcome the 'reality gap' where a simulator irons out the wrinkles and warts of reality. Often, when robot con-

trollers are developed using a simulator (typically cheaper and faster than using real robots), the controllers in reality perform worse than might be expected due to infidelities in the simulation and particularities of individual hardware (see Brooks, 1992). Walker et al. take a two-staged approach: they first employ an off-line, centralised evolutionary algorithm to develop controllers in simulation (the 'training phase'). They then transfer the resulting controllers onto a real robot, and implement an on-line, on-board evolutionary algorithm that further refines the controller and adapts it to a changing environment. In their experiments, the robot has to avoid randomly placed obstacles while moving towards a goal. The environment changes by moving the obstacles and by varying the number of obstacles in the arena. The dynamics of the environment highlight the hands-free adaptivity that on-line, on-board evolution enables. Walker et al.'s on-line algorithm has a single champion genome that serves as parent to a challenger; the challenger replaces its parent if it performs better. To this fairly common arrangement they add a buffer memory: a second child can replace the challenger (but not the parent directly) if it outperforms it. Every iteration, the second child is replaced with a newly mutated version of the parent, so the first child is the current best challenger. This two-tiered approach was designed to overcome the problem mentioned above: as a result of sequential evaluation, the circumstances of evaluation could differ significantly from one individual to the next and 'a poor chromosome could perform uncharacteristically well and be rewarded and vice-versa.' The population of the on-line algorithm is (again) tiny: only three individuals are considered, in all. Walker et al. mention that this carries the benefit of promoting rapid adaptation to changes in the environment: 'the smaller the population size, the faster each generation of chromosomes is evaluated and the sooner the effects of evolution manifest.'

Haroun Mahdavi and Bentley (2006) describe experiments involving encapsulated on-line evolution in robots that use shape memory alloy actuators: metal filaments that change shape when a small current is applied. In one set of experiments, Haroun Mahdavi and Bentley use on-line evolution to let a snake-shaped robot learn to move forward. These experiments don't actually exhibit the level of autonomy suggested by on-line, on-board evolution because the fitness was not measured by the robot itself but by the experimenters (and therefore evolution took place, at least in part, off-board), but they, nonetheless, provide a striking example of one of the benefits of on-line adaptation.

During one experiment, one of the filaments broke while the robot was evolving a gait. Normally, this would spell disaster and mean that the robot must be fixed and the experiment restarted. Haroun Mahdavi and Bentley, however, realised that this was an excellent opportunity to see the controller adapt to the new circumstances and continued the experiment with one broken actuator. In short order, the robot adapted its gait to move with one less actuator, showing how on-line evolution can help robots cope with hardware failure. Similar robustness under on-line, on-board adaptation was reported by Christensen et al. (2010), who purposely introduced hardware faults (for instance, failing modules in a modular robot body) during some experiments with an on-line, on-board stochastic adaptation method akin to Newton-Raphson minimisation methods.

Bredeche et al. (2009) implement the $\mu + 1$ on-line evolutionary algorithm, later elaborated by Karafotias et al. (2011), where a robot maintains a population of μ individuals on-board. In this algorithm, some evaluation cycles are used not to generate and test new controllers, but to re-assess controllers already in the population. This should reduce the impact of the fact that two individuals can be evaluated in very dissimilar circumstances.

4.4 The Distributed Approach

In their seminal papers, Ficici et al. (1999) (elaborated in Watson et al. (2002)) coin the phrase *Embodied Evolution* for a system where "a population of physical robots [...] autonomously reproduce with one another while situated in their task environment." In other words, the controllers evolve on-line ("in their task environment") through the physical robots exchanging genetic material with one another. This constitutes a groundbreaking example of the *distributed* approach: the evolutionary process is not embedded in individual robots or in some external overseeing module, but it emerges from the interactions between the robots.

Watson et al. introduce a fully autonomous scheme where the robots broadcast (mutated) genes on local-range communication channels at a rate proportional to their fitness (Probabilistic Gene Transfer Algorithm, PGTA). Also, robots resist 'infection' with genes broadcast by other proportionally to their fitness. Robots incorporate received genes into their own genome and so immediately update the controller and continually evaluate its performance. In this scheme, there is no central authority or global view of the population, but there is, in fact, one item of

global information: the maximum possible fitness which is used to determine the broadcast and resistance rates. This seems to limit the applicability of PGTA, but it would seem reasonably straightforward to propagate at least the best achieved fitness through a gossiping-like algorithm as described in Jelasity et al. (2005) and so avoid the need for a pre-specified maximum fitness. Wickramasinghe et al. (2007), described below provide an example of such an approach. The research shows that the robots learn to tackle a phototaxis task very efficiently, even outperforming a Braitenberg-based benchmark controller.

Elfwing et al. (2005) show another example of a distributed algorithm where the robots have to learn to harvest batteries to maintain their energy level. The robots run controllers sequentially; controllers 'die' after a certain time expires or if the energy runs out. When a controller completes a full lifetime (i.e., they do not run out of energy because they successfully harvest batteries), a child is created from the tested controller and one of the genomes that the robot received from another robot (the robots have to be physically close to exchange genomes). If a controller's lifetime is cut short because the energy runs out, a child from an earlier controller that did survive is selected. Elfwing et al. compare this scheme with a standard centralised algorithm. The results show that the centralised scheme performs better, but not significantly so, due to the large variance in performance. The authors note that this may also be a result of the number of batteries captured being an explicit measure for the centralised case, while it is only indirectly rewarded in their on-line scheme.

Wischmann et al. (2007) investigate the interplay between embodied evolution and individual learning by introducing a *maturation period* during which no mating or replacement can occur. During this maturation period, part of the artificial neural net that controls a robot adapts through backpropagation; this allows the controllers to adapt using individual learning before feeling any selective pressure. The evolutionary algorithm itself is a slightly modified version of Watson et al.'s PGTA. In their experiments, a population of prey agents evolves and learns to move around while avoiding non-evolving (but learning through backpropagation) predators. The results show that whether or not learning may facilitate evolution depends on the right timing of individual maturation and that the choice of maturation time significantly influences the rate of evolutionary adaptation. Purely in terms of distributed on-line, on-board evolutionary robotics, this research does not add much to Watson et al. (2002) other than offering another

successful application of Embodied Evolution. It is, however, noteworthy because of its focus on the interplay between evolution and learning.

Simões and Dimond (2001) implement a panmictic (fully connected) distributed evolutionary algorithm where the robots use radio to transmit their fitnesses; the best individuals are selected and recombined to create new controllers with which to reprogram the entire population. The algorithm is synchronised through internal timers on the robots; once a minute, all robots broadcast a 'mating call' over the radio containing their identification, fitness values, and chromosomes. Subsequently, the best individuals are selected to procreate and the worst robots are reprogrammed with the results. How and by what agency the selection takes place remains somewhat unclear. Simões and Dimond's experiments show that this algorithm is capable of evolving controllers that efficiently solve the task of collision-free navigation: to move as fast as possible while avoiding obstacles. This implementation requires that each robot be connected to all other robots (by radio), which might imply issues with scalability for large populations or for populations spread over a large area, where not all robots are in range of each other.

Nehmzow (2002) describes a similar system where the population consists of two robots that regularly suspend their normal tasks to switch to a mating mode. In mating mode, the robots physically seek each other out by homing on each other's infrared emissions to exchange fitness and genomes. The weaker of the two robots recombines the two genomes and replaces its current controller with the result, while the stronger mutates its current genome either randomly or by taking a single bit from a cached 'best-so-far' genome. Subsequently, both robots resume their tasks to evaluate the new genomes. Just as that of Simões and Dimond (2001), this is a synchronised scheme where the robots suspend 'regular' behaviour after a fixed amount of time to reproduce. The robots successfully learn to tackle phototaxis and obstacle avoidance tasks in this research.

As noted in eminent books such as those by Dawkins (1976) and Richerson and Boyd (2005), social interactions also result in processes of development of common knowledge or culture that can be viewed as evolutionary. Correspondingly, some social learning approaches in agent-based systems can also be seen as evolutionary. For these cases, as Kendall and Su (2007) put it, "evolution occurs through the storage and dissemination of information and knowledge by means of social learning". Typically, these approaches give rise to evolutionary schemes that we would classify as distributed.

Smith et al. (2000), for instance, describe a social learning algorithm where agents invite potential partners by sending 'plumage' objects containing 'sperm' (the genome) at fixed intervals. When an agent has received five of these objects, the best is chosen (note that this boils down to tournament selection with tournament size five) and the agent's own genetic material is combined with the sperm to create an 'egg' that replaces the current agent. Smith et al. note that the effect of certain design decisions in embodied evolution can be unexpected, even counterintuitive, for practitioners of traditional evolutionary algorithms. For instance, adding elitism (agents only accept plumage objects that have higher fitness than they themselves) could be expected to lead to faster convergence, but, in terms of wall-clock time, the reverse is true, probably due to added communication overhead because the agents have to receive more genomes before they find five that perform better than their own.

Vogt and Haasdijk (2010) describe a system where agents continually broadcast a partial description of their controller (implemented as a decision tree) by explaining what action they are taking and how they selected that action. Nearby agents that receive these messages sometimes (depending on proclivity and relative quality) incorporate the description in their own controller. In the context of the well-known poisonous plant task, Vogt and Haasdijk show that this scheme leads to rapid dissemination of useful strategies throughout the population of agents and that a population with social learning enabled provides a reliable knowledge repository.

At first glance, *Cultural Algorithms* (Reynolds, 1994) ought to be a prime candidate for an overview such as this, but the system for which Reynolds coined this phrase is centralised: it requires a centralised knowledge repository in which commonly available useful knowledge bits are stored and shared. The knowledge bits are distributed through *acceptance* and *influence* functions that implement selection based on a global view of the population. Obviously, such centralisation goes against our requirement of autonomous adaptation. Similarly, Kendall and Su's imperfect evolutionary system (Kendall and Su, 2007) describes a scheme with a centralised knowledge repository.

4.5 The Hybrid Approach

With the hybrid approach, the robots each implement a complete encapsulated evolutionary algorithm and exchange individuals with each other, just as those in distributed approaches. This leads to an architecture very similar to the island-based model in parallel evolutionary algorithms. Alba and Tomassini (2002) investigate a number of issues from the perspective of parallel evolutionary algorithms; they differentiate between panmixia, island-based and structured models (and various hybrids). Apart from the speed increase due to parallel evaluation, they note two benefits of island-based and cellular variants: "better sampling of the search space and improve the numerical as well as runtime behavio[u]r of the basic algorithm in many cases". For the island model, high diversity and species formation are well-reported features.

Perez et al. (2008) provide a case study of a hybrid implementation in collision-free movement. The robots run an encapsulated genetic programming algorithm and exchange parts of individuals (subtrees) asynchronously. In Watson et al.'s experiments (Watson et al., 2002), the rate of transfer of genetic material depends on the fitness level; in contrast, Perez et al.'s Distributed Genetic Programming system broadcasts genetic material at a fixed rate, regardless of utility, but with the utility attached. To determine whether or not to incorporate received genetic material, the utility associated with a received subtree (that of the individual it was part of) is compared to that of the worst local individual; if the associated utility is higher, the subtree is incorporated into the worst local individual. Perez et al.'s results show that all robots (five simulated Khepera units) learn to move around the arena without colliding into walls, obstacle for each other. Also, transferring genetic material in this manner between robots helps to increase the variability of the local population in each robot.

Usui and Arita (2003) implement an evolutionary algorithm that runs in a traditional manner within the robot itself. Locally created new individuals and individuals received from other robots – very similar to migration in island-based parallel evolutionary algorithms – are queued for evaluation, after which they are placed into the pool of the local algorithm, replacing the worst individual (unless they themselves turn out to be worse). In Usui and Arita's experiments, six Khepera robots are placed in an arena where they have to learn to move around without

hitting obstacles. Experimental results show that migration enhances performance when compared to the purely encapsulated case.

4.6 Considerations in On-line, On-board Evolutionary Robotics

From the work outlined above, we identify a number of issues in on-line, on-board evolutionary robotics that normally do not feature prominently in regular evolutionary computation. Some of these considerations, such as parameter control, have been investigated in other evolutionary computation research, while others, for instance the need for good average performance, have, to our knowledge, not yet benefitted from such attention. Let us take a closer look at these issues.

4.6.1 Actual performance matters

In typical applications of evolutionary algorithms, the be-all and end-all is a champion individual that is as good as possible: the best performing individual at termination has to be as close to optimal as we can get. The performance of the remainder of the population is, in the end, of no consequence as they will be discarded as the champion is deployed; their only reason for existence is to guide evolution's search process to the pinnacle that is the best individual in the population.

Things are very different for on-line evolution as we envisage here: remember that controllers evolve as the robots go about their tasks and so the robots' performance is determined by the quality of *all* individuals that they evaluate, not that of any single individual controller alone.

When a robot evaluates poor controllers, that robot's actual performance will be inadequate, no matter how good the best known individuals as archived in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution (even if it is not the best) and search prudently: it must display an acceptable level of performance throughout the continuing search. Balancing this need for rapid convergence and more or less stable performance with the variation that evolution requires is an especially challenging task when using on-line evolution, similar to the considerations concerning the trade-off between exploration and exploitation in reinforcement learning.

Endowing robots with a self-modelling capabilities, for instance as described by Bongard et al. (2006) may alleviate this issue: the robots could use their self-model in an on-board simulator as a surrogate model for an approximate assessment of candidate controllers before actually deploying them and truly assessing them. The increasing capabilities of robotic hardware (small, low-power processors in particular) make this an increasingly feasible and attractive option.

4.6.2 Evaluation in vivo

In the end, fitness must be evaluated *in vivo*: the quality of any given controller is determined by actually using that controller in a robot as it goes about its tasks. The real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances: any controller's evaluation will start wherever and in whatever state the previous evaluation left the robot. The very dissimilar evaluation conditions caused by one (possibly very poor) individual setting the scene for the evaluation of another individual result in very noisy fitness assessments. As Nordin and Banzhaf (1997) note:

> *Each individual is thus tested against a different real-time situation leading to a unique fitness case. This results in "unfair" comparison where individuals have to navigate in situations with very different possible outcomes. However, our experiments show that over time averaging tendencies of this learning method will even out the random effects of probabilistic sampling and a set of good solutions will survive.*

When generating new individuals, Nordin and Banzhaf select two parents in a four-way tournament, replacing the two losers with the offspring of the two best-performing individuals. To determine the tournament outcome, every individual in the tournament is evaluated, regardless of any previous evaluations. This implies that, to stay in the population, individuals have to perform well time and again, or they will be replaced – these re-evaluations provide the averaging effect Nordin and Banzhaf claim.

Bredeche et al. (2009) suggest an explicit re-evaluation mechanism where evaluation cycles are sometimes (the frequency is a parameter) used to re-assess existing solutions and update that individual fitness with a moving average.

Walker et al. (2006) also realised that 'a poor chromosome could perform uncharacteristically well and be rewarded and vice-versa.' They propose a two-tiered replacement strategy to overcome this problem.

Another consequence of evaluating performance on-the-fly in real-life is that the robot has to decide how long to evaluate individuals for: too long, and adaptation moves at too slow a pace, too fast and the fitness assessments become too noisy. Haasdijk et al. (2010) showed that this introduces a parameter that has great influence on robot performance.

Finally, all the information for the fitness assessment must be accessible through the sensors that the robots have: in terms of the 'fitness space' introduced by Nolfi and Floreano (2000), fitness assessment must be performed internally. This precludes outside agents (like experimenters) that monitor and measure robot performance.

Research into regular evolutionary computation in noisy and/or dynamic environments such as that by Beyer (2000) or Yang et al. (2007) provides some guidance on dealing with similar issues.

4.6.3 Parameter control and/or robust parameter settings

It is well known that the performance of evolutionary algorithms depends heavily on their parameter settings (Lobo et al., 2007). In particular, the optimal parameter values for an evolutionary algorithm may differ from problem to problem. The common practice in traditional evolutionary computing is to seek good parameter values by parameter tuning, performed by the experimenter before the 'real' run of a given algorithm.

Unfortunately, this is not an option in scenarios that we consider here, because the robots have to adapt to operational circumstances – without human intervention – that are unknown beforehand and/or dynamic. Such unknown, possibly dynamic circumstances imply that the evolutionary mechanism through which the controllers adapt must be somehow (re)tuned to be able to deal with these (new) circumstances. Christensen et al. (2010) present an interesting example demonstrating this problem: running experiments evolving modular controllers for different multi-module body shapes, they noted that the evolutionary algorithm successfully developed gaits in both cases, but only *after recalibrating the algorithm's learning rate parameter*.

Obviously, (re)tuning the evolutionary algorithm parameters in the traditional manner is impossible in the hands-free applications of on-line evolutionary robotics. Such hand-free adaptation requires evolutionary algorithms that are either capable of calibrating themselves on the fly or use robust parameter settings that work well under (almost) all circumstances. A combination is also possible: some parameters can have a robust value, while others can undergo permanent recalibration. To meet this challenge, knowledge in traditional evolutionary computing can be utilised. The relevant sub-areas here are that of evolutionary algorithm parameter control and parameter tuning aiming at robust settings. The need for evolutionary algorithms with good parameter control mechanisms was noticed as early as the 1990s, but most progress has been achieved in recent years. The main approaches are based on the idea of adapting (selecting) evolutionary operators or adapting evolutionary algorithm parameters (Eiben et al., 1999b; Fialho et al., 2008; Kramer, 2010; Maturana et al., 2010; Montero and Riff, 2011). There is less readily available knowledge about parameter tuning specifically to obtain robust settings in the existing evolutionary computation literature. In fact, there are few generic parameter tuning methods and these are typically used to achieve top algorithm performance on a narrow range of problems (Eiben and Smit, 2011). Robust settings, although they were a popular subject in the early days of evolutionary computing (Goldberg, 1989), are rarely discussed these days in mainstream evolutionary computing or in evolutionary robotics.

4.6.4 Situatedness

Distributed (and, by extension, hybrid) implementations of on-line, on-board evolution share – sometimes implicitly – the concept of a neighbourhood from which partners are selected. This neighbourhood can be physical and exist of other robots that happen to be within communication range or it can be in terms of a social network across the population using long-distance communication. This prompted Schut et al. (2009) to label these kinds of algorithm as 'situated' because this situates the robots (or agents) vis a vis each other. In Wickramasinghe et al. (2007), for instance, the neigbourhood is fluid as a result of the dynamics of peer-to-peer networks. The gossiping algorithm in fact defines a dynamic overlay network that defines which nodes are neighbours. Nehmzow (2002) and Simões and Dimond (2001) implement panmixia: in the spirit of a global village, all the robots (in fact,

only two in Nehmzow's case) are interconnected and the neighbourhood extends to all robots. Others, like Watson et al. (2002), implicitly define a dynamic neighbourhood that consists of those agents (robots) that meet – that is, that come within communication range of each other – by chance.

Partners are selected by sampling from this neighbourhood and a new individual is created by combining the partners' genomes. For virtual agents, for instance in Wickramasinghe et al. (2007), a new agent may be constructed ad libitum to evaluate this new individual, but it is difficult to envisage how to achieve this in the case of robots – although Schwarzer (2008) achieve something along these lines by having robots 'die' and then wait to be reprogrammed with a new individual. In most cases, however, an existing robot has to replace its controller with the new individual to evaluate it – just as robots evaluate consecutive controllers in the time-sharing scheme described above.

The agent that replaces its controller can be – and in robotics, usually is – one of the parents, for instance in Smith et al. (2000), but it can also be selected in turn from the neighbourhood as in Schwarzer (2008).

Selecting parents and survivors within a (local) neighbourhood is a recurring theme in spatially structured or *cellular* evolutionary algorithms (Alba and Dorronsoro, 2008). Here, the elements that make up the population are located in a (possibly virtual) grid, often with the nodes of this grid located on different CPUs. To minimise communication overhead, selection schemes that limit communication to relatively small neighbourhoods in the grid are preferable. Distributed on-line, on-board evolution in robotics, with chance encounters providing the sampling mechanism, can be seen as equivalent to cellular evolutionary algorithms with continuous random rewiring of the grid connections. Therefore, it is useful to consider some research into neighbourhood selection schemes from this field.

In Gorges–Schleuter's investigation of spatially structured evolution strategies (Gorges–Schleuter, 1998), experiments show that the spread of knowledge through the network is faster when connectivity extends in multiple directions (a torus topology outperforms a ring, for instance) and is fastest for a panmictic topology. This is deemed due to increasing selection pressure and therefore "coupled with a decrease of the loss of variability of the gene pool". As in all cellular evolutionary algorithms, the potential mates are the nodes in each other's neighbourhood (one might say that this is the defining attribute of structured evolutionary algo-

rithms). Parents are either both randomly selected from the neighbourhood (local selection) or one is randomly selected and the other is the central node in a neighbourhood (centric selection); centric parent selection outperforms local selection in all experiments in this study.

De Jong and Sarma (1995) analyse three different mechanisms for local selection in a spatially structured evolutionary algorithm. They find that local ranking as well as local binary tournament selection (providing constant selection pressure independent of actual fitness value) outperform local proportional selection, which is sensitive to the actual fitness value. In their research, a neighbourhood size of around nine seems to be optimal. Based on their experiments, De Jong and Sarma suggest to combine local tournament selection on small neighbourhoods with an elitist survivor strategy. Analysing selective pressure, they conclude that "higher variance is generally more strongly correlated with poor search performance when small population sizes are involved", so one should either decrease selection variance or increase population size if this is an issue. As we have seen, the latter is not always an option in on-line, on-board evolutionary robotics.

Eklund (2004) tests various cellular evolutionary algorithm variations, comparing grid topologies and neighbourhood shapes. He investigates various selection schemes: binary tournament, roulette, ranking, fitness uniform, best and random selection to confirm De Jong and Sarma's finding that local elitism is required for good performance. In terms of situatedness, the most interesting conclusion is that larger neighbourhoods lead to faster convergence. Fitness uniform selection is reported to perform well, but note that this performs poorly vis a vis actual fitness (Hutter, 2002), so it would seem to be at odds with requiring good performance across the population.

Wickramasinghe et al. (2007) show that the gossiping algorithm (Jelasity et al., 2005) can be employed to compare an agent's fitness to the population average without central calculation, even in the face of a randomly rewiring grid. Laredo et al. (2010) also employ the gossiping algorithm to disseminate genetic material through the population.

4.7 Directions For Future Research

The considerations listed in the previous section identify the need for further research in the field of on-line, on-board evolutionary robotics. This research may

be novel and chart previously unknown ground, or it may draw on established practices in other evolutionary computing disciplines.

In the evolutionary computing literature, we do not know of much work to draw on when it comes to developing evolutionary algorithms that perform well across the population instead of focussing on finding the best possible champion solution only. Research in the field of reinforcement learning, where the trade-off between exploration and exploitation is a similar issue may be of benefit here. Also, as mentioned, providing self-modelling capabilities could provide surrogate models, allowing for pre-selection stages based on internal simulations; work like that of Bongard et al. (2006) provides insight into how this could be achieved.

Sequentially testing controllers gives rise to unequal circumstances in which controllers are assessed. Further research is required to fully understand the implications of such noisy or 'unfair' comparisons. Research into evolutionary computing in noisy or dynamic environments may provide inspiration for this issue (Beyer, 2000; Yang et al., 2007).

The area of parameter control is rapidly developing in mainstream evolutionary computing, where it is mostly applied to (static) optimisation problems. Research is needed to investigate which (if any) of the control techniques developed in this area are applicable in on-line, on-board evolutionary robotics. A very recent development is that of solid tuning algorithms. These may help discover robust parameter settings that allow algorithms to tackle a wide range of circumstances without the need for recalibration. However, only one implementation we know of is designed specifically to find robust parameter values across multiple problems (Smit and Eiben, 2011). Little is known about the relevance of typical evolutionary algorithm parameters in on-line evolution; this requires further study.

As can be seen from subsection 4.6.4, there is a substantial body of work to draw on when it comes to the design of overlay networks in which multiple robots can exchange genetic material. The most important fields to consider are those concerning parallel and cellular evolutionary algorithms, but interesting work can also be found concerning evolutionary agent-based systems in artificial life.

To stop worrying about it will require worrying about it a lot at first

Ken Arnold

5 Growing Pains

Developing an Encapsulated Algorithm

The development of the $(\mu + 1)$ ON-LINE algorithm was undertaken to provide continuous on-line adaptivity to the robots under development in the SYMBRION project. Originally, these robots were to be equipped with Cortex M3 processors and very limited amounts of memory (256kb). The first implementation of $(\mu + 1)$ ON-LINE was therefore designed specifically to fit in these confines. Among other things, this led to a population size of only one ($\mu = 1$).

This small population size increased the susceptibility to noisy or 'unfair', in Nordin and Banzhaf's words, evaluations, because the single individual might easily be replaced by an individual that was lucky enough to be evaluated in a particularly easy circumstances. That is bad enough, but one may hope that a robust search process can recover from such setbacks. However, there is a more sinister and crippling consequence of these unrealistic high appraisals: such a lucky individual would be very hard to replace by other candidate solutions that, although inherently better, are less fortunate in the circumstances of their evaluations. To address this issue, $(\mu + 1)$ ON-LINE introduced the notion of re-evaluation: every once in a while (how often exactly is determined by the re-evaluation rate

parameter), an evaluation cycle is spent not to assess a newly created candidate, but to re-assess an individual from the current population.

Section 5.1 contains the first published paper on $(\mu + 1)$ ON-LINE and focusses on some of the intricacies of its re-evaluation scheme. Note, that in this paper, we used the term 'enclosed' for on-line evolutionary systems that we have meanwhile dubbed 'encapsulated.' Our initial $(\mu + 1)$ ON-LINE efforts also implemented a control heuristic for one of the algorithm's parameters that we expected to be influential: the mutation step-size, σ.

The second publication, which makes up section 5.2, took this a step further and investigated some of $(\mu + 1)$ ON-LINE's parameters in more detail, including alternative, more common schemes for σ adaptation.

Section 5.3, published at GECCO 2011, introduces *racing* to cut short evaluations that seem not to be worthwhile. This has two beneficial effects: it allows the algorithm to focus evaluations on promising candidate solutions and it reduces the time the robots spend performing badly; the latter is an important improvement when we take the requirement of good overall performance of the robots in an on-line learning setting. More important, though, is that we see racing as a first step towards or at least an inspiration for a control scheme for the evaluation period length τ. Further research in this area is under way.

5.1 On-line, On-board Evolution of Robot Controllers

This section reports on a feasibility study into the evolution of robot controllers *during* the actual operation of robots (on-line), using only the computational resources within the robots themselves (on-board). We identify the main challenges that these restrictions imply and propose mechanisms to handle them. The resulting algorithm is evaluated in a hybrid system, using the actual robots' processors interfaced with a simulator that represents the environment. The results show that the proposed algorithm is indeed feasible and the particular problems we encountered during this study give hints for further research.

5.1.1 Background and Introduction

Evolutionary computing has proved a powerful technology for developing robot controllers (Floreano et al., 2008) and has resulted in the establishment of Evolutionary Robotics. The overwhelming majority of evolutionary robotics applications use an off-line flavour of evolution. In these cases an evolutionary algorithm is used to optimise the robot controllers *before* the robots start their actual operation. This process may rely on real-life fitness evaluations or on a simulation-based assessment of controller quality, but in all cases the evolutionary algorithm is executed on one or more computer(s) distinct from the robots. Once the development process has terminated, the controllers are deployed on real robots and remain fixed while the robots go about their given tasks. Thus, during the operational period of the robots, the controllers do not adapt anymore (or at least, not by evolutionary operators (Nolfi and Parisi, 1993; Nolfi et al., 1994; Urzelai and Floreano, 2001)).

The present study was undertaken as part of the SYMBRION project that explicitly aims at using evolution on-line. That is, the evolutionary algorithm is required to adapt the robot controllers *during* the actual operation period of the robots. Such

---

Section 5.1 was published as:

Nicolas Bredeche, Evert Haasdijk and A.E. Eiben (2009). On-line, On-board Evolution of Robot Controllers. In Pierre Collet et al., *Artificial Evolution, 9th International Conference, Evolution Artificielle, EA, 2009, Strasbourg, France, October 26-28, 2009*, Pages 110–121, Springer Berlin / Heidelberg.

---

a switch from (off-line) optimisation to pervasive adaptation offers advantages in cases where the environment is changing and/or it is impossible to optimize the robots for circumstances in which they will operate (for instance, because they are not known well enough in advance). One of the premises of the SYMBRION project is the presence of a large group of robots that form a changing "social environment" for each other, which in turn, necessitates on-line adaptation again. All in all, we aim at a system that is decentralised, on-board, without any master computer that executes the evolutionary operators, and fully autonomous, with no human intervention. These requirements imply two major restrictions:

1. Fitness must be evaluated *in vivo*, i.e., the quality of any given controller is determined by actually using that controller in a robot as it goes about its tasks.

2. All necessary computation must be performed by the robots themselves, implying limited processing power and storage capacity.

The real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances; for instance, the nth controller will start at the final location of the $(n-1)$th one. This leads to very dissimilar evaluation conditions and ultimately to very noisy fitness evaluations. The limited processor power and storage capacity implies that we must use a "lightweight" evolutionary algorithm, with a small population per robot. Obviously, this could limit the exploratory power of the evolutionary algorithm , with a high risk of premature convergence at a local optimum. Taking these considerations into account, we formulate the following research objectives:

1. Provide an evolutionary mechanism that can cope with noisy fitness evaluations.

2. Provide an evolutionary mechanism that can perform balanced local and global search even with very small populations.

Related work on the on-line, on-board evolution of robot controllers can be roughly divided into two categories:

The distributed embodied evolution approach. Each robot carries one genotype and is controlled by the corresponding phenotype. Robots can reproduce autonomously and asynchronously and create offspring controllers by recombination and/or mutation. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary process that emerges from the exchange of genetic information among the robots. See Watson et al. (2002) for an example of this approach.

The enclosed embodied evolution approach. Each robot has an evolutionary algorithm implemented on-board, maintaining a population of controllers inside itself. The robots run these (possibly different) evolutionary algorithm s locally and perform the fitness evaluations autonomously. This is typically done in a time-sharing system, where one member of the inner population is activated (i.e., decoded into a controller) at a time and is used for a while to gather feedback on its quality. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary algorithm s running in parallel on the individual robots. Walker et al. (2006), for example, take this approach.

Note, that both approaches inherently work with a heterogeneous population of robot controllers. The two approaches can also be combined, and often are, resulting in a setup akin to an island model as used in parallel genetic algorithms. In such a combined system, there are two ways of mixing genetic information: intra-island variation (i.e., within the "population" of the enclosed evolutionary algorithm in one robot) and inter-island migration (between two, or more, robots). Nehmzow (2002); Usui and Arita (2003); Wischmann et al. (2007); Perez et al. (2008); Elfwing et al. (2005) provide examples of this hybrid approach.

The work presented here falls in the second category, i.e. the enclosed approach, explicitly aiming at online adaptation for a single robot.

5.1.2 The (1+1)-online Evolutionary Algorithm

We propose an evolutionary algorithm based on the classical (1+1) Evolution Strategy (Schwefel, 1981). In our experiments, the genome consists of the weights in an artificial neural networks that controls the robot, formally a real-valued vector $\bar{x} = \langle x_1, \ldots, x_n \rangle$. The controlling neural network is a perceptron with 9 input

nodes (8 sensor inputs and a bias node), no hidden nodes and 2 output nodes (the left and right motor values) –18 weights in total. Thus, the genome is a vector of 18 real values. The perceptron uses a hyperbolic tangent activation function. Variation in a (1+1) evolution strategy is necessarily restricted to mutation. This is implemented as straightforward Gaussian mutation, adding values from a distribution $\mathcal{N}(0, \sigma)$ to each x_i in the genotype $\bar{x}$. Parent selection in a singleton population is trivial and for survival selection we rely on the so-called + *strategy*: the child (challenger) replaces the parent (champion) if its fitness is higher. This simple scheme defines the core of our evolutionary algorithm, but it is not sufficient to cope with a number of issues in our particular application. Therefore, we extend this basic scheme with a number of advanced features, described below.

Adapting σ values A singleton population is very sensitive to premature convergence to a local optimum. To overcome this problem, we augment the evolutionary algorithm with a mechanism that varies the mutation stepsize σ on the fly, switching from local to global search and back, depending on the course of the search. In particular, σ is set to a pre-defined minimum to promote local search whenever a new genome is stored (that is, when the challenger outperforms the champion). Then, σ gradually increases up to a maximum value (i.e., the search shifts towards global search) while the champion outperforms its children. If local search leads to improvements, σ remains low, thus favouring local search. If no improvement is made on a local basis, either because of a neutral landscape or a local optimum, the increasing σ values ensure that the search will move to new regions in the search space.

Recovery period Because we use *in vivo* fitness evaluation, a new genome $\bar{x}$ needs to be "activated" to be evaluated: it has to be decoded into a neural network and take over the control of the robot for a while. One of the essential design decisions is to avoid any human intervention during evolution, such as repositioning the robot before evaluating a new genome. Consequently, a new controller will start where the previous one finished, implying the danger of being penalised for bad behaviour of its predecessor that, for instance, may have manoeuvred itself into an impossibly tight corner. To reduce this effect, we introduce a *recoveryTime*, during which robot behaviour is not taken into account for the fitness value computation. This favours genomes that are effi-

cient at both getting out of trouble during the recovery phase and displaying efficient behavior during the evaluation phase.

Re-evaluation The evaluation of a genome is very noisy because the initial conditions for the genomes vary considerably: an evaluation must start at the final location of the previous evaluation, leading to very dissimilar evaluation conditions from one genome to another. For any given genome this implies that the measurenemt of its fitness, during the evaluation period, may be misleading, simply because of the lucky/unlucky starting conditions. To cope with such noise, we re-evaluate the champion (i.e., current best) genome with a probability ρ. This is, in effect, resampling as advocated by Beyer (2000) to deal with noisy fitness evaluations and it implies sharing the robot's time between producing and evaluating new genomes and re-evaluating old ones.

The fitness value that results from this re-evaluation could be used to refine a calculation of the average fitness of the given genome. However, we choose to overwrite the previous value instead. This may seem counterintuitive, but we argue that this works as a bias towards genomes with low variance in their performance. This makes sense as we prefer controllers with robust behaviour. It does, however, entail an intrinsic drawback as good genomes may be replaced by inferior, but lucky genomes in favourable but specific conditions. Then again, a lucky genome which is not good on average will not survive re-evaluation, avoiding the adaptive process getting stuck with a bad genome.

5.1.3 Experimental Setup

We evaluate the (1+1)-ONLINE algorithm in a set-up that features actual robotic hardware, a Cortex M3 board with 256kb memory. This controls a simulated autonomous robot in a Player/Stage[i] environment. Using the Cortex board instead of a fully simulated setup is due to administrative constraint in the project within which this research takes place: the Cortex board is the same hardware that is currently being integrated in the SYMBRION robot prototypes and there is a strong emphasis on validation with similar hardware constraints. After N time-steps, the evaluation of the current controller is complete and the controller parameters are

[i] http://playerstage.sourceforge.net

```
for evaluation = 0 to N do
  if random() < ρ then
    Recover(Champion)
    Fitness_Champion = RunAndEvaluate(Champion)
  else
    Challenger = Champion + N(0, σ) {Gaussian mutation}
    Recover(Challenger)
    Fitness_Challenger = RunAndEvaluate(Challenger)
    if Fitness_Challenger > Fitness_Champion then
      Champion = Challenger
      Fitness_Champion = Fitness_Challenger
      σ = σ_min
    else
      σ = σ · 2
    end if
  end if
end for
```

Algorithm 1: The (1+1)-ONLINE evolutionary algorithm.

replaced with values from a new genome, which is evaluated *from the location the previous controller left it in*. This means that **no** human intervention is **ever** needed. We run the experiment 12 times.

Figure 5.1 illustrates the experimental set-up, with a Cortex board connected to the computer running Player/Stage. The simulated robot is modelled after an ePuck mobile robot with two wheels and eight proximity sensors. The maze environment used in our experiment is exactly as shown in this figure.

For each run of the experiment, the robot starts with a random genome and a random seed. The fitness function is inspired by a classic one, described by Nolfi and Floreano (2000) which favours robots that are fast and go straight-ahead, which is of course in contradiction with a constrained environment, implying a trade-off between translational speed and obstacle avoidance. The following equation describes the fitness calculation:

$$fitness = \sum_{t=0}^{evalTime} (speed_{translational} * (1 - speed_{rotational}) * (1 - minSensorValue))$$

All values are normalised between 0 and 1. $minSensorValue$ is the value of the proximity sensor closest to any obstacle, normalised to $[0, 1]$ (i.e., the value decreases as an obstacle gets closer). We used the following settings during our experiments: both $recoveryTime$ and $evaluationTime$ are set to 30 time-steps, $P_{reeavulate}$

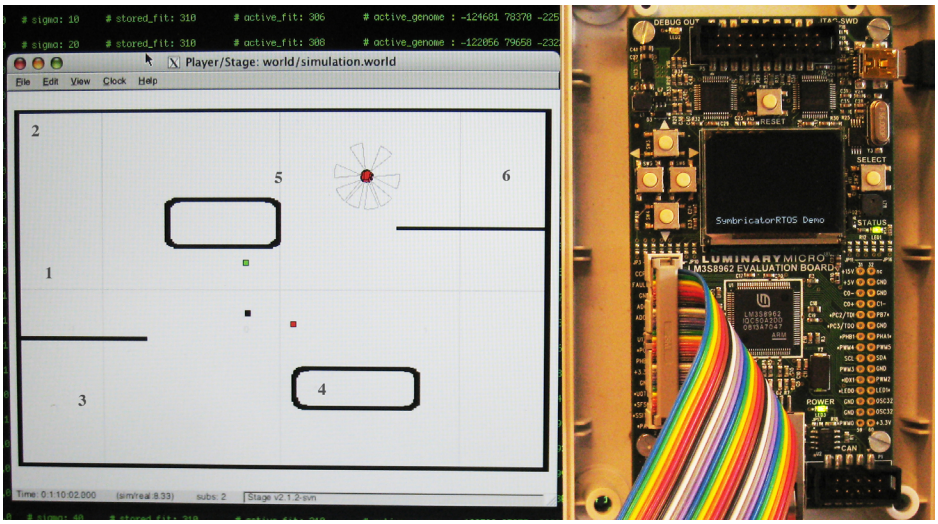


Figure 5.1 – The experimental setup: the Cortex board connected to Player/Stage. The numbers in the player-stage arena indicate the starting positions for the validation trials.

is set to 0.2, the σ initial value is set to 1 and may range from 0.01 up to a maximum of 4 and the gene values are defined to be in $[-4, +4]$.

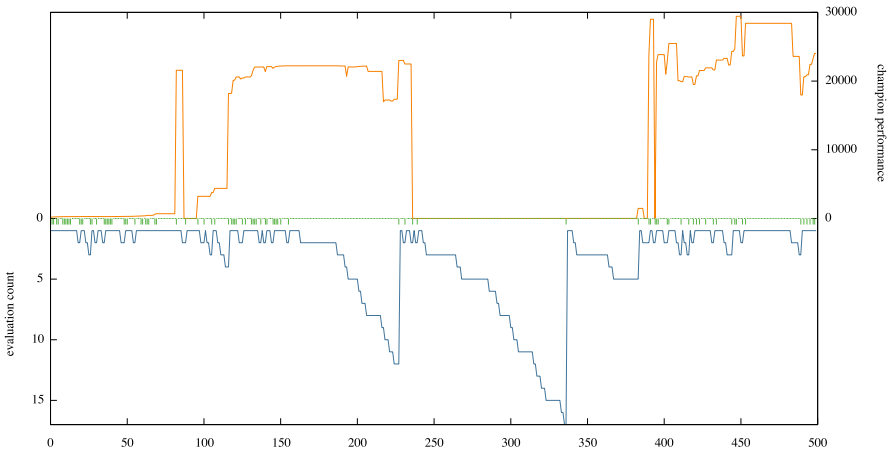
It is important to note that this fitness function is used as a test function. Indeed, the current algorithm is by no mean limited to optimize collision avoidance. Relying on such a fitness function makes it possible to focus on the dynamics of the evolutionary algorithm with regards to desired properties.

To provide an indication of the true performance and reusability of the best individuals found by (1+1)-ONLINE evolution, a hall-of-fame is computed during the course of evolution from the champions of all runs. The 10 best genomes from the hall-of-fame are validated by running each from six initial positions in the environment, indicated in figure 5.1. Starting from each of these positions, the genomes are evaluated for ten times the number of steps used for evaluation during evolution. Note, that one of the validation starting positions has certainly never been visited during development (test no.4, within a small enclosed area) and provides an extreme test case in a very constrained environment. This decomposition into an evolution (development) phase and a post-experiment testing phase is similar to the learning and testing phases commonly seen in Machine Learning and does not imply a deployment phase as in traditional, off-line evolutionary robotics approaches.

5.1.4 Results

Evolution dynamics. We conducted a series of twelve independent experiments (1+1)-ONLINE evolution, with parameters set as stated above. Each experiment started with a different random controller (with very poor behaviour indeed) and a different random seed. The experiments ran for 500 evaluations and displayed different overall fitness dynamics with very similar patterns. Figure 5.2 shows typical statistics from one of those runs. Evaluations are denoted on the x-axis. The y-axis consists of two parts: the top half shows the fitness of the current champion genome. When a champion is re-evaluated very poorly or is replaced by an individual that upon re-evaluation turns out to be very bad, the fitness drops dramatically, as happens in this case after about 250 evaluations. The bottom half of the y-axis shows the number of (re-)evaluations of the current champion (downwards; the lower the line, the higher the number of re-evaluations). Every time the champion is re-evaluated, the line drops down a notch, until a new champion is found; then, the number of re-evalations is reset and the line jumps back to the top. The small vertical markers near the x-axis indicate whenever a new champion is adopted, i.e., when the challenger outperforms the current champion.

Figure 5.2 – Evolution dynamics of a typical run



By analysing the course of the evolutionary process for the experiments, we can observe important mechanisms such as local search (small continuous improvements in the fitness values due to nearby genomes), global search (the ability to get out of a neutral landscape or to jump from a local optimum to a different re-

gion), performance and robustness (the ability of certain genomes to display good performance *and* to remain champion even through re-evaluation).

Initially, performance is quite low, as is the number of re-evaluations; in effect, we are waiting for random search (σ is very high at this point) to bootstrap the adaptation. Then, after about 90 evaluations, an interesting individual is selected as champion that produces children that are even better. We observe a quick sequence of new, better performing individuals and an increasing fitness. After about 160 evaluations, we find that the champion has good fitness and is very robust: the individual survives many re-evaluations (the bottom line goes down) while displaying similar fitness values after successive re-evaluations.

During this period, σ steadily increases (as prescribed by the algorithm), causing mutation to become more and more aggressive, approaching random search. Eventually, this results in a challenger that beats the champion –either because this newcomer is actually very good or because it was lucky (favourable environmental conditions or taking advantage of an unfortunate re-evaluation of the champion). Observation during experiments showed that the latter option is more likely: at some point, the champion encounters a very difficult set-up and is re-evaluated as performing badly so that almost any challenger has a good chance of beating it. In the plot, this is exactly what happens at the precipitous drop in performance after 250 evaluations.

In all our experiments, we saw a similar pattern of initial random search characterised by many different genomes with poor fitness; then, local search characterised by subsequent genomes with increasing fitness until a robust genome is found that survives re-evaluation for some time and then a switch to another region that yields good results or towards an inferior genome that got lucky (almost a restart, in effect).

From the point of view of operational robot control such performance degradation may seem undesirable, but bear in mind that the (1+1)-ONLINE algorithm is meant as a *global* search algorithm. Therefore, such regular fitness reversals are a desired property as long as the search is slightly conservative around good individuals (as is evident from the lengthy episodes of re-evaluation in Figure 5.2). The regular re-evaluation of the champion promotes (in addition to the varying σ discussed below) global search; because of the noisy fitness calculation, if nothing else, it will occasionally be assessed as performing very poorly indeed. Such an

occurrence provides an opportunity for a lucky new and possibly quite different genome to overthrow the champion.

Validation of the hall-of-fame. As described in section 5.1.3, a hall-of-fame was maintained during the course of the experiments for further validation of the -apparently- best genomes. Figure 5.3 shows the results of the validation of the hall-of-fame for the selected re-evaluation scheme (the champion's fitness is overwritten after every re-evaluation) and for two alternatives: one where the fitness is the average of all re-evaluations and one where there is no re-evaluation at all. This allows us to assess two things: whether high ranking genomes in the hall-of-fame are also efficient in a new set-up and whether the "overwrite fitness" re-evaluation scheme is relevant.

The y-axis shows the normalised performance: the best of all individuals for a scenario is set to 1.0, the performance of the other individuals is scaled accordingly. For each scenario (arranged along the x-axis), the graphs show a mark for each individual from the hall-of-fame. All results for a given genotype are linked together with a line.

The graph clearly shows that re-evaluation improves performance substantially; from the ten best solutions without re-evaluation, only a single one performs at a level comparable to that of the ones with re-evaluation. The best individuals in the hall-of-fame for both re-evaluation variants are, on the whole, quite efficient; some come quite close to the maximum possible performance for these test cases (30,000). It is harder to distinguish between the performance of either variants: On the one hand, the spread of performance *seems* greater for the case with averaging fitness than it does for overwriting fitness, which would endorse the reasoning that overwriting after re-evaluation promotes individuals with high average fitness and low standard deviation. On the other hand, however, the nature of real world experiments have a negative impact on the amount of data available for statistically sound comparison of re-evaluation strategies, as is often the case with real hardware, and keep from formulating a statistically sound comparaison. In particular, hardware contingencies implies strong constraints regarding time and human intervention, as robots should be re-located for each experiment and the Cortex board should be reloaded with the genome to be tested, as opposed to the completely autonomous setup during the evolution phase. Overall, the testing

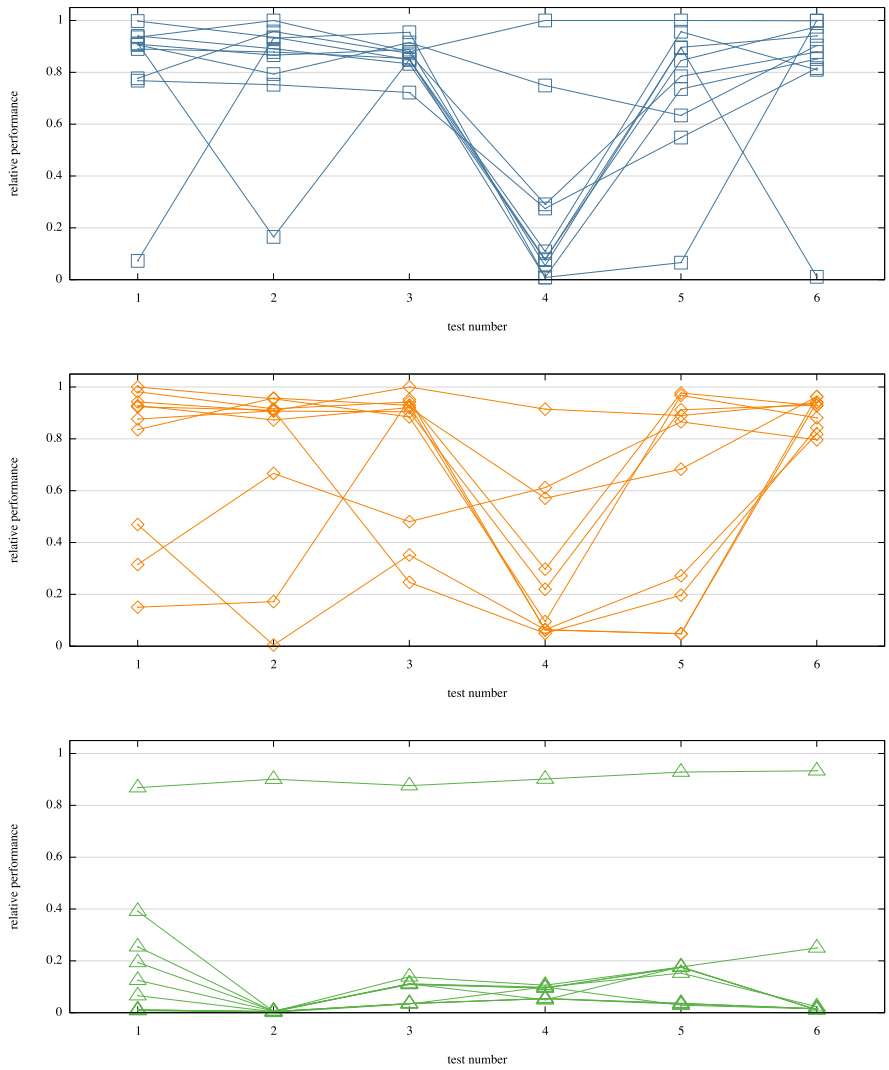


Figure 5.3 – Performance on validation scenarios for various re-evaluation schemes. Top: overwrite-last-fitness scheme ; Middle: average-fitness scheme ; Down: no re-evaluation scheme. X-axis shows the results on the six different validation setups (see fig.1), y-axis shows normalized fitness performance for each run. For a given genome, results in the six validation setups are joined together with a line.

of ten genomes took approx. a full day of work with full investment from the experimenter[ii].

[ii]To some extent, an illustrative metaphor is that of a biologist performing an experiment with mice.

Behavioural diversity. Further analysis of the ten best individuals with the overwrite-fitness re-evaluation scheme shows that the controllers actually display different kinds of behaviour –all good, robust, but different wall avoidance and/or open environment exploration strategies, ranging from cautious long turns (reducing the probability of encountering walls) to exploratory straight lines (improved fitness but more walls to deal with). Figure 5.4 illustrates this by showing the pathways of these individuals, starting from an initial position on the left of the environment. This reflects the genotypic diversity observed in the hall-of-fame and hints at the algorithm's capability to produce very different strategies with similar fitness.

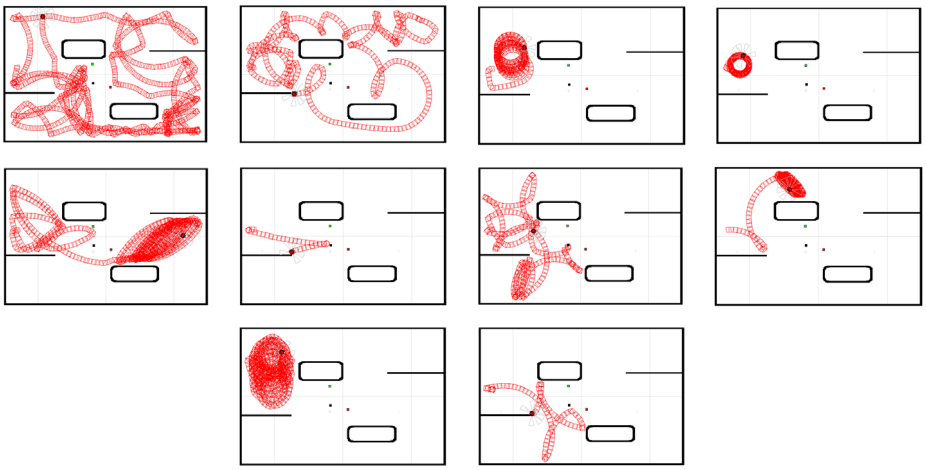


Figure 5.4 – Traces for the ten best controllers (using fitness replacement after re-evaluation)

5.1.4.0.1 Strong causality and mutation. The reasoning behind the scheme to update σ relies on Strong Causality (Rechenberg, 1973): it only holds if small changes in the genome lead to small changes in behaviour and big changes in the genome to big changes in behaviour. To investigate if this property holds, a separate set of experiments was performed. For a range of σ values, 200 mutants were created from some fixed initial genome; every one of these mutants was then tested in our arena, from 193 different starting locations (homogeneously distributed over the environment), each with four orientations (i.e., a total of 772 tries per genome); each evaluation lasted 30 time-steps. Because such experiments using the Player/Stage and Cortex set-up as described above would require ap-

prox. 3.5 years to run, we used a simplified autonomous robot simulator. Each experiment started from one specific genome, the first experiment from the best genome in the hall-of-fame (Fig. 5.5.(a)) and the second experiment from a randomly generated genome (Fig. 5.5.(b)). In both figures, The x-axis shows σ. The y-axis shows the range of fitness values: the sum of fitnesses for each mutant over all 772 trials. For every value of σ, the candle bars show the minimum, maximum, median and lower and upper quartile. Figure (c) shows a histogram of the frequency of σ values over 12 runs (approximately 4,700 evaluations) of the original experiment. The (logarithmic) x-axis shows occurring values for σ, ranging from 0.01 to 4. The count of occurrences is displayed along the y-axis.

(a) starting from one of the best genomes

(b) from a random start

(c) Incidence of σ values

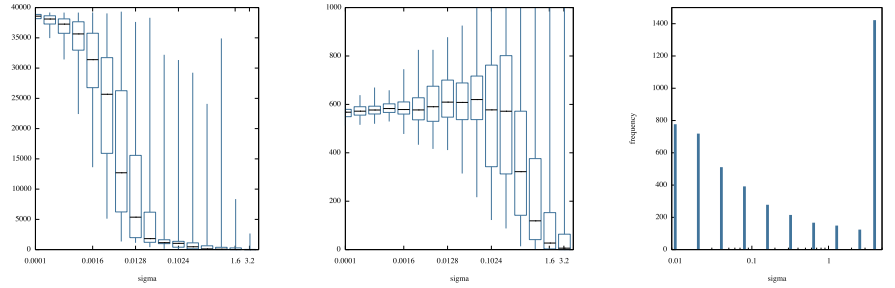


Figure 5.5 – Strong causality experiments

Graphs (a) and (b) show that, as σ increases, the performance of mutated individuals becomes increasingly different; it actually covers the whole domain for medium to large values of σ. When starting from the 'best' genome, the average performance decreases as the mutations move further and further away from the original genome. From a randomly generated start point, performance changes either way as we move away from the original genome. This shows that there is strong causality: small changes in the genome (low σ) lead to small variations in fitness, and big changes lead to large variations. Finally, figure 5.5.(c) shows the density of σ values over the 12 original runs of the original experiment. The (logarithmic) x-axis shows occurring values for σ, ranging from 0.01 to 4, and the count of occurrences is displayed along the y-axis. As shown in the graph, all possible values of σ from very small (entailing local search) to large (global search) frequently occurred in the course of our experiments, with more occurences of both the minimum value (ie. local search) and maximum value (ie. global search). This

provides a sound validation of the (1+1)-ONLINE algorithm ability to conduct both local and global search thanks to the self-updating σ.

5.1.5 Conclusions and Further Work

This section provides a proof-of-concept for the viability of on-line, on-board evolution in autonomous robots. We have presented the (1+1)-ONLINE evolutionary algorithm to provide continuous adaptation in autonomous robots in unknown and/or changing environments, without help from any supervisor (human or otherwise). The (1+1)-ONLINE evolutionary algorithm is based on an "enclosed" evolutionary algorithm approach, as explained in the introduction. It was tested on very constrained, embedded hardware – the Cortex board we used in the experiments is limited in terms of performance as well as memory (256kb, including the operating system, complying with the robot prototype actually under construction). This requires a light-weight, low-complexity algorithm such as the one presented here, which is derived from the well known and well established $(1+1)$ evolution strategies.

One of the main contributions of the (1+1)-ONLINE evolutionary algorithm is that, by using re-evaluation, it specifically deals with intrinsically noisy performance evaluation in real world environments. This greatly increases the real-life applicability of this method and constitutes an original contribution compared to previous research into similar on-line setups. The second contribution is that of balancing local and global search through the σ update. Walker et al. (2006) described a similar $(1+1)-$evolution strategy inspired scheme with self-tuning σ – however, the proposed approach updates σ through a heuristic that explicitly tunes the local and global search.

An on-line approach such as presented here tackles problems beyond the scope of traditional off-line evolutionary robotics , such as dealing with dynamic or unknown environments for which the robots could not be optimised before their deployment and it also addresses issues such as the reality gap and the lack of fidelity in simulation (Brooks, 1991; Watson et al., 2002).

In the experiments shown here, the (1+1)-ONLINE evolutionary algorithm yielded a great variety of behaviours in a limited amount of time; good performance was typically reached in under an hour. While the task at hand is relatively simple (obstacle avoidance and maximisation of translation speed), it should be noted

once again that it requires no background knowledge whatsoever about the task and that the current algorithm can be applied in different contexts, simply by rewriting the fitness function.

The dynamics of evolution often result in the loss of a very good genome –this is actually desired behaviour of the algorithm as it ensures continued exploration of new or changed regions in the search space. It could, however, be interpreted as a complication from the engineer's viewpoint in a production environment where one wants to retain good genomes. This is in fact an instance of the well-known issue of exploration vs. exploitation in reinforcement learning; in this context, the algorithm proposed here provides the *exploration*. A reservoir such as the Hall-of-Fame introduced above can keep track of the best genomes and allow them to be re-used for exploitation.

Further research focusses on the following issues. Firstly, we consider alternative schemes to update the champion's fitness value after re-evaluation to combine the benefits of the "last fitness" and the "average fitness" approaches. This could for instance be achieved by averaging over a sliding window or by weighting the influence of the latest fitness estimate and previous ones. Secondly, we intend to extend the algorithm towards a multi-robot set-up combining the embedded and enclosed approaches (cf. Section5.1.1): adding a genome migration feature would make it possible to spread good genomes through a population of robots – similar to an island-based parallel evolutionary algorithm. Global search would then still be possible on the local scale of individual robots while retaining good genomes on the global scale of the population. The underlying hypothesis behind this assumption is that a genome duplicated on many robots in the population may have a good estimation of the average fitness distributed over the population –with the estimation confidence growing with population size. Yet again, this remains to (and will) be implemented and tested, as the trade-off between local "enclosed" adaptation and distributed "embodied" adaptation remains an open issue. Thirdly, we are to test on-line, on-board evolution in a group of robots within dynamic environments.

5.2 On-line evolution of robot controllers by an encapsulated evolution strategy

This section describes and experimentally evaluates the viability of the $(\mu + 1)$ ON-LINE evolutionary algorithm for on-line adaptation of robot controllers. Secondly, it explores the parameter space for this algorithm and identifies four important parameters: the population size μ, the re-evaluation rate ρ, the mutation step-size σ and the controller evaluation period τ. Subsequently, it investigates their influence on controller performance, stability of behaviour and speed of adaptation. The results indicate that the encapsulated on-line evolutionary approach is a viable one and merits further research. In agreement with existing research, the mutation step-size σ proves to be of overriding importance to finding good solutions. Specific to on-line evolution, the results show that longer evaluation times greatly benefit the quality of controllers as well as stability of behaviour and speed of adaptation.

5.2.1 Introduction

Evolutionary algorithms have various applications within robotics, as designers, respectively optimisers of robot controllers, morphological or functional features (Nolfi and Floreano, 2000). This section is concerned with optimizing robot controllers. To position our approach we use a small taxonomy whose topmost junction distinguishes two cases by considering when the evolutionary algorithm is applied, before deployment or after deployment of the controllers. The corresponding terminology distinguishes *off-line* (development time) and *on-line* (run time) evolutionary algorithm applications as outlined by Eiben et al. (2010a).

Traditionally, evolutionary robotics focusses on off-line applications of evolutionary computation, where an evolutionary algorithm is used to design, respectively optimise, controllers before deployment. Controllers (phenotypes) are rep-

---

Section 5.2 was published as:

Evert Haasdijk, A.E. Eiben and Giorgos Karafotias (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Pages 1–7, IEEE Press, Piscataway, NY.

---

resented by appropriate genotypes and a population of such genotypes undergoes evaluation, selection, and variation in a computer external to the robot. This process terminates at some point with a controller that is deployed onto real robots that will subsequently perform their task without further adaptation (at least, without further evolution). During the evolutionary process, evaluation of controllers can be performed by testing them either in simulation or in real robots. However, even in the latter case we have to do with off-line evolution, since the real-life tests with robots using a given controller merely serve as fitness calculations. The results are passed back to the evolutionary algorithm running on the computer that carries out the variation and selection operators and initiates new trials until some termination condition is met and the best evolved controller is deployed as the end result. Figure 5.6 illustrates this approach.

Here, by contrast, we consider the on-line application of evolutionary computation to design robot controllers, where an evolutionary algorithm is used to provide continuous adaptation as the robots perform their tasks in real life. The major difference with off-line evolution is that in this case controllers do undergo evaluation, selection, and variation after deployment.

Broadly speaking, there are two kinds of approach to on-line evolution of robot controllers. One approach, *distributed* evolution, exemplified by Watson

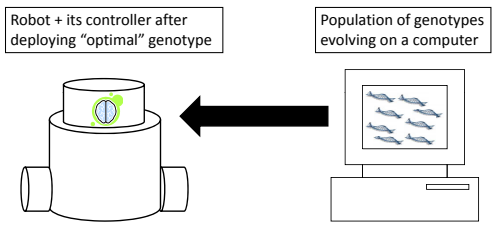


Figure 5.6 – The classical approach to evolve robot controllers. Evolution takes place off-line, before deployment, in an external computer. The population of controllers undergoes selection and variation inside this computer. Fitness evaluation can be either done in simulation (inside this computer again), or "in vivo" by sending the controller to a real robot that uses it for a while to collect information on its quality. The black arrow indicates the deployment of the final contoller.

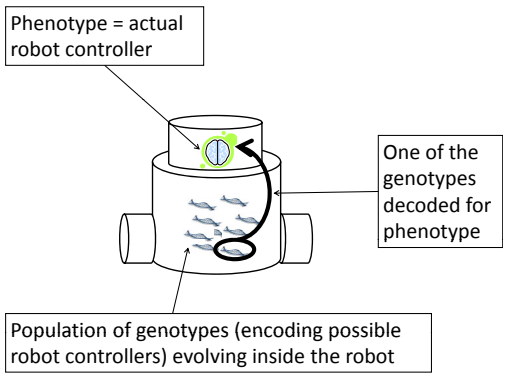


Figure 5.7 – The encapsulated approach to evolve robot controllers. Evolution takes place on-line, after deployment, in an internal computer. The population of controllers undergoes selection and variation inside the robot itself. Fitness evaluation is done "in vivo" by decoding one of the genotypes into an active controller and let the robot use it for a while to collect information on its quality. To evaluate all genotypes some kind of time-sharing mechanism must be used.

et al.'s method 2002, has a single controller in each robot and implements selection and variation (reproduction) operators through the interactions between individual robots.The second approach encapsulates a complete evolutionary algorithm with a population of controllers within each robot; the robots individually adapt through evolution without the necessity of interaction amongst themselves.Figure 5.7 illustrates this *encapsulated* approach. Of course, these two methods may be combined, yielding a system analogous to that of an island-based parallel evolutionary algorithm with each robot running its own evolutionary algorithm and the interactions between robots amounting to migration between islands.

The work in this section falls in the second category with an encapsulated population in each robot, without migration. We are investigating this approach in the context of a running research project, SYMBRION, where on-line evolution is one of the pivotal mechanisms for adaptive robot control. Inherent to this project, and to some extent to on-line evolutionary approaches in general, are the physical limitations:

1. Even though the robot's evolving population contains multiple controllers, at any time only one of them can actually control the robot. Consequently, a

time sharing system must be implemented that activates controllers one by one.

2. The evolutionary process must be autonomous, without any human intervention or central control. Hence, when a new controller is activated for evaluation, its test period starts at the location where the previous controller led the robot.

3. To obtain sufficient feedback on the quality of a given controller, its test period –the time-span where it is activated and actually controls the robot– should be sufficiently long.

5.2.2 Considerations in On-Line Evolution

The constraints listed above imply some considerations specific to on-line evolution and its analysis.

The first challenge derives from the real-time character of the evolutionary process. Fitness evaluations need a test period with a reasonable length l (say, in minutes) to obtain realistic performance figures. Meanwhile, the whole experiment is constrained by a reasonable maximum duration L (again, in minutes). Consequently, the total number of fitness evaluations available to the evolutionary process is limited to $\frac{L}{l}$. Obviously, this ratio can vary depending on various practical details, but in our practice it falls in the range between 500 to 1500. In general, it is impossible to say what the minimum number of fitness evaluations is for a decent evolutionary progress, but one thousand is definitely a very low budget to spend compared to what is common in evolutionary algorithms.

The second challenge is the noisy nature of the fitness evaluations. Using the off-line evolutionary approach with human intervention it is possible to test a given controller starting at different locations (in general: under different circumstances). This helps to obtain good fitness information in two ways, by producing more data –one fitness value for each starting point– and by the ability to use representative or otherwise well-selected locations. However, in the on-line case, where human intervention is excluded, starting locations are arbitrary and we only have one measurement for each activated controller. Consequently, the evaluation of a genome is inherently very noisy because of the very dissimilar evaluation conditions from one genome to another. For any given genome, this implies that the

evaluation of its fitness may be misleading, simply because of lucky or inauspicious starting conditions. Given these considerations, the viability of the on-line evolutionary approach itself is an open question.

Thirdly, actual performance matters: in contrast to typical applications of evolutionary algorithms, the best performing individual is not necessarily the most important when applying on-line adaptation. Remember that controllers evolve as the robots go about their tasks; if a robot is continually evaluating poor controllers, that robot's actual performance will be inadequate, no matter how good the best known individuals as archived in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution (even if it is not the best) and search prudently: it must display a more or less stable level of performance throughout the continuing search. This leads to considerations very similar to those concerning the trade-off between exploration and exploration in reinforcement learning.

This section proposes an algorithm for encapsulated on-line evolution of robot controllers and concerns itself with two questions. The first question is whether, in the face of the considerations outlined above, such an algorithm can evolve good controllers on-the-fly. Secondly, it investigates the interplay between a number of parameters of the proposed algorithm. To this end, we implement the mechanism in the well-known simulation platform *Webots*[iii] and conduct a series of experiments where controllers to perform a simple task must evolve from scratch. The next section describes the algorithm in detail.

5.2.3 The $(\mu + 1)$ on-line Evolutionary Algorithm

The challenge concerning the low number of fitness evaluations mandates that the evolutionary algorithm must converge very quickly to an acceptable level of solution quality. Therefore, we have chosen to base our method on evolution strategies (Schwefel, 1995), because 1) the controllers we have in mind can be parameterised, hence represented by a vector of real-valued numbers, 2) evolution strategies have a very good reputation as evolutionary solvers of numerical optimisation problems (Bäck, 1996). As the notation indicates, $(\mu + 1)$ ON-LINE generates $\lambda = 1$ child per cycle. This value is extremely low to evolution strategy standards, where the $\frac{\lambda}{\mu}$ is usually between 4 and 8, but using $\lambda = 1$ can save on fitness evaluations.

[iii] `http://www.cyberbotics.com/`

Our $(\mu + 1)$ ON-LINE evolutionary algorithm comprises an encapsulated evolutionary algorithm, where a population of μ individuals is maintained within each robot. As an encapsulated evolutionary algorithm it is similar to the algorithms described by Haroun Mahdavi and Bentley (2006); Nehmzow (2002); Walker et al. (2006) concerning its main design principle, but it has a number of specific novel features. It is also different from its earlier version described in section 5.1 in that

- the present version uses fitness-based parent selection, rather than selecting parents by a uniform distribution,
- the present version uses recombination (crossover), rather than mutation only,
- in the present version the mutation step-sizes are either constant or self-adaptive, while Bredeche et al. used a heuristic adaptive scheme to adjust them on-the-fly,
- in the present version the extra information obtained by re-evaluation (see details later) is used to update, rather than replace, old information.

Below we discuss the specific properties of our $(\mu + 1)$ ON-LINE evolutionary algorithm; its pseudo code is shown in Alg. 2.

To cope with the issue of inherently noisy fitness evaluations, $(\mu + 1)$ ON-LINE re-evaluates genomes in the population with a given probability. This means that at every evolutionary cycle two things can happen: either a new individual is generated and evaluated (with probability $1 - \rho$), or an existing individual is re-evaluated (with probability ρ). To ensure that re-evaluation efforts are spent on promising individuals, the individual to be re-evaluated is chosen by binary tournament selection from the whole population. The fitness values from subsequent (re-)evaluations of any given individual are combined using an exponential moving average; this emphasises newer performance measurements and so is expected to promote adaptivity in changing environments. This is, in effect, a resampling strategy to deal with noisy fitness evaluations as advocated by Beyer (2000).

To promote rapid convergence we diverge from the common practice of uniform random parent selection in evolution strategies and use binary tournament parent selection, increasing the selective pressure. For the same reason, we use $\lambda = 1$ and apply recombination. Thus, in each cycle, one new individual is created from two parents, each of which is selected with a binary tournament. Se-

lective pressure is increased even further by using an plus-strategy, even though self-adaptive mutation rates such as we have here usually call for using a comma-strategy (Eiben and Smith, 2008).

```
for i = 1 to μ do
    // Initialisation
    population[i] = CreateRandomGenome ;
    population[i].σs = σ_initial;
    population[i].Fitness = RunAndEvaluate(population[i]);
for ever do
    // Continuous adaptation
    if random() < ρ then
        // Don't create offspring, but re-evaluate selected individual
        Evaluatee = BinaryTournament(population);
        Recover(Evaluatee) ;
        // Brief intermezzo of random movement to get out of bad
           situations due to previous evaluation
        Evaluatee.Fitness = (Evaluatee.Fitness +
        RunAndEvaluate(Evaluatee)) / 2;
        // Combine re-evaluation results through exponential moving
           average
    else
        // Create offspring and evaluate that as challenger
        ParentA = BinaryTournament(population);
        ParentB = BinaryTournament(population - parentA);
        Challenger = AveragingCrossover(ParentA, ParentB);
        // Crossover also recombines σs
        Mutate(Challenger);
        // Mutation also updates σs cf. Eiben and Smith (2008, p. 76)
        Recover(Challenger);
        // Brief intermezzo of random movement to get out of bad
           situations due to previous evaluation
        Challenger.Fitness = RunAndEvaluate(Challenger);
        // Replace last (i.e. worst) individual in population w.
           elitism
        if Challenger.Fitness > population[μ].Fitness then
            population[μ] = Challenger;
            population[μ].Fitness = Challenger.Fitness;
    Sort(population);
```

Algorithm 2: The $(\mu + 1)$ ON-LINE evolutionary algorithm.

5.2.4 Experimental Set-up

As mentioned in Section 5.2.2, we conduct a series of experiments. Firstly, to verify that the $(\mu + 1)$ ON-LINE algorithm is capable of producing robot controllers with acceptable quality within acceptable time. Secondly, to investigate the effect of a number of parameters of the $(\mu + 1)$ ON-LINE algorithm.

$(\mu + 1)$ ON-LINE sports two parameters that are peculiar to the challenges posed by on-line, on-board evolution and directly influence the speed of evolutionary adaptation. These are:

- ρ The re-evaluation rate: larger values for ρ lead to better fitness value estimations, thus improving the quality of selection, meanwhile slowing down the search. ρ's value governs the likelihood of using an evaluation cycle for re-evaluation of one of the current population members instead of evaluating a newly generated controller. We tried three values for ρ: 0.2, 0.4 and 0.6;

- τ The duration of controller evaluation: increasing τ increases the evaluation's reliability while it obviously decreases the number of evaluations per time-unit and thus the search. Controller evaluations are measured in ticks: the simulator invokes the controller once per tick, one tick lasting 50 milliseconds simulated time in our experiments. We tried two settings: 60 and 300, corresponding to 3 and 15 seconds simulated time, respectively.

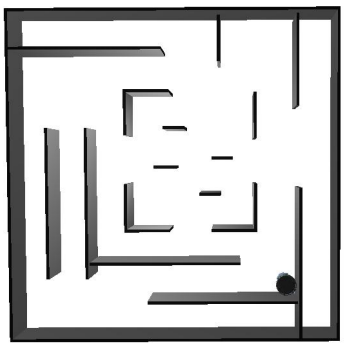


Figure 5.8 – The arena used in the experiments. The circle represents an e-puck robot to scale.

Two further parameters that might be expected to be influential from general evolutionary algorithm point-of-view are:

μ The population size: a larger population size reduces the danger of getting stuck in local optima, meanwhile it slows down the search. We ran experiments with μ set to 6, 10 and 14;

σ The mutation step-size. We compare two regimes that manage the mutation step-size: the standard self-adaptation mechanism used in evolution strategies (see Eiben and Smith (2008, p. 76)) and a simplistic approach using a constant σ value. As fixed σ values, we used 0.2 and 0.8.

Note that in section 5.1, we used a σ adaptation scheme that varied the σ values based on a heuristic in an attempt to balance exploration and exploitation. Here we look at alternatives, but strictly speaking we cannot consider it a comparison with the previous version of $(\mu + 1)$ ON-LINE because many other details of the evolutionary algorithm have changed as well.

All together, we have 54 algorithm variants to compare here: 3 values for μ, and 3 different values for ρ, 3 different σ management mechanisms/values and two τ values. The details of the experimental settings are shown in Table 5.1.

As a test case, we have chosen e-puck robots in an arena and a classical task after Nolfi and Floreano (2000). The fitness function representing this task favours robots that are fast and go straight-ahead, which, in a constrained environment, forces a trade-off between translational speed and obstacle avoidance. Equation 5.1 describes the fitness calculation:

$$f = \sum_{t=0}^{evalTime} (v_t \cdot (1 - v_r) \cdot (1 - d)) \tag{5.1}$$

where v_t and v_r are the translational and the rotational speed, respectively. v_t is normalised between -1 (full speed reverse) and 1 (full speed forward), v_r between 0 (movement in a straight line) and 1 (maximum rotation); d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle)

The evolutionary algorithm governs the weights in a neural net-based robot controller. This neural net is a perceptron with a hyperbolic tangent activation function using 9 input nodes (8 proximity sensor inputs and a bias node), no hidden nodes and 2 output nodes (the left and right motor values), resulting in a total of 18 weights. To evolve these 18 weights, the evolutionary algorithm uses the obvious representation of real-valued vectors of length 18 for the genomes.

For each single run of the experiment, the robot starts with a fresh random seed and a population of μ randomly generated genomes.

The experiments were performed in pure simulation using the Webots simulator. Each experiment has a single robot running its own autonomous instance of $(\mu + 1)$ ON-LINE. For each combination of parameter settings, we conducted 100 trials. The robot controller is called once every time-step, each time-step lasting 50 milliseconds simulated time.

Table 5.1 – Experiment description table for the $(\mu + 1)$ ON-LINE tests

Experiment details	
Task	fast forward
Arena	see Fig. 5.8
Robot group size	1
Simulation length	10,000 seconds (simulation time)
Number of repeats	100
Controller details	
ANN type	perceptron
Input nodes	9 (8 sensory inputs and 1 bias node)
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length L	18
Fitness	See Eq. 5.1
Recovery time	10 time steps
Evaluation time	60 or 300 time steps
Re-evaluation rate ρ	0.2, 0.4, 0.6
Re-evaluation strategy	exponential moving average
Population size μ	6, 10, 14
Mutation	Gaussian $N(0, \sigma)$
Mutation rate	Self-adaptive with $\sigma_0 = 0.8$ or fixed at 0.2 and 0.8
Crossover	averaging
Crossover rate	1.0
Parent selection	binary tournament
Survivor selection	replace worst in population if better

5.2.5 Results and discussion

As noted before, we are specifically interested in the actual performance of the algorithm, i.e., the performance of the active controllers averaged over a period of time (a number of evaluations). Clearly, this includes performance information of controllers that evaluate poorly and are discarded after the evaluation period, as well as controllers that survive the (re-)evaluation period and (re-)enter the population. Another performance indicator is that of the best performance, i.e., the performance of the best controller in the population, regardless whether this controller is currently active. In addition to performance we will consider two other indicators of behavioural quality.

First, consider the stability, or rather the noisiness of the adaptive process. Even though a run may exhibit good actual performance on average, it is preferable if performance is more or less constant. Robots that often lapse into very poor behaviour as they consider candidate controllers are less desirable than robots that operate at a consistent level. To measure this quantity, we analyse the differential entropy (Lazo and Rathie, 1978) of the actual performance in runs of the experiment.

Secondly, we are interested in the speed of adaptation, that is the rate of performance improvement over time. In particular, we are looking for the turtle-and-hare effect.

In the following subsections we will discuss the experimental results from the perspective of these indicators.

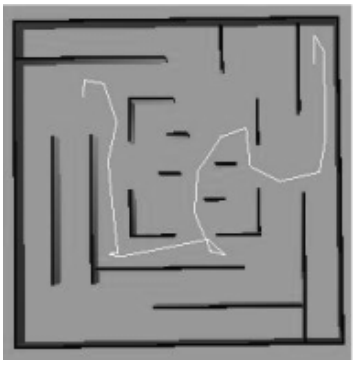


Figure 5.9 – Trajectory of one of the better controllers towards the end of the run.

5.2.5.1 Performance of controllers

Our test landscape lies within a four-dimensional space, with one dimension belonging to each of the parameters we vary over the experiments: mutation step-size σ, evaluation period τ, population size μ, and re-evaluation rate ρ. Intensive inspection of the data reveals that these parameters differ greatly in their impact on the outcomes, but showed no noticeable interactive effects, so we feel justified to consider them independently. We do so in order of decreasing influence. The statistics shown in this subsection have been compiled over approximately the last 8 minutes of simulated time in the experiments.

Mutation step size The most influential parameter turns out to be σ. In Figure 5.10 we present the actual and best performance observed over the last 8 minutes (simulated time) of each run as a function of different σ values. These plots show the average values, taken over the complete set of experiments, that is, for all investigated values of all other parameters, amounting to $2 \cdot 3 \cdot 3 \cdot 100 = 1800$ data points behind each bar. The results show that small σ values (0.2 is about $\frac{1}{40}$-th of the domain of the variables x_i) do not work. The high value for σ we tried (0.8 is about $\frac{1}{10}$-th of the domain of the variables x_i) is clearly the best choice for actual performance and finishes as a close runner-up to the self-adaptive regime in the plot for best performance.

To interpret these results it is important to note that the self-adaptive regime regulates 18, possibly different, step sizes for each controller: one separate σ value for each of the real-valued object variables in the artificial genome representing a controller. This means that in this case evolution is solving a double task: optimising the 18 object variables and finding good step sizes on-the-fly, and there are simply not enough evaluations available to pull that off.

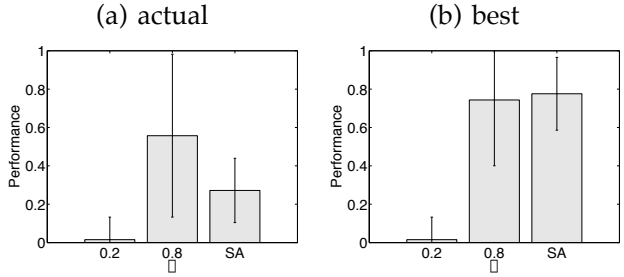


Figure 5.10 – Effect of σ on performance.

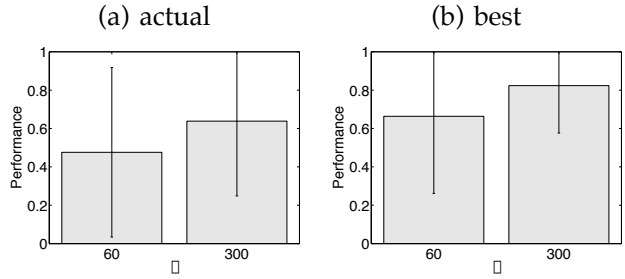


Figure 5.11 – Effect of τ on performance.

Evaluation period Next, we consider the effect of the evaluation period τ. Figure 5.11 exhibits the actual and best performance observed over the last 8 minutes (simulated time) of each run as a function of different τ values when using $\sigma = 0.8$. The results clearly indicate the superiority of longer evaluation periods. $\tau = 300$ delivers better results than $\tau = 60$. This is significant information, as in general it is not obvious whether more, but shorter evaluations or fewer, but longer evaluations lead to better performance. In essence, this is a trade-off between quantity (lower τ) and quality (higher τ) and our experiments support a preference for the latter.

Population size The population size μ is the third most influential parameter among those we consider here. The significance of this parameter is obvious, evolutionary search through smaller populations can proceed faster, but it is also more easily trapped in local optima. As Figure 5.12 shows μ does not have a great effect on best performance, but as for actual performance we can articulate a preference for the smallest value we tested, $\mu = 6$. This is good news, considering the hardware limitations in robots.

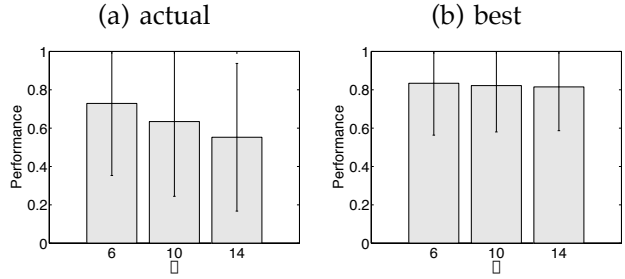


Figure 5.12 – Effect of μ on performance.

Re-evaluation rate Finally, our experiments also shed light on the effects of different re-evaluation rates. Somewhat to our surprise, ρ is not a very influen-

tial parameter as the results in Figure 5.13 indicate. Smaller values of ρ seem to advance better best performance, even though the differences are not too big. In theory, this makes sense considering that spending less time on re-evaluating known candidate solutions allow to visit more points in the search space. Looking at the data regarding actual performance we see again rather small differences between the values considered. Having noted this we choose the middle value, $\rho = 0.4$ as our favourite.

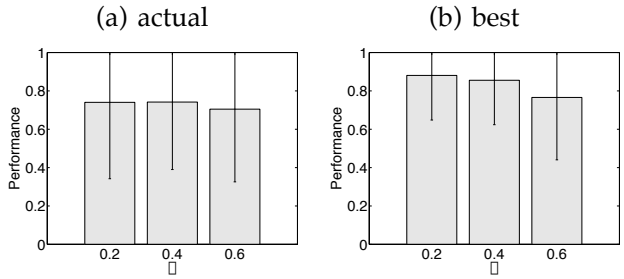


Figure 5.13 – Effect of ρ on performance.

5.2.5.2 Stability of actual performance

To assess the volatility of the robot's actual performance over the course of the experiments, we calculate the differential entropy of actual performance over the full length of each run. For this analysis, runs with step size $\sigma = 0.2$ were excluded as their performance was uniformly low and therefore had minimal entropy; this muddles the analysis for more interesting values of σ. We found that only the evaluation period τ has an appreciable influence on the level of entropy Fig. 5.14 shows the average entropy for different values of τ. All runs apart from the runs with $\sigma = 0.2$ are included in the calculations for this graph. Lower entropy (in this case, a longer bar as the values are negative[iv]) indicates a lower level of volatility: the runs with $\tau = 300$ clearly lead to much more consistent behaviour.

5.2.5.3 Speed of adaptation

Fig. 5.15 shows the development of performance over time for actual and best performance and for the two values of τ. Each graph contains three series: one for each value of ρ. Only results from runs with σ set to the optimal value of 0.8

[iv] This is not regular Shannon entropy but the *differential* entropy, which can be less than 0.

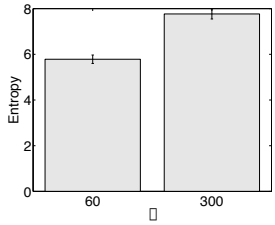


Figure 5.14 – τ against entropy of actual performance

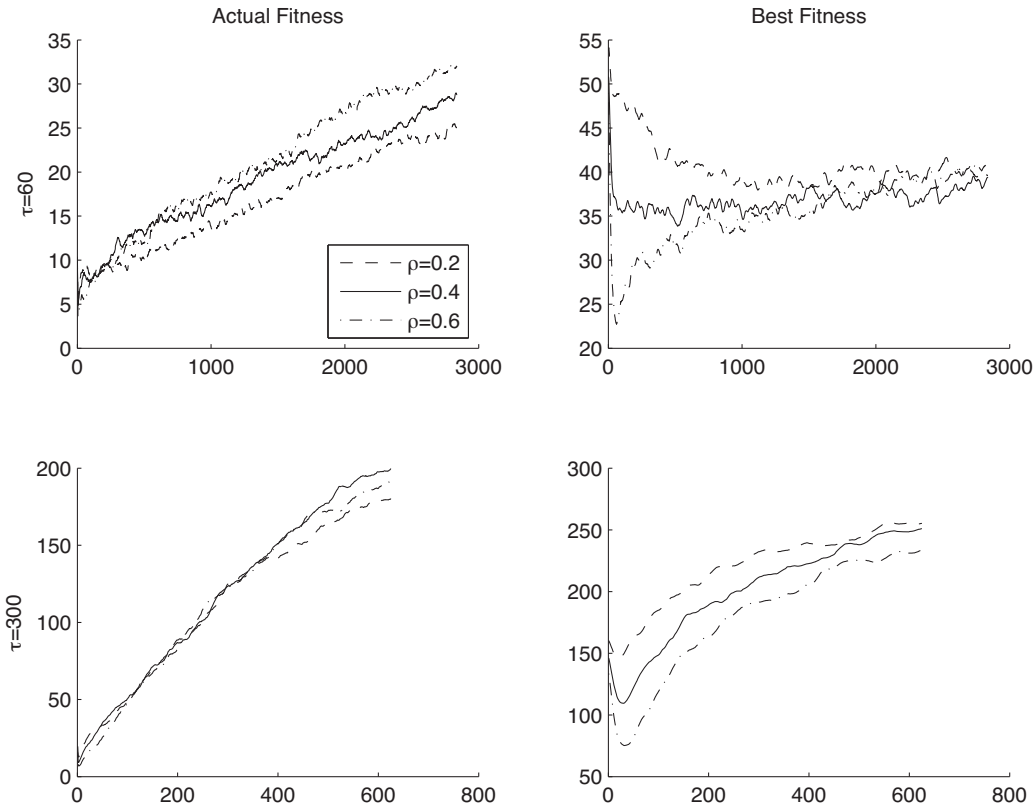


Figure 5.15 – Average actual and best (left and right column, respectively) performance over time for different values of τ and ρ

are included. The influence of μ on the speed of adaptation appeared negligible, hence μ is disregarded in these graphs: the results are taken across all values of μ.

To construct these graphs, we first calculated a moving window average to smooth the performance curves for each individual. The resulting figures were then averaged over all appropriate runs to yield the values plotted here.

In Subsection 5.2.5.1, we already saw that $\tau = 300$ yields the best results, and here we see that the performance also increases fastest for $\tau = 300$, and again to our surprise, ρ does little to influence the rate of performance increase.

Note, that the performance graphs have not yet leveled of at the end of the runs, from which one may conclude that performance could increase further yet as time progresses.

5.2.6 Conclusion

This section presented the $(\mu + 1)$ ON-LINE evolutionary algorithm to provide the possibility of on-line evolutionary adaptation in robotics. This algorithm was specifically designed to address three challenges inherent in on-line adaptation: noisy evaluations, relatively few evaluations and the primacy of actual as opposed to best performance throughout the developmental process. While drawing on the well-established field of evolution strategies, $(\mu + 1)$ ON-LINE diverges from common evolution strategy implementations to increase algorithm speed by using $\lambda = 1$ and non-random parent selection.

Revisiting the research questions we posed in Sec 5.2.2, we can firstly conclude that our experiments show that the $(\mu + 1)$ ON-LINE algorithm is indeed capable of developing acceptable controllers as the robot performs its task.

Secondly, the results show that the mutation step-size σ is the single most decisive parameter when it comes to delivering good controllers. This is in line with previous research into parameter setting for evolutionary algorithms (Nannen et al., 2008). The controller evaluation period τ, specific to on-line evolution, is the next most important parameter when it comes to quality. Moreover, it is the deciding parameter when considering the stability of performance and speed of adaptation. Regarding the population size μ, we have seen that having larger values does not improve the performance of even the best known controllers, while the penalty of storing lower-quality alternative controllers manifests itself in decreasing actual performance. It may yet prove beneficial, however, in dynamic environments where it can enable falling back on remembered solutions. The $(\mu + 1)$ ON-LINE algorithm has proved to be fairly insensitive to variations in the re-evaluation rate ρ. While these results are promising and merit our consideration into on-line evolution of robot controllers, they cannot be indiscriminately generalised to other tasks, robots or even environments without further investigation.

5.3 Racing to Improve On-line, On-board Evolutionary Robotics

In evolutionary robotics, robot controllers are often evolved in a separate development phase preceding actual deployment – we call this off-line evolution. In on-line evolutionary robotics, by contrast, robot controllers adapt through evolution while the robots perform their proper tasks, not in a separate preliminary phase. In this case, individual robots can contain their own self-sufficient evolutionary algorithm (the *encapsulated* approach) where individuals are typically evaluated by means of a time sharing scheme: an individual is given the run of the robot for some amount of time and fitness corresponds to the robot's task performance in that period.

Racing was originally introduced as a model selection procedure that quickly discards clearly inferior models. We propose and experimentally validate racing as a technique to cut short the evaluation of poor individuals before the regular evaluation period expires. This allows an increase of the number of individuals evaluated per time unit, but it also increases the robot's actual performance by virtue of abandoning controllers that perform inadequately. Our experiments show that racing can improve the performance of robots that adapt their controllers by means of an on-line evolutionary algorithm significantly.

5.3.1 Introduction

The work presented in this section is inspired by a vision of autonomous, self-sufficient robots and robot collectives that can cope with situations unforeseen by their designers. An essential capability of such robots is the ability to adapt their controllers in the face of challenges they encounter in a hands-free manner (Bredeche et al., 2009), "the ability to learn control without human supervision,"

Section 5.3 was published as:

Evert Haasdijk and Arif Atta-ul-Qayyum and A.E. Eiben (2011). Racing to Improve On-line, On-board Evolutionary Robotics. In Natalio Krasnogor et al., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Pages 187–194, ACM, NY.

---

as Nelson et al. (2009) put it. Crucially, the robot's controller changes on the fly, as it goes about its tasks: adaptation –evolution, in our case– occurs *on-line*.

This approach contrasts with the majority of work in the field of evolutionary robotics, which to date has primarily focussed on *off-line* evolution of robot controllers, where the evolutionary process takes place as a separate development stage before proper deployment of the robots and there is no subsequent adaptation –at least, not through evolution– of the controllers. Evolution is orchestrated by an overseer outside the robots themselves, typically an external computer. The population of controllers undergoes selection and variation inside this computer and fitness can either be evaluated in simulation (again inside this computer), or *in vivo* by uploading the controller onto a real robot that uses it for a while to collect information on controller quality. While the latter is often referred to as 'embodied evolution', strictly speaking it only amounts to embodied fitness calculations: the evolutionary operators for selection and variation are not embodied in the robots. We investigate on-line evolution as implemented in the $(\mu + 1)$ ON-LINE algorithm by Haasdijk et al. (2010), where individual robots run a self-contained evolutionary algorithm on-board.

To evaluate individuals in the on-board population, $(\mu + 1)$ ON-LINE employs a time-sharing mechanism where individuals are evaluated sequentially by being given control of the robot and measuring robot performance during a pre-defined evaluation period. Because each of the individuals actually controls the robot for some time, the robot's overall task performance (the *actual* performance) is determined by the quality of all the genomes it considers, not only the best in the population it has on board. Thus, for on-line evolution of robot controllers average performance matters more than the quality of the best individual in the population in this setting: if a robot time and again evaluates poor controllers, its actual performance will be inadequate, no matter how good the best known individuals as archived in the population. $(\mu + 1)$ ON-LINE uses re-evaluation to deal with the inherent noisiness of *in vivo* evaluations, exacerbated by the effects running controllers sequentially without repositioning the robot or otherwise being able to ensure that the system is in the same or even similar state from one evaluation to another (Bredeche et al., 2009; Haasdijk et al., 2010).

Racing was introduced by Maron and Moore (1997) as a model selection technique that tests a set of models in parallel, quickly discards those models that are clearly inferior and concentrates the computational effort on differentiating among

the better models. We propose to use racing as a procedure to cut short the evaluation of particularly un-promising candidate controllers before the end of the full evaluation period. When using racing, a candidate solution can be evaluated for at most τ time steps, but the evaluation is aborted prematurely if there is strong evidence (as defined by equations 5.2 and 5.3) that the candidate will be discarded anyway. Obviously, this imposes two requirements on the evaluation process: it has to be possible to calculate an intermediate fitness during the evaluation period and these intermediate results must be comparable to the final evaluation results of solutions that did make the cut.

In evolutionary algorithms with lengthy fitness evaluations (e.g., in constraint satisfaction problems or data mining applications), one can similarly use racing to abandon the evaluation of un--promising individuals —as long as one has intermediate results during evaluation that compare with those of completed evaluations. In any such evolutionary algorithm, racing would speed up the evolutionary search process because more candidates can be evaluated in the same time frame. In on-line evolution, however, it offers an additional and potentially greater benefit: as noted above, time spent evaluating poor individuals is time spent performing poorly at the actual task. Therefore, cutting short the evaluation of poor individuals not only speeds up the search for good controllers: it should also increase the quality of actual control.

Using autonomous on-line evolution as we envisage, in situations where no human intervention is possible, the evolutionary algorithm in the robots must be able to optimise under unforeseen and possibly very different conditions. Unfortunately, the performance of evolutionary algorithms is, in general, quite dependent on their settings, so it requires the capability to calibrate itself on-the-fly (parameter control mechanisms cf. Eiben et al. (1999b)) or parameter settings that perform well in any circumstances. This brings us to a further reason for our interest in racing: the evaluation period τ was shown to be an influential parameter of the $(\mu + 1)$ ON-LINE algorithm in Haasdijk et al. (2010). We hope that our investigations into racing may point the way to a robust control scheme for this influential parameter, but that goal itself lies beyond the scope of this section.

Research Question The main question we ask in paper, then, is whether the robot's actual performance does indeed increase when employing racing. Secondly, racing has parameters of its own, and we analyse the impact of these parameters on the algorithm's performance. To this end, we conduct an experimental com-

parison of $(\mu + 1)$ ON-LINE's performance with and without racing on a number of robot tasks.

5.3.2 Related work

As mentioned above, traditional evolutionary robotics uses a conventional evolutionary algorithm to find good controllers in a fashion that can be identified as off-line, off-board (i.e., the evolutionary algorithm does not run inside the robots themselves), and overseen by a computer that executes the evolutionary operators (variation and selection) centrally. If, by contrast, evolution should run on-line, *after* deployment, i.e., as the robots carry out their tasks, one can recognise two approaches:

The encapsulated evolution approach is on-line, on-board, and centralised. Each robot has a self-sufficient evolutionary algorithm implemented on-board, maintaining a population of genotypes inside itself. The robots run these (possibly different) evolutionary algorithms locally and perform the fitness evaluations autonomously. This is typically done in a time-sharing system, where one member of the inner population is activated (i.e., decoded into a controller) at a time and is used for a while to gather feedback on its quality. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary algorithms running in parallel on the individual robots (Walker et al., 2006; Haasdijk et al., 2010). The $(\mu + 1)$ ON-LINE algorithm we consider here falls into this category;

The distributed evolution approach can be described as on-line, on-board, and distributed. Each robot has a single genotype and is controlled by the corresponding phenotype. Robots can reproduce autonomously and asynchronously and create offspring controllers by recombining and/or mutating their genotypes. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary process that emerges from the exchange of genetic information among the robots (Watson et al., 2002).

These approaches can be combined, resulting in a set-up akin to an island model as used in parallel genetic algorithms. In such a combined system, each robot is an island and genetic information is exchanged through intra-island variation (i.e., within the population of the encapsulated evolutionary algorithm in one robot) and inter-island migration (between two or more robots) (Nehmzow, 2002;

Usui and Arita, 2003; Wischmann et al., 2007; Perez et al., 2008; Elfwing et al., 2005; Haroun Mahdavi and Bentley, 2006).

Racing Our implementation of racing to cut short some individual's time share is based on Yuan and Gallagher (2007)'s use of Hoeffding racing to reduce the cost of parameter tuning: they used racing for early elimination of candidate algorithms while finding out which algorithm out of a set of candidates performs best on a particular benchmark problem, achieving a cost reduction of around 90% with negligible loss of reliability.

Similar applications of racing have been proposed for evolutionary algorithms in constraint satisfaction problems, e.g., by Bowen and Dozier (1995) and genetic programming for symbolic regression or data mining by Teller and Andre (1997).

Note, that we do not propose racing as a mechanism to deal with noisy fitness evaluations –$(\mu + 1)$ ON-LINE already uses re-evaluation to that end. We add racing simply to maximise the time spent evaluating promising solutions.

5.3.3 Racing in $(\mu + 1)$ on-line

In essence, the $(\mu + 1)$ ON-LINE algorithm is an evolution strategy (Beyer and Schwefel, 2002) that employs standard evolutionary algorithm operators (selection, variation and recombination) on a population of size μ to develop a new individual. That new individual –the challenger– is then evaluated by letting it take control of the robot for τ time steps and measuring the robot's task performance over that period. If the challenger's performance proves better than that of the worst in the population, the challenger replaces the current worst and the next iteration commences.

As mentioned above, we propose racing to abandon the evaluation of less than promising individuals. This means that during a challenger's evaluation, intermediate results are compared to the fitnesses of individuals already in the population to estimate the likelihood that the challenger is good enough to become part of the population. If it is fairly certain that this challenger is going to turn out worse than the worst in the current population, further evaluation is most likely a waste of time, especially with an elitist replacement scheme as in the $(\mu + 1)$ ON-LINE algorithm.

At intermediate time steps during a challenger's evaluation, its intermediate fitness $F_{challenger}$ is compared to a lower bound which depends on the current worst

fitness in the population F_{worst}. If the performance drops below this lower bound, the evaluation is aborted and a new iteration of the algorithm commences, otherwise, evaluation continues—at least until the next comparison. To estimate the likelihood that the challenger is at least as good as the current worst in the population, we use a modified version of the Hoeffding (1963) inequality:

$$F_{challenger} \geq F_{worst} - 2\xi(t) \tag{5.2}$$

with $\xi(t)$ calculated as follows:

$$\xi(t) = \sqrt{\frac{(F_a - F_b)^2 log(2/\alpha)}{\beta t}} \tag{5.3}$$

with F_a and F_b the best and the worst fitness values of the population, respectively; α is the significance level of the comparison. In Yuan and Gallagher (2007), there is no parameter β (instead, there is a fixed value 2). We introduce β to allow more robust tuning of the comparison's strictness. Using formulas 5.2 and 5.3, an individual's evaluation has a large likelihood of continuing in the early stages of evaluation (high values of ξ lower the bar), but the pressure increases as time passes.

Note, that racing in this manner requires that intermediate results are available and can be compared with 'proper' evaluation results. Subsection 5.3.4 gives examples of how we achieve this in our experiments.

As mentioned above, $(\mu + 1)$ ON-LINE implements re-evaluation to combat the effects of noisy evaluations and changes in the environment: with some probability ρ, an evaluation slot can be assigned to an unmodified individual in the population to refine and update its performance assessment. During re-evaluations no racing takes place: this would prevent proper reassessment of individuals already in the population that are no longer as good (or as bad) as they once were.

Algorithm 3 describes the result of adding racing as described to the original $(\mu + 1)$ ON-LINE algorithm.[v]

[v] Source code for the algorithm as well as the experiments described here is available at `http://www.few.vu.nl/~ehaasdi/papers/MuPlusOneAndRacing`

```
for i ← 1 to μ do                                              // Initialisation
    population[i] ← CreateRandomGenome();
    population[i].σ ← σ_initial;
    population[i].Fitness ← RunAndEvaluate(population[i]);
for ever do                                                    // Continuous adaptation
    if random() < ρ then        // Don't create offspring, but re-evaluate
    selected individual
        Evaluatee ← BinaryTournament(population);
        Evaluatee.Fitness ← (Evaluatee.Fitness +
        RunAndEvaluate(Evaluatee)) / 2;
        // Combine re-evaluation results through exponential moving
           average
    else              // Create offspring and evaluate that as challenger
        ParentA ← BinaryTournament(population);
        ParentB ← BinaryTournament(population - parentA);
        Challenger ← AveragingCrossover(ParentA, ParentB);
        Mutate(Challenger);
        a ← population[1].Fitness;
        b ← population[μ].Fitness;
        for n ← 1 to τ do          // Racing: monitor evaluation, abort if
        challenger appears worse than worst in population
            Challenger.Fitness ←
            RunAndEvaluateForOneTimeStep(Challenger);
            ξ ← sqrt((a−b)²log(2/α) / βt) ;
            if Challenger.Fitness < population[μ].Fitness - 2ξ then
                abort evaluation;
        if Challenger.Fitness > population[μ].Fitness then    // Replace last (i.e.
        worst) individual in population w. elitism
            population[μ] ← Challenger;
            population[μ].Fitness ← Challenger.Fitness;
    Sort(population);
```

Algorithm 3: The $(\mu + 1)$ ON-LINE evolutionary algorithm with racing

5.3.4 Experimental comparison

We compare the performance of $(\mu + 1)$ ON-LINE with and without racing on three tasks that are described below.

The $(\mu + 1)$ ON-LINE parameters μ, ρ and τ are set to the best obtained values for each task after a modest amount of tuning. To assess the influence of racing's

α and β parameters, we try several combinations of settings. The common settings for all three experiments are summarised in Table 5.2.

Common experiment details	
Simulation length	10,000 seconds (simulation time)
Number of repeats	20
Evolution details	
Representation	Real-valued vector with $-4 \leq x_i \leq 4$
Fitness	See task descriptions
Mutation	Gaussian $N(0, \sigma)$
Mutation step-size	Derandomised self-adaptive
Crossover	averaging
Parent selection	binary tournament
Crossover rate	1.0
Survivor selection	replace worst in population if challenger is better
Racing settings	
α	0.5, 0.7, 0.9
β	0.5, 1.5, 2

Table 5.2 – Experimental set-up

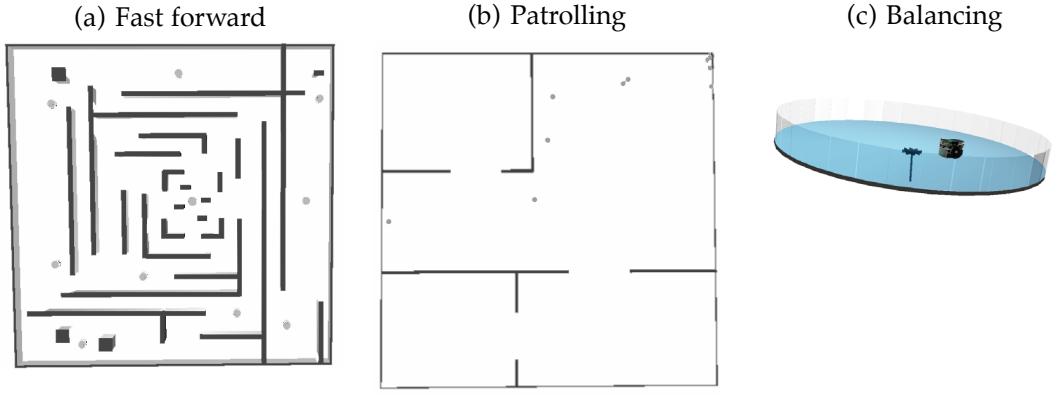


Figure 5.16 – Arenas for the tasks; where applicable, the circles represent the robots to scale.

