AUTONOMOUS ROBOT NAVIGATION SYSTEM USING A NOVEL VALUE ENCODED GENETIC ALGORITHM

Thomas Geisler, Theodore W. Manikas

Department of Electrical Engineering
The University of Tulsa
Tulsa, Oklahoma 74104, USA

ABSTRACT

This paper describes the development of a genetic algorithm (GA) based path-planning software for local obstacle avoidance. The GA uses a novel encoding technique, which was developed to optimize the information content of the GA structure. Simulation results were used to further optimize the developed software and determine its optimum field of operation. The results show that the GA finds valid solutions to the path-planning problem within reasonable time and can therefore be used for real world applications.

1. INTRODUCTION

As a result of increasing automation in almost every section of our lives, robots have become an important part of many applications. Recently the field of autonomous navigating robots has become increasingly interesting for many researchers [2][7]. Autonomous navigating robots operate in a given environment without being remotely controlled by a human operator. The work presented in this paper is part of a project to build an autonomous robot, which can be used as a platform for various applications.

This project has three major divisions: visual detection of the environment, path planning, and controls. The path-planning component is again divided in two sections: global path planning and local path planning. Running simulations on both global and local path planning in different environments will determine which approach yields the best performance. The following are general specifications for both approaches:

1) Given are the length and width of the room in which the path planning takes place. A grid system is applied to the room, similar to a chessboard. Thus, the room is divided into rows and columns. Locations of known obstacles are marked as "occupied cells" in the grid.
2) Also given are the row and column coordinates of the start-point and the end-point of the desired robot's movement.
3) The robot can move on all 'free' cells, where the center of the robot moves along an imaginary line from the center of one cell to the center of another cell.
4) A genetic algorithm [3][5][6] is used for the path planning software since the path-planning problem is NP-hard [4].

Specific requirements for global path planning are that the visual obstacle detection is done before the path planning and any navigation of the robot. After the obstacle detection procedure has been completed, the locations of all obstacles in the navigation area are assumed to be known. The path planning software is then applied to the entire room to find the best feasible path from the start point to the end-point.

For local path planning, the locations of any obstacles are unknown. Assuming that there are no obstacles in the navigation area, the shortest path between the start point and the end point is a straight line. The robot will proceed along this path until an obstacle is detected. At this point, the path planning software is applied to find a feasible path around the obstacle. After avoiding the obstacle, the robot continues to navigate towards the end-point along a straight line until (1) the robot detects another obstacle or (2) the desired position is reached. An example for local path-planning is shown in Figure 1.

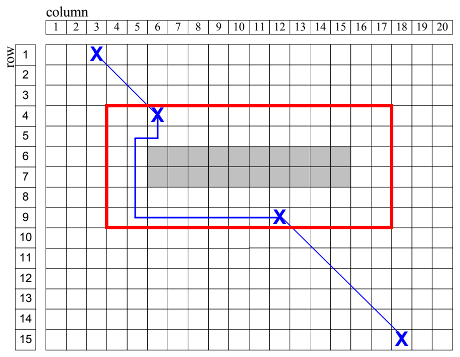


Figure 1. Path planning example for local obstacle avoidance, applied on a subsection of the search space.

This paper concentrates on the software for local path planning. The research objective was to produce a software routine that will be incorporated in the main software that controls all three different parts of the project: the visual obstacle detection, the path planning and the actual navigation of the robot.

2. NOVEL ENCODING TECHNIQUE FOR PATH-PLANNING GA

2.1 Basic Approach and Preliminaries

A genetic algorithm (GA) is used for path planning as specified earlier. Genetic algorithms are heuristic optimization methods whose mechanisms are analogous to biological evolution. A GA operates on a population of chromosomes, which represent possible solutions for a given problem. Each chromosome contains a sequence of genes. The following is a pseudo code for a general GA:

- **Generate** the **initial** parent population

- **Evaluate** the initial parent population
- **Loop** until **termination** criteria is satisfied
    - **Select** chromosomes for reproduction
    - Create offspring using reproduction operators such as **crossover** and **mutation**
    - **Replace** parent population by offspring population
- **Return fittest chromosome** of last parent population

The speed of genetic algorithm depends heavily on the encoding scheme of the chromosomes and on the genetic operators that work on these chromosomes [1][8]. In order to speed up a GA, the chromosome's and gene's structures need to be as simple as possible. In addition, only a few, but very effective, reproduction operators should be applied on the chromosomes. Having that in mind, a novel encoding technique, based on value encoding [5], was developed. In value encoding, any type of number, character or object can be assigned to each gene. Most traditional methods that use value encoding do not use the information of a gene's position. However, this information allows the novel technique to more efficiently use the value-encoding scheme, which keeps the gene structure as simple as possible. The remainder of this paper presents the novel gene structure and the GA operators.

2.2 Gene Structure I: Location

The proposed encoding technique uses the information of a gene's position as well as the value stored at that position as an x- and a y-coordinate. These coordinates define the location of a cell within the row and column system. Thus, a gene's position within a chromosome corresponds to a row-number (*RowIndex*). The value, stored in a gene, in a variable called *location*, corresponds to a column-number (*ColumnIndex*).

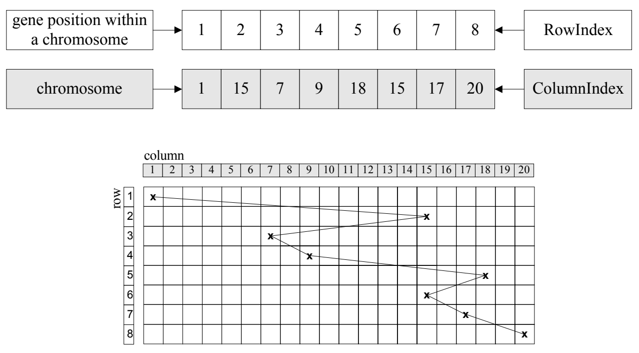


Figure 2. Example of a path being represented by a value encoded chromosome

The length of the room determines the total number of rows. Therefore, the length of the chromosomes is also given, since the total number of rows directly corresponds to the total number of genes within a chromosome. A complete chromosome represents a 'path' from a cell in the first row to a cell in the last row, as shown in Figure 2.

2.3 Gene Structure II: Direction

The gene structure described so far only represents vertices ('corner points' or 'intermediate steps') of a path. To send the robot on a straight line directly from the center of one vertex to the center of the next vertex would mean that the robot moves on a diagonal line across many adjacent cells. This will cause problems if not all adjacent cells that the robot traverses from one row to the next are free of obstacles. A better approach is to go to the side (horizontal) first, turn, and then go down (vertical), or vice versa. To indicate the first direction the robot will turn to proceed to the next vertex, a second variable called *direction* is added to the gene structure. *Direction* is a two-state variable (Boolean), which has either the value horizontal (true) or vertical (false).

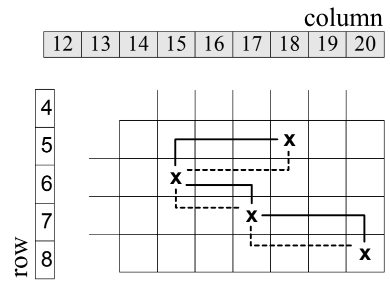


Figure 3. continued example with horizontal/vertical movement included

Figure 3 shows the last four steps of the example path of Figure 1. Now the connection, and therefore the path, from one vertex to the next one is not a diagonal line, but a combination of a horizontal and a vertical movement. Since the first direction that the robot turns to can be either horizontal (solid line) or vertical (dotted line), there are two possible ways to get from one vertex to the next one for each step. The introduced variable *direction* indicates which of the two ways the robot will proceed towards the next vertex.

2.4 Gene Structure III: Validity/Feasibility

A chromosome that uses the proposed gene structure with the variables *location* and *direction* represents an entire path. However, so far we do not have any information about whether a path is feasible; i.e., if all steps between the start-point and end-point of the path are feasible. A step is *feasible* if all cells between the start-point and the end-point of that step do not contain any obstacles. To indicate the feasibility of each step, a third variable is added to the gene structure, *feasibility*. Feasibility is a Boolean variable that will be set true if the step is feasible and false otherwise. This variable is not a true part of the gene structure, since it is not assigned a random value during initialization. The information on a step's feasibility is later used to determine a path's overall fitness.

2.5 Chromosome Structure

In addition to the array of genes, some variables are added to the chromosome structure. *Length,* the number of cells in between the start and the end position, is needed for fitness evaluation purposes. Another variable is *fitness*. The fitness of a chromosome within a population is evaluated during each reproduction cycle. The chromosome's fitness determines its probability to be chosen during the selection process.

The variable *NumberOfInfeasibleSteps* stores the total number of infeasible/invalid steps of the path and the variable *NumberOfTurns* stores the total number of turns that the robot performs on its way from the start position to the end position. The complete chromosome structure is presented in Figure 4.

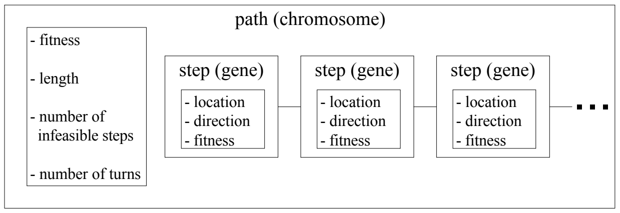


Figure 4. Example of a path being represented by a value encoded chromosome

3. GA ELEMENTS

3.1 Fitness Evaluation

The population of paths is evaluated during each reproduction cycle. The evaluation is based on the paths' fitness, which depends on how suitable the solution (path) is according to the problem. In preliminary evaluations, the values for the path length, the number of turns and the number of infeasible steps are determined for each path in the population. Now these values are set in relation to the entire population and therefore stored as fractional values from 0 to 1, where 1 indicates the optimal fitness value. The shortest path length corresponds to length-fitness $f_{Length} = 1.0$; the longest path corresponds to $f_{Length} = 0$. The greatest number of infeasible steps corresponds to $f_{InfeasibleSteps} = 0$; the least number of infeasible steps corresponds to $f_{InfeasibleSteps} = 1.0$ and $f_{NumberOfTurns}$, respectively. f_{Length} is the fitness value associated to the path length, $f_{InfeasibleSteps}$ is the fitness value associated with the number of infeasible steps, and $f_{NumberOfTurns}$ is the fitness value associated with the number of turns in the path.

The attempt was made to weigh each part of a path's fitness (length, number of infeasible steps and number of turns) according to its importance to the algorithms objective. A path's most important feature is the number of infeasible steps and it should therefore have the biggest influence on the path's fitness. Thus, the other two features will be multiplied by this fitness value. Recall that all three values are fractions between 0 and 1. The other two parts of the path's fitness are multiplied with a weighing factor and added. This sum will be divided by the sum of the weighing factors in order to end up with an over all fitness value between 0 and 1. Finally, the entire fitness function will be multiplied times 100 to end up with fitness values between 0 and 100, which is necessary for the selection process.

$$f_{path} = f_{NumberOfInfeasibleSteps} \cdot \left[a \cdot f_{Length} + b \cdot f_{NumberOfTurns}\right] \cdot \frac{100}{a+b}$$

f_{Path} is the fitness value for the entire path, a is the weighing factor for f_{Length} and b being weighing factor for $f_{NumberOfTurns}$. For our GA, weighing factor a was set to one and weighing factor b was set to two to emphasize the number of turns over the path length.

$$\text{if } \textit{Number\_Of\_Infeasible\_Steps} \mathrel{!=} 0 \text{ then } f_{path} = 0.5 \cdot f_{path}$$

The "if-term" introduces a penalty for paths that contain infeasible steps to emphasize the importance of all steps needing to be feasible for a usable result path.

After the fitness values for all chromosomes in the population have been computed, Rank Selection [5] is used to determine the parent chromosomes that will be used for reproduction.

3.2 Crossover

During the operation of reproduction crossover is applied on the chosen parent chromosomes only within a certain probability, the crossover probability. In the chosen crossover operator, two parent chromosomes are combined applying a single-cross-point, value-encoding crossover [5]. The crossover operator has been modified to produce two offspring chromosomes with each crossover operation. This is achieved by using the gene information, which were not used to build offspring one, in order to build a second chromosome. The function of the crossover operator is illustrated in Figure 5.

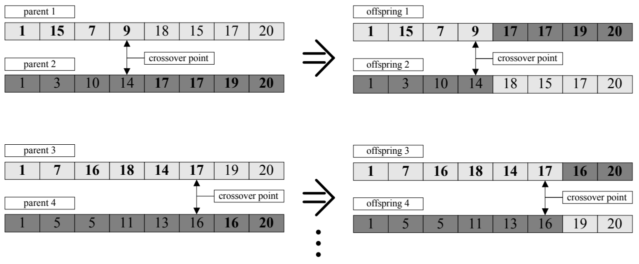


Figure 5. Single-Cross-Point, value-encoding crossover

3.3 Mutation

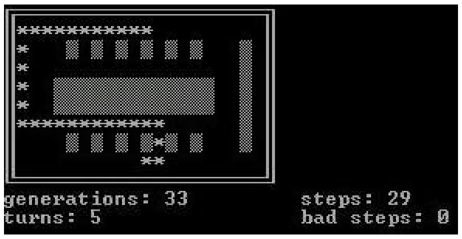
For mutation, almost every operation that changes the order of genes within a chromosome or that changes a gene's value (such as *location* or *direction*) is a valid mutation operator. The mutation operator has been designed according to the addressed path-planning problem.

The chosen mutation operator checks with a mutation probability for every single gene whether it should be mutated or not. If a gene is to be mutated, a random number between 1 and the total number of columns in the search space is assigned to *location* and a random direction, either vertical or horizontal, is assigned to *direction*. This mutation variant has the advantage that it gives the opportunity for a chromosome to become significantly altered. That means that the complete search space will be explored and it therefore prevents the GA from getting stuck in a local optimum.

The fitness of all affected genes (steps) is re-evaluated and stored in the variable *feasibility* immediately after the changes in *location* and *direction* are made. Each step's fitness is therefore always up to date.

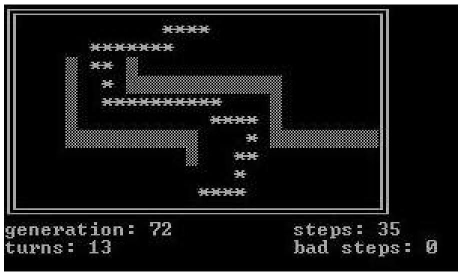
4. SIMULATION RESULTS

After determining the optimum operators and parameters for the GA, path-planning simulations have been conducted on different sized search spaces and with different obstacle configurations.



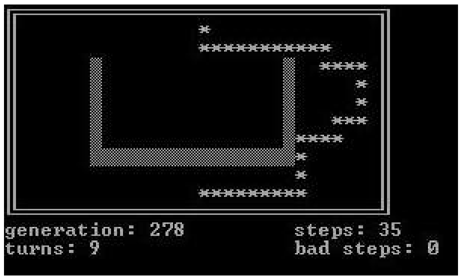
```
generations: 33          steps: 29
turns: 5                 bad steps: 0
```

Figure 6. Example search space (8 rows; 20 columns) for path-planning simulations



```
generation: 72           steps: 35
turns: 13                bad steps: 0
```

(a)



```
generation: 278          steps: 35
turns: 9                 bad steps: 0
```

(b)

Figure 7. Examples for search space (10 rows; 30 columns) for path-planning simulations

Figure 6 and Figure 7 show an example of those simulations. Our path-planning genetic algorithm yields the best performance on small to medium sized search spaces (up to 10 rows and 20 columns). This meets the needs of local path planning, whose requirements are to find a path around individual obstacles and is thus more likely to be applied on relatively small search spaces.

Future work includes further optimizing the reproduction operators, incorporating the local path-planning software into the overall robot control software and testing the algorithm performance on an actual robot.

5. REFERENCES

[1] Thomas Bäck, Ulrich Hammel, "Evolutionary Computation: Comments on the History and Current State", IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, April 1997

[2] Shane Farritor, Steven Dubowsky, "A Genetic Algorithm Based Navigation and Planning Methodology for Planetary Robotic Exploration", Proceedings of the 8th International Conference on Advanced Robotics, 1997

[3] José L. Ribeiro Filho, Philip C. Treleaven, "Genetic-Algorithm Programming Environments", IEEE Computer, June 1994

[4] Y.K. Hwang, N. Ahuja, "Gross Motion Planning – A Survey", ACM Computing Surveys, volume 24, issue 3, pages 219-291, September 1992

[5] Marek Obitko, "Genetic Algorithms", Internet publication, 1998, http://cs.felk.cvut.cz/~xobitko/ga/main.html

[6] M. Srinivas, Lalit M. Patnaik, "Genetic Algorithms: A Survey", IEEE Computer, June 1994

[7] Prahlad Vadakkepat, Kay Chen Tan, "Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path Planning", Proceedings on Congress on Evolutionary Computation, 2000

[8] Jing Xiao, Lixin Zhang, "Adaptive Evolutionary Planner/Navigator for Mobile Robots", IEEE Transactions on Evolutionary Computation, Vol. 1 No. 1, April 1997