UCRL-JC-133191
PREPRINT

Evolutionary Software for Autonomous Path Planning

M. Hage
S. Couture

This paper was prepared for submittal to the
Eighth International Topical Meeting on Robotics and Remote Systems
Pittsburgh, PA
April 25-29, 1999

February 10, 1999

EVOLUTIONARY SOFTWARE FOR AUTONOMOUS PATH PLANNING

Matthew Hage (Lawrence Livermore National Laboratory)
Scott Couture (Lawrence Livermore National Laboratory)
California Institute of Technology, MSC 235, Pasadena, CA 91126
hage@its.caltech.edu

Abstract

This research project demonstrated the effectiveness of using evolutionary software techniques in the development of path-planning algorithms and control programs for mobile vehicles in radioactive environments. The goal was to take maximum advantage of the programmer's intelligence by tasking the programmer with encoding the measures of success for a path-planning algorithm, rather than developing the path-planning algorithms themselves. Evolutionary software development techniques could then be used to develop algorithms most suitable to the particular environments of interest. The measures of path-planning success were encoded in the form of a fitness function for an evolutionary software development engine. The task for the evolutionary software development engine was to evaluate the performance of individual algorithms, select the best performers for the population based on the fitness function, and breed them to evolve the next generation of algorithms. The process continued for a set number of generations or until the algorithm converged to an optimal solution. The task environment was the navigation of a rover from an initial location to a goal, then to a processing point, in an environment containing physical and radioactive obstacles. Genetic algorithms were developed for a variety of environmental configurations. Algorithms were simple and non-robust strings of behaviors, but they could be evolved to be nearly optimal for a given environment. In addition, a genetic program was evolved in the form of a control algorithm that operates at every motion of the robot. Programs were more complex than algorithms and less optimal in a given environment. However, after training in a variety of different environments, they were more robust and could perform acceptably in environments they were not trained in. This paper describes the evolutionary software development engine and the performance of algorithms and programs evolved by it for the chosen task.

Introduction:

Evolutionary software has been used for path planning previously [3]. The problem being evaluated here differs in that the algorithm is to guide a robot to the core of a melted nuclear reactor, such as at Chernobyl, retrieve a piece of that core, and return it to processing site, avoiding physical obstacles and prolonged exposure to radiation. Our research used evolutionary software for path planning in known and unknown environments. Genetic *algorithms* were evolved for path planning in known environments, as an alternative to methods such as potential fields. Evolved algorithms demonstrated an ability to avoid the physical obstacles as well as to minimize the distance and time spent around radiation sources. Genetic *programs* were evolved in the form of sensor-based path-planning control programs. Genetic programs were evolved on a variety of known environments to develop a control program for the given task. Once this control program is evolved, it may be used for autonomous path planning in an unknown environment, as an alternative to such methods as Lumelsky's bug[5] or the A* algorithm[6].

Evolutionary software may be able to perform path planning more optimally than alternative path-planning algorithms. A genetic algorithm will test several different possible paths through a known environment, continuously attempting to maximize the fitness of the path through this environment. Genetic programs may be more optimal than traditional sensor-based path-planning algorithms, as genetic programs can be trained in an increasing number of known environments. Evolutionary software may exploit opportunities that traditional algorithms are unable to examine.

A genetic algorithm is used to determine a path through an environment known *a priori* composed of goal (reactor core), processing point, physical obstacles, and radiation obstacles. A genetic program is used to create a control program that again guides the robot through the environment, using sensory input compared to constants. This more robust controller may be capable of performing the specified tasks in unknown environments. The implementation of the genetic program was based on the work of Koza [1].

Objective:

The objective of the evolutionary software was to obtain an algorithm that planned the path of a mobile rover in an environment containing physical and radioactive obstacles. The goal of the algorithm was to have the rover begin at an initial starting position, move to a goal to retrieve some radioactive material, then take that material to a pre-determined dumping site where the material could be processed. While executing these tasks, the robot was to avoid any physical obstacles as well as avoid prolonged exposure to radiation sources. Several approaches were taken to solve this problem.

The simplest approach to this problem was to use a genetic algorithm that assumed that the environment would be known *a priori*, and that the algorithm would need to navigate through only one environment. In this case, the algorithm was not fed any sensory information and developed in response to physical interactions with its environment.

A more complex genetic program approach was pursued in an attempt to develop a robust control algorithm that would operate at every step in the path, acting based on comparisons between sensory input and randomly chosen values. The genetic program was trained over several different environments in order to make a more robust algorithm. The objective of the genetic program was to create a control program capable of sensor-based path-planning through a variety of environments containing physical and radioactive obstacles.

Environment:

The training environment was a cartesian coordinate space, containing an initial position of the robot, a goal, a processing point, and physical as well as radioactive obstacles. The physical obstacles were defined, stationary points in the cartesian coordinate system. The physical obstacles were impenetrable, so that a robot would bounce off an obstacle were it to attempt to move into the cartesian coordinate of that obstacle. Each time a robot hit an obstacle, it was "hurt" by an amount specified by the user. The radiation obstacles were also defined and stationary, but were not impenetrable. The robot was "hurt" by each radiation source depending on a K/r^2 relationship, with the constant, K, set by the user to represent the strength of each radiation obstacle. If the robot attempted to move to the coordinates of the radiation source, the robot was "hurt" quite badly, but was allowed to pass through that point. This was to simulate radiation sources that may be either too small to be physical obstacles or not directly in the environment, possibly being on another level of the reactor.

Evolutionary Software Development System:

An evolutionary software development system (engine) was developed to solve the given path-planning problem. The evolutionary software development engine begins by generating an initial population of individuals. The initial population is generated by randomly choosing the number of operations that make up each member of the population, then randomly selecting the function to be carried out during each operation of each member of the population. The initial population algorithms are then executed to evaluate the fitness of the algorithms. The population is then ranked based on the score assigned to each algorithm by the fitness function. Then, the population is bred. The breeding program preserves the best 10% from each generation into the next generation. The remaining 90% of the next generation are formed by sexual recombination of the top 50% of the prior (existing) generation. Then, the new generation is evaluated, ranked and bred. The process continues until the maximum number of generations is achieved.

The evolutionary software engine is generic in nature. In order to evolve a new algorithm for a specific situation or environment, it is provided:

information describing the training environment including the starting point, core location, etc.;
a set of elemental functions that could be combined to form an algorithm or program;
a fitness function; and,
parameters describing the maximum length of an individual, number of individuals in a generation and the maximum number of generations to evaluate.

When building simple algorithms, the only functions provided are the motion operations: move up, move down, move left, and move right. When developing more complex genetic programs, the operations stay, move back, move forward, and move-to-goal are added. In addition, in order to create a sensor-based path-planning algorithm, sensing functions are provided to generate the distance and direction of the nearest obstacle and the direction and strength of the largest radiation source.

The fitness function evaluates the performance of each algorithm in the population. The fitness function scores each algorithm based on how well it performs the given tasks in the example environment, with a lower score corresponding to a better performance. Points are added to each member's score for hitting physical obstacles, for distance from radiation obstacles, for each step taken, and for finishing either away from the goal, or, if the goal has already been reached, for the distance away from the processing point. Points are deducted from each member's score for reaching the goal, and reaching the processing point after reaching the goal. The counter that incremented for each step taken is allowed to end whenever the robot reaches the processing point after reaching the goal, or at a number of steps specified by the user. In this manner, the algorithms that slowly reach the processing point are not ranked as highly as those that quickly move around the environment. This feature is included to simulate real cumulative overall radiation as well as the reality of limited amounts of power available to an autonomous rover.

Implementation:

The evolutionary software is implemented by numerically encoding the motion operations and the sensory information available to the genetic program as shown in Figure 1. For the genetic algorithm, each member of the population is an array of motion numbers. These numbers (motions) are executed sequentially until the counter reaches its maximum value or until the robot reaches the processing point after reaching the goal. If the end of the array is reached before either of these two events occurs, the genetic algorithm loops around to the beginning of the array. The fitness of each individual is evaluated and the population is ranked. The breeding program randomly selects pairs (couples) of individuals ranked in the top one half of the population. Each member of a couple is then subdivided into two parts at a randomly selected point in their array. The four parts around these random points are then recombined to form the two children of each couple. The first part of the first parent is matched with the second part of the second parent for the first child, and the second child consists of the first part of the second parent and the second part of the first parent. These new members of the new population are then evaluated in the fitness test, ranked, and then returned to the breeding program.

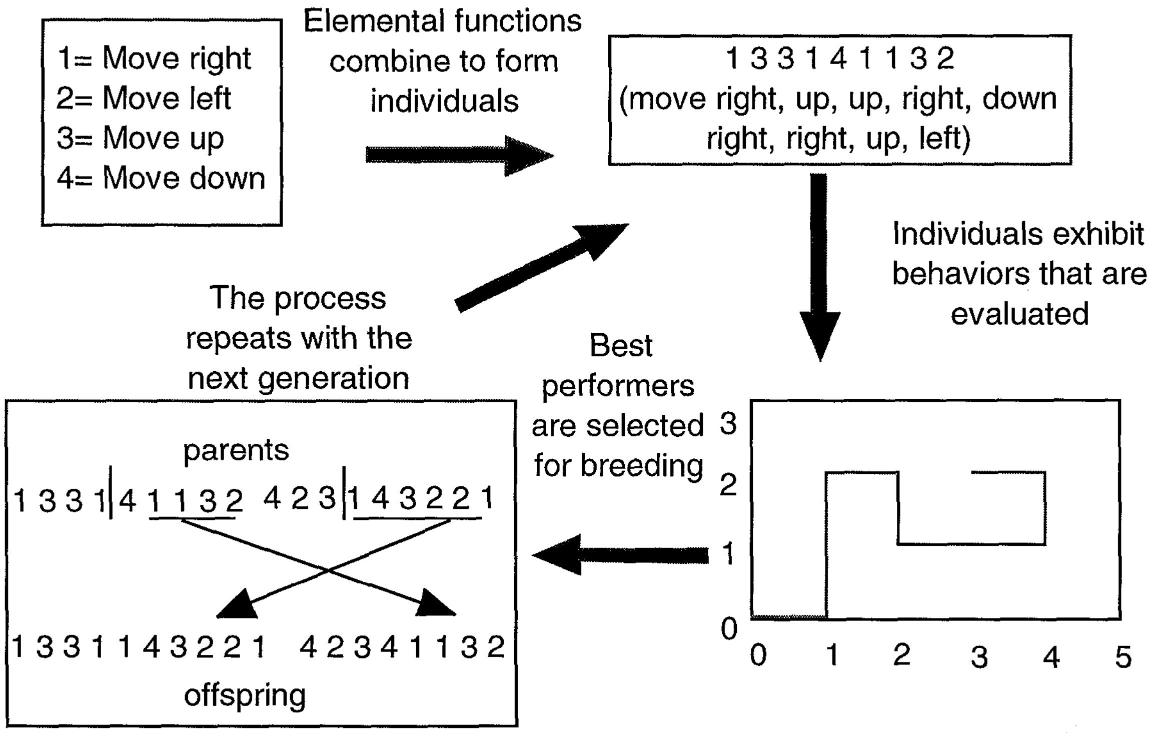
1= Move right
2= Move left
3= Move up
4= Move down
Elemental functions combine to form individuals
1 3 3 1 4 1 1 3 2
(move right, up, up, right, down right, right, up, left)
Individuals exhibit behaviors that are evaluated
Best performers are selected for breeding
The process repeats with the next generation
parents
1 3 3 1|4 1 1 3 2 4 2 3|1 4 3 2 2 1
1 3 3 1 1 4 3 2 2 1 4 2 3 4 1 1 3 2
offspring
0 1 2 3
0 1 2 3 4 5

Figure 1. Genetic algorithms are implemented using series of integers representing behaviors

The more complex implementation of the genetic program requires more complex member algorithms as well as a more complex breeding algorithm. The genetic program creates decision tree algorithms that are parsed through at each cartesian coordinate reached by the algorithm. Individual genetic programs are represented by parse trees of numbers representing motions or comparisons of sensory data to constants, as shown in Figure 2. Comparisons utilize the *if less than or equal to* statement to compare sensory input to constant values. Depending upon the result, either moves or additional comparison statements may be executed. Each individual in each generation is a parse tree represented by a three dimensional array. These arrays are depth of the parse tree, width of the parse tree, and operations at each node in the parse tree. Each node consists of four statements: a 0,1 trigger to tell if this node is a motion or a comparative statement, a motion number, a sensor number, and a constant value number.

When a given algorithm created by the genetic program is being evaluated by the fitness function, the algorithm is enacted as the member passed through each cartesian point. At each of these points, the algorithm begins executing at the top node and proceeds through a parse tree. The algorithm operates at each node based on the binary trigger. If the trigger indicates a move, then the motion of that given node is executed, the algorithm moves to the cartesian point indicated, and the algorithm is started over at the new cartesian coordinate. If the node indicates a comparison, the value returned from the sensor indicated in the node is compared to the constant value indicated in the node. The algorithm then proceeds to either of two nodes, based on the outcome of the *if less than or equal to* comparison between these two values.

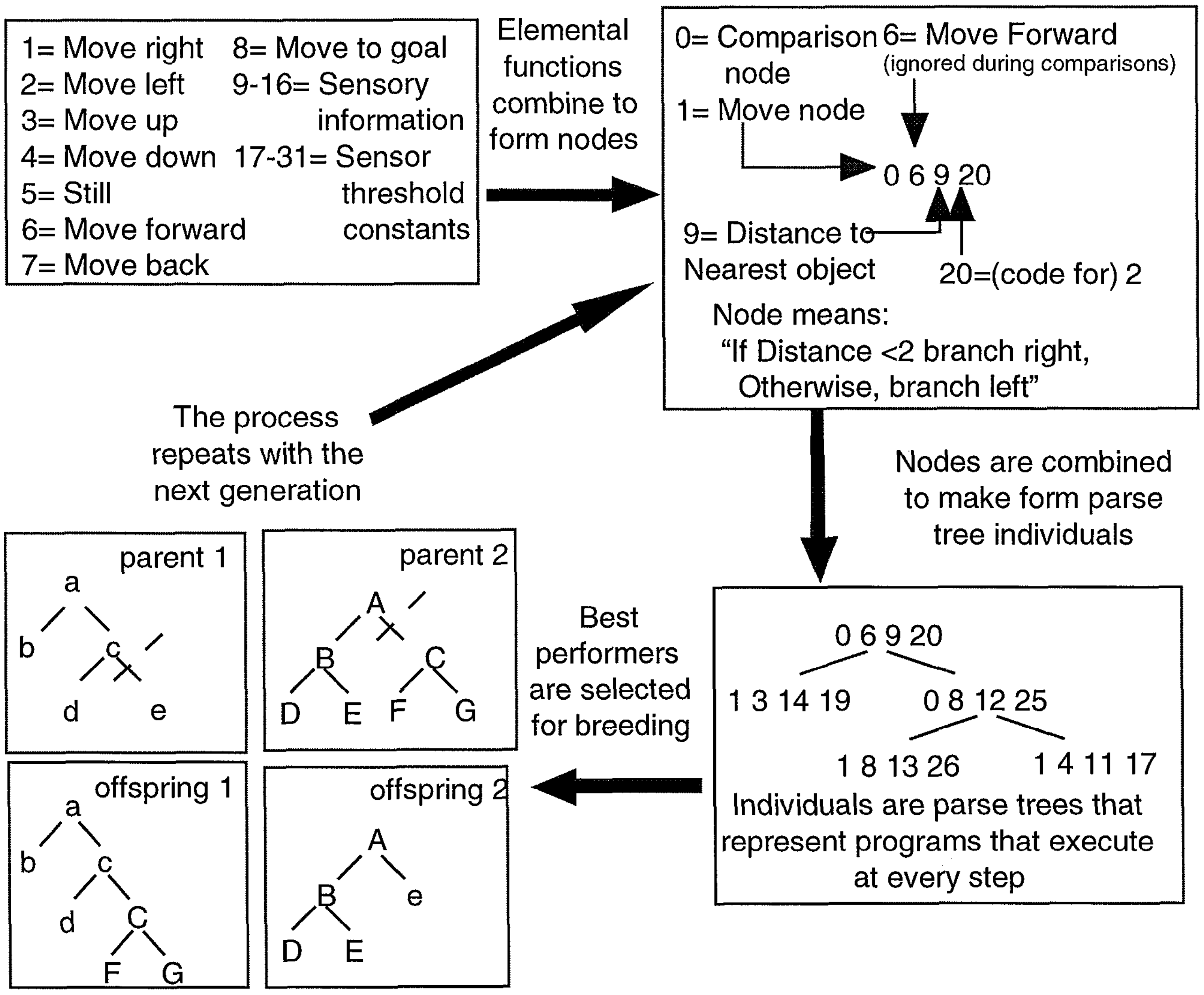
1= Move right
2= Move left
3= Move up
4= Move down
5= Still
6= Move forward
7= Move back
8= Move to goal
9-16= Sensory information
17-31= Sensor threshold constants
Elemental functions combine to form nodes
0= Comparison node
1= Move node
6= Move Forward (ignored during comparisons)
0 6 9 20
9= Distance to Nearest object
20=(code for) 2
Node means: "If Distance <2 branch right, Otherwise, branch left"
The process repeats with the next generation
Nodes are combined to make form parse tree individuals
0 6 9 20
1 3 14 19
0 8 12 25
1 8 13 26
1 4 11 17
Individuals are parse trees that represent programs that execute at every step
Best performers are selected for breeding
parent 1
a
b
c
d
e
parent 2
A
B
C
D
E
F
G
offspring 1
a
b
c
d
C
F
G
offspring 2
A
B
e
D
E

Figure 2. Genetic programs are implemented as node trees representing decisions and actions

Breeding of genetic programs is accomplished by exchanging portions of the parse trees that define individuals (genetic programs) created by the evolutionary software. Essentially, the breeding program operates by swapping branches of the parse trees between two different members. Two parents are chosen randomly from the top half of the population. A random node is chosen in each of the parents, at which the parse tree of the parent is cut. This forms two smaller parse trees segments from each parent tree. The children are formed by recombining tree segments from the parents in a manner similar to that of the genetic algorithm. The first child consists of the tree segment from above the cut on the first parent, spliced to the tree segment from below the cut on the second parent. The second child consists of the top tree of the second parent and the bottom tree of the first parent.

Results:

Evolutionary software was able to solve the problems presented here.

Evolutionary software was able to develop an algorithm that executes a path through a relatively simple environment in a few generations (optimal after approximately 250 generations) with a small population (1000 individuals). The results are shown in Figure 3. Genetic algorithms encoding plans through much more complicated environments were developed given a larger population and more generations over which to evolve.

The evolutionary software development system required an even larger population (2000 individuals) and many generations (50) to create a genetic control program to solve a single environment. In order to solve several environments simultaneously, a population of 1000 was used over 100 generations. However, after "training" on 3 different environments, the genetic software *was able* to create a control program that was capable of performing the given tasks in all of the environments. It was then tested on a fourth and fifth environment that were not known *a priori* and performed acceptably in one of them and marginally in the other. Results in the training environments and a four (unknown) environment are shown in Figure 4. It is clear that in order to develop a robust genetic program that operates in a wide variety of environments that are not known *a priori*, the set of training environments needs to be sufficiently rich to encompass a wide variety of possible scenarios.

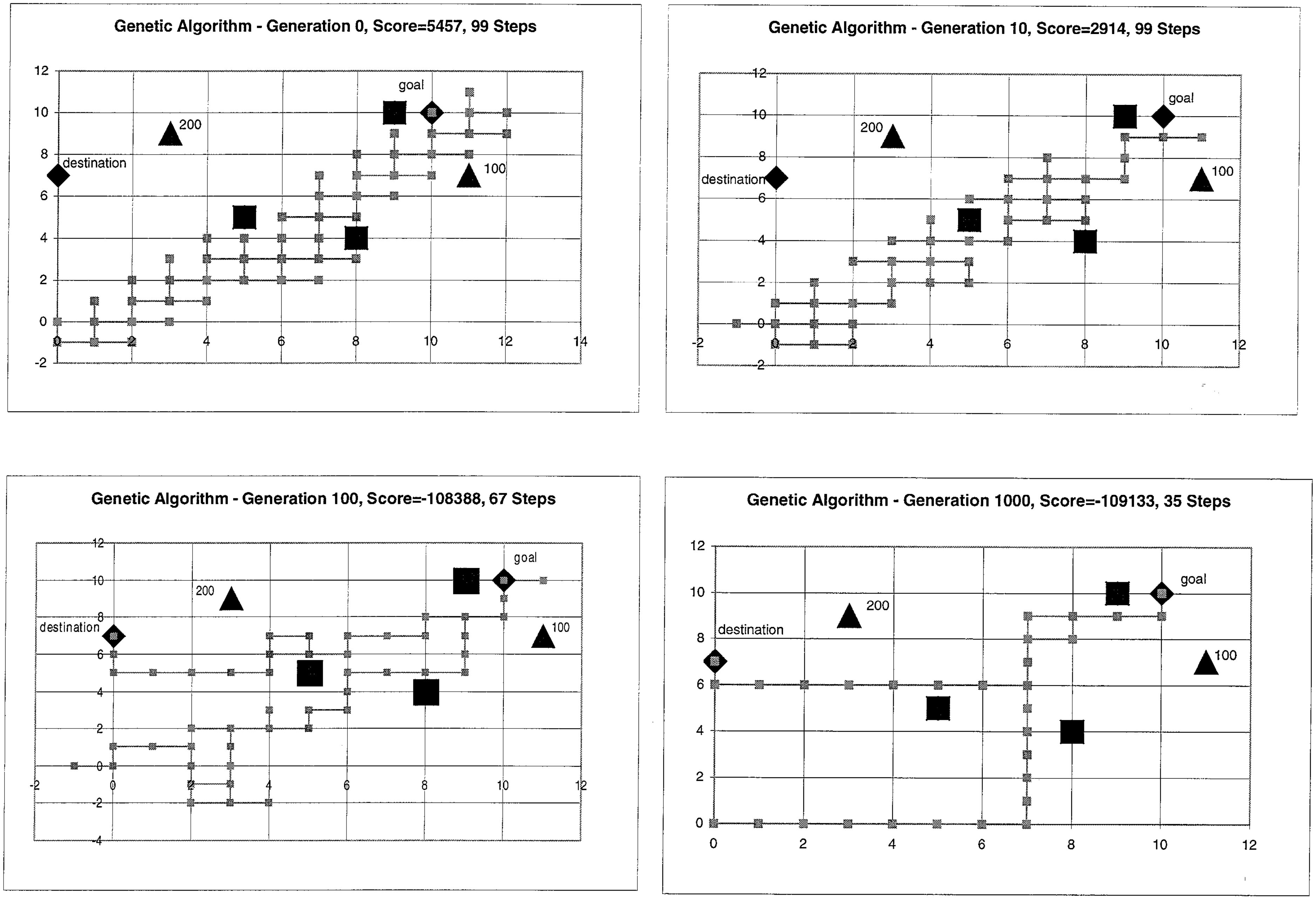

Figure 3. Comparison of genetic algorithm performance as algorithm is evolved over 1000 generations

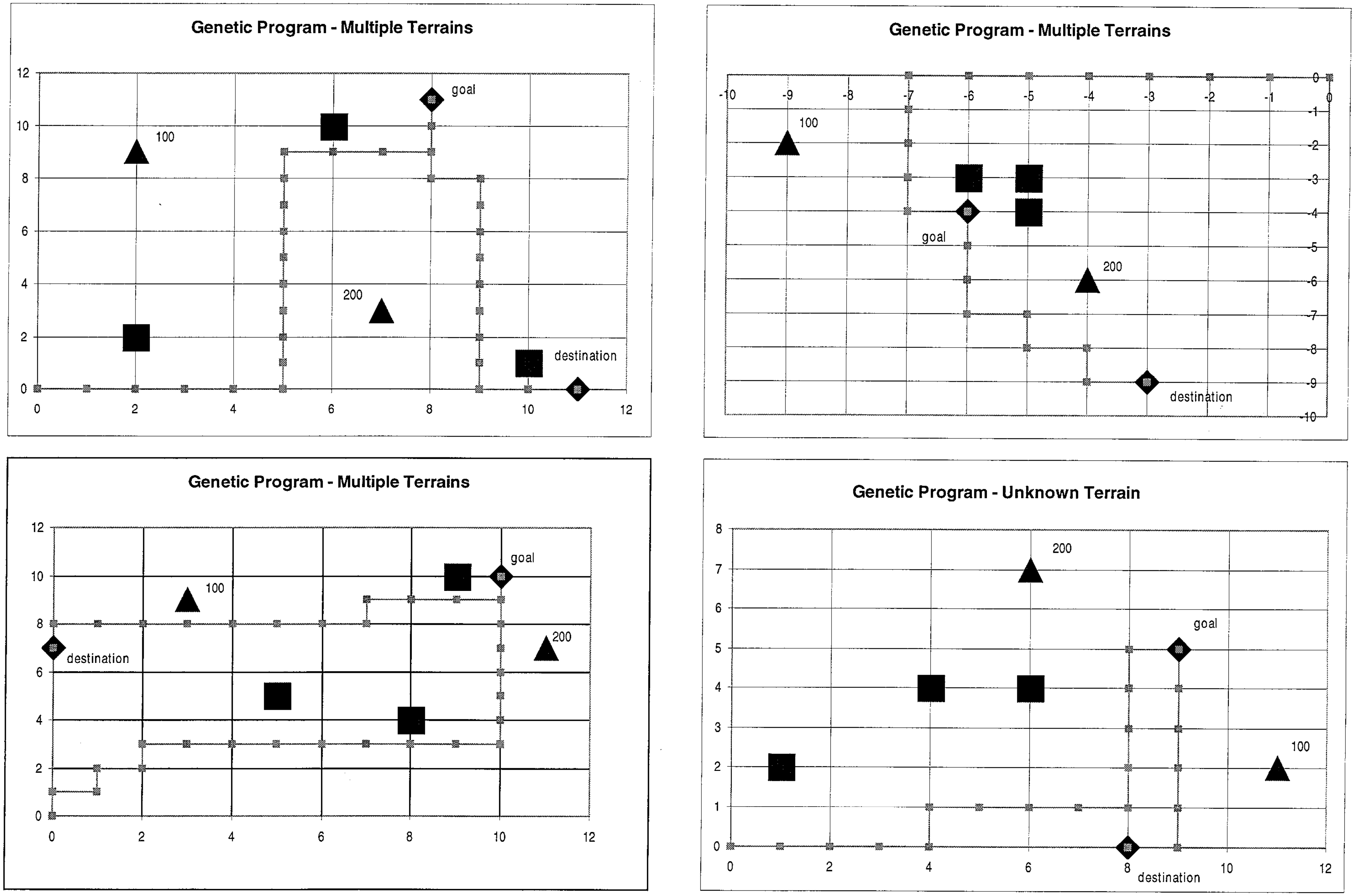

Figure 4. Genetic program evolved over multiple terrains performed acceptably on some unknown terrains

Future Work and Discussion:

Evolutionary software was able to perform autonomous path planning. Genetic algorithms were capable of path planning through an environment known *a priori*. Genetic programs were capable of creating sensor-based path-planning control programs that were trained on several environments and then successfully tested on an environment not previously encountered. Evolutionary software may be an additional tool for use in the field of path planning, through both known and unknown environments.

Future work will include increasing the complexity of the simulated environment in order to ascertain the viability of the evolutionary software in increasingly complex environments. The genetic algorithm described appears to be a capable path-planning algorithm through an environment known *a priori*, and further work will include utilizing genetic algorithms to solve path-planning problems.

Sensor-based path planning with evolutionary software is more complex. The genetic program described was trained over several environments and then utilized successfully in an unknown environment. Future work will include training the genetic program in more environments and more complex environments. The genetic program may also be allowed to include more sensory information, such as the history of the path taken up to the current point, the amount of steps already taken, or possibly even be allowed to create a roadmap of terrain already encountered.

References:

[1] Koza, J. (1992), Genetic Programming, The MIT Press

[2] 'Genetic Programming: Papers from the 1995 AAAI Fall Symposium Technical Report, FS-95-01

[3] Haupt, R.; Haupt, S. (1998), Practical Genetic Algorithms, John Wiley & Sons

[4] Goldberg, D. (1989), Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley Pub.

[5] Lumelsky, V.J. and Stepanov, A.A. (1986), Dynamic Path Planning for a mobile automaton with limited information on the environment, *IEEE transactions of Automatic control*, pages 1058-1063

[6] Pearl, J. (1984), Heuristics: Intelligent Search Strategies for Computer Problem Solving., Addison Wesley Pub.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.