in: Proceedings to Seventh Scandinavian Conference on Artificial Intelligence, SCAI'01, ed. H.H.Lund, B.Mayoh, J.Perram, IOS Press 2001, p.3-12.

A Step towards Incremental On-Board Evolutionary Robotics

Pavel Petrovic

Department of Computer and Information Science,
Norwegian University of Science and Technology, 7491 Trondheim, Norway,
pavel.petrovic@idi.ntnu.no

Abstract. We apply evolutionary algorithm (EA) to the design of controller for adaptive robots. EAs can be successful for more complicated tasks, where traditional engineering methods struggle to provide a good solution. However a simple evolutionary process is computationally expensive and works only for tasks of limited complexity. Incremental evolution can provide a solution. Complex tasks are divided into easier steps. This article gives an overview of incremental evolution and explains our incremental evolution experiments. We use EA as an on-board learning method to solve the novel task encountered by the robot in its environment. A robot equipped with a simulator of itself evolves a routine to overcome an obstacle in the execution of its task. We demonstrate embedded incremental evolution of a robot program for a task with computationally limited LEGO[1] robots.

1. Introduction

A challenge of evolving a robot controller for a complex task is too difficult for a simple evolutionary algorithm. However, EAs can be used for automatic controller design for more complex behaviors, if the difficulty is decreased by dividing the evolutionary process into steps, each one easy enough to be solved. After a robot controller for a simple problem is evolved, the problem difficulty is incremented and the evolution continues. More than two steps are allowed in incremental evolution. The incremental difficulty can take several different forms:

- Environment difficulty – for example, the number of obstacles on a robot's path, presence or speed of moving objects in the environment, etc.
- Task difficulty – for example, first we might require a football playing robot to approach the ball, later we could require also to approach it from the right direction.
- Robot's abilities – a high number of robot's sensors and actuators can be too confusing for a learning algorithm. A subset of robot's equipment can be used in early steps and more specific sensors and actuators added later.

[1] LEGO is a trademark of LEGO Group.

- Controller abilities – the task for the robot might require a complex controller, for example one with an internal state. Evolution can start with a simple controller that is sufficient for initial task and the controller can be extended later during the evolution. The change can be either quantitative, i.e. incrementing the number of nodes in a neural network, or qualitative, i.e. introducing a new set of primitives for a GP-evolved program.

Incrementing the difficulty thus can be achieved by changing either the experimental setup or the fitness function in case of task difficulty or functional fitness function, see [12]. When entering a new step, already learned parts of the genotype can either remain frozen or continue to evolve. A population that converged to a solution at the end of one step might need to be reinitialized before entering another step with preserving what is already learned. Subparts of the problem can be either independent, for example individual modules that can be tested separately, or depend to some extend on other parts – 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 and parallelize the algorithm more easily. This is a general framework, but various researchers already executed more concrete and specific experiments. The following sections give an overview of previous work on incremental evolution, describe our experiment and the proposed method, and finally discuss the results and future directions.

2. Related work

This section's aim is to catch most of the current and past uses of the incremental evolution. We believe that even though it is extensive, it will provide a good starting point for studying the related work about this topic. We also avoided a possible classification of the papers since there are many different viewpoints and classification criteria.

An exercise of incrementally evolving a simple NN mapping of 4-bit binary vectors was presented in [14]. After evolving weights for a 12-node NN for 3 binary vector pairs, 4 nodes and 1 vector pair were added to the evolved networks and the evolution continued. In the incremental case (4 vector pairs and 16 nodes), the resulting solutions had lower fitness than in a non-incremental case, which was also faster. The task was easy even for a standard GA and there was high redundancy in learning after the 3 vector pairs were already learned by part of the network.

The behavior language BL for Brooks' Subsumption architecture was extended to a version suitable for EA called GEN. In [1], Brooks reasons 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.

Inman Harvey's Species Adaptation Genetic Algorithm (SAGA) [17] is a modified GA that allows genotypes with variable (growing) length. The Sussex group used SAGA in experiments with incremental evolution of NN architectures for adaptive behavior [3]. Harvey points out that contrary to a GA, which is a general problem solving optimization and search tool working with a finite search space, SAGA fits for evolving structures with arbitrary and potentially unrestricted capabilities that require genotypes with unrestricted length. In SAGA, incremental refers to the fact that the length of the genotype increments over the evolutionary run. Sussex group controllers are typically arbitrary interconnected networks with inhibitory and excitatory connections. Internal uniform noise is added to node's excitation. Inhibition is binary: once a node receives at least one inhibitory signal, it does not produce any excitatory output. Certain level of excitation sets the inhibitory output of a node. Network connections are found by the evolution. 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 learned to navigate towards a triangle in four steps: forward movement, movement towards a large target, movement towards a small target, distinguishing a triangle from a square [18].

Lund and Miglino incrementally evolved a Khepera robot with recurrent NN controller that performed a detour behavior [23]. The task for the robot was to reach a target placed behind a U shaped obstacle. They failed to evolve such a controller non-incrementally, but succeeded with two-step process. The robot was first trained for a rectangular obstacle, which was later replaced by one of a U shape.

In [10], changing environment lead to better results than a static one. Authors experiment with nest-based foraging strategies of feed-forward reactive NN controllers. An environment with a constant amount of food was compared to one with decreasing amount of food, thus making the task incrementally more difficult. Environmental change caused a drastic improvement in the quality and efficiency of the foraging strategies. In the later work at Lausanne [33], an advanced modular architecture was trained using Behavior Analysis and Training (BAT) [5, 7]. Evolution as a global search was combined with the Reinforcement Learning during the individual's lifetime. The robot learned to move around the arena as long as possible avoiding obstacles and regularly recharging batteries. Later the robot had to collect and deliver objects. New modules were added to the controller architecture and the genotype was augmented. Evolved parts of the genotype were masked so crossover and mutation operators did not affect them in the later stage.

A group at the University of Texas [15] used incremental evolution in combination with Enforced Sub-Populations (ESP) to evolve recurrent NN architectures. 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 [34] was used for reinitialization of the population based on differences from best-fit individual. Prey capture behavior of a mobile agent was evolved in 8 incremental steps. The prey, first static, was later allowed to perform several initial steps, to be finally made mobile with incrementing speed in consecutive evolutionary steps. Authors formulate 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) Making 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 the case of incrementally evolved controllers compared to the direct evolution.

Perkins and Hayes [27] argue that evolving NN controllers incrementally is too difficult. Their arguments are: networks with internal states are not 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 a behavior of a 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 analogue to the bucket-brigade algorithm from CS. In their later experiments [28] and Perkins' Ph.D. thesis [29], 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 focusing on 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 the 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 a non-shaped, although handcrafted controller was simpler and better.

A systematic study of incremental evolution in GP is in [35]. Authors distinguish between the true incremental evolution, such as in [23] 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.

Incremental evolution was used with GP in [13]. They evolved programs for 2 tasks. 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 detailed analysis of various speeds (both slower and faster than in the final task) and time moments 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. Most likely, a similar effect can be obtained also by changing other GA parameters. In the second experiment, a controller for an artificial ant following a food trail is evolved. Authors use 2 steps: in the first step, a simpler (more difficult) trail is used. Again, authors' results indicate that more difficult task as the first incremental step can speed up the evolution. In both experiments, they were able to evolve the required behaviors in a shorter time using incremental evolution compared to a non-incremental GP.

The issue of determining effective function nodes for GP was addressed in [24]. They successfully compare their method to ADF GP on a simulated incremental evolution on parity problem.

Controllers based on fuzzy rules are evolved using incremental evolution strategy in [19]. Evolution starts from a knowledge base containing single rule. Later, the rules are allowed to expand by either partitioning the domain of some input variable or by adding a linear term to the consequence part of the rule.

Currently active research on incremental evolution is done in AnimatLab [9,20]. Their SGOCE paradigm evolves tree-like programs that generate recurrent NN controllers incrementally for different versions of the problem. Good solutions to a simpler version are frozen and used to seed the initial population for harder problem, where also inter-modular connections to other parts of the controller are created. For example in [2], the group evolved a robust obstacle avoidance behavior with Khepera mobile robot in two steps, the second with a higher environmental difficulty.

Impact of combining the evolution with learning in order to maintain population diversity during incremental evolution was analyzed shortly in [8] on a binary mapping task.

Incremental evolution where several populations from the earlier evolutionary step are merged was used in [6] on the domain of the theorem proving. Neural networks were evolved to provide heuristics for constructing proofs of theorems from a simple set of axioms and two inference rules. Incremental case performed better compared to non-incremental evolution.

Researchers in the field of Evolvable Hardware are facing the problem of high complexity tasks (and thus long genotypes) as well and incremental evolution comes very handy in this domain. The advantage here is that it is relatively easier to divide the task (typically a binary function) into subtasks. This approach (divide and conquer) was suggested and later elaborated on by Torresen [32]. It was further built upon by Kalganova [21], where the incremental evolution is running in two directions: from complex system to sub-systems and from sub-systems to complex system.

Another common method used in ER for arranging the gradual increase of the task difficulty is the co-evolution [4, 30, 31, 20, 11]. 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.

3. Embedded evolutionary computation

The nature of many tasks where robots might be useful requires adaptivness, learning and dealing with unpredictable, changing and unstructured environments. It is therefore almost impossible to predict situations that the robot will have to handle during the execution of its task. Robots should be able to learn new operations and skills. Particular learning algorithms for this purpose are needed.

It's becoming a real possibility to equip mobile robots with high-performance computers on-board. Their computational power can be used for much better vision and signal processing algorithms, better planning, reasoning, processing natural language commands, etc. In addition, we propose to include a simulator of the robot itself to test robot's planned actions without the risk of failure in the real world. The simulator is used by an embedded evolutionary algorithm to evolve a good strategy or actual program code that performs a required operation. In this way, the robot can learn new skills when they become needed. A similar approach by Grefenstette and Ramsey [16] is called Anytime Learning. Our approach is slightly different, namely we evolve a program for the robot instead of a set of rules; the learning component is started on demand when the novel situation is experienced and is not running all the time in the background; we don't require real-time performance of the simulator since the robot can wait before continuing the execution of its task, although high performance of the simulator is desired as the simulation is used to compute the fitness; our model is simpler, for example it doesn't require a feedback to simulation module provided by an execution module (which in turn requires the execution module to be fairly complex since it has to know the details of the simulation module).

4. Experimental Setup

Consider a delivery task (similar to a taxi driver) for a robot placed in a grid maze. Robot starts at initial location $[x_i,y_i]$, picks the object at source location $[x_s,y_s]$ and delivers it to a destination at $[x_d,y_d]$. The plan for the maze (each grid cell is either free or contains an obstacle) and the initial location are known to the robot, which is controlled by a sequential program consisting of simple self-explanatory commands: *forward n, back n, right, left, turn180, nop, stop*. The challenge is to find a program that will safely navigate the robot to execute the task after the source and destination locations are disclosed. This can be done either manually, using a deterministic algorithm, or using some stochastic search. In our experiment, this challenge is solved using an embedded evolutionary algorithm with a simulator of the grid environment. We have chosen this task for the following reasons: it is suitable for tests with different environmental difficulty, it allows making the incremental steps in the task difficulty, it is very simple and allows a small set of primitives in the program, which can be easily represented in a genome, and its practical implementation with LEGO Robotics sets is straight-forward. A similar, single target task was addressed by work of Xiao et. al. [36], where the environment was continuous and combines off-line and on-line learning for changing environment.

Practical experiments were performed with a two-wheeled vehicle built from the parts of the LEGO Mindstorms construction set. The robot contained two motors to propel independent wheels, a third motor to load and unload cargo (fig.1. left), one light sensor to perceive the environment, and two rotation sensors to measure and compensate the built-in differences of the motors' drive (another solution we used was an adder/subtracter differential, but the rotation sensors were more precise since a lower number of loosely connected gear wheels were used).

At the start of the experiment, the robot first learned about the current situation using its light sensor (fig.1. right): the coordinates were encoded in binary (black and yellow color) in the wall constructed from 12 LEGO bricks. Next, the robot spent some time evolving a program (for 100 generations and 90 individuals in a population, evolution took ca. 3 min on LEGO RCX computer). Each program in the population was evaluated with a fitness function (see below) based on the simulated performance in a simple simulator that contained the environment map. Finally, after the simulated evolution produced a solution, the robot executed the task in the real world.

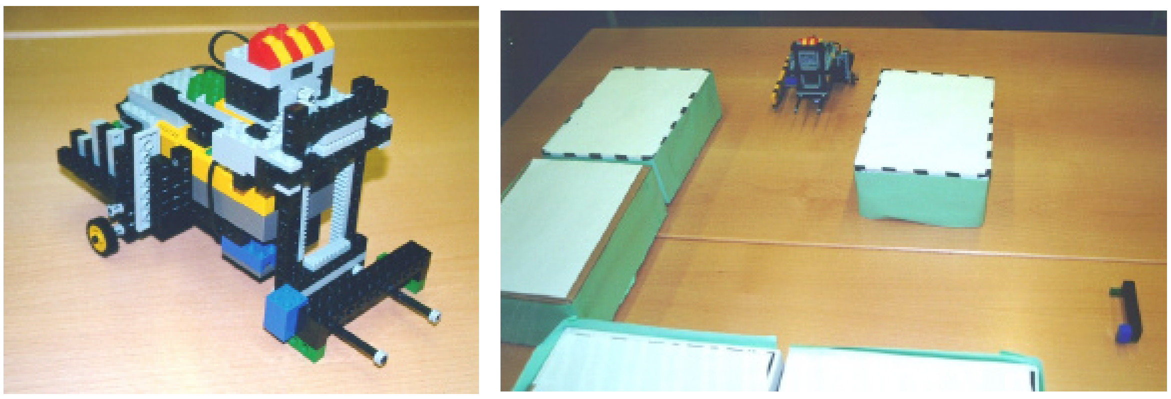

Fig. 1. Experimental robot lifting an object at source location (left) and reading the task from the colour wall (right).

The robot program was a sequence of instructions described above (the probability of *forward* instruction was double). Maximum program length was limited and a standard GA with 50% truncation selection was used (mainly because of the implementation reasons: truncation selection requires space for only 1 copy of the population in the memory). The crossover operator exchanged parts of the programs of the same length at random positions in the parent individuals. Mutation worked on a per-instruction basis: one half of mutations replaced a single instruction by any random instruction; in the remaining cases, only an instruction argument was mutated (applies only to *forward* and *back* instructions). The following fitness function was used: $f = max - 10(d_1 + d_2) - (len / 2) + a\ (s_1+s_2) - b\ hit$, where d_1 (d_2) was the length of the shortest straight vertical or horizontal line without obstacles connecting the source (destination) location with the robot, i.e. the robot could "see" the source (destination) in a distance d_1 (d_2) from some point on its trajectory, *len* was the length of the program until the *stop* instruction (shorter programs were preferred by evolution), s_1 (s_2) were 0/1 flags that were set, if the robot stopped at source (destination) location to (un)load the cargo, *hit* was the number of times that the robot hit the wall, and *a*, and *b* were weight parameters. The software was written in C using LegOS by Markus Noga [25].

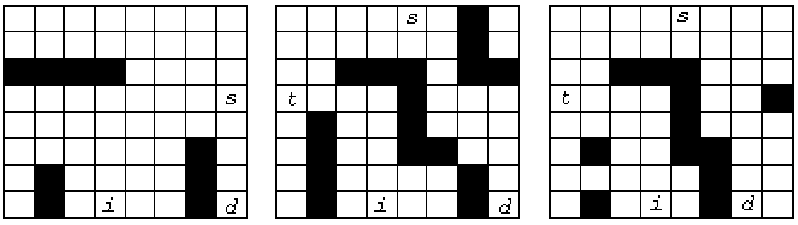

Fig.2. Robot's grid world environments. Init. position of the robot is marked with *i*, source location *s*, destination location *d*, and *t* is a temporary source location. Grid cells with obstacles are black.

5. Results

Early experiments completely in simulation were used to tune the GA parameters: crossover rate 0.75, mutation rate 0.3, population size 90, 100 generations, $a = 5$, and $b = 60$ performed satisfactorily well on simple environments with 9 obstacles (fig.2. left). The same performance, using the same program, was measured in real robot experiments (fig.1).

To work with a more interesting task, we used a more complex environment: it contained 15 obstacles and the robot had to travel double the distance and turn a few more times to find both source and destination locations (fig.2. middle). Since the fact whether the remaining experiments were performed in simulation or on a real robot was not relevant to our argument and observations, we performed them in simulation to save time.

Even with a larger population size (120) and a higher number of generations (350), the same GA found the solution only in a few cases. Therefore we temporarily moved the source location closer to a robot for γ generations in the beginning of the evolution, thus making our EA incremental. Fig.3. left shows the development of the best fitness (average from 500 runs) for a non-incremental case, γ=25 and γ=250. First of all, we can see that non-incremental evolution was not able to find a very good solution. The drop of the best fitness after moving the source location results from the fact that the programs

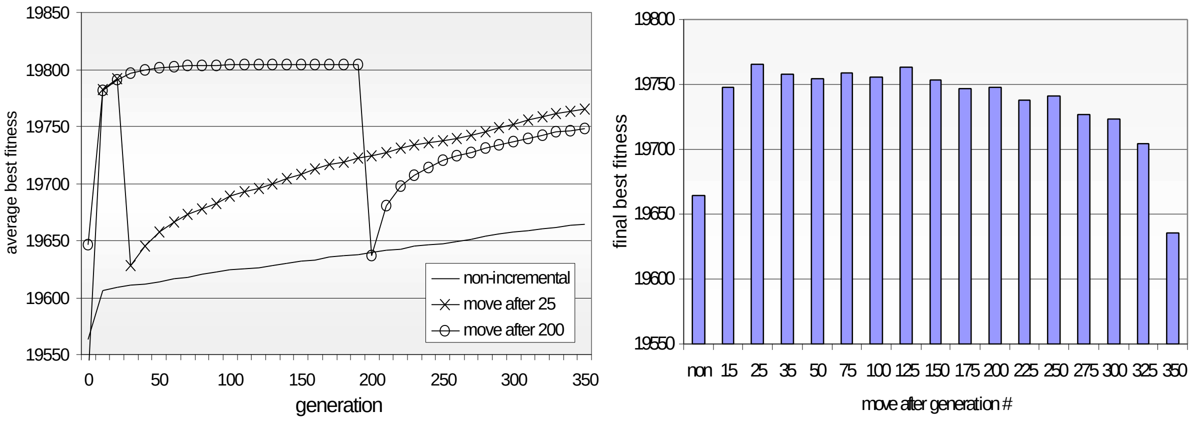

Fig.3. Influence of the moment of source shift, 2nd environment in fig 2. In case of γ=25, the standard deviations of the best fitness ($s_{25\text{-}best}$) were 15.03, 13.65, 81.10, 118.04, and 123.02 in generations 10, 25, 50, 250, and 350 respectively. The complete solution was found in 5.6% of runs and in 29.4% of runs, no points for seeing the destination were earned. In the case γ=250, $s_{250\text{-}best}$ were 15.58, 14.06, 9.31, 0.04, and 102.52 in the same respective generations. The complete solution was found in 0.4% of runs, and only the source was seen in 31.8% of runs. In the non-incremental case, the respective standard deviations of the best fitness were 28.36, 37.53, 45.35, 88.06, and 102.17, complete solution in 1.2% of runs, only the source in 69.2% of runs.

were not able to find the new source immediately, but they were not worse than programs evolved in the non-incremental case. Besides, the population contained enough genetic material to evolve quickly towards programs that could find the new source location. Further we studied the appropriate time moment for incremental change. Fig.3. right shows the final fitness for various γ (again, average over 500 runs). The final fitness was highest when the incremental change occurred just before the best fitness ceased to improve. Fig.3. left shows that if the evolution continued with incremental source longer, even though the recovery from the shift was faster, the final fitness would not exceed the γ=25 case. However, the problem was too difficult in this experiment and even 350 generations were not enough to evolve programs that could reach both the source and the destination locations. We repeated the same experiment with a simplified environment (3rd at fig.2) and the fig. 4 shows a similar result.

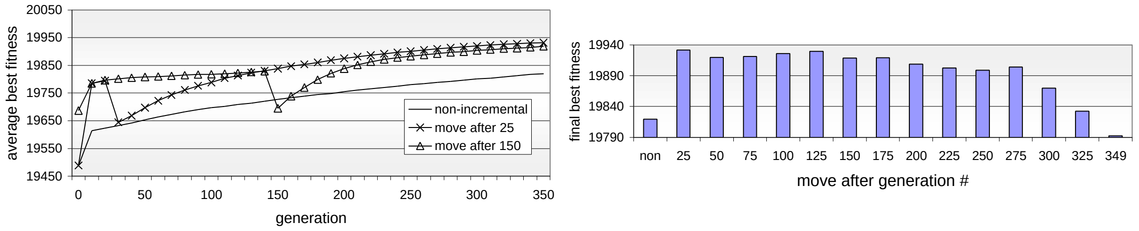

Fig.4. Influence of the moment of source shift, 3rd environment in fig. 2.

In both cases, the generation when the source was moved had to be specified in advance. This is not possible, when an adaptive robot is solving a novel problem. We therefore proposed a method to determine this generation automatically. In addition, we made the environment difficult enough (fig.5 left) so that more than one incremental change was needed. Inspired by the results from previous experiments, we measured the improvement of the best fitness. If its momentum was lower than a certain threshold and the best fitness had improved since the beginning of this stage, then the evolution progressed to another incremental stage. Precisely, we introduced 2 new parameters: discount j, and threshold q, and we measured the improvement m (initialized to some low constant) using the rule: $m_{new} = jm + (bf - pbf)$, where bf is the current best fitness and pbf is the best fitness in the previous generation. The evolution entered a new stage, when $m < q$. In other words, the evolution naturally proceeds to another stage when the learning starts to cease.

The robot found the path to execute the task after ca. 2000 generations, with 200 individuals in the population and $j = 0.9$, $q = 0.001$. Fig.5 right shows the development of the fitness for an example run. Each drop corresponds to one shift of an incremental source location. The final steep improvement corresponds to discovering the target location.

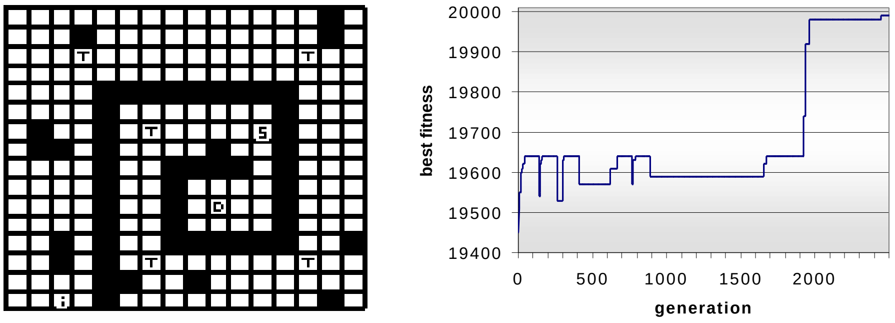

Fig.5. Environment with several incremental source locations and corresponding evolution of best fitness. Evolution found the target in 22% of 100 runs. The average generation numbers of incremental steps and their standard deviations were 132, 259, 488, 802, 1171, and 13.75, 21.25, 254.17, 328.40, 427.49.

6. Conclusion and Future directions

Plain evolutionary algorithms are not sufficient search mechanisms for finding solutions to difficult problems, such as controllers for robots working in complex environments. Providing additional guidelines to the evolution by dividing it into several incremental steps is a possible way to tackle this hurdle.
This article gives a broad overview of previous and current incremental evolution research and describes an incremental evolution experiment with an embedded evolutionary algorithm running on a simple robot built from flexible LEGO construction sets.

Our experiments show that the decision about the time (generation number) for the incremental change has an influence on the quality of the final solution. A robot facing a novel problem to solve needs to determine this generation automatically. We proposed a method that uses the information that is available about the current population. In the future work, we will focus on analyzing the nature of incremental evolution, providing a useful framework for practical users of evolutionary computation. We will study the interactive aspect of EA for design of robot controller, i.e. in which stages, how and to what degree should the user influence the evolutionary process. We will also investigate combining of the visual perceptions of mobile robots with other sensory information in incremental evolutionary algorithms.

7. Acknowledgment

The author is grateful to Dr. Keith Downing for useful discussions and comments, and corrections on the previous versions of this article.

8. References

1. Brooks R.A., Artificial Life and Real Robots, Towards a Practice of Autonomous Systems: Proc. of the First European Conf. on Artif. Life, p. 3-10, MIT Press, Cambridge, MA, 1992.
2. Chavas, J., Corne, C., Horvai, P., Kodjabachian, J. and Meyer, J.A, Incremental Evolution of Neural Controllers for Robust Obstacle-Avoidance in Khepera, in Husbands, P. and Meyer, J.A. (Eds.). Proceedings of The First European Workshop on Evolutionary Robotics - EvoRobot'98, Springer Verlag, 1998.
3. Cliff D., Harvey, I., Husbands, P., Incremental Evolution of Neural Network Architectures for Adaptive Behaviour, CSRP256, University of Sussex Technical Report, 1992.
4. 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.
5. Colombetti, M., Dorigo, M. & Borghi, G., Behavior Analysis and Training: A Methodology for Behavior Engineering, IEEE Trans. on Syst., Man, & Cyb.-Part B, 26, 3, p. 365-380, 1996.
6. Desai, N.S., Miikkulainen, R., Neuro-Evolution and Natural Deduction, in Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, 2000.
7. Dorigo, M. & Colombetti, M. (1998), Robot Shaping: An Experiment in Behavior Engineering, MIT Press/Bradford Books, 1998.
8. Eriksson, R.I., An Initial Analysis Of the Ability Of Learning To Maintain Diversity During Incremental Evolution, GECCO Workshop On Memetic Algorithms, 2000.
9. Filliat, D., Kodjabachian, J. and Meyer, J.A. Incremental Evolution of Neural Controllers for Navigation in a 6-legged Robot. In Sugisaka and Tanaka (Eds.). Proceedings of the Fourth International Symposium on Artificial Life and Robotics. Oita Univ. Press. 1999.
10. 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.

11. Floreano, D., Nolfi, S., Mondada, F., Competitive Coevolutionary Robotics: From Theory to Practice, Proc. of the 5th Int. Conf. on Simul.of Adapt. Behavior, MIT Press, 515-524, 1998.
12. Floreano, D., Urzelai, J., Evolutionary Robots with On-line Self-Organization and Behavioral Fitness, in Neural Networks, vol. 6, p. 431-443, 2000.
13. 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.
14. 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.
15. Gomez, F., Miikkulainen, R., Incremental Evolution of Complex General Behavior, in Adaptive Behavior 5, p. 317-342, 1997.
16. Grefenstette, J. J. and Ramsey, C. L. (1992). An Approach to Anytime Learning. Proc. Ninth Intl. Machine Learning Conf., San Mateo: Morgan Kaufmann, p. 189-195.
17. Harvey, I., Species Adaptation Genetic Algorithms: A Basis for a Continuing SAGA, Toward a Practice of Autonom. Syst.: Proc. of 1st Eur. Conf. on A. Life, MIT Press, 1992.
18. 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.
19. Hoffmann, F., Incremental Tuning of Fuzzy Controllers by Means of an Evolution Strategy, GP-98 Conference, 1998.
20. Juillé, H., Pollack, J.B., Dynamics of Co-evolutionary Learning, in From Animals to Animats 4: Proc. of the Fourth Int. Conf. on Simulation of Adaptive Behavior, p. 526-534, 1996.
21. Kalganova, T., Bidirectional Incremental Evolution in Evolvable Hardware. Proc. of The Second NASA/DoD Workshop on Evolvable Hardware, IEEE Press, 2000.
22. Kodjabachian, J., Corne, C. and Meyer, J.A. Evolution of a Robust Obstacle Avoidance Behavior in Khepera: A comparison of Incremental and Direct Strategies. Robotics and Autonomous Systems. In Press.
23. Lund, H.H., Miglino, O., Evolving and Breeding Robots, in Proceedings of First European Workshop on Evolutionary Robotics, Springer-Verlag, 1998.
24. Naemura, T., Hashiyama, T., and Okuma S., Module generation for genetic programming and its incremental evolution, in Charles Newton, ed., Second Asia-Pacific Conference on Simulated Evolution and Learning, 1998.
25. Noga, M., Designing the LegOS Multitasking Operating System, in Dr. Dobb's Journal, Miller Freeman, Inc., November 1999.
26. Nolfi, S., Floreano, D., Miglino, O., Mondada F., How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robots, Proc. of Artificial Life IV, p. 190-197, 1994.
27. Perkins, S., Hayes, G., Robot Shaping – Principles, Methods and Architectures, Workshop on Learning in Robots and Animals at AISB'96, University of Sussex, 1996.
28. Perkins, S., Hayes, G., Evolving Complex Visual Behaviours using Genetic Programming and Shaping, 7th European Workshop on Learning Robots, Edinburgh, 1998a.
29. Perkins,S., Incremental Acquisition of Complex Visual Behaviour using Genetic Programming and Robot Shaping, PhD. thesis, University of Edinburgh, 1998b.
30. 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.
31. 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.
32. Torresen, J., Increased complexity evolution applied to evolvable hardware. In Smart Engineering System Design, ANNIE, 1999.
33. Urzelai, J., Floreano, D., Dorigo, M., Colombetti, M., Incremental Robot Shaping, Connection Science Journal, Vol 10 (3 &4), p. 341-360, 1998.
34. Whitley, L.D., Mathias, K., Fitzhorn, P., Delta Coding: An Interactive Strategy for Genetic Algorithms, in Proc.of the 4th Int. Conf. on Genetic Alg., p. 77-84, Morgan Kauffman, 1991.
35. Winkler, J.F., Manjunath, B.S., Incremental Evolution in Genetic Programming, in Proceedings of the Third Annual Conference, p. 403-411, 1998.
36. Xiao, J., Michalewicz, Z., Zhang L., Trojanowski K., Adaptive Evolutionary Planner/Navigator for Mobile Robots, in IEEE Transactions on Evolutionary Computation, vol.1, no.1, 1997.