11.4 Path planning in a mobile robot environment

Navigation is the science (or art) of directing the course of a mobile robot as it traverses the environment. Inherent in any navigation scheme is the desire to reach a destination without getting lost or crashing into any objects.

Often, a path is planned off-line for the robot to follow, which can lead the robot to its destination assuming that the environment is perfectly known and stationary and the robot can track perfectly. Early path planners were such off-line planners or were only suitable for such off-line planning (e.g., [248, 218, 205]). However, the limitations of off-line planning led researchers

to study on-line planning, which relies on knowledge acquired from sensing the local environment [222] to handle unknown obstacles as the robot traverses the environment.

The evolution program that we describe here, i.e., the Evolutionary Navigator (EN), unifies off-line and on-line planning with a simple map of high fidelity and an efficient planning algorithm [243, 244]. The first part of the algorithm (off-line planner) searches for the optimal global path from the start to the destination, whereas the second part (on-line planner) is responsible for handling possible collisions or previously unknown objects by replacing a part of the original global path by the optimal subtour. It is important to point out that both parts of the EN use the same evolutionary algorithm, just with different values of various parameters.

During the last five years other researchers have been experimenting with evolutionary computation techniques for the path planning problem. Davidor [69] used dynamic structures of chromosomes and a modified crossover operator to optimize some real world processes (including robot paths applications). In [355] a genetic algorithm to the path planning problem is described, and in [356] a genetic algorithm for the development of real-time multi-heuristic search strategies is presented. Both approaches assume a predefined map consisting of knot points. Other researchers used classifier systems [414] or genetic programming paradigm [176] to approach the path planning problem. Our approach is unique in the sense that the Evolutionary Navigator (1) operates in the entire free space and does not make any a priori assumptions about feasible knot points of a path, and (2) it combines together off-line and on-line planning algorithms.

Before we explain the algorithm in detail, let us first explain the map structure. In order to support path search in the entire, continuous free space, vertex graphs are used to represent objects in the environment. Currently, we restrict the environment to be two-dimensional with polygonal objects only and motions of the robot to be translational only. Therefore, the robot can be shrunk to a point while the objects in the environment "grow" accordingly [248]. A mobile robot equipped with ultrasonic sensors (e.g., a Denning robot) is assumed for the EN. A known object is represented by the ordered list (clockwise fashion) of its vertices. On-line encountered unknown obstacles are modeled by pieces of "wall", where each piece of "wall" is a straight-line and represented by the list of its two end points. This representation is consistent with the representation of known objects, while it also accommodates the fact that only partial information about an unknown obstacle can be obtained from sensing at a particular location. Finally, the entire environment is defined as a rectangular area.

Now it is important to define paths that the EN generates. A path consists of one or more straight-line segments, with the starting location, the goal location, and (possibly) the intersection locations of two adjacent segments defining the *nodes*. A feasible path consists of feasible nodes; an infeasible path contains at least one infeasible node. Assume there is a path $p = \langle m_1, m_2, \ldots, m_n \rangle$ $(n \geq 2)$, where m_1 and m_n denote start and goal nodes, respectively. A node m_i $(i = 1, \ldots, n-1)$ is infeasible if it is either not connectable to the next node m_{i+1} due to obstacles, or it is located inside (or too close to) some obstacle. We assume that the start and goal nodes are located outside the obstacles, and not too close to them. Note, however, that the start node need not be feasible (it may be not connectable to the next node), whereas the goal node is always feasible. Note also that different paths may have different numbers of nodes.

Now we are ready to go through the EN procedure (Figure 11.1).

```
procedure Evolutionary Navigator
begin
  begin (off-line planner)
    get map
    obtain the task
    perform planning:
      current path := FEG(start, goal)
  end (off-line planner)
  if current path is feasible then
  begin (on-line planner)
  repeat
    move along the current path while
    sensing the environment
    if too close to any object then
    begin
      local_start := current location
      local_goal := next node on the current path
      if the object is new
        then update the object map
        else virtually grow the object
          at the closest spot
      perform planning:
        local_path := NEG(local_start, local_goal)
      update current path
    end
  until (at goal) or (failure condition)
  end (on-line planner)
end
```

Fig. 11.1. The structure of the Evolutionary Navigator

The EN first reads the map and obtain the start and goal locations of the task. Then the oFf-line Evolutionary alGorithm (FEG) generates a near-optimal global path, a piece-wise straight-line path consisting of feasible knot points or nodes. Figure 11.2 shows such a global path generated by FEG. (The filled circle simulates the robot).

As the robot starts to follow the path to move towards the goal, it senses the environment for its proximity to nearby objects, and the oN-line Evolutionary alGorithm (NEG) is used to generate local paths to deal with unexpected

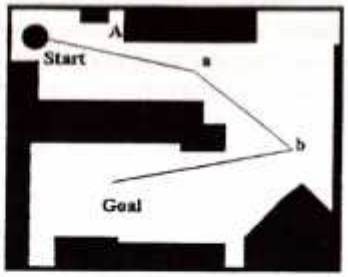

Fig. 11.2. An environment and a global path

collisions and objects. To simulate the effect of unknown objects in the environment, additional data files were created to represent such obstacles (like pieces of "wall" as explained earlier). We experimented with five different sets of unknown objects; Figure 11.3a–d presents the actions of the robot on one of these sets.

When the robot moved too close to the lower left corner of the nearby object 'A', the NEG virtually "grew" 'A' at the spot and generated a local path to steer away from 'A', which was also a piece-wise straight-line path. The robot then followed the current path successfully to reach the point 'a'. While the robot moved from 'a' to 'b', it detected an unknown or new object 'B'. Now the EN updated the map, and again, the NEG generated a local path with the knot point 'd' (Figure 11.3a). As the robot moved from 'd' towards 'b', it became too close to the object 'B': consequently, another local path was generated as represented by the knot point 'e' (Figure 11.3b). The robot then moved from 'd' to 'e' and finally reached the subgoal 'b'. The next step was to move from 'b' towards the goal; as shown in Figure 11.3c, the path segment was too close to the lower right corner of the object 'C'. Therefore, another local path was generated as represented by the knot point 'f' and then to the 'goal'. Figure 11.3d shows the original global path and the actual path traveled. Note that the navigation process terminates when the robot arrives at the goal or a failure condition is reported, i.e., when the EN fails to find a feasible path in certain time period (i.e., within specified number of generations of the NEG).

As we already mentioned, the EN combines off-line and on-line planning with the same data structure and the same planning algorithm. That is, the only difference between FEG and NEG is in the parameters they use: population size pop_size_g, number of generations T_g, maximum length of a chromosome n_g, etc. for FEG, and pop_size_l, T_l, n_l, etc. for NEG. Note that both FEG and NEG do *global planning*; even if NEG usually generates a local path, it operates on the updated *global map*. Moreover, if no object is initially known in the environment, or no initially known object is between the start location and the goal location, then FEG will generate a straight-line path with just two nodes:

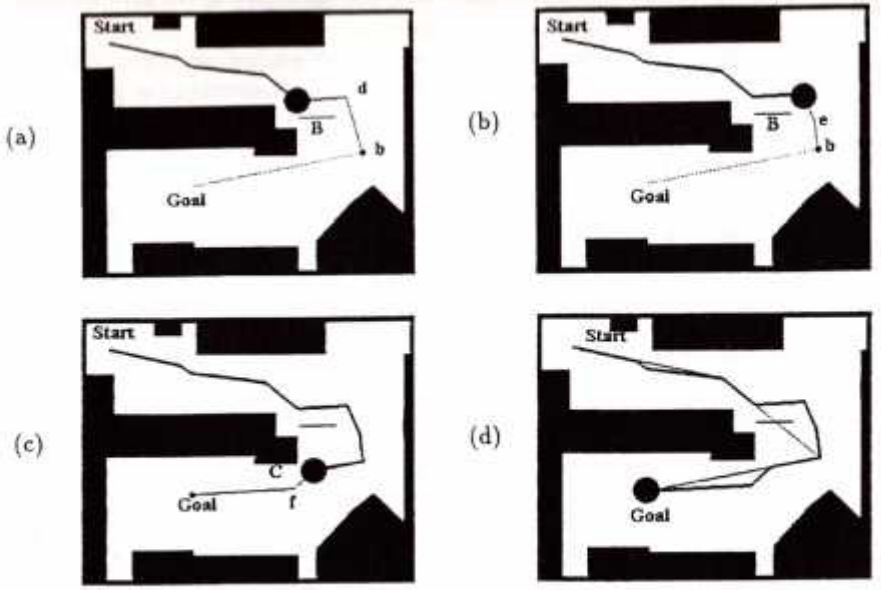

Fig. 11.3. An actual path traveled

the start and the goal locations. It will solely depend on the NEG to lead the robot towards the goal while avoiding unknown obstacles.

In the following, we discuss components of FEG and NEG in detail.

Chromosomes are ordered lists of path nodes as shown in Figure 11.4. Each of the path nodes, apart from the pointer to the next node, consists of x and y coordinates of an intermediate knot point along the path, and a Boolean variable b, which indicates whether the given node is feasible or not.

Fig. 11.4. Chromosome representing a path

The length of the chromosomes (the number of the path nodes represented in a chromosome) is variable. In off-line planning, the maximum length of a chromosome is set to be the number n_g of vertices representing known objects in the environment. It is unlikely that all feasible paths would require a large number of (e.g., n_g) intermediate nodes: even in complex environments a feasible

path might be quite simple. Therefore, we make the length of the chromosomes variable to deal with such situations gracefully.

During the on-line planning, the local path for getting around an obstacle is likely to contain only a small number of nodes, consequently, the parameter n_l, as the maximum length of the chromosome in this phase, is relatively small at the beginning of the local search. However, if the evolution process fails to find a feasible path after some number of generations, the maximum length of the chromosome should grow: in such situation it is quite likely that feasible paths have more complex structures. In the EN system we assumed that the parameter n_l was a function of the current generation number t, more precisely, $n_l(t) = t$.

The initial populations of (pop_size_g for FEG, and pop_size_l for NEG) chromosomes were generated randomly. For each chromosome, a random number was generated from the range $2..max(2, n_g)$ (for the off-line planner) to determine its length. The coordinates x and y were created randomly for each node of such a chromosome (the values of coordinates were restricted to be within the confine of the environment, of course).

For each node of each chromosome, the value of the Boolean variable b is determined (feasibility check). If the node is feasible, its b value is set to TRUE, otherwise, it is set to FALSE. The methods for checking the feasibility of a node (i.e., location validity, clearance from nearby objects, and connectivity) are relatively simple and are based on algorithms described by Pavlidis [310].

The fitness (the total path cost) of a chromosome $p = \langle m_1, m_2, \ldots, m_n \rangle$ is determined by two separate evaluation functions (for feasible and infeasible individuals):

- for a feasible path p:

$$\text{Path\_Cost}(p) = w_d \cdot dist(p) + w_s \cdot smooth(p) + w_c \cdot clear(p),$$

where the weights w_d, w_s, and w_c normalize the total cost of a path, and

  - $dist(p) = \sum_{i=1}^{n-1} d(m_i, m_{i+1})$, where $d(m_i, m_{i+1})$ is the distance between knot points m_i and m_{i+1}; i.e., the function $dist(p)$ returns the total length of the path p.
  - $smooth(p) = \max_{i=2}^{n-1} s(m_i)$, where

$$s(m_i) = \frac{\theta_i}{\min\{d(m_{i-1}, m_i), d(m_i, m_{i+1})\}};$$

    i.e., the function $smooth(p)$ returns the largest curvature of p at a knot point.
  - $clear(p) = \max_{i=1}^{n-1} c_i$, where

$$c_i = \begin{cases} d_i - \tau & \text{if } d_i \geq \tau \\ a(\tau - d_i) & \text{otherwise,} \end{cases}$$

    d_i is the minimum distance between the segment (m_i, m_{i+1}) of the path and all known objects, τ defines a safe distance, and a is a coefficient; i.e., the function $clear(p)$ returns the largest number which measures clearance between all segments of p and the objects.

- for an infeasible path p:

$$\text{Path\_Cost}(p) = \alpha + \beta + \gamma,$$

where α is the number of intersections of the path p with all walls of the objects, β is the average number of intersections per infeasible segment, and γ provides the cost of the worst feasible path in the current population; because of this last variable, any feasible path in the population is better than any infeasible one (see also section 15.3, part C).

Several operators (crossover, two mutations, insertion, deletion, smooth, and swap) were included in the FEG and NEG. We discuss them in turn.

Crossover. This operator is similar to the classical one-point crossover widely used in genetic algorithms. It recombines "good" parts of the paths present in both parents to produce hopefully better path represented by the offspring. Two selected chromosomes are cut in some positions and glued together: the first part of the first chromosome with the second part of the second chromosome, and the first part of the second chromosome with the second part of the first chromosome. However, the crossing points in both chromosomes are not selected randomly: if infeasible nodes are present in the chromosome, the crossing points fall after one of them.

Mutation_1. This mutation is responsible for fine tuning values of coordinates of the nodes listed in the chromosome. If a node of a chromosome is selected for this mutation, its coordinates are modified. For example, the coordinate $x \in \langle a, b \rangle$ (as well as coordinate y) is changed in the following way:

$$x' = \begin{cases} x - \delta(t, x - a), & if \ \ r = 0 \\ x + \delta(t, b - x), & if \ \ r = 1 \end{cases}$$

where r is a random bit, and the function $\delta(t, z)$ returns a value in the range of $[0..z]$ such that the probability of $\delta(t, z)$ being close to zero increases as t increases (t is the current generation number of the evolution process). The operator is modeled on non-uniform mutation used in evolutionary systems for nonlinear optimization (Chapter 7). This mutation is responsible for "smoothing over" the shape of the path.

Mutation_2. This mutation is useful in cases when a larger change in a value is required (this situation occurs often during the on-line planning phase, when an obstacle is blocking the path). If a node of a chromosome is selected for this mutation, its coordinates are modified. For example, the coordinate $x \in \langle a, b \rangle$ (as well as coordinate y) is changed in the following way:

$$x' = \begin{cases} x - \Delta(t, x-a), & if \ r = 0 \\ x + \Delta(t, b-x), & if \ r = 1 \end{cases}$$

where r is a random bit, and the function $\Delta(t, z)$ returns a value in the range of $[0..z]$ such that the probability of $\Delta(t, z)$ being close to z increases as generation number t increases.

Insertion. This operator inserts a new node into the existing path; every place between two nodes has the same probability of such insertion.

Deletion. This operator deletes a node from the path; every node has the same probability for such deletion.

Smooth. This operator smooths a part of the path by cutting sharp turns. For selected knot point m_i (with a high curvature), the operator selects two new knot points k_1 and k_2 (from segments (m_{i-1}, m_i) and (m_i, m_{i+1}), respectively), inserts them into the path, removes m_i; so it creates a new path p':

$$p' = (m_1, \ldots, m_{i-1}, k_1, k_2, m_{i+1}, \ldots, m_n).$$

Swap. This operator splits the selected chromosome into two parts (the splitting point is determined at random) and swaps these parts.

Based on the preliminary experimental results, the EN has proved to be efficient and effective in comparison with navigators using traditional approaches (e.g., [130]). Results of the current version of the system on two different environments are presented in Figures 11.5 and 11.6.

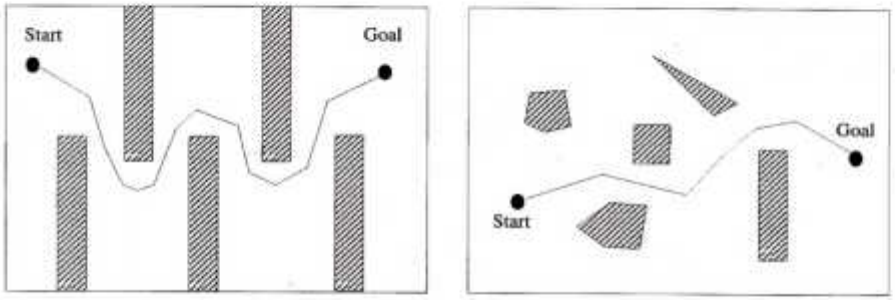

Fig. 11.5. Results of the EN on two environments

Of course, there is a need to explore the EN's potential by conducting more tests under different environments, most importantly, by implementation of the EN on a real robot. At the same time, several issues of such evolutionary navigators remain to be resolved; these include (1) design of smarter termination conditions for FEG and NEG to better realize the optimization goals (currently the algorithms terminate either when a feasible path is found or some fixed number of generations have elapsed), (2) introduction of adaptive frequencies

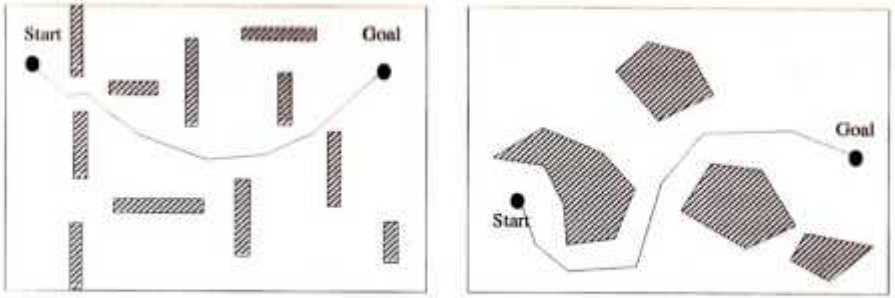

Fig. 11.6. Results of the EN on another two environments

of the genetic operators, as opposed to the constants in the current version of the system (this modification should enhance the performance of the system and is based on the simple observation that different operators may play different roles at different stages of the evolution process), (3) extension of the EN to operate in an environment with non-polyhedral objects, (4) incorporation of the knowledge of the current stage of the search into workings of operators (e.g., it might be more meaningful to cross two paths at infeasible knot points), and (5) exploration of some learning mechanism so that the EN can take advantage of past experiences.

Despite its efficiency and effectiveness in many cases, however, the EN has a major limitation: it assumes that a feasible and sufficiently good actual path can be obtained by minor perturbation from the current best path; the system is not designed to be able to replace the current global path, at some stage of the traversal, by another (possibly better) global path entirely. Thus it might be worthwhile to experiment with other solutions; for example, an adaptive navigator (AN) is currently under construction. Unlike the EN, which consists of off-line and on-line planners, an adaptive navigator would be an on-line planner completely; it would constantly adapt the path connecting the current location of the robot and the goal based on newly gathered sensing information.

11.5 Remarks

In this final section we discuss briefly a few relatively recent applications of evolutionary techniques, which, for various reasons, are interesting (from the perspective of constructing an evolution program). We discuss them in turn.

There are some applications (e.g., network design problems), in which a solution is a graph. The problem of representing graphs in genetic algorithms is quite interesting as such. Recently, Palmer and Kershenbaum [304] reported on experiments with various ways of representing trees. They identified desirable properties for a good representation; these include: