G3.11 Evolutionary planner/navigator in a mobile robot environment

Jing Xiao

University of North Carolina - Charlotte

Abstract

Based on evolutionary computation concepts, the Evolutionary Planner/Navigator (EP/N) represents a new approach to path planning and navigation. The major advantages of the EP/N include being able to achieve both near-optimality of paths and high planning efficiency, being able to accommodate different optimization criteria, being flexible to changes, and being robust to uncertainties. The EP/N unifies off-line planning and on-line planning/navigation processes with the same evolutionary algorithm to deal with unknowns in an environment gracefully and flexibly. It provides high safety for the robot without requiring complete information about the environment.

G3.11.1 Project overview

The *motion planning* problem for mobile robots is typically formulated as follows (Yap 1987): given a robot and a description of an environment, plan a path of the robot between two specified locations, which is collision-free and satisfies certain optimization criteria. Traditionally there are two approaches to the problem: off-line planning, which assumes perfectly known and stable environment, and on-line planning, which focuses on dealing with uncertainties when the robot traverses the environment. On-line planning is also referred to by many researchers as the *navigation* problem[1].

A great deal of research has been done in motion planning and navigation (see Yap 1987 and Latombe 1992 for surveys). However, different existing methods encounter one or many of the following difficulties:

- high computation expenses,
- inflexibility in responding to changes in the environment,
- inflexibility in responding to different optimization goals,
- inflexibility in responding to uncertainties,
- inability to combine advantages of global planning and reactive planning.

The EP/N system was developed to address these difficulties; the inspiration to use evolutionary techniques was triggered by the following ideas/observations:

- randomized search can be the most effective in dealing with NP-hard problems and in escaping local minima,
- parallel search actions not only provide great speed but also provide ground for *interactions* among search actions to achieve even greater efficiency in optimization,
- creative application of the evolutionary computation *concept* rather than dogmatic imposition of a standard algorithm proves to be more effective in solving specific types of real problems,
- intelligent behavior is the result of a collection of simple reactions to a complex world,

[1] Although some researchers also interpret *navigation* as a low-level control problem for path-following, we do not use such an interpretation here.

- a planner can be greatly simplified, much more efficient and flexible, and increase the quality of search, if search is not confined to be within a specific map structure,
- it is more meaningful to equip a planner with the flexibility of changing the optimization goals than the ability of finding the absolutely optimum solution for a single, particular goal.

The EP/N embodies the above ideas by following the evolution program approach, i.e. combining the concept of evolutionary computation with problem-specific chromosome structures and genetic operators (Michalewicz 1994). With such an approach, the EP/N is pursuing all the advantages as described above. Less obvious though, is that with the unique design of chromosome structure and genetic operators, the EP/N does not need a discretized map for search, which is usually required by other planners. Instead, the EP/N "searches" the original and continuous environment by generating paths based on evolutionary computation. The objects in the environment can simply be indicated as a collection of straight-line "walls." This representation accommodates both known objects as well as partial information of unknown objects obtained from sensing. Thus, there is little difference between off-line planning and on-line navigation for the EP/N. In fact, the EP/N unifies off-line planning and on-line navigation with the same evolutionary algorithm and chromosome structure.

The structure of the EP/N is shown in Figure G3.11.1, where FEG — the oFf-line Evolutionary alGorithm, and NEG — the oN-line Evolutionary alGorithm — are essentially the same evolutionary algorithm as to be described. The only difference between FEG and NEG is in certain values of parameters (see Section G3.11.5) one may choose. The different parameter values are to accommodate slightly different objectives of FEG and NEG: FEG emphasizes the optimality of a path while NEG emphasizes the swiftness in generating a feasible path. Note that both FEG and NEG do global planning, and NEG generates an alternative subpath by global planning based on the updated knowledge of the environment obtained from sensing. Moreover, if no object is initially known in the environment, then FEG will generate a straight-line path with just two nodes: the start and the goal locations. It will solely depend on the NEG to lead the robot towards the goal while avoiding unknown or newly emerged obstacles.

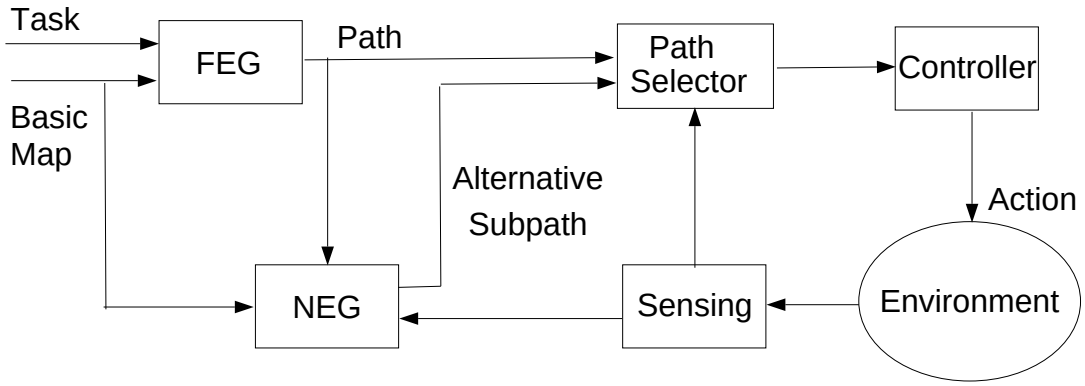

Figure G3.11.1. The EP/N structure

G3.11.2 Design process

We now describe the evolutionary algorithm which both FEG and NEG adopts in detail.

Chromosomes and initialization

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 only feasible nodes. An unfeasible path has at least one unfeasible node which is either not connectable to the next node on the path due to obstacles or it is located inside some obstacle.

Chromosomes are represented as ordered lists of path nodes: each node, apart from the pointer to the next node, consists of x and y coordinates of the knot point and a state variable b, which indicates whether or not the node is feasible (Figure G3.11.2). Each chromosome can have varied number of nodes, which provides great flexibility. The methods for checking the feasibility of a node (i.e., location validity and connectivity) are relatively simple and are based on algorithms described in Pavlidis (1982).

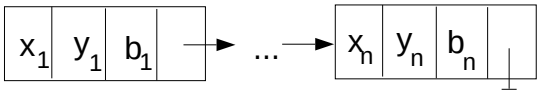

Figure G3.11.2. A chromosome representing a path

The initialization of chromosomes is a random process subject to the following input parameters: a population size P and the maximum number of nodes in a chromosome N. For each chromosome, a random number is generated within $[2, N]$ to determine its length, i.e., the number of nodes. The coordinates x and y are also created randomly for each node of such a chromosome within the confine of the environment. P chromosomes are generated in this way.

Evaluation

The evaluation function Path_Cost(p), measures the path cost of a chromosome p. Since p can be either feasible or unfeasible, we adopt two separate evaluation functions $eval_f$ and $eval_u$ to handle the feasible and unfeasible cases respectively. Our design of the evaluation function Path_Cost(p) has gone through a long process of development, as will be discussed in Section 3.11.3. The $eval_f$ and $eval_u$ to be described are the most recent results of such development.

It seems to be relatively easy to compare two feasible paths. Intuitively, we think $eval_f$ should be a function of the total length of a path *dist*, its smoothness *smooth* and the clearance *clear* between the path and the surrounding obstacles. There can be many ways to define the function $eval_f$. At present, we simply define it as the linear combination of *dist*, *smooth*, and *clear*:

$$eval_f(p) = w_d \cdot dist(p) + w_s \cdot smooth(p) + w_c \cdot clear(p)$$

where the constants w_d, w_s, and w_c represent the weights on the total cost of the path's length, smoothness, and clearance, respectively. We define *dist*, *smooth*, and *clear* as the following:

- $dist(p) = \sum_{i=1}^{n-1} d(m_i, m_{i+1})$, the total length of the path, where $d(m_i, m_{i+1})$ denotes the distance between two adjacent path nodes m_i and m_{i+1}.
- $smooth(p) = \max_{i=2}^{n-1} s(m_i)$, the maximum "curvature" at a knot point, where "curvature" is defined as
$$s(m_i) = \frac{\theta_i}{\min\{d(m_{i-1}, m_i), d(m_i, m_{i+1})\}}$$
and $\theta_i \in [0, \pi]$ is the angle between the extension of the line segment connecting nodes m_{i-1} and m_i and the line segment connecting nodes m_i and m_{i+1} (Figure G3.11.3).
- $clear(p) = \max_{i=1}^{n-1} c_i$, where
$$c_i = \begin{cases} g_i - \tau & \text{if } g_i \geq \tau \\ e^{a(\tau - g_i)} - 1 & \text{otherwise,} \end{cases}$$
g_i is the smallest distance from the segment $\overline{m_i m_{i+1}}$ to all detected objects, τ is a parameter defining a "safe" distance, and a is a coefficient.

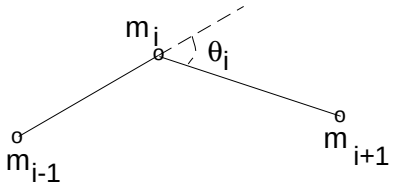

Figure G3.11.3. θ_i at each node m_i

With this formulation, our goal is to minimize the function $eval_f$.

We took into account several factors in the design of $eval_u$: the number of intersections of a path with obstacles, the depth of intersection (i.e., how deep a path cuts through obstacles), the ratio between the numbers of feasible and unfeasible segments, the total lengths of feasible and unfeasible segments, and so on, and implemented two designs for $eval_u$.

One design of $eval_u$ is as follows:

$$eval_u(q) = \mu + \eta$$

where μ is the number of intersection of a whole path with obstacles, η is the average number of intersections per unfeasible segment. With this evaluation function, the path costs of the three paths from Figure G3.11.4 are:

$$eval_u(p) = 2 + 2 = 4$$
$$eval_u(q) = 4 + 2 = 6$$
$$eval_u(r) = 4 + 4 = 8$$

which match our intuition: path p is the easiest one to generate a feasible offspring, and path q is much more promising than the path r. This $eval_u$, however, may not be perfect, since q could be considered the best among the three paths (i.e., it should have the lowest cost) from a different perspective.

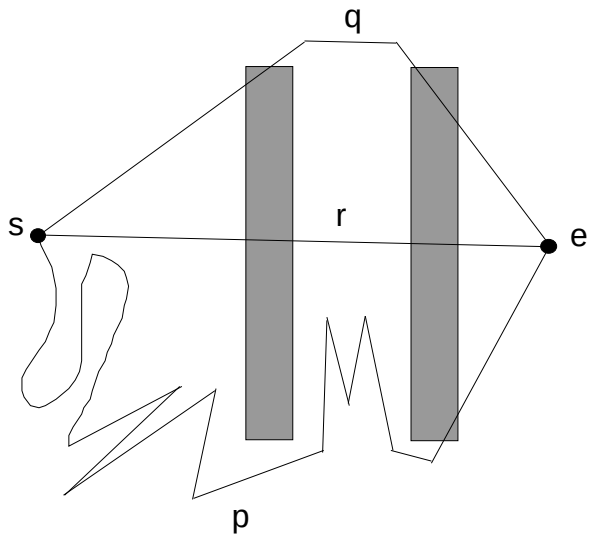

Figure G3.11.4. Three unfeasible paths p, q, and r

The other design makes $eval_u$ equal to the summation of all the penetration distances, where a penetration distance D is defined as the minimum distance to move an unfeasible path segment out of an obstacle it penetrates. This design is reasonable in almost all cases but is more computationally expensive than the first design.

Associated with this approach of designing $eval_u$ independent of $eval_f$ is the issue of how to compare feasible paths against unfeasible ones. This issue requires the answer of the following question:

Is *any* feasible solution better than *any* unfeasible one?

In the EP/N, we have chosen the (somewhat risky) answer 'yes', which makes such comparisons relatively easy for us and is also consistent with our designs of $eval_u$. With this choice, we basically add to the value of $eval_u$ of any unfeasible path p a constant ρ (within a given generation of the evolutionary process) to make the path less attractive than a feasible one:

$$\rho = \max\{0, \max_{p \in F}\{eval_f(p)\} - \min_{q \in U}\{eval_u(q)\}\},$$

where F and U denote the sets of feasible and unfeasible paths respectively. Note that ρ measures the difference between the worst feasible and the best unfeasible paths.

In actual implementation, we do not really compute ρ, instead, we simply sort the feasible paths and unfeasible paths separately from the best to the worst based on their separate evaluation functions. Then, we "append" the sorted list of unfeasible paths at the tail of the sorted list of feasible paths.[2]

Genetic operators

The current version of EP/N uses eight types of genetic operators to evolve chromosomes into possibly better ones. These operators are sufficient to generate an *arbitrary* path, but may not all be needed in all situations. The application of each operator is controlled by a probability. How to select the best combination of operators, i.e., how to determine those probabilities, very much depend on environmental characteristics and specific constraints imposed on a task. Our current version of EP/N is able to feedback how useful an operator is, which helps us in determining the probabilities. However, more research is needed (see G3.11.5). From our current experience on fairly complex environments, the EP/N system performed the best with all eight types of operators present with considerable probabilities (e.g., in the range 0.5–0.9).

Now we introduce each types of operators, which are also illustrated in Figure G3.11.5:

Crossover: it recombines two (parent) paths into two new paths. The parent paths are divided randomly into two parts respectively and recombined: the first part of the first path with the second part of the second path, and the first part of the second path with the second part of the first path. Note that there can be different number of nodes in the two parent paths.

Mutation_1: it is used for fine tuning node coordinates in a path for shape adjustment.

Mutation_2: it is used for large change of node coordinates in a path.

Insertion: it inserts new nodes into a path.

Deletion: it deletes nodes from a path.

Swap: it swaps the coordinates of selected adjacent nodes in a path.

Smooth: it smooths turns of a feasible path by "cutting corners", i.e., for a selected node, the operator inserts two new nodes on the two path segments connected to that node respectively and deletes that selected node.

Repair: it repairs an unfeasible segment in a path by "pulling" the segment around its intersecting obstacles.

Note that we deliberately left out details on how exactly nodes were *selected* and *changed* in many operators, since such decisions could be made in various ways from purely random to incorporating much heuristic knowledge. In the earlier EP/N, such decisions were made mostly randomly. The current EP/N is equipped with versions of operators using more knowledge. For example, it has two versions of **Mutation_1**. The first version changes the coordinates of a node randomly within some bounds which decrease as evolution proceeds; it applies to any path. The second version, however, applies to only a feasible path, and it changes the coordinates of a (feasible) node randomly within some local clearance of the path so that the path remains feasible afterwards.

[2] This works with any ranking selection.

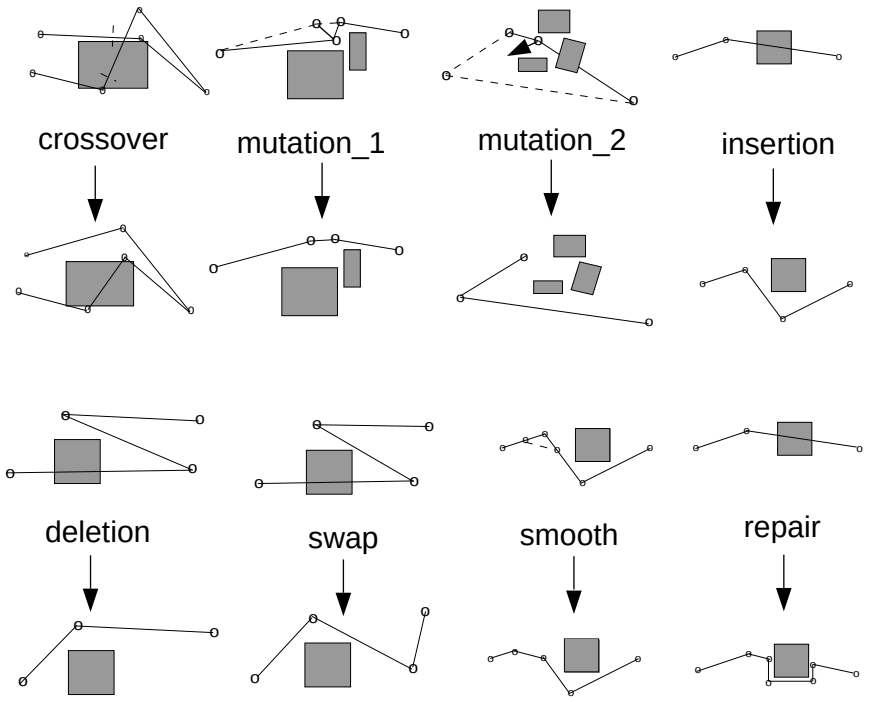

Figure G3.11.5. The roles of the genetic operators

Both versions select nodes randomly. The merits of different types and versions of operators will be further discussed in G3.11.3.

Reproduction

For the selection process, a population of P chromosomes are first sorted based on their fitness values (i.e., cost values) from the best to the worst, and a roulette wheel of P slots is then produced with the ith slot sized proportional to the fitness value of the ith chromosome (Michalewicz 1994, also see Section C2.2). By "spinning" the wheel, the chromosomes which have better fitness values (i.e., lower cost values) will have better chances to be selected for reproduction.

In order to be more efficient, in a later version of the EP/N, we adopted a fixed roulette wheel of P slots with linearly decreasing slot sizes instead of generating a different roulette wheel at each generation. In this way, the chance for a chromosome to be selected is not necessarily proportional to its fitness value but is still better than the chance for a worse chromosome to be selected.

Generally, a parameter $S \leq P$ determines the number of chromosomes to be selected for reproduction. At generation t, the selected S chromosomes from the population $P(t)$ are altered by the genetic operators to generate S offsprings. The S offsprings plus the $P - S$ best chromosomes in the original population $P(t)$ form the next generation of population $P(t + 1)$.

In our latest version of EP/N, only one genetic operator is used at each generation, and $S = 1$ (or 2 if the operator chosen is **crossover**). The selection of operators is also based on a roulette wheel with slots sized proportional to the probabilities of the operators. Note that in this version, the time period for a single generation is the shortest.

G3.11.3 Development and implementation

The development of the EP/N is an ever living "evolution" process itself: different ideas have been experimented and many improvements have been made since the earliest version, but there are yet many new ideas and features that can be incorporated in the EP/N system (see G3.11.5). Instead of seeking a complete product, we see the EP/N more as representing a new direction, along which there are many new hopes but also new challenges, and a new framework, under which these new

hopes, in terms of new ideas and strategies, can be explored and tested, and the new challenges can be dealt with. Indeed, we have already discovered a mixture of hopes and challenges so far.

Development of fitness function

The earliest version of the EP/N (Lin 1993, Lin *et al.* 1994) can be characterized as having a single fitness criterion and a simple penalty function. Only the shortest distance criterion was used[3]: the path cost was simply the length of the path, and a path was better than another one if it was shorter. Unfeasible paths were penalized by adding large penalty constants to their costs, making their lengths exceedingly long.

Such treatment hampered the ability of the EP/N to work well in difficult environments because of its many drawbacks. First, the shortest path may not be safe, i.e., sufficiently away from obstacles, and it may not be more efficient than a longer path if it is not smooth. For example, in Figure G3.11.6, the path q is longer than p but is obviously better. Hence, we changed the evaluation (or fitness) function to include factors of clearance and smoothness. We experimented with various ways of defining the fitness function and encountered the problem of how to evaluate the fitness of an unfeasible path, for which clearance and smoothness do not make much sense. This investigation deepened our understanding to the problems introduced by using simple penalties to discriminate against unfeasible paths.

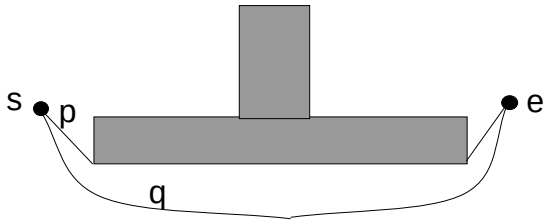

Figure G3.11.6. The longer path is better

The major problem with using a simple penalty function is that it does not provide a reasonable basis for comparing two unfeasible paths, since the merit of a path is not merely reflected by its length and what makes one feasible path better than another simply may not be applicable to the comparison of two unfeasible paths[4]. For example, in Figure G3.11.7, path p has the shortest distance (a straight line) and a perfect smoothness, if smoothness is counted. The other path q has longer distance and worse smoothness. Thus, with the same constant penalty on both paths, p will be ranked better than q, although it seems that q is actually better in the sense that q can be mutated into a feasible path relatively easily.

One may ask what if we simply eliminate unfeasible paths altogether and only evolve the feasible ones. Unfortunately, in our problem, except for cases with very simple environments which have only a few obstacles, the randomly generated initial population usually consists of unfeasible paths only, and since the feasible solution space is nonconvex and with complex boundary depending on obstacles, it is often more difficult to produce/reproduce only feasible paths than to deal with unfeasible ones. Therefore, evaluating unfeasible paths is extremely important and almost inevitable. Another important incentive of evolving unfeasible paths is that it can speed up the search for the optimum solution by providing "shortcuts" across the unfeasible solution space.

Hence, our investigation results in the current solution of evolving feasible and unfeasible paths by separate fitness functions as described in G3.11.2.

Development of operators

Initially, the first six operators were used in the EP/N. Different schemes and probabilities of applying those operators were experimented, and the effects of the operators were investigated. We

[3] This was just as in so many traditional approaches to path planning.
[4] In fact, as we do not even consider comparing two unfeasible paths in our daily lifes, we have much less intuition to help us than in the case of comparing two feasible ones

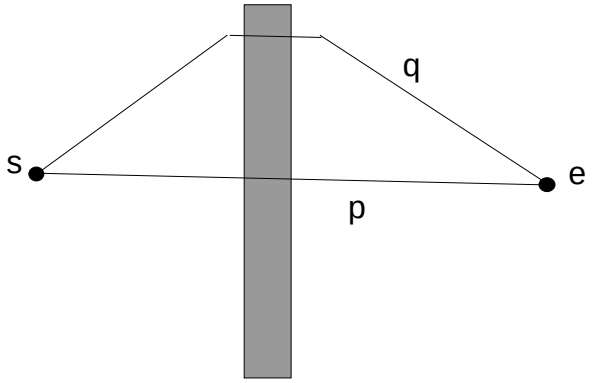

Figure G3.11.7. Two unfeasible paths

later added the **Smooth** operator to improve feasible paths. We tested the operators in different environments and found that for complex planning tasks in certain complex environments, purely random operators did not work very well. This led us to design operators using more knowledge about the environment. The **Repair** operator was introduced using knowledge of obstacles. In fact, we found that much of the knowledge needed by more "intelligent" operators had already been made available during evaluation of path fitness. In the latest version of EP/N, we added new versions for **Mutation_1** (as explained in G3.11.2 about genetic operators), as well as for **Deletion** and *Smooth* using such knowledge.

Our experience showed that **Repair** was highly effective in generating feasible paths. **Smooth** and the more "intelligent" version of **Mutation_1** were highly effective in improving feasible paths. **Crossover** was consistently effective in evolving both unfeasible and feasible paths. This was particularly important since **Crossover** was completely random. Its simplicity also seemed to speed up considerably the evolution process. **Deletion** was the only operator which removed nodes from a path and thus was highly effective in keeping the EP/N system efficient (in time and space) and allowing other operators to be active. It seemed that the combined effort of different operators generally worked better in complex situations. As mentioned in G3.11.2, how to determine the best combination of operators (i.e., probabilities) is not a trivial issue and is definitely one of the major future research topics (see G3.11.5).

Implementation

The earlier versions of the EP/N program were run on 486 or Pentium PCs. The later versions of the EP/N were run under unix in Sun SparcStations. No commercial EA tools were used.

G3.11.4 Results

In Figures G3.11.8– G3.11.11, we present some off-line planning results obtained from running the latest version of the EP/N system on a Sun Sparc 20 in different environments with the same set of parameter values as the following:

- Probabilities of application for operators **Crossover**, **Mutation_1**, **Mutation_2**, **Insertion**, **Deletion**, **Swap**, **Smooth**, and **Repair** are 0.6, 0.8, 0.5, 0.5, 0.5, 0.5, 0.9, and 0.8 respectively.
- Population size is 30.
- Coefficients w_d, w_s, w_c, a, τ in the evaluation function $eval_f(p)$ are 1.0, 1.0, 1.0, 7.0, 10 respectively.

Snapshots were taken at four different states, indicated by four different values of generation index T, of evolution for each task/environment, where two-third of the population were displayed at states (a)'s and (b)'s, and only the best path was displayed at states (c)'s and (d)'s. Despite the fact that the parameter values were chosen rather arbitrarily and the same "one size fit all" values were applied to different environments with no individual adjustment, the EP/N system performed quite well as clearly shown by the results. Especially noteworthy is the efficiency the EP/N demonstrated

in finding feasible paths as shown in states (b)'s and the near-optimal paths as shown in states (c)'s. From states (c)'s to states (d)'s, however, the pace of evolution was much slowed as expected.

G3.11.5 Conclusions

The EP/N represents a promising new approach in robot planning which is full of potential and a new application of evolutionary computation concepts which is full of interesting challenges. The EP/N is remarkably robust despite of imperfections in the design of evaluation functions, the design and application (i.e., probabilities) of genetic operators, etc. It confirms the nature and advantage of an evolutionary system.

One important issue in future research is how to further use domain knowledge (i.e., specific environmental knowledge) effectively in the EP/N system to improve performance. Although in our latest version of the EP/N, we incorporated domain knowledge in both fitness evaluation and genetic operators, there are other components/processes, such as initialization process and determination of parameter values, which may benefit from domain knowledge. For example, rather than random initialization, an initial population may consist of (a) a set of paths created by mutating or repairing the shortest path between start and goal locations, (b) some mixture of chromosomes having randomly-generated coordinates and chromosomes having coordinates with "problem-specific knowledge" as obtained from (a).

It is highly desirable to make the EP/N capable of adapting its parameter values based on domain knowledge and the states of evolution. Currently, all operators of the EP/N have constant probabilities of application, which are fixed at the beginning of an evolution process. However, different operators may have different impacts (roles) at different stages of the evolution process due to different situations encountered in an environment. For example, in on-line navigation, if the robot follows the current best path without running into any unexpected obstacles, the significance of **Mutation_1** should grow, whereas the probability of **Mutation_2** should be kept at the minimum level. On the other hand, if the robot is trapped in some location of the environment (e.g., surrounded by previously unknown obstacles), the probability of **Mutation_2** should increase; at the same time the significance of **Mutation_1** could shrink. While the role of **Repair** should be very significant at early stage of evolution, the role of **Smooth** should become more significant at the later stage when the population consists of more feasible chromosomes. Similar observations and comments could be made for the other parameters.

Another important issue is to improve the organization of the EP/N to stress adaptability and learning for on-line navigation. For example, instead of generating a subpath for the robot to "get around" an obstacle as it is the case in the current version, the NEG may simply generate an alternative path for the robot to reach its goal, where the path is based on the "past experience", which can be a pool of feasible paths obtained previously. It could also be interesting to study other forms of "memory", such as one based on multi-chromosome structures with a dominance function (Goldberg 1989) or one employing machine learning techniques.

Acknowledgement

The author would like to thank Zbigniew Michalewicz and Lixin Zhang for their important contribution to the improvement and implementation of the latest version of the EP/N.

References

Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

Latombe, J.C., *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

Lin, H.-S., "Dynamic Path Planning for a Mobile Robot Using Evolution Programming", Master Thesis, UNCC, 1993.

Lin, H.-S., Xiao, J., and Michalewicz, Z., "Evolutionary Navigator for a Mobile Robot," Proc. IEEE Int. Conf. Robotics & Automation, San Diego, May 1994, pp. 2199-2204.

Lin, H.-S., Xiao, J., and Michalewicz, Z., "Evolutionary Algorithm for Path Planning in Mobile Robot Robot Environment," Proc. First IEEE Conference on Evolutionary Computation (part of the IEEE World Congress on Computational Intelligence), Orlando, Florida, June 1994, pp. 211-216.

Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*. Second, extended edition, Springer, 1994.
Pavlidid, T., *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.
Yap, C.-K., "Algorithmic Motion Planning", *Advances in Robotics, Vol.1: Algorithmic and Geometric Aspects of Robotics*, J.T. Schwartz and C.-K. Yap Ed., Lawrence Erlbaum Associates, 1987, pp. 95-143.

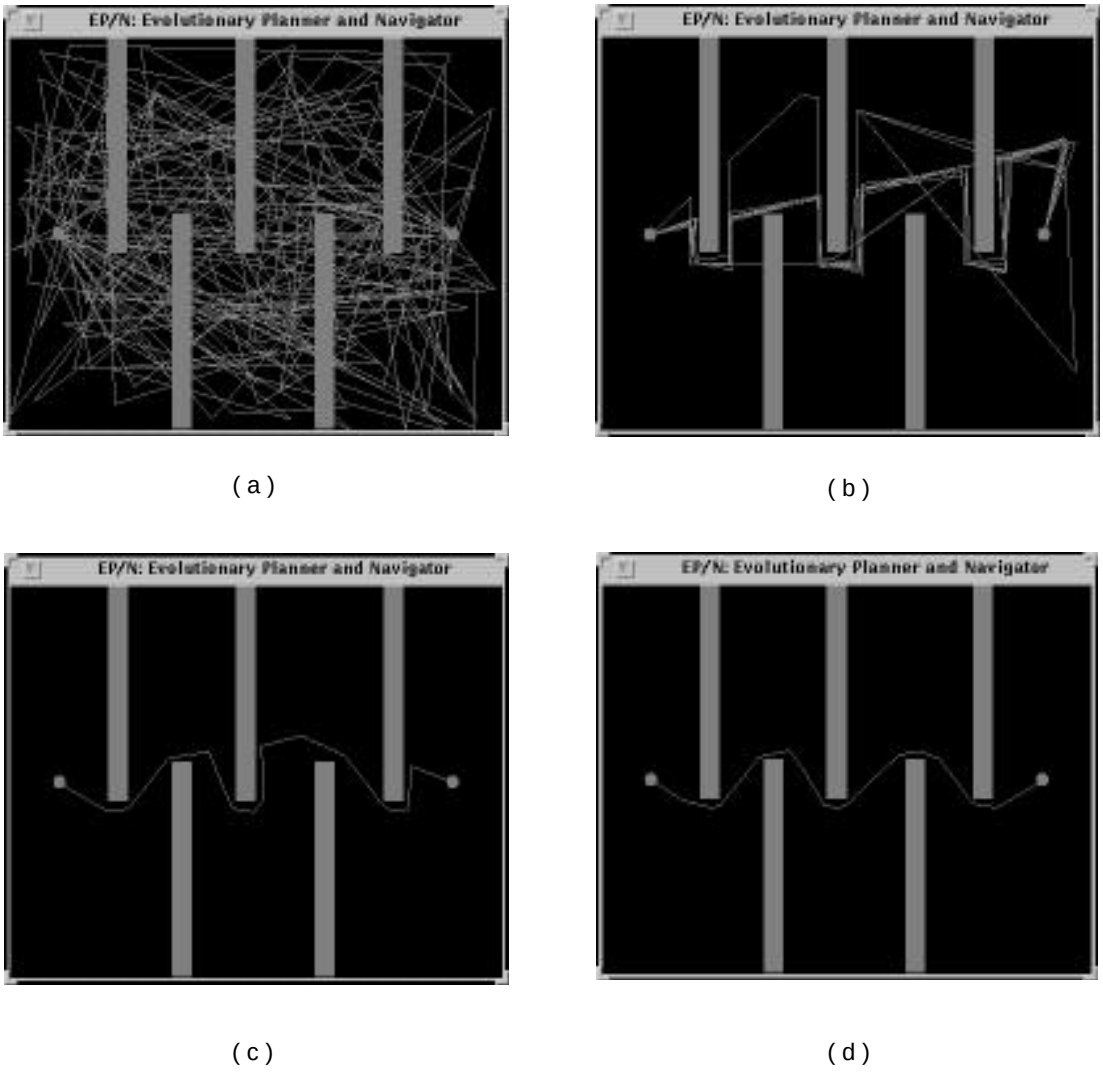

Figure G3.11.8. (a) $T = 0$: paths are generated randomly.
(b) $T = 100$: evolution has taken 0.91 seconds.
(c) $T = 600$: evolution has taken 14.67 seconds; the best path has 25 nodes and a cost of 630.19.
(d) $T = 1000$: evolution has taken 28.16 seconds; the best path has 20 nodes and a cost of 598.62.

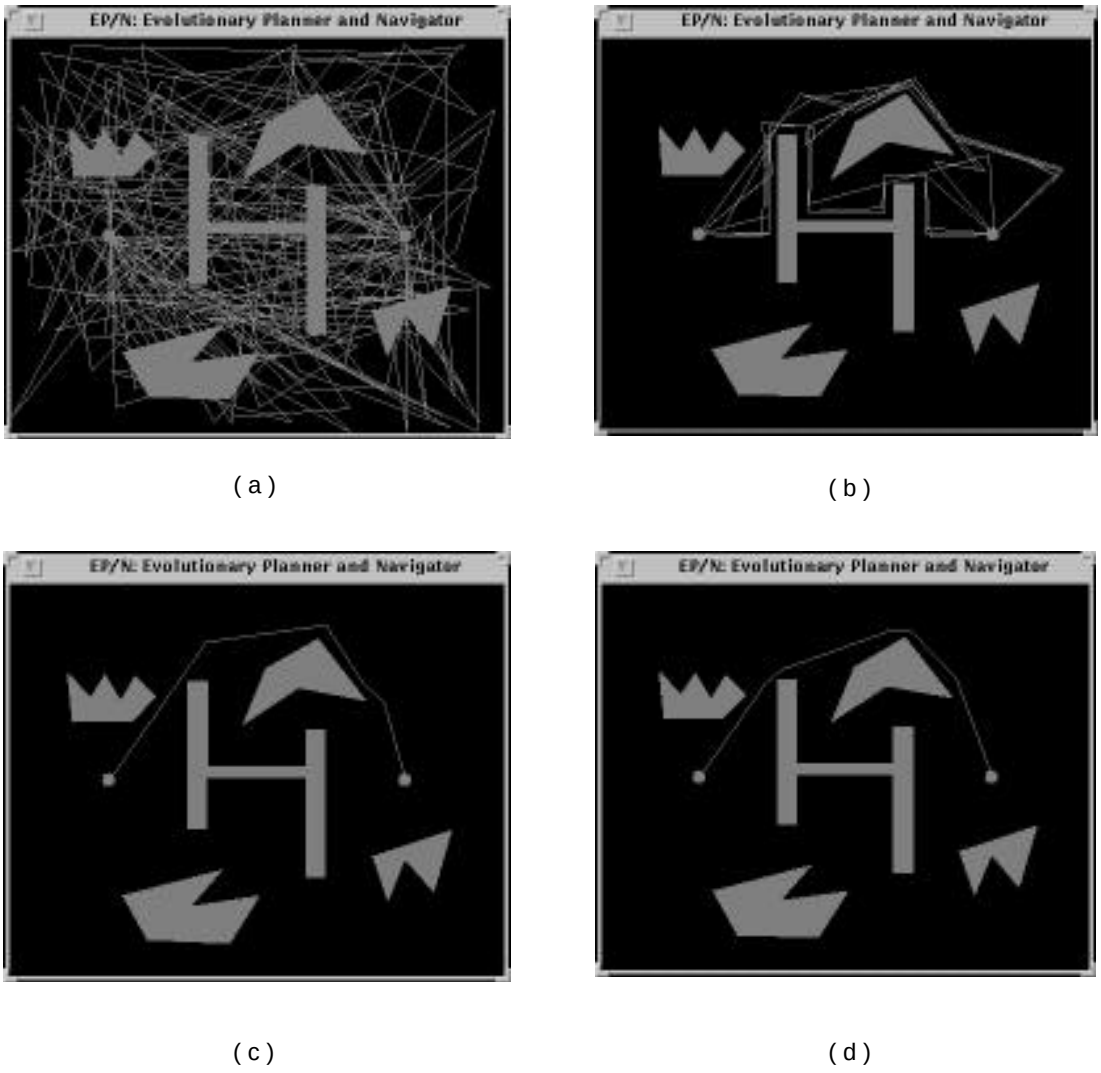

Figure G3.11.9. (a) $T = 0$: paths are generated randomly.
(b) $T = 150$: evolution has taken 2.13 seconds.
(c) $T = 300$: evolution has taken 3.92 seconds; the best path has 4 nodes and a cost of 483.95.
(d) $T = 500$: evolution has taken 7.06 seconds; the best path has 5 nodes and a cost of 473.88.

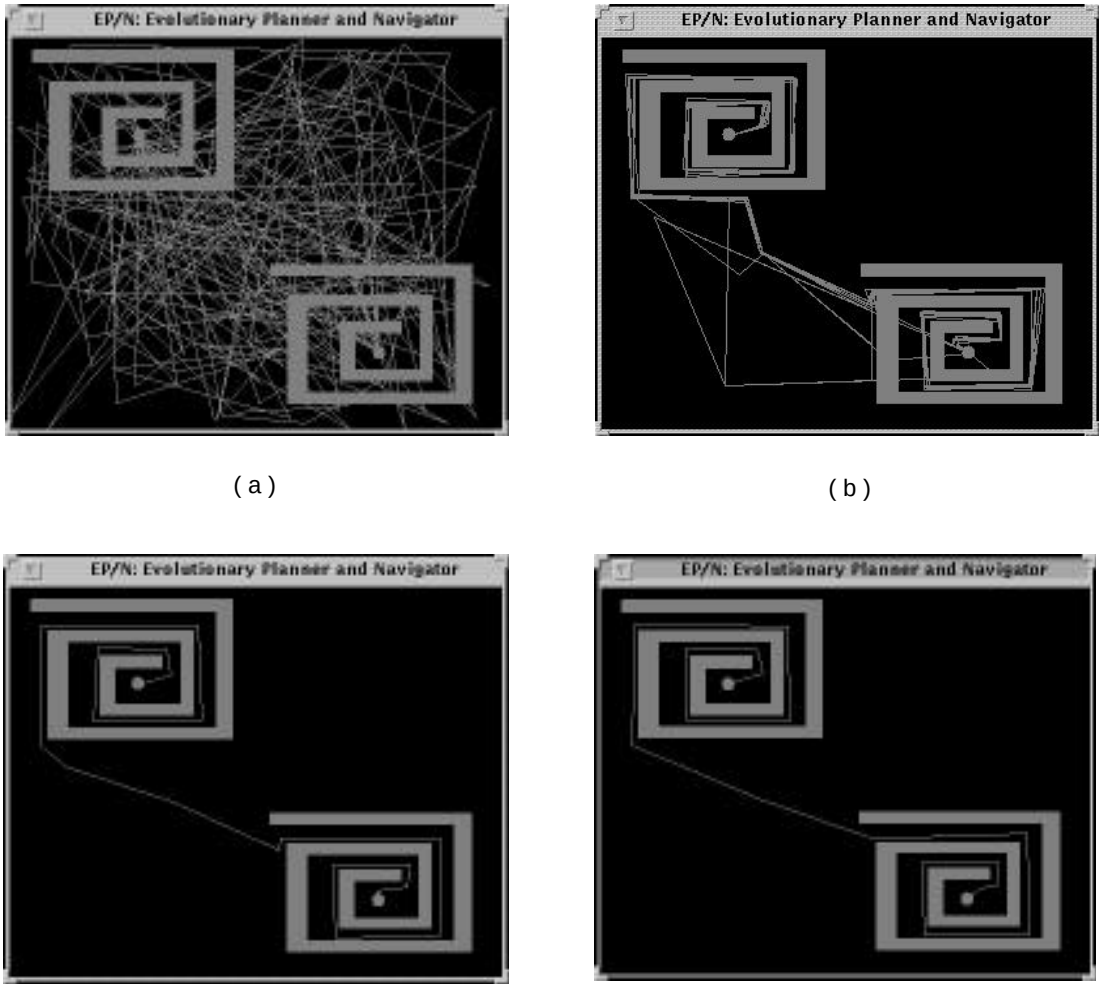

Figure G3.11.10. (a) $T = 0$: paths are generated randomly.
(b) $T = 100$: evolution has taken 1.60 seconds.
(c) $T = 350$: evolution has taken 9.88 seconds; the best path has 19 nodes and a cost of 2434.88.
(d) $T = 1000$: evolution has taken 27.45 seconds; the best path has 17 nodes and a cost of 2381.53.

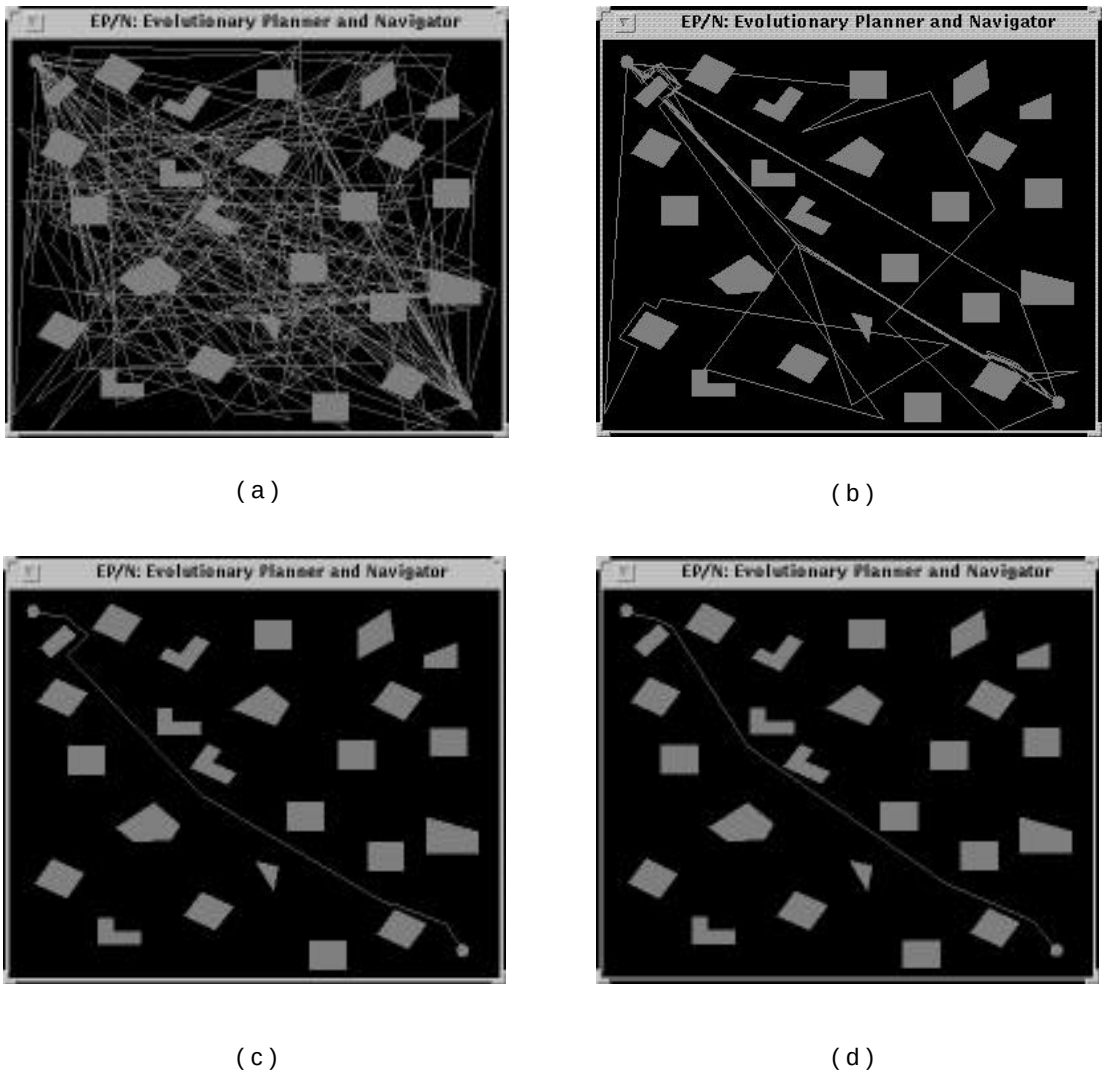

Figure G3.11.11. (a) $T = 0$: paths are generated randomly.
(b) $T = 60$: evolution has taken 1.74 seconds.
(c) $T = 150$: evolution has taken 6.19 seconds; the best path has 7 nodes and a cost of 950.79.
(d) $T = 400$: evolution has taken 19.78 seconds; the best path has 6 nodes and a cost of 910.60.