Robotica (1997) volume 15, pp 207–211. © 1997 Cambridge University Press

Implementation of a genetic algorithm for routing an autonomous robot

Peter Wide and Holger Schellwat

Institute for Technology and Natural Sciences, University College of Örebro, S-701 82 Örebro (Sweden)

(Received in final form: June 24, 1996)

SUMMARY

A practical implementation of a genetic algorithm for routing a real autonomous robot through a changing environment is described. Moving around in a production plant the robot collects information about its environment and stores it in a temporal map, which is virtually a square grid, taking account of changing obstacles. The evolutional optimizer continuously searches for short paths in this map using string representations of paths as chromosomes. The main features of the implementation include physical realization, random walk exploration, temporal mapping, and dedicated genetic operators.

KEYWORDS: Genetic algorithm; Autonomous robot; Optimal paths; Changing environments.

1. INTRODUCTION

In order to widen the application area for robotics and to make robots more flexible and independent of pre-programming, the use of an adaptive strategy is crucial.[1] This is even more important for an autonomous robot moving in a changing environment, where an adaptive routing strategy is indispensable. But even in a fixed environment, pure off-line planning is not feasible for an autonomous robot, since small tracking errors will add up and lead to inaccurate information about its location. Several learning strategies have been proposed (see Van de Velde[2] for an overview), but only few of them have on-line learning capabilities.

An adaptive learning strategy can be provided by genetic algorithms, which have already proved useful in many applications. For an introduction to genetic algorithms we refer to the book by Goldberg,[3] the articles by Austin,[4] Forrest,[5] Holland,[6] and the survey by Srinivas and Patnaik.[7] Genetic algorithms are a field of intensive theoretical investigations.[8] However, applications to autonomous robot routing are less explored.[9,10]

In Alander,[9] learning robots are simulated in a computer, obstacles are fixed, and the objective of the evolutional optimization is the enhancement of location estimation. De Boer[11] presents a practical implementation of a real autonomous robot, but restricts the scope of investigation to basic behavior, optimizing the tactile – actuational interplay in one dimension. The approach of Lin et al[10] is similar to ours, yet they keep to computer experiments and do not use a real robot.

The purpose of this paper is to show that a genetic algorithm in combination with an internal partial map of the environment is suitable to provide the flexibility to route an autonomous robot through an environment containing changing obstacles in a practical application.

Let us choose the paradigm of a building site. The task of the autonomous robot is to serve a moving working team on the site by continuously fetching supplies from a central store. In shuttling back and forth it will encounter obstacles with its sensors, thus it is collecting information about its environment which it stores in a map. Thus the robot can use the information stored in its temporary map to find a short path from the store to the team by applying evolutional optimization. Whenever it returns from the store finding that the team has moved, the robot will scan its neighborhood to find the team again, and update its map. By even considering some paths which cross obstacles for optimization, it is able to detect the possible removal of obstacles, and keep its map up to date.

In the next section we will define the paradigm, and in section 3 we will describe the genetic routing algorithm, followed by a discussion of our results and conclusions.

2. THE PARADIGM

Using a cell decomposition of the environment, we may represent it by a square grid, $V := (\mathbb{Z}_n)^2$, where $\mathbb{Z}_n = \{0, \ldots, n-1\}$, for some fixed $n \in \mathbb{Z}^+$. These lattice points are marked by a mapping $q_t : V \to \{\cdot, X, O, T\}$, where the value O marks the fixed origin of all walks (the central tool store), T the terminus (the location of the working team), the dot $\cdot$ unobstructed locations, and X an obstacle. Moreover, there is only one origin, i.e. $|q_t^-(O)| = 1$, and one terminus, i.e. $|q_t^-(T)| = 1$, at any time, where q_t^- denotes the pre-image, and we will write $O \in V$ and $T \in V$ for the unique $x \in V$ such that $q_t(x) = O$ or $q_t(x) = T$, respectively.

This environment is allowed to change, with the exception that the position of the origin is fixed. Viewing V as a subset of $\mathbb{Z}^2$, it is naturally equipped with the induced Euclidean metric. However, in order to make the metric compatible with the length of a path, we rather use the taxicab metric, $d((x, y), (u, v)) = |x - u| + |y - v|$, and we define the neighborhood $\partial(x)$ of the point $x \in V$ by $\partial(x) := \{y \in V : d(x, v) = 1\}$. This square grid V can be viewed naturally as a graph G having vertex set V and edge set $E := \{(x, y) \in V \times V : d(x, y) = 1\}$. We will

https://doi.org/10.1017/S0263574797000222 Published online by Cambridge University Press

call a path $(v_1, \ldots, v_k)$ in G *admissible at time t* if for all $1 \leqslant i \leqslant k$ the condition $q_t(v_i) \neq X$ holds.

We do not assume that the robot knows the position of the target in advance. Rather it will have to explore its surroundings in an early phase of its operation until it encounters the target for the first time, which we will call the exploration phase in the sequel. This strategy of the robot is characterized by a mixture of backtracking and random walk. More precisely, the robot will turn right with a probability α when it runs into an obstacle and with the probability $1 - \alpha$ it will proceed in a random direction. The turn right strategy has the advantage of guaranteed escape out of a (connected) maze, but it will not explore the interior of the accessible environment, whereas random walks are locally dense, yet they are unlikely to lead far from the origin.[12] Of course, this strategy could be easily improved, but for our purposes it works well, as its only intention is to provide the robot with some partial information about its environment and thus to implement it into the chromosome pool of its initial population.

We will not turn to the two basic mechanisms of the robot navigator, the temporary map and the evolutional optimizer. Whenever the robot is walking, it maintains a string (of maximal length $4n$ in our implementation) over the alphabet $A := \{l, r, u, d, *\}$, describing a path from the origin as a sequence of left, right, up, or down moves. The asterisk represents a path which terminates due to an obstacle or due to finding the target. The length of this string needs to be limited, in order to keep the length of the chromosomes fixed, and we introduce a rule that the robot has to return to the origin if it runs into an obstacle, into the target, or if the string length exceeds its maximum. This rule is also motivated by its contributing to enhancing the map.

The structure of the robot's map resembles the structure of the environment. At time t, the map is given by $m_t : V \to \{\cdot, X, O, T, ?\}$, where the additional value ? stands for unexplored terrain. Every attempt to move the robot to an unexplored point will result in a map value distinct from ?, thus building up knowledge of the environment. So far obstacles would have to be static. However, by simply forgetting obstacles, i.e. replacing the value $m_t(x) = X$ by the value ?, at some constant rate, the robot is able to consider routes via former obstacles. Hence, slowly moving obstacles are allowed.

The task of the evolutional optimizer is to find short paths from the origin to the target, as soon as the latter is found for the first time. It will be discussed in detail in the next section, for now we will only describe its interplay with the map for the navigator. The chromosomes of a population consist of strings over the alphabet A described above, having their interpretation as descriptions of paths from the origin. The initial population consists of the paths traveled during the exploration phase searching for the target, complemented by random chromosomes to make up a population of fixed size. The fitness function tests chromosomes in their interpretation as paths in the momentary map. Chromosomes that lead to the target relative to the temporary map get the highest ranking, shorter paths even higher, whereas among the chromosomes which do not lead to the target or run into obstacles (in the map) those ones which lead furthest away from the origin are favored. This simple mechanism prevents the optimizer from trying to run through walls. After a fixed number of generations or after the target is found in the map, whatever takes longer, the evolutional optimizer terminates, and the robot travels a path having the best fitness. If this leads to the target, the robot travels back to the origin and the process starts anew. If the path leads to an obstacle, the robot travels back, updates the map, replaces a weakest chromosome with this new path, and the process starts over again. Finally, if the path leads to the map position of the target, but the target is not found there due to a move of the target, the robot scans its neighborhood until it finds the target again, updates the map, travels back, replaces a weakest chromosome with this new path, and starts over again.

This way the robot constantly travels between the origin and the target, improving both its map and the population. Let us summarize this algorithm in pseudo-code:

```
procedure Genetic Router
begin
  begin(Exploration phase)
    perform (quasi) random search until target found
    update map
    build up population
  end(Exploration phase)
  begin(Shuttle phase)
    repeat
      run evolutional optimizer
      travel fittest path
      if obstacle then update map, update population fi
      if not target then search target, update map,
      update population fi
      travel home
    until forever
  end(Shuttle phase)
end
```

3. THE GENETIC ROUTING ALGORITHM

The idea of using probabilistic algorithms to solve combinatorial optimization problems is rather natural. Genetic algorithms have been applied, just to name two, to the traveling salesman problem by Jog et al.[13] and to the MAX-SUM function by Park[14]. The obvious benefits of genetic algorithms in such applications include parallel nature, flexibility, and simplicity. However, the question arises whether genetic algorithms are superior to Dijkstra's algorithm for graph routing in terms of computational complexity. For the related MAX-CLIQUE problem the answer is no.[8] But in our paradigm the problem to solve in each run of the robot is not always a new optimization problem, rather a slightly modified problem, since the target is not expected to

https://doi.org/10.1017/S0263574797000222 Published online by Cambridge University Press

move fast, and a good performance of a genetic algorithm may well be expected.

Following the terminology by Goldberg,[3] we are using a modified simple genetic algorithm (SGA). We will now describe its features.

3.1 Representation of Chromosomes

Most of our testing has been done for $n = 8$. We will denote implementation constants and parameters in typewriter style, and write `SideLength` $= 8$. As pointed out earlier, an individual (chromosome) is a string over A of fixed length `ChromLength` $= 4n$. The factor 4 works well if we do not expect too many obstacles. We keep two populations, the main population, and an interim population for breeding, each of them containing `PopSize` chromosomes. Both are continuously sorted according to the fitness of their members using the quick sort algorithm. Whenever new information about the environment is gathered, the corresponding strings are merged into the main population, replacing one of the weakest individuals. The initial population consists of the strings traveled during the exploration phase, but not more than the $\lceil$`PopSize`$/2\rceil$ most recent ones, the remaining chromosomes are random strings over A.

3.2 Fitness Function

The fitness function assigns to every chromosome a non-negative integral fitness value, and the objective of the optimizer is to minimize this value. If a chromosome c represents an admissible path of length $l(c)$ in the map leading from O to T, then we assign the length of the path $l(c)$ to the fitness value. Note that this distance function is available to the autonomous robot as it is derived from the internal map. If the path does not lead to T, but to $p \in V$ with $m_t(p) \in \{\cdot, ?\}$ instead, we roll a dice. With probability β we assign the value $4n + 2d(p, t)$, favoring chromosomes leading close to the target, and with probability $1 - \beta$ we assign the value $8n - l(c)$, favoring longer admissible paths. Finally, if c leads into an obstacle or if c would lead outside the environment, we assign the lethal fitness value $8n + 1$.

3.3 Offspring production

We have implemented 3 different selection schemes for offspring production. By offspring we mean potential children inheriting genetic material from a particular parent. All the offspring chromosomes of the population are kept for recombination into the interim population.

- Exponential selection. The fittest parent generates $\lceil$`PopSize`$/2\rceil$ chromosomes, the next but fittest generates $\lceil$`PopSize`$/4\rceil$ chromosomes, and so on.
- Rank selection with elitism. The $\lceil 0.8 *$ `PopSize`$\rceil$ fittest individuals generate 1 offspring chromosome each.
- Proportionate selection. The $[0.2 *$ `PopSize`$\rceil$ fittest individuals are considered for offspring production. Those proportionally allocate from 4 down to 1 offspring chromosomes.

Next, the selected offspring chromosomes are used for recombination.

3.4 Recombination and breeding

This step forms the interim population. It contains as many chromosomes as have been produced previously by the offspring production. For each parent chromosome from the offspring, a partner chromosome is selected at random from the offspring. With the probability `CrossRate`, the crossover operator described below is used to create a chromosome in the interim population. If no crossover has to occur, the interim chromosome is a copy of a random parent. Now a portion of the interim population determined by `MuteRate` is exposed to mutation. For those chromosomes a gene, i.e. character in the string, is randomly selected and replaced by a random value. The generational cycle is concluded by breeding, that is by replacing the weaker chromosomes of the main population by the fitter chromosomes of the interim population. Now we will describe the other genetic operators we use, including crossover.

3.5 Genetic Operators

We have implemented only the classic crossover operators, single point crossover, two point crossover, and uniform crossover.[15] With these ingredients the modified simple genetic algorithm worked satisfactorily, and we added a mechanism to monitor its performance by keeping basic statistics. In particular, we monitored the standard deviation of the fitness values of the chromosomes of the main population. This revealed that once some basic overall fitness is achieved, the standard deviation quickly approached zero, indicating a uniform population. Increasing the mutation rate, however, slowed down the finding of reasonable solutions. Instead we introduced a diversification operator, to be applied after breeding. If the standard deviation of the fitness values of the newly formed population is less than 1, then a `DivRate` portion of the population is exposed to mutation. The diversification operator improved the performance significantly, as we will see in the next section.

4. RESULTS

As pointed out earlier, we used an 8×8 environment for most of our testing, and our testing regards the performance of the genetic optimizer in the first case. Several tests using different selection schemes showed that rank selection with elitism worked best, so we kept to this scheme. First we ran one exploration phase for the environment shown in Figure 1. It took 8 (quasi) random walks to find the target T, resulting in 8 chromosomes for the initial population and the map shown in Figure 2.

This data was used in all subsequent tests, for which we also kept the following parameters constant.

- Population size `PopSize` $= 100$ chromosomes,
- Mutation rate `MuteRate` $= 0.08$,

https://doi.org/10.1017/S0263574797000222 Published online by Cambridge University Press

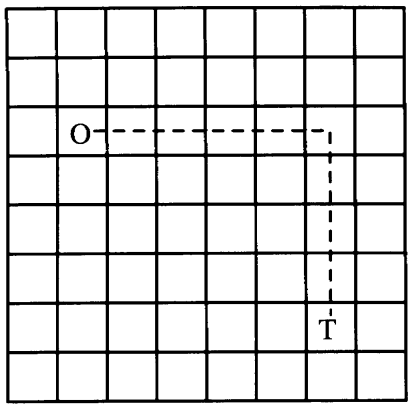

Fig. 1. The empty environment with an optimal path.

- Crossover rate `CrossRate` = 0.99,
- Mutation rate `DevRate` = 0.98 for the diversification operator,
- Timeout `MaxGen` = 10000 generations,
- Probability of favoring paths leading close to the target $\beta = 0.7$,
- Oakenfull mixed congruential random number generator, $x_{n+1} := 16333x_n + 25887 \pmod{2^{15}}$.

The termination condition for the genetic router in all tests was finding an optimal path, the length of which we entered manually according to the environment. The tables below show the average number of generations needed to find an optimal path of 10 tests run for each of the parameter combinations single point crossover/two point crossover, diversification operator/no diversification.

4.1 Test set 1: No obstacles

The first set of tests was performed using the same environment as in the exploration phase, shown in figure 1, where a typical solution is depicted by the dashed line.

n crossover points	diversification	generations
2	no	4.3
2	yes	6.1
1	no	35.7
1	yes	5.0

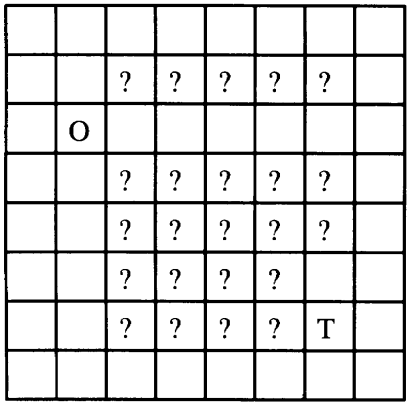

Fig. 2. The map.

In this optimization problem the solution space is rather large, so uniformization of the population is less likely, and the application of the diversification operator does not improve the performance.

4.2 Test set 2: Wall of obstacles

The second set of tests was performed using the environment shown in Figure 3, with obstacles at $(2,5)$, $(3,5)$, $(4,5)$, $(5,5)$, $(6,5)$, and at $(7,5)$. Thus a wall had been inserted since the exploration phase common for all tests. We used the same map that resulted from the exploration phase, and merged the same 8 chromosomes from it into the initial population for each test of this set. For every test we first ran an evolution, which in most cases came up with a path running into an obstacle. In that case we let the robot travel this fittest path, detect the obstacle, merge the resulting admissible path into the population, and we looped back to the evolution, until an optimal path was found or timeout occured. A shortest path has length 11.

- Two point crossover, no diversification operator: Four out of ten tests times out. However, a path of sub optimal length 13 was found quickly. For the 6 tests in which an optimal solution was found, the average total number of generations was 38.5. This parameter combination cannot be regarded as a stable solution to the problem, since timeout occurs frequently.
- One point crossover, no diversification operator: Nine out of ten tests resulted in an optimal solution. The average total number of generations was 3436.8. Again, a path of sub optimal length 13 was found quickly. This parameter combination yields no practicable result at all, since most tests ran out of time.
- Two point crossover, diversification operator: All ten tests resulted in an optimal solution. The average total number of generations was 54.3.
- One point crossover, diversification operator: All ten tests resulted in an optimal solution. The average total number of generations was 57.1.

These tests clearly show that the diversification

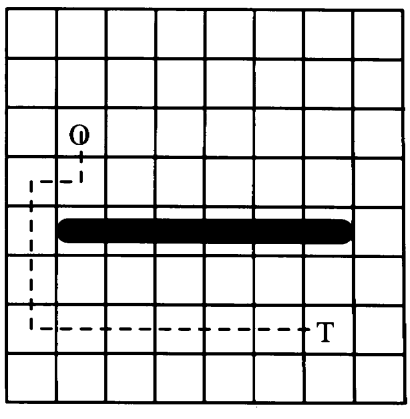

Fig. 3. The environment with obstacles and an optimal path.

https://doi.org/10.1017/S0263574797000222 Published online by Cambridge University Press

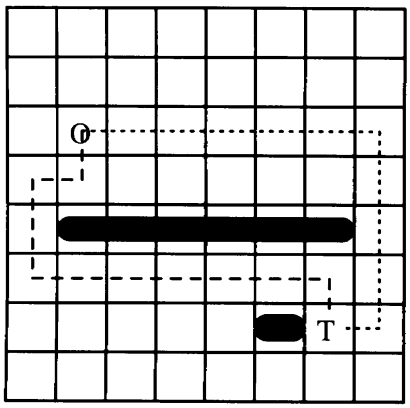

Fig. 4. One more obstacle and two solutions.

operator ensures reliable and fast solutions. Moreover, two point crossover improves the performance only slightly.

4.3 Test set 3: One more obstacle

From the previous test set we kept the map (updated to include the wall) and the chromosomes that resulted from finding the obstacles and one chromosome representing an optimal solution. All other chromosomes were randomized for each test, and for each test we inserted one more obstacle at (6, 7), shown in Figure 4. Then we let the robot travel the optimal solution path, which revealed the new obstacle, and we merged this individual into the population. From this situation we started the different test sets. Two typical solutions are depicted in Figure 4, by the dashed line and the dotted line.

- Two point crossover, no diversification operator: Eight out of ten tests timed out. One solution was found after 5236 generations, the other after 5116.
- One point crossover, no diversification operator: Nine out of ten tests timed out. One solution was found after 58 generations.
- Two point crossover, diversification operator: All ten tests resulted in an optimal solution. The average total number of generations was 3212.8.
- One point crossover, diversification operator: Nine out of ten tests resulted in an optimal solution; one ran out of time. The average total number of generations was 3804.8.

Again, these tests show that the diversification operator ensures reliable and fast solutions.

5. CONCLUSION

We have seen that genetic algorithms are suitable for routing an autonomous robot through a changing environment in a typical real life application, and that the application or rather problem oriented genetical operators improve the performance of such an evolutional optimizer.

We believe that a major improvement of the performance of the genetic router could be accomplished by implementing genetic operators that reflect the structure of the problem. For instance, often a sub-optimal solution was found where the paths contained cycles. This could be avoided by a genetic operator searching for substrings representing cycles, like for instance *lldrru,* and cutting them out.

In an industrial application one might use the algorithm at two levels. First, a coarse cell decomposition of the environment can be chosen to route the robot close to the target. While the robot is approaching the target, the algorithm could run using a fine cell decomposition of the environment to optimize the path near the target.

6. ACKNOWLEDGMENTS

This work has been done at the Laboratory of Measurement and Technology, Linköping University, and at the Department of Technology and Natural Sciences, University of Örebro. The authors would like to thank professor Alexander Lauber for valuable discussions. This project is sponsored by the University of Örebro, Sweden, whose support is gratefully acknowledged.

References

1. H.F. Durrant-Whyte, *Integration, Coordination and Control of Multi-sensor Robot Systems* (Kluwer Academic Publishers, Boston, 1988).
2. W. Van de Velde (ed.), *Toward Learning Robots* (MIT Press, Cambridge, Massachusetts, 1993).
3. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Reading, Mass, 1989).
4. S. Austin, ''An Introduction to Genetic Algorithms'', *AI Expert* **90**(3), 49–53 (1990).
5. S. Forrest, ''Genetic Algorithms: Principles of Natural Selection Applied to Computation'', *Science* **261,** 872–878 (1993).
6. J.H. Holland, ''Genetic Algorithms'' *Scientific American* **267**(1), 44–50 (1992).
7. M. Srinivas and L.M. Patnaik, ''Genetic Algorithms: A Survey'', *IEEE Computer* 17–26 (June, 1994).
8. K. Park and B. Carter, ''On the Effectiveness of Genetic Search in Combinatorial Optimization'' *Proceedings of the 10th ACM Symposium on Applied Computing* (1995) pp. 329–336.
9. J.T. Alander, ''On Robot Navigation Using a Genetic Algorithm'', *Artificial Neural Nets and Genetic Algorithms International Conference* (1993) pp. 471–478.
10. H.S. Lin, J. Xiao and Z. Michalewicz, ''Evolutionary Navigator for a Mobile Robot'', *Proceedings 1994 IEEE International Conference on Robotics and Automation* (1994) pp. 2100–2204.
11. B.G. de Boer, ''An Autonomous Robot Learning Basic Behaviours'', *Tech. Report* (Leiden University, 1994).
12. J. Rudnick and G. Gaspari, ''The Shapes of Random Walks'', *Science* **237,** 384–389 (1987).
13. P. Jog, J. Suh and D. Gucht, ''Parallel Genetic Algorithms Applied to the Travelling Salesman Problem'', *SIAM Journal on Optimization* **27**(2), 515–529 (1991).
14. K. Park, ''A Lower-bound Result on the Power of a Genetic Algorithm'', *Proc. 5th International Conference on Genetic Algorithms* (1993) p. 651.
15. M. Mitchell, ''Genetic Algorithms: Theory and Applications'', *Course material, Short Course, 1994 World Congress on Neural Networks, San Diego* (June 1994).

https://doi.org/10.1017/S0263574797000222 Published online by Cambridge University Press