5.3.4.0.1 Fast Forward Fast forward –moving in as straight line as possible as fast as possible while avoiding obstacles– is maybe the most common task in evolutionary robotics research. In a confined environment with obstacles this task

implies a trade-off between avoiding obstacles and maintaining speed and forward movement. The fitness function we use has been adapted from (Nolfi and Floreano, 2000); it favours robots that are fast and go straight ahead. Equation 5.4 describes the fitness calculation:

$$f = \sum_{t=0}^{\tau} (v_{trans} \cdot (1 - v_{rot}) \cdot (1 - d)) \tag{5.4}$$

where v_{trans} and v_{rot} are the translational and the rotational speed, respectively. v_{trans} is normalised between -1 (full speed reverse) and 1 (full speed forward), v_{rot} between 0 (movement in a straight line) and 1 (maximum rotation); d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

Intermediate results sum the reward from $t = 0$ to the current time step; they can then be compared with complete evaluation results after simply dividing both results by the number of time steps used to calculate them.

Although fast forward is considered a trivial task, here some extra difficulty is added by using a complicated maze-like arena (Figure 5.16(a)) with tight corners and narrow corridors that fit only a single robot and sometimes lead to dead ends. This arena structure, compounded by the fact that multiple robots will be simultaneously deployed, makes the task considerably harder than commonly seen instances.

Experiment details	
Robot group size	10
Controller details	
NN type	Simple perceptron
Input nodes	8 obstacle sensors + bias;
Output nodes	2 (left and right motor values)
($\mu + 1$) on-line settings	
Chromosome length	18
Evaluation time τ	300 time steps
Re-evaluation rate ρ	0.6
Population size μ	6

Table 5.3 – Fast forward experimental set-up

5.3.4.0.2 Collective Patrolling An obvious real-world task for a group of autonomous mobile robots is that of a distributed sensory network, where the robots have to patrol an area as a group and detect events that occur periodically. It differs from the previous tasks since it requires some level of co-ordination: the success of the group depends not only on the efficient movement of the individual robots but also on the spread of the group across the arena to maximise the probability of detecting events. Somehow, robots need to liaise so as not to patrol the same areas. To this end, they are equipped with a pheromone system: robots continuously drop pheromones (this is a fixed behaviour and not controlled by the evolved controller) while sensors detect the local pheromone levels. The collective patrolling task is described in (Martinoli, 1999) where controllers evolve off-line, although in that work the approach to events is more complicated and the robots use other sensory inputs.

The experiments for this task take place in the arena shown in Figure 5.16(b).

Every $T_e = 50ms$ with probability $p_e = 0.0005$, an event occurs at a random location with a duration of $d_e = 500 + \mathcal{N}(0, 2)$ seconds. Thus, in one run (10,000 seconds) approximately 100 events occur and that at any time around 5 events are active in the whole arena.

A robot detects an event in a 360^o circle whenever it comes within $0.3m$ of the event, so a robot's sensory coverage is $0.283m^2$. Since the arena is $25m^2$, a group of 10 robots can at any moment cover at most 11% of the whole arena; conversely, a group of stationary robots should detect around 11% of the events.

Pheromones are simulated as follows: the $5m \times 5m$ arena is divided into 500×500 cells, each with a pheromone level between $[0, 2]$. Every second, each robots drops 1 unit of pheromones at the cell the robot is currently in, and a linearly decreasing amount in nearby cells up to a range of $R_p = 0.07m$. Pheromone levels decay over time at a rate of $R_c = -0.024/s$ in each cell.

As sensory input to the controller, 4 pheromone sensors are placed at the periphery of the circular body at $\frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$. Each sensor detects the accumulated pheromone levels of all cells in a range of $0.05m$ (with detected levels decreasing linearly with distance from the sensor). For fitness calculation only, a similar sensor is positioned on the center of the robot.

The fitness function penalises pheromones presence (detected by the central pheromone sensor) and proximity to obstacles:

$$f = \sum_{t=0}^{\tau}((1-p)\cdot(1-d)) \tag{5.5}$$

where p indicates pheromone presence between 0 (no pheromones) and 1 (strongest pheromone level) at the current location and d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

Just as in the fast forward scenario, intermediate results sum the reward from $t = 0$ to the current time step; they can then be compared with complete evaluation results after simply dividing both results by the number of time steps used to calculate them.

Although movement is not included explicitly, it should emerge due to the continous dropping of pheromones and the deleterious effect of staying in a place where the robot just dropped them.

Experiment details	
Robot group size	10
Controller details	
NN type	Simple perceptron
Input nodes	8 distance sensors + 4 pheromone sensors + bias
Output nodes	2 (left and right motor values)
$(\mu + 1)$ on-line settings	
Chromosome length	26
Evaluation time τ	300 time steps
Re-evaluation rate ρ	0.6
Population size μ	10

Table 5.4 – Patrolling experimental set-up

5.3.4.0.3 Balancing In this experiment, a single robot finds itself on a tray which is balanced on a thin spike as shown in Fig. 5.16(c). The centre of mass of the tray is positioned well below the tray itself (as if a weight on a stick were attached to the bottom), so it tilts smoothly with the distribution of weight on its surface, which depends only on the position of the robot on the tray's surface. Around the edge of the tray is a barrier that prevents the robot moving off the tray. The robot's

task is obvious: balance the tray so that it is horizontal and it is easy to see that this equates to "stay in the centre of the tray." The task is similar to pole balancing or inverted pendulum, but it is much harder because the robot's movement are in two dimensions rather than one.

The fitness of an individual is the average angle of the tray with the z-axis (so a horizontal tray is at 90°) during its evaluation except when touching the surrounding barrier. To encourage moving away from the barrier, the fitness is 0 for any time step when the robot touches the barrier.

As inputs, the controller has the values from 8 distance sensors, 3 coordinates from a gps sensor (which returns $0, 0, 0$ at the tray centre) and the compass heading. For fitness evaluation only, the robot has an accelerometer sensor to measure the tray's tilt.

Contrary to the other two tasks, we use only a single robot in each experiment because having multiple robots on the same tray greatly increases the task complexity with an individual's fitness depending on the aggregate behaviour of the group rather than only its own.

Because the fitness already is averaged over the evaluation evaluation period, no further normalisation is required to compare intermediate and final results.

Experiment details	
Robot group size	1
Controller details	
NN type	Perceptron with 2-node hidden layer
Input nodes	8 distance sensors + 3 gps co-ordinates + compass heading + bias
Output nodes	2 (left and right motor values)
($\mu + 1$) on-line settings	
Chromosome length	30
Evaluation time τ	300 time steps
Re-evaluation rate ρ	0.2
Population size μ	10

Table 5.5 – Balancing experimental set-up

5.3.5 Results and Discussion

To answer this section's main question –does the robot's actual performance increase when employing racing– figures 5.17(a) to 5.17(c) show box plots comparing the performance (normalised between 0 and 1) for $(\mu + 1)$ ON-LINE runs without racing ("native") to runs with racing with optimal settings for α and β. The comparisons are over 20 repeats; each box extends from the 25th to the 75th percentile with the central mark at the median; the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. The notches indicate the 95% confidence interval for the median: interval endpoints are the extremes of the notches, so if two intervals do not overlap, we may conclude that the difference between the corresponding two medians is significant with 95% certainty.

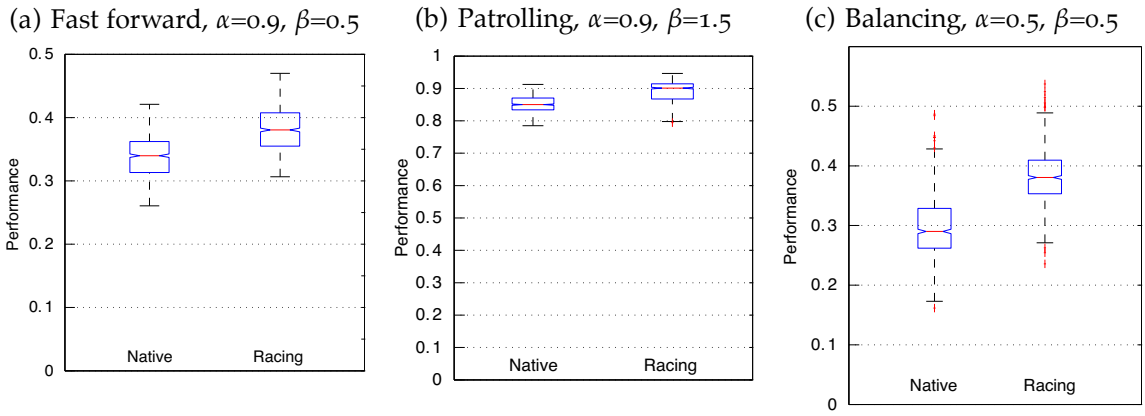


Figure 5.17 – Median performance with and without racing on the three tasks over the last 60 minutes of simulation (approx.).

Figures 5.17(a) to 5.17(c) show that racing –with the best found settings for α and β– improves performance significantly in all three tasks: for each task, even the lower quartile with racing is well above the 95% confidence interval for the median performance without racing. Improvement ranges from nearly 6% for the patrolling task to over 27% for the balancing task.

To assess the influence of racing's parameters α and β, figures 5.18 to 5.20 show the performance for each α, β combination we considered in our trials. The graphs are divided into four sections: one (leftmost) shows a box plot for the orginal $(\mu + 1)$ ON-LINE implementation, the three sections to the right of that –one for each α setting– each show boxplots for the β values we considered. The horizontal

band across the graphs denotes the 95% confidence interval for the native $(\mu + 1)$ ON-LINE median performance.

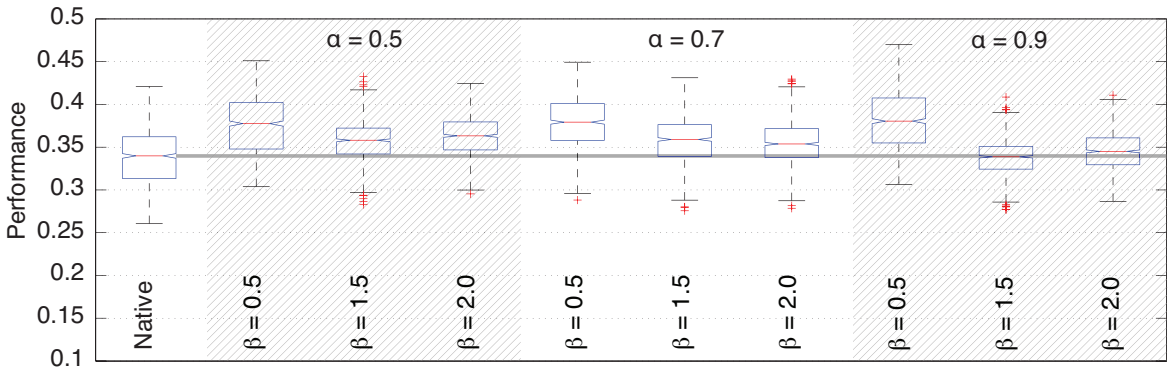


Figure 5.18 – Median performance on the fast forward task over the last 60 minutes of simulation (approx.) for all considered combinations of α and β.

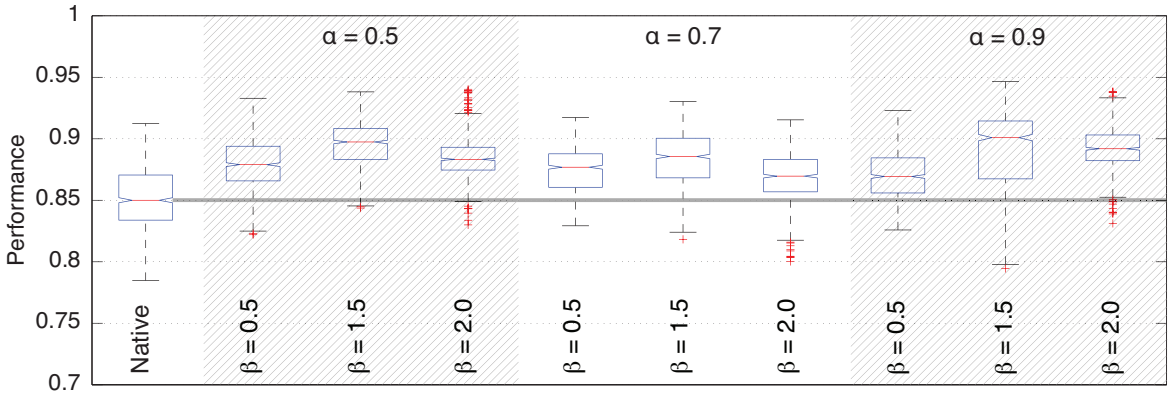


Figure 5.19 – Median performance on the patrolling task over the last 60 minutes of simulation (approx.) for all considered combinations of α and β.

The plots show significant differences among combinations of α and β, but only for two combinations is the performance even comparable to that of the runs without racing, in all other cases it is significantly better. Also note that in all sections for the different α values, at least one combination ranks among the highest performing for that problem. This seems to indicate that, even though α is the first parameter one thinks of considering the theoretical background of Hoeffding's inequality, β actually is the prime parameter in this application: it seems that we can safely set α to a fixed value and trust that some value of β will combine with that setting to yield (near-)optimal performance. Looking at figure 5.21, which shows

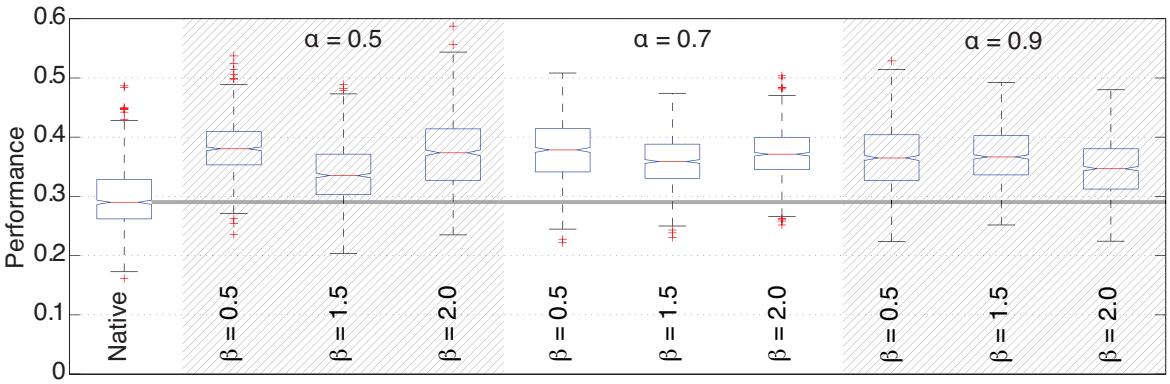


Figure 5.20 – Median performance on the balancing task over the last 60 minutes of simulation (approx.) for all considered combinations of α and β.

the development of $\xi(t)$ as an evaluation progresses, this is borne out by the difference in influence of α and β on the curve's progress. The $\xi(t)$-curves for differing β values vary considerably more than those for various α settings and it is easy to see that any reasonable change in α (denoting a probability, it should be between 0 and 1) can be overruled by changing β.

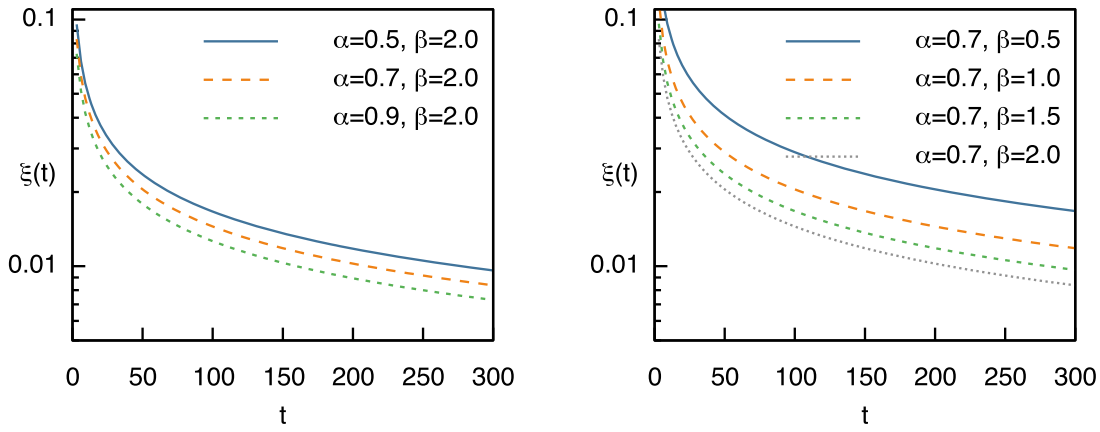


Figure 5.21 – Influence of α and β on $\xi(t)$ with $a = 1.0$ and $b = 0.8$. Note the logarithmic scale for $\xi(t)$.

5.3.6 Conclusions

First and foremost, our experiments show that adding racing to the on-line evolutionary process can improve the actual performance of the robots significantly,

with the median performance improving from nearly 6% for the patrolling task to over 27% for the balancing task.

For the balancing task in particular it must be said, however, that the robot in no case robustly comes to grips with this task beyond staying away from the edges of the tray, so even though racing improves performance by as much as 27%, it doesn't give adaptation that extra push needed to solve the balancing problem properly. In this task, we observed a subtle dynamic that makes it extra challenging for continuous adaptation: once a robot has learned to stay in the centre of the tray, at some point it will try a candidate that causes it to move away from the middle, and then controllers that previously worked very well (e.g., moving around in small circles) suddenly are no good at all and the robot has to re-learn moving towards the centre all over again. This causes tremendous fluctuation in performance in almost all balancing runs, but more importantly highlights an inherent problem for continuous evolution: the adaptive process has to remember how it solved a problem (in this case, moving to the centre from near the edge) even though other behaviour (here: standing still or driving in small circles) was appropriate over a possibly large number of evaluations. This reminds the authors of problems in dynamic fitness landscapes and we hope to find inspiration in that field to tackle this issue.

Given our remark in the introduction that on-line, on-board evolution requires parameter control schemes or robust settings, adding racing to $(\mu + 1)$ ON-LINE can be seen as a bad thing because it means adding two parameters α and β and the experiments' results show that the setting for α and β do impact performance. However, in all but two parameter combinations we tried, median performance increased significantly vis-à-vis the vanilla $(\mu + 1)$ ON-LINE implementation without racing. Moreover, our analysis suggests that, in fact, one of racing's parameters, α may be set to a fixed value so that only β remains to be tuned or controlled.

Of course, our findings very likely depend on the tasks we investigated as well as our choice of controller – as far as we know, the field of evolutionary robotics lacks a taxonomy of robot tasks or controllers that allow for a meaningful generalisation of our findings. Still, the results are promising and warrant further research into racing as a method of improving performance in on-line evolutionary robotics. As already noted introduction, a potentially greater benefit could be that racing may help us find a robust method for control of the influential τ parameter

that sets the evaluation period. Our future work in this area focusses on this latter aspect.

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" (I found it!) but "That's funny..."

Isaac Asimov

6 The Proof of the Pudding

Analysing an Encapsulated Algorithm

6.1 Introduction

Imagine a collective of autonomous robots that find themselves in a dynamic environment that they (or rather, their designers) didn't know to expect. Obviously, they cannot rely on pre-defined behaviour determined by fixed controllers to cope robustly with the unknown challenges in such a scenario. Rather, the robots have to learn to cope with their environment and any changes in it, just as they have to learn to react to changes in their own bodies, for instance as a result of hardware failure or reconfiguration. An essential capability, therefore, of such robots

---

This chapter was submitted as:

Evert Haasdijk, S.K. Smit and A.E. Eiben. Exploratory Analysis of an On-line Evolutionary Algorithm in Simulated Robots, to appear in *Evolutionary Intelligence*, Springer-Verlag, Berlin/Heidelberg.

---

is the ability to adapt their controllers – to learn – in the face of challenges they encounter in a hands-free manner, without supervision – human or otherwise.

Our research is inspired by the vision of robots one day being able to adapt like this as so eloquently articulated by Nelson et al. (2009):

> *"Advanced autonomous robots may someday be required to negotiate environments and situations that their designers had not anticipated. The future designers of these robots may not have adequate expertise to provide appropriate control algorithms in the case that an unforeseen situation is encountered in a remote environment in which a robot cannot be accessed. It is not always practical or even possible to define every aspect of an autonomous robot's environment, or to give a tractable dynamical systems-level description of the task the robot is to perform. The robot must have the ability to learn control without human supervision."*

To provide such levels of autonomous adaptivity through evolution, an evolutionary algorithm must run on-board, without any external master computer to execute evolutionary operators or evaluations. Crucially, the robots' controllers change on the fly, as they go about their proper tasks: adaptation occurs *on-line*, during the operational period of the robots.

The overwhelming majority of evolutionary robotics research to date, however, involves *off-line* adaptation during the development stage prior to the operational period. Additionally, the evolutionary algorithm that develops the robot controllers actually doesn't run on the robots themselves at all, but it runs on a separate computer that centrally maintains the population of robot controllers (or rather, their genetic encodings) and performs selection and genetic operators. The robots themselves – be it real or simulated – only come into play to evaluate candidate controllers. Once the controllers are deployed on the actual robots genuinely to perform their tasks, the evolutionary process is terminated and the controllers are modified no more – at least not through evolution. Nolfi and Floreano (2000) provide ample illustration of this approach to evolutionary robotics. Without detracting from the impressive work with this approach to controller evolution, it cannot provide the on-line adaptive capabilities that we envisage.

Essentially, the off-line approach to evolutionary robotics is no different from other fields of evolutionary computation as we have known them since the 1960s — Evolution Strategies, Genetic Algorithms, Genetic Programming, Differential

Evolution, to name but a few. Traditional evolutionary robotics and mainstream evolutionary algorithms share the centralised management of evolutionary operators, where would-be parents do not select mates and produce offspring in autonomously, but are being selected by an 'oracle' (the outer loop of the main evolutionary algorithm) and undergo variation operators passively.

For the on-line adaptive capabilities without supervision as we envisage, this is not an appropriate scheme. For that, the adaptation mechanism – the evolutionary algorithm, in our case – must operate in a decentralised, autonomous fashion. We distinguish three ways in which the evolutionary algorithm can be implemented on the robots themselves without recourse to some external overseer that orchestrates evolution or assesses fitness:

Encapsulated In this case, the complete evolutionary algorithm runs within the robot itself. In effect, the robot orchestrates its own evolution without referral to other robots or external assessments of its performance;

Distributed In systems with multiple robots, each robots contains only a single individual. Robots select mates from their neighbours and autonomously exchange genetic material to generate new candidate solutions that they test themselves;

Hybrid The two previous approaches can be combined: robots now maintain a self-sufficient evolutionary algorithm on-board, but they also exchange genetic material with their neighbours – like islands with migration in the well-known island model of parallel evolutionary algorithms.

The subject of this study is an algorithm that embraces the first approach: the $(\mu + 1)$ ON-LINE algorithm (introduced in detail in Sec. 6.3) encapsulates an evolutionary algorithm within each individual robot, thus enabling it to evolve its controller on-the-fly, independently from the others.

Rather than simply testing the algorithm and reporting on the outcomes, we adopt the scientific testing approach advocated by Hooker (1995). This approach aims at "learning about your algorithm", rather than "testing your algorithm" and emphasises the analysis of robustness and parameter interactions. These different aspects and their relevance to evolutionary robotics are discussed in Sec. 6.4. In our analysis of $(\mu + 1)$ ON-LINE we use an automated parameter tuning method

called Bonesa that is specifically designed for this purpose. Its basic concepts are explained in Sec. 6.5.

Through scientific testing, we want to investigate which of $(\mu + 1)$ ON-LINE's parameters have the most profound impact on the robots' performance in a variety of tasks, which parameter settings work well in what kind of task and which settings (if any) work well overall.

The purpose of our tuning exercise is not simply to find the best parameter vector for a particular task and environment, but it is to learn about the algorithm's applicability in a range of circumstances, to identify parameters that require problem-specific tuning and to gain insights into algorithm behaviour that will allow for proper deployment in real hardware with either well-established, tuned, parameter values or with (as yet to be developed and validated) parameter control schemes.

Such a thorough exploration of the algorithm's parameter space requires a very large number – literally thousands – of experiments. For reasons of time, but also because of expected hardware failures due to wear and tear in so many runs, it is not possible to perform these experiments with real robots, and we have to turn to simulated trials for our analysis. Obviously, we cannot expect the performance of real robots to be the same as that of simulated robots (the reality gap), or even that parameter settings found to be optimal will be the best settings when deploying $(\mu + 1)$ ON-LINE in real robots. However, based on our extensive experience with evolutionary computing, we expect that such simulations do give good indications about algorithm behaviour such as sensitivity to parameter settings and identification of important parameters, allowing for focussed, more tractable numbers of experiments with real robots in subsequent research.

6.2 On-line, On-board Evolutionary Robotics

In their seminal paper, Watson et al. (2002) coin the phrase *Embodied Evolution* for a system where "a population of physical robots [...] autonomously reproduce with one another while situated in their task environment." In other words, the controllers evolve on-line ("in their task environment") through the physical robots exchanging genetic material with one another. This groundbreaking example of the distributed approach introduces a fully autonomous scheme where the robots broadcast (mutated) genes on local-range communication channels at

a rate proportional to their fitness (Probabilistic Gene Transfer Algorithm, PGTA). Also, robots resist 'infection' with genes broadcast by other proportionally to their fitness. Robots incorporate received genes into their own genome and so immediately update the controller and continually evaluate its performance. In this scheme, there is no central authority or global view of the population, but there is, in fact, one item of global information: the maximum possible fitness which is used to determine the broadcast and resistance rates. This seems to limit the applicability of PGTA, but it would seem reasonably straightforward to propagate at least the best achieved fitness through a gossiping-like algorithm as described in Jelasity et al. (2005) and so avoid the need for a pre-specified maximum fitness. Wischmann et al. (2007); Simões and Dimond (2001); Nehmzow (2002) describe similar distributed implementations of on-line evolutionary robotics.

Before focussing on encapsulated implementations, let us first note that evolutionary algorithms that embrace the encapsulated approach do not seem materially different from 'regular' evolutionary algorithms because the whole process runs on a single robot, just as it would run inside a single computer. There are, however, particularities of on-line evolution as we will see in Sec 6.3. One practicality of on-line evolution (and of the encapsulated variant in particular) is *time-sharing*; obviously, only a single controller can be active at any one time on a single robot. Therefore, the encapsulated algorithm can only evaluate a single individual (which defines a single controller) at a time. To evaluate multiple individuals – as any evolutionary process must – the algorithm has to try them in sequence, one after the other: a controller runs for a certain amount of time and the task performance over that period determines the controller's fitness. Then, the next controller is loaded and gets its chance to control the robot while fitness is measured, and so on. As a consequence, there can be no guarantee that two individuals are evaluated in the same or even similar circumstances: a controller's evaluation starts where the previous left off, and there is no re-initialisation, repositioning or other mechanism to ensure similar circumstances of evaluation. As Nordin and Banzhaf (1997) note in their description of one of the earliest implementations on-line evolution of robot controllers:

> *"Each individual is thus tested against a different real-time situation leading to a unique fitness case. This results in 'unfair' comparison where individuals have to navigate in situations with very different possible outcomes. However, our experiments show that over time averaging tendencies of this learning*

> *method will even out the random effects of probabilistic sampling and a set of good solutions will survive."*

Floreano et al. (2002) implement an encapsulated algorithm that evolves spiking neural networks for obstacle avoidance. This implementation illustrates two characteristics that are common to almost all encapsulated and hybrid implementations of on-line, on-board evolutionary algorithms. Firstly, the algorithm implements steady-state evolution: an individual is tested and possibly replaces an individual in the current population so that parents and offspring may exist side-by-side. In this case, the new individual replaces the worst in the population if it performs better. Secondly, the population on a single robot (consisting of only 6 individuals) is tiny compared to what is common in evolutionary computation; after all, it has to fit inside the robot's on-board memory.

Walker et al. (2006) show that, in addition to providing hands-free adaptivity, on-line, on-board evolution can help overcome the 'reality gap' where a simulator irons out the wrinkles and warts of reality. Often, when robot controllers are developed using a simulator (which is typically cheaper and faster than using real robots), the controllers in reality perform worse than might be expected due to infidelities in the simulation and particularities of individual hardware (see Brooks, 1992). Walker et al. add a second stage to mainstream evolutionary robotics to tackle this issue. They first employ an off-line, centralised evolutionary algorithm to develop controllers in simulation (the 'training phase'). They then transfer the resulting controllers onto a real robot, and implement an on-line, on-board evolutionary algorithm that further refines the controller and adapts it to a changing environment. In their experiments, the robot has to avoid randomly placed obstacles while moving towards a goal. The environment changes by moving the obstacles and by varying the number if obstacles in the arena. The dynamics of the environment highlight the hands-free adaptivity that on-line, on-board evolution enables. Walker et al.'s on-line algorithm has a single champion genome that serves as parent to a challenger; the challenger replaces its parent if it proves to perform better. To this fairly common arrangement they add a buffer memory: a second child can replace the challenger (but not the parent directly) if it outperforms it. Every iteration, the second child is replaced with a newly mutated version of the parent, so the first child is the current best challenger. This two-tiered approach was designed to overcome the problem mentioned above: as a result of sequential evaluation, the circumstances of evaluation could differ signif-

icantly from one individual to the next and 'a poor chromosome could perform uncharacteristically well and be rewarded and vice-versa.' The population of the on-line algorithm is – again – tiny: only three individuals are considered, in all. Walker et al. mention that this carries the benefit of promoting rapid adaptation to changes in the environment: 'the smaller the population size, the faster each generation of chromosomes is evaluated and the sooner the effects of evolution manifest.'

Haroun Mahdavi and Bentley (2006) describe experiments involving encapsulated on-line evolution in robots that use shape memory alloy actuators: metal filaments that change shape when a small current is applied. In one set of experiments, Haroun Mahdavi and Bentley use on-line evolution to let a snake-shaped robot learn to move forward. These experiments don't actually exhibit the level of autonomy suggested by on-line, on-board evolution because the fitness was not measured by the robot itself but by the experimenters (and therefore evolution took place, at least in part, off-board), but they, nonetheless, provide a striking example of one of the benefits of on-line adaptation: during one experiment, one of the filaments broke while the robot was evolving a gait. Normally, this would spell disaster and mean that the robot must be fixed and the experiment restarted. Haroun Mahdavi and Bentley, however, realised that this was an excellent opportunity to see the controller adapt to the new circumstances and continued the experiment with one broken actuator. In short order, the robot adapted its gait to move with one less actuator, showing how on-line evolution can help robots cope with hardware failure. Similar robustness under on-line, on-board adaptation was reported by Christensen et al. (2010), who purposely introduced hardware faults – for instance, failing modules in a modular robot body – during experiments with an on-line, on-board stochastic adaptation method akin to Newton-Raphson minimisation methods.

6.3 The $(\mu + 1)$ on-line Evolutionary Algorithm

The $(\mu + 1)$ ON-LINE algorithm as described by Haasdijk et al. (2010) and Bredeche et al. (2009) was devised specifically to provide autonomous adaptation through on-line, on-board evolution. It is an encapsulated evolutionary algorithm: a robot maintains a self-sufficient evolutionary process on-board, without recourse to other robots or to some central overseer. It is inspired by evolution strategies (Bäck,

1996; Schwefel, 1995); these provide a likely basis for $(\mu + 1)$ ON-LINE because the the robot controllers' parameters in our experiments can be represented by a vector of real-valued numbers and evolution strategies have a very good reputation as evolutionary solvers of numerical optimisation problems (Bäck, 1996). Other encapsulated algorithms for on-line adaptation that share this basis are described by Haroun Mahdavi and Bentley (2006); Nehmzow (2002); Walker et al. (2006).

The main features of $(\mu + 1)$ ON-LINE algorithm are the following: 1) it relies on a population of μ individuals that produce one child per generation, 2) it uses a time-sharing scheme to evaluate the fitness of individuals (robot controllers), 3) it allows for re-evaluation to reduce noise, 4) it can self-adapt the mutation parameters and the length of the evaluation period. The exact details are given in the pseudo code in Algorithm 4. The main design decisions and the underlying rationale are discussed in the following subsections.

6.3.1 Evolutionary operators

In typical applications of evolutionary algorithms, the overall success is determined by the best individual at termination. The quality of the other individuals considered during evolution is of no importance as they will be discarded when the champion is deployed. Things are very different for on-line evolution as we envisage here: because the controllers evolve as the robots go about their tasks the real performance of the robots is determined by the quality of *all* individuals (controllers) they evaluate. While a robot evaluates poor controllers, that robot's *actual* performance will be inadequate, no matter how good the best known individuals as archived in the population.

In evolution strategies, it is common to describe algorithms using a pair μ, λ where μ indicates the size of the population and λ indicates the number of offspring generated at each iteration. The '+' indicates that the algorithm uses an elitist replacement strategy where new individuals are only introduced into the population if they outperform current members of the population. Thus, $(\mu + 1)$ ON-LINE generates $\lambda = 1$ child per cycle, which then may replace the worst in the population if it achieves higher fitness. Normally in evolution strategies $\frac{\lambda}{\mu}$ is usually between 4 and 8, but the cost of evaluating poor individuals (about which more below) is so high that we choose to set $\lambda = 1$. To promote rapid convergence we diverge from the common practice of uniform random parent selection

in evolution strategies and use binary tournament parent selection, increasing the selective pressure. Also, we use recombination to promote convergence. Thus, each new individual is created from two parents, each of which is selected with a binary tournament. When the representation allows (it does in all the tests we conducted), we apply the evolution strategy standard of gaussian mutation with self-adaptation of the mutation step-sizes. We consider various self-adaptation schemes as described in Sec. 6.6.

6.3.2 Re-evaluation to combat noise

In on-line evolution, fitness must be evaluated *in vivo*: the quality of any given controller can only be determined by actually giving that controller the run of the robot and see how well the robot performs its tasks. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances: any controller's evaluation will start wherever and in whatever state the previous evaluation left the robot. The very dissimilar evaluation conditions caused by one –possibly very poor– individual setting the scene for the evaluation of another individual result in very noisy ("unfair" in the words of Nordin and Banzhaf (1997)) fitness assessments.

To level the playing field among genomes, $(\mu + 1)$ ON-LINE re-evaluates genomes in the population with a given probability ρ. This means that every evolutionary cycle one of two things happens: either a new individual is generated and evaluated (with probability $1 - \rho$), or an existing individual is re-evaluated (with probability ρ). To ensure that re-evaluation efforts are spent more or less on those individuals that actually become parents, the individual to be re-evaluated is chosen by a binary tournament from the whole population. The fitness values from subsequent (re-)evaluations of any given individual are combined using an exponential moving average as advocated by Bredeche et al. (2009); this emphasises newer performance measurements and so is expected to promote adaptivity in changing environments. Bredeche et al. also show that using an exponential moving average outperforms alternative averaging strategies.

Re-evaluating in this manner is similar to a resampling strategy to deal with noisy fitness evaluations as advocated by Beyer (2000), although the moving average introduces a temporal aspect and it makes convergence to the true fitness

less likely (although, in a dynamic environment, that would be problematic in any case).

```
for i ← 1 to μ do                                              // Initialisation
    population[i] ← CreateRandomGenome();
    population[i].σ ← σ_initial;
    population[i].Fitness ← RunAndEvaluate(population[i]);
Sort(population);
for ever do                                                    // Continuous adaptation
    if random() < ρ then          // Don't create offspring, but re-evaluate
        Evaluatee ← BinaryTournament(population);
        Evaluatee.Fitness ← (Evaluatee.Fitness +
        RunAndEvaluate(Evaluatee)) / 2;  // Combine re-evaluation results
    else                  // Create offspring and evaluate that as challenger
        Challenger ← BinaryTournament(population);
        if random() < P_c then
            ParentB ← BinaryTournament(population - Challenger);
            AveragingCrossover(Challenger, ParentB);
        Mutate(Challenger);    // Also updates σs depending on adaptation
        scheme
        a ← population[1].Fitness;
        b ← population[μ].Fitness;
        for n ← 1 to τ do          // Racing:  monitor evaluation, abort if
        challenger fails
            Challenger.Fitness ←
            RunAndEvaluateForOneTimeStep(Challenger);
            ξ ← sqrt((a−b)²log(2/α) / βt) ;
            if Challenger.Fitness < population[μ].Fitness - 2ξ then
                abort evaluation;
        if Challenger.Fitness > population[μ].Fitness then    // Replace last (i.e.
        worst) individual
            population[μ] ← Challenger;
            population[μ].Fitness ← Challenger.Fitness;
    Sort(population);
```

Algorithm 4: The $(\mu + 1)$ ON-LINE evolutionary algorithm.

6.3.3 Racing to shorten fitness evaluations

An often heard criticism of stochastic search methods in general and evolutionary algorithms in particular is that they are computationally inefficient because they spend so much time evaluating poor candidate solutions. This is especially painful in the context of on-line evolution, because the actual performance of the evolving system (the robot's controller, in this case) drops when these sub-optimal solutions are evaluated.

To minimise the amount of time spent trying less than promising individuals, Haasdijk et al. (2011b) suggested adding *racing* to $(\mu + 1)$ ON-LINE. This entails that during a new individual's evaluation, intermediate results are compared to the fitnesses of individuals already in the population to estimate the likelihood that the challenger is good enough to beat the worst individual in the population and replace it. If it is fairly certain that this challenger is going to turn out worse than the worst in the current population, further evaluation is most likely a waste of time with an elitist replacement scheme such as $(\mu + 1)$ ON-LINE uses. What exactly 'fairly certain' means is determined by two parameters α and β as described below. At intermediate time steps during a challenger's evaluation, its intermediate fitness $F_{challenger}$ is compared to a lower bound which depends on the current worst fitness in the population F_{worst}. If the performance drops below this lower bound, the evaluation is aborted and a new iteration of the algorithm commences, otherwise, evaluation continues—at least until the next comparison. To estimate the likelihood that the challenger is at least as good as the current worst in the population, we use a modified version of the Hoeffding inequality (Hoeffding, 1963):

$$F_{challenger} \geq F_{worst} - 2\xi(t) \tag{6.1}$$

with $\xi(t)$ calculated as follows:

$$\xi(t) = \sqrt{\frac{(F_a - F_b)^2 log(2/\alpha)}{\beta t}} \tag{6.2}$$

with F_a and F_b the best and the worst fitness values of the population, respectively; α is the significance level of the comparison. The β parameter, introduced by Haasdijk et al. (2011b), allows more robust tuning of the comparison's strictness than the original bounds, where it has a fixed value of 2. Using formulas 6.1 and

6.2, an individual's evaluation has a large likelihood of continuing in the early stages of evaluation (high values of ξ lower the bar), but the pressure increases as time passes.

Putting all this together – and assuming real-valued vectors as genotypes representing robot controllers – we obtain an evolutionary algorithm with the following main components and parameters:

Evolutionary Algorithm Components	
Representation	Real valued vectors
Mutation	Adding Gaussian noise and self-adapting σ's
Crossover	Averaging parents
Parent selection	Binary tournament
Survivor selection	Replace worst on improvement
Evolutionary Algorithm Parameters	
Population size	μ
Crossover rate	p_c
Evaluation period	τ
Re-evaluation rate	ρ
Significance level for racing	α
Strictness level for racing	β

Table 6.1 – Main components and parameters of $(\mu + 1)$ ON-LINE

6.4 Scientific Testing

For many years, so called *horse-race-papers* (Johnson, 2002) have dominated the field of evolutionary computing. Researchers showed how their algorithm outperformed some other algorithm on some test-suite. These papers shed light on the question 'whether a certain algorithm instance can outperform another algorithm instance on a particular problem'. They do not, however, indicate when or why this is the case, nor do they give an indication of an algorithm's performance on other problems.

Hart and Belew (1991) noted that the fact that an algorithm performs well on a certain problem is no guarantee that it also works on a different one. Even worse, the fact that an algorithm with a particular set of parameter values works well on

a certain problem gives no indication of its performance on any other function. This is especially painful because common parameter settings – for instance, a population size of 100 – tend to be much less robust than assumed. In practice, parameter values are mostly selected by conventions (mutation rate should be low), ad hoc choices (let's use uniform crossover), and experimental comparisons on a limited scale (testing combinations of three different crossover rates and three different mutation rates) rather than based on solid foundations. Until recently, there were not many workable alternatives for such manual tuning.

However, developments over the last couple of years have resulted in a number of automated tuning methods and corresponding software packages that enable evolutionary computation practitioners to perform a more detailed analysis of the algorithm, and the appropriate parameter values, without much effort.

Performance is the most straightforward and most commonly used measure of algorithm quality. Because performance varies markedly with parameter settings, with the problem type and even with the inherent stochasticity of evolutionary algorithms, it is often better to consider the robustness of algorithms.

There are different interpretations of the notion of robustness in the literature. The existing (informal) definitions do have a common feature: robustness is related to the variance of algorithm performance across some dimension, but they differ in what this dimension is. Below, we take a closer look at different type of robustness considered in literature.

Problem Sensitivity In the simplest case, we test an evolutionary algorithm A on a single problem f. Then, the performance of an instance of A (with a specific parameter vector) can be measured as the performance of A on f. It might be the case that A is very good at solving f, however, there can be no claims or indications regarding its performance on other problem instances. This can be a satisfactory result if one is only interested in solving f. However, algorithm designers in general, and evolutionary computing experts in particular, are often interested in (instances of) evolutionary algorithms that work well on many different problems. In such cases, tests need to be performed on a test suite consisting of many different problems $f_1, \ldots, f_n$.

Parameter Influence Another popular interpretation of algorithm robustness is related to performance variations caused by different parameter settings. Most algorithms are sensitive to parameter settings, which means that the parame-

ters need to be set carefully. An ill-chosen parameter value may cause a huge drop in performance. Such behaviour is often undesirable, especially if the algorithm is also sensitive to changes of the problem. In that case, a lot of effort is needed to identify the correct parameter values for each situation. It is then important to know how sensitive the algorithm is to each of the parameters, in order to focus the search for good settings. If, on the other hand, the algorithm is quite robust against changes in the problem space, a sensitivity in the parameter space may not be a big issue. The only requirement is that such an algorithm is carefully tuned beforehand to identify the "sweet spot" for parameter values; these settings will then work well for a large range of problems.

Variance across different random seeds Evolutionary algorithms are inherently stochastic. Therefore, all experimental investigations should be statistically sound, requiring a number of independent repetitions of a run with the same setup, but with different random seeds. This yields information about the third kind of robustness. To what extent an algorithm's performance varies from one instance to the next, varying only the random seed, of course greatly influences its applicability.

6.5 Bonesa

Bonesa (Smit and Eiben, 2011) is an iterative model-based procedure to search noisy response surfaces where evaluations can be very expensive. Like other search methods using surrogate models (Jin, 2005; El-Beltagy et al., 1999), it is based on an intertwined searching and learning loop. Bonesa has been developed for automated parameter tuning of (evolutionary) algorithms, based on ideas behind REVAC (Nannen and Eiben, 2007) and Sequential Parameter Optimisation (Bartz-Beielstein et al., 2005).

The pseudo code in Algorithm 5 describes the initialization, the two loops and their interaction. First, M random parameter vectors are generated and tested to obtain an initial model of the landscape for each of the problems at hand. Then, the search and learning loops start.

The search loop is a generate-and-test procedure that iteratively generates K new parameter vectors, pre-assesses their quality using the information gained in

```
for i ← 1 to M do                                              // Initialisation
    archive[i] ← CreateRandomParameterVectors();
    archive[i].RawQualities[] ← RunAndEvaluate(archive[i], problems);
i ← M ;
while maximum budget not reached do    // Intertwined Searching and Learning
    i ← i + 1;
    archive.PredictedQualities[] ← KernelFilterMean(archive, problems);
    archive.PredictedVariances[] ← KernelFilterVariance(archive,
    problems);
    for j ← 1 to K do                                        // Generate Candidates
        candidates[j] ← DrawFromArchiveDensity(archive) ;
        candidates[j].PredictedQualities[] ←
        KernelFilterMean(candidates[j], problems);
        candidates[j].PredictedVariances[] ←
        KernelFilterVariance(candidates[j], problems);
        candidates[j].PredictedParetoRank ←
        CalculateParetoRank(candidates[j], archive);
    Sort(candidates, candidates.PredictedParetoRank);
    archive[i] ← candidates[1] ;
    archive[i].RawQualities[] ← RunAndEvaluate(archive[i], problems);
for j ← 1 to i do                                       // Create termination set
    archive[j].ParetoRank ← CalculateParetoRank(archive[j], archive) ;
    if archive[j].ParetoRank == o then
        terminationset ← { terminationset, archive[j] };
```

Algorithm 5: The Bonesa algorithm.

the learning loop, and finally tests the most promising vector by executing a run with these specific parameter values. In its turn, the learning loop uses the information obtained about the quality of the tested parameter vector and all previously tested ones to develop a model of the performance surfaces. These performance surface models can then be used to pre-assess the K generated parameter vectors in the next search loop. Notice the two-way interaction between the two loops: information generated by the search loop is used to update the model, while estimations based on the model direct the search.

Pre-assessing the vectors with the model rather than blindly testing them greatly reduces the computational costs of tuning parameters. However, establishing the performance of the parameter vectors that do pass the model-based test is still computationally expensive. Namely, the result of a single run is often not representative due to the stochasticity of the evolutionary algorithm. Therefore,

several expensive runs of the evolutionary algorithm have to be executed in order to establish the real performance of a certain parameter vector; this still causes high tuning costs.

To reduce the noise caused by the stochasticity with the least computational costs, Bonesa uses a kernel filter. This is a common method of reducing noise in spatial data that does not depend on repetitive testing of the same point but uses proximity data (Haralick and Shapiro, 1992; Branke et al., 2001) to smooth out the noise. This allows Bonesa to test every parameter vector only once and still produce statistically sound results.

Compared to other currently known iterative model-based search procedures, Bonesa distinguishes itself by its multi-objective approach. Rather than optimising an algorithm's parameters for a single problem or on a weighted sum of performance values (Smit and Eiben, 2010; Pedersen and et al., 2008), Bonesa uses the Pareto-strength approach from SPEA2 by Zitzler et al. (2001) to optimise parameters for a whole range of problems in one go. Therefore, one is ultimately able to select not only the best parameter vector for a single problem, but also for a class of problems. In order to determine dominance, one important change has been made with respect to SPEA2: since the distributions of performance need to be compared rather than a single value, the calculation of dominance is based on significance testing. A certain parameter vector $\bar{z}$ dominates another parameter vector $\bar{y}$ if and only if its performance is significantly better (using a Welch-T test) on at least one problem, and not significantly worse on the others. This means that, also in case of single-problem tuning, the result is a set of 'best' vectors rather than a single one.

For a more detailed description of the Bonesa algorithm and toolbox we refer to Smit and Eiben (2011).

The most important feature of model-based tuners such as Bonesa is that rather than delivering only the 'perfect' parameter settings for each (set of) problem(s), they can analyse the complete spectrum of performance, robustness and parameter interactions, allowing for much deeper insights into the algorithm and its behaviour.

6.6 Experimental Set-up

We use Bonesa to optimise the following $(\mu + 1)$ ON-LINE parameters:

α The significance level of the comparison for racing (see Sec. 6.3.3). α can range from 0 to 1, with 0 disabling racing altogether.

β The second parameter that regulates the strictness of the racing comparisons. This can range from 0.5 to 2.0.

σ Or, rather, the choice of σ adaptation scheme. In $(\mu + 1)$ ON-LINE, new individuals are mutated using a gaussian mutation with step-size σ. We consider a number of ways to update σ: two 'standard' self-adaptive schemes, where either a single σ for the whole genome or multiple σs, one for each real-valued gene in the genome, evolve as additional part of the genome as described by Eiben and Smith (2008, pp. 75–77). The third scheme updates σ using the derandomised approach proposed by Ostermeier et al. (1994), who specifically deem this method useful with small populations. In the result plots, these schemes are identified as 'ss', 'ms' and 'dr', respectively.

To level the playing field among genomes, $(\mu + 1)$ ON-LINE re-evaluates genomes in the population with a given probability ρ. This means that every evolutionary cycle one of two things happens: either a new individual is generated and evaluated (with probability $1 - \rho$), or an existing individual is re-evaluated (with probability ρ). To ensure that re-evaluation efforts are spent more or less on those individuals that actually become parents, the individual to be re-evaluated is chosen by a binary tournament from the whole population.

ρ The probability that an existing individual is re-evaluated rather than that a new individual is generated and tested. ρ can range from 0 to 0.6.

τ The length of an episode during which a controller is given the run of the robot to evaluate it. To (re-)evaluate an individual, the genome is expressed in the controller (the weights of the neural net are set to those specified in the genome), which then determines the robot's actions over a certain number – τ – of discrete time-steps, unless evaluation is aborted by the racing procedure. τ can range from 100 to 1200.

P_c The crossover rate. The likelihood of performing recombination of two parents when producing a new individual. Ranges from 0 to 1.

The population size μ has been fixed at a value of 10 in these experiments. There are two reasons not to include μ as a tunable parameter: firstly, μ cannot be very large: in the kind of robotic hardware we consider, memory is at a premium, so there is simply no room for large populations. Moreover, a large population would necessitate a longer episode of testing the initial population, which is not desirable in the kind of on-line adaptation we envisage because it would mean very poor performance for a fairly long period with all its attendant risks. Secondly, we have seen that, within these limitations, μ seems not to be a very influential parameter (Haasdijk et al., 2010).

6.6.1 Four Tasks

To understand how $(\mu + 1)$ ON-LINE behaves across a range of problems, we test it on four tasks, all commonly found in evolutionary robotics literature: phototaxis, fast forward (or obstacle avoidance), collective patrolling and locomotion.

Even with Bonesa's optimisations to reduce the number of tests, this type of tuning is not feasible with real robots: it still requires hundreds of experiments to generate a reliable surrogate model. Therefore, we conducted our experiments in simulation using the Webots simulator[i]. Note that we do not advocate tuning as extensively as this for every deployment of an evolutionary algorithm, but we use tuning to help us understand algorithm behaviour. This understanding will allow for more focussed tuning for specific deployment or guide the development of parameter control algorithms to adapt parameters on-line.

In the phototaxis, fast-forward and patrolling experiments, the robots are simulated e-pucks, controlled by simple perceptron neural networks and the evolutionary algorithms determine the weights of the connections between the neurons. The perceptrons use a *tanh* activation function and receive inputs from light, distance and pheromone sensors (depending on the task) and have two output neurons that drive the wheels. All inputs are normalised in the $[0, 1]$ interval before being fed to the neurons. The outputs governing the e-puck wheel speeds are interpreted as fraction of the full speed for the motors. In each of these experiments, we simulate 10 robots in a single arena. In the phototaxis task, the robots do not interact in any way. In the fast-forward task, they pose additional, moving, obstacles for

[i] `http://www.cyberbotics.com/`

each other. The patrolling experiment adds communication between robots by spreading pheromones.[ii]

The locomotion experiment is based on that described by Christensen et al. (2010); it simulates 5 roombot modules that are linked to form a quadruped shape. This task requires the most intricate level of collaboration: the modules are physically linked and have to synchronise their movements to obtain an effective gait. In this experiment, the controller is not a neural network, but a central pattern generator.

In all cases, a single trial runs for 10,000 seconds of simulated time; we use time rather than number of evaluations or generations because we are interested in the performance of the robots in real time, regardless of how that is achieved by the evolutionary algorithm. As mentioned, τ is measured in discrete time steps; in fact, these time steps are defined by the simulator calling the robots controller at periodic intervals. The length of a single time step (in simulated time) is 50 ms for the the phototaxis, fast-forward and patrolling experiments. The more intricate physics of the locomotion simulations require a lower settings of 16 ms. Note that the settings for τ, α and β influence the number of evaluations performed per timespan: lower values of τ obviously increase the number of evaluations per timespan, just as increasing the strictness of racing comparisons (by increasing α and β). We are primarily interested in the actual performance of robots, not in the performance of the best individual in the population at any given time.

Actual performance is measured as the average performance during a timespan, irrespective of how many controllers may have been activate during that time. In these experiments we measure actual performance over the last 6,000 seconds of simulated time. For the phototaxis, fast-forward and patrolling experiments, this yields ten measurements for each trial (one per robot) as input for Bonesa. For the locomotion experiment, only a single value is reported because the distance travelled by the compound robot already combines the information for all constituent modules. The exact fitness functions are obviously task dependent and are described with each task, below.

Bonesa can be used for both single-problem tuning and multi-problem tuning. When doing multi-problem tuning, it uses the Pareto-strength approach from SPEA2 (Zitzler et al., 2001) to optimize on a whole range of different problems in

[ii]Source code for the algorithm as well as Webots configuration files for the experiments described here is available at `http://www.few.vu.nl/~ehaasdi/papers/MuPlusOne-2012`

one go. Since such a method terminates with a whole set of parameter-vectors, an experimenter can choose which of those suits his needs best. For example, certain parameter vectors could be well-suited for solving a subset of problems, rather than only a single problem. Most of these "silver bullets", would not have been found with a single-problem approach, since they are often outperformed by a true "specialist" on the problem. Often, "silver bullets" are much more interesting than true specialist because they lead to much more robust algorithm behaviour. Finally, such a multi-problem approach can be used to identify similar problems. If we assume that problems that can be solved with the same parameter-values can be regarded as "similar", we can try to determine if there are any characteristics that they have in common. Such knowledge can then be reused when facing new problems.

Phototaxis Seeking out or tracking a light source is a very straightforward task that has been addressed by many researchers in evolutionary robotics. The task is frequently combined with other tasks such as goal homing (Tuci et al., 2002) and flocking (Baldassarre et al., 2002). In our comparison, we use the simplest version of phototaxis: robots only have to move towards a stationary light source and then remain as close to it as possible. In the phototaxis task, the robots use eight light sensors to detect light intensity and base their behaviour on that. The fitness function simply rewards intensity of received light:

$$f = \sum_{t=0}^{\tau} \max_{i=1}^{8}(lightSensor_i) \tag{6.3}$$

where $lightSensor_i$ is the normalised input from a light sensor between 0 (no light) and 1 (brightest light).

The arena is an empty (apart from the ten robots) square with a light source in the middle: we ignore collisions between robots in these experiments, so the robots' distance sensors can be ignored.

Fast Forward Moving in as straight line as possible as fast as possible while avoiding obstacles – also known as obstacle avoidance – is maybe the most commonly tackled task in evolutionary robotics research. In a confined environment with obstacles this task implies a trade-off between avoiding obstacles and maintaining speed and forward movement. The fitness function we use is based on

Experiment details	
Task	phototaxis
Robot group size	10
Simulation length	10,000 seconds (simulation time)
Controller details	
Controller	Perceptron neural net
Input nodes	8 light sensors + bias
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length	18

Table 6.2 – Phototaxis set-up

one described by Nolfi and Floreano (2000); it favours robots that are fast and go straight ahead:

$$f = \sum_{t=0}^{\tau} (v_{trans} \cdot (1 - v_{rot}) \cdot (1 - d)) \tag{6.4}$$

where v_{trans} and v_{rot} are the translational and the rotational speed, respectively. v_{trans} is normalised between -1 (full speed reverse) and 1 (full speed forward), v_{rot} between 0 (movement in a straight line) and 1 (maximum rotation); d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

Intermediate results sum the reward from $t = 0$ to the current time step; they can then be compared with complete evaluation results after simply dividing both results by the number of time steps used to calculate them.

Although fast forward is considered a trivial task, we added some extra difficulty by using a complicated maze-like arena (Figure 6.1(a)) with tight corners and narrow corridors that fit only a single robot and sometimes lead to dead ends. This arena structure, combined with the fact that multiple robots will be simultaneously deployed, makes the task considerably harder than most commonly seen instances. This additional complexity is confirmed by results of baseline trials in this arena that Karafotias et al. (2011) reported: the baseline Braitenberg controllers invariably get stuck after a while.

Experiment details	
Task	fast forward
Robot group size	10
Simulation length	10,000 seconds (simulation time)
Controller details	
Controller	Perceptron neural net
Input nodes	8 distance sensors + bias
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length	18

Table 6.3 – Fast-forward set-up

Collective Patrolling An obvious real-world task for a group of autonomous mobile robots is that of a distributed sensory network, where the robots have to patrol an area as a group and detect events that occur periodically. It differs from the previous tasks since it requires some level of co-ordination: the success of the group depends not only on the efficient movement of the individual robots but also on the spread of the group across the arena to maximise the probability of detecting events. Somehow, robots need to liaise so as not to patrol the same areas. To this end, they are equipped with a pheromone system: robots continuously drop pheromones (this is a fixed behaviour and not controlled by the evolved controller) while sensors detect the local pheromone levels. The collective patrolling task is described by Martinoli (1999) where controllers evolve off-line, although in that work the approach to events is more complicated and the robots use other sensory inputs.

The experiments for this task take place in the arena shown in Figure 6.1(b).

Pheromones are simulated as follows: the $5m \times 5m$ arena is divided into 500×500 cells, each with a pheromone level between $[0, 2]$. Every second, each robots drops 1 unit of pheromones at the cell the robot is currently in, and a linearly decreasing amount in nearby cells up to a range of $R_p = 0.07m$. Pheromone levels decay over time at a rate of $R_c = -0.024/s$ in each cell.

As sensory input to the controller, 4 pheromone sensors are placed at the periphery of the circular body at $\frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$. Each sensor detects the accu-

(a) Fast forward arena (b) Patrolling arena

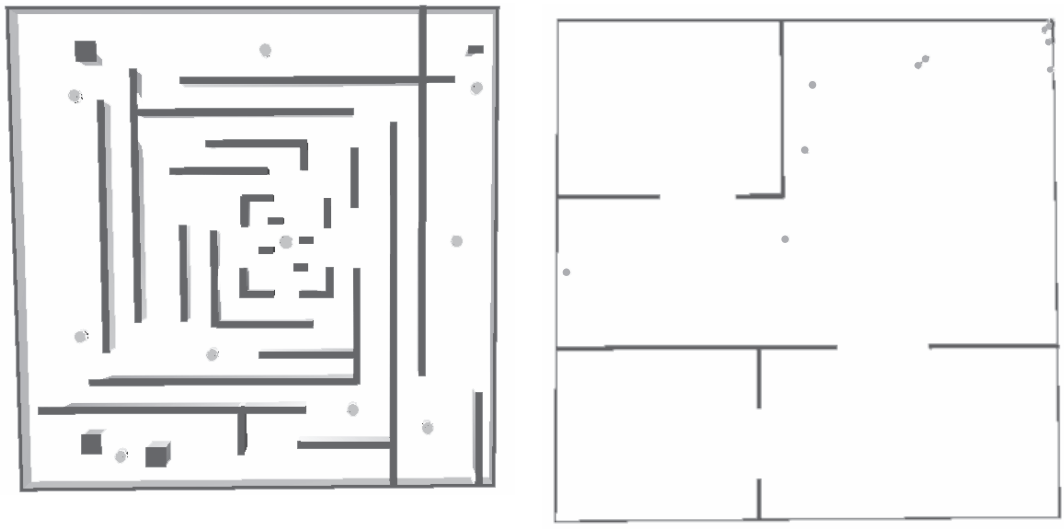


Figure 6.1 – Arenas for two tasks; the circles represent the robots to scale.

mulated pheromone levels of all cells in a range of $0.05m$; detected levels decrease linearly with distance from the sensor. For fitness calculation only, a similar sensor is positioned in the centre of the robot.

The fitness function penalises pheromones presence (detected by the central pheromone sensor) and proximity to obstacles:

$$f = \sum_{t=0}^{\tau}((1-p)\cdot(1-d)) \tag{6.5}$$

where p indicates pheromone presence between 0 (no pheromones) and 1 (strongest pheromone level) at the current location and d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

Although movement is not explicitly rewarded, it should emerge due to the continuous dropping of pheromones and the deleterious effect of staying in a place where the robot just dropped them. Just as in the fast forward scenario, intermediate results sum the reward from $t = 0$ to the current time step; they can then be compared with complete evaluation results after simply dividing both results by the number of time steps used to calculate them.

Experiment details	
Task	collective patrolling
Robot group size	10
Simulation length	10,000 seconds (simulation time)
Controller details	
Controller	Perceptron neural net
Input nodes	8 distance sensors + 4 pheromone sensors + bias
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length	26
Mutation	Gaussian $N(0, \sigma)$
Crossover	averaging

Table 6.4 – Patrolling set-up

Locomotion The locomotion task differs from the preceding three: it concerns individual robots that are physically linked together to form an *organism* that to all intents and purposes looks like a single entity. In fact, five Roombots robots form a four-legged organism as shown in Fig. 6.2.

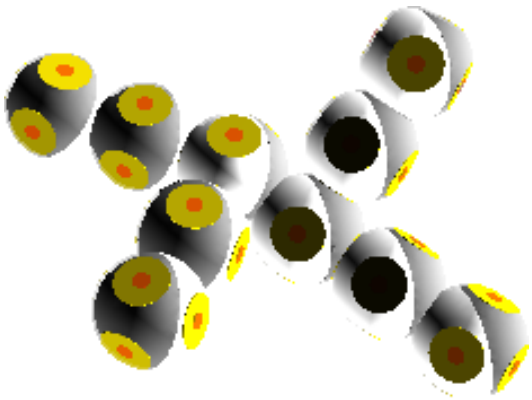


Figure 6.2 – Organism for the locomotion task. Each of the five consituent modules consists of two spheres. Each module has three degrees of freedom: it can rotate the two (dark and light) halves of each sphere and it can rotate the connection between the spheres.

Another difference is that the controllers are not neural networks but oscillators that operate the roombots' three degrees of freedom: rotating the two halves of each sphere and rotating the connection between the two spheres. Each module's controller runs with only local communication.

These oscillators are parameterised by a common base frequency and, for each of the three actuators, a phase shift, amplitude and offset that determine the actual movement: ten real values in total. Our choice for the genetic representation of

these parameters is obvious: a vector of these ten real values. Each module runs its own, independent, $(\mu + 1)$ ON-LINE instance to optimise the oscillation parameters. This implies co-evolution between the modules' controllers: they must evolve to some common ground to synchronise their oscillations.

The fitness function is very straightforward: it simply sums the Euclidean distance travelled per time-step. As before, intermediate results can be compared with complete evaluation results by simply dividing both results by the number of time steps used to obtain them.

This experiment was first described by Christensen et al. (2010), who used a non-evolutionary stochastic approximation method, SPSA, to implement on-line adaptation. We refer to that publication for a more detailed description of the robots and the coupled oscillator controller.

Experiment details	
Task	locomotion
Robot group size	5
Simulation length	10,000 seconds (simulation time)
Controller details	
Controller	Central Pattern Generator
Oscillator parameters	phase shift, amplitude, offset for each actuator, one common base frequency
Evolution details	
Representation	real valued vectors with $0 \leq x_i \leq 1$
Chromosome length	10

Table 6.5 – Locomotion set-up

6.7 Results

We review the experimental results in the light of three aspects of $(\mu + 1)$ ON-LINE's parameters: how problem-specific are good parameter settings, which parameters influence task performance most and how do the parameters interact?

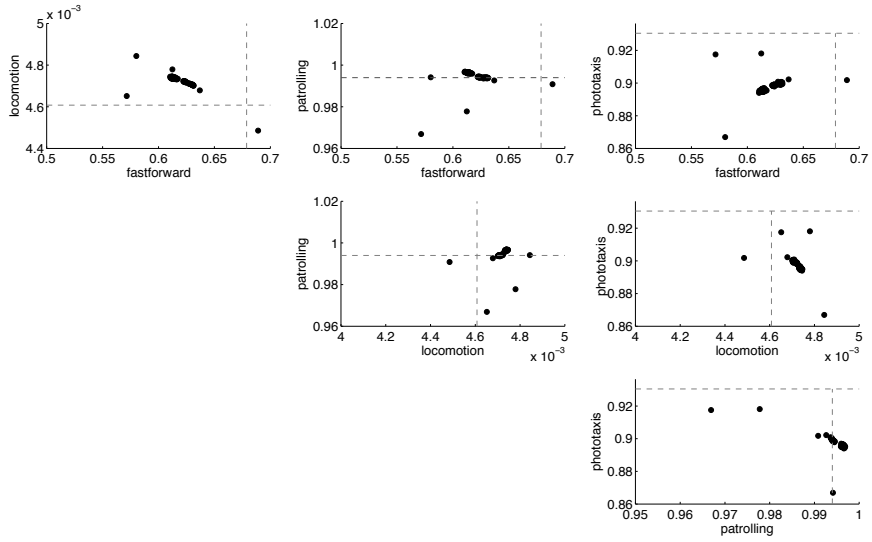


Figure 6.3 – 2-dimensional projections of the Pareto-front. Each plot shows the performance of all non-dominated parameter vectors from the multi-objective run on two tasks. The horizontal and vertical dotted lines indicate the maximum performance reached when tuning for that specific task only. Thus, points on or above/to the right of dotted lines indicate parameter vectors found with multi-objective tuning that match or outperform the best parameter vector found with task-specific tuning.

6.7.1 Generalism vs Specialism

The object of this chapter is not to prove that tuning leads to good results – that has been amply illustrated by, for instance, Eiben and Smit (2011). Nevertheless, we do look at the performance of the tuned instances of $(\mu + 1)$ ON-LINE to compare the results of tuning for a specific task with those of the multi-objective approach. This provides insight into the performance penalty of using generalist parameter settings (the result of multi-problem tuning) as opposed to specialist settings that result from tuning specifically for each problem.

From Fig. 6.3, we can see that for three of the four tasks – fast forward, patrolling and locomotion – multi-objective tuning carries no performance penalty. In fact, the locomotion performance of the multi-objective runs is often better than that of tuning specifically for locomotion. For phototaxis, however, the performance is slightly worse when tuning for all these four tasks at once. This seems to indicate that phototaxis requires parameter settings that conflict with the other tasks, something we will see more proof of later. We also see that parameter vectors that work well for the fast forward task also work well on locomotion and/or

patrolling, but that the converse is not necessarily true: the highest performance for patrolling is achieved by parameter vectors with indifferent performance on the fast forward task.

6.7.2 Parameter Tolerance

task	τ	σ	ρ	P_c	α	β
fastforward	**0.0262110**	0.0000177	0.0168660	<u>0.0253330</u>	0.0002089	0.0001982
phototaxis	<u>0.0000622</u>	0.0000010	**0.0002482**	0.0001162	0.0000287	0.0000371
patrolling	**0.0029074**	0.0000082	<u>0.0027625</u>	0.0000057	0.0000076	0.0000088
locomotion	**0.0000463**	0.0000009	<u>0.0000069</u>	0.0000025	0.0000025	0.0000029

Table 6.6 – The variance of the best performance across possible values for each of the parameters. The parameter with the highest variance for each problem is shown in bold, the second highest is underlined

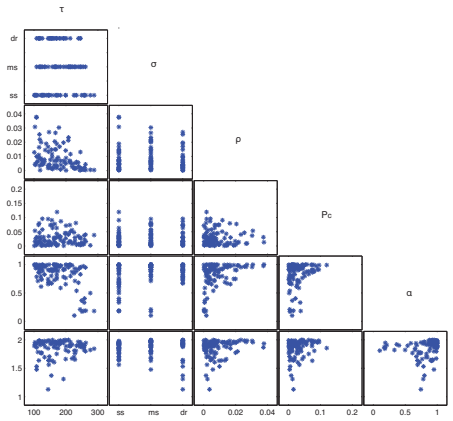
An important insight from tuning exercises like these is what parameters matter most for algorithm performance; for an analysis of this parameter tolerance, we turn to Table 6.6. It shows, per problem, the variance of the best performance over the range of each parameter. For each of the three σ-strategies, for example, the best performance achieved with that specific strategy is noted for each problem; the variance of these performances is divided by the difference between the best and worst achieved performance on that problem and then listed. Numerical parameters were binned into 100 intervals prior to this calculation. Large values therefore indicate a parameter that causes large fluctuation in performance – that has a large impact. Using the terminology introduced by Eiben and Smit (2011), this is a measure for both the tunability and the tolerance of a parameter.

It is clear that τ is the most sensitive parameter, since for all of the problems it has the highest or second-highest variance. This indicates that it needs to be set very carefully because it has the biggest influence on the performance. ρ comes in second place, and P_c on the third place. The σ-adaptation strategy, interestingly, has least effect on performance, except in the case of patrolling.

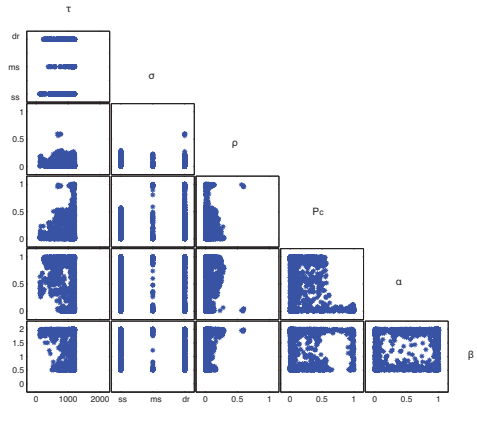
6.7.3 Parameter Interaction

Figure 6.4 shows the make-up of non-dominated parameter vectors in pairs of parameter values. For single-task experiments (plots 6.4(a) to 6.4(d)), non-domination

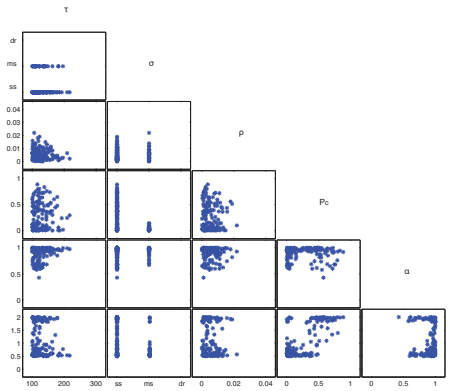
(a) fast forward



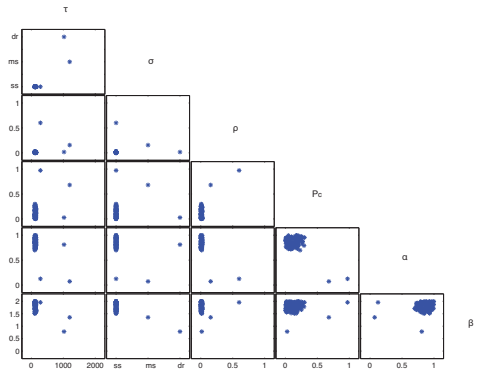
(b) phototaxis



(c) patrolling



(d) locomotion



(e) multi-objective

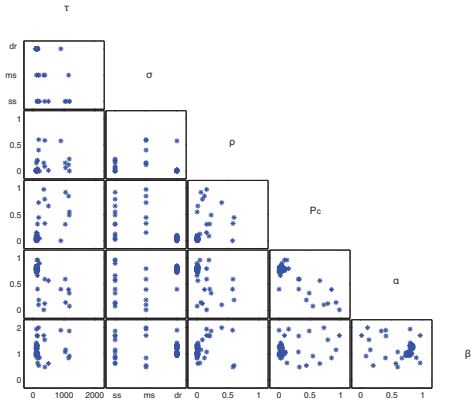


Figure 6.4 – Non-dominated parameter vectors matrices. Each matrix details the non-dominated parameter vectors for an experiment. A cell in one of the five matrices shows two entries of each non-dominated parameter vector, for instance τ vs. ρ in the first column, third row. Each dot indicates a combination of values that occurs in the non-dominated set.

is defined as not performing statistically worse (using a Welch-T test with $\alpha = 5\%$) than the best parameter vector found. For the multi-objective results in Fig. 6.4(e), Pareto dominance is used, in which "better" and "worse" is replaced with "significantly better" and "significantly worse".

Each dot represents a pair of values in a non-dominated vector; the top-left plot in each matrix, for instance, shows combinations of σ and τ; the dots' vertical position denotes the σ control scheme (single-step self-adaptive (*ss*), multi-step self-adaptive (*ms*) or the derandomised approach (*dr*)), their horizontal position shows the τ value.

As an example, consider Fig. 6.4(d). In the cell that shows combinations of τ and σ, we see that $\sigma = 0$ is much more prevalent than other values and that the non-dominated vectors combine that value with low values of τ. This means that, although it hardly influences the best possible performance level that can be reached (cf. Table 6.6), it does influence the sensitivity of the other parameters.

Fig. 6.4(d)'s plot for P_c vs α shows a rectangular region that contains the majority of dots; this indicates that no interaction between these two parameters was found and they can therefore be set to optimal values independently. This contrasts with a cell like that of ρ and α in Fig. 6.4(a), which shows interaction: there, if ρ is low, α may have any value and vice versa.

The results from the fast forward task in Fig. 6.4(a) show interaction between ρ and the racing parameters α and β: in those vectors where racing is less strict (low α and β), re-evaluation is all but turned off (although ρ is never very high for this task). There are similar interactions between racing and crossover with low values of α and β occurring only when P_c is almost zero. For this task, the strategy for updating mutation step-size seems immaterial. Note that the values for τ are very low: even the highest values are less than 300, while the full range extends to 1200. Within this relatively small range of τ values, there is an interesting pattern of interaction with ρ: the triangular shape of the region containing the value pairs seems to indicate a minimum number of new evaluations: if τ is lightly higher, ρ is less as if to compensate and vice versa.

Phototaxis (Fig. 6.4(b)) is by far the easiest task that the robots have to learn as borne out by the number of different non-dominated vectors: many parameter settings lead to near-optimal behaviour. It is the only task that seems to require – or should we say allow – lengthy evaluations (note that the scales differ from

problem to problem). Again, there is evidence of interaction between crossover and racing: racing may be almost off (low α), but only when P_c is high.

The patrolling task (Fig. 6.4(c)) is the first to show a clear preference when it comes to σ-adaptation schemes: the derandomised approach does not occur in the non-dominated set. In the cases where multi-step adaptation is selected, the crossover rate is very low. The results for this task show an interaction between τ and ρ similar to that for fast forward: if one is high (within the small range that occurs in the non-dominated vectors), the other is low.

The locomotion task is the hardest of the four tasks and it is hard to distinguish any interactions from Fig. 6.4(d). It is clear, though, that τ and ρ are mostly low, that racing is quite strict and that the preferred σ-adatation scheme is single-step self-adaptation.

Figure 6.4(e) shows the non-dominated parameter vectors when optimising settings for all four tasks at the same time. Here, we see a strong preference for strict racing, but only if P_c is low. Although there are far fewer points with higher crossover probabilities, there seems to be a trend to lower values for α and β as P_c increases. All three σ-adaptation schemes occur, but the derandomised scheme seems to require very specific settings for the remaining parameters: the dots for *dr* are very closely clustered around specific values for the other parameters.

6.8 Discussion

For all tasks except the easy phototaxis task, the best parameter settings tend towards low values for the re-evaluation rate ρ and evaluation length τ and towards settings that imply strict comparisons for racing (high α and β). These settings all indicate that it is preferable to try many points in the search space and not dwell too much on the effects of noisy or unfair evaluations.

From all this, one might conclude that re-evaluation is not a worthwhile feature of $(\mu + 1)$ ON-LINE. This seems at odds with the findings presented by Bredeche et al. (2009), where re-evaluation was shown to be beneficial in e-pucks evolving controllers for the fast-forward task. However, the tests there were conducted with $\mu = 1$, so the population could not as easily recover from genomes that were unluckily evaluated as very poor. Also, our experiments did not consider dynamic environments where higher population diversity could be essential - which in turn might imply a need for higher re-evaluation rates.

It is possible that unfair evaluations pose less of a problem in our experiments because the influence of an unrealistically high evaluation is limited: an individual that actually performs quite poorly may become part of the population, but enough properly good individuals remain to make sure that sufficient good offspring is generated. If this 'lucky' individual is selected as parent, its offspring will be cut short by racing (hence the high values for α and β) and so not even have lasting impact on the robot performance. This may also explain why P_c is set to low values: crossover increases the likelihood of such a bad individual contaminating the offspring of good individuals. Moreover, crossover acts as a macro-mutation (Jones, 1995), so low P_c values lead to a lower population diversity as illustrated in Fig. 6.5. This increases the likelihood that when a good individual is replaced unfairly with a bad one, a copy or at least a very similar individual remains to continue the bloodline.

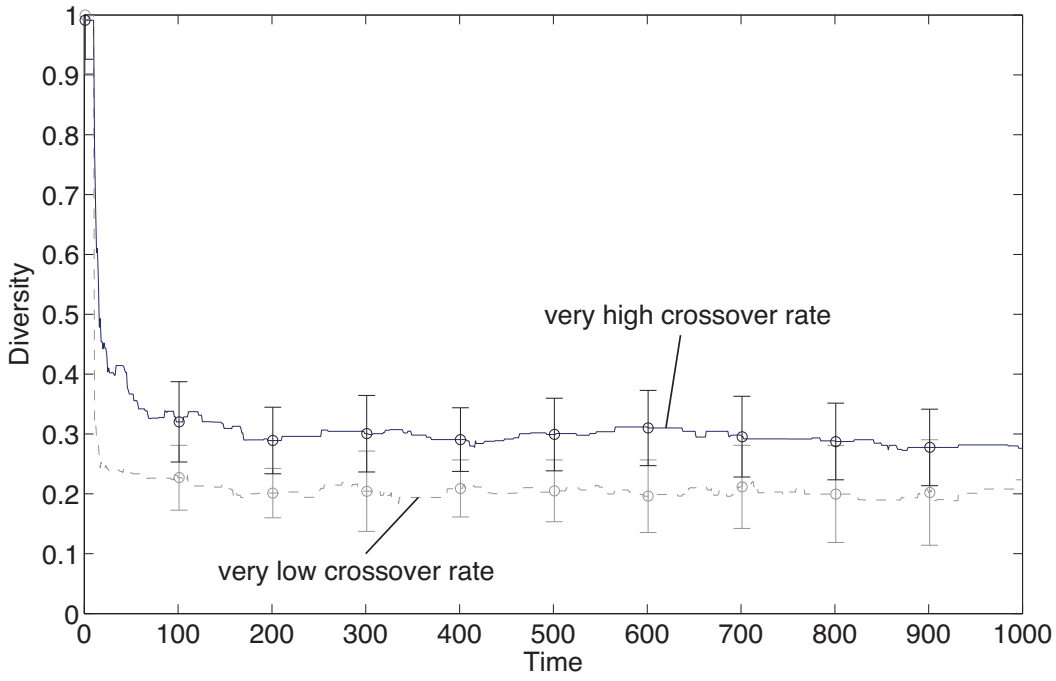


Figure 6.5 – Average population diversity over time for 10 runs of the fast forward task with low and with high P_c. Although the difference is not spectacular, a high crossover rate seems to increase diversity, suggesting macro-mutation effects.

We have seen that the phototaxis task requires parameter settings that conflict with those for the other tasks. We can only hypothesise what makes phototaxis so different – possibly the fact that it is so easy to solve. Be that as it may, we

can certainly conclude that there is no 'silver bullet' parameter vector that yields optimal $(\mu + 1)$ ON-LINE performance for all tasks.

Let us restate the vision that underlies our research into on-line, on-board evolutionary algorithms such as $(\mu + 1)$ ON-LINE: it is that of robots autonomously adapting to any circumstances, particularly ones that we cannot foresee. Without a 'silver bullet,' this requires robust control schemes that allow $(\mu + 1)$ ON-LINE or similar algorithms to change their parameters on the fly and so ensure universal adaptability.

6.9 Conclusion

In this chapter we investigated an evolutionary algorithm for developing robot controllers on-the-fly. Our study offered innovations in two different dimensions, algorithmically as well as methodologically. The $(\mu + 1)$ ON-LINE evolutionary algorithm is encapsulated within each individual robot, thus enabling the robots to evolve their controllers independently from each other. $(\mu + 1)$ ON-LINE has a number of essential properties: 1) it relies on a population of μ individuals that produce one child per generation, 2) it uses a time-sharing scheme to evaluate the fitness of individuals (robot controllers), 3) it allows for re-evaluation to reduce noise, 4) it can self-adapt the mutation parameters and the length of the evaluation period. Our experiments showed that this algorithm can make the robots develop appropriate controllers in a number of tasks, ranging from 'simple' individual tasks (obstacle avoidance) through swarm tasks with light coupling between robots (patrolling) up to a task where robots are very strongly coupled (organism locomotion).

Our experimental analysis of $(\mu + 1)$ ON-LINE showed that:

- It seems preferable to try many alternative solutions and spend little effort on refining possibly faulty assessments;
- There is no single combination of parameters that performs well on all problem instances, indicating a need for on-line parameter control schemes to achieve robust autonomous adaptivity;
- The most influential parameter of this algorithm – and therefore the prime candidate for a control scheme – is the evaluation length τ.

These conclusions identify issues that warrant further research: firstly, there is an apparent need for control schemes, in particular for the very influential τ parameter which defines the length of evaluations. Secondly, we hypothesise that re-evaluation is not needed with even as small a population size as 10; this should be investigated further, especially in circumstances where the environment and/or task is dynamic. As we already noted in Section 6.1, we cannot expect the results we find in our simulated trials to persist unaltered when $(\mu + 1)$ ON-LINE is applied in real robots. In that light, our results serve as a precursor for more focussed experiments to be conducted with real robots in subsequent research.

Methodologically, we deliberately deviated from the conventional horse-race approach: rather than just testing the algorithm and reporting the performances (perhaps comparing them to some benchmark), we adopted the scientific testing approach. This approach aims at "learning about your algorithm", rather than "testing your algorithm" and emphasises the analysis of robustness and parameter interactions. To this end, we used an automated parameter tuning method called Bonesa that is specifically designed for this purpose. Through scientific testing, we obtained valuable insights into $(\mu + 1)$ ON-LINE: which parameters have the most profound impact on the robots' performance in a variety of tasks, which parameter settings work well in what kind of task and which settings (if any) work well overall.

Maybe the most important conclusion from this research is a methodological one: analysing the make-up of the non-dominated parameter vectors as provided by Bonesa yields tremendous insight into the tuned algorithm. This insight can help us formulate new research questions and identify possibilities for improvement of the algorithm. These results of tuning are much more valuable than merely finding the best possible setting for some parameter on a particular problem or even on a range of problems.

Tous pour un, un pour tous

Alexandre Dumas

7

United We Stand, Divided We Fall

Distributed and Hybrid Approaches

The previous two chapters detailed the development of the $(\mu + 1)$ ON-LINE algorithm, which embraces the encapsulated approach and runs an entire, self-sufficient algorithm inside each individual robot without referring to any other robots. Viewing a collective of robots from the perspective of population-based adaptive systems introduced in chapter 3, such encapsulated evolution equates to individual learning. The fact that adaptation occurs through an evolutionary process is somewhat confusing, but when we view each robot as an individual in a population of agents, it is clear that they learn only individually, in splendid isolation.

An obvious question, then, is how such isolated adaptation compares to the more collaborative distributed and hybrid schemes – the fact that we performed many $(\mu + 1)$ ON-LINE experiments with multiple robots shows that this question was on our mind from the outset. Distributed evolution equates to social learning in chapter 3's framework, while the hybrid variant combines individual and social learning (by combining distributed and encapsulated evolution).

Section 7.1 is the result of our first efforts towards this end. It reports on the implementation of a distributed algorithm, EDEA, experimentally evaluates EDEA using a number of well-known tasks in the evolutionary robotics field to determine whether it is a viable implementation of on-line, on-board evolution. Finally, it compares EDEA to $(\mu + 1)$ ON-LINE in terms of (the stability of) task performance and the sensitivity to parameter settings. The Experiments show that EDEA provides an effective method for on-line, on-board adaptation of robot controllers. Compared to $(\mu + 1)$ ON-LINE, there is no clear winner in terms of performance, but in terms of sensitivity to parameter settings and stability of performance EDEA seems to improve significantly on its encapsulated counterpart.

Section 7.2 investigates another distributed implementation based on EVAG, a peer-to-peer evolutionary algorithm introduced by Laredo et al. (2010). Again, this is experimentally compared with the $(\mu + 1)$ ON-LINE algorithm. We find that distributed on-line on-board evolutionary algorithms that share genomes among robots such as our EVAG implementation effectively harness the pooled learning capabilities, with an increasing benefit over encapsulated approaches as the number of participating robots grows.

Section 7.3 considers a hybrid adaptation of the EVAG-based algorithm introduced in section 7.2. The hybrid approach implies a similarity to the island model as found in parallel evolutionary algorithm literature, and one of the issues in that field is that of migration policies: which individuals are exchanged between islands? We experimentally compare different migration policies in the context of hybrid on-line evolutionary robotics, and find that Araujo and Merelo's *multikulti* algorithm (2011) yields the best results by a narrow margin.

The final section of this chapter, section 7.4, takes the research in a different direction from earlier experiments. It considers an open-ended variant of evolution, reminiscent of artificial life experiments, for instance the poisonous food experiments from chapter 3. There is no explicit task set for the robots: they only have to maintain their (virtual) energy by charging regularly. They do this using the by now well-known EVAG-based algorithm to evolve controllers that amass energy. It is also different because it considers the aggregation of robot modules into 'organisms' that consist of multiple robotic modules that are physically connected. The environment of the robots is set up so that forming an organism is beneficial, but the reward is indirect: it becomes easier to gain energy. Experiments show that this does result in the emergence of multi-cellular robotic organisms.

7.1 Distributed On-line, On-board Evolutionary Robotics

This section presents part of an endeavour towards robots and robot collectives that can adapt their controllers autonomously and self-sufficiently and so independently learn to cope with situations unforeseen by their designers.

We introduce the Embodied Distributed Evolutionary Algorithm (EDEA) for on-line, on-board adaptation of robot controllers. We experimentally evaluate EDEA using a number of well-known tasks in the evolutionary robotics field to determine whether it is a viable implementation of on-line, on-board evolution. We compare it to the encapsulated $(\mu + 1)$ ON-LINE algorithm in terms of (the stability of) task performance and the sensitivity to parameter settings.

Experiments show that EDEA provides an effective method for on-line, on-board adaptation of robot controllers. Compared to $(\mu + 1)$ ON-LINE, in terms of performance there is no clear winner, but in terms of sensitivity to parameter settings and stability of performance EDEA is significantly better than $(\mu + 1)$ ON-LINE.

7.1.1 Introduction

Evolutionary computing techniques for optimisation and design have been used in robotics for well over a decade(Nolfi and Floreano, 2000). An overwhelming majority of the work in this field has focussed primarily on *off-line* evolution of robot controllers, where the evolutionary process takes place as a separate development phase before proper deployment of the robots and there is no subsequent adaptation –at least by evolution– of the controllers. Evolution is orchestrated by an overseer –an external computer– outside the robots themselves: the population of controllers undergoes selection and variation inside this computer. Fitness can either be evaluated in simulation (again inside this computer), or *in vivo* by uploading the controller onto a real robot that uses it for a while to collect infor-

---

Section 7.1 was published as:

Giorgos Karafotias, Evert Haasdijk and A.E. Eiben (2011). An Algorithm for Distributed On-line, On-board Evolutionary Robotics. In Natalio Krasnogor et al., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Pages 171–178, ACM, NY.

---

mation on controller quality. While the latter is often referred to as "embodied evolution", strictly speaking it only amounts to embodied fitness calculations; the evolutionary operators for selection and variation are not embodied in the robots.

We advocate a radically different approach to evolutionary robotics with genuine embodiment and on-line evolution. The essence of this approach is to implement evolutionary operators (selection, mutation, crossover) on-board and to evolve designs on the fly, as the robots go about their tasks (Eiben et al., 2010a). As explained in (Haasdijk et al., 2011a), this approach offers the necessary adaptivity in collective robotic systems to cope with a number of fundamental challenges:

- Unforeseen environments that are not fully known during the design period.
- Changing environments where the extent and/or type of the change make the pre-designed solutions inadequate.
- Reality gap, that is, the phenomenon that off-line design is based on approximations and simulations, necessitating that robots be fine-tuned to the real operational conditions after deployment.

The essence of the problem we address here is producing/adapting robot controllers on-the-fly, without humans in the loop. This problem is highly relevant in the light of the global trend of increasing adaptivity and autonomy of computer systems, including those running on mobile hardware.

Considering a swarm of robots, we can distinguish two approaches to on-line, on-board evolution (Eiben et al., 2010a):

Encapsulated evolution Each robot autonomously runs an independent evolutionary algorithm: each robot implements a centralised evolutionary algorithm and maintains a population of genomes using some time- sharing scheme to evaluate each controller;

Distributed evolution Each robot carries a single genome and uses that as its controller. The population comprises of the collection of the controllers of all robots and evolutionary operations take place in an autonomous and distributed manner by the robots interacting to exchange and recombine genetic material.

Obviously, these approaches can be combined to yield a hybrid approach where each robot runs an autonomous evolutionary algorithm and individual controllers

are transferred between robots, similar to the island model in parallel evolutionary algorithms.

We introduce the Embodied Distributed Evolutionary Algorithm (EDEA), a new algorithm that adopts the distributed approach, and experimentally compare it with the $(\mu + 1)$ ON-LINE algorithm as an exemplar of the encapsulated approach, which was described in detail in (Eiben et al., 2010a,b; Haasdijk et al., 2010). The first assessment of the new algorithm is based on task performance, using a number of well-known tasks in the evolutionary robotics field: phototaxis, obstacle avoidance, and collective patrolling.

Furthermore, we note that the robustness required for the robots' controllers is also required for the evolutionary algorithm that adapts the controllers: it, too, has to operate reliably under unforeseen and possibly very different conditions. Unfortunately, the performance of evolutionary algorithms is, in general, quite dependent on their settings (Eiben et al., 1999b). Hence, on-line evolutionary robotics requires an evolutionary algorithm with robust parameter settings that perform well over a wide range of problems, or an evolutionary algorithm that is capable of calibrating itself on-the-fly. Therefore, we evaluate the two algorithms not only in terms of performance, but also consider the number of parameters they have and the sensitivity to settings for these parameters (tuneability).

A third consideration is the stability of performance: reliable robot control requires consistently good or at least acceptable performance: a large variance in performance implies that an algorithm may perform well in one instance only to fail in another without any apparent difference in circumstances.

Summarising, the main objectives of this section are: 1) to introduce the EDEA algorithm and to determine whether it is a viable implementation of on-line, on-board evolution for producing robot controllers without humans in the loop, 2) to compare the task performance of the distributed approach with the encapsulated one (exemplified by EDEA and $(\mu + 1)$ ON-LINE, respectively), 3) to compare the robustness (i.e., parameter sensitivity) of the distributed and the encapsulated evolutionary algorithms.

7.1.2 Related Work

In this section we examine on-line on-board evolution (Eiben et al., 2010a) in a bio-inspired manner motivated by the vision of self-adaptive, reliable, self-organising

and self-developing swarms of robots and artificial multi-robot organisms (Kernbach et al., 2010). Other work on on-line evolution of robot colonies is presented in (Elfwing et al., 2005) that describes the evolution of controllers for activating hard-coded behaviours for feeding and mating. In (Bianco and Nolfi, 2004), Bianco and Nolfi experiment with open-ended evolution for robot swarms with self-assembling capabilities and report results indicating successful evolution of survival methods and the emergence multi-robot individuals with co-ordinated movement and co-adapted body shapes.

Watson et al. describe the notion of embodied evolution as the evolution taking place within a population of real robots in a distributed and asynchronous manner and report results on a resource gathering experiment (Watson et al., 2002). Although their definition of embodied evolution is similar to our concept of on-line distributed evolution, EDEA's implementation is quite different from their probabilistic gene transfer algorithm, in which single genes are broadcast by every robot at a rate proportionate to its fitness. Experiments with on-line distributed evolution also appear in (Bianco and Nolfi, 2004). The $(\mu + 1)$ ON-LINE algorithm –our exemplar for the encapsulated approach– was reported on in (Haasdijk et al., 2010) and (Eiben et al., 2010b). Floreano et al use encapsulated evolution in (Floreano et al., 2002) to evolve spiking circuits for a fast forward task. Encapsulated on-line evolution as a means for continuous adaptation by using genetic programming is suggested by Nordin and Banzhaf (1997). An island model evolutionary algorithm is used by Usui and Arita (2003) to evolve a fast forward behaviour. Hybrid approaches are also taken in (Elfwing et al., 2005) (island model) and (Nehmzow, 2002) (hall-of-fame approach) though in both cases evolved controllers merely activate hard-coded behaviours.

The majority of the experimental work in the field of evolutionary robotics has concentrated on the off-line evolution of robot controllers, e.g. (Harvey et al., 1996), (Floreano and Mondada, 1996), (Nolfi, 1997). In many of these cases incremental evolution is used to tackle complicated problems while co-evolution has also been examined as a way to address complex tasks (Floreano et al., 2001). Collective robotics settings have been addressed with off-line evolution as well: an extensive framework is presented in (Martinoli, 1999) while application examples can be found in (Baldassarre et al., 2002), (Potter et al., 2001) and (Marocco and Nolfi, 2006).

A recent extensive review of the literature in the evolutionary robotics field can be found in (Nelson et al., 2009).

7.1.3 On-line, On-board Evolution

Any algorithm that implements on-line, on-board evolution has to take some uncommon considerations into account:

- On-board evolution implies (possibly very) limited processing power and memory, so the evolutionary algorithm must show restraint concerning computations, evaluations and population sizes;
- The best performing individual is not as important as in off-line evolution: because controllers evolve as the robots go about their tasks, if a robot continually evaluates poor controllers, that robot's actual performance will be inadequate, no matter how good the best individuals in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution and display a more or less stable level of performance throughout the continuing search;
- On-line evolution requires that the robots autonomously load and evaluate controllers without human intervention or any other preparation: the evaluation of a controller simply picks up the task where the previous evaluation left off. This introduces significant noise in fitness evaluations because the starting conditions of an evaluation obviously can have great impact on a controller's performance;
- Because the evolutionary algorithm has to be able to contend with unforeseen circumstances, it must either be able to (self-) adapt its parameter values as it operates or its parameters must be set to robust values that produce good performance under various conditions.

Subsection 7.1.3.1 lists design choices specific to distributed evolutionary algorithms for on-line, on-board evolution and introduces the EDEA as a implementation of a distributed evolutionary algorithm that takes all pertinent considerations in its stride. Subsection 7.1.3.3 provides some details on the $(\mu + 1)$ ON-LINE algortihm that was designed to address these considerations with the encapsulated approach.

7.1.3.1 A Distributed Algorithm

In distributed implementations of on-line, on-board controller evolution, each robot contains a single genotype that it decodes into its controller and evaluates during regular operation. The population of the evolutionary process is the aggregate of genotypes held by all the robots together; selection and variation occur through robot interactions. Distributed on-line evolution renders many standard centralised evolutionary algorithm concepts inapplicable, specifically requiring a different approach to selection and reproduction.

A centrally orchestrated algorithm would be possible, but it would limit the robots' autonomy, lead to scalability issues for large populations and introduce a single point of failure into the system. Thus, the crucial distinction of our envisioned distributed evolutionary algorithm is the lack of a central authority to guide selection or recombination and the ability of the robots to decide autonomously with whom and when to exchange genetic material, to generate new individuals from it and to deploy them.

To mate –to exchange genetic controller encodings– autonomously, the robots must identify potential partners, select one (disregarding the possibility of multi-parent recombination) and, once offspring genetic material has been constructed, embody it: actually run/evaluate the resulting controller on a robot, replacing that robot's current controller.

Partner identification For small populations –as in the experiments described below– where all robots are constantly in communication range, the robots can have a global view of all the group and contact random robots when interested in mating. This does, however, not scale to large numbers of robots or to environments where only some of the other robots are within communication range. Alternatively, the robots could engage in some hard-coded mate locating behaviour every so often as presented in (Elfwing et al., 2005). Obviously, this detracts from the time robots actually spend tackling their proper tasks. Another approach relies on incidental physical colocation with information being transmitted through some communication channel with limited range as in the Probabilistic Gene Transfer Algorithm (PGTA) (Watson et al., 2002). While an elegant and scalable approach, it does assume a group of robots that is densely deployed in space.

Although not used in the experiments here, EDEA can maintain a peer-to-peer network (using wireless communication) where each individual has a small number of contacts and this overlay network is preserved through gossiping protocols

that maintain connectedness even in the face of massive node failures (Jelasity and van Steen, 2002). Similar networks for structuring the population of an evolutionary algorithm have been successfully employed in experiments presented by Laredo et al. (2010). Note the similarity of this set-up to cellular evolutionary algorithms (Tomassini, 2005).

Partner selection Once potential partners have been contacted, the robot has to select one to mate with. Selection strategies need not be uniform: one role may display eager behaviour (willing to mate with anyone) while another may hold a 'picky' stance (subjecting mating candidates to stricter criteria) – not unlike male and female behaviour in nature. It has been speculated that such a split between male and female behaviour is beneficial because males are forced to explore the genotype space as a result of the females' selectiveness (Darwin, 1871; Miller and Todd, 1995). In a distributed evolutionary algorithm, there is no need to fix an individual's role one way or another: in PGTA, for instance, every robot plays both roles by constantly broadcasting its genes and at the same time evaluating received genes before incorporating them into its own genome (Watson et al., 2002).

In EDEA, robots play both an eager and a selective role, on the one hand selectively initiating the mating process using binary tournament while on the other hand eagerly responding to any mate proposal. Once a candidate has been selected, the initiating robot (which plays the selective role) compares its own fitness with that of the prospective partner to weight a probabilistic decision whether or not to press on with mating.

Embodiment Once a partner has been selected and a new genome created using standard recombination and mutation operators, the resulting genome must be deployed in a robot to be evaluated, replacing the current controller. Since there is no global view of the population in EDEA, the new controller must replace one in the direct neighbourhood (in terms of the overlay network) if it is to be deployed immediately. In fact, EDEA replaces the genome (only one offspring is created per mating interaction) of the initiating robot, justifying its fastidiousness during mate selection.

7.1.3.2 The Embodied Distributed Evolutionary Algorithm

We introduce EDEA as an implementation of the distributed approach that follows from these considerations.

```
genome ← CreateRandomGenome;                                  // Initialisation
initiating ← false;
myFitness ← 0;
for ever do                                                   // continuous adaptation
    act();
    age++;
    fitness ← updateFitness();
    if age > α then
        offers ← P2PGetOffers();                              // eagerly accept
        for o ∈ offers do
            P2PSend(o.sender, genome, myFitness);
        if initiating then                                    // selectively initiate
            candidates ← P2PGetCandidates();
            partner ← BinaryTournament(candidates);
            if random() < candidate.Fitness / (s_c · myFitness) then
                genome ← Crossover(candidate, genome);
                Mutate(genome);                               // Gaussian N(0, σ)
                age ← 0;
            initiating ← false;
        else
            initiating ← (random() < p_c);
```

Algorithm 6: The EDEA evolutionary algorithm.

Algorithm 6 provides pseudo-code for EDEA, which has the following parameters:

Maturation age α Before a genome can be considered in the mating process, it must have been evaluated for at least some time α to make its fitness measure reliable. The maturation age does not define a standard duration of evaluation but rather a lower bound, as a controller may continue to be active after it reaches age α. Adjusting the value of α affects speed of convergence because α implements a trade-off between the reliability of fitness evaluations (long evaluation times increase reliability) and the number of generations that can be achieved in a fixed amount of time (short evaluation time increase the number of evaluations).

Selection coefficient s_c Once a potential partner has emerged as the winner of a binary tournament from the neighbours of the initiator, it is selected based on its fitness in comparison to the fitness of the initiator. This confirmation is probabilistic and the selection coefficient s_c defines how fastidious the re-

ceiver is: the probability of mating is calculated as:

$$\frac{fitness_{candidate}}{fitness_{receiver} \cdot s_c}$$

Thus, larger values for s_c increase the selective pressure.

Preliminary experiments showed that the probability of initiating mating does not have any appreciable impact on the evolutionary process, so it has been set to a fixed value of 0.2. In large groups, however, it may perhaps be used to regulate network load.

From (Haasdijk et al., 2010), we expect the mutation step size σ to have considerable impact on the algorithm's performance. We employ the derandomised self-adaptive strategy (Ostermeier et al., 1994) to control this parameter during EDEA runs: this has been shown to work well in similar settings with $(\mu + 1)$ ON-LINE (Eiben et al., 2010b).

7.1.3.3 An Encapsulated Algorithm

As benchmark, we consider an example of the encapsulated case: the $(\mu + 1)$ ON-LINE algorithm as studied in (Eiben et al., 2010a,b; Haasdijk et al., 2010). In an encapsulated scheme, each robot contains an evolutionary algorithm to adapt its controller without reference to other robots or any central authority, therefore, it is not limited to situations involving groups of robots: it can be applied to a single robot as well. Such an application of encapsulated on-line evolution to a single robot realises the basic notion of on-line on-board evolution: an evolutionary process that continuously runs during deployment and task execution in order to provide constant adaptation in a changing and unpredictable environment. The $(\mu + 1)$ ON-LINE algorithm is an adaptation of the classic evolution strategy (Schwefel, 1995) with a fairly small population generating only $\lambda = 1$ child per cycle[i] to save on fitness evaluations. The $(\mu + 1)$ ON-LINE algorithm employs standard evolutionary algorithm operators (selection, variation and recombination) on a population of size μ to develop a new individual. That new individual –the challenger– is then evaluated by letting it take control of the robot for τ time steps and measuring the robot's task performance over that period. If the challenger's performance proves better than that of the worst in the population, the challenger replaces the current

[i] A value that would be considered extremely small by evolution strategy standards

worst and the next iteration commences. To cope with noisy fitness evaluations, $(\mu + 1)$ ON-LINE re-evaluates genomes in the population with a given probability. This means that at every evolutionary cycle, either a new individual is generated and evaluated (with probability $1 - \rho$), or an existing individual is re-evaluated (with probability ρ). The fitness values from subsequent (re-)evaluations of any given individual are combined using an exponential moving average; this emphasises newer performance measurements and so is expected to promote adaptivity in changing environments;

For a detailed description of and discussion on $(\mu + 1)$ ON-LINE, refer to (Eiben et al., 2010a,b; Haasdijk et al., 2010).

7.1.4 Experimental Assessment

To assess EDEA, we compare it to $(\mu + 1)$ ON-LINE in a number of well-established settings as described below. While having multiple robots around is not a requirement for the encapsulated algorithm, it is obviously essential in the distributed case and to ensure equal circumstances, we have a group of 10 robots simultaneously tackling the problem in each instance. For the distributed approach, this means that there is a single evolutionary process using 10 robots with a population of 10 (one genotype per robot). In the encapsulated runs, there are 10 evolutionary processes (one in every robot), each with a separate population of μ individuals.

To ensure a fair comparison, we perform a modest parameter sweep for each algorithm in each task: multiple values are chosen for the most influential parameters and the algorithm is run several times for all parameter value vectors per task. We perform 20 repeats with different random seeds for each combination of task, algorithm and parameter vector. With 10 robots in each run, this yields 200 observations of robot performance for each combination. The values in the parameter sweep are chosen based on previous experience with the $(\mu + 1)$ ON-LINE (Haasdijk et al., 2010) (Eiben et al., 2010b) and preliminary experiments with EDEA.

For each task, only the best parameter vector for each algorithm was used in the final comparison of the performance of encapsulated and distributed evolution. The variety of performance across parameter vectors can be seen as an indication of each algorithm's tuneability –the sensitivity to the parameter settings– the lower the tuneability, the less effort one needs to spend to get the parameter values just right.

In all experiments, the robots –simulated e-pucks in the Webots simulator[ii]– are controlled by simple perceptron neural networks and the evolutionary algorithms determine the weights of the connections between the neurons. The fitness functions are obviously task dependent and are described with each task, below. The perceptrons use a *tanh* activation function and receive inputs from light, distance, pheromone and food sensors and camera (depending on the task) and have two or three (depending on the task) output neurons that drive the wheels and LEDs. All inputs are normalised in the $[0, 1]$ interval before fed to the neurons. Equally, the outputs are normalised to $[0, 1]$ and interpreted as fraction of the full speed for the motors and as an on/off value for the LEDs (values less than 0.5 turn the LEDs off while larger values turn them on).

The experiments run for 10,000 seconds of simulated time; we use time rather than number of evaluations or generations because we are primarily interested in the performance of the robots in real time, regardless of how that is achieved by the evolutionary algorithm.

As stated in Section 7.1.3, we are primarily interested in the actual performance of robots, not in the performance of the best individual in the population at any given time. Actual performance is measured as the average performance during a time-span, irrespective of how many controllers may have been activate during that time.

Because of the limited group size of the experiments, we do not use the gossiping maintenance scheme described in Section 7.1.3.2, but randomly select potential partners from the whole population.

The settings for the experiments are summarised in Table 7.1.[iii]

7.1.4.1 Phototaxis

Phototaxis –seeking out or tracking a light source– is a very straightforward task that has been addressed by many ER researchers. The task is frequently combined with other tasks such as goal homing (Tuci et al., 2002) and flocking (Baldassarre et al., 2002). In our comparison, we use the simplest version of phototaxis: robots only have to move towards a stationary light source and then remain as close to it as possible. In the phototaxis task, the robots use eight light sensors to detect light

[ii] `http://www.cyberbotics.com/`

[iii] Source code for the algorithm as well as the experiments described here is available at `http://www.few.vu.nl/~ehaasdi/papers/GECCO-EncapsvsDistr`

Experiment details	
Task	phototaxis, fast forward, collective patrolling
Robot group size	10
Simulation length	10,000 seconds (simulation time)
Number of repeats	20
Controller details	
NN type	Perceptron
Input nodes	*Phototaxis:* 8 light sensors + bias; *Fast forward:* 8 distance sensors + bias; *Collective patrolling:* 8 distance sensors + 4 pheromone sensors + bias;
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length	*Phototaxis, Fast forward:* 18; *Collective patrolling:* 26
Fitness	See task descriptions
Mutation	Gaussian $N(0, \sigma)$
Mutation step-size	Derandomised self-adaptive
Crossover	averaging
edea settings	
Maturation time α	300, 600, 1200, 1800 time steps
Selection coefficient s_c	0.5, 0.75, 1.0
Partner location	peer-to-peer network
Partner selection	binary tournament and fitness-based probabilistic
Embodiment	replace initiating parent
$(\mu + 1)$ on-line settings	
Evaluation time τ	300, 600, 1200 time steps
Re-evaluation rate ρ	0.2, 0.4, 0.6
Re-evaluation strategy	exponential moving average
Population size μ	6, 10, 14
Parent selection	binary tournament
Crossover rate	1.0
Survivor selection	replace worst in population if challenger is better

Table 7.1 – Experimental set-up

intensity and base their behaviour on that. The fitness function simply rewards intensity of received light:

$$f = \sum_{t=0}^{\tau} \max_{i=1}^{8}(lightSensor_i) \tag{7.1}$$

where $lightSensor_i$ is the normalised input from a light sensor between 0 (no light) and 1 (brightest light).

The arena is an empty (apart from the ten robots) square with a light source in the middle: we ignore collisions between robots in these experiments, so we can do without the robot's distance sensors. For this simple experiment, we also compare the performance of both algorithms against a Braitenberg (Braitenberg, 1984) controller as a baseline.

7.1.4.2 Fast Forward

Fast forward –moving in as straight line as possible as fast as possible while avoiding obstacles– is maybe the most common task in evolutionary robotics research. In a confined environment with obstacles this task implies a trade-off between avoiding obstacles and maintaining speed and forward movement. The fitness function we use has been adapted from (Nolfi and Floreano, 2000); it favours robots that are fast and go straight ahead. Equation 7.2 describes the fitness calculation:

$$f = \sum_{t=0}^{\tau}(v_{trans} \cdot (1 - v_{rot}) \cdot (1 - d)) \tag{7.2}$$

where v_{trans} and v_{rot} are the translational and the rotational speed, respectively. v_{trans} is normalised between -1 (full speed reverse) and 1 (full speed forward), v_{rot} between 0 (movement in a straight line) and 1 (maximum rotation); d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle). As for phototaxis, we also compare the performance against a Braitenberg controller as a baseline.

Although fast forward is considered a trivial task, here some extra difficulty is added by using a complicated maze-like arena (Figure 7.1(a)) with tight corners and narrow corridors that fit only a single robot and sometimes lead to dead ends. This arena structure, compounded by the fact that multiple robots will be simultaneously deployed, makes the task considerably harder than commonly seen

instances. This additional complexity is confirmed by the results of the baseline trials: the Braitenberg controllers invariably get stuck after a while (Section 7.1.5).

7.1.4.3 Collective Patrolling

An obvious real-world task for a group of autonomous mobile robots is that of a distributed sensory network, where the robots have to patrol an area as a group and detect events that occur periodically. It differs from the previous tasks since it requires some level of co-ordination: the success of the group depends not only on the efficient movement of the individual robots but also on the spread of the group across the arena to maximise the probability of detecting events. Somehow, robots need to liaise so as not to patrol the same areas. To this end, they are equipped with a pheromone system: robots continuously drop pheromones (this is a fixed behaviour and not controlled by the evolved controller) while sensors detect the local pheromone levels. The collective patrolling task is described in (Martinoli, 1999) where controllers evolve off-line, although in that work the approach to events is more complicated and the robots use other sensory inputs.

The experiments for this task take place in the arena shown in Figure 7.1(b).

Every $T_e = 50ms$ with probability $p_e = 0.0005$, an event occurs at a random location with a duration of $d_e = 500 + \mathcal{N}(0, 2)$ seconds. Thus, in one run (10,000 seconds) approximately 100 events occur and that at any time around 5 events are active in the whole arena.

A robot detects an event whenever it comes within $0.3m$ of the event, so a robot's sensory coverage is $0.283m^2$. Since the arena is $25m^2$, a group of 10 robots can at any moment cover at most 11% of the whole arena; conversely, a group of stationary robots should detect around 11% of the events.

Pheromones are simulated as follows: the $5m \times 5m$ arena is divided into 500×500 cells, each with a pheromone level between $[0, 2]$. Every second, each robots drops 1 unit of pheromones at the cell the robot is currently in, and a linearly decreasing amount in nearby cells up to a range of $R_p = 0.07m$. Pheromone levels decay over time at a rate of $R_c = -0.024/s$ in each cell.

As sensory input to the controller, 4 pheromone sensors are placed at the periphery of the circular body at $\frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$. Each sensor detects the accumulated pheromone levels of all cells in a range of $0.05m$ (with detected levels decreasing linearly with distance from the sensor). For fitness calculation only, a similar sensor is positioned on the centre of the robot.

The fitness function penalises pheromones presence (detected by the central pheromone sensor) and proximity to obstacles:

$$f = \sum_{t=0}^{\tau}((1-p)\cdot(1-d)) \tag{7.3}$$

where p indicates pheromone presence between 0 (no pheromones) and 1 (strongest pheromone level) at the current location and d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

This fitness function rewards covering behaviour and does not include the number of events detected, but we report the percentage of detected events as performance (e.g., in Fig. 7.4) rather than on the fitness itself.

Although movement is not included explicitly, it should emerge due to the continous dropping of pheromones and the deleterious effect of staying in a place where the robot just dropped them.

(a) Fast forward arena (b) Patrolling arena

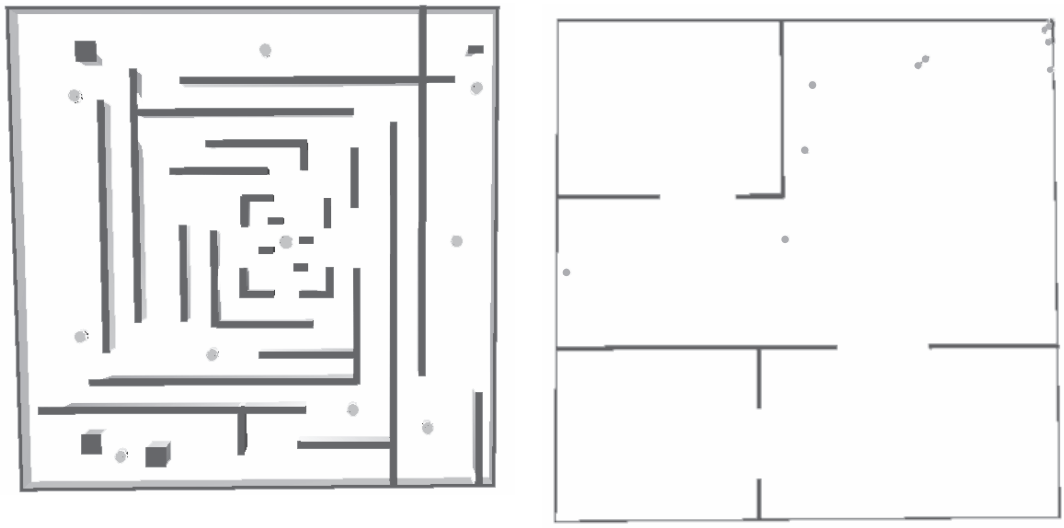


Figure 7.1 – Arenas for two tasks; the circles represent the robots to scale.

7.1.5 Results

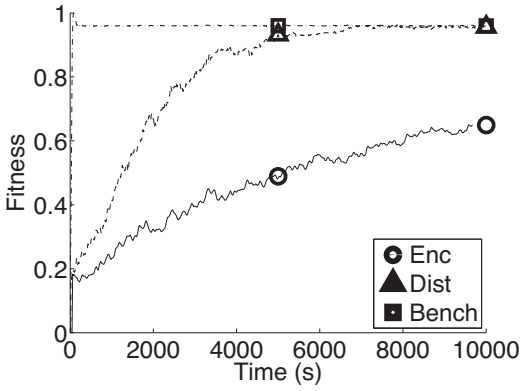
Figures 7.2 to 7.4 compare the performance of the encapsulated $(\mu+1)$ ON-LINE and distributed EDEA algorithms on the various tasks, using the best performing parameter settings for that task. The graphs on the left present performance –

averaged over all repeats and robots– versus time. In these plots, the performance is scaled so that the theoretical optimum is 1. For the phototaxis and fast forward tasks, we additionally plot performance for the Braitenberg vehicles as a baseline.

To assess the volatility of the robot's actual performance over the course of the experiments, we calculate the differential entropy of actual performance over the last 20% of each run: lower entropy indicates a lower level of volatility. The right-hand plots show average entropy (grey bars) with standard deviation (black whisker lines) for the 20 runs with the best performing parameter vector over all robots for both algorithms.

Far all comparisons, we test the significance of difference with t-tests at 95% confidence.

(a) Performance vs. time



(b) Entropy (lower means more stable performance)

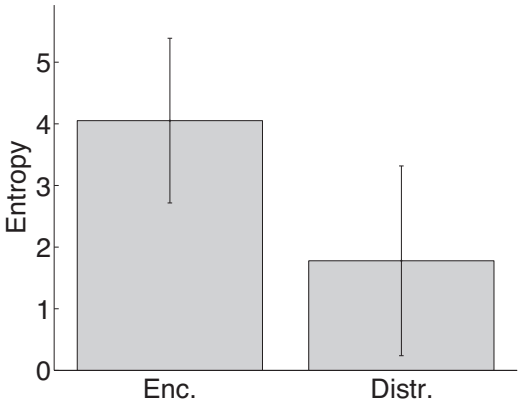
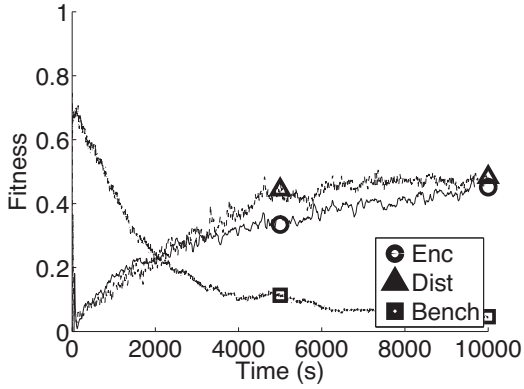


Figure 7.2 – Phototaxis results; EDEA is significantly better in terms of both performance and stability.

Summarising these comparisons, EDEA performs significantly better in the phototaxis and fast forward tasks and is more stable in the phototaxis task (while there is no significant difference in stability for the fast forward task). However, $(\mu + 1)$ ON-LINE has significantly better performance in the patrolling task, although EDEA is more stable there, as well.

For the phototaxis and fast forward tasks, the evolved controllers compare very well to the Braitenberg baseline. Evolved phototaxis controllers match the behaviour of the benchmark while the evolved fast forward controllers outperform the benchmark controller –which gets stuck in cul-de-sacs where left and right sensory input are equal– by a considerable amount.

(a) Performance vs. time



(b) Entropy (lower means more stable performance)

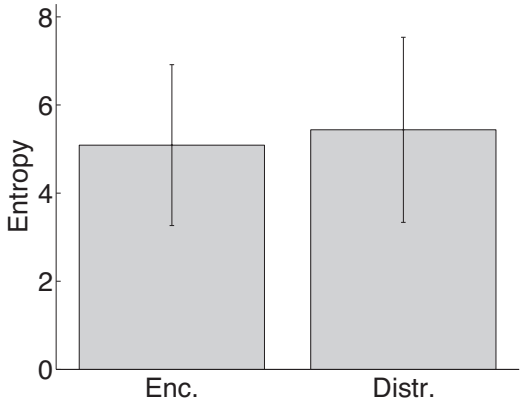
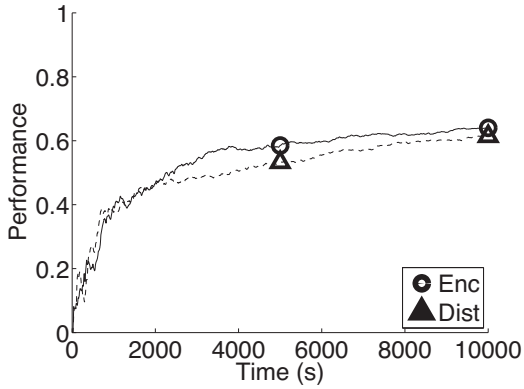


Figure 7.3 – Fast forward results; EDEA performs better at this task, the difference in stability cannot be shown to be significant.

(a) Performance vs. time



(b) Entropy (lower means more stable performance)

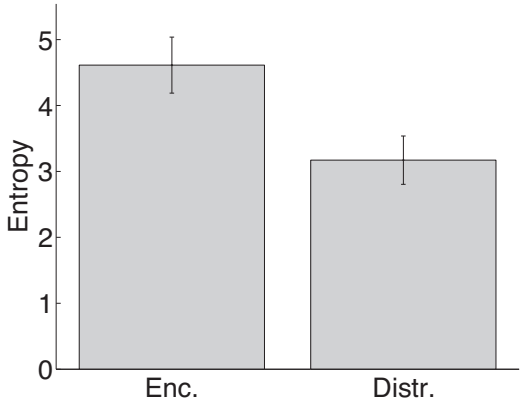


Figure 7.4 – Collective patrolling results; $(\mu + 1)$ ON-LINE performs better, EDEA is significantly more stable.

Table 7.2 shows the parameter vectors that lead to optimal or near optimal (not statistically different from optimal according to 95% t-test) performance. Interestingly, for all tasks there is at least one (near) optimal vector with every value of s_c tested. This seems to indicate that s_c may be kept constant as there will always be a value for α that will work well with it.

Table 7.3 compares the tuneability –the dependence on parameter settings to achieve good performance– of the two algorithms by comparing the ratio of the parameter vectors that result in near optimal performance out of all parameter vectors that were considered. This shows that EDEA is more resilient when it comes

to parameter tuning: a substantially higher ratio of vectors leads to near optimal performance for two tasks, while for phototaxis the same ratio of vectors is near optimal.

7.1.6 Discussion and Conclusion

On the whole, on-line, on-board evolution of robot controllers is a success: with only very small populations and within short times, both EDEA and $(\mu + 1)$ ON-LINE are able to evolve good controllers for all three tasks, matching or even outperforming benchmark performance. Although the tasks used are somewhat

	EDEA		$(\mu + 1)$ ON-LINE		
	α	s_c	μ	ρ	τ
Phototaxis	***1800***	***0.5***	**10**	**0.4**	**300**
	1800	1	6	0.2	600
	1800	0.75	6	0.4	300
			6	0.6	1200
			10	*0.6*	*300*
			10	0.4	300
			10	0.4	600
Fast forward	**1200**	**0.75**	**6**	**0.6**	**300**
	1800	*0.5*	*10*	*0.6*	*300*
	600	1	6	0.4	300
	600	0.75			
Patrolling	**1200**	**0.5**	**10**	**0.6**	**300**
	1800	0.75	14	0.6	300
	1800	1	14	0.4	300
	1200	1	*10*	*0.6*	*300*
	1800	*0.5*	10	0.4	300
	1200	0.75	6	0.6	600

Table 7.2 – The best performing vectors are listed in bold; the remainder are vectors whose performance is not significantly worse (according to 95% t-tests) than the best. Settings in italics denote a parameter vector that is common for all tasks.

	EDEA	$(\mu + 1)$ ON-LINE
Phototaxis	25% (3/12)	25% (7/27)
Fast forward	33% (4/12)	11% (3/27)
Patrolling	50% (6/12)	22% (6/27)

Table 7.3 – Tuneability comparison of EDEA and $(\mu + 1)$ ON-LINE. Shows, for each task, the percentage of parameter vectors that yield (near) optimal performance.

straightforward, these same tasks have been used for off-line evolutionary robotics experiments that had superior resources available in terms of population size, generations and time. In that light, the success of on-line evolution as shown here is noteworthy and we can conclude that EDEA does provide a successful implementation of the general idea. It is a matter of further investigation whether the on-line algorithms tested here will scale up to more complex tasks and environments.

In terms of performance, the comparison of these implementations of the encapsulated and distributed approaches shows no clear winner: EDEA outperforms $(\mu + 1)$ ON-LINE for the (simpler) phototaxis and fast forward tasks, but $(\mu + 1)$ ON-LINE is better when it comes to patrolling. Of course, this comparison relies on the parameter values of the evolutionary algorithms, which are determined by our tuning process. Forced by computational limitations we only tested a small number of different parameter values. Nevertheless, it seems safe to say that the relative performance of the algorithms depends on the nature of the task, and the results simply beg for a combined island model algorithm that may have the best of both worlds.

In terms of sensitivity to parameter settings, EDEA does seem to be the clear winner: even in the patrolling task, where $(\mu + 1)$ ON-LINE performs better, EDEA has the advantage that half of the tested parameter vectors were optimal while the performance is not much (although significantly) worse than that of $(\mu + 1)$ ON-LINE. Moreover, substantially more effort was put into tuning $(\mu + 1)$ ON-LINE than EDEA (2 parameters and 12 vectors for EDEA versus 3 parameters and 27 vectors for $(\mu + 1)$ ON-LINE). Table 7.2 indicates that EDEA's two parameters may be reduced to one: for all tasks there is at least one (near) optimal vector with every value of s_c tested, so it seems that it is merely a matter of finding the appropriate setting for α for some constant s_c (possibly through on-line parameter control (e.g. based on racing), making EDEA completely parameter-free).

One reason for EDEA's success may lie in the fact that a distributed evolutionary algorithm exploits the presence of multiple robots by effectively implementing concurrent evaluation of the population, allowing evolution to progress rapidly in real time. Meanwhile, the encapsulated approach can only evaluate its populations sequentially, falling rapidly behind in terms of evolutionary steps taken.

On the other hand, the multiple instance nature of $(\mu + 1)$ ON-LINE's encapsulated approach can offer an advantage when dealing with tasks or environments

that involve competition or require various skills: here, the separate evolutionary algorithms of robots can promote co-evolution, speciation and/or specialisation.

The advantage of the concurrent evaluation can be easily understood for the tasks where EDEA shines, but the connection between the patrolling task, where the encapsulated $(\mu + 1)$ ON-LINE algorithm performs better, and the advantage of co-evolution and/or speciation is not so straightforward. Here, performance is based on the presence of pheromones –that all robots emit continuously– and a robot must learn to move around efficiently while avoiding its own as well as others' pheromones to claim fresh locations. Robots that have the same strategy for moving around are hamstrung in such a scenario: they always trip over each other's pheromones, exactly because they follow similar paths. Obviously, in the distributed case, robots are more likely to have similar controllers (as they are from the same population) than in the encapsulated case (where each controller stems from a different, independently evolving population), and so are more likely to follow a trodden path. In fact, the set-up here introduces a subtle form of competitive co-evolution: although the task is a collective one, the robots actually compete for sites in the arena to claim.

An obvious next step from this research is to try and have the best of both worlds by merging the encapsulated and distributed approach into an island-like model of autonomously evolving populations in each robot with individuals migrating from one to another. Research in this direction is described in the next sections.

Our conclusions need to be confirmed with more extensive experiments of increased complexity –both in terms of task and controller structure. Work to this end is underway.

7.2 A Peer-to-Peer Distributed Algorithm

Imagine autonomous, self-sufficient robot collectives that can adapt their controllers autonomously and self-sufficiently to learn to cope with situations unforeseen by their designers. As one step towards the realisation of this vision, we investigate on-board evolutionary algorithms that allow robot controllers to adapt without any outside supervision and while the robots perform their proper tasks. We propose an EvAg-based on-board evolutionary algorithm, where controllers are exchanged among robots that evolve simultaneously. We compare it with the $(\mu + 1)$ on-line algorithm, which implements evolutionary adaptation inside a single robot. We perform simulation experiments to investigate algorithm performance and use parameter tuning to evaluate the algorithms at their *best possible* parameter settings. We find that distributed on-line on-board evolutionary algorithms that share genomes among robots such as our EvAg implementation effectively harness the pooled learning capabilities, with an increasing benefit over encapsulated approaches as the number of participating robots grows.

7.2.1 Introduction

The work presented in this section is inspired by a vision of autonomous, self-sufficient robots and robot collectives that can cope with situations unforeseen by their designers. An essential capability of such robots is the ability to adapt – evolve, in our case– their controllers in the face of challenges they encounter in a hands-free manner, "the ability to learn control without human supervision," as Nelson et al. (2009) put it . In a scenario where the designers cannot predict the operational circumstances of the robots (e.g, an unknown environment or one with complex dynamics), the robots need to be deployed with roughly optimised controllers and the ability to evolve their controllers autonomously, on-line and on-board.

Section 7.2 was published as:

Robert-Jan Huijsman, Evert Haasdijk and A.E. Eiben (2011). An On-line On-board Distributed Algorithm for Evolutionary Robotics. In Jin-Kao Hao, et al., *Artificial Evolution, 10th International Conference, Evolution Artificielle, EA, 2011, Angers, France, October 24-26, 2011*, Pages 119–131, Springer-Verlag, Berlin / Heidelberg.

This contrasts with the majority of evolutionary robotics research, which focusses on *off-line* evolution, where robot controllers are developed -evolved- in a separate training stage before they are deployed to tackle their tasks in earnest.

When dealing with multiple autonomous robots, one can distinguish three options to implement on-line evolution (Haasdijk et al., 2010):

Encapsulated Each robot carries an isolated and self-sufficient evolutionary algorithm, maintaining a population of genotypes inside itself;

Distributed Each robot carries a single genome and the evolutionary process takes place by exchanging genetic material between robots;

Hybrid Which combines the above two approaches: each robot carries multiple genomes and shares these with its peers.

We compare instances of each of these three schemes: the encapsulated $(\mu + 1)$ ON-LINE algorithm, the distributed Evolutionary Agents algorithm (EVAG) Laredo et al. (2010) and a hybrid extension of EVAG.

One of EVAG's distinguishing features is that it employs the newscast algorithm (Jelasity and van Steen, 2002) to exchange genomes between peers (in our case: robots) and to maintain an overlay network for peer-to-peer (robot-to-robot) communication. Newscast-based EVAG has proved very effective for networks of hundreds or even thousands of peers, but in the case of swarm robotics, network sizes are likely to be much smaller. Therefore, it makes sense to compare the efficacy of newscast-based EVAG with a panmictic variant where the overlay network is fully connected.

It is well known that the performance of evolutionary algorithms to a large extent depends on their parameter values (Nannen et al., 2008). To evaluate the algorithms at their *best possible* parameter setting, we tune the algorithm parameters with REVAC, an evolutionary tuning algorithm specifically designed for use with evolutionary algorithms, introduced in Nannen and Eiben (2007).

Summarising, the main question we address in this section is: how do the three algorithms compare in terms of performance and can we identify circumstances in which to prefer one of the three schemes over the others? Secondly, we investigate how EVAG's newscast population structure influences its performance compared to a panmictic population structure. Thirdly, we briefly consider the sensitivity of the algorithms to parameter settings.

7.2.2 Related work

The concept of on-board, on-line algorithms for evolutionary robotics was discussed as early as 1995 in Nordin and Banzhaf (1995), with later research focussed on the 'life-long learning' properties of such a system inWalker et al. (2006).

A distributed approach to on-line, on-board algorithms was first investigated as 'embodied evolution' in Watson et al. (2002), where robots exchange single genes at a rate proportional to their fitness with other robots that evolve in parallel. Other work on on-line evolution of robot controllers is presented in Elfwing et al. (2005) that describes the evolution of controllers for activating hard-coded behaviours for feeding and mating. In Bianco and Nolfi (2004), Bianco and Nolfi experiment with open-ended evolution for robot swarms with self-assembling capabilities and report results indicating successful evolution of survival methods and the emergence of multi-robot individuals with co-ordinated movement and co-adapted body shapes.

The $(\mu + 1)$ ON-LINE algorithm – our exemplar for the encapsulated approach – has been extensively described in Haasdijk et al. (2010) and Eiben et al. (2010b), where it was shown to be capable of evolving controllers for a number of tasks such as obstacle avoidance, phototaxis and patrolling. Floreano et al. (2002) uses encapsulated evolution to evolve spiking circuits for a fast forward task. Encapsulated on-line evolution as a means for continuous adaptation by using genetic programming is suggested in Nordin and Banzhaf (1997).

The distributed approach to on-line evolutionary robotics has a clear analogy with the field of parallel evolutionary algorithms, in particular to the fine-grained approach, where each individual in the population has a processor of its own. The primary distinguishing factor among fine-grained parallel algorithms is their population structure, with small-world graphs proving competitive with panmictic layouts (Giacobini et al., 2006).

The hybrid scheme can be implemented as what in parallel evolutionary algorithms is known as the island model: the population is split into several separately evolving sub-populations (the islands), that occasionally exchange genomes. This approach is used in Usui and Arita (2003) and Elfwing et al. (2005). A variant where the robots share a common hall of fame is implemented in Nehmzow (2002). As will become apparent, we take a slightly different approach where genome exchange between sub-populations is the norm.

7.2.3 Algorithms

Autonomous on-line adaptation poses a number of requirements that regular evolutionary algorithms don't necessarily have to contend with. We take a closer look at two especially relevant considerations.

To begin with, fitness must be evaluated *in vivo*, i.e., the quality of any given controller must be determined by actually using that controller in a robot as it goes about its tasks. Such real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances; for instance, the nth controller will start at the final location of the $(n-1)$th one. This leads to very dissimilar evaluation conditions and ultimately to very noisy fitness evaluations. To address this issue, the algorithms we investigate here implement re-evaluation: whenever a new evaluation period commences, the robot can choose (with a probability ρ) not to generate a new individual but instead re-evaluate an existing individual to refine the fitness assessment and so combat noise.

The second issue specific to on-line evolution is that, in contrast to typical applications of evolutionary algorithms, the best performing individual is not the most important factor when applying on-line adaptation. Remember that controllers evolve as the robots go about their tasks; if a robot continually evaluates poor controllers, that robot's *actual* performance will be inadequate, no matter how good the best known individuals as archived in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution (even if it is not the best) and search prudently: it must display a more or less stable but improving level of performance throughout the continuing search.

7.2.4 $(\mu + 1)$ on-line

The $(\mu + 1)$ ON-LINE algorithm is based on the classical $(\mu + 1)$ evolutionary strategy (Schwefel, 1981) with modifications to handle noisy fitness evaluations and promote rapid convergence. It maintains a population of μ individuals within each robot and these are evaluated in a time-sharing scheme, using an individual's phenotype as the robot's controller for a specified number of time units. A much more detailed description of $(\mu + 1)$ ON-LINE is given in Haasdijk et al. (2010).

7.2.5 EvAg

EvAg was originally presented in Laredo et al. (2010) as a peer-to-peer evolutionary algorithm for parallel tackling of computationally expensive problems, with the ability to harness a large number of processors effectively. The analogies between parallel evolutionary algorithms and a swarm of robots adapting to their environment and tasks in parallel make EvAg a suitable candidate for an on-board, on-line distributed evolutionary algorithm for evolutionary robotics.

The basic structure of EvAg is straightforward and similar to a $1 + 1$ evolution strategy: each peer (robot) maintains a record of the best solution evaluated by that peer up until that point – the champion. For every new evaluation a new candidate is generated, using crossover and mutation; if the candidate outperforms the current champion it replaces the champion.

The basic definition of EvAg leaves many decisions open to the implementer, such as the choice of recombination and mutation operators and the details of parent selection (other than that it should select from peers' champions). Because we are interested in the effects of the distributed nature of the algorithm rather than those due to, say, different recombination schemes, we have chosen our evolutionary operators to match the $(\mu + 1)$ on-line algorithm. As a result, the only difference between EvAg and $(\mu + 1)$ on-line (with $\mu = 1$) lies in the exchange of genomes between robots and using a cache of received genomes rather than only a locally maintained population when selecting parents.

In this light, the extension of regular EvAg to a hybrid form is a straightforward one: rather than maintaining only a single champion on-board, the robots now maintain a population of μ individuals locally. With $\mu = 1$, this implements the distributed scheme, with $\mu > 1$, it becomes a hybrid implementation. With the cache of received genomes disabled, it boils down to $(\mu + 1)$ on-line, our encapsulated algorithm. The pseudo code in algorithm 7 illustrates the overlap between these three implementations.

EvAg normally uses newscast Jelasity and van Steen (2002) to exchange solutions efficiently while maintaining a low number of links between peers: each robot locally maintains a cache of recently received genomes. Periodically, each robot randomly takes one of the genomes in its cache and contacts the robot at which that genome originated. These two robots then exchange the contents of their cache. When needed, parents are selected from the union of this cache and the local champion using binary tournament selection. Because Jelasity and van

```
for i ← 1 to μ do                                                    // Initialisation
    population[i] ← CreateRandomGenome();
    population[i].σ ← σ_initial;   // Mutation step size, updated cf. Eiben
    et al. (2010b)
    population[i].Fitness ← RunAndEvaluate(population[i]);
for ever do                                                  // Continuous adaptation
    if random() < ρ then          // Don't create offspring, but re-evaluate
    selected individual
        Evaluatee ← BinaryTournament(population);
        Evaluatee.Fitness ← (Evaluatee.Fitness +
        RunAndEvaluate(Evaluatee)) / 2;  // Combine re-evaluation results
        through exponential moving average
    else                 // Create offspring and evaluate that as challenger
        ParentA ← BinaryTournament(pool of possible parents);
        ParentB ← BinaryTournament(pool of possible parents - parentA);
        if random() < crossoverRate then
            Challenger ← AveragingCrossover(ParentA, ParentB);
        else
            Challenger ← ParentA;
        if random() < mutationRate then
            Mutate(Challenger);              // Gaussian mutation from N(0, σ)
        Challenger.Fitness ← RunAndEvaluate(Challenger);
        if Challenger.Fitness > population[μ].Fitness then    // Replace last (i.e.
        worst) individual in population w.  elitism
            population[μ] ← Challenger;
            population[μ].Fitness ← Challenger.Fitness;
    Sort(population);
```

Algorithm 7: The on-line evolutionary algorithm. For $(\mu + 1)$ ON-LINE, the pool of possible parents is the on-board population of size μ; for both regular and hybrid EVAG, it is the union of individuals received from the robot's peers through newscast and the on-board population (with $\mu = 1$ for standard EVAG).

Steen (2002) showed that with this update scheme, picking a genome randomly from the cache of received genomes is all but equivalent to picking one randomly from the entire population, this assures that the binary tournament takes place as if the contestants were randomly drawn from the combined population across all robots.

Earlier research showed very promising results for EVAG Laredo et al. (2010), but these results were obtained using thousands of nodes, while evolutionary robotics generally takes place with group sizes of no more than a few dozen robots.

To investigate if EvAg's newscast overlay network remains efficient in these smaller populations we evaluate not only the standard newscast-based EvAg, but also an EvAg variant that uses a panmictic population structure where a robot's choice of genomes for the binary tournament parent selection is truly uniform random from the entire population across all robots. This variant of EvAg requires full connectivity among peers (robots) and is therefore not suitable for use in truly large-scale applications. However, evaluation of the panmictic structure compared to that of newscast is interesting, since it allows us to determine the performance penalty of using of a peer-to-peer approach.

7.2.6 Experiments

To investigate the performance of the algorithms and their variants we conduct experiments with simulated e-pucks in the roborobo[iv] environment. The experiments have the robots running their own autonomous instance of $(\mu + 1)$ on-line, EvAg or its hybrid extension EvAg, governing the weights of a straightforward perceptron neural net controller with hyperbolic tangent activation function. The neural net has 9 input nodes (8 sensors and a bias), no hidden nodes and 2 output nodes (the left and right motor values for the differential drive), giving a total of 18 weights. To evolve these 18 weights, the evolutionary algorithm uses the obvious representation of real-valued vectors of length 18 for the genomes.

The robots' task is movement with obstacle avoidance: they have to learn to move around in a constrained arena with numerous obstacles as fast as possible while avoiding the obstacles. The robots are positioned in an arena with a small loop and varied non-looping corridors (see Fig. 7.5). The fitness function we use has been adapted from Nolfi and Floreano (2000); it favours robots that are fast and go straight ahead. Fitness is calculated as follows:

$$f = \sum_{t=0}^{\tau} (v_t \cdot (1 - v_r)) \tag{7.4}$$

where v_t and v_r are the translational and the rotational speed, respectively. v_t is normalised between -1 (full speed reverse) and 1 (full speed forward), v_r between 0 (movement in a straight line) and 1 (maximum rotation). In our simulations, whenever a robot touches an obstacle, $v_t = 0$, so the fitness increment for time-

[iv] http://www.lri.fr/~bredeche/roborobo/

steps where the robot is in collision is 0. A good controller will turn only when necessary to avoid collisions and try to find paths that allow it to run in a straight line for as long as possible.

We run our simulations with each robot in an arena of its own so that they can't get in each others' way, although the robots obviously can communicate across arena instances. The reasons for this are twofold: firstly, eliminating physical interaction between the robots ensures that a robot's performance is due to its own actions rather than that of others around it; this allows us a clearer view of the effects of genome exchange. Secondly, it allows us to scale our simulations from a very small number of robots to a very large number of robots while using the exact same arenas; this guarantees that in those cases any change in performance of the robots is due to their increased group size rather than a change in environment.

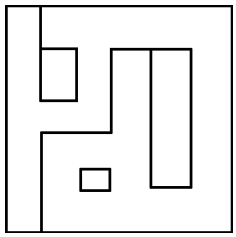


Figure 7.5 – The arena

We evaluate the EvAg variants with group sizes of 4, 16, 36 and 400 robots: we hypothesise that differences in group size influence the performance of distributed and hybrid algorithms due to their facilities for genome exchange. Since a larger group of robots is able to evaluate a larger number of candidate solutions simultaneously, the odds of finding a successful genome are higher. EvAg is intended to distribute these successful genomes across all robots, thus improving the performance of the entire group.

Because the experiments were designed so that genome exchange is the only possible interaction between robots in a group, the robots can have no physical interaction. Without physical interaction and no genome exchange the *average* performance of a group of 4 robots running $(\mu + 1)$ on-line would be identical to that of a group of 400 robots doing the same. Since average performance is our only metric for success and since for $(\mu + 1)$ on-line this metric is not influenced by the number of robots in the experiment, we can perform experiments for $(\mu + 1)$ on-line for a group of 4 robots and use these results as a fair comparison to an experiment with EvAg using 400 robots. We can therefore safely omit the costly $(\mu + 1)$ on-line simulations for the group sizes 16, 36 and 400 as their performance would be the same as that of the group of 4.[v]

[v] Source code and scripts to repeat our experiments can be found at `http://www.few.vu.nl/~ehaasdi/papers/EA-2011-EvAg`.

Rather than comparing the algorithms at identical (so far as possible) parameter settings, we compare the performance at their relative *best possible* parameter settings after tuning as described in Sec 7.2.7. Note that, where applicable, values of μ may vary from one experiment to the next: this does not imply that the number of evaluations is influenced as the robots have a fixed amount of (simulated) wall-clock time in which to learn the task.

7.2.7 Evaluation with parameter tuning

It is a well-known fact that the performance of an evolutionary algorithm is greatly determined by its parameter values. Despite this, many publications in the field of evolutionary computing evaluate their algorithms using fairly arbitrary parameter settings, based on ad hoc choices, conventions, or a limited comparison of different parameter values. This approach can easily lead to misleading comparisons, where method A is tested with very good settings and method B based on poor ones. An recommendable alternative is the use of *automated parameter tuning*, where a tuning algorithm is used to optimize the parameter values and one compares the best variants of the evolutionary algorithms in question Eiben and Smit (2011). This approach helps prevent misleading comparions.

In our experiments with 4, 16 and 36 robots we evaluate the performance of the algorithms by performing parameter tuning for a fixed length of time and comparing the results. We use the MOBAT toolkit[vi] to automate our tuning process. MOBAT is based on REVAC (Nannen and Eiben, 2007), which has been shown to be an efficient algorithm for parameter tuning (Smit and Eiben, 2009). For every combination of robot group size and algorithm MOBAT evaluates 400 parameter settings; each parameter setting is tested 25 times to allow statistically significant comparisons. Unless otherwise specified, performance comparisons were made with the best-performing parameter setting that was found for each of the algorithms-group size combinations.

Due to the computational requirements of tuning the parameters of very large simulations it was infeasible to tune parameters for our experiments with a group size of 400 robots. Instead, these experiments were performed at the parameter settings found for the 36 robots.

[vi] `http://sourceforge.net/projects/mobat/`

Each algorithm variant has its own parameters that need tuning. These parameters and the range within which they were tuned are listed in Table 7.4.

Parameter	*Tunable range*
τ (Evaluation time steps per candidate)	300 – 600
μ (Population size – for encapsulated and hybrid schemes)	3 – 15
$\sigma_{initial}$ (Initial mutation step size)	0.1 – 10.0
ρ (Re-evaluation rate)	0.0 – 1.0
Crossover rate	0.0 – 1.0
Mutation rate	0.0 – 1.0
Newscast item TTL (for newscast-based EvAg variants)	3 – 20
Newscast cache size (for newscast-based EvAg variants)	2 – group size

Table 7.4 – The parameters as tuned by REVAC.

7.2.8 Results and Discussion

Fig. 7.6 shows the results for the experiments for group sizes 4, 16, 36 and 400. Each graph shows the results for $(\mu + 1)$ ON-LINE (*encapsulated*), for EvAg (*distributed*) and for EvAg's *hybrid* extension. The latter two have results for the panmictic as well as the newscast variant. The white circles indicate the average performance (over 25 repeats) of the best parameter vector for that particular algorithm variant and the whiskers extend to the 95% confidence interval using a t-test. To investigate how sensitive the algorithm variants are to the choice of parameter settings, we also show the performance variation in the top 5% of parameter vectors, indicated by the grey ovals. The results are normalised so that the highest performance attained overall is 1. Note that because –as discussed above– we only ran $(\mu + 1)$ ON-LINE experiments with group size 4, the data for $(\mu + 1)$ ON-LINE are the same in all four graphs.

The performances shown are always the average performance of the entire group of robots in an experiment, not just the performance of the best robot: we are interested in developing an algorithm that performs well for *all* robots, rather than an algorithm that has a very high peak performance in one robot but does not succeed in attaining good performance for the entire group. Performances are compared using a t-test with $\alpha = 0.05$.

In the scenario with four robots (Fig. 7.6(a)) the hybrid algorithm significantly outperforms the encapsulated scheme, but it is not significantly better than the

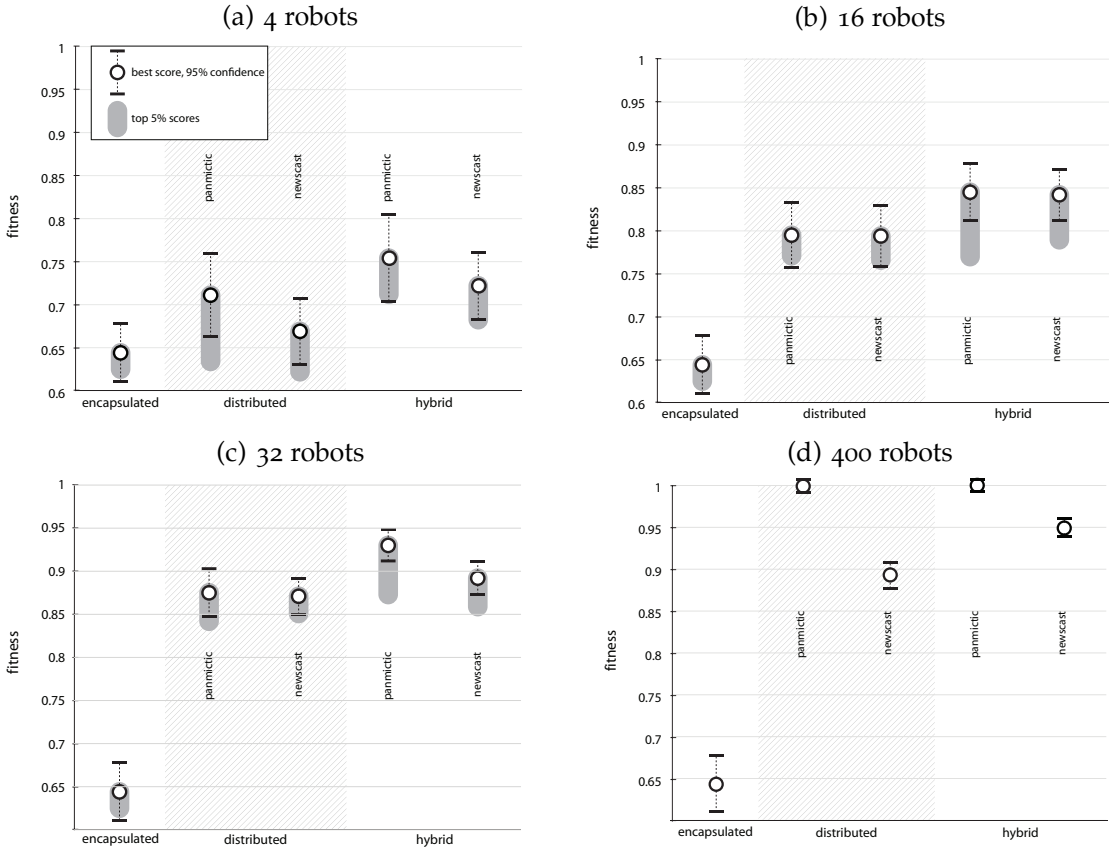


Figure 7.6 – Performance plots for various group sizes: performance is averaged over 25 repeats. Note that there was no tuning for group size 400 and hence no top 5% parameter vectors.

distributed scheme. The differences between the distributed and the encapsulated scheme are not significant. The same goes for the differences between the panmictic and the newscast variants. All four EvAg variants show a relatively large variation in the top-5% performances, indicating that they are more sensitive to the quality of parameter settings than is $(\mu + 1)$ on-line.

It is surprising that the distributed scheme matches (even improves, although not significantly) performance with the encapsulated scheme when we consider that four robots running EvAg have access to only four (shared) genotypes, compared to the 12 (isolated) genotypes that are stored by each of the robots running $(\mu + 1)$ on-line (with μ tuned to 12).

With group size 16 (Fig. 7.6(b)), EvAg starts to come into its own: both the distributed and the hybrid schemes outperform $(\mu + 1)$ on-line significantly, although the differences among the EvAg variants are not significant at 95%. More-

over, the distributed variants now perform almost as consistently as their encapsulated counterpart: both variation in scores for a single parameter setting and the variation of scores in the top-5% are at the level of $(\mu + 1)$ ON-LINE, indicating that the distributed algorithm becomes less sensitive to sub-optimal parameter settings as the number of robots participating in the evolution grows.

With the increase in group size from 16 to 36 (Fig. 7.6(c)) EVAG again shows a significant performance increase and the confidence intervals and range of top-5% values are further reduced. For the first time we see a significant difference between the EVAG variants, with the panmictic hybrid approach performing significantly better than the alternatives.

Finally, Fig. 7.6(d) shows the results for 400 robots. The computational requirements of so large a simulation make parameter tuning infeasible; instead we investigate the performance of the algorithms at the best parameter settings found for the 36-robot scenario. The plot shows a large jump in performance for the panmictic variants, a respectable increase for the hybrid newscast variant, but little difference for the distributed newscast implementation. For all EVAG variants the confidence interval of the scores has shrunk considerably, indicating that EVAG continues to become more reliable as the robot group size increases.

The results show that, as the group size increases, there is an increasing benefit to using an algorithm such as (the hybrid extension of) EVAG rather than a purely encapsulated algorithm such as $(\mu + 1)$ ON-LINE. In particular, the hybrid scheme consistently reaches the highest performance (although the difference with the distributed scheme is rarely shown to be significant at 95%). EVAG's panmictic variants perform consistently better than its newscast-based version, but the difference is only significant at the largest group sizes we consider. In truly large-scale environments it would be infeasible to have a panmictic population structure; unfortunately it is especially in the large-scale 400-robot scenario that the newscast-based EVAG lags behind its panmictic counterpart.

One possible explanation for the lower performance of the newscast-based EVAG is that it has two extra parameters to tune (newscast cache size and the news items' TTL), making it more difficult for REVAC to find optimal parameter settings within the 400 attempts that we allowed it. However, this does not explain the large performance gap in the 400-robot scenario, where the same parameter settings as in the 36-robot scenario were used. One suspect is the 'newscast cache size' parameter, of which an interesting trend can be seen when looking at the

progression of the value across the different scenarios: as the number of robots grows, so does the value of the cache size parameter. Earlier research on newscast suggests that a cache size of 10 should be sufficient for very large-scale applications Jelasity and van Steen (2002), but nevertheless REVAC favours higher numbers, approximately in the range of $\frac{3}{4}$th of the number of robots. To see if the setting of a cache size of 27 is sub-optimal for the 400-robot scenario we have investigated if increasing the cache size to 300 leads to an improved performance; this turned out not to be the case, with both cache sizes performing at the same level. This indicates that a lack of cache space is not to blame for the gap in performance between newscast-based EvAg and its panmictic counterpart and its exact cause remains unknown.

7.2.9 Conclusion

In this section, we have compared the $(\mu + 1)$ ON-LINE on-board encapsulated algorithm to a distributed and a hybrid implementation of EvAg for on-line, on-board evolution of robot controllers. We have performed simulation experiments to investigate the performance of these algorithms, using automated parameter tuning to evaluate each algorithm at its *best possible* parameter setting.

Comparing the algorithms in terms of performance, EvAg performs consistently better than $(\mu + 1)$ ON-LINE, with the effect being especially prominent when the number of robots participating in the scenario is large. Even at the smallest group size we considered, the distributed scheme is competitive, despite having a population of only four individuals.

To estimate the sensitivity to sub-optimal parameter settings we have observed the variation in performance in the top-5% of parameter vectors. We have seen that $(\mu + 1)$ ON-LINE is quite stable, with the top-5% close to the best performance. In both the distributed and the hybrid variant, EvAg's performance is quite unstable when the group of robots is small, but becomes increasingly reliable as the group size increases.

For large numbers of robots, the newscast population structure has a small negative effect on the performance of the EvAg variants when compared to a panmictic population structure. However, in a large-scale scenario it may be infeasible to maintain a panmictic population structure. In those scenarios the small performance loss when using a newscast-based population structure may well be

outweighed by the practical advantages of being able to implement the algorithm at all.

Although we have only performed experiments with a single and quite straightforward task, we conclude that both the distributed and hybrid approaches to on-board on-line evolutionary algorithms in evolutionary robotics are feasible and provide a promising direction for research in this field.

The hybrid scheme can be preferred over the encapsulated and the distributed case because it efficiently harnesses the opportunities of parallelising the adaptation process over multiple robots while performing well even for small numbers of robots. Although the panmictic variant does outperform the newscast-based implementation for very large numbers of robots, we do not know if or how tuning specifically for 400 robots would have influenced the apparent performance difference between newscast and panmixia. We would still prefer the former because of its inherent scalability and robustness.

Future research should confirm our findings in different scenarios. Additionally, there is research to be done regarding the study of which parameters are influential and why certain parameter settings are more effective than others.

7.3 Migration Policies for Hybrid On-line Evolution of Robot Controllers

We investigate on-line on-board evolution of robot controllers based on the so-called hybrid approach (island-based). Inherently to this approach each robot hosts a population (island) of evolving controllers and exchanges controllers with other robots at certain times. We compare different exchange (migration) policies in order to optimize this evolutionary system and compare the best hybrid setup with the encapsulated and distributed alternatives. We conclude that adding a difference-based migrant selection scheme increases the performance.

7.3.1 Introduction

Evolutionary robotics concerns itself with evolutionary algorithms to optimise robot controllers (Nolfi and Floreano, 2000). Traditionally, robot controllers evolve in an off-line fashion, through an evolutionary algorithm running on some computer searching through the space of controllers and only calling on the actual robots when a fitness evaluation is required. To distinguish various options regarding the evolutionary system Eiben et al. proposed a naming scheme based on *when*, *where* and *how* this evolution occurs (Eiben et al., 2010a). The resulting taxonomy distinguishes between design time and run-time evolution (off-line vs. on-line) as well as between evolution inside or outside the robots themselves (on-board vs. off-board). In a system comprising of multiple robots, there are three options regarding the *'how'*:

Encapsulated: A population of genotypes encoding controllers evolves inside each robot independently, without communication with other robots.

Section 7.3 was published as:

P. García-Sánchez, A. E. Eiben, E. Haasdijk, B. Weel and J.J. Merelo (2012). Testing diversity-enhancing migration policies for hybrid on-line evolution of robot controllers. In Di Chio et al., *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, Pages 52–62, Springer-Verlag, Berlin/Heidelberg.

Distributed: Each robot carries a single genotype and reproduction requires the exchange of genotypes with other robots. The evolving population is formed by the combined genotypes of all the robots.

Hybrid: Each robot has its own locally evolving population and there is exchange of genotypes between robots. In terms of parallel evolutionary algorithms, this can be seen as an island-model evolutionary algorithm with migration.

In this section we investigate aspects of the hybrid approach: we test the effects of the *migration policy* (migration of the best, random, or most different individual), the *admission policy* (always accept the migrant, or accept only after re-evaluation) and the *island topology* (ring vs. fully connected). Furthermore, we look into these effects for different numbers of robots (4, 16 or 36).

Specifically, our research questions are:

- Using the hybrid approach (island model), which is the best combination of migration policy, admission policy, and island topology?
- Is this combination better than the encapsulated and distributed alternatives?

The rest of the work is structured as follows: after the state of the art, we present the developed algorithms and experimental setting. Then, the results of the experiments are shown (Section 7.3.4), followed by conclusions and suggestions for future work.

7.3.2 State of the art

Migration among otherwise reproductively isolated populations has been proven to leverage the inherent parallelism in evolutionary algorithms, not only by obtaining speed-ups, but also by increasing the quality of results, since the reproduction restrictions inherent in the division of the population into islands is a good mechanism to preserve population diversity, as shown in, for instance, Cantú-Paz (2001).

To improve population diversity in an island model evolutionary algorithm, the MultiKulti algorithm (Araujo and Merelo, 2011) takes the genotypic differences of individuals when selecting migrants into account. It is based in the idea that the inflow of migrants that differ from the rest of an island's population increases diversity and thus improves performance. An island requests a migrant from one of its neighbours by sending a genotype that represents the population. This can either be the the best individual (based in the assumption that when a population

tends to converge after a few generations, the best is a fair representation of the whole population) or a *consensus sequence* (the most frequent allele in each position of the genotype using binary genomes). In answer, an island selects the most different genotype in either its whole population or the top individuals (the elite). In their experiments, the islands were connected in a ring topology, with migration taking place asynchronously. Results of the experiments performed in Araujo and Merelo (2011) show that MultiKulti policies outperform classic migration policies (send the best or random individuals from the population), especially with a low number of individuals but larger number of islands. It is shown to be better to send the consensus as a representation and that sending the most different of a well-chosen elite (those with the best fitness) is better than sending the most different overall.

On-line evolutionary robotics has been studied in works like Nordin and Banzhaf (1997), where genetic programming was used to evolve a robot in real time, and Watson et al. (2002), where several robots evolve at the same time, exchanging parts of their genotypes when within communication range. Huijsman et al. (2011) compare an encapsulated and a distributed version; the latter is implemented as a variant of EvAg (Laredo et al., 2010), where each robot has one active solution (genotype) a cache of genotypes that are active in neighbouring robots. Parents are selected through a binary tournament in each robot's cache. If the new solution (candidate) is better than the active, it replaces the active solution. The work compares this algorithm with a panmictic version, where parents are selected (again using binary tournament) from the combined active solutions of all robots.

One of the peculiarities of evolutionary robotics, particularly on-line, is that the fitness evaluations are very noisy (Haasdijk et al., 2010). The conclusions in Araujo and Merelo (2011), however, are based on experiments with noiseless fitness functions, so we cannot take these conclusions for granted in on-line evolutionary robotics and we have to test the MultiKulti algorithm in our particular setting.

7.3.3 Algorithms and Experimental Setup

We carried out our experiments with e-puck like robots simulated in the RoboRobo simulator[vii]. The robot is controlled by an artificial neural net with 9 inputs (cor-

[vii] `http://www.lri.fr/~bredeche/roborobo/`

responding to the robot's distance sensors and a bias node), 2 outputs (wheel speeds). Genetically, this was represented as a vector coding the network's 18 weights. All algorithms were evaluated using the *Fast Forward* task and next fitness function:

$$f = \sum_{t=0}^{\tau} (v_t \cdot (1 - v_r)) \tag{7.5}$$

where v_t and v_r are the translational and the rotational speed, respectively. v_t is normalised between -1 (full speed reverse) and 1 (full speed forward), v_r between 0 (movement in a straight line) and 1 (maximum rotation). Whenever a robot touches an obstacle, $v_t = 0$, so the fitness increment during collisions is 0. There is more information about this function in Huijsman et al. (2011). This fitness is noisy: a controller configuration can produce different fitness values depending on the robot's position in the arena when evaluation starts. The robots are placed in a small maze-like arena (Fig. 7.8). To ensure a fair comparison across different numbers of robots, each robot is placed in a separate instance of the arena to avoid physical interaction between robots. Robots can communicate across arenas instances.

In our experiments, we compare three algorithms:

Encapsulated evolutionary algorithm The encapsulated algorithm we use is the $\mu + 1$ on-line algorithm presented in Haasdijk et al. (2010). Here, each robot runs a stand-alone evolutionary algorithm with a local population of μ individuals. In each cycle, one new solution (controller) is created and evaluated. This solution replaces the worst individual of the population if it has higher fitness. To combat the effects of noisy evaluations, an existing solution can be re-evaluated, instead of generating and testing a new one, depending on the re-evaluation rate ρ.

Distributed evolutionary algorithm As a benchmark distributed algorithm we use the panmictic algorithm presented in Huijsman et al. (2011). Here, a single controller is present in each robot. New controllers are created using the controllers of two robots as parents. In each iteration, a robot randomly selects two others to create a new chromosome by crossover and mutation. If the new chromosome is better, it replaces the actual one.

Hybrid evolutionary algorithm This algorithm is an adaptation of the $\mu + 1$ on-line algorithm that includes a migration mechanism to exchange genotypes among robots (every robot is an island) as shown in Fig. 7.7. We test two migrant acceptance mechanisms: a migrant can be added to the local population either

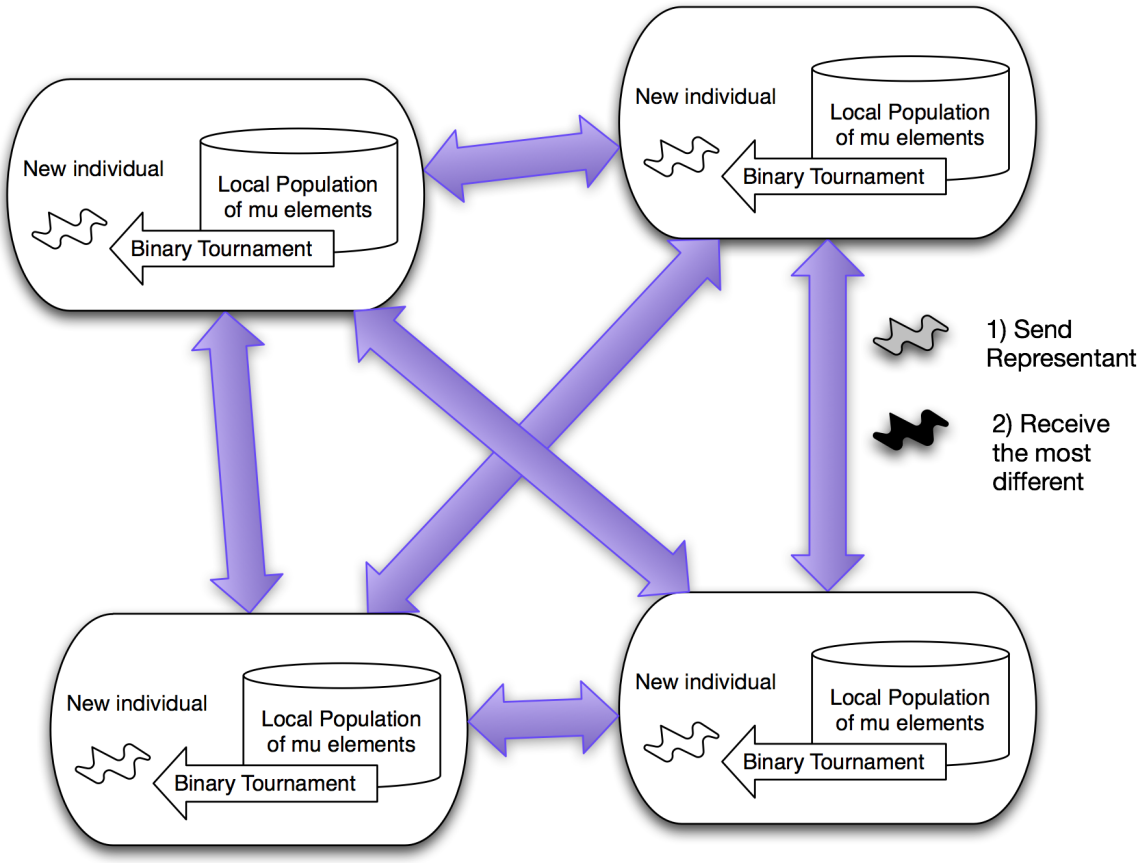


Figure 7.7 – Migration mechanism: each robot has a local population and in each migration cycle request a different type individual from others robots' populations. If MultiKulti is used, then a message is sent (gray genotype) to receive the most different (black genotype).

regardless of its fitness (to give it a chance to be selected) or only if it is better than the worst.[viii]

Each experiment lasts for 50,000 evaluation steps.

In on-line evolution, the robots train on the job: this means that the *robot's* performance is not (only) determined by the best individual it stores at any one time, but by the joint performance of all the candidate controllers it considers over a period. Therefore, we assess the algorithms' performance using the average of the last 10% evaluations over all robots.

[viii] Source code of the presented algorithms is available in `http://atc.ugr.es/~pgarcia`, under a GNU/GPL license.

As stated in Smit and Eiben (2009), an algorithm's parameters should be tuned to obtain (approximately) the best possible parameter settings and so ensure a fair comparison between the best possible instances of the algorithms. We used Bonesa (Smit and Eiben, 2011) to tune the parameters for the algorithms we investigate in the following configurations:

- Number of robots: executions with 4, 16 and 36 robots have been performed.
- Migrant selection: select the *Best*, *random* or *most different* (MultiKulti) individual as a migrant.
- Admission policy: when a new migrant arrives, it is evaluated and accepted only if is better than the worst (*no-replacement*) or accepted regardless, always replacing the worst of the population (*replacement*).
- Topology: migration can move between neighbours and the islands are arranged in a *ring* or in a *random* topology, which is rewired after every evaluation.

We conducted Bonesa runs for each possible combination of these configurations to tune the settings for canonical parameters (e.g., mutation step size, crossover rate) and the following more specific parameters:

Along the canonical GA parameters (like mutation or crossover rate) the MultiKulti parameters to study are the next:

- Migration rate: likelihood of migration occurring per evaluation cycle.
- Best rate: probability of representing the population with the best individual or with a consensus sequence (average of genes). This parameter applies only for MultiKulti instances.
- Elite percentage: the size of the elite group to select the migrant from (if 1, receive the most different of all the population). This parameter applies only for MultiKulti instances.

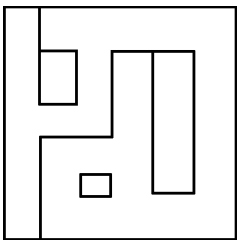


Figure 7.8 – Arena used in the experiments.

Population size μ was fixed to 10 individuals to isolate the interactions between the other parameters. Figure 7.5 lists all tuned parameters and their ranges. For the final analysis, we ran 50 iterations of each configuration with the parameters set to those reported as optimal by Bonesa.

7.3.4 Results and Analysis

7.3.4.1 Comparing Migration Configurations

Parameter Name	*Range*
Evaluation steps	300-600
Mutation step size	0.1-10
μ	10
Re-evaluation rate	0-1
Crossover rate	0-1
mutation rate	0-1
migration rate	0-1
elite percentage	0-1
best Rate	0-1

Table 7.5 – Parameters to tune.

The first question we asked ourselves was "which is the best combination of migration policy, admission policy, and island topology?" To answer this question, we analyse the results as reported by Bonesa for each of the configurations we considered. Table 7.6 shows the best parameters obtained for all configurations with 4, 16 and 36 robots. We discuss the results in the following four paragraphs, each discussing the results for one combination of admission policy and island topology.

Replacement Admission Policy and Panmictic Topology In all cases, the re-evaluation, crossover, mutation and migration rates are very high. Also, EliteSize is almost 1 everywhere: the migrant is selected from almost the whole population. It also turns out that is better to send a consensus sequence rather than the best individual as a representative of the population (bestRate has low values). There is no clear trend for migration rate.

Replacement Admission Policy and Ring Topology Changing the island topology to a ring arrangement, three settings change materially: as can be see in Table 7.6 for MultiKulti with 4 and 16 robots, the migration rate is much lower, but for 36 robots it remains very high. Also, but only for 4 robots, BestRate is higher (send the best individual as representative, not the consensus sequence).

Replacement admission policy and panmictic topology									
	4 ROBOTS			16 ROBOTS			36 ROBOTS		
	MK	RANDOM	BEST	MK	RANDOM	BEST	MK	RANDOM	BEST
evolutionSteps	345	310	312	310	306	425	538	561	584
stepSize	9.038	9.874	5.38	8.804	8.786	9.199	4.842	8.096	9.684
reEvaluation	0.868	0.72	0.739	0.619	0.812	0.949	0.964	0.751	0.777
Crossover	0.926	0.816	0.929	0.017	0.879	0.917	0.963	0.915	0.941
Mutation	0.943	0.977	0.936	0.98	0.839	0.909	0.937	0.923	0.938
Migration	0.809	0.989	0.958	0.987	0.499	0.993	0.956	0.988	0.567
EliteSize	0.849	-	-	0.988	-	-	0.995	-	
BestRate	0.04	-	-	0.192	-	-	0.181	-	
Replacement admission policy and ring topology									
	4 ROBOTS			16 ROBOTS			36 ROBOTS		
	MK	RANDOM	BEST	MK	RANDOM	BEST	MK	RANDOM	BEST
evolutionSteps	304	319	312	304	311	372	554	589	573
stepSize	9.29	8.149	8.769	7.008	7.37	9.953	9.465	9.307	9.94
reEvaluation	0.868	0.749	0.792	0.953	0.721	0.861	0.935	0.705	0.939
Crossover	0.999	0.983	0.96	0.83	0.955	0.455	0.996	0.848	0.991
Mutation	0.986	0.952	0.691	0.914	0.809	0.889	0.971	0.777	0.98
Migration	0.597	0.892	0.974	0.559	0.624	0.996	0.988	0.816	0.955
EliteSize	0.49	-	-	0.93	-	-	0.827	-	
BestRate	0.862	-	-	0.172	-	-	0.145	-	
No-replacement admission policy and panmictic topology									
	4 ROBOTS			16 ROBOTS			36 ROBOTS		
	MK	RANDOM	BEST	MK	RANDOM	BEST	MK	RANDOM	BEST
evolutionSteps	305	304	308	302	304	306	567	362	516
stepSize	9.895	4.04	9.547	9.731	9.146	9.8	8.832	7.526	9.988
reEvaluation	0.385	0.039	0.489	0.449	0.291	0.692	0.048	0.344	0.528
Crossover	0.828	1	0.934	0.847	0.945	0.671	0.31	0.822	0.963
Mutation	0.976	0.927	0.899	0.849	0.958	0.969	0.921	0.986	0.879
Migration	0.577	0.788	0.72	0.658	0.757	0.577	0.835	0.753	0.7
EliteSize	0.279	-	-	0.716	-	-	0.911	-	-
BestRate	0.198	-	-	0.703	-	-	0.013	-	-
No-replacement admission policy and ring topology									
	4 ROBOTS			16 ROBOTS			36 ROBOTS		
	MK	RANDOM	BEST	MK	RANDOM	BEST	MK	RANDOM	BEST
evolutionSteps	325	302	323	314	303	306	600	375	581
stepSize	9.821	9.726	9.731	9.925	8.44	9.329	9.661	9.686	9.493
reEvaluation	0.045	0.007	0.53	0.044	0.332	0.505	0.752	0.317	0.396
Crossover	0.311	0.51	0.933	0.286	0.986	0.867	0.963	0.992	0.952
Mutation	0.983	0.805	0.873	0.751	0.964	0.889	0.93	0.869	0.913
Migration	0.533	0.517	0.554	0.662	0.706	0.685	0.59	0.71	0.698
EliteSize	0.772	-	-	0.952	-	-	0.413	-	-
BestRate	0.018	-	-	0.624	-	-	0.061	-	-

Table 7.6 – Parameters obtained with Bonesa for all admission policies and topology configurations.

No-replacement Admission Policy and Panmictic Topology When changing the replacement policy a remarkable decrease can be seen in the migration rate and,

more importantly, the re-evaluation rate across the board. For 4 robots, EliteSize is much lower than in all three other combinations of admission policy and topology.

No-replacement Admission Policy and Ring Topology Apart from lower migration rates for most of the policies and a drop in EliteSize for 16 robots, Bonesa reports similar values for this combination of admission policy and topology and the previous one. For 4 robots, EliteSize again has a high value.

7.3.4.2 Comparing Performance

Figures 7.9(a), 7.9(b) and 7.9(c) show box plots summarising 50 repeats of each configuration, grouped by number of robots.

Although in terms of performance levels there is no clear trend it is clear that the admission policy does have an appreciable impact: choosing the no-replacement admission policy always leads to a marked decrease in performance variation, with an increase of minimum performance. So we can conclude that evaluating an immigrant and only admitting it if it outperforms the worst individual in the population leads to more consistent performance with fewer very poor results.

Combined with the no-replacement admission policy, MultiKulti is either the best or at a par with the best migrant selection scheme, especially as the number of robots increases.

Finally, the ring topology shows a slight, but not always significant, drop in performance. This may be explained by the fact that in a ring topology, good solutions spread over the islands at a much slower rate than in a randomly connected topology.

Selecting migrants randomly seems always to lead to a smaller spread in performance than either selecting the best or the most different. Vis a vis the MultiKulti algorithm at least, this makes sense because this specifically aims at increasing population diversity, so a larger variation in performance is to be expected.

To conclude, we select a configuration with no-replacement admission policy, MultiKulti migrant selection and a random island topology to compare with the encapsulated and distributed algorithms.

	4 ROBOTS			16 ROBOTS			36 ROBOTS		
	μ+1	Distr	MK	μ+1	Distr	MK	μ+1	Distr	MK
evolutionSteps	308	303	305	308	301	302	308	583	567
stepSize	9.615	4.306	9.895	9.615	5.621	9.731	9.615	8.197	8.832
reEvaluation	0.091	0.647	0.385	0.091	0.558	0.449	0.091	0.002	0.048
Crossosver	0.19	0.399	0.828	0.19	0.122	0.847	0.19	0.1	0.31
Mutation	0.978	0.908	0.976	0.978	0.86	0.849	0.978	0.606	0.921
Migration	-	-	0.577	-	-	0.658	-	-	0.835
EliteSize	-	-	0.279	-	-	0.716	-	-	0.911
BestRate	-	-	0.198	-	-	0.703	-	-	0.013

Table 7.7 – Parameters obtained with Bonesa for the encapsulated, distributed and hybrid algorithms.

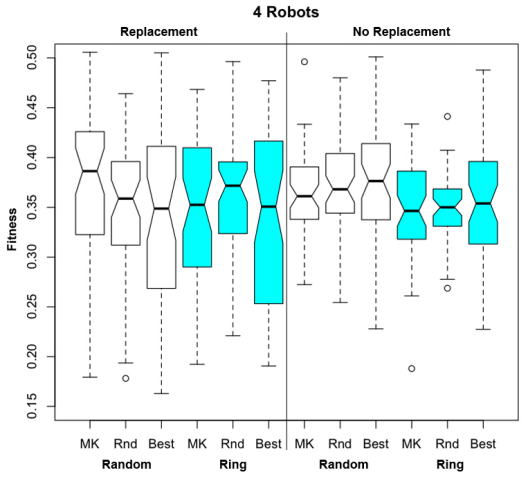
7.3.4.3 Comparing Encapsulated, Distributed and Hybrid On-line Evolution

The second question we asked ourselves is whether the optimal hybrid instance we selected in the previous section outperforms its encapsulated and distributed counterparts. Table 7.7 shows the settings that Bonesa reported as optimal for the three algorithms that we compare for groups of 4, 16 and 36 robots. Note that the optimal parameters for $\mu + 1$ have only been calculated once: since there is no interaction between robots, $\mu + 1$'s performance and settings are independent of the number of robots. Running 50 repeats with these settings resulted in performances as reported in Figure 7.9(d).

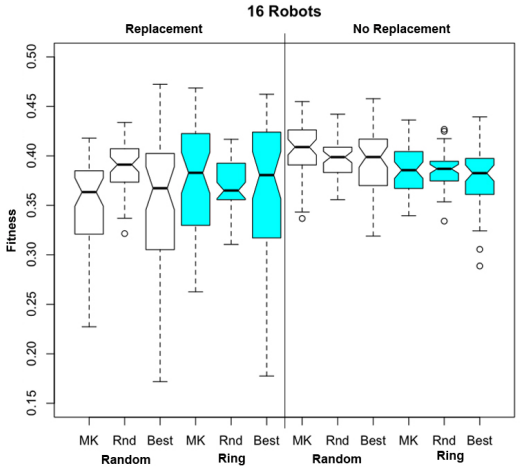
Even for as small a number of robots as 4, the distributed and hybrid algorithms both significantly outperform the encapsulated algorithm. The difference between the algorithms that share the population across robots is only significant for 36 robots, but even there not material. The difference with the encapsulated algorithm may lie in the exploitation of evolution's inherent parallelism, but we think this is also due to the increased diversity that stems from dividing the total population across islands. This would explain the large benefit of communication even for small numbers of robots, where the distributed algorithm actually has a smaller total population than the individual robots with $\mu + 1$.

Set to the best found parameters, the hybrid algorithm causes much less communication overhead than the distributed algorithm: the latter shares genotypes among all robots at every evaluation, while the hybrid algorithm has comparatively low migration rates (0.577, 0.658 and 0.835). This reduction of communication cost comes at no significant loss of performance, and even a significant gain for 36 robots.

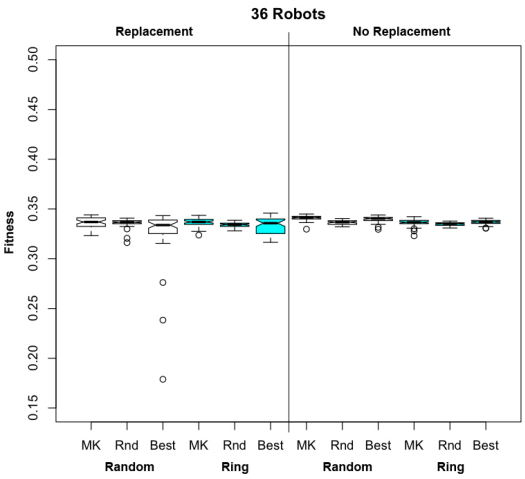
(a) Box plot of all hybrid configurations with 4 robots.



(b) Box plot of all hybrid configurations with 16 robots.



(c) Box plot of all hybrid configurations with 36 robots.



(d) Box plot comparing $\mu + 1$, distributed and multikulti migration with replacement for 4, 16 and 36 robots.

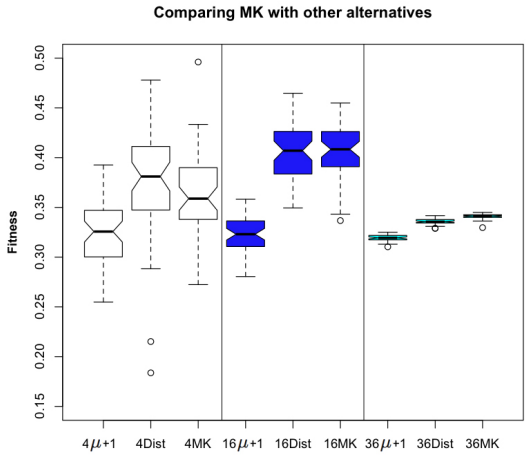


Figure 7.9 – Box plots of executing each algorithm with the best parameters obtained with Bonesa 50 times.

7.3.5 Conclusions and future work

In this section, we compared combinations of migrant selection schemes, migrant admission policies and island topologies in a hybrid algorithm for on-line, on-board Evolutionary Robotics. Results show that the migrant admission policy –which determines when a migrant is admitted into the population– is more important in performance than migrant selection or the island topology. But the most important finding is that adding migration between robots significantly and materially increases performance. We have demonstrated that adding a difference-based migrant selection scheme (MultiKulti) leads to optimal or at least near-optimal performance compared to another migration mechanisms. This migration mechanism can compete with the on-line distributed algorithm, where only an individual per robot exist, even with a lower number of data transmissions. Our aim is to continue exploring other techniques, like a self-adaptative migration mechanism to ask for new migrants when the population stagnates and perform new tests for new tasks other than the Fast-Forward. New experiments with different number of individuals in the local population also will be carried out. Also, further investigation will be performed in swarming and cooperation techniques among robots, with different communication mechanisms.

7.4 The Emergence of Multi-Robot Organisms using On-line On-board Evolution

We investigate whether a swarm of robots can evolve controllers that cause aggregation into 'multi-cellular' robot organisms without a specific reward to do so. To this end, we create a world where aggregated robots receive more energy than individual ones and enable robots to evolve their controllers on-the-fly, during their lifetime. We perform experiments in six different implementations of the basic idea distinguished by the system of energy distribution and the level of advantage aggregated robots have over individual ones. The results show that 'multi-cellular' robot organisms emerge in all of these cases.

7.4.1 Introduction

Swarm-robot systems and (self-)reconfigurable modular robot systems paradigms have been invented to facilitate multi-purpose robot design. Swarm-robot systems use large numbers of autonomous robots which cooperate to perform a task (Mondada et al., 2004). Similarly, self-reconfigurable modular robot systems use many modules to form a larger, reconfigurable, robot that can tailor its shape to suit a particular task (Yim et al., 2007a). These two subjects have largely been studied separately, with swarm robotics focusing on cooperating robots which do not assemble into an organism. Research in reconfigurable modular robotics focuses on creating actual modules, finding suitable morphologies for a task, and reconfiguring from one morphology to another.

Recently, a new kind of self-reconfigurable robots has been proposed based on modules that are also capable of autonomous locomotion (Kernbach et al., 2009a). In such a system, the modules can form a swarm of autonomous units as well as a 'multi-cellular' robot organism consisting of several physically aggregated units.

Section 7.4 was published as:

Berend Weel and Evert Haasdijk and A.E. Eiben (2012). The Emergence of Multi-Robot Organisms using On-line On-board Evolution. In Di Chio et al., *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, Pages 124–134, Springer-Verlag, Berlin/Heidelberg.

To date, there has been very little research on self-assembly – the transition from swarm to organism – as emergent, not pre-programmed, behaviour.

The main subject of this section is emergent self-assembly through evolution. We are interested in the emergence of robot organisms from swarms as a response to environmental circumstances. To this end, we design environments where organisms have an advantage over individual modules and make the robots evolvable. In particular, we implement an on-line and on-board evolutionary mechanism where robot controllers undergo evolution on-the-fly: selection and reproduction of controllers is not performed by an outer entity in an off-line fashion (e.g. a genetic algorithm running on an external computer), but by the robots themselves (Haasdijk et al., 2010). One of the premises of our study is that we do not include a specific fitness measure to favour organisms. Rather, we build a system with an implicit environmental pressure towards aggregation by awarding more energy to robots in an aggregated state. The environmental advantage is scalable and we compare the effects of low, medium and high values.

The research questions we seek to answer are the following:

1. Will organisms evolve purely because the environment favours modules that are part of an organism? Or, does the system need a specific user defined fitness to promote aggregation?

2. How does system behavior depend on the level of environmental benefit? Will organisms evolve even if the extra advantage is low?

3. What are the characteristics of the evolved organisms? How large and how old do organisms become?

7.4.2 Related Work

7.4.2.1 Swarm Robotics

As mentioned, our work is situated between swarm robotics and self-reconfigurable modular robot systems. Swarm Robotics (Mondada et al., 2004) is a field that stems from Swarm Intelligence (Bonabeau et al., 1999), where swarm-robots often have the ability for physical self-assembly. Swarm-bots were created in order to provide a system which was robust towards hardware failures, versatile in performing different tasks, and navigating different environments.

Şahin (2005) categorizes tasks for which to use swarm robots as: tasks that cover a region, tasks that are too dangerous, tasks that scale up or down in time, and tasks that require redundancy. In Mondada et al. (2004), the self-assembly of s-bots allows for the navigation of crevices and objects too large for a single robot, as well as the transport of objects which are too heavy to be transported by a single robot.

7.4.2.2 Self-reconfigurable modular robot systems

Self-reconfigurable modular robot systems (SMRSs) were designed with three key motivations: versatility, robustness and low cost. The first two are identical to motivations for swarm-robots, while low cost can be achieved through economy of scales and mass production as these systems use many identical modules. The main advantage advocated is the adaptability of these systems to different tasks, however most of the research in this field is still exploring the basics: module design, locomotion and reconfiguration.

Yim et al. (2007a) gives an overview of self-reconfigurable modular robot systems, the research is mainly on creation of modules in hardware and showcasing their abilities to reconfigure and lift other modules. Most of these self-reconfigurable modular robot systems are incapable of locomotion as independent modules. In recent years however, a number of SMRSs were developed that incorporate independent mobility as a feature, for instance Wei et al. (2010); Kutzer et al. (2010); Kernbach et al. (2009a).

The SYMBRION project, of which this research is part, develops its own SMRS, exploring two alternatives for hardware, as presented by Kernbach et al. (2009a). Both versions are independently mobile and so can operate as a swarm, both also have a mechanical docking mechanism allowing the modules to form and control a multi-robot organism.

7.4.2.3 Self-Assembly

The task of multiple robots connecting autonomously is usually called self-assembly, and has been demonstrated in several cases: Groß et al. (2006); Yim et al. (2007b); O'Grady et al. (2008); Wei et al. (2010). Most of these however, are limited to pre-programmed control sequences without any evolution. In self-reconfig-

urable robots, self-assembly is restricted to the docking of two modules as demonstrated in Wei et al. (2010); Kutzer et al. (2010).

The work in this section is most closely related to that of Groß, Nolfi, Dorigo, and Tuci: Bianco and Nolfi (2004) and Groß et al. (2006), Groß and Dorigo (2008), in which they explore self-assembly of swarm robots. They evolved Recurrent Neural Networks for the control of s-bots to be capable of Self-Assembly in simulation, they then took the best controllers evolved in this manner and tested them in real s-bots. This research shows it is possible to evolve controllers which create organisms. As it is difficult to evolve controllers in a situation where either robot can grip the other, they use a target robot in most of their research. This target robot, also called a seed, bootstraps the problem of who grips who, by showing which robot should be gripped by the other robot. Furthermore they assign fitness to the s-bots based on whether they succeeded in forming an organism, or if failed the distance between the robots.

7.4.2.4 On-line On-board Evolutionary Algorithms

We use an evolutionary algorithm (EA) as a heuristic optimiser for our robot controller, as do many robotics projects. The field of evolutionary robotics in general is described by Nolfi and Floreano (2000). Eiben et al. (2010a) describe a classification system for evolutionary algorithms used in evolutionary robotics. They distinguish evolution based on *when* evolution takes place: off-line or design time vs. on-line or run time. *Where* evolution takes place: on-board or intrinsic vs. off-board or extrinsic. And *how* evolution takes place: encapsulated or centralised vs. distributed.

Whereas most evolutionary robotics research uses offline and extrinsic approaches to evolving controllers. We use an on-line on-board (or intrinsic) hybrid approach, based on EvAg (Laredo et al., 2010) and $(\mu + 1)$ on-line (Haasdijk et al., 2010). It is described in detail in Huijsman et al. (2011). Each robot maintains a population of μ individuals locally, and performs cross-over and mutation to produce offspring. These individuals can be exchanged between robots as part of the parent selection mechanism. The offspring is then instantiated as the controller for evaluation.

We do not include a task in our system other than gathering energy. Nor do we include any type of morphology engineering, or purposeful reconfiguration of an

organism. Our goal is to investigate only the very first step: forming an organism under environmental pressure.

7.4.3 System Description & Experiments

7.4.3.1 Simulator

We conduct our experiments with simulated e-puck robots in a simple 2D simulator: RoboRobo[ix]. The robots can steer by setting their desired left and right wheel speeds. Each robot has 8 sensors to detect obstacles (static obstacles as well as other robots), indicated by the lines protruding from their circular bodies in Fig. 7.10. While a such a simple 2D simulation ignores a lot of the intricacies of robots in the real world, it is still complex enough that creating intentional, meaningful, and effective organisms is not trivial and serves our purpose of investigating organism creation under environmental pressure.

7.4.3.2 Connections

In our experiments robots can create new organisms, join an already existing organism, and two existing organisms can merge into a larger organism. When working with real robots, creating a physical connection between two robots can be challenging, and movements of joints are noisy because of actuator idiosyncrasies, flexibility of materials used, and sensor noise. We choose to disregard these issues and create a very simple connection mechanism which is rigid the moment a connection is made. The connection is modelled as a magnetic slip-ring, which a robot can set to 'positive', 'negative' or 'neutral'. When robots are close enough, they automatically create a rigid connection if neither of them has the slip-ring on 'negative' and at least one of them has it on the 'positive' setting. The connection remains in place as long as these conditions hold (i.e., neither sets its slip-ring to 'negative' and at least one is set to 'positive').

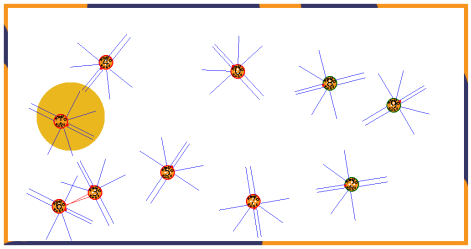


Figure 7.10 – 10 robots in an arena with a feeding ground which is also a scale, the scale regularly changes position

[ix] `http://www.lri.fr/~bredeche/roborobo/`

7.4.3.3 Environment

The robots start each evaluation cycle with a fixed amount of energy and lose energy at regular intervals to simulate power consumption by actuators. When a robots energy reaches o it is deactivated, and is unable to move for the rest of the evaluation cycle. The environment provides energy through a 'feeding ground' from which robots can gather energy by standing on top. The amount of energy gathered during evaluation is the fitness measure for on-line evolution.

There are two ways in which the environment provides an advantage to organisms over single robots. The first is that the amount of energy awarded depends on whether or not the robots are part of an organism as described below. The second advantage is that organisms can move faster, by driving in the same direction, although manoeuvring is slightly more complicated. It is important to note that there is no *direct* reward for being part of an organism; the benefit is indirectly defined by the environment. We have implemented the advantages for organisms in two different scenarios: one based on a scale metaphor, and one on a riverbed. We compare these with a separate *baseline* experiment, where organisms have no benefit.

Table 7.8 – Power gain formulas for power scale and riverbed scenarios

	Power Scale	*Riverbed*
Logarithmic	$log\left(\lvert O\rvert + 1\right) * 2$	$log\left(\frac{P}{W} * 2 * \lvert O\rvert\right)$
Linear	$(\lvert O\rvert + 1) * 2$	$\frac{P}{W} * 2 * \lvert O\rvert$
Exponential	$exp\left(\lvert O\rvert + 1\right) * 2$	$exp\left(\frac{P}{W} * 2 * \lvert O\rvert\right)$

Power Scale In a rectangular arena without stationary obstacles, there is a single feeding ground (the circle in Fig. 7.10); the environment awards energy to robots in this feeding ground. This feeding ground acts as a scale: the environment supplies more energy to modules belonging to an organism. For each organism on the scale, the gain in awarded energy increases with the organism's size $|O|$. This gain can depend linearly, logarithmically or exponentially on $|O|$ as shown in Table 7.8. Robots on the scale but not part of the organism do not affect the amount of energy received by the organism.

Riverbed In this scenario the arena is analogous to a river which pushes the robots downstream. Again, there are no obstacles. Now, the entire arena is a

feeding ground, but there is an upstream gradient: the amount of energy awarded increases as a robot finds itself more upstream. To counteract the current that pushes robots down the energy gradient, robots can aggregate into an organism: together they are faster and able to move or stay upriver, and so receive more energy. Again, this gain can increase linearly, logarithmically or exponentially as shown in Table 7.8. In the formulas used W is the width of the arena and P the position of the centre of the organism.

Baseline As a baseline, we use an experiment in which being part of an organism holds no benefit. We set up an empty environment where the robots receive a fixed amount of energy at every time-step, regardless of their position in the arena or whether they were part of an organism. There is no current driving the organisms anywhere, and organisms are not able to move faster either. The baseline can be viewed as an extension of either experiment: the power scale the scale now extends to the entire arena and no longer takes the number of robots on it into account. For the riverbed scenario, the current is reduced to 0 and the feeding ground has no gradient.

Table 7.9 – Neural Network inputs (left) and outputs (right)

inputs	*outputs*
8 Distance sensors	Vote for Random Walk
1 Size of the organism	Vote for Wall Avoidance
1 Angle to nearest feeding ground	Vote for Go to feeding ground
1 Distance to nearest feeding ground	Vote for Create organism
1 Energy Level	Magnetic ring setting
1 Bias node	

7.4.3.4 Controller

The controller consists of a feed-forward artificial neural network that selects one of 4 pre-programmed strategies based on sensory inputs. The neural net has 13 inputs (cf. Table 7.9), 5 outputs and no hidden nodes. It uses a *tanh* activation function.

The inputs are normalised: Distance sensors, Organism Size, Distance to nearest feeding ground, and Energy level are normalised between 0 and 1, angle to nearest feeding ground is normalised between -1 and 1.

The output of the neural network, as described in Table 7.9, is interpreted as follows: the first four outputs each vote for an action, the action with the highest activation level is selected. The fifth output governs the magnetic ring setting: 'negative' if the output is smaller than -0.33, 'positive' if it is bigger than 0.33 and 'neutral' otherwise.

7.4.3.5 Evolutionary Algorithm & Runs

We use a genome which directly encodes the weights of the neural net using a real-valued vector. It also includes N mutation step sizes for N genes.

Each controller is evaluated for 800 time steps, followed by a 'free' phase of 200 time steps to allow it to get out of bad situations. Each 1000 time steps therefore constitutes 1 generation. At the end of the evaluation cycle the active controller is compared to the local population, and added if it is better than the worst one. A new controller is created using either mutation or crossover.

Mutation is a Gaussian perturbation using self-adaptation. The algorithm uses averaging crossover and parents are selected using a binary tournament on the entire population across all robots (panmictic layout) as described in Huijsman et al. (2011).

At the start of the new generation, control is switched to the new controller, which potentially has a completely different setting for the magnet than the previous one, potentially destroying an organism. We ran the experiments with different reward functions described above using 10 robots. We used this number of robots as we expect to have a similar number of real robots to repeat the experiment with available from the project this research is part of. To ensure good parameter settings, we used the BONESA toolbox[x] (Smit and Eiben, 2011) to optimise settings for crossover rate, mutation rate, initial mutation step size, re-evaluation rate, and population size. Using the best parameters found, as shown in Table 7.10, we repeated each experiment 40 times, each run lasting 1000 generations.

7.4.4 Results & Analysis

The results we obtained are shown in Figs. 7.11 and 7.12. They show that random interactions already lead to some organisms, however, under influence of

[x] `http://sourceforge.net/projects/tuning/`

environmental pressure the organisms become much larger and older than without. The amount of pressure did not result in large differences in either organism size or longevity. Even a small amount of environmental advantage suffices to cause significantly bigger and older organisms – the logarithmic reward function that implements the least pressure in our comparison may not even represent the minimum amount of pressure needed.

Comparing results for the two scenarios, the riverbed scenario leads to bigger and older organisms than the power scale scenario – a markedly larger difference than that between logarithmic, linear and exponential benefit. This indicates that the environmental pressure is determined by more than only the reward functions we tested: since there is no quantitative difference between the reward functions in each scenario, the difference in organism longevity and size can only be caused by other, qualitative, differences between the scenarios. This shows that the design of the environment itself can be more important than the specific function that determines the environmental benefit of being in an organism.

7.4.4.1 Organism size

Figure 7.11 shows the mean organism size of a generation averaged over 40 runs for the power scale experiment on the left and the riverbed experiment on the right. The x-axis is the time measured in generations. The y-axis displays the mean number of robots in an organism. The raw data is plotted as a grey line for each fitness scaling, for each we also show a second order exponential trend line. The bottom line is that of the baseline experiment.

The baseline experiment results in organisms that are not very large, on average approximately 2.3. The other experiments produce much larger organisms:

Table 7.10 – Parameters of the EA for the Riverbed (R.B.) and Power Scale (P.S.) experiments

	scaling	*re-evaluation rate*	*crossover rate*	*population size*	*mutation rate*
R.B.	logarithmic	0.62385	0.03602	3	0.57369
	linear	0.44908	0.04369	4	0.00509
	exponential	0.40806	0.03411	3	0.21407
P.S.	logarithmic	0.50294	0.93562	3	0.07154
	linear	0.54149	0.46759	3	0.06662
	exponential	0.56736	0.99323	3	0.05807
	baseline	0.62385	0.03602	3	0.57369

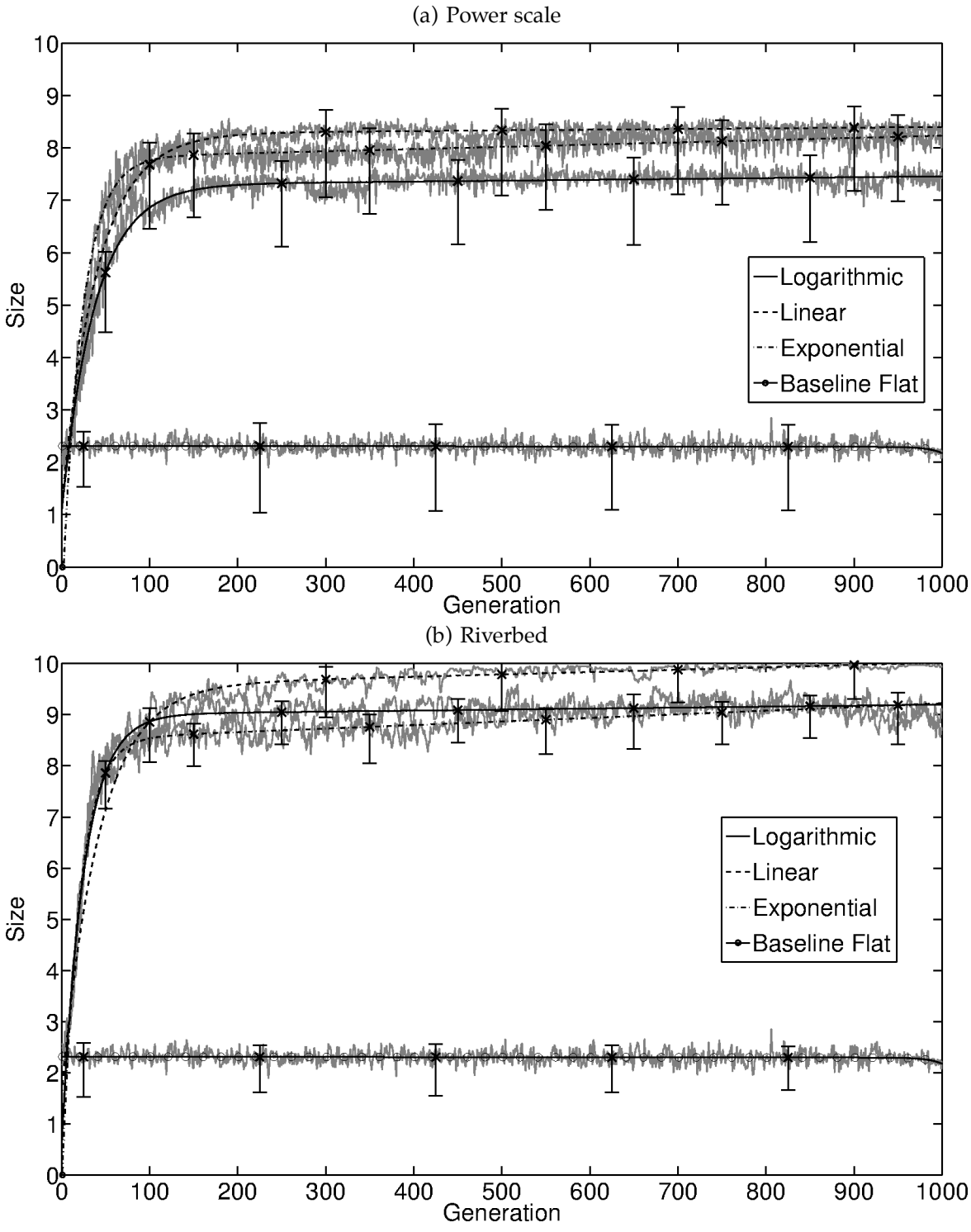


Figure 7.11 – Sizes of organisms against time (in generations) for both scenarios. The vertical bars indicate a 99% confidence interval based on a beta distribution.

averages between 7 and 9 for power scale, between 8 and 10 for riverbed. These are significantly higher than the baseline, at 99% confidence, as seen by the disjoint confidence intervals between the baseline and other plots. The difference in averages between the logarithmic, linear and exponential experiments are not significant at 99% confidence (overlapping confidence intervals).

In both experiments, environmental benefit positively influences the emergence of organisms, but the level of influence does not seem to differ between the tested reward functions. With logarithmic being the lowest reward tested we can conclude that the minimum pressure lies somewhere between no advantage and logarithmic. Note that in both scenarios the linear reward leads to the highest average. This suggests that there is a sweet-spot somewhere between logarithmic and exponential scaling.

7.4.4.2 Organism Age

Figure 7.12 shows the mean organism age for each generation averaged over 40 runs for the power scale experiment on the left and the riverbed experiment on the right. The x-axis is the time measured in generations. The y-axis displays the mean age of an organism in number of 10^5 ticks. The raw data is plotted as a grey line for each fitness scaling, for each we also show a trend line based on the fifth Fourier series. The baseline experiment's results are included in both graphs.

The lines for the three reward functions rise rapidly to values significantly higher than the baseline in both scenarios. In the power scale scenario, the average organism age reaches more than 400.000 ticks, or 400 generations. In the riverbed scenario this goes up to almost 700.000 ticks, or 700 generations for the linear reward.

The lines are rising rapidly and almost monotonously, suggesting that the organisms do not 'die' between generations. The age values in the riverbed scenario are higher than those in the power scale scenario, notable is also that the incline of the plots for riverbed are steeper than the ones for power scale.

These graphs support our earlier conclusion that the reward positively influences the size, and the age of organisms. The steeper and higher graphs of the riverbed scenario lead us to conclude that the environmental pressure is determined by more than just the reward function. It also shows that the reward has a different impact on the size of organisms than it does on the age of organisms.

Overall we conclude that, when designing experiments, more effort should go into creating an appropriate environment than into designing the reward.

We also observe that organisms do not disintegrate when switching from one controller to another, from which we can conclude that the evolutionary algorithm converges very quickly (within 50 generations) to values for a positive ring setting.

7.4.5 Conclusion & Further Research

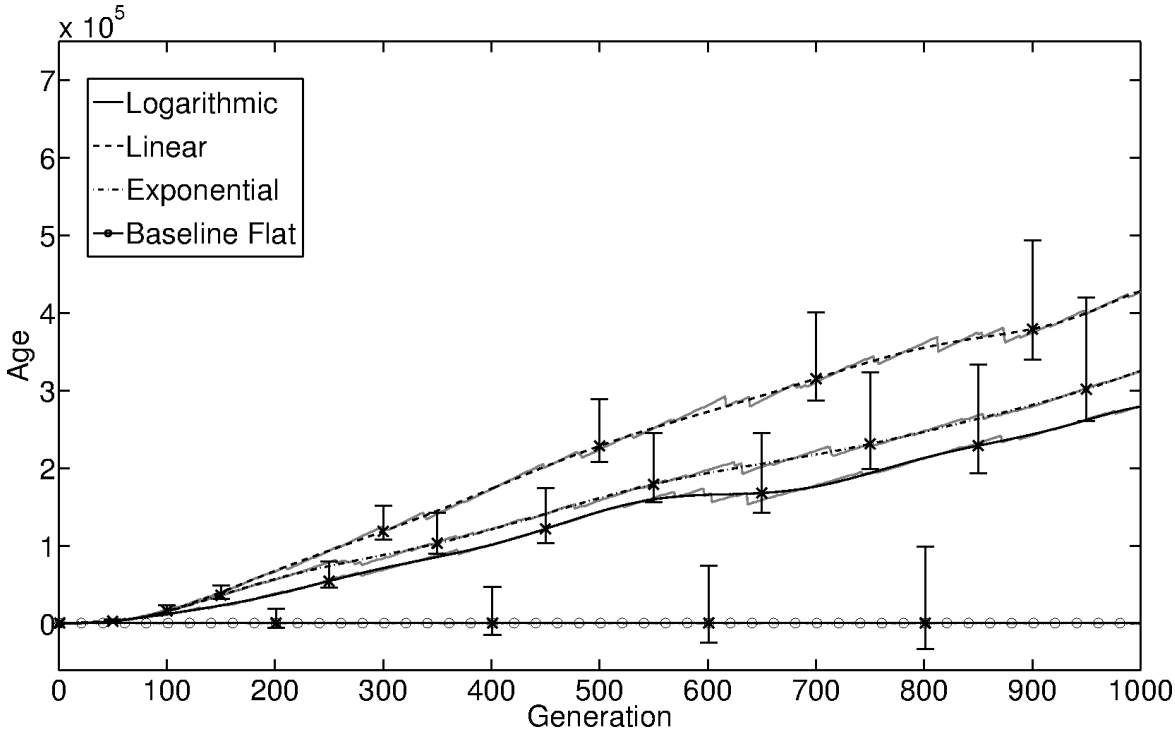
We have shown that large organisms emerge in an environment which favours modules that are part of an organism, without the need for a specific fitness function to promote aggregation. Organisms emerge even without environmental pressure by chance, but these are significantly smaller and have a significantly shorter life span.

We tested three reward functions in two separate scenarios. The amount of pressure from the reward functions did not result in large differences. Even the lowest amount of pressure, – logarithmic with respect to organism size – leads to significantly bigger and older organisms. We notice a trend: the linear reward performs slightly better than both logarithmic and exponential, suggesting an optimal setting between logarithmic and exponential. The differences between the scenarios did result in different sizes and organisms. In other words, the environmental pressure is more than just a reward function: our results also show that the design of the environment is more important when designing an experiment than the reward function.

