Overview of Incremental Approaches to Evolutionary Robotics

Pavel Petrovic, Department of Computer and Information Science,
Norwegian University of Science and Technology, 7041 Trondheim, Norway

Abstract - Evolutionary Computation (EC) as a general learning method can be applied to the design of robot controllers, programs, hardware or even physical architectures. Our focus is on those that aim on the design of adaptive behaviors for autonomous and possibly learning agents. Many successful experiments showed that controllers for task requiring simple behaviors could be evolved. The amount of work required to set up the evolutionary process has been usually greater than the amount required by a human design. A ground for continuing the research is a vision that EC methods can be used for the design of controllers for robots with more complicated tasks. However, a simple evolutionary process is computationally too expensive and works only with tasks of limited complexity. Incremental evolution can provide a solution by dividing the complicated tasks into reasonable steps, easier for evolution to solve. This is the approach we investigate also at Evolutionary Computation and Artificial Life (EVAL) Laboratory at Norwegian University of Science and Technology. This article provides a short introduction to the Evolutionary Robotics (ER) and reviews the work on incremental evolution done by different research groups.

Introduction

Typical tasks, where robots can be useful, share common properties, such as hazardous or uncomfortable for humans, requiring heavy manipulation or tools, which are awkward to handle, necessitating repetitive execution. Whereas some jobs are suitable for fixed robots on a factory production line, other tasks require mobility, flexibility, and adaptation to dynamic environment. To certain extend, it is possible to use remotely controlled or human operated robots. However, the amount of information sent from sensors and to actuators can be too large for transmission, the mission can be too distant for controlling each step of the robot in real time, and the connection can be lost easily in certain environments. Certain responses to events in the environment might have to be performed very quickly. In all such cases, it is inevitable and/or more cost efficient to equip the robot with its own controller, which makes it autonomous.

A traditional approach of Artificial Intelligence (AI) uses the top-down method, where the overall goal of the system (often overestimating the technological possibilities) is partitioned into sub-modules that are developed individually. When putting such modules together, there is a large risk that the interfaces are incompatible. It can also be found that a particular module cannot satisfy the physical constraints of some robot parts implied by the top-down design. Moreover, the internal architecture typically consists of a large planning module with symbolic reasoning mechanisms. This module receives the sensory information and should generate next action of the robot. However, symbolic reasoning mechanisms are generally slow and futile for reactive behaviors that each mobile robot tackles. Even if the technological possibilities allow building a robot according to a plan, the system is very difficult to debug and maintain.

A modern approach of the ‘new AI’ relies on building the robot incrementally. The most trivial reactive behaviors, such as obstacle avoidance are fully implemented, tested, and debugged before carefully appending higher levels of behaviors. These ideas

are applied in the Subsumption Architecture [Brooks, 1986]. It's disadvantage of hard-wiring the behaviors was later addressed in [Maes, 1990], where individual behaviors compete for robot control and the robot can learn the conditions of applicability of each behavior in different situations. The research challenge that has not been solved yet is whether the robot controller can be designed automatically so that it would perform some nontrivial useful task, and whether the automatic method can be more time or cost efficient than human engineered solution. The following sections describe the ER field, explore the tasks for which the robot controllers were successfully evolved, sketch different controller architectures that are used in ER, and a brief overview of issues related to evolution in simulation. The main focus of the article is on what can be incremental in ER and how different research groups tried to do evolution incrementally. Future directions and conclusions finalize the article.

Evolutionary Robotics

Different approaches to automatic design of robot controllers were studied. Most known are Reinforcement Learning [Sutton, 1984], training various Neural Networks, Classifier Systems (CS) [Wilson, 1989], Genetic Programming (GP) [Koza, 1992a], and artificial evolution used with various controller architectures. The latter three use some form of evolutionary technique and roughly compose the sub-field of AI labeled Evolutionary Robotics.

Briefly, ER (and EC in general) relies on evolution of population (set) of individuals (solutions to a problem). This set is usually of a fixed size. Solutions are not only good or bad, their quality can be measured by the fitness evaluation function. Fitness function returns a number, which determines the quality of a solution. In the beginning of EC run, a random population is created (generation zero). Next generation is created from the previous by combining higher quality solutions and introducing few random changes into them. This requires the solutions to be encoded in some uniform way (genotype). For example, bitstrings are used in Genetic Algorithm (GA) [Holland, 1975], lisp S-expressions in GP or sequences of machine instructions, C-language programs [Ryan, 1998], etc. in other methods. EC run goes on for many generations, and the result is the best individual found during the evolution. Therefore a problem of designing a robot controller using ER method has typically the following steps:

- Define the target behavior(s) that the robot will have to accomplish – the task for the robot.
- Design the physical body of the robot and its hardware, decide on the sensors and actuators that the robot will use. Create a specification of signals coming from sensors and to actuators.
- Decide on the controller architecture, modularity, and software platform.
- Choose which of the modules and their parts can be easily designed manually and identify those, which are suitable for automatic design. If there are more parts for automatic design, choose whether they will be evolved simultaneously or individually.
- Find uniform encodings for automatically designed parts and define fitness functions.
- Based on experience and some experiments, set up the parameters for particular evolutionary algorithm and run the evolution.
- Optimize, analyze, and test the evolved parts (ideally, prove their correctness), integrate with the other parts of the controller.

There are still some unanswered questions about how to define the behavior and how to analyze and check the evolved controller based on that definition later. This process could be also automated. The combination of several behaviors can take different forms, such as independent sum, combination, suppression, and sequence as studied in [Colombetti, 1992].

Some researchers argue for simultaneous evolution of the robot physical topology (body) or hardware and its controller (brain) [Brooks, 1992, Lund, 1998].

Evolvable tasks

Although ER has been tested on many different tasks, most of them were simple. These include the following:

- Wall following, where the robot is placed in a closed environment and has to learn navigation along the walls without collision. Robot is usually equipped with laser, sonar, or infrared proximity sensors and sometimes has a vision.
- Obstacle avoidance is typically a part of some more complicated task. The goal for the robot is to navigate in the environment without running into obstacles. The environment can be static or contain moving objects.
- Docking and recharging, where the robot has to find its docking station and successfully approach it.
- Artificial ant following is usually used in Artificial Life simulation. Robot is trained to follow a chemical trace by using its smell sensor.
- Box pushing has several variations. A robot or a group of them are given a task of pushing box(es) to the wall, corners or specified positions.
- In lawn mowing task, the robot has to move around inside of a defined arena and cover the largest possible area. The environment may contain obstacles, irregularities and moving objects.
- Legged walking is used with 2,4,6,8-legged robot. The task is to train the controller to synchronize the movements of the robot.
- T-maze navigation is a standard benchmark task. The robot first reads the direction at the entrance to a corridor. It has to follow the corridor to the crossing and turn right or left based on the initial instruction.
- Various foraging strategies are also related mainly to Artificial Life, where the artificial environment contains food sources and the robot has to learn the strategies for finding the food.
- Trash collection, is another example of searching. The robot has to pick up the trash objects, and carry them to an assigned area to drop them.
- In various vision discrimination and classification tasks, the controller is trained to steer the robot depending on the vision sensory inputs.
- In target tracking and navigation, the robot has to follow the target so that it remains in its vision system.
- Various aspects of pursuit-evasion behaviors were also analyzed and the co-evolution method was usually applied (see below).

Controller architectures

A robot controller is responsible for selecting an action for the robot to perform, based on the current and past sensory readings and its knowledge. It is usually a combination of specialized hardware and a software running on some embedded microprocessor.

However, we are interested only in the conceptual (logical) view abstracting from the platform, implementation or other technical details.

Many experimental robot controllers are built as some sort of neural network. The simplest is perhaps a feed-forward NN, see for example [Floreano, 1994]. Direct sensory inputs are fed into the layered network, the values propagate through weighted connections, and the sum of inputs in each node is usually transformed by nonlinearity before the node outputs the signal to the next layer. The output signals from the last layer are sent to the robot actuators. Since this type of network cannot have an internal state, it is limited to tasks and environments that are tractable with completely reactive behavior. The connection weights can be either evolved or trained by a learning algorithm.

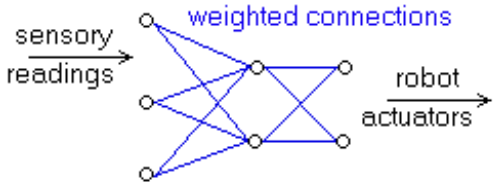

Figure 1. Feed-forward NN architecture.

A feed-forward NN can be extended to contain an internal state by appending memory units as additional input units. They contain a copy of the outputs of nodes from previous iteration. In general, this type of architecture – recurrent NN, can achieve any type of behavior.

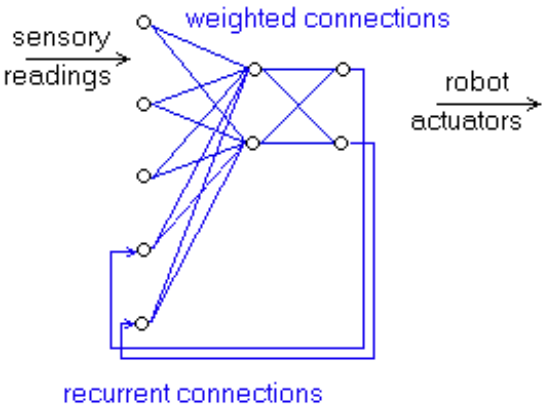

Figure 2. Recurrent NN with memory units.

In ER, more popular architectures are modifications of recurrent NN that contain arbitrary connections from any node to any other node. These include for example dynamical neural networks of Gallagher and Beer [Beer, 1992], in which the neuron

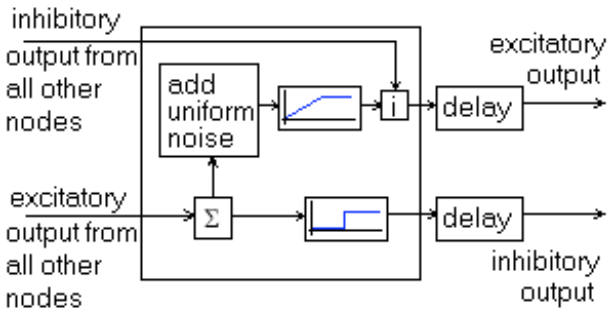

Figure 3. Node of a fully interconnected recurrent NN used by a Sussex group.

excitations are updated in continuous time and each neuron has its memory constant that determines how quickly is the activation changed. A Sussex group also uses fully interconnected recurrent NN with binary inhibitory connections.

NN were also used in modular architectures. One possibility is to allocate a separate NN for each module. In another solution (emergent modular architecture, [Nolfi, 1997]), a single NN embraces all the modules, but the outputs consist of two values produced by selector neurons and output neurons. The job of selector neurons is to indicate whether the situation is appropriate for the output neuron value to be taken into account. Modular NN architectures can be evolved the same way as traditional NN, or the modularity description can be also contained in the genotype.

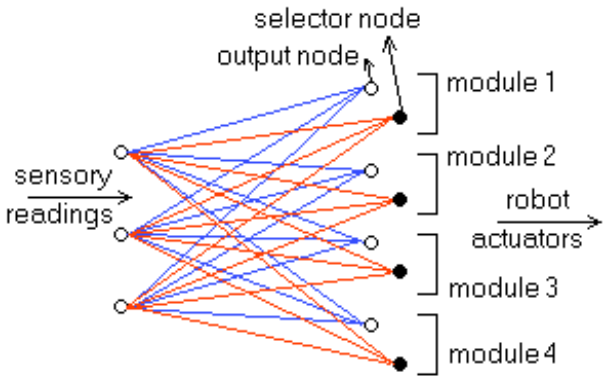

Figure 4. Emergent modular architecture; modules compete for control over their assigned actuator.

Classifier Systems (CS) compose an extensive group of controller architectures for adaptive robots. They typically consist of three levels. The lowest level is an immediate control of the robot actuators based on the sensory readings and a memory. The actions are usually selected by IF-THEN rules. Second level is a learning mechanism responsible for assigning the credit to the rules, which lead to a successful behavior. Rules with higher credit survive, unsuccessful rules die out. CS are in this way a version of reinforcement learning and use a Bucket-brigate algorithm to propagate the credit backwards through the sequence of rules that result in the useful action of the robot. The third level is responsible for finding new promising rules and is implemented using GA. For a review of CS, see [Wilson, 1989].

A very popular way of automatic controller design is the Genetic Programming. Typically, an action of a robot is determined by an output of one or more lisp-like S-expressions consisting of aritmetical operations and mathematical functions, constants, and special operators for sensory readings. These S-expressions are evolved by GA, and because the representation is not length-uniform, special recombination operators should be used. A very first example of aplication of GP to robot control is in [Koza, 1992b].

Instead of S-expressions, GP techniques can be used with sequential assembler-like programs, sets of IF-THEN or fuzzy rules [Braunstingl, 1995] or programs in any language, for which it is possible to generate them from definition grammar rules.

Simulation and evolution on real robots

The proper use of simulation is a notorious topic in ER. Evaluation of evolved controllers on real robots is very time consuming. In addition, it is difficult to ensure the same conditions for evaluating the individuals in a population. Several researchers successfully demonstrated that evolution on real robots is possible [Floreano, 1994,

Floreano, 1996]. Nevertheless, even very simple tasks required too many evaluations (each individual program in each generation should be tested, preferably from several starting points, different world configurations, etc.). For example, Marc Ebner demonstrated at the EvoRobot'99 [Ebner, 1998] a wall following mobile robot equipped with sonars, with an evolved controller. GP evolution was running for two months. More reliable and human-designed program for the same task was created in few minutes! The use of simulation is accordingly necessary in most cases. The difficulty lies in the inconceivability of modeling the real world in simulation. The simulated sensory inputs will always differ from those obtained in the real world by real sensors. Real sensors are usually not ideal and may require calibration. In addition, EC techniques are known to be very good in finding any unwanted regularity, and thus creating fragile solutions, which hardly transfer from simulation to reality.

The traditional solution to this problem is based on adding the noise to the simulated sensory readings and combining the evaluation of individuals in simulation with evaluation on real robots. Usually, the controllers are first evolved in simulation, and later tested, tuned, or even evolved further on real robots. Real world tests can provide feedback for the simulator to modify the simulation [Brooks, 1992]. In [Lund, 1998], the simulated sensory readings were not computed mathematically from the sensor parameters, but retrieved from look-up tables that were created by measuring the real sensors in different situations. More details on the issue of simulation can be found for example in [Meeden, 1998].

Incremental evolution

A challenge of evolving a robot controller for a complex task is too difficult for a simple evolutionary algorithm. However, EC can be used for automatic controller design for more complex behaviors, if the task is divided into smaller subtasks. These subtasks can be either independent, for example individual modules that can be tested separately, or depend to some extend on other tasks – thus creating a dependence hierarchy (i.e. arranged in a tree, or in a graph). In this case, one can incrementally evolve the behaviors from the bottom of the hierarchy, and later add upper layers. This is a general framework, but various researchers already executed more concrete and specific experiments.

We should make a comment here that an interesting work on incremental learning is done in the Machine Learning community, [Lange, 1995] to name one.

A small and not very promising exercise of incrementally evolving a simple NN mapping of 4-bit binary vectors was presented in [Garis, 1993]. He first evolved weights for a 12-node NN for 3 binary vector pairs. Then he added 4 nodes to the evolved networks and continued the evolution with one more binary vector pair. He compared the process with the evolution in one non-incremental step (4 binary vector pairs and 16-node NN). Incremental case resulted in worse fitness solutions and took longer time. The task was easy even for a simple evolution and in the second evolutionary step; the level of redundancy in learning was high, because the 3 vectors were already learned by part of the network.

Brooks' Subsumption architecture was also evolved. The behavior language BL was extended to version suitable for EC called GEN. In [Brooks, 1992], he describes that the robot should be initially operated with only some of its sensors, and perhaps only some of its actuators. Once the fundamental behaviors are present, additional sensors and actuators can be made available. The fitness function can vary over the time.

Much attention is given to Inman Harvey's Spiecies Adaptation Genetic Algorithm (SAGA) [Harvey, 1992] and experiments with incremental evolution of NN architectures for adaptive behavior [Cliff, 1992]. SAGA is a label for a modified version of GA that allows genotypes with variable (growing) length. The main distinction that Harvey makes is that GA is a general problem solving optimization and search tool working with a finite search space. In contrast, SAGA is required when trying to evolve a structure with arbitrary and potentially unrestricted capabilities, i.e. when the genotypes must be unrestricted in length. In SAGA, incremental refers only to the fact that the length of the genotype increments over the evolutionary run. In their practical experiments, group at Sussex usually uses an arbitrary interconnected network with inhibitory and excitatory connections. Internal uniform noise is added to node's excitation. Inhibition is binary and once the node receives at least one inhibitory signal, it does not produce any excitatory output. Certain level of excitation results in inhibitory output. The task for evolution is to find the excitatory and inhibitory connections. SAGA has been tested on simple visually guided robot that had to remain in the centre of the arena. Later, the Sussex group experimented also with increments in the task difficulty. A visually guided gantry robot was learned the task of navigation towards a triangle in four steps: forward movement, movement towards a large target, movement towards a small target, ditinguishing a triangle from a square [Harvey, 1997].

Lund and Miglino incrementally evolved a Kephera robot with recurrent NN controller that performed detour behavior [Lund, 1998]. The task for the robot was to reach a target placed behind a U shaped obstacle. They failed to evolve such a controller in a single evolution, but succeeded with two-step process. First, the robot was trained for a rectangular obstacle. Later, one with an original U shape replaced the rectangular.

In [Floreano, 1992], it is demonstrated that evolution in changing environment can lead to better results than static environment. In his experiments with nest-based foraging strategies, Floreano first evolved feed-forward reactive NN controllers in environment with a constant amount of food. He compared it to an evolution in environment, where the amount of food decreases, thus the task becomes incrementally more difficult. He reported that environmental change caused a drastic improvement in the quality and efficiency of the foraging strategies. In the later work at Lausanne [Urzelai, 1998], a more advanced modular architecture was trained using Behavior Analysis and Training (BAT) [Colombetti, 1996, Dorigo, 1998]. Evolution as a global search was combined with the Reinforcement Learning during the individual's lifetime. The robot first learned to move around the arena as much and as long as possible, avoiding obstacles, and periodically recharging the batteries at the recharging station. Later, a gripper was attached to the robot and a new task of collecting the highest number of objects and releasing them outside the arena was given. New modules were added to the controller architecture and the genotype was augmented. However, already evolved parts of the genotype were masked so crossover and mutation operators did not affect them.

A group at the University of Texas [Gomes, 1997] used incremental evolution in combination with Enforced Sub-Populations (ESP). In this method, recurrent NN architectures were evolved. But members of the population were individual neurons segregated into sub-populations. The network was formed by randomly selecting one neuron from each sub-population. Delta-coding GA [Whitley, 1991] was used for reinitialization of population based on differences from best-fit individual. Prey capture behavior of a mobile agent was evolved in 8 incremental steps. First, the prey was static. Later, it was allowed to perform several initial steps and finally it was made mobile with incrementing speed in consecutive evolutionary steps. Authors mention two heuristics

for devising an incremental sequence of evaluation tasks. (i) Increasing the density of the relevant experiences within a trial so that a network can be evaluated based on greater information in a shorter amount of time. (ii) Make the evaluation-task easier so that the acquisition of fundamental goal-task (final task) skills is more feasible. Results of the prey capture experiments showed significantly better fitness in case of incrementally evolved controllers compared to the direct evolution.

Similar approach is taken by a group at the University of Edinburgh. Perkins and Hayes argue [Perkins, 1996] that evolution of NN controllers as individuals in the population is too difficult in incremental steps. Their arguments are: networks with internal states are not very suitable for breeding; there are difficulties with using the converged population of the previous incremental step as an initial population for the next step; protecting parts of the network responsible for behaviors learned in previous steps is impossible because these parts are not identifiable. Their Robot Shaping method is closer to a classifier system. A population of neurons that compete and cooperate to produce the behavior of the robot is evolved. The network is made up of several different species of neurons, which have different connectivity characteristics and different mutation operators. Neurons are evaluated by an analog to the bucket-brigade algorithm from CS. In their later experiments [Perkins, 1998a] and Perkins' Ph.D. thesis [Perkins, 1998b], they evolved a target following behavior for B21 mobile robot. The task was to keep the brightest object in front, centered in its visual field. Robot was trained in two incremental steps: first only turning towards its target and later also moving towards it (pan and tilt axes). The controller consisted of several tree-like programs (agents) evolved with GP. Agents had two outputs: validity and value and they competed for the control over their assigned actuator. Before proceeding to next evolutionary step, the current agents in the controller were frozen and the evolution continued only with appended agents. The incrementally shaped controller performed significantly better than non-shaped, although handcrafted controller was simpler and better.

A systematic study of incremental evolution in GP is in [Winkler, 1998]. Authors distinguish between the true incremental evolution, such as in [Nolfi, 1994] from other techniques where the controller is trained in steps and previously evolved parts are held constant in subsequent steps. Authors experiment with two different termination criteria: fixed number of generations and achieving the performance limit. They employ two different population organization strategies: standard undivided population and demetic grouping, where the population consists of several isolated sub-populations that exchange the genotypes only occasionally. They statistically compare named methods applied to target tracking task of pan and tilt controlled mobile robot.

An interesting example of incremental evolution with GP is in [Fukunaga, 1995]. They evolved programs for 2 tasks. First, a controller for evasion in pursuit-evasion game was evolved in 2 steps: in the first step, the speed of the evader was different. They give a thorough analysis of various speeds (both slower and faster than the final task) and time when the shift between the two steps occurs. They show that even a more difficult task when used as a first step, can sometimes speed-up the evolution. The reason for (problem of) this approach is that by giving a more difficult task in the beginning, the selection pressure is altered thus speeding up the evolution. Similar effect can most likely be obtained also by varying other GA parameters. In the second experiment, Fukunaga and Kahng evolve a controller for an artificial ant following food trail. They use 2 steps again: in the first step, they use simpler (more difficult) trail. Again, their results indicate that more difficult task as the first incremental step can speed up the overall evolution. In both experiments, they were able

to evolve the required behaviors in a shorter time using incremental evolution compared to non-incremental GP.

Another common method used in ER for arranging the gradual increasing of the task difficulty is the co-evolution [Cliff, 1995; Reynolds, 1994; Smith, 1996; Juillé, 1996; Floreano, 1998]. In this approach, two simultaneous competing populations of individuals are evolved. They share the same environment and might have opposite goals. As a consequence, the environment complexity is gradually increased over the generations as the evolution finds better individuals in both populations.

A possible source of fruitful inspiration lies in the theoretical research of EC. Delta-coding [Whitley, 1991] algorithm is a kind of GA that periodically reinitializes the population, monitors the population diversity and incrementally modifies the genotype encoding.

Population-Based Incremental Learning (PBIL) algorithm [Baluja, 1994] is another form of EC, where a single probabilistic vector is maintained and evolved instead of the whole GA population. In each step, the new population is probabilistically generated from this vector and the vector is in incremental steps moved towards the best (or best M) individual(s) in the population. Application of this algorithm into ER is currently being investigated at LAMI in Lausanne.

Ideas of PBIL were extended to GP representations in Probabilistic Incremental Program Evolution (PIPE) [Sałustowicz, 1997a]. This algorithm maintains a single GP-like program tree – a probabilistic prototype tree (PPT) that contains probability distributions of instructions located in all tree nodes. In each generation, a population of GP programs is stochastically generated from the PPT and the probabilities of individual instructions in PPT are moved towards the best individual in the population. The initial experiments (function regression, 6-bits parity, agents in partially observed environments) show that the variance in reliability of PIPE is high. In their later experiments [Sałustowicz, 1997b, 1998a, 1998b], IDSIA group compares PIPE to TD-Q reinforcement learning in case of multiagent learning, “Long Short-Term Memory” for prediction tasks. PIPE was also extended by a general data-driven divide-and-conquer technique for automatic task decomposition (filtering) and hierarchical extension (H-PIPE).

Future directions

Our desire is to further develop the framework for incremental evolution. Are there any robotic tasks that can be mastered by a robot programmed automatically faster, cheaper or with less effort than by one with a full human design? Will the evolutionary approach show successful applications? Will it be possible to enrich the evolutionary automatic process to take into consideration high level definitions of the task of the robot and also the detailed technical description about the robot from some database, perhaps combined with some symbolic reasoning? Will it be possible to speed up the evolutionary process so that it could be used in simulation directly on the robot when it encounters a new problem? How to handle simulated evolution of learning individuals in an unknown environment? What should be the preferred controller architectures for particular tasks?

In addition to addressing these questions with small research and student projects, we use the LEGO Mindstorms robotic construction sets to test our ideas on ‘real’ hardware and develop tools for practical experiments (for example the finite state machine controller).

Conclusion

This article provides a concise summary of currently known studies, experiments, comparisons, and methods of the use of incremental evolution in automatic programming of robots using evolutionary computaion. It outlines the field of Evolutionary Robotics. The architectures used for designing the robot controllers are explained. Issue of simulation is discussed and the descriptions and pointers to work on incremental evolution are given.

References

Baluja S., *Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning*, Technical report CMU-CS-94-163, Carnegie Mellon University, 1994.

Beer R.D., Gallagher J.C., *Evolving Dynamical Neural Networks for Adaptive Behavior*, Adaptive Behavior, Volume 1, Number 1, p. 91-122, 1992.

Braunstingl R., Mujika J., Uribe J.P., *A Wall Following Robot with a Fuzzy Logic Controller Optimized by a Genetic Algorithm*, FUZZ-IEEE/IFES'95 Fuzzy Robot Competition, Yokohama, March 1995.

Brooks R.A., *A robust layered control system for a mobile robot*, IEEE Journal of Robotics and Automation, Volume 2, Number 1, 1986.

Brooks R.A., *Artificial Life and Real Robots*, Towards a Practice of Autnomous Systems: Proceedings of the First European Conference on Artificial Life, p. 3- 10 MIT Press, Cambridge, MA, 1992.

Cliff D., Harvey I., Husbands, P., *Incremental Evolution of Neural Network Architectures for Adaptive Behaviour*, CSRP256, University of Sussex Technical Report, 1992.

Cliff D., Miller G.F., *Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations*, in Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life, p. 200-218, Springer Verlag, Berlin, 1995.

Colombetti, M., Dorigo, M., *Robot Shaping: Developing Situated Agents through Learning*, Technical Report 92-040, International Computer Science Institute, Berkeley, CA, 1992.

Colombetti M., M.Dorigo & G.Borghi, *Behavior Analysis and Training: A Methodology for Behavior Engineering*, IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26, 3, p. 365-380, 1996.

Dorigo M. & M. Colombetti (1998), *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press/Bradford Books, 1998.

Marc Ebner, *Evolution of a control architecture for a mobile robot*, in Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES 98), Lausanne, Switzerland, Springer-Verlag, p. 303-310, 1998.

Floreano D., *Emergence of Nest-Based Foraging Strategies in Ecosystems of Neural Networks*, in From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior, p. 410-416, 1992.

Floreano D., Mondada F., *Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot*, in From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior, p. 421-430, MIT Press, 1994.

Floreano D., Mondada F., *Evolution of Homing Navigation in a Real Mobile Robot*, IEEE Transactions on Systems, Man, and Cybernetics, 1996.

Floreano D., Nolfi S., Mondada F., *Competitive Coevolutionary Robotics: From Theory to Practice*, in Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, MIT Press, 515-524, 1998.

Fukunaga A.S., Kahng A.B., *Improving the Performance of Evolutionary Optimization by Dynamically Scaling the Evaluation Function,* in Proc. IEEE Intl. Conf. on Evolutionary Computation, p. I-182 - I-187, 1995.

Garis, H., *Multistrategy learning in neural nets: An Incremental Approach to Genetic Programming*, in Proceedings of the Second International Workshop on Multistrategy Learning, p.138-149, Harpers Ferry, West Virginia 1993.

Gomez F., Miikkulainen R., *Incremental Evolution of Complex General Behavior*, in Adaptive Behavior 5, p. 317-342, 1997.

Harvey I., *Species Adaptation Genetic Algorithms: A Basis for a Continuing SAGA*, in Toward a Practice of Autonomous Systems: Proceedings of First European Conference on Artificial Life, MIT Press/Bradford Books, 1992.

Harvey, I., Husbands, P., Cliff, D., Thompson A., Nicobi N., *Evolutionary Robotics: the Sussex Approach*, in Robotics and Autonomous Systems, vol. 20, p. 205-224, 1997.

Holland J., *Adaptation in Natural and Artifial System*, Ann Arbor, MI: University of Michigan Press, 1975.

Juillé H., Pollack J.B., *Dynamics of Co-evolutionary Learning*, in From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, p. 526-534, 1996.

Koza J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992a.

Koza J.R., Rice J.R., *Automatic Programming of Robots using Genetic Programming*, in Proceedings of Tenth National Conference on Artificial Intelligence. Menlo Park, CA: AAAI Press / The MIT Press. p. 194-201, 1992b.

Lange S., Zeugmann T., *Refined Incremental Learning*, Proc. 8th Australian Joint Conference on Artificial Intelligence - AI'95, p. 147-154, World Scientific Publ. Co., 1995.

Lund H.H., Miglino O., *Evolving and Breeding Robots*, in Proceedings of First European Workshop on Evolutionary Robotics, Springer-Verlag, 1998.

Maes P., Brooks R.A., *Learning to Coordinate Behaviors*, AAAI-90, Boston, MA, p. 796-802, 1990.

Meeden L.A., Kumar D., *Trends in Evolutionary Robotics*, in Soft Computing for Intelligent Robotic Systems, Physica-Verlag, New York, NY, p. 215-233, 1998.

Nolfi S., Floreano D., Miglino O., Mondada F., *How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robots*, Proceedings of Artificial Life IV, p. 190-197, 1994.

Nolfi S., *Using emergent modularity to develop control system for mobile robots*, Adaptive Behavior (5), 3-4, p. 343-364, 1997.

Perkins S., Hayes G., *Robot Shaping – Principles, Methods and Architectures*, Workshop on Learning in Robots and Animals at AISB'96, University of Sussex, 1996.

Perkins S., Hayes, G., *Evolving Complex Visual Behaviours using Genetic Programming and Shaping*, 7th European Workshop on Learning Robots, Edinburgh, 1998a.

Perkins S., *Incremental Acquisition of Complex Visual Behaviour using Genetic Programming and Robot Shaping*, PhD. thesis, University of Edinburgh, 1998b.

Reynolds C.W., *Competition, Coevolution and the Game of Tag*, in Proceedings of the Fourth Workshop on Artificial Life, p. 59-69, Boston, MA, MIT Press, 1994.

Ryan C., O'Neill M., Collins J.J., *Grammatical Evolution: Evolving Programs for an Arbitrary Language*, Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming, p. 83-95, Springer-Verlag, 1998.

Sałustowicz R., Schmidhuber J., *Probabilistic Incremental Program Evolution,* in Evolutionary Computation 5(2), p. 123-141, 1997a.

Sałustowicz R., Wiering M., Schmidhuber J., *Learning Team Strategies With Multiple Policy-Sharing Agents: A Soccer Case Study,* tech.report IDSIA-29-97, 1997b.

Sałustowicz R., Schmidhuber J., *H-PIPE: Facilitating Hierarchical Program Evolution through Skip Nodes,* tech.report IDSIA-8-98, 1998a.

Sałustowicz R., Schmidhuber J., *Learning to predict Through Probabilistic Incremental Program Evolution and Automatic Task Decomposition,* tech.report IDSIA-11-98, 1998b.

Smith R.E., Cribbs III H.B., *Cooperative Versus Competitive System Elements in Coevolutionary Systems*, in From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, p. 497-505, 1996.

Sutton R.S., *Temporal credit assignment in reinforcement learning*, Ph.D. disertation, Dep. of computer and information science, University of Massachussets, Amherst, MA, 1984.

Urzelai, J., Floreano D., Dorigo M., Colombetti M., *Incremental Robot Shaping*, Connection Science Journal, Vol 10 (3 &4), p. 341-360, 1998.

Whitley L.D., Mathias K., Fitzhorn P., *Delta Coding: An Interactive Strategy for Genetic Algorithms*, in Proceedings of the Fourth International Conference on Genetic Algorithms, p. 77-84, Morgan Kauffman, 1991.

Wilson S.W., Goldberg D.E., *A Critical Review of Classifier Systems*, in Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufman, 1989.

Winkler J.F., Manjunath B.S., *Incremental Evolution in Genetic Programming*, in Proceedings of the Third Annual Conference, p. 403-411, 1998.