Evolution of automatic robot control with genetic programming

Lovro Paić-Antunović, Domagoj Jakobović

***Abstract*—This paper investigates the problem of automatic control of a robot model in an arbitrary two-dimensional environment. The robot control is based on behavior evolution with the use of genetic programming (GP). A simulation environment is developed which allows automatic synthesis of different behaviors suited to the environment and user requirements. The influence that different robot properties and learning mechanisms have on the evolved behavior are studied in depth. The results show that the presented approach is able to produce effective control procedures.**

I. Introduction

Autonomous mobile robots are becoming more and more important and available to a widening range of uses - from autonomous vehicles to household appliances. They are expected to solve tasks that humans are not able to cope with in real-time conditions, or at least to simplify the tasks humans perform on a daily basis. This requires extensively autonomous robots, because with growing complexity of the problems it will be no longer possible for the programmer to take all eventualities into account from the outset. Developing robust control programs is an active research issue in many environments.

Lately, this development has also been performed as an evolution process, where appropriate logic is automatically generated with the help of evolutionary algorithms [1], [2], [3], [4]. This paper is an example of this approach, where genetic programming (GP) is used to evolve control algorithms for the desired robot behavior. The main advantage of the presented method is its reusability for different target functionalities of the evolved control program - e.g. finding a route, mapping an unknown environment etc.

In addition to the approaches previously described in the literature, this paper briefly investigates the generalization ability of the method and suggests the means to obtain it. We also include memory capabilities to the control program and show the advantages it may have to the development of target behaviors in a complex environment. Finally, the performance of the algorithm and quality of the obtained solutions is improved with the algorithm parallelization in a distributed manner.

To evolve the control algorithm, we use simulated robots with sensors with arbitrary placement. Simulating robots allows investigation of variable architectures and rapid prototyping, which is, compared to experiments with real hardware, less expensive. Additionally, simulation process is fast and easily reproducible.

The remainder of this paper is organized as follows: Section II states the problem and simulation environment, as well as related work on the subject. Section III describes the genetic programming and related evolution of the control algorithm; Section IV presents experimental results and efficiency analysis, and short conclusions are given in Section V.

II. Automatic Control of a Model Robot

The problem of automatic control design may be formulated with various assumptions; one of the most distinctive features is the knowledge of the environment. The environment may be predefined and given as an input to the robot, i.e. we may have the information about the topology which may be used by the control algorithm. On the other hand, the autonomous agent may have to discover the topology of the environment, and any previous information about it is unavailable. In this work we concentrate on the latter case, where the aim is to develop a desired behavior in an unknown surroundings.

The model of the simulated robot is mostly equivalent to the model from [5], where a wall following behavior was evolved. The robot has 12 sensors, equally distributed over 360 degrees, which report the distance to the nearest wall as a floating point value. These sensors are the only direct information the robot is given about its environment. As the robot moves, the sensor data changes accordingly, depending on the room topology.

The robot's ability to move is implemented with the following possible actions:

- turn left 30 degrees;
- turn tight 30 degrees;
- move forward by a constant distance and
- move backward by a constant distance.

The control program has to generate one of these outputs as a result to any combination of current robot state and sensor input values. After the defined action is performed, new input values are collected and another step may be taken. Finally, the robot has the option to stop any movement if a certain condition is met (i.e. a target position is reached or a time interval has elapsed).

In the simulation process, the robot is simulated within an appropriate software framework in discrete time steps. In each time step, input sensor values are calculated and fed to the control program, which decides upon an action. The action is then simulated by updating robot position or orientation and the process is repeated. The program is presumed to produce one action at a time, as opposed to a series of actions.

Using a different program, various target behaviors may be developed. The control program itself may be human-made or previously evolved with genetic programming, which is detailed in the next section.

III. Genetic Programming Applied to Behavior Modeling

A. Genetic programming

Genetic programming [5] is an optimization and machine learning technique that uses evolutionary concept to automatically discover symbolic procedures (functions, programs) to the problem at hand. The main idea behind GP is that the solution to the problem may be represented as a (computer) program, in most applications in the form of a tree (which allows mapping to any procedural language). The elements of the programs (tree nodes) must be predefined by the user and must be sufficient to describe the solution to the problem (e.g. mathematical and logical functions, actions such as move forward, turn left etc). The algorithm randomly generates programs (potential solutions) and evaluates each program on a predefined set of test cases (e.g. how well does the program control a robot). Each potential solution thus receives its quality estimate - the fitness value - which is then used in the selection process.

The selection process imitates natural evolution where weaker individuals (solutions) are eliminated, and better individuals survive. Additionally, better individuals also participate in recombination, where two (or more) individuals are combined to form a new solution. The algorithm also incorporates a mutation mechanism, where a single individual is subject to a change, with a relatively small probability. The process continues, building new generations from old ones, until a suitable termination criterion is reached. These criteria usually include finding a solution of the desired quality or performing the algorithm for a predefined amount of time. The examples of human-competitive results of genetic programming may be found in [6].

B. Evolving a robot with GP

In this work the control program is represented as a tree, where each node may server as an input, perform a certain logical function or a certain action. There are two main types of tree nodes which every generated program contains - terminal and function nodes. Terminal nodes are the leaves of tree program structure; they don't have children nodes and they return certain value which may be used by their parent function node.

Terminal nodes in this application can be one of the following:

- 12 different nodes (S01, S02, ..., S11) that return the distance from one of 12 sensors surrounding simulated robot;
- a node which returns dynamically computed shortest distance among all the sensors (SS);
- nodes which perform one of four basic motor functions (MF - move forward, MB - move backward, TR - turn right, TL- turn left), in which case they also return the shortest distance from three front sensors.

Function nodes are the inner nodes of a tree and have at least one child node. In our implementation each function node controls the program flow in a specific manner. Function PROGN2 is the simplest function node which is used for connecting parts of program together. It evaluates both of its two subtrees in sequence, and returns the value from the second one. Function IFLTE stands for 'If Less Than or Equal' and has a total of four child nodes. It is used for deciding on a behavior based on results from the first two of its nodes. If the value received after traversing the first subtree is less than or equal to the value received from its second subtree, the third subtree will be traversed and its value returned, and if not, the fourth subtree will be traversed and its value returned.

In this work we also include the optional memory functions, denoted IMEM1 and IMEM2, which have the ability of keeping the track of the internal state of the robot. Function IMEM1 requires that the robot keeps track of coordinates of walls he was already close to in his run. The function then controls the program flow by checking the table of visited walls; if there exists a wall which is directly in front of the robot, it evaluates and returns the first subtree. If such a wall doesn't exist, it evaluates and returns the second subtree value.

Function IMEM2 has three child nodes, and it requires remembering the history and location of previous robot actions. It decides which subtree will be evaluated by checking if the table of all actions for the current position contains the action from the first child node. If it does, it evaluates and returns value from the second subtree, and if it doesn't, the third tree's value is evaluated and returned.

The inclusion of memory functions may serve the purpose of completing the tasks whose solution cannot be represented with a stateless program. An example of this problem includes the task of visiting the walls in an unknown room which has inner walls that are not connected to any of the outer walls (see Fig. 2). This kind of control problem was successfully solved with the help of these functions.

An example of an evolved control program comprising of the described nodes is given in Fig. 1.

In this work we used two different target behaviors. One goal is visiting all walls in an arbitrary unknown room, and the other is traversing the whole room surface. The examples of both evolved behaviors are given in Fig. 2 and 3.

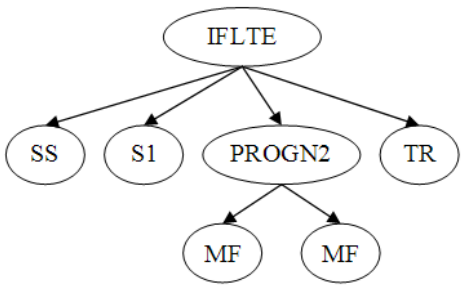

Figure 1. An example control program (genetic programming tree)

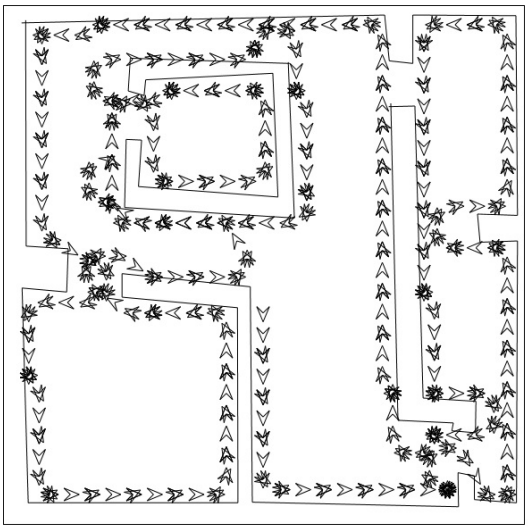

Figure 2. An example of an evolved behavior: following walls in unknown room

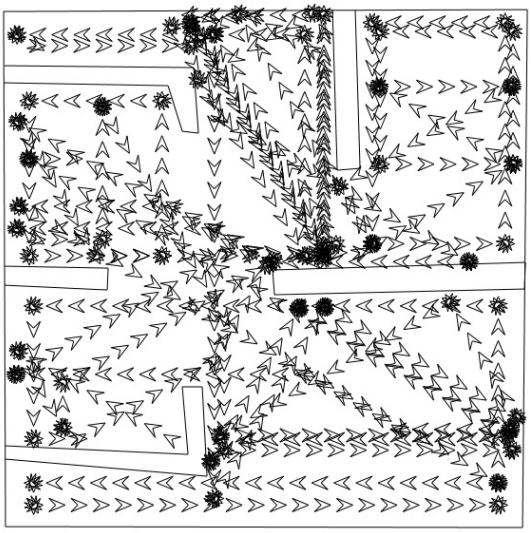

Figure 3. An example of an evolved behavior: covering an unknown room

C. Evaluating a control program

During the evolution process, a great number of candidate solutions (control programs) is created that have to be evaluated. Evaluation is performed by simulating the robot movements in a given environment using the selected program. Each solution (individual) is given a fitness value that is used in a selection process and represents the individuals ability to compete with other individuals.

In both of the goals used in this work it is required that the robot drives over a set of targeted positions. Because of this property, the fitness function value is defined by the number of visited discrete *target locations* (each location is counted only the first time that the robot moves over it). The target locations in the simulation process are set in a uniform grid over the room surface or along the walls. The fitness value of a control program is thus represented as the total number of visited target locations. Although the number of target locations depends on the desired behavior, room shape and the number of rooms used in the learning, all the scores in the results are scaled to a maximum value of 100.

Additionally, each action a robot performs (a single movement or a single rotation) is presumed to take a constant amount of time, and is represented as a single simulation *time step*. If two different evolved programs visit the same number of target locations, the one which has visited them in less time steps is considered better.

It is important to note that, to evolve one or the other behavior, no modifications are needed in the GP evolutionary system other than the appropriate calculation of the fitness function. In other words, to evolve a different (custom) behavior, only the fitness function needs to be altered, whereas the functional and terminal tree elements may remain the same.

IV. Results and Analysis

The following experiments aim to evaluate the GP efficiency in evolution of the target robot behavior. The following parameters were used in all experiments, unless stated otherwise:

- population size of 1000 individuals;
- termination condition of 500 generations;
- at least 10 repetitions for all configurations.

A. Initial tree depth parameter

The purpose of this experiment is to empirically estimate the appropriate initial tree depth that any solution may attain. This parameter is one of the most influential ones in genetic programming [5]. On one hand, it should be large enough to allow the creation of the programs of needed complexity, i.e. the ones that can represent the solution to the problem. On the other hand, it should not be too large so the search space is appropriately limited and so that the complexity is maintained at the lowest possible level.

The experiment was conducted by learning on 10 manually selected rooms and summing the fitness function of the same program on all the rooms. The convergence results are presented as the average score of best individuals in all the repetitions (i.e. average fitness value of best individuals at a certain generation across all repetitions). The convergence rates are shown in Fig. 4, and the corresponding average number of time steps in Fig. 5. The actual values for best individuals at the end of runs are given in Table I.

While the solutions with initial depth of 3 are clearly worse than the others, the results show that there is no significant statistical difference between solutions with depth of 4-7. On the other hand, the average size (number of nodes) of solutions with initial depths greater than 4 was several thousand, whereas with depth 4 it averaged on a few hundred at most. For these reasons, the initial depth of 4 was chosen for the remaining experiments.

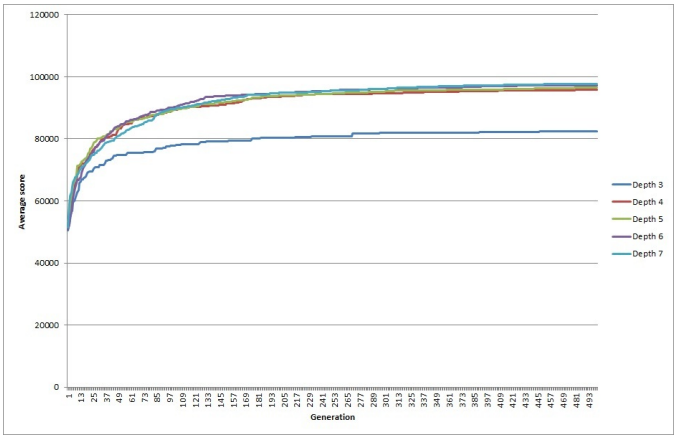

Figure 4. Average score for different initial tree depth

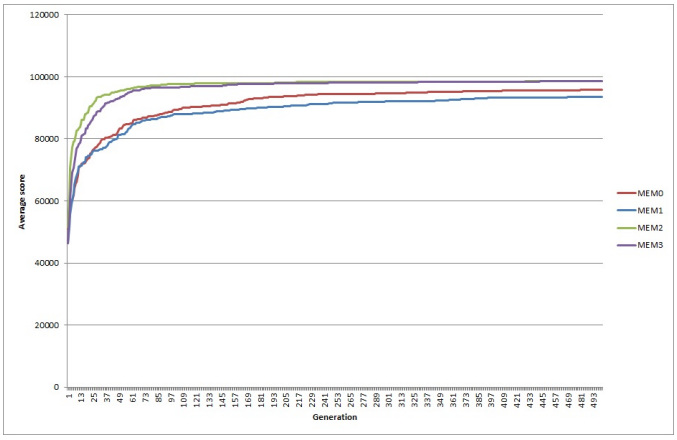

Figure 6. Average score for different memory configurations

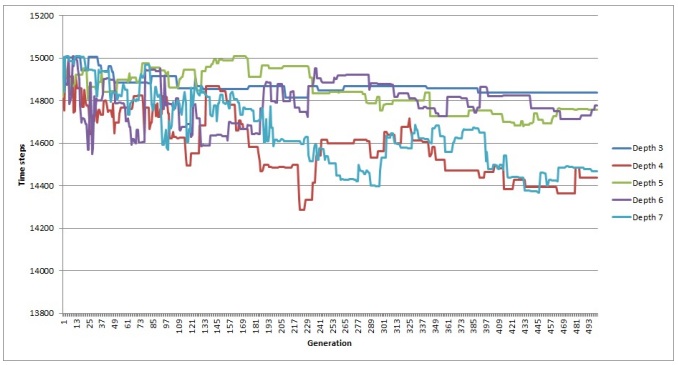

Figure 5. Average time steps for different initial tree depth

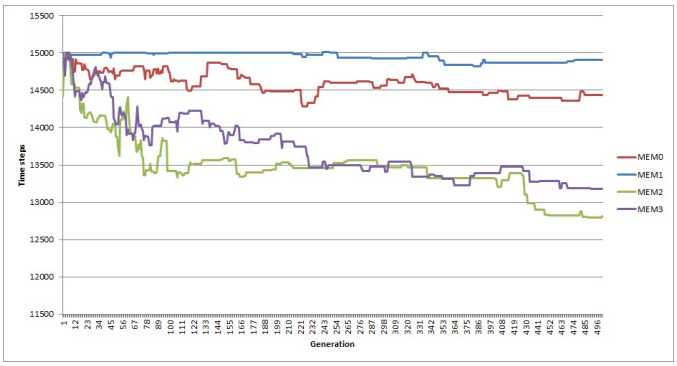

Figure 7. Average time steps for different memory configurations

B. The influence of memory functions

In this experiment we tried to estimate the influence of implemented memory functions (IMEM1, IMEM2, Sec. III) on the efficiency of the evolved control programs. A total of 4 configurations are investigated with learning on 10 different rooms as in the previous experiment, with the following notation:

- MEM0: memory functions are not used;
- MEM1: function IMEM1 is included in the function set;
- MEM2: function IMEM2 is included in the function set;
- MEM3: both functions IMEM1 and IMEM2 are in the function set.

The convergence results for these configurations are given in Fig. 6, and the corresponding average time steps in Fig. 7. Table II shows the fitness values of best individuals and their distribution at the end of the runs. The results indicate that the inclusion of IMEM2 function is beneficial to the evolution of control programs, while the IMEM1 function can even degrade the average solution quality. This suggests that, while it may complicate the development and implementation, a memory function such as IMEM2 is highly desirable in achieving complex behaviors.

C. Single-population and multiple-population GP

Since the evolution process may be quite time-consuming, and it should also be repeated a number of times to obtain the best possible result, a common extension to many evolutionary systems is their parallelization [7], [8]. In this work we chose to implement a distributed algorithm, which may be deployed on a network of computing nodes, as well as on a single multi-core machine [9].

Apart from the speedup, the parallelization may also result with different algorithm properties, such as the improved convergence rate or probability of finding the global optimum.

Table I
FITNESS VALUES OF BEST INDIVIDUALS FOR DIFFERENT INITIAL TREE DEPTH

Depth	3	4	5	6	7
Min	74.8	91.2	94.2	95.4	95.4
Average	82.5	95.8	96.5	97.3	97.8
Max	89.8	97.7	98.2	99.0	99.1

Table II
FITNESS VALUES OF BEST INDIVIDUALS FOR DIFFERENT MEMORY CONFIGURATIONS

Mode	MEM0	MEM1	MEM2	MEM3
Min	91.2	89.2	97.8	97.9
Average	95.8	93.6	98.7	98.6
Max	97.7	96.8	99.3	99.3

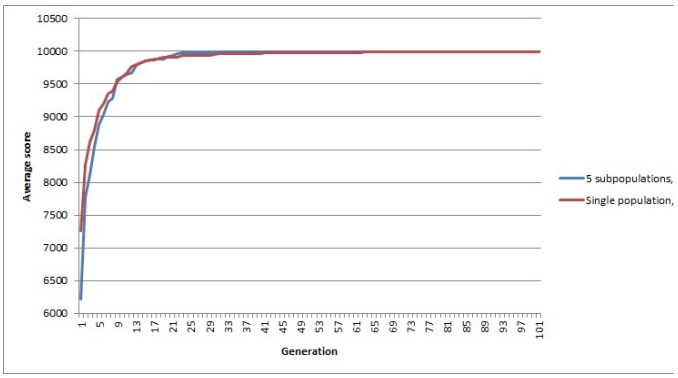

Figure 8. Average score for sequential and distributed algorithm

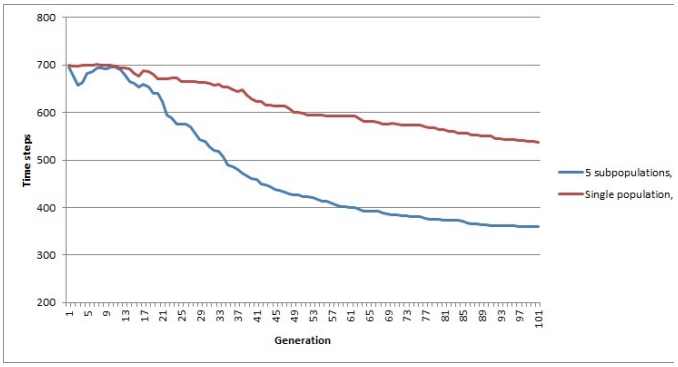

Figure 9. Average time steps for sequential and distributed algorithm

In this experiment we compared the following configurations:

- single population (single island), 2000 individuals;
- 5 subpopulations (islands), 400 individuals each.

In the distributed version, the migration interval is 5 generations, the number of migrants is 10% of the island population, and the migration pattern is a fully connected graph. The convergence results and number of time steps are shown in Fig. 8 and Fig. 9. The difference in fitness values are negligible in this experiment, since both variants exhibited similar average score. On the other hand, the solutions evolved in the distributed version achieved better results in terms of total number of time steps, which is shown in more detail in Table III. This indicates that the distributed version may have the potential to generate better solutions in terms of program efficiency, while maintaining the same fitness value level.

D. GP generalization ability

The last experiment considers the quality of the evolved solutions when used in an unseen environment. In the learning process, the control programs are evolved using the simulation of the robot in one or possibly more different rooms. After the control program is evolved, it should be tested in a previously unseen environment to asses its generality. The problem of GP generalization ability has been studied before [10] and the findings suggest that including a number of test cases (e.g. simulated rooms) may be beneficial in the search for more general solutions.

Table III
TOTAL TIME STEPS FOR BEST INDIVIDUALS, SEQUENTIAL AND DISTRIBUTED ALGORITHM

	5 islands, 400 individuals each	1 island, 2000 individuals
Min	220	292
Average	359.55	536.95
Max	700	701

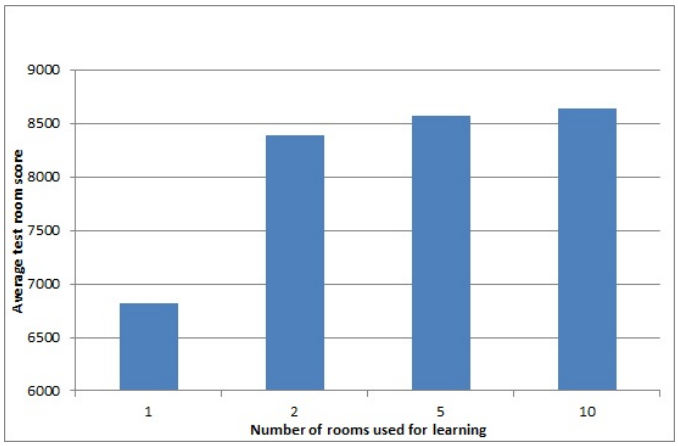

Figure 10. Average score for different number of learning rooms

Table IV
FITNESS VALUES OF BEST INDIVIDUALS, DIFFERENT NUMBER OF LEARNING ROOMS

	1 room	2 rooms	5 rooms	10 rooms
Min	6.85	3.85	45.6	16.8
Average	68.2	83.9	85.7	86.4
Max	100	100	100	100

In this experiment a total of 60 rooms were available in the learning process, and in each configuration a different number of randomly selected rooms is used in every algorithm run. The number of learning rooms assumed values of 1, 2, 5, and 10. After the learning process, following the standard machine learning paradigm, the best solution from every run is evaluated on a single room that was unavailable in the evolution process. The results in terms of average fitness are shown in Fig. 10 and in terms of total time steps in Fig. 11.

The presented results clearly indicate that the increased number of learning rooms enables the control program to behave better in an unseen environment. Note, however, that this does not guarantee good behavior in every instance, since the generalization ability depends greatly on the features of rooms that are used in the learning process.

V. CONCLUSIONS

This paper presents the use of genetic programming in the evolution of control programs for a robot in a simulated environment. The results show that this method is able to produce control programs that satisfy the functional requirements of robot behavior. Additional results may be summarized as follows:

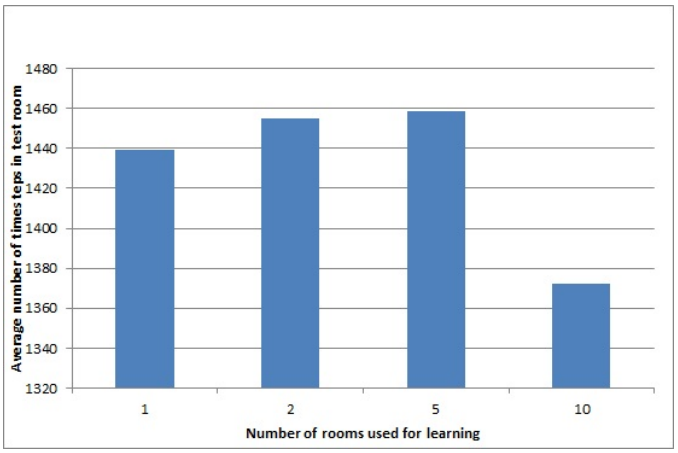

Figure 11. Average time steps for different number of learning rooms

- the memory capabilities of the control programs may be beneficial and are necessary for certain complex behaviors;
- distributed genetic programming may produce results of better quality;
- a larger number of learning instances is necessary for successful generalization.

Further work in this area may include the implementation of this approach for a real robot with actual sensor inputs. In this paper only the simulation results are reported, but in a real-world scenario the input noise must also be taken into account, and its influence on the solution quality must be evaluated.

REFERENCES

[1] C. Lazarus and H. Hu, "Using genetic programming to evolve robot behaviours," *Nature*, no. April, 2001.

[2] P. Nordin and W. Banzhaf, "An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming," *Adaptive Behavior*, vol. 5, no. 2, pp. 107–140, 1997.

[3] ——, "Real time control of a khepera robot using genetic programming," *CYBERNETICS AND CONTROL*, vol. 26, pp. 533–561, 1997.

[4] J. Busch, J. Ziegler, C. Aue, A. Ross, D. Sawitzki, and W. Banzhaf, "Automatic generation of control programs for walking robots using genetic programming," in *Proceedings of the 5th European Conference on Genetic Programming*, ser. EuroGP '02. London, UK: Springer-Verlag, 2002, pp. 258–267. [Online]. Available: http://dl.acm.org/citation.cfm?id=646810.704243

[5] J. R. Koza, "Genetic programming - on the programming of computers by means of natural selection."

[6] ——, "Human-competitive results produced by genetic programming," *Genetic Programming and Evolvable Machines*, vol. 11, pp. 251–284, September 2010. [Online]. Available: http://dx.doi.org/10.1007/s10710-010-9112-3

[7] E. Cantú-Paz, "Efficient and accurate parallel genetic algorithms," 2001.

[8] T. Alba, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, 1999.

[9] E. Cantú-Paz, "Topologies, migration rates, and multi-population parallel genetic algorithms," in *Banzhaf et al., eds, Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 1, pp. 91–98.

[10] I. Kushchu, "Genetic programming and evolutionary generalization," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 5, pp. 431 – 442, oct 2002.