We only used 10 robots for the experiments, this raises the question whether the same results would be obtained when using more robots. More robots would also imply a larger arena, so care should be taken to correctly scale the experiments. Furthermore, we used a controller which had pre-programmed parts that were very solution specific. To alleviate this specificity, further experiments can be executed in which the control is at a lower level, and hence the problem more difficult. Here the use of different controllers, which are more powerful, can be investigated. We observed evidence that the reward function may have an optimum, this could be further investigated to answer questions like: where is the optimum? Is it the same in different scenarios? Does finding the optimum lead to significantly bigger or longer lasting organisms?

We noted evidence that differences in qualitative environmental pressure may be more important than differences in quantitative pressure in their influence on the emergence of organisms, and therefore should be investigated. This research could be part of the upcoming discipline of complexity-engineering: harnessing emergent phenomena to create interesting or useful characteristics in complex systems. Lastly we would like to investigate the 'unlearning' of organism forming behaviour by letting controllers evolve in a changing environment which first favours organisms and over time puts organisms at a disadvantage.

(a) Power scale



(b) Riverbed

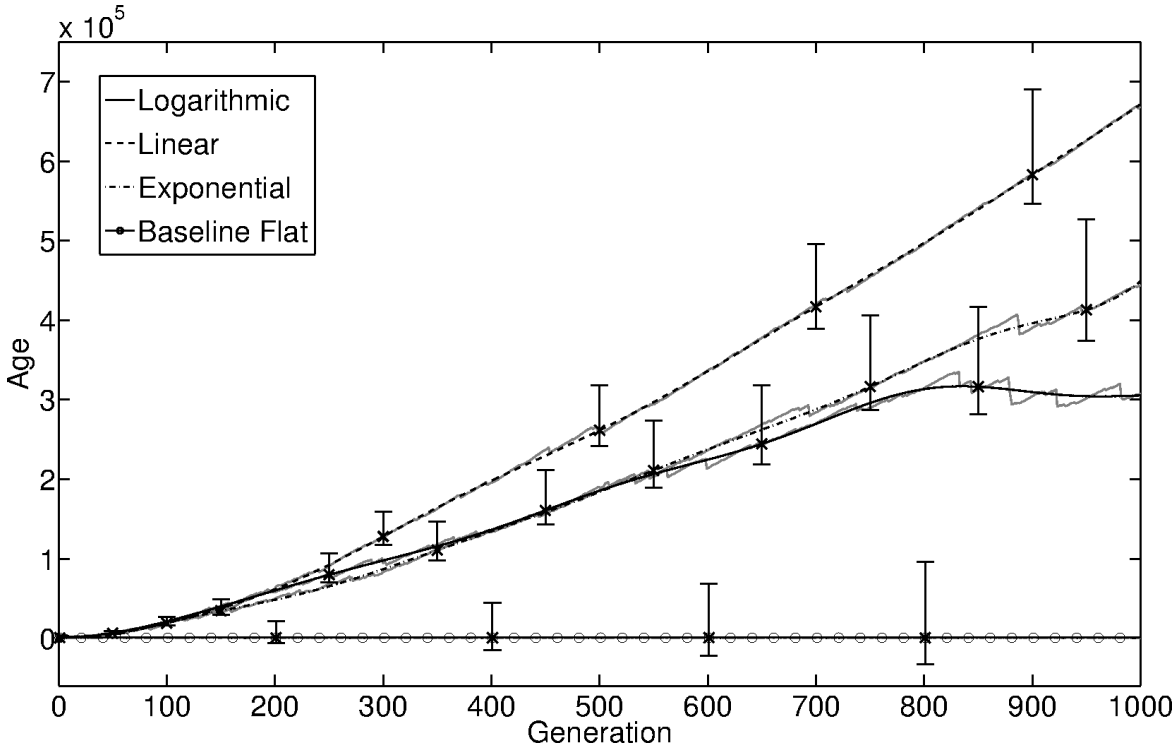


Figure 7.12 – Ages of organisms against time for both scenarios. The vertical bars indicate a 99% confidence interval based on a beta distribution.

Prediction is very difficult, especially about the future

Niels Bohr

8 It's Life, But Not As We Know It

Embodied Artificial Evolution

Evolution is one of the major omnipresent powers in the universe that has been studied for about two centuries. Recent scientific and technical developments make it possible to make the transition from passively understanding to actively using evolutionary processes. Today this is possible in digital spaces, in Evolutionary Computing, where human experimenters can design and manipulate all components of evolutionary processes. We argue that in the near future it will be possible to implement artificial evolutionary processes outside such imaginary spaces and make them physically embodied. In other words, we envision the "Evolution of Things", rather than just the evolution of digital objects, leading to a new field of Embodied Artificial Evolution. The main objective of this chapter is to

---

This chapter has been accepted for publication as:

A.E. Eiben, S. Kernbach and Evert Haasdijk (2012). Embodied Artificial Evolution – Artificial Evolutionary Systems in the 21st Century. To appear in *Evolutionary Intelligence*, Springer Verlag, Heidelberg/Berlin.

---

present a unifying vision in order to aid the development of this high potential research area. To this end, we introduce the notion of Embodied Artificial Evolution, discuss a few examples and applications, and elaborate on the expected benefits as well as the grand challenges this developing field will have to address.

8.1 Introduction

This chapter states our position on what we call embodied artificial evolution. Perhaps the best way to introduce this vision is to follow a historical perspective concerning the notion of evolution.

In the 19th century the theory of evolution was put forward to explain the emergence of Life on Earth. Thus, originally, evolution was a passive notion that helped us understand things. In the 20th century the invention of the computer made it possible to create worlds where we could actively engineer evolutionary processes. The resulting field, called Evolutionary Computing, was groundbreaking in that it converted evolution from a passive explanatory *theory* to clarify a past process into an active *tool* to create a new process. Of course, such an evolutionary computing process takes place in an imaginary space, while natural evolution takes place in the biosphere on Earth. And thus, the birth of Evolutionary Computing represents another major transition, that of transporting evolution from biological spaces to digital spaces.

Evolutionary Computing has radically changed the way we think about evolution and it has enabled us to play around with it. We have constructed various forms of evolvable digital objects. We have invented and tested various selection and variation mechanisms, including ones that do not exist in Nature, e.g., crossover mechanisms between more than two parents (Eiben, 2002). And we have designed numerous evolutionary algorithms inspired by natural mechanisms, but not limited by constraints of physical or biological reality. All in all, we have learned a lot about how to set up and to control evolutionary processes and have developed the know-how to use them for solving optimisation, design, and modelling problems (De Jong, 2006; Eiben and Smith, 2008).

To date, the one space where we can design, implement, and execute *all* components of an evolutionary process is inside computers, in digital space.[i] There-

[i]Many biologists consider these processes too simplistic compared with real evolution. We agree, but note that this issue is not relevant for our main argument.

fore, the only type of evolution that we fully master is inherently disembodied. However, in some cases the result of such a digital evolutionary process can be constructed physically. Hence we have two principal kinds of applications. In the first kind, the evolutionary process and the result are both digital. Well known areas in this category are evolutionary optimisation, evolutionary data modelling and evolutionary simulations in artificial life, evolutionary economy, etc. (Ashlock, 2006; Epstein and Axtell, 1996; Langton, 1995). In the second kind, the evolutionary process is digital, but the result of evolution (e.g., the blueprint of a chair or an antenna) is made physical by an extra construction step afterwards. This is known as evolutionary design with evolutionary art as a special sub-area (Bentley and Corne, 2002; Bentley, 1999). Recent advances in rapid prototyping (3D printing), material science, soft robotics, molecular engineering, synthetic biology, combinatorial chemistry, programmable matter, etc. now open the door to create evolvable objects and to implement evolutionary operators in physical space. This enables artificial evolution of the third kind, where the evolutionary process and the result are both physical. The resulting system means a radically new use of *evolution as a tool in a physical medium*. From the historical perspective, this will be the 21st century variant defined by two essential features: It is fully embodied –similar to biological evolution– and artificially engineered –similar to evolutionary computing. Hence the name Embodied Artificial Evolution (EAE).

We argue that EAE forms a high potential research and application area that offers great opportunities and poses great challenges. However, to realise the vision, very diverse and presently segregated fields need to interact and cross-fertilise each other. This necessitates a unifying view, corresponding terminology, and vision to catalyse developments in this direction. This is exactly the main objective of this chapter.

8.2 What is Embodied Artificial Evolution?

The general concept of embodied artificial evolution (EAE) as assumed here is independent from the specific form of embodiment. One can think of cell-like structures in a liquid solvent, a population of robots exploring another planet, or anything else, as long as the given system satisfies the following properties:

1. It involves physical units instead of just a group of virtual individuals in a computer.

2. It has real 'birth' and 'death', where reproduction creates new (physical) objects, and survivor selection effectively eliminates them.

3. Evolution is driven by environmental selection or a combination of environmental fitness and a user defined task-based fitness.

4. In contrast to mainstream evolutionary computing, reproduction and survivor selection are not coupled. They are not executed through a centrally orchestrated main loop, but in a distributed manner, controlled by the individuals who 'decide' themselves when and with whom to mate.

Observe that in terms of evolutionary computing these properties concern representation, variation, selection, and population management. Furthermore, it can be noted that it is properties 1 through 3 that represent the physical embodiment. The fourth feature smoothly fits this set of properties and it is literally more natural than centrally controlled population management. However, in a strictly formal sense, it is not necessary for being embodied.

To aid further elaboration, we consider a number of concrete examples and tasks and use these to illuminate some important properties of EAE systems.

1. **The evolutionary design of a robot controller for a given robot body and some task(s) in a certain environment.** Here, the objects to be evolved are digital, but are inherently part of a (mechatronic) physical entity. To solve this design problem one could port all evolutionary operators to the robot and execute on-the-fly evolution of controllers. Birth and death, i.e., reproduction and survivor selection, is restricted to the digital space of all possible controllers, on the robot's processors. However, fitness evaluation happens *in vivo* here as the reproductive probabilities of any given controller are determined by the real-world performance of the robot driven by that controller.

2. **The evolutionary design of a robot body for some task(s) in a certain environment.**[ii] Here, the objects to be evolved are physical. Thus, one could solve this problem by truly embodied evolution, with physical birth and death. In such a system all evolutionary operators work *in vivo*, including reproduction that creates new robots and survivor selection that effectively eliminates

[ii] For the sake of simplicity, let us disregard the design of the corresponding robot controller.

them. The main challenge here is obviously formed by the reproduction operators crossover and mutation: how to engineer a system where robots can be born (and die)?

3. **The evolutionary design of a bacterium for some medical or chemical task(s) in a certain environment.** Here again, the objects to be evolved are physical. However, while (re)production of mechatronic bodies is a huge challenge, bacteria reproduce by themselves. Thus, that part of the evolutionary machinery is for free in this context. The challenge here is to implement fitness evaluation and the selection operators suited to the given application objectives. Furthermore, one could implement special reproduction operators (mutation and/or crossover) that do not exist in nature, but are useful to solve the given problem.

We can note a couple of things about these examples that help understand some essential aspects of EAE. To begin with, observe that Example 1 is different from Examples 2 and 3 in that it is not truly embodied. To be specific, Examples 2 and 3 illustrate applications where the objects to be evolved are physical. In contrast, the objects to be evolved in Example 1 are digital, only embodied in the sense that they are hosted by a physical robot. Ironically, the term embodied evolution has been introduced for systems like the one in Example 1, cf. Watson et al. (2002). If needed, we can make a distinction by calling this type of systems weakly embodied and using the term strongly embodied for the ones in Examples 2 and 3.

Furthermore, let us note that in case of a robotic application it is possible to separate the body, i.e., the physical robot with its wheels, sensors, etc. and the mind, i.e., the controller regulating the behavior of the robot. Consequently, the task of designing them also can be split in two (and combined, if needed). For the task of designing bacteria, this is not possible, because the regulatory and control mechanisms in bio-chemical organisms are not separated so clearly from the bodies to be regulated.

Yet another difference between a robotic application and a bio-chemical one is the fact that a robotic object is more controllable for the experimenter. Robot bodies are built and robot controllers are programmed by the human experimenters. Even if we consider evolutionary development of robot bodies and controllers, the process is driven by human designed operators. These operators are usually simple; complexity emerges by their interactions. This is not the case for bio-chemical

organisms, where the operators are those invented by nature. These are often very complex to understand and to manipulate. For instance, replacing one mutation operator by another one can be easy in an evolutionary robotics application, but switching off one molecular interaction and switching on another one in a cell can be (nearly) impossible.

8.3 Motivations, Expected Benefits

A straightforward motivation to use a technology is that it is ... useful. Considering breeding livestock or plants as EAE systems (technically: artificial selection and natural reproduction in an embodied setting) we can argue that their usefulness has already been proven. As for the new kind of EAE systems we advocate here, there are multiple reasons to investigate them.

Firstly, EAE can lead to solving new design and engineering problems, and solving existing ones in new ways. In fact, EAE technology can be the basis of a paradigm change in how design tasks are solved. Traditionally, the design process of some artefact ends with manufacturing it. Using embodied artificial evolution, design and manufacturing become an intertwined, continuous, on-line activity, propelled by the evolutionary operators (see Figure 8.1 and the example application in Section 8.5.3). In the long term, the basic design-and-manufacture loop of the production industry may be transformed from the present off-line type with a critical role for the human designer to a more on-line process. In this process new designs arise though evolutionary variations (are 'born'), tested immediately *in vivo*, and reproduce to seed new designs, if successful. While this is clearly not an appropriate workflow for all production industries, there are several potential application areas ranging from fashion items to bio-medical nano-robots.

Secondly, there is much evidence in traditional evolutionary computing that evolution can solve problems not solvable otherwise and that evolution can generate unexpected solutions. (Which, then, can be analysed and reverse-engineered, and thus lead to new insights and better understanding of the problem.) Well-known examples of evolution outperforming human experts or surprising researchers range from Keane and Brown's experiments in satellite boom design (Keane and Brown, 1996) to Koza et al.'s 2000 inventory of human-competitive genetic programming results . Once we equip certain groups of artefacts with the

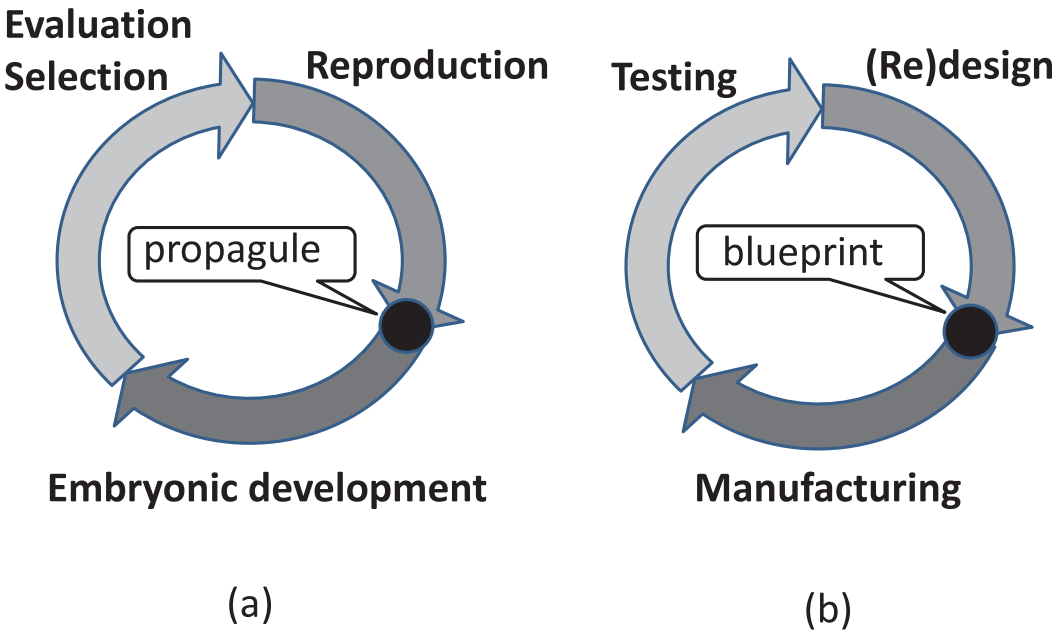


Figure 8.1 – Two circles showing the analogies between the biological circle of reproduction (a) and the new kind of *in vivo* evolutionary design (b). The effective lifetime is captured by the light gray arrow labeled "Evaluation, Selection" and "Testing", respectively.

ability to evolve, we create the possibility that some of the evolved designs may be truly original, stepping out of the box with respect to human thinking.

Thirdly, EAE systems can provide a basis for a new experimentalism in biology, where evolution can be studied in a radically new way in a new medium. To this end it is worth noting that mankind has thousands of years of experience with artificial selection, for instance to breed livestock or plants. As mentioned above, technically speaking this amounts to artificial selection and natural reproduction applied to natural bodies and it has been a valuable tool in using as well as understanding biology and evolution. The new kind of EAE systems we envision extend this in two important ways: artificial evolvable objects (bodies) and artificial reproduction operators (mutation and recombination). The resulting artificial evolutionary systems offer tools to perform real-world evolutionary experiments that are controllable, repeatable, and (relatively) fast, challenging current thinking about the evolutionary process per se. This will enable a deeper understanding of evolution in general, not restricted to or constrained by evolution-as-we-know-it based on our only example, life on Earth. Mastering all components of the system enables us, for instance, to investigate the minimum requirements of evolution, to

estimate how (un)likely evolution is, to distinguish different types of evolution, etc. In the long term, this will lead to new scientific insights regarding evolution and the origins of life.

Finally, EAE systems represent a great challenge from the perspective of algorithm design. The 20th century science/art of designing and analysing (evolutionary) algorithms needs to be reinvented, once we change the medium from purely digital to embodied, physical. The fundamental problem lies in the inevitable physical restrictions concerning the representation, the algorithmic operators, and the limited options a user has in controlling the algorithm as a whole. Simply put, in evolutionary computing experimenters have great freedom in choosing any data type to represent candidate solutions and defining suitable mutation/crossover operators (De Jong, 2006; Eiben and Smith, 2008). However, in an EAE system the bodies to be evolved and the reproduction operators must be physically viable. Further to operator design, we also face the problem of process control. Just to mention one thing, population size management is trivial in a genetic algorithm, but keeping an evolving population of robots or bacteria from extinction as well as from explosion can be a hard nut to crack (Wickramasinghe et al., 2007). Furthermore, EAEs mean a great paradigm shift from evolving digital objects to evolving things in the real world. This implies that the environment where evolution takes place becomes orders of magnitude more complex with inherent randomness (" the noise and the physics are for free") and a dynamics never encountered in traditional evolutionary computation. In fact, we can say that adopting this new technology, digital algorithm design will become physical process design, where the convenient distinction of algorithm components (representation, variation operators, selection operators, population management) may not be applicable at all. All in all, Embodied Artificial Evolution represents a new angle for Evolutionary Algorithms for three main reasons: 1) the design of the evolvable objects (representation) and the evolutionary operators is constrained by physical restrictions, 2) process control is much harder as we are not the superusers or omnipotent system administrators in real life, 3) the dynamics, noise etc. of the real world is much more complex than in digital spaces.

8.4 Relevant Research Areas

We distinguish four possible scenarios for realisation of embodied artificial evolution: *micro-/nano- mechatronic*, *top-down bio-synthetic*, *bottom-up chemo-synthetic* and *hybrid* ones. In this section, we briefly describe the current state-of-the-art research for each of these areas.

8.4.1 Micro- and Nano- Mechatronic Systems, Evolvable Hardware

Mechatronic systems are attributed to different areas of robotics (Kernbach, 2011), (Siciliano and Khatib, 2008). In the context of EAE, the embodiment (Pfeifer and Bongard, 2006) of robotic systems (using specific properties of materials to achieve a desired functionality, e.g. locomotion for small jumping robots (Kovac et al., 2008) or embodied sensor-actor coupling (Kernbach et al., 2009b)) has a decisive role. Modern robotics utilises different fields of material science, e.g. Gong et al. (2005), which vary from modifications of surface properties up to composite materials with specific mechanical features; miniaturisation of micro-systems (Nelson et al., 2008) and structuring of material by micro-/nano- manipulation (Fatikow, 2008; Nelson et al., 2008). To underline these research areas, we denote this scenario as micro- and nano- mechatronics. The relevance to EAE lies in three approaches: using stand-alone robots for exploring situated evolution, creating a programmable mechatronic matter through guided self-assembling and non-biological self-reproduction.

In the literature various references can be found to work related to EAE in a population of stand-alone robots for exploring evolutionary properties of such systems (Floreano et al., 2008). Watson et al. (2002) (also Ficici et al. (1999)) envisioned embodied evolution: a "large number of robots freely interact with each other in a shared environment, attempting to perform some task". In this sense, a population of individuals (in this case, robots) evolves in a completely autonomous manner, i.e. evaluation, reproduction and selection operators are carried out by and between individuals themselves. As in natural evolutionary systems, adaptive mechanisms are asynchronous, decentralised and distributed. Schut et al. (2009) present a related concept called *situated evolution*, where reproduction creates new

minds that become active in a pre-existing robot body, replacing an old one.[iii] Usui and Arita (2003) address embodied evolution as in Watson et al. (2002): robots evolve based on interactions with the environment and other robots. Nakai and Arita (2010) extend this framework by introducing a pre-evaluation mechanism, intended to restrain robot behaviours that are estimated to be have a low fitness contribution. Following then same argumentation, Elfwing et al. (2005) also make use of a subpopulation of virtual agents for each (physical) robot in order to overcome the restriction on population size.

Another state of the art approach applies evolutionary operators not only to the robot controllers but to the robots themselves. In this case the body of the robot has a modular structure and is created through self-assembling process guided by evolution. Multiple research projects, such as HYDRA (Jorgensen et al., 2004), Molecubes (Zykov et al., 2007), Polypod (Yim et al., 2003), M-TRAN (Kamimura et al., 2005), SuperBot (Shen et al., 2006), SYMBRION (Levi and Kernbach, 2010) develop heterogeneous reconfigurable platforms. A number of publications are devoted to application of evolutionary approaches (Stradner et al., 2009) or guided self-assembling (Kernbach et al., 2011) to create a body of modular robots. Not just the evolution of robot's body, but also the co-evolution of body and mind is an important aspect of such research (Pollack et al., 2001). The general technological trend here is to switch from current mini-scale modules to micro- and potentially to nano-scale elements (Scholz et al., 2011). In the context of body-mind evolution, the concept of evolvable hardware (Gordon and Bentley, 2002) needs to be mentioned. Flexibility and a developmental plasticity of such devices allow deriving an advanced computational functionality in hardware (Haddow and Tyrrell, 2011), which is used in robotics, image processing and other technological areas. Several open issues in the development of evolvable hardware are in discussion, e.g. Djupdal and Haddow (2007).

Finally, self-reproduction of micro- and nano-mechatronic systems is of interest for EAE. One of the oldest ideas is Von Neumann's kinematic self-reproduction (von Neumann, 1966). There are multiple attempts to create macroscopic self-reproduction, e.g. by NASA (Freitas and Gilbreath, 1982) or in the context of modular robotics (Zykov et al., 2007). They argue that mechanical self-reproduction is possible and not unique to biology. Recent works attribute capabilities

[iii] Although it may be an oversimplification to view a human body (including the brain) as hardware and the mind as software, we find this distinction helpful when considering the parallel development of bodies and their controllers.

of self-reproduction to nano-technological systems (Lee and Chirikjian, 2007), to additive plastic moulding (Sells et al., 2009) (see also RepRap.org), or to advanced 3D prototyping technology (Rieffel and Sayles, 2010). However, none of these technologies is capable of reproducing complex functional elements, see Section 8.6.2.

8.4.2 Top-down Bio-Synthetic Systems

Biological systems have an advantage over mechatronic devices because biological properties, such as reproduction, can be taken for granted: a biological system is naturally equipped to carry out evolutionary processes. Reproduction, self-preservation, but also selection and adaptation are inherent capabilities of the system. However, an important challenge is how one can manipulate the system to obtain the behaviour one is looking for. Programming cells does not aim to substitute silicon computing, but seeks access to the numerous functionalities and properties on those cells in a predictable, reliable way.

Advances in the area of synthetic biology have allowed some interesting recent results. For instance, Tamsir et al. (2011) show how logic gates can be built in *Escherichia coli* cells and how complex computations can be produced by "rewiring" communication between cells. Works in this area are related e.g. to a development of bacterial systems (Martel et al., 2009), genome engineering (Carr and Church, 2009), or molecular synthesis of polymers (Pasparakis et al., 2010). Intensive research is also devoted to biologically engineering multi-cellular systems (Basu et al., 2005); see more about general fields and challenges of synthetic biology in Alterovitz et al. (2009).

In biological computing, natural processes can be often described in terms of networks of simple computational components, or *biobricks* (Amos, 2009). The main objective is to use the power of natural processes for the purpose of computation. Because natural processes are intrinsically random, changing functionalities of a cell, as well as adding new desired behaviours is not a trivial exercise. Using an alternative approach, Regot et al. (2011) describe how to implement complex Boolean logic computations, which reduces wiring constraints. This is obtained through a redundant distribution of the desired output among the engineered cells. Following the idea of biobricks, a number of cells can be combined into more complex circuits.

8.4.3 Bottom-up Chemo-Synthetic Systems

The bio-synthetic systems utilise existing biological cellular systems with their very complex metabolism. The approach from bottom-up chemistry uses another methodology: creating elementary basic cellular (so-called vesicles) and multi-cellular structures "from scratch". Advantages of this approach are multiple degrees of freedom in designing metabolic networks (in simple cases – autocatalytic reactions) and different internal and external interaction mechanisms (McCaskill et al., 2007).

Examples of bottom-up chemical systems can be found in artificial chemistries (Dittrich et al., 2001), self-replicating systems (Hutton, 2009), using bio-chemical mechanisms for, for example, cognition (Dale and Husbands, 2010). This approach is also denoted as swarm chemistry (Sayama, 2009). Researchers hope that such systems will answer questions related to developmental models (Astor and Adami, 2000), chemical computation (Berry and Boudol, 1992), self-assembly, self-replication, and simple chemistry-based ecologies (Breyer et al., 1997) or that they will yield technological capabilities for creating large-scale functional patterns (Yin et al., 2008). Several approaches consider meso- and nano-objects, such as particles with functionalised surfaces (Schmid, 2004), colloidal systems (Fujita and Yamaguchi, 2009), or molecular networks (Nitschke, 2009); a system of elementary autonomous agents, which possess rudimentary capabilities of sensing and actuation. Information processing and collective actuation are performed collectively as, for example, stochastic behavioural rules. Several phenomena, such as meso-scale self-assembling or diverse self-organising processes (Davies et al., 2009), make these type of systems attractive in applications. L. Cronin *et al.*'s work with polyoxometalate clusters provides an example of chemical synthesis of advanced functional materials on both the molecular level and the nano/microscale (Cooper et al., 2011; Cronin, 2011).

For the design of EAE in molecular, colloidal and particle systems, large-scale interaction patterns for whole systems (Kumar, 2006) can be used. Projects such as ECCell (Chemnitz et al., 2008), BACTOCOM (ni Moreno and Amos, 2010), MATCHIT (Maurer et al., 2011) or "Behavior-Based Molecular Robotics" (Lund et al., 2010) are addressing the questions of programmable chemo-ICT interfaces. Essential attention is paid to a self-replication of chemo-synthetic systems (Fellermann and Rasmussen, 2011; Fernando et al., 2007). Research in collective nanoro-

botics is also focused on the technological capabilities of creating such large-scale patterns in molecular systems, e.g., Yin et al. (2008).

8.4.4 Hybrid Mechatronic and Biochemical Systems

Hybrid mechatronic and biochemical systems combine advantages of both types of technologies and are of essential interest for EAE. There are several reasons for this: sufficient computational properties, high developmental plasticity, utilisation of natural self-reproduction processes. Examples of hybrid systems are bacterial cellular sensors (Wood, 1999), development of bio-hybrid materials (Ruiz-Hitzky et al., 2010), molecular synthesis of biofuels (Alper and Stephanopoulos, 2009). Another example of hybrid technologies are attempts to interact with biological populations by means of technological artifacts: managing the grazing of cattle over large areas (Schwager et al., 2008), (Correll et al., 2008), controlling mixed societies of robot and insects (Caprari et al., 2005), or a social communication between robots and chickens (Gribovskiy and Mondada, 2009). A similar approach is related to the integration of different robot technologies into human societies, for example the management of urban hygiene based on a network of autonomous and cooperating robots (Mazzolai et al., 2008).

One of the interesting approaches in the area of hybrid technologies is a combination of cultured (living) neurons and robots (Novellino et al., 2007) to investigate the dynamical and adaptive properties of neural systems (Reger et al., 2000). This work is also related to understanding of how information is encoded (Cozzi et al., 2006) and processed within a living neural network (DeMarse et al., 2001). The hybrid technology can be used for neuro-robotic interfaces, different applications of *in vitro* neural networks (Miranda et al., 2009) or for bidirectional interaction between the brain and the external environment in the EAE system. Several research projects, e.g. NeuroBit, already addressed the problem of controlling autonomous robots by living neurons (Martinoia et al., 2004).

8.5 Applications

The proof of the pudding is in the eating: new technology is largely justified by useful applications. In the present embryonic stage of the EAE field, it is impossible to predict what the best applications will be. To this end, we see an analogy

with the first decade(s) of the computer industry in the 1950s. This was when when an IBM executive foresaw a world market for perhaps 5 computers all together. Half a century later, there are more computing devices than human beings and countless applications that one could not imagine in the early years of the technology.

As for EAE, we are at the down of the technology, and we dare not predict specific applications. Hence, in the rest of this section we just briefly discuss some potential application areas. In general, EAE systems are suitable for the design and production of artefacts under complex circumstances, for instance in case of (i) changing environments, (ii) unforeseen environments, (iii) ill-defined (implicit) objectives, or (iv) multiple objectives with complex interactions (possibly conflicting). Furthermore, we can distinguish between artefacts that are passive, e.g., jewellery, and those that are active, e.g., micro-robots. These two types differ substantially in that active objects need an inner controller to govern their behaviour, while passive ones do not. With a biological analogy we may say that passive object need a only a body, while active ones need a body and a mind.

8.5.1 Evolving Robots

One could imagine whole ecosystems of robots on different scales of size. On a very small scale we could have medical nano-robots to be deployed in a human body. For example, they could be used as "personal virus scanners", evolving to the metabolism of the host and adapting to fight any new threat be it a germ of cancer. On a larger scale, evolving robot populations for planetary exploration could be interesting. These could be sent to other (unknown) planets with just a rough initial design but with the ability to evolve to the given circumstances. This will enable them to perform exploration and maybe even build a base station from the locally available resources. Still on the large scale, we can conceive evolving robot companions in domestic and industrial environments. Regarding their bodies, these could range from cat and dog size up to human comparable sizes. As for their mental features, they should be human-friendly and intelligent. From a functional point of view they should perform specific tasks and in a domestic setting they could provide more generic 'emotional' services (keeping company, being good listeners, acting as partners in simple conversations) (Wilks, 2010).

8.5.2 Functional Organisms

April 2010 saw the largest oil spill in US history: the equivalent of around 4 million barrels of oil flowed into the Gulf of Mexico, with numerous ecological implications. Analysis on the site, a couple of weeks after the disaster, showed that many groups of bacteria were helping to clean up the waters. These bacteria were able to break down the chemicals found in crude oil and, in fact, responded quite effectively to the incident. In general, there are many possible applications of bacteria, or some other type of organism, that are synthetically designed for a specific functionality. Such artificially developed organisms can be used, for instance, to provide environmental services, create building material or biofuel, to store data, or to stop desertification. An evolutionary approach is literally natural in this application area. At this moment, this line of research – positioned in synthetic biology– is perhaps the closest to a breakthrough, cf. Section 8.4.2.

8.5.3 Evolutionary Personal Fabrication

Imagine a world in which anything can be produced with just a few clicks. Customised products are at the reach of your hand, ranging from a child's toy to a meal. (Vilbrandt et al., 2008) introduce the idea of *universal desktop fabrication* (UDF) that can produce essentially any complete, finished, and functional object. Fab@Home (`www.fabathome.org`) is a desktop rapid prototyper (3D-printer) and a first step towards UDF. Such personal fabricators can build a great variety of objects from different materials and thus enable a large group of people to produce stuff to fit their needs locally. The range of applications is not restricted to solid objects, such as personalised fashion items (jewels, sunglasses, smartphone cases), but may also include consumables, like food: "You can imagine a 3-D printer making homemade apple pie without the need for farming the apples, fertilising, transporting, refrigerating, packaging, fabricating, cooking, serving and the need for all of the materials in these processes like cars, trucks, pans, coolers, etc,"[iv]. Embodied evolutionary technology is expected to play an important role in the development of such fabricators, cf. Rieffel (2006) and Rieffel and Sayles (2010) : "Ultimately, the evolution of form and formation become fully intertwined when the language of assembly itself becomes subject to evolution [...]. Through this

[iv] Homaro Cantu states in the BBC News article "The printed future of Christmas dinner": `http://www.bbc.co.uk/news/technology-12069495`

co-evolution of form and formation, Evolutionary Fabrication discovers both how to build objects and what to build them out of."

In general, evolutionary technology can be used on local and global level. Locally, a limited set of users (one person, a family, or a small firm) would represent the fitness function governing evolution. The system could adapt to their preferences advancing customisation. On a global scale, such personal fabricators could be networked to yield an evolutionary system involving billions of users, evolutionary app stores, and almost incomprehensible dynamics.

8.6 Grand Challenges

At this moment it is impossible to foresee how this field will develop. However, we are able to identify some of the grand challenges that certainly will have to be addressed.

8.6.1 Body Types

The essence of embodied evolution is the body. To this end, we can distinguish hardware in the broad sense (mechatronic-robotic systems, new materials, etc) and wetware (bio-chemical systems) that may also be hybridised. Regarding wetware, there are two options again: bottom-up, relying on chemistry, or top-down, based on biology. Recent developments in microfluidics, functional fluids, or programmable matter also seem very promising. The first grand challenge is thus to find body types suited for (self-)reproduction. In essence, this means that we need to inject dead matter with a human requested functionality. This question is also known in other formulations, e.g., "programmability of synthetic systems", or "open-ended embodied evolution", and is one of the key points in understanding principles of synthetic life. It is also addressed by the European bio-ICT initiative and several research projects, e.g. PACE (PACE, 2008) and e-FLUX (e Flux, 2011), to name but a few.

Summarising, one of the principal challenges of EAE is to find physical constructs that are suited to be the evolvable objects forming the population. Technically this requires that they can be produced and reproduced. This is akin to one of the main problems in Evolutionary Computing: how to find a suitable repre-

sentation, that is, a data structure that can be used for the individuals representing candidate solutions (Rothlauf, 2006).

8.6.2 How to Start – Reproduction of Functional Elements

The implementation of birth (reproduction operators) for human engineered physical artifacts is a critical prerequisite for EAE. These operators must also realise some form of inheritance. The approaches based on mechatronics, chemistry, or biology differ greatly in this respect. (Self-)reproducing mechatronic and chemical units are far from being trivial, whereas it comes for free in biological systems.

As mentioned in Section 8.4.1, in current micro- and nano-mechatronic systems there are two concepts that are crucial for EAE: self-assembling and self-replication. Self-assembling is a process which creates complex systems from basic elements, whereas self-replication means a reproduction of these basic elements. Robots are able to make functional copies of artificial organisms from basic building blocks provided there exists an essential reserve of such basic modules. However, the self-replication of basic modules obstructed by functional elements, such as motors, gears or generally silicon-based microelectronics. Due to their high technological complexity, the self-replication of these functional elements remain so far unsolved.

8.6.3 How to Stop – Kill Switch

A serious concern regarding EAE is the possibility of runaway evolution. By this term we do not mean the Fisherian notion of sexual selection reinforcing useless traits (Fisher, 1930). Runaway evolution as we use it here stands for the process of uncontrolled population growth. Such a growth might also be accompanied by the emergence of new, unwanted features in the population. Obviously, it would be highly irresponsible to expose ourselves to such a risk. To reduce this risk, all such experiments could be carried out in highly secured isolated environments, not unlike current research into certain germs, bacteria, viruses, etc. involving bio-hazard. However, this might disable the whole application in cases where the evolving population is inherently free, acting 'out in the wild' (robot companions, waste-eating organisms, medical nano-robots in the human body, etc.). In such cases a 'kill switch' is required to guarantee that human supervisors are able to shut down the system, if and when they deem necessary.

As of today, the kill switch problem has been already recognised within synthetic biology. There are various approaches to obtain a solution, such as for instance suicide genes, programmed cell death (PCD) and apoptosis (Callura et al., 2010; Carmona-Gutierrez et al., 2010; Engelberg-Kulka et al., 2006; Khalil and Collins, 2010), just to name a few. A particular challenge stemming from the inherent use of evolution is possibility that an evolutionary systems will find solutions that are well 'outside the box' for the human designers of these systems (cf. the originality argument in Section 8.3). It is therefore essential that great care be taken when designing kill switches to ensure that evolution will not be able to circumvent them. In common parlance, we need to prevent the 'Jurassic Park problem'.

8.6.4 Evolvability and Rate of Evolution

It is well-known in biology as well as in evolutionary computing that evolution is a relatively slow form of adaptation. To put it simply, it can take many generations to achieve a decent level of development. Obviously, 'slow', 'many', and 'decent' depend on the application context. For instance, medical nano-robots put to work in a human body should adapt within a few hours to their environment (the patient's body). In case of sending evolving robot explorers with a rough initial design to Mars, one can wait months for appropriate designs to emerge. In general, we can say that useful EAE systems must exhibit a high degree of evolvability and a high rate of evolution (Hu and Banzhaf, 2010). In practice, they must make good progress in real time: have short reproduction cycles and/or large improvements per generation. The main factors here are the application dependent time requirements and quality criteria that define how progress is measured, and the speed of progress determined by the evolutionary operators.

Building fast evolutionary systems is a nontrivial challenge on its own. Failing to meet this challenge would imply that the real time performance of EAE systems is too low. Ultimately, this could even disqualify the whole approach – at least, for certain applications. In general, the speed of evolution should be used as one of the essential assessment criteria for judging the feasibility of potential applications.

8.6.5 Process Control and Methodology

A radical change caused by EAE technology is that design and manufacturing become an intertwined, continuous activity. This poses an unprecedented challenge

for maintaining human control during the process. In Evolutionary Computing, on-line control of an evolutionary algorithm is exercised through changing its parameter values on-the-fly (Eiben et al., 1999b). Such control is directed to improving the working of the given algorithm, e.g., increasing its speed or recovering from local optima. In the EAE systems we envision, there is additional challenge: we need to combine open-ended and directed evolution on-the-fly. This means that human users should be able to perform on-line monitoring and steering in line with the given user preferences. This could be perhaps realised by directed selection (akin to breeding) and/or directed reproduction (as in genetic manipulation). On a conceptual level, this requires a new kind of methodology that must contain the traditional elements, such as specification, validation, and tuning (Eiben and Smit, 2011). Meanwhile, we have to address the novel aspects, such as the combination of free evolution and specific design objectives. Part of this challenge is the 'freeze switch', that is, the ability to recognise if/when the evolving objects have obtained the required properties and stop further evolution without killing the system.

8.6.6 Body-mind Coevolution and Lifetime Learning

As explained in the introduction of Section 8.5, in general we can distinguish passive and active artefacts. Obviously, an active artefact needs an entity governing its activities. In some life forms, e.g., bacteria, the control and regulatory mechanisms form a unity with the body. In higher life forms, such control mechanisms are augmented with a designated control entity, the mind (the 'software'), carried by a separate part of the body, the brain (the 'hardware').[v] Similarly, in EAE systems active artefacts can have a dual structure with a body and a mind (controller) that must fit the given body. This implies that bodies and minds have to coevolve, they will be subject to reproduction and inheritance. Obviously, we do not know how the reproduction and inheritance mechanisms for bodies will be related to those concerning the minds in any specific EAE system. However, in general it cannot be assumed that the inherited mind will perfectly match the inherited body. Therefore, the system must include the possibility that a newborn object undergoes a lifetime learning process – not unlike baby animals have to learn walking, seeing, etc. soon after birth. Depending on the given EAE system at hand, it may be

[v] This dualist view is in fact an oversimplification that could be debated. Nevertheless, we find it a helpful distinction that reflects the current robotics and computer technology.

possible to make individually learned skills inheritable, i.e. to make the system Lamarckian. The 'Artificial' in EAE offers a possibly large degree of technical freedom, and experimenters of such systems could make their systems Lamarckian, even though biological evolution is not.

8.7 Final Remarks

We have presented the concept of Embodied Artificial Evolution or the Evolution of Things. The systems we envision are embodied because evolutionary operators (reproduction, selection, fitness evaluation) are implemented in/by the physical objects that undergo evolution. Furthermore, they are artificial because (i) the evolvable objects and the population as a whole are being fabricated and/or programmed to fulfil a certain human purpose, to execute a certain task,[vi] and (ii) the evolutionary operators (reproduction and selection) and their particular combination into one working system are human engineered.

We believe that Embodied Artificial Evolution offers a high potential research and application area with exciting scientific and technological challenges. This field is in an embryonic stage, where relevant developments take place within different scientific communities and technological areas that do not naturally interact with each other. At the moment we see three main streams of research towards building EAE systems: top-down, biology-based, bottom-up working from chemistry and 'head-on' engineering based on robotics and material science. Furthermore, Evolutionary Computing can play an important role as the field that collected a large body of knowledge about designing, implementing, and executing *all* components of an evolutionary process. We hope that by introducing a unifying vision we can bring all stakeholders together raising awareness of the shared research issues and possible solutions.

Last, but not least, let us mention a particular issue all approaches must address: the related ethical questions. In this respect, several problems have already been noticed in Life Sciences (Cho and Relman, 2010). However, EAE systems based on non living mediums could lead to very similar challenges, be it in different forms. For instance, bio-hazard can turn into robo-hazard. The ethical questions therefore form a clearly horizontal issue, cross-cutting over different disciplines and technical approaches to EAE. One of the main goals of this chapter

[vi] This does not exclude open-ended evolution to take place in parallel.

is to create an overarching vision, which in turn could contribute to help research communities and institutions develop a solid system of checks and balances thus making such research a safe enterprise.

We have not succeeded in answering all our problems – indeed we sometimes feel we have not completely answered any of them. The answers we have found have only served to raise a whole set of new questions. In some ways we feel that we are as confused as ever, but we think we are confused on a higher level and about more important things.

Earl C. Kelley

9 Discussion

In the introduction of this thesis, we stated that the goal of our research would be to develop novel evolutionary algorithms that meet the challenges of on-line, on-board evolutionary robotics, and we posed three research questions related to this goal.

We introduced a classification of on-line evolutionary robotics into encapsulated, distributed and hybrid variants. The encapsulated case simply runs a self-sufficient evolutionary algorithm in each robot, the distributed approach relies on the exchange of genetic information between multiple robots and the hybrid variant combines these two approaches. We have successfully developed and experimentally validated algorithms for on-line evolutionary robotics in each of these classes, in particular the encapsulated $(\mu + 1)$ ON-LINE algorithm and distributed and hybrid variants of EVAG-based algorithms.

We have seen that these algorithms are capable of developing controllers for a number of tasks in an on-line fashion, and in doing so allow robots to adapt continuously while performing their regular tasks. This answers the first of the three research questions: on-line evolution *is* capable of providing the ability to learn as required and provide consistent task performance. At least, it can in

the experiments that we have performed. To what extent we may generalise this conclusion to other, more complex tasks and other controller architectures than the ones we considered is a matter for further investigation, but the signs are promising.

The second question we asked was how we can tackle the particular demands that on-line evolution poses.

In section 5.1, we introduced the notion of re-evaluating individuals to tackle the issue of 'unfair' comparisons between candidate controllers. Subsequent research showed that re-evaluating individuals may not be needed, maybe because of redundant copies of good individuals when the population size is larger than 1. In fact, the results of chapter 6 indicate that on-line evolution should focus on trying many individuals rather than on the accuracy of the assessments: the rate of re-evaluation was best set to a low value, and the individuals should be tested over short timespans.

This is an interesting result: in more general terms, it indicates that evolutionary algorithms produce better results when sampling many solutions approximately than when they sample fewer solutions with more exactitude. Particularly in applications where the wall-clock time in which a solution is reached matters, using fast fitness approximations instead of slow exact assessments could improve the quality. Examples of such applications can be found in real-time systems for data mining or constraint satisfaction. More generically, this would make the case for surrogate models as fast approximators of a solution's performance. It is a matter of further research to what extent this finding applies to such cases.

We have seen that on-line evolution can work well with small, in the distributed and hybrid case even very small, populations.

The results in section 5.2.5.2 showed that good overall or actual performance requires different parameter settings than does optimising only the performance of the best individual in the population. In general, we have evaluated the algorithms in terms of actual performance without considering the best individuals separately.

We have seen that we can do away with the need for an external overseer of controller evolution by either encapsulating the whole evolutionary algorithm within each robot or by distributing it across the robots, with the possibility of hybridising these variants. In chapter 7, we saw that the distributed and hybrid variants scale up with the number of robots: the more robots, the better the performance. This is an obvious effect of the fact that the distributed and hybrid models concurrently

evaluate individuals in all robots; more robots simply means a larger population. Distributed on-line evolution scales down to a surprising extent: even with as few as 4 robots, it produces good results. Obviously, a pure encapsulated approach cannot benefit from multiple robots: there is no scaling effect. On the other hand, it also works if there is only a single robot. The hybrid approach combines the best of both worlds as it maintains a population in each robot, allowing autonomous evolution even when there is only a single robot. The exchange of individuals between robots allows hybrid algorithms to profit from the increased potential for evaluation that multiple robots offer. The results in section 7.1 indicate, however, that encapsulated evolution may be preferable to distributed or hybrid evolution in cases where the task introduces 'subtle forms of competitive co-evolution': in such cases, the less diverse behaviour resulting from a shared pool of genomes may be detrimental. Further research into this issue is needed.

As expected, we saw substantial differences in performance of our algorithms depending on the parameter settings. This is the reason that we turned to the automated parameter tuning toolbox REVAC and its successor Bonesa to ensure fair comparisons between algorithms after equal tuning effort. What we also noted, particularly in chapter 6 is that there is no 'silver bullet' set of parameter values that allow an algorithm like $(\mu + 1)$ ON-LINE to operate near optimally in any task, so we will need to employ on-line parameter control schemes to adapt the algorithm parameters to the circumstances. In section 5.2, we found the proposed distributed (EDEA) algorithm much less sensitive to parameter settings than its encapsulated counterpart ($(\mu + 1)$ ON-LINE). The results in section 7.2 indicate that this sensitivity depends on the number of robots over which evolution is distributed. Chapter 6 provides a methodological contribution when it comes to investigating parameter sensitivity: by analysing the results of Bonesa runs, we learn which parameters influence the performance of an algorithm and how the parameter values interact.

The last question we put to ourselves in the introduction was which parameters of the algorithms we investigate have the most influence on the quality of robot behaviour. We already noted that there is no single setting of parameter values that works well in all cases. This means that we have to use and possibly develop control schemes for the parameters that we find to have the most profound influence on algorithm performance. Chapter 6 already considers three control schemes for the important mutation step-size parameter σ in $(\mu + 1)$ ON-LINE.

A parameter that is particular to (on-line) evolutionary robotics is the evaluation length τ: it determines how long an individual is given control of the robot to evaluate that individual. This is a very important parameter, having the largest impact on performance in the experiments in chapter 6. Section 5.3 moves towards a control scheme through the racing technique that cuts short unpromising evaluations. These are only first steps, though, and further research is actually underway to develop a robust control scheme.

The exchange of genomes in distributed and hybrid evolution raises the issues of selecting genomes to exchange and selecting partners to exchange them with. We investigated this matter in section 7.3, finding evidence of a slight benefit of a selective admission policy (i.e., only accepting incoming genomes that after re-evaluation prove better than the worst in the current population). We also saw that submitting the most different candidate solution for migration (Araujo and Merelo's MultiKulti algorithm 2011) was a good policy: where it didn't yield the best performance, it was not significantly worse than the alternative. The results of these experiments confirm our earlier findings that, at least for the task of learning to move fast while avoiding obstacles, distributed and hybrid algorithms outperform the encapsulated $(\mu + 1)$ ON-LINE algorithm, even with a small number of robots.

Comparing the results of sections 7.2 and 7.3, we note a striking difference in the performance trend when increasing the number of robots. In 7.2, we clearly see the performance of the EVAG-based algorithms increase with the number of robots, but in section 7.3, performance drops dramatically from 16 to 36 robots! After detailed analysis it turned out that only for these runs, the evaluation period τ was set to very large values as a result of the tuning process in section 7.3's experiments, while the other experiments ran with a very low τ value. this resulted in far fewer evaluations in the experiments with 36 robots, which may account for the difference. It remains to be seen why this difference occurred, but it does highlight a potential pitfall of using (automated) tuning to compare instances of an algorithm: if tuning results in sub-optimal values (because the tuner got it wrong, or because the circumstances of tuning instances varied in some way), the results can be just as misleading as results achieved without tuning. It shows that merely comparing performance, even after tuning (chapter 6's *horse-racing papers*) tells only half the story. To get a complete picture, we must also analyse the

parameter values that were found so that we may understand, not merely notice, the difference in outcome.

Section 7.4 describes an experiment where the robots have no specific task: on-line evolution is used to learn to amass energy which is needed to survive. We can see this section as a precursor to an open-ended approach as discussed below. Our experiments showed that in such cases, the environment can cause robots form organisms without the robots being specifically rewarded for that. We saw that the indirect reward, the benefit of being in an organism, doesn't need to be large: robots will consistently form organisms even if the reward scaled logarithmically with the size of the organisms. This research indicates a shift in focus in our research towards organism-centric and more open-ended evolutionary robotics – the next phase of the SYMBRION effort.

9.1 The Scheme of Things

Chapter 8 presents a view of where artificial evolution may go in the coming decade. We distinguish two substrates to achieve embodied artificial evolution in robots: one is based on 3D printing, the other on self-reconfiguring, modular robots. To see how on-line evolution of controllers as we investigated here suits the modular robotics approach, consider the following scenario:

> *Imagine a collective of robotic organisms in a pit where they perform a mining task. Each organism consists of several robot modules, and each has its own distinctive shape. These organisms move; some aptly, some are only just starting to learn how to use their limbs. The shapes and controllers of the organisms adapt to suit the environment and tasks: some organisms have shapes and controller so that they can easily pry chunks of ore from the walls, others are shaped to carry the ore to carts for further transport. Under the hood, the organisms carry genomes that encode their shape. To reproduce, organisms need to find an 'egg': an individual stationary robot module, perhaps a remnant of an old organism, that they can fertilise with their genome. An egg that has been fertilised by a number of organisms can select from the received genomes for recombination and/or mutation. The resulting new genotype is the basis for a new organism that will be born through a process of morphogenesis that the egg initiates. The egg recruits available modules in a complicated dance,*

where more and more of them dock together to form the required shape; when this dance stops after a while, a new organism is born. This new organisms body is modified from that of its parent(s), so it has to learn how to articulate this body through lifetime learning, just as a newborn calf has to learn to stand up and walk. Thus, the newborn organism starts moving in a clumsy way. It takes a while until its 'mind' learns to control its 'body', but after some time its movements become smooth, just in time to walk to the nearest charging station and replenish its batteries. By now, the 'baby' has grown up and is now of autonomous navigation and feeding in a manner suited to the environment and its own body shape. It can live its life rambling around and trying to propagate its genes. All of a sudden our organism changes its free-wheeling attitude. A human user defined a task that is now imposed on this organism (activated by a time stamp or communicated by an external message). Performing the task well provides special rewards that the organism is keen to collect. Again, its learning abilities come in handy, and over time it is getting better and better, collecting more and more rewards. This does not go unnoticed with the eggs that have a basic instinct to prefer the 'rich'. Meanwhile, an existing organism – maybe one of the parents of this new organism – has come to the end of its life: now it is time to disassemble. It lies down and its individual modules detach; some of them drive away in search of a forming organism that they will become part of, others drive away a small distance, spreading themselves out to become eggs to be fertilised.

Considering a scenario as sketched above in terms of the Population-based Adaptive Systems framework presented in chapter 3, we see that evolution takes place at the level of organisms: the procreate by spreading their genome to the eggs. The algorithms we presented in this thesis, e.g. $(\mu + 1)$ ON-LINE, provide the capability of lifetime learning, although it may be confusing that they employ evolutionary techniques to achieve it. In this light, encapsulated evolutionary algorithms implement individual learning; their impact is limited to a single module per instance. Distributed and hybrid evolutionary algorithms can be seen as providing social learning, but there is an issue in this particular scenario: the exchange of information takes place not between organisms (the evolving entities), but between robot modules that may or may not be part of different organisms. To what extent this straddling of aggregation levels invalidates the findings in chapter 3 remains to be seen.

In chapter 3, we saw that lifetime learning, in particular individual learning, may hinder evolution when the decision to mate is in the remit of the adapting controller. Casting on-line evolution as providing lifetime learning means that this should be taken into account and that some form of redress, for instance a specific reward for mating (the word 'orgasm' comes to mind) may be required.

The work in this thesis was largely undertaken as part of the SYMBRION effort; on-line evolution as described here has been adopted by the project partners to provide the robots with continuous and autonomous adaptation, and it has been validated in hardware, although at a limited scale because reliable hardware is only now becoming available. The evolutionary computing effort in SYMBRION is culminating in a scenario similar to that described above, with on-line evolution implementing lifetime learning.

9.2 Current and Future Research

As mentioned earlier, it is difficult to ascertain to what extent our conclusions can be generalised to tasks, environments and control architectures that we did not consider. Therefore, an obvious track for further research is to extend the experiments described in this paper so that we can determine which findings do and which ones do not hold in general. In part, this is current work in the SYMBRION project: there are now, for instance, trials with $(\mu + 1)$ ON-LINE adapting controllers based on the artificial homeostatic hormone system described by Hamann et al. (2012).

One aspect of the original vision that we haven't, to date, paid much attention to is that of on-line evolution in dynamic environments. We are now running experiments that do introduce dynamism, in particular to investigate the possibilities of on-line parameter control.

On the topic of parameter control, it is worth mentioning a recent paper by Bim et al. (2012) in which we introduce *fate* agents that perform the various selection and reproductive tasks in situated evolution. Because the fate agents themselves evolve as well, this amounts to a scheme with adaptive operators. It also provides a possible avenue of solving the 'kill switch' issue from section 8.6.3 without sacrificing scalability: should evolution run out of control and need to be curtailed, eliminating the fate agents will bring evolution to a halt. Because there would be

much fewer fate agents than regular agents (robots, in our case), eliminating them may be feasible.

A direction of research that gains increasing popularity in evolutionary robotics and that we already briefly mentioned considers objective-free or open-ended evolution: it considers no task beyond survival. Open-ended evolution, for instance described by Bianco and Nolfi (2004) and Bredeche et al. (2012) is inherently online, because the robots have no real task other than surviving to spread their genetic material – just as in real life. Most of the research described here, with the notable exception of section 7.4, focusses on more traditional task-driven evolution: in most experiments, the robots must learn to perform a well-defined task and an explicit fitness function determines rewards based on how well a robot performs the task at hand. In fact, the work described here can be seen as a precursor to the open-ended evolutionary scenario described above, and our current research investigates how to combine open-ended and task-driven adaptation robustly, for instance using the reward exchange rate sketched in the scenario.

References

Acerbi, A., Marocco, D., and Vogt, P. (2008). Social learning in embodied agents. *Connection Science*, 20(2):69–72.

Acerbi, A. and Nolfi, S. (2007). Social learning and cultural evolution in embodied and situated agents. In *Proceedings of the First IEEE Symposium on Artificial Life*, Piscataway, NJ. IEEE Press.

Alba, E. and Dorronsoro, B. (2008). *Cellular Genetic Algorithms*. Springer.

Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443 –462.

Alper, H. and Stephanopoulos, G. (2009). Engineering for biofuels: exploiting innate microbial capacity or importing biosynthetic potential? *Nature Reviews Microbiology*, 7(10):715–723.

Alterovitz, G., Muso, T., and Ramoni, M. F. (2009). The challenges of informatics in synthetic biology: from biomolecular networks to artificial organisms. *Briefings in bioinformatics*, 11(1):80–95.

Amos, M. (2009). Bacterial computing. In Meyers, R. A., editor, *Encyclopedia of Complexity and Systems Science*, pages 417–426. Springer New York.

Araujo, L. and Merelo, J. (2011). Diversity through multiculturality: Assessing migrant choice policies in an island model. *IEEE Transactions on Evolutionary Computation*, 15(4):456 – 469.

Ashlock, D. (2006). *Evolutionary Computation for Modeling and Optimization*. Springer.

Astor, J. C. and Adami, C. (2000). A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6(3):189–218.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK.

Baldassarre, G., Nolfi, S., and Parisi, D. (2002). Evolving mobile robots able to display collective behaviours. *Artificial Life*, 9:255–267.

Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M., Honavar, V., Jakiela, M., and Smith, R., editors (1999). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*. Morgan Kaufmann, San Francisco.

Bartz-Beielstein, T., Lasarczyk, C., and Preuss, M. (2005). Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 773–780 Vol.1. IEEE.

Basu, S., Gerchman, Y., Collins, C. H., Arnold, F. H., and Weiss, R. (2005). A synthetic multicellular system for programmed pattern formation. *Nature*, 434(7037):1130–1134.

Belew, R., McInerney, J., and Schraudolph, N. (1990). Evolving networks: Using the genetic algorithm with connectionist learning. In et al., C. L., editor, *Proceedings of the Second Conference on Artificial Life*, pages 511–547, Reading, MA. Addison-Wesley.

Belew, R. and Mitchell, M. (1996). *Adaptive Individuals in Evolving Populations: Models and Algorithms*. Addison-Wesley.

Bentley, P., editor (1999). *Evolutionary Design by Computers*. Morgan Kaufmann, San Francisco.

Bentley, P. and Corne, D., editors (2002). *Creative Evolutionary Systems*. Morgan Kaufmann, San Francisco.

Berry, G. and Boudol, G. (1992). The chemical abstract machine. In *Selected papers of the Second Workshop on Concurrency and compositionality*, pages 217–248, Essex, UK. Elsevier Science Publishers Ltd.

Best, M. (1999). How Culture Can Guide Evolution: An Inquiry into Gene/Meme Enhancement and Opposition. *Adaptive Behavior*, 7(3-4):289.

Beyer, H.-G. (2000). Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):239 – 267.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies – A comprehensive introduction. *Natural Computing*, 1:3–52.

Bianco, R. and Nolfi, S. (2004). Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 4:227–248.

Bim, J., Karafotias, G., Smit, S. K., Eiben, A. E., and Haasdijk, E. (2012). It's fate: A self-organising evolutionary algorithm. In Coello, C. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *The 12th International Conference on Parallel Problem Solving from Nature – PPSN XII*. Accepted for publication, to appear.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient Machines Through Continuous Self-Modeling. *Science*, 314(5802):1118–1121.

Bosman, P. A. N., Yu, T., and Ekárt, A., editors (2007). *GECCO '07: Proceedings of the 2007 GECCO conference on Genetic and evolutionary computation*, New York, NY, USA. ACM.

Bowen, J. and Dozier, G. V. (1995). Solving constraint satisfaction problems using a genetic/systematic search hybrid that realizes when to quit. In Eshelman, L., editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 122–129. Morgan Kaufmann, San Francisco.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.

Branke, J., Schmidt, C., and Schmec, H. (2001). Efficient fitness estimation in noisy environments. In *Proceedings of Genetic and Evolutionary Computation*, pages 243–250.

Bredeche, N., Haasdijk, E., and Eiben, A. (2009). On-line, on-board evolution of robot controllers. In Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., and

Lutton, E., editors, *Artificial Evolution*, Lecture Notes in Computer Science, pages 110–121. Springer.

Bredeche, N., Montanier, J.-M., Liu, W., and Winfield, A. F. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129.

Breyer, J., Ackermann, J., and McCaskill, J. (1997). Evolving reaction-diffusion ecosystems with self-assembling structures in thin films. *Artificial Life*, 4(1):25–40.

Brooks, R. A. (1991). Intelligence without reason. In Myopoulos, J. and Reiter, R., editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia. Morgan Kaufmann.

Brooks, R. A. (1992). Artificial life and real robots. In Varela, F. and Bourgine, P., editors, *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life*, pages 3–10, Cambridge, MA, USA. MIT Press.

Bull, L., Studley, M., Bagnall, A., and Whitley, I. (2007). Learning Classifier System Ensembles With Rule-Sharing. *IEEE Transactions on Evolutionary Computation*, 11(4):496–502.

Buresch, T., Eiben, A. E., Nitschke, G., and Schut, M. (2005). Effects of evolutionary and lifetime learning on minds and bodies in an artifical society. In Corne, D., Michalewicz, Z., McKay, B., Eiben, A. E., Fogel, D., Fonseca, C., Greenwood, G., Raidl, G., Tan, K., and Zalzala, A., editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1448–1454. IEEE Press.

Callura, J. M., Dwyer, D. J., Isaacs, F. J., Cantor, C. R., and Collins, J. J. (2010). Tracking, tuning, and terminating microbial physiology using synthetic riboregulators. *PNAS*, 107(36):15898–15903.

Cangelosi, A. and Parisi, D. (1998). The emergence of "language" in an evolving population of neural networks. *Connection Science*, 10:83–93.

Cantú-Paz, E. (2001). Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of heuristics*, 7(4):311–334.

Caprari, G., Colot, A., Siegwart, R., Halloy, J., and Deneubourg, J.-L. (2005). Building mixed societies of animals and robots. *IEEE Robotics & Automation Magazine*, 12(2):58– 65.

Carmona-Gutierrez, D., Eisenberg, T., Büttner, S., Meisinger, C., Kroemer, G., and Madeo, F. (2010). Apoptosis in yeast: triggers, pathways, subroutines. *Cell Death and Differentiation*, 17:763–773.

Carr, P. A. and Church, G. M. (2009). Genome engineering. *Nature biotechnology*, 27(12):1151–1162.

Chemnitz, S., Tangen, U., Wagler, P., Maeke, T., and McCaskill, J. (2008). Electronically programmable membranes for improved biomolecule handling in microcompartments on-chip. *Chemical Engineering Journal*, 135, Supplement 1(0):S276 – S279.

Cho, M. K. and Relman, D. A. (2010). Synthetic "life," ethics, national security, and public discourse. *Science*, 329:38–39.

Christensen, D. J., Spröwitz, A., and Ijspeert, A. J. (2010). Distributed online learning of central pattern generators in modular robots. In Doncieux, S., Girard, B., Guillot, A., Hallam, J., Meyer, J.-A., and Mouret, J.-B., editors, *From Animals to Animats 11*, volume 6226 of *Lecture Notes in Computer Science*, pages 402–412. Springer Berlin / Heidelberg.

Cooper, G. J. T., Boulay, A. G., Kitson, P. J., Ritchie, C., Richmond, C. J., Thiel, J., Gabb, D., Eadie, R., Long, D.-L., and Cronin, L. (2011). Osmotically driven crystal morphogenesis: A general approach to the fabrication of micrometer-scale tubular architectures based on polyoxometalates. *J. Am. Chem. Soc.*, 133:5947–5954.

Correll, N., Schager, M., and Rus, D. (2008). Social control of herd animals by integration of artificially controlled congeners. In *Proc. of the 10th International Conference on Simulation of Adaptive Behavior (SAB). Springer Lecture Notes in Artificial Intelligence LNAI 5040*, pages 437–447, Osaka, Japan.

Cozzi, L., D'Angelo, P., and Sanguineti, V. (2006). Encoding of time-varying stimuli in populations of cultured neurons. *Biol. Cybern.*, 94(5):335–349.

Cronin, L. (2011). Defining new architectural design principles with 'living' inorganic materials. *Archit. Design*, pages 34–43.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. *Swarm Robotics*, pages 10–20.

Curran, D. and O'Riordan, C. (2006). Increasing population diversity through cultural learning. *Adaptive Behavior*, 14(4):315–338.

Dale, K. and Husbands, P. (2010). The evolution of reaction-diffusion controllers for minimally cognitive agents. *Artif. Life*, 16(1):1–19.

Darwin, C. (1871). *The Descent of Man*. John Murray, London.

Dautenhahn, K. and Nehaniv, C. L. (2002). The agent-based perspective on imitation. In Dautenhahn, K. and Nehaniv, C. L., editors, *Imitation in animals and artifacts*, pages 1–40. MIT Press, Cambridge, MA, USA.

Davies, R. P. W., Aggeli, A., Boden, N., McLeish, T. C. B., Nyrkova, I. A., and Semenov, A. N. (2009). *Mechanisms and Principles of 1D Self-Assembly of Peptides into ?-Sheet Tapes*, volume 35 of *Advances in Chemical Engineering*, pages 11–43. Elsevier.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press, Oxford, UK.

De Jong, K. (2006). *Evolutionary Computation: A Unified Approach*. The MIT Press.

De Jong, K. and Sarma, J. (1995). On decentralizing selection algorithms. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 17–23, San Francisco, CA. Morgan Kaufmann.

DeMarse, T. B., Wagenaar, D. A., Blau, A. W., and Potter, S. M. (2001). The neurally controlled animat: Biological brains acting with simulated bodies. *Auton. Robots*, 11(3):305–310.

Denaro, D. and Parisi, D. (1996). Cultural evolution in a population of neural networks. In *Proceedings of the 8th Italian Workshop on Neural Nets*, pages 100–111. Springer.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistries—a review. *Artif. Life*, 7(3):225–275.

Djupdal, A. and Haddow, P. C. (2007). Evolving redundant structures for reliable circuits – lessons learned. In *Adaptive Hardware and Systems (AHS)*, pages 455–462.

e Flux (2011). Evolutionary microfluidix.

Eiben, A. E. (2002). Multiparent recombination in evolutionary computing. In Ghosh, A. and Tsutsui, S., editors, *Advances in Evolutionary Computing*, Natural Computing Series, pages 175–192. Springer.

Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors (1998). *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, number 1498 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, New York.

Eiben, A. E., Elia, D., and van Hemert, J. I. (1999a). Population dynamics and emerging mental features in AEGIS. In Banzhaf et al. (1999), pages 1257–1264.

Eiben, A. E., Haasdijk, E., and Bredeche, N. (2010a). Embodied, on-line, on-board evolution for autonomous robotics. In Levi, P. and Kernbach, S., editors, *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, chapter 5.2, pages 361–382. Springer.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999b). Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141.

Eiben, A. E., Karafotias, G., and Haasdijk, E. (2010b). Self-adaptive mutation in on-line, on-board evolutionary robotics. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 147–152. IEEE Press, Piscataway, NJ.

Eiben, A. E., Kernbach, S., and Haasdijk, E. (2012). Embodied artificial evolution – artificial evolutionary systems in the 21st century. *Evolutionary Intelligence*, to appear.

Eiben, A. E., Schoenauer, M., Laredo, J. L. J., Castillo, P. A., Mora, A. M., and Merelo, J. J. (2007). Exploring selection mechanisms for an agent-based distributed evolutionary algorithm. In Bosman et al. (2007), pages 2801–2808.

Eiben, A. E. and Smit, S. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

Eiben, A. E. and Smith, J. E. (2008). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer.

Eklund, S. E. (2004). A massively parallel architecture for distributed genetic algorithms. *Parallel Computing*, 30(5 – 6):647 – 676.

El-Beltagy, M., Nair, P., and Keane, A. (1999). Metamodeling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In Banzhaf et al. (1999), pages 196–203.

Elfwing, S., Uchibe, E., Doya, K., and Christensen, H. (2005). Biologically inspired embodied evolution of survival. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Edinburgh, UK. IEEE Press.

Engelberg-Kulka, H., Amitai, S., Kolodkin-Gal, I., and Hazan, R. (2006). Bacterial programmed cell death and multicellular behavior in bacteria. *PLoS Genetics*, 2(10):1518–1526.

Epstein, J. and Axtell, R. (1996). *Growing Artificial Societies: Social Sciences from Bottom Up*. Brooking Institution Press and The MIT Press.

Fatikow, S. (2008). *Automated Nanohandling by Microrobots*. Springer-Verlag, London.

Fellermann, H. and Rasmussen, S. (2011). On the growth rate of non-enzymatic molecular replicators. *Entropy*, 13(10):1882–1903.

Fernando, C., Von Kiedrowski, G., and Szathmáry, E. (2007). A stochastic model of nonenzymatic nucleic acid replication: "elongators" sequester replicators. *Journal of Molecular Evolution*, 64:572–585.

Fialho, A., Costa, L., Schoenauer, M., and Sebag, M. (2008). Extreme value based adaptive operator selection. In Rudolph et al. (2008), pages 175–184.

Ficici, S., Watson, R., and Pollack, J. (1999). Embodied evolution: A response to challenges in evolutionary robotics. In Wyatt, J. L. and Demiris, J., editors, *Proceedings of the Eighth European Workshop on Learning Robots*, pages 14–22.

Fisher, R. (1930). *The Genetical Theory of Natural Selection*. Oxford University Press, Oxford, UK.

Floreano, D., Husbands, P., and Nolfi, S. (2008). Evolutionary Robotics. In *Handbook of Robotics*. Springer Verlag, Berlin.

Floreano, D. and Mondada, F. (1996). Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:396–407.

Floreano, D., Nolfi, S., and Mondada, F. (2001). Co-Evolution and Ontogenetic Change in Competing Robots. In *Advances in the Evolutionary Synthesis of Intelligent Agents*. MIT Press.

Floreano, D., Schoeni, N., Caprari, G., and Blynel, J. (2002). Evolutionary bits'n'spikes. In Standish, R. K., Bedau, M. A., and Abbass, H. A., editors, *Artificial Life VIII : Proceedings of the eighth International Conference on Artificial Life*, pages 335–344, Cambridge, MA, USA. MIT Press.

Freitas, R. and Gilbreath, W. P., editors (1982). *Advanced Automation for Space Missions*. NASA Conference Publication CP-2255 (N83-15348), Illinois.

Fujita, M. and Yamaguchi, Y. (2009). Mesoscale modeling for self-organization of colloidal systems. *Current Opinion in Colloid & Interface Science*.

García-Sánchez, P., Eiben, A., Haasdijk, E., Weel, B., and Merelo-Guervós, J.-J. (2012). Testing diversity-enhancing migration policies for hybrid on-line evolution of robot controllers. In Di Chio, C., Agapitos, A., Cagnoni, S., Cotta, C., Fernandez de Vega, F., Di Caro, G., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Langdon, W., Merelo-Guervós, J.-J., Preuss, M., Richter, H., Silva, S., Simões, A., Squillero, G., Tarantino, E., Tettamanzi, A., Togelius, J., Urquhart, N., Uyar, A., and Yannakakis, G., editors, *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, pages 52–62.

Giacobini, M., Preuss, M., and Tomassini, M. (2006). Effects of scale-free and small-world topologies on binary coded self-adaptive CEA. In Gottlieb, J. and Raidl, G. R., editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006*, volume 3906 of *LNCS*, pages 85–96, Budapest. Springer Verlag.

Gilbert, N., den Besten, M., Bontovics, A., Craenen, B., Divina, F., Eiben, A. E., Griffioen, A. R., Hévézi, G., Lörincz, A., Paechter, B., Schuster, S., Schut, M., Tzolov, C., Vogt, P., and Yang, L. (2006). Emerging artificial societies through learning. *Journal of Artificial Societies and Social Simulation*, 9(2).

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Gong, J., Wan, L., Yuan, Q., Bai, C., Jude, H., and Stang, P. (2005). Mesoscopic self-organization of a self-assembled supramolecular rectangle on highly oriented pyrolytic graphite and au(111) surfaces. *PNAS*, 102(4):971–974.

Gordon, T. and Bentley, P. (2002). On evolvable hardware. In Ovaska, S. and Sytandera, L., editors, *Soft Computing in Industrial Electronics*, pages 279–323. Physica-Verlag, Heidelberg, Germany.

Gorges–Schleuter, M. (1998). A comparative study of global and local selection in evolution strategies. In Eiben et al. (1998), pages 367–377.

Gribovskiy, A. and Mondada, F. (2009). Real-Time Audio-Visual Calls Detection System for a Chicken Robot. In *Proceedings of the 4th International Conference on Advanced Robotics*, pages 1–6. IEEE Press.

Groß, R., Bonani, M., Mondada, F., and Dorigo, M. (2006). Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22:1115–1130.

Groß, R. and Dorigo, M. (2008). Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285.

Haasdijk, E., Eiben, A., and Winfield, A. F. (2011a). Individual, social and evolutionary adaptation in collective systems. In Kernbach, S., editor, *Handbook of Collective Robotics - Fundamentals and Challenges*, chapter 13. Pan Stanford, Singapore.

Haasdijk, E., Eiben, A. E., and Karafotias, G. (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 1–7, Barcelona, Spain. IEEE Computational Intelligence Society, IEEE Press.

Haasdijk, E., ul Qayyum, A. A., and Eiben, A. (2011b). Racing to improve on-line, on-board evolutionary robotics. In Krasnogor et al. (2011), pages 187–194.

Haddow, P. C. and Tyrrell, A. M. (2011). Challenges of evolvable hardware: past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, 12(3):183–215.

Hamann, H., Schmickl, T., and Crailsheim, K. (2012). A hormone-based controller for evaluation – minimal evolution in decentrally controlled systems. *Artificial Life*, 18(2):165–198.

Hao, J.-K., Legrand, P., Collet, P., Monmarché, N., Lutton, E., and Schoenauer, M., editors (2011). *Artificial Evolution, 10th International Conference, Evolution Artificielle, EA 2011, Angers, France, October 24-26, 2011*, Lecture Notes in Computer Science. Springer.

Haralick, R. and Shapiro, L. (1992). *Computer and Robot Vision*, volume 1, chapter 7. Addison-Wesley Publishing Company.

Haroun Mahdavi, S. and Bentley, P. J. (2006). Innately adaptive robotics through embodied evolution. *Autonomous Robots*, 20(2):149–163.

Hart, W. and Belew, R. (1991). Optimizing an arbitrary function is hard for the genetic algorithm. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the-FourthInternational Conference on Genetic Algorithms(ICGA '91)*, pages 190–195, San Mateo CA. Morgan Kaufmann Publishers, Inc.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. (1996). Evolutionary robotics: the sussex approach. *Robotics and Autonomous Systems*, 20:205–224.

Hinton, G. and Nowlan, S. (1996). How learning can guide evolution. *Santa Fe Institute Studies In The Sciences Of Complexity*, pages 447–454.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Hooker, J. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42.

Hu, T. and Banzhaf, W. (2010). Evolvability and speed of evolutionary algorithms in light of recent developments in biology. *Journal of Artificial Evolution and Applications*.

Huijsman, R.-J., Haasdijk, E., and Eiben, A. E. (2011). An on-line on-board distributed algorithm for evolutionary robotics. In Hao et al. (2011), pages 119–131.

Hutter, M. (2002). Fitness uniform selection to preserve genetic diversity. In *2002 Congress on Evolutionary Computation (CEC'2002)*, pages 783–788. IEEE Press, Piscataway, NJ.

Hutton, T. J. (2009). The organic builder: A public experiment in artificial chemistries and self-replication. *Artificial Life*, 15(1):21–28.

Ijspeert, A., Hallam, J., and Willshaw, D. (1998). From lampreys to salamanders: evolving neural controllers for swimming and walking. In Pfeifer, R., Blumberg, B., Meyer, J.-A., and Wilson, S., editors, *From Animals to Animats, Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 390–399, Cambridge, MA, USA. MIT Press.

Jelasity, M., Montresor, A., and Babaoglu, O. (2005). Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23:219–252.

Jelasity, M. and van Steen, M. (2002). Large-scale newscast computing on the internet. Technical report, Vrije Universiteit Amsterdam.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12.

Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. In Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C., editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society, Providence.

Jones, T. (1995). Crossover, macromutation, and population-based search. In Eshelman, L., editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, San Francisco.

Jorgensen, M. W., Ostergaard, E. H., and Lund, H. H. (2004). Modular atron: Modules for a self-reconfigurable robot. In *Proc. of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan.

Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., and Kokaji, S. (2005). Automatic locomotion design and experiments for a modular robotic system. *Mechatronics, IEEE/ASME Transactions on*, 10(3):314–325.

Karafotias, G., Haasdijk, E., and Eiben, A. (2011). An algorithm for distributed online, on-board evolutionary robotics. In Krasnogor et al. (2011), pages 171–178.

Keane, A. J. and Brown, S. M. (1996). The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. In Parmee, I. C., editor, *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 96*, pages 107–113, Plymouth. PEDC.

Kendall, G. and Su, Y. (2007). Imperfect evolutionary systems. *IEEE Transactions on Evolutionary Computation*, 11(3):294 –307.

Kernbach, S., editor (2011). *Handbook of Collective Robotics: Fundamentals and Challenges*. Pan Stanford Publishing, Singapore.

Kernbach, S., Girault, B., and Kernbach, O. (2011). On self-optimized self-assembling of heterogeneous multi-robot organisms. In Meng, Y. and Jin, Y., editors, *Bio-Inspired Self-Organizing Robotic Systems*, volume 355 of *Studies in Computational Intelligence*, pages 123–141. Springer Berlin / Heidelberg.

Kernbach, S., Meister, E., Scholz, O., Humza, R., Liedke, J., Rico, L., Jemai, J., Havlik, J., and Liu, W. (2009a). Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution. *2009 IEEE Congress on Evolutionary Computation*, pages 1079–1086.

Kernbach, S., Scholz, O., Harada, K., Popesku, S., Liedke, J., Raja, H., Liu, W., Caparrelli, F., Jemai, J., Havlik, J., Meister, E., and Levi, P. (2010). Multi-robot organisms: State of the art. In *2010 IEEE International Conference on Robotics and Automation workshop*, pages 1 – 10. IEEE Press.

Kernbach, S., Thenius, R., Kernbach, O., and Schmickl, T. (2009b). Re-embodiment of honeybee aggregation behavior in artificial micro-robotic system. *Adaptive Behavior*, 17(3):237–259.

Khalil, A. S. and Collins, J. J. (2010). Synthetic biology: applications come of age. *Nature Reviews Genetics*, 11:367–379.

Kovac, M., Fuchs, M., Guignard, A., Zufferey, J.-C., and Floreano, D. (2008). A miniature 7g jumping robot. In Hutchinson, S., editor, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2008)*, pages 373 – 378.

Koza, J. (1992). *Genetic Programming*. MIT Press, Cambridge, MA.

Koza, J. R., Keane, M. A., Yu, J., Bennett, F. H., and Mydlowec, W. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1:121–164.

Kramer, O. (2010). Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3(2):51–65.

Krasnogor, N. (2002). *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of the West of England. Supervisor: Dr. J.E. Smith.

Krasnogor, N., Lanzi, P. L., Engelbrecht, A., Pelta, D., Gershenson, C., Squillero, G., Freitas, A., Ritchie, M., Preuss, M., Gagne, C., Ong, Y. S., Raidl, G., Gallager, M., Lozano, J., Coello-Coello, C., Silva, D. L., Hansen, N., Meyer-Nieberg, S., Smith, J., Eiben, A. E., Bernado-Mansilla, E., Browne, W., Spector, L., Yu, T., Clune, J., Hornby, G., Wong, M.-L., Collet, P., Gustafson, S., Watson, J.-P., Sipper, M., Poulding, S., Ochoa, G., Schoenauer, M., Witt, C., and Auger, A., editors (2011). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Dublin, Ireland. ACM.

Kumar, S. (2006). Self-organization of disc-like molecules: chemical aspects. *Chemical Society Reviews*, 35(1):83–109.

Kutzer, M. D. M., Moses, M. S., Brown, C. Y., Scheidt, D. H., Chirikjian, G. S., and Armand, M. (2010). Design of a new independently-mobile reconfigurable modular robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2758–2764. IEEE.

Lamarck, J. B. (1809). *Philosophie zoologique, ou Exposition des considérations relatives à l'histoire naturelle des animaux*. H.R. Engelmann.

Langton, C., editor (1995). *Artificial Life: an Overview*. MIT Press, Cambridge, MA.

Laredo, J. L. J., Eiben, A. E., van Steen, M., and Guervós, J. J. M. (2010). EvAg: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246.

Lazo, A. and Rathie, P. (1978). On the entropy of continuous probability distributions. *IEEE Transactions on Information Theory*, 24(1):120–122.

Lee, K. and Chirikjian, G. (2007). Robotic self-replication. *Robotics Automation Magazine, IEEE*, 14(4).

Levi, P. and Kernbach, S., editors (2010). *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*. Springer Verlag.

Lobo, F., Lima, C., and Michalewicz, Z., editors (2007). *Parameter Setting in Evolutionary Algorithms*. Springer.

Lund, K., Manzo, A. J., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M. N., Walter, N. G., Winfree, E., and Yan, H. (2010). Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210.

Marocco, D. and Nolfi, S. (2006). Origins of communication in evolving robots. In *From Animals to Animats 9: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior*, Lecture Notes in Computer Science, pages 789–803. Springer, Berlin.

Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11:193–225.

Martel, S., André, W., Mohammadi, M., Lu, Z., and Felfoul, O. (2009). Towards swarms of communication-enabled and intelligent sensotaxis-based bacterial microrobots capable of collective tasks in an aqueous medium. *2009 IEEE International Conference on Robotics and Automation*, pages 2617–2622.

Martinoia, S., Sanguineti, V., Cozzi, L., Berdondini, L., van Pelt, J., Tomas, J., Masson, G. L., and Davide, F. (2004). Towards an embodied in vitro electrophysiology: the NeuroBIT project. *Neucomputing*, 58-60:1065–1072.

Martinoli, A. (1999). *Swarm Intelligence in Autonomous Collective Robotics from Tools to the Analysis and Synthesis of Distributed Control Strategies*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne.

Maturana, J., Lardeux, F., and Saubion, F. (2010). Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 16:881–909.

Maurer, S. E., DeClue, M. S., Albertsen, A. N., Dörr, M., Kuiper, D. S., Ziock, H., Rasmussen, S., Boncella, J. M., and Monnard, P.-A. (2011). Interactions between catalysts and amphiphilic structures and their implications for a protocell model. *ChemPhysChem*, 12(4):828–835.

Mayley, G. (1996). Landscapes, learning costs, and genetic assimilation: Modeling the evolution of motivation. *Evolutionary Computation*, 4(3):213–234.

Mazzolai, B., Mattoli, V., Laschi, C., Salvini, P., Ferri, G., Ciaravella, G., and Dario, P. (2008). Networked and cooperating robots for urban hygiene: the eu funded dustbot project. In *The 5th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2008)*.

McCaskill, J. S., Packard, N., Rasmussen, S., and Bedau, M. A. (2007). Evolutionary self-organization in complex fluids. *Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences*, 362(1486):1763–1779.

Menczer, F. and Belew, R. (1996). From complex environments to complex behaviors. *Adaptive Behavior*, 4(3–4):317–363.

Miller, G. F. and Todd, P. M. (1995). The role of mate choice in biocomputation: Sexual selection as a process of search, optimization and diversification. In *Evolution and Biocomputation, Computational Models of Evolution*, pages 169–204. Springer-Verlag.

Miranda, E. R., Bull, L., Gueguen, F., and Uroukov, I. S. (2009). Computer music meets unconventional computing: Towards sound synthesis with in vitro neuronal networks. *Comput. Music J.*, 33(1):9–18.

Mitchell, M. and Forrest, S. (1994). Genetic algorithms and artificial life. *Artificial Life*, 1(3):267–289.

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. W., Floreano, D., Deneubourg, J.-L., Nolfi, S., Gambardella, L. M., and Dorigo, M. (2004). Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17(2/3):193–221.

Montero, E. and Riff, M.-C. (2011). On-the-fly calibrating strategies for evolutionary algorithms. *Inf. Sci.*, 181:552–566.

Moscato, P. (1999). A gentle introduction to memetic algorithms. In Corne, D., Glover, F., and Dorigo, M., editors, *New Ideas in Optimisation*, chapter 14. McGraw-Hill.

Munroe, S. and Cangelosi, A. (2002). Learning and the evolution of language: the role of cultural variation and learning costs in the baldwin effect. *Artificial Life*, 8(4):311–339.

Nakai, J. and Arita, T. (2010). A framework for embodied evolution with pre-evaluation applied to a biped robot. *Artificial Life and Robotics*, 15(2):156–160.

Nannen, V. and Eiben, A. E. (2007). Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 975–980, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Nannen, V., Smit, S. K., and Eiben, A. E. (2008). Costs and benefits of tuning parameters of evolutionary algorithms. In Rudolph et al. (2008), pages 528–538.

Nehmzow, U. (2002). Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In Prince, C., Demiris, Y., Marom, Y., Kozima, H., and Balkenius, C., editors, *Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, number 94 in Lund University Cognitive Studies, Edinburgh, UK. LUCS.

Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370.

Nelson, B., Dong, L., and Arai, F. (2008). Micro/nanorobotics. In Bruno Siciliano, O. K., editor, *Springer Handbook of Robotics*, pages 411–450. Springer.

ni Moreno, A. G. and Amos, M. (2010). Engineered microbial communication for population-level behaviour. In Fellermann, H., Dorr, M., Hanczyc, M. M., Laursen, L. L., Maurer, S., Merkle, D., Monnard, P.-A., Stoy, K., and Rasmussen, S., editors, *Artificial Life XII: Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems*, pages 184–185. MIT Press.

Nitschke, J. R. (2009). Systems chemistry: Molecular networks come of age. *Nature*, 462(7274):736–738.

Nolfi, S. (1997). Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems*, 22(3-4):187–198.

Nolfi, S. and Floreano, D. (1999). Learning and evolution. *Autonomous Robots*, 7(1):89–113.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA.

Nolfi, S. and Parisi, D. (1993). Auto-teaching: networks that develop their own teaching input. In J.L.Deneubourg, H.Bersini, S.Goss, G.Nicolis, and R.Dagonnie, editors, *Proceedings of the Second European Conference on Artificial Life*, pages 845–862, Brussels. MIT Press.

Nolfi, S. and Parisi, D. (1995). Learning to adapt to changing environments in evolving neural networks. Technical Report 95-15, Institute of Psychology, National Research Council, Rome, Italy.

Nolfi, S., Parisi, D., and Elman, J. L. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28.

Nordin, P. and Banzhaf, W. (1995). Genetic programming controlling a miniature robot. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 61–67. AAAI.

Nordin, P. and Banzhaf, W. (1997). An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140.

Novellino, A., D'Angelo, P., Cozzi, L., Chiappalone, M., Sanguineti, V., and Martinoia, S. (2007). Connecting neurons to a mobile robot: an in vitro bidirectional neural interface. *Intell. Neuroscience*, 2007:2–2.

O'Grady, R., Christensen, A., and Dorigo, M. (2008). Autonomous reconfiguration in a self-assembling multi-robot system. *Ant Colony Optimization and Swarm Intelligence*, pages 259–266.

Ostermeier, A., Gawelczyk, A., and Hansen, N. (1994). A derandomized approach to self adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380.

PACE (2004-2008). *PACE: Programmable Artificial Cell Evolution, FP6*. European Communities, Project reference: 002035.

Pasparakis, G., Krasnogor, N., Cronin, L., Davis, B. G., and Alexander, C. (2010). Controlled polymer synthesis-from biomimicry towards synthetic biology. *Chemical Society Reviews*, 39(1):286–300.

Pedersen, M. E. H. and et al. (2008). Parameter tuning versus adaptation: proof of principle study on differential evolution. Technical report, Hvass Laboratories.

Perez, A. L. F., Bittencourt, G., and Roisenberg, M. (2008). Embodied evolution with a new genetic programming variation algorithm. *International Conference on Autonomic and Autonomous Systems*, 0:118–123.

Pfeifer, R. and Bongard, J. C. (2006). *How the Body Shapes the Way We Think: A New View of Intelligence (Bradford Books)*. The MIT Press.

Pollack, J., Lipson, H., Hornby, G., and Funes, P. (2001). Three generations of automatically designed robots. *Artificial Life*, 7(3):215–223.

Potter, M. A., Meeden, L. A., and Schultz, A. C. (2001). Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1337–1343. Morgan Kaufmann.

Ratle, A. (1998). Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In Eiben et al. (1998), pages 87–96.

Rechenberg, I. (1973). *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart.

Reger, B. D., Fleming, K. M., Sanguineti, V., Alford, S., and Mussa-Ivaldi, F. A. (2000). Connecting brains to robots: an artificial body for studying the computational properties of neural tissues. *Artificial Life*, 6(4):307–324.

Regot, S., Macia, J., Conde, N., Furukawa, K., Kjellen, J., Peeters, T., Hohmann, S., de Nadal, E., Posas, F., and Sole, R. (2011). Distributed biological computation with multicellular engineered networks. *Nature*, 469(7329):207–211.

Reynolds, R. G. (1994). An introduction to cultural algorithms. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific Press.

Reynolds, R. G. (1999). Cultural algorithms: Theory and applications. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 367–377. McGraw-Hill, London.

Richerson, P. and Boyd, R. (2005). *Not By Genes Alone: How Culture Transformed Human Evolution*. University of Chicago Press, Chicago, IL.

Rieffel, J. (2006). *Evolutionary Fabrication: The Co-Evolution of Form and Formation*. PhD thesis, Brandeis University.

Rieffel, J. and Sayles, D. (2010). Evofab: a fully embodied evolutionary fabricator. In *Proc. of the Ninth International Conference on Evolvable Systems*, LNCS 6274, pages 372–380.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2nd edition.

Rudolph, G., Jansen, T., Lucas, S. M., Poloni, C., and Beume, N., editors (2008). *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*. Springer.

Ruiz-Hitzky, E., Darder, M., Aranda, P., and Ariga, K. (2010). Advances in biomimetic and nanostructured biohybrid materials. *Adv Mater*, 22(3):323–36.

Ruppin, E. (2002). Evolutionary autonomous agents: A neuroscience perspective. *Nature Reviews Neuroscience*, 3:132–141.

Sayama, H. (2009). Swarm chemistry. *Artificial Life*, 15(1):105–114.

Schmid, G. (2004). *Nanoparticles*. Wiley-VCH Verlag, Weinheim.

Scholz, O., Dieguez, A., and Corradi, P. (2011). Minimalistic large-scale micro-robotic systems. In Kernbach, S., editor, *Handbook of Collective Robotics: Fundamentals and Challenges*, pages 517–541. Pan Stanford Publishing, Singapore.

Schut, M., Haasdijk, E., and Eiben, A. E. (2009). What is situated evolution? In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 3277–3284, Trondheim. IEEE Press.

Schwager, M., Detweiler, C., Vasilescu, I., Anderson, D. M., and Rus, D. (2008). Data-driven identification of group dynamics for motion prediction and control. *Journal of Field Robotics*, 25(6-7):305–324.

Schwarzer, C. (2008). Investigation of evolutionary reproduction in a robot swarm. Master's thesis, Institute of Parallel and Distributed Systems, University of Stuttgart.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Wiley, New York.

Sells, E., Smith, Z., Bailard, S., Bowyer, A., and Olliver, V. (2009). RepRap: the replicating rapid prototyper-maximizing customizability by breeding the means of production. *Handbook of Research in Mass Customization and Personalization*, 1:568–580.

Shen, W.-M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M., and Venkatesh, J. (2006). Multimode locomotion for reconfigurable robots. *Autonomous Robots*, 20(2):165–177.

Siciliano, B. and Khatib, O., editors (2008). *Springer Handbook of Robotics*. Springer.

Simões, E. D. V. and Dimond, K. R. (2001). Embedding a distributed evolutionary system into population of autonomous mobile robots. In *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference*.

Smit, S. K. and Eiben, A. E. (2009). Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 399–406, Piscataway, NJ, USA. IEEE Press.

Smit, S. K. and Eiben, A. E. (2010). Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In et al., C. D. C., editor, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 542–551. Springer.

Smit, S. K. and Eiben, A. E. (2011). Multi-problem parameter tuning using Bonesa. In Hao et al. (2011), pages 222–233.

Smith, R. E., Bonacina, C., Kearney, P., and Merlat, W. (2000). Embodiment of Evolutionary Computation in General Agents. *Evolutionary Computation*, 8(4):475–493.

Stradner, J., Hamann, H., Schmickl, T., Thenius, R., and Crailsheim, K. (2009). Evolving a novel bio-inspired controller in reconfigurable robots. In *ECAL (1)*, pages 132–139.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Tamsir, A., Tabor, J. J., and Voigt, C. A. (2011). Robust multicellular computing using genetically encoded nor gates and chemical /'wires/'. *Nature*, 469(7329):212–215.

Teller, A. and Andre, D. (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H., and Riolo, R., editors, *Proceedings of the 2nd Annual Conference on Genetic Programming*, pages 321–328. MIT Press, Cambridge, MA.

Todd, P. M. and Miller, G. F. (1990). Exploring adaptive agency ii: simulating the evolution of associative learning. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 306–315.

Tomassini, M. (2005). *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Tuci, E., Quinn, M., and Harvey, I. (2002). Evolving fixed-weight networks for learning robots. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1970–1975. IEEE Computer Society.

Turney, P., Whitley, D., and (eds.), R. A. (1996). Evolution, learning, and instinct: 100 years of the baldwin effect. *Special Issue of Evolutionary Computation*, 4(3):iv–viii.

Urzelai, J. and Floreano, D. (2001). Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments. *Evolutionary Computation*, 9(4):495–524.

Usui, Y. and Arita, T. (2003). Situated and embodied evolution in collective evolutionary robotics. In *Proceedings of the 8th International Symposium on Artificial Life and Robotics*, pages 212–215.

Vilbrandt, T., Malone, E., Lipson, H., and Pasko, A. (2008). Universal desktop fabrication. In Pasko, A., Adzhiev, V., and Comninos, P., editors, *Heterogeneous objects modelling and applications*, volume 4889 of *LNCS*, pages 259–284.

Vogt, P. and Haasdijk, E. (2010). Modelling social learning of language and skills. *Artificial Life*, 16(4):289–309.

von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press, Edited and completed by A. W. Burks, Illinois.

Walker, J. H., Garrett, S. M., and Wilson, M. S. (2006). The balance between initial training and lifelong adaptation in evolving robot controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(2):423–432.

Watson, R. A., Ficici, S. G., and Pollack, J. B. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18.

Weel, B., Haasdijk, E., and Eiben, A. (2012). The emergence of multi-robot organisms using on-line on-board evolution. In Di Chio, C., Agapitos, A., Cagnoni, S., Cotta, C., Fernandez de Vega, F., Di Caro, G., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Langdon, W., Merelo-Guervós, J.-J., Preuss, M., Richter, H., Silva, S., Simões, A., Squillero, G., Tarantino, E., Tettamanzi, A., Togelius, J., Urquhart, N., Uyar, A., and Yannakakis, G., editors, *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, pages 124–134. Winner of the best paper award for EvoCOMPLEX 2012.

Wei, H., Cai, Y., Li, H., Li, D., and Wang, T. (2010). Sambot: A self-assembly modular robot for swarm robot. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 66–71. IEEE.

Wickramasinghe, W. R. M. U. K., van Steen, M., and Eiben., A. E. (2007). Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In Bosman et al. (2007), pages 1460–1467.

Wilks, Y. (2010). *Close engagements with artificial companions: key social, psychological, ethical and design issues*. Natural language processing. John Benjamins Pub. Company.

Wischmann, S., Stamm, K., and Wörgötter, F. (2007). Embodied evolution and learning: The neglected timing of maturation. In Almeida e Costa, F., editor, *Advances in Artificial Life: 9th European Conference on Artificial Life*, volume 4648 of *Lecture Notes in Artificial Intelligence*, pages 284–293. Springer-Verlag, Lisbon, Portugal.

Wood, J. M. (1999). Osmosensing by Bacteria: Signals and Membrane-Based Sensors. *Microbiol. Mol. Biol. Rev.*, 63(1):230–262.

Yang, S., Ong, Y.-S., and Jin, Y. (2007). *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*. Springer, Berlin / Heidelberg.

Yim, M., Shen, W. M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., and Chirikjian, G. S. (2007a). Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14(1):43–52.

Yim, M., Shirmohammadi, B., Sastra, J., Park, M., Dugan, M., and Taylor, C. (2007b). Towards robotic self-reassembly after explosion. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2767–2772. IEEE.

Yim, M., Zhang, Y., Roufas, K., Duff, D., and Eldershaw, C. (2003). Connecting and disconnecting for chain self-reconfiguration with polybot. *IEEE/ASME Transactions on mechatronics, special issue on Information Technology in Mechatronics*.

Yin, P., Choi, H. M. T., Calvert, C. R., and Pierce, N. A. (2008). Programming biomolecular self-assembly pathways. *Nature*, 451(7176):318–322.

Yuan, B. and Gallagher, M. (2007). Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In *Parameter Setting in Evolutionary Algorithms*, number 54 in Studies in Computational Intelligence, pages 121–142. Springer, Berlin Heidelberg.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zürich, Gloriastrasse 35, CH-8092 Zürich, Switzerland.

Zykov, V., Mytilinaios, E., Desnoyer, M., and Lipson, H. (2007). Evolved and designed self-reproducing modular robotics. *IEEE Transactions on Robotics*, 23(2):308–319.

Nooit Te Oud Om Te Leren

Online Evolutie van Controllers in Swarm- en Modulaire Robotica

Dit proefschrift beschrijft onderzoek dat is gebaseerd op een visie van autonome robots die hun gedrag, misschien zelfs hun vorm, zelfstandig kunnen aanpassen aan nieuwe taken en omstandigheden: robots, die leren om hun taken uit te voeren zonder tussenkomst van (al dan niet menselijk) toezicht.

Zulke flexibiliteit is de meeste robots niet gegeven; ze zijn ontworpen om specifieke taken ("las de carrosserie aan het chassis") uit te voeren in duidelijk vastgelegde omstandigheden. Om te kunnen omgaan met onverwachte omstandigheden en nieuwe taken moeten robots kunnen leren: ze moeten hun gedrag en/of hun vorm kunnen aanpassen op het moment dat ze met nieuwe omstandigheden en taken geconfronteerd worden. Ons onderzoek richt zich op het vermogen van robots om autonoom te leren, om zich aan te passen zonder extern toezicht.

We kunnen robots zo programmeren dat ze individueel hun eigen gedrag aanpassen zonder ruggespraak of toezicht. Maar als meerdere robots (een *swarm*) dezelfde taak moeten uitvoeren kunnen ze ook van elkaar leren door kennis uit te wisselen. Robots kunnen hun kennis delen door evolutie (evolutionaire aanpassing) of ze kunnen kennis uitwisselen in 'gesprekken' (sociaal leren) om hun individuele leerproces te versterken. Er zijn dus drie manieren waarop een verzameling robots kan leren: individueel, sociaal en evolutionair.

Het onderzoek in dit proefschrift is grotendeels uitgevoerd als onderdeel van het SYMBRION project, dat zich richt op groepen van tientallen robots die zich aan elkaar kunnen vastmaken om zo een groter 'organisme' te maken (als een soort bewegende legosteentjes), maar die ook individueel aan de slag kunnen in 'swarm mode'. De robots moeten kunnen leren, zowel gezamenlijk als individueel, om zo verschillende taken uit te kunnen voeren in allerlei omgevingen: ze moeten de adaptiviteit hebben die gewone hedendaagse robots ontberen.

We hebben vooral veel aandacht besteed aan het individueel aanleren van gedrag. Daarbij was een fundamentele keuze die van evolutie als de belangrijkste manier om adaptiviteit te realiseren. We implementeren dus ook individueel leren door middel van een evolutionair algoritme, waarbij we een verzameling van evoluerende controllers in elke individuele robot inkapselen (*encapsulated evolution*). Dit is misschien wat verwarrend omdat dit geen evolutionair leren op het niveau van een groep robots betekent: de robots wisselen immers geen kennis uit en ze leren onafhankelijk van elkaar. Als we evolutie wel door middel van zo'n uitwisseling van kennis zouden implementeren zou dat wel 'echte' evolutie op het niveau van de groep robots zijn, waarbij de evolutie verdeeld wordt over alle robots (*distributed evolution*). We kunnen ook deze beide mechanismen combineren om zo tot een vorm van sociaal leren te komen.

We hebben aan de hand van vele experimenten de encapsulated aanpak vergeleken met algoritmes die evolutie verdelen over meerdere robots in een collectief en met algoritmes die deze beide aanpakken combineren tot een soort sociaal leren.

We proberen daarmee de volgende vragen te beantwoorden:

- Kunnen we überhaupt evolutionaire algoritmes ontwikkelen waarmee robots autonoom kunnen leren eenvoudige taken aan te pakken?

- Welke aanpak werkt het beste: encapsulated, distributed of een combinatie van die twee?

- Hoe beïnvloeden de instellingen van parameters de kwaliteit van onze algoritmes?

We hebben een encapsulated algoritme genaamd $(\mu + 1)$ ON-LINE ontwikkeld waarmee robots individueel kunnen leren. We hebben gezien dat robots met dit algoritme robots eenvoudige taken kunnen leren zoals zonder botsen rondrijden en patrouilleren. Daarmee kunnen robots dus zulke taken leren uitvoeren zonder dat extern toezicht nodig is.

We hebben ook gedistribueerde en hybride alternatieven voor $(\mu + 1)$ ON-LINE getest en we hebben vastgesteld dat het in het algemeen beter is om gezamenlijk te leren, maar dat men moet oppassen als de taak impliciet tot en wedloop tussen de individuele robots leidt.

Zoals verwacht hebben uitgebreide experimenten aangetoond dat de kwaliteit van de gevonden oplossingen sterk afhangt van de instellingen van de algoritmes.

We hebben ook gezien dat er geen Haarlemmerolie is: er zijn geen instellingen die in alle gevallen goed uitpakken. Daaruit concluderen we dat verder onderzoek nodig is naar de mogelijkheid om de instellingen van de algoritmes automatisch, dus terwijl de robots doorwerken, in te stellen al naar gelang de omstandigheden en de taak.

SIKS Dissertation Series

2009

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks – Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation – Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controlling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

2010

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia Documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History – Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting – the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

2011

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age – Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind's Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering – A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication – Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein (UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov (TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes

2012-16 Fiemke Both (VU)
Helping people by understanding them – Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text

2012-27 Hayrettin Görkük (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN)
A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OU)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics