Adaptive Evolutionary Planner/Navigator for Mobile Robots

Jing Xiao*, Zbigniew Michalewicz†,
Lixin Zhang‡, and Krzysztof Trojanowski§

Abstract

Based on evolutionary computation (EC) concepts, we developed an adaptive Evolutionary Planner/Navigator (EP/N) as a novel approach to path planning and navigation. The EP/N is characterized by generality, flexibility, and adaptability. It unifies off-line planning and on-line planning/navigation processes in the same general and flexible evolutionary algorithm which (1) accommodates different optimization criteria and changes in these criteria, (2) incorporates various types of problem-specific domain knowledge, (3) enables good trade-offs among near-optimality of paths, high planning efficiency, and effective handling of unknown obstacles. More importantly, the EP/N can self-tune its performance for different task environments and changes in such environments, mostly through adapting probabilities of its operators and adjusting paths constantly even during a robot's motion towards the goal.

1 Introduction

The *path planning* problem for mobile robots is typically formulated as follows [20]: 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. Although a great deal of research has been done towards solving this problem, conventional approaches tend to be inflexible in responding to (1) different optimization goals and changes of goals, (2) different environments or changes and uncertainties in an environment, and (3) different constraints on computational resources (such as time and space). Traditional off-line planners often assume that the

*Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA; e-mail: *xiao@uncc.edu.*

†Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA; e-mail: *zbyszek@uncc.edu* and Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland; e-mail: *zbyszek@ipipan.waw.pl.*

‡Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA.

§Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland; e-mail: *trojanow@ipipan.waw.pl.*

environment is perfectly known and they try to search for *the* optimal path based on some fixed criteria (most commonly: the shortest path) by any costs (see [20, 9] for surveys). On-line planners, on the other hand, are often purely reactive and do not try to optimize a path (e.g., [3, 13, 12, 1, 4]). There are also approaches combining traditional off-line planners with incremental map building to deal with a partially-known environment such that global planning is repeated whenever a new object is sensed and added to the map [6, 15, 21]. Such approaches, however, suffer from the same inflexibility as the traditional off-line planners do. The many advantages of evolutionary computation have inspired the emergence of EC-based path planners. However, early planners often use standard evolutionary algorithms (e.g., [16, 22, 17]) without being empowered by more domain-specific knowledge. In addition, they often assume discrete search maps derived from known environments and thus they are also inflexible like many traditional planners. More recently, there emerged EC-based planners to deal with dynamic environment with parallel implementation [2] and to create diversity in paths [5].

However, there is still a need for more general, flexible, and preferably adaptive planners *capable of meeting any changes in requirements and environments*. EC provides a promising paradigm for such a general planner, but to be effective, such a planner (1) should be the product of creative application of the EC *concept* incorporating heuristic knowledge rather than dogmatic imposition of any standard algorithm, (2) should not be limited to searching paths in some fixed abstract map structure, and (3) should be able to accommodate or to *adapt to* diversities and changes in optimization goals, environments, and computing resources.

The EP/N was developed to embody the above ideas for a general, flexible, and adaptive planner. It combines the concept of evolutionary computation with problem-specific chromosome structures and operators. Unlike many other planners which need to first build a discretized map for search, the EP/N simply "searches" the original and continuous environment to generate paths, and there is little difference between off-line planning and on-line navigation for the EP/N. In fact, the EP/N combines off-line planning and on-line navigation in the same evolutionary algorithm using the same chromosome structure.

Since its first version [11], the development of the EP/N system has been an ever living "evolution" process itself: major effort in the past was focused on operators and fitness evaluation [18, 14], and more recently—on system performance and self-tuning [19]. In this paper, we focus on the adaptability of the EP/N system, which is characterized by (1) the adaptability of operator probabilities, and (2) the adaptability of the on-line (real time) navigation process, and we present related new results. Such new development enables the EP/N to deal with changes, unknowns and uncertainties gracefully.

The paper is organized as follows. Section 2 introduces the evolutionary algorithm of the EP/N in detail. Section 3 describes the self-tuning capabilities of the EP/N. Section 4 presents a set of off-line experiments done on the EP/N which demonstrate its adaptability to diverse environments. Section 5 discusses the on-line process and presents simulation results of the on-line navigation on a few environments. Section 6 concludes the paper and discusses further research issues.

2 Description of the EP/N algorithm

As introduced in Section 1, the EP/N uses the same evolutionary algorithm and chromosome structure for both off-line planning and on-line navigation. The outline of the adaptive EP/N is shown in Figure 1.

```
procedure EP/N
begin
   t ← 0
   if known_path then
      input P(t)
   else
      initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
   begin
      t ← t + 1
      select operator o
      select parent(s) from P(t)
      produce an offspring by applying the operator o
         to the selected parent(s)
      evaluate new offspring
      replace the worst member of the population P(t)
         by the produced offspring
      select the best individual p from P(t)
      if online and p feasible and (t mod n) = 0 then
      begin
         move one step k_max along the path determined
            by p while sensing the environment
         modify the values in all individuals
            due to a new starting position
         if there is any change sensed then
            update the object map
         evaluate P(t)
      end
   end
end
```

Figure 1: The Adaptive Evolutionary Planner/Navigator

In the EP/N algorithm, a chromosome in a population $P(t)$ of generation t represents a (feasible or infeasible) path leading the robot to the goal location (see Section 2.1 for details on the chromosome structure). Each chromosome is evaluated (Section 2.2) and the algorithms enters evolutionary loop (while statement). An operator (Section 2.3) is selected on the basis of some probability distribution (Section 3); the set of operators consists of a number of unary transformations (mutation type), which create offspring by a small change in a single individual, and higher order transformations (crossover type), which create offspring by combining parