To appear in *Genetic Programming 1996: Proceedings of the First Annual Conference*, July 28–31, 1996, Stanford University.
J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (eds.) Cambridge: The MIT Press

Variations in Evolution of Subsumption Architectures Using Genetic Programming: The Wall Following Robot Revisited

Steven J. Ross, Jason M. Daida, Chau M. Doan, Tommaso F. Bersano-Begey, and Jeffrey J. McClain

The University of Michigan
Artificial Intelligence Laboratory and Space Physics Research Laboratory
2455 Hayward Avenue Ann Arbor, Michigan 48109-2143
stevross@engin.umich.edu, daida@eecs.umich.edu, doan@engin.umich.edu, tombb@engin.umich.edu, jeffjm@engin.umich.edu

ABSTRACT

The wall following robot is examined as a potential benchmark problem for applications of genetic programming (GP) to emergent robotic behavior. This paper describes experiments that were performed to characterize the performance, solution space, search space, and robustness using GP with and without automatically defined functions (ADFs). GP with ADFs was unable to significantly outperform GP without ADFs on this problem. A sub-optimal modality was discovered across all four program architectures. Many of the optimal solutions that were discovered tended to limit the number of sensors used for wall-following behavior; some used as few as three sensors. Tests for robustness indicate a "handedness" to the evolved solutions, which does seem to contribute to solution brittleness.

1. Introduction

1.1 Background

Several key works have demonstrated that GP shows promise as a means for behavior-based robot programming. The wall following robot, box pushing robot, lawn mower problem, and artificial ant are a few of the several examples presented by [Koza 1992, 1994b]. [Reynolds 1994a, 1994b] demonstrated that GP can evolve solutions for obstacle avoidance and corridor following behavior. [Handley 1994] demonstrated that an ADF GP can evolve a robotic planner that operates on predicates. Recent work by [Nordin 1995] demonstrated that a compiling GP system can evolve robust real-time obstacle avoidance behavior for real robots in noisy physical environments.

However, because many of these previous efforts have focused on demonstrating the solutions GP is capable of evolving, little is known about the overall nature of many of these problems. Subsequently, only a few suitable benchmarks exist as a basis for measuring gains by a particular GP methodology for behavior based or animat programming. Consequently, the goal of this work is to further characterize the problem of the wall following robot. Our eventual goal is to craft a suitable benchmark for measuring performance of new GP methodologies that may apply to problems involving emergent robotic behavior.

This work is divided into several sections. Sections 1.2 to 1.3 discuss considerations of the wall following robot as a benchmark. Section 2 presents a brief summary of the wall following robot problem and describes the experiments performed in this work. Section 3 presents and discusses the results in terms of overall performance, possible modalities in the solution space, observations on the search, and robustness of evolved solutions. Section 4 highlights a few conclusions from this paper.

1.2 Benchmark considerations

The wall following robot is particularly attractive as a benchmark because it is a well-known problem in GP [Koza 1992a], as well as in robotics [Brooks 1992]. With respect to GP, the problem is rich, as a number of different solutions are possible with only a few general functions and terminals. The problem also has solutions that are easily identifiable and has behaviors that are understandable to a lay audience. Furthermore, the problem is attractive because of previous work. [Koza 1992b] has demonstrated that GP can evolve a successful solution to this problem, provided code for the successful individual, and described behaviors seen in the evolution of the successful run. [Koza 1992a] adds to this work by demonstrating that the problem can be solved with an even more general implementation of the terminal set. [Koza 1994b] provides a performance curve for an experiment using the [Koza 1992b] terminal set.

With respect to robotics, this problem is also attractive because it has been examined as a case study in subsumption architectures [Brooks 1986, 1992, Mataric 1990]. The problem contains great potential for evolutionary robotic programming since the problem's functions and terminals have a close analog to realizable hardware components. For this reason, the problem avails itself to pragmatic concerns such as the effect of damage to any of the sensors or motors on an evolved robot program. Other

similar problems like the lawn mower problem and its variants [Koza 1994a] have been well characterized, but have used component abstractions that are not easily realized in hardware. The lack of sensors and types of motor functions used in these problems link them tightly to the realm of animat problems like the artificial ant [Jefferson et al. 1992]. These types of idealizations aid in the discovery of fundamental principles but constrain them to the realm of computational life rather than as a means of evolving controls for physically embodied robots [Brooks 1992].

Consequently, the wall following robot is a reasonable marriage between problems in genetic programming and robotics and is suitable for studying aspects of evolutionary robotic programming from the perspectives of both domains. Furthermore, the physical representation of the robot and the introduction of ADFs into GP provide a means to study hypothesis regarding the use of GP techniques to build behavior-based programs to run real robots. [Brooks 1992] proposes that for GP to be an effective tool for programming real robots, the language that GP is operating on should include a macro capability. Brooks hypothesizes that the macro capability of the language should be effective if the macros can capture symmetries inherent to programming the robot and should greatly accelerate the production of programs and traversal of the search space. From the GP perspective, the wall following robot is a difficult problem and studying it may provide further insight into fundamental techniques needed for scaling GP and identifying break points in this problem.

For the wall following robot to be considered as a benchmark, several elements must be added to the knowledge of the problem. An independent verification of the results is necessary. Additional characterization is needed regarding the solution space, search space, average case performance of GP, and robustness of solutions. ADFs should be applied to test their effectiveness.

1.3 Contributions

This paper:

- Provides an independent verification of a well-known problem in GP.
- Characterizes several modalities of this problem's solution space to help qualify the problem for future use as a GP benchmark.
- Provides a limited test of Brooks' macro-language hypotheses in behavioral-based robot programming by applying ADFs.
- Gives a first assessment on robustness of evolved solutions for this problem.

2. Overview of Experiments

This section starts with a brief description of the problem and details of our problem specific code. The choice of ADF architectures and tests of robustness used in the experiments are also discussed. It includes a description of the parameters used and implementation notes.

2.1 Code description

Our implementation of the robot follows that which is described in [Koza 1994b, 1992a, 1992b]. The TOTO robot has 12 sensors, each covering an area of 30°. The robot can move forward 1.0 ft, backward 1.33 ft, turn left 30°, and turn right 30°. Koza's room has an irregular shape with protrusions on the south and east walls. The objective of the wall following robot problem is to evolve a computer program that allows a simulated TOTO robot to move along the perimeter of Koza's room.

Table 1 presents the key elements of our problem specific code used in the experiments. Note that the derived

Table 1: Wall following robot problem specific code

Terminal Set	Twelve sonar measurements `[S00, S01..S11]`; Derived minimum of measurements `[SS]`; minimum safe distance and preferred edging distance from wall `[MSD, EDG]`; Primitive motor functions `[MF, MB, TR, TL]`
Function Set	If-Less-Than-Or-Equal-To macro `[IFLTE(arg1,arg2,arg3,arg4)]` (i.e., IF(arg1 <= arg2) then arg3, else arg4); Connective function `[PROGN2(arg1, arg2]]` (i.e., eval arg1 return eval arg2)
Fitness Function	Fitness cases: one fitness case Koza's irregular room with an initial starting location near the middle of the room(13.8, 13.8)[1] and an initial facing direction of south(270°) Hit: robot touches a 2.3 square foot tile along the wall of the room Raw fitness: number hits in 400 time steps Standardized fitness: total number of wall tiles minus the number of hits Success predicate: 56 out of 56 hits

Table 2: Program Architectures

Experiment I	Non-ADF: Terminal set includes all 19 terminals described in Table 1 Function set includes both functions described in Table 1
Experiment II	One result producing branch, and two function-defining branches(`ADF0`, `ADF1`) Function-defining branches require no arguments and can only be called by the result-producing branch `ADF0`, `ADF1`, and `RPB` include all functions and terminals described in Table 1
Experiment III	One result-producing branch, and two function-defining branches(`ADF0`, `ADF1`) Function-defining branches require no arguments and can only be called by the result-producing branch `ADF0`, `ADF1`, and `RPB` include all functions and terminals described in Table 1 with these notable exceptions: • `MF`, `MB` are omitted from terminal set of `ADF0` • `TR`, `TL` are omitted from the terminal set of `ADF1`
Experiment IV	One result-producing branch, and four function-defining branches(`ADF0`, `ADF1`, `ADF2`, `ADF3`) Function-defining branches require no arguments and can only be called by the result-producing branch `ADF0`, `ADF1`, `ADF2`, `ADF3`, and `RPB` include all functions and terminals described in Table 1

value SS was included in the terminal set and movement and turning functions return the minimum of the two forward looking sensors as described in [Koza 1992a] Example I.

2.2 Experiments

Experiment I was performed without ADFs to verify our results against Koza's. Experiments II, III, and IV used ADFs for two reasons. First, ADFs are a logical extension of Koza's work. Second, they provide a test of Brooks' hypothesis [Brooks 1992] that a macro capability in the language of GP would greatly accelerate the production of good programs on problems in emergent robotics. Choices regarding ADF architectures were made using the method of prospective analysis [Koza 1994a] since no previous work suggests an optimal architecture for this problem.

A choice common to all ADF architectures was that all function-defining branches would take no arguments and could only be called by the result-producing branch. We chose this type of hierarchy for the ADF architectures because Mataric's subsumption architecture used mutually exclusive behaviors [Mataric 1990]. Experiment II used an ADF architecture with two function-defining branches because it was the maximum default supported by the ADF kernel found in [Koza 1994a]. Experiment III used an ADF architecture with two function-defining branches whose terminal sets were restricted to try to coax GP into using ADFs more effectively. (Preliminary results indicated that many of the evolved individuals were not taking advantage of ADFs in the search.) Experiment IV used an architecture with four function-defining branches to see if GP would evolve a solution similar to Mataric's that relied on four behaviors [Mataric 1990]. Table 2 summarizes the program architectures used in each experiment.

2.3 Robustness

The robustness of each successful individual was tested by evaluating the code produced over additional fitness cases not used in the evolution of the solution. One additional fitness case consisted of moving the initial starting position of the robot eight feet to the west. The other case consisted of changing the initial direction of the robot from facing south to facing north.

2.4 Run parameters

All four experiments used the same parameters as a means of control. The parameters used were identical to those found in [Koza 1992a, 1994b]: 10% probability of reproduction, 90% probability of crossover, maximum depth of six for new individuals, ramped-half-and-half method of generation for individuals in the initial population, a maximum depth of 17 for individuals after crossover, and fitness-proportionate-with-overselection selection method. Each experiment consisted of 61 runs[2] with a population size of 1000 for a maximum of 57 generations.

2.5 Implementation notes

Runs were performed concurrently in a distributed computing environment, consisting mostly of Sun SparcStation 20 and HP 715 workstations using Allegro Common Lisp. Runs averaged from about 48 - 100 hours[3]. We used a lookup table for sensor values and terminated the execution of programs when the state of the robot stabilized as recommended in [Koza 1992a]. The "on-the-fly" redimensioning technique was not used.[4] Experiment I was performed with the canonical GP kernel found in [Koza 1992a]. Experiments II, III, and IV were performed with the ADF GP kernel found in [Koza 1994a] and a variant to support the use of architectures with four function-defining branches.[5] *Mathematica* was used in creating the lookup table of sensor values and in visualizing each of the solutions.

3. Results and discussion

This section discusses the results of our experiments in regards to performance of GP, modalities in the solution space, observations on the search space, and robustness of solutions.

3.1 Performance

Table 3 presents results that show the overall performance of the program architectures used in the four experiments. Table 4 presents a detailed breakdown of the successful runs in Table 3.

No experiment produced enough successful runs to generate meaningful performance curves. However, the success rate for Experiment I was consistent with the performance curve shown in [Koza 1994b].[6] Furthermore, the successful solutions were comparable to the best individuals presented in [Koza 1994b, 1992a, 1992b] as they contained a similar number of program nodes, used a similar number of time steps, and exhibited the same types of behavior. Consequently, the independently produced problem specific code consistently reproduced Koza's earlier results.

GP with ADFs was able to evolve at least one successful solution for each program architecture used. Solutions with ADF architectures contained a similar number of program

Table 3: Overall Performance

	Trials	Successes	Success Rate (%)
Experiment I	61	6	9.8%
Experiment II	61	5	8.1%
Experiment III	61	1	1.6%
Experiment IV	61	2	3.2%

Table 4: Characteristics of successful runs

Experiment	Generation Found	Nodes	Time Steps
I	9	15	375
	16	23	310
	19	87	397
	25	57	353
	36	89	380
	52	193	393
II	10	123	327
	11	161	286
	34	93	399
	45	152	374
	47	121	390
III	9	55	388
IV	22	37	365
	39	155	363

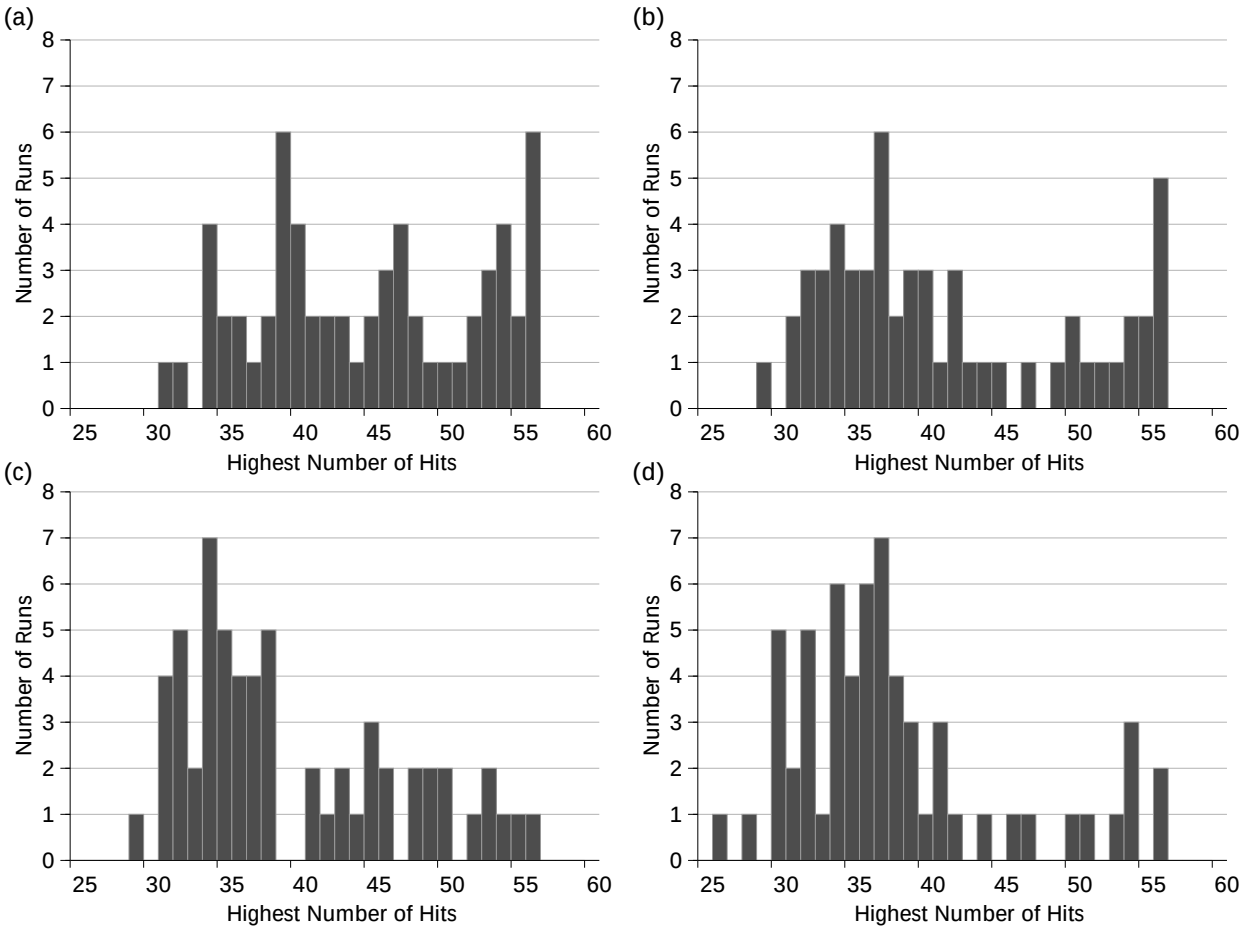

Figure 1: Hits histograms. Experiment I(a), Experiment II(b), Experiment III(c), Experiment IV(d)

nodes and used a similar number of time steps as those with non-ADF architectures. In spite of these successes, GP with ADFs did not out-perform GP without ADFs. Experiment II had a success rate comparable to that of Experiment I. The success rates of Experiments III and IV were slightly lower. Further experiments are needed to determine whether ADFs will improve the performance of GP on this problem. Once enough successful solutions are evolved to generate meaningful performance curves, the computational effort with ADFs (E_{with}) can be compared with that without ($E_{without}$) to determine if they are indeed more effective for this problem.

3.2 Fitness Modalities

Figure 1 presents histograms that provide an overall picture of the number of hits the best-of-run individuals converged upon. The horizontal axis of each histogram refers to the number of hits. The vertical axis is the frequency of runs whose best-of-run individual scored the specified number of hits.

The histograms for all four experiments display a prominent grouping near the 35 to 40 hit range suggesting a sub-optimal modality in the problem. Histograms for Experiments II, III, and IV also suggest something other than a unimodal distribution. Examination of best-of-run individuals from runs at the common sub-optimal modality revealed that the sub-optimal modality is not correlated to any feature inherent to the room or a specific type of behavior exhibited by the individuals. Further experiments need to be performed to determine if the modality is inherent to other aspects of the solution space.

3.3 Search Space

Figure 2 presents spider graphs that provide an overall picture of the manner in which the available primitives were used in optimal and sub-optimal[7] programs evolved in the experiments. Each spider graph consists of 21 axes, one for each of the available primitives. The spider graphs are normalized and present the ratio of occurrences of a primitive to occurrences of all primitives used in active branches of each program.[8]

The sub-optimal solutions found in each of the experiments used `IFLTE` and `PROGN2` frequently, movement and turning sparingly, and a wide range of sensors. The optimal solutions used `IFLTE` and `PROGN2` frequently, and movement and turning sparingly. However, the optimal solutions needed only a subset of the available sensors. Consequently, GP appears to be narrowing the search space by focusing on a few sensors. [Brooks 1992] hypothesized GP may search the solution space more effectively if the robot is started with only some of its sensors and actuators, adding more as fundamental

behaviors are found. Further experiments need to be performed to test the effectiveness of this method.

[Brooks 1990] also hypothesized that a macro capability in the language of GP, especially one that took advantage of symmetry and regularity in a robot, would accelerate the production of good programs. ADFs can be considered as a macro capability in the language of GP because they provide a means for GP to repeatedly use a function. A qualitative observation of suboptimal and optimal solutions across ADF architectures revealed that result-producing branches did not use function-defining branches effectively. Many of the result-producing branches consisted of a statement that called just one function-defining branch `(e.g., (progn (defun ADF 1…) (defun ADF2…) (defun ADF3…) (defun ADF4…) (values (ADF1))`. Consequently, either this suggests that the overhead associated with ADFs may outweigh their benefit in GP's search of the solution space for this problem or that ADFs are not being used in a manner to take advantage of symmetry in the robot.

3.4 Robustness

Figure 3 presents the range of successful solutions found in the experiments. Figures 4 and 5 illustrate the successful individuals executed on fitness cases not used in their evolution.

These figures display a diverse range of solutions and robustness but reveal a commonality in the problem. The figures suggest that GP is parsing the problem into three tasks: wall finding, wall following, and cornering. Strategies for finding the wall range from looping until the wall is hit to walking directly towards the wall once a particular room feature is identified. Many of the solutions exhibited tight looping and oscillating snake-like motion solving the tasks of wall following and maneuvering around corners with one behavior. However, some solutions handled wall following and cornering separately walking straight along the wall and performing tight cornering.

An ideal test of robustness would have consisted of executing the solutions in one to several additional rooms. However, executing the solutions from different initial positions in the same room was sufficient to show the failure modes for the evolved solutions. The solutions evolved were general in the sense that most of them still exhibit wall-following behavior when executed on fitness cases not used to evolve the solution. However, most of the evolved solutions were relatively brittle. Only one of the 14 solutions was able to hit all 56 wall tiles for both tests of robustness. The majority of the individuals timed out, got stuck, or missed a significant number of wall tiles. Some individuals performed well on one of the additional fitness tests but poorly on the other. This may be attributed to an evolved "handedness" of the solutions. The solutions performed better on the robustness tests when the robots, after finding a wall, traveled along that wall around the room in the same direction (i.e., clockwise or counterclockwise) that the robots traveled along the wall in the fitness case used in their evolution. This result suggests a modification in the fitness case used. In particular, we hypothesize that a room that would inherently demand a solution requiring a robot to switch directions during a run may yield a higher percentage of robust solutions. The results suggest that a single fitness case (consisting of one initial facing direction, initial position, and room shape) was

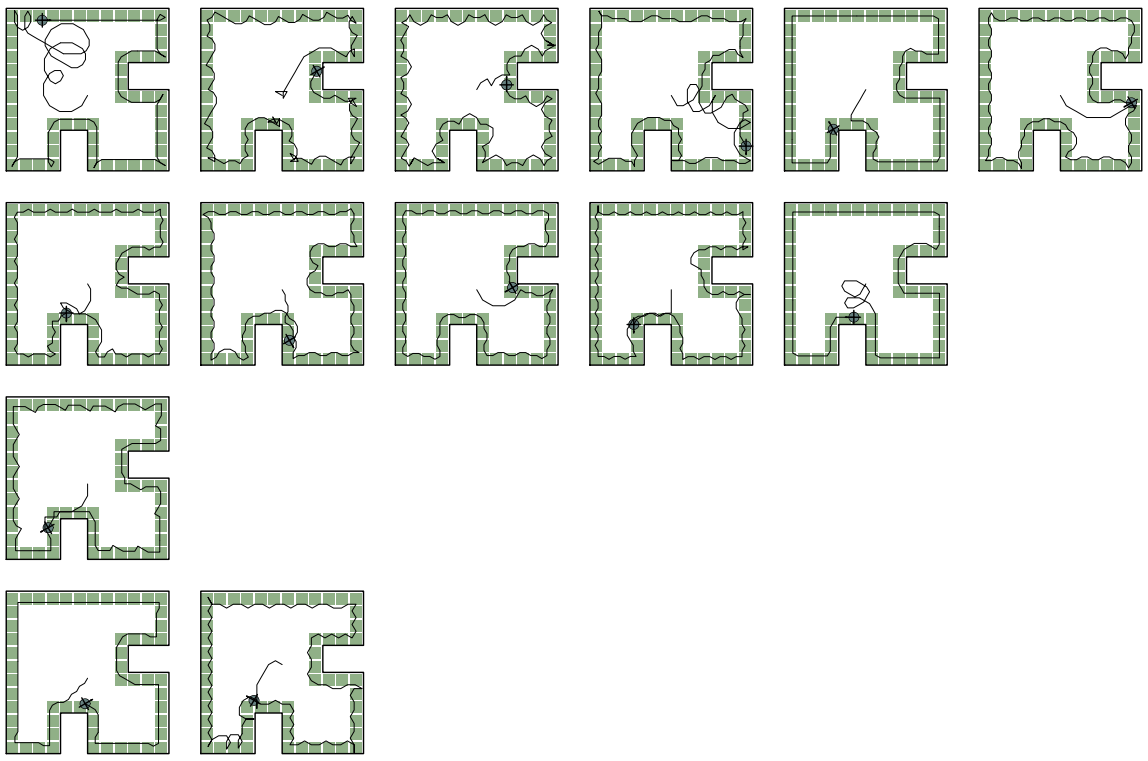

Figure 3: Successful Solutions. Experiment I(row 1), II(row 2), III(row 3), IV(row 4)

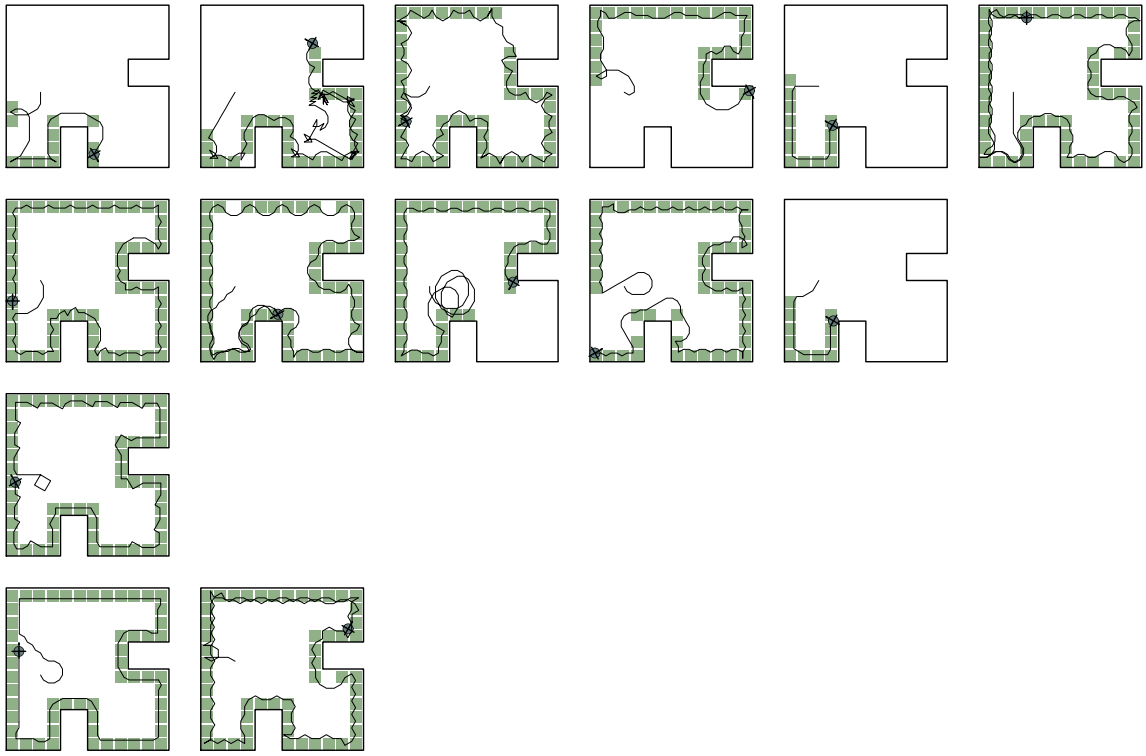

Figure 4: Robustness: initial location changed. Experiment I(row 1), II(row 2), III(row 3), IV(row 4)

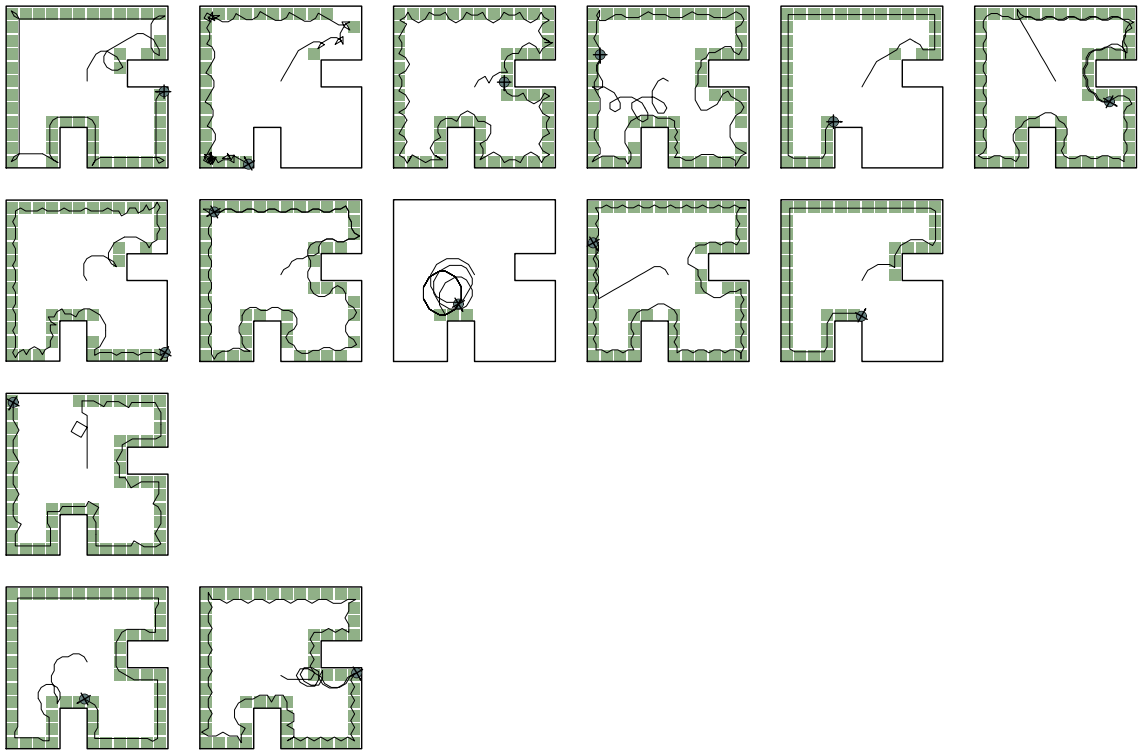

Figure 5: Robustness: initial direction changed. Experiment I(row 1), II(row 2), III(row 3), IV(row 4)

not sufficient to evolve a high percentage of robust solutions to the wall-following problem. Further experiments are needed to determine whether multiple starting positions in the same room, different rooms, or the suggested modification of the room fitness case would help in evolving a higher percentage of robust solutions.

4. Conclusions

This paper contributes several key elements to the benchmark for the wall following robot problem including:

- An independent implementation of the wall following robot problem produced results similar to those presented by Koza (Section 3.1)
- Identification of a modality in this problem that exists across architectures and cannot be attribute to a feature of the room or the robots' behavior (Section 3.2)
- Optimal solutions seem to need only a subset of available sensors(Section 3.3)
- A test of ADFs that suggests the introduction of ADF architectures to this problem does not improve GP's ability to search the solution space as it is unable to use them effectively (Section 3.3)
- The solutions evolved using GP on this problem are relatively brittle (Section 3.4)

Future work includes:

- Further tests with ADFs
- Identification of possible causes for the sub-optimal modality
- Use of another fitness set, consisting of either a different room or the same room but with several different starting locations for the robot
- Tests of Brooks' incremental functionality (sensors/actuators) hypothesis

(The software used to create this benchmark is available at http://www-personal.engin.umich.edu/~daida/)

Acknowledgments

This research has been partially funded with internal grants within U-M. We gratefully acknowledge the following people: J. Koza, J. Rice, for their original work with GP and evolutionary robotics; R. Riolo, for discussions on GP; S. Daida, for logistical support; the reviewers for their insightful feedback; E. Linas, for help in preparing figures for the camera-ready paper; J. Ross, M. A. Ross, and A. Ross, for their support.

Bibliography

Brooks, R. A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*. March. pp. 14-23.

Brooks, R. A. 1992. Artificial Life and Real Robots. In Varela, F.J., and Bourgine, P.(editors). *Towards a Practice of Autonomous Systems: Proceeding of the First European Conference on Artificial Life.* The MIT Press.

Handley, Simon. G. 1994. The Automatic Generation of Plans for a Mobile Robot via Genetic Programming with Automatically Defined Functions. In Kinnear, K. E. Jr. (editor). *Advances in Genetic Programming*. The MIT Press.

Jefferson, D., R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang, 1992. Evolution as a Theme in Artificial Life: The Genesys/Tracker System. *Artificial Life II*, SFI Studies in the Sciences of Complexity. Vol X. Addison-Wesley.

Koza, J.R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, J.R. 1994b. Evolution of a Subsumption Architecture that Performs a Wall Following Task for an Autonomous Mobile Robot. In Hanson S. J., et al.(editors) *Computational Learning Theory and Natural Learning Systems.* The MIT Press.

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

Koza, J.R. 1992b. Evolution of subsumption using genetic programming. In Varela, F.J., and Bourgine, P.(editors). *Towards a Practice of Autonomous Systems: Proceeding of the First European Conference on Artifical Life.* The MIT Press.

Mataric, Maja J. 1990. *A Distributed Model for Mobile Robot Environment-Learning and Navigation.* MIT Artificial Intelligence Laboratory technical report. AI-TR-1228.

Nordin, P., W. Banzhaf 1995. Genetic Programming Controlling a Miniature Robot. *In Working Notes of the AAAI-95 Fall Symposium Series, Symposium on Genetic Programming.* MIT

Reynolds, C.W. 1994a. An evolved vision-based behavioral model of obstacle avoidance behavior. In Langton, C.G. (editor) *Artificial Life III*, SFI Studies in the Sciences of Complexity. Volume XVII. Addison-Wesley.

Reynolds, C.W. 1994b. Evolution of obstacle avoidance behavior. Using noise to promote robust solutions. In Kinnear, K. E. Jr. (editor). *Advances in Genetic Programming*. The MIT Press.

---

[1] Koza's original work starts the robot at a position near the center of the room(12, 16). This work is a slight variation starting it in the center of the room at position(13.8, 13.8).

[2] The initial goal was 60 runs. The hostile nature of the computing environment caused many of our runs to be killed so multiple runs were started for redundancy. Experiment II completed one additional run so for consistency, we decided to perform an extra run for each of the other experiments.

[3] Our runs were considerably slower than that reported in [Koza 1992a, 1994b]. In retrospect, we believe that three factors contributed to this slowdown. First, we may have unintentionally penalized runtime performance with our code implementation. Second, our tests ran in the background in a computing environment that is shared extensively by students, staff, and faculty. Third, we used a generic workstation rather than a Texas Instrument Explorer II Plus workstation (a LISP-specific machine) used in Koza and Rice's original work.

[4]To control all experiments as much as possible, we decided that at no time during a run would we intervene. This non-intervention policy consequently precludes “on-the-fly” redimensioning and extensioning.

[5] The GP kernel found in [Koza 1994a] only supports architectures with up to two function-defining branches.

[6] The performance curve found in [Koza 1994b] suggests that "processing a total of 1,176,000 individuals (i.e. 2,000 times 28 generations times 21 runs) is sufficient to yield a solution to this problem with 99% probability.”

[7] The sample of sub-optimal programs was taken from the modality common across all architectures.

[8] A distinction is made between active branches and non-active branches as a basis for comparison between the non-ADF and ADF architectures. The non-ADF programs consisted of only one branch that must be activated. The ADF architectures consisted of one result-producing branch that must be activated and several function-defining branches which may be activated. The primitives in a function-defining branch were counted each time the designated function-defining branch was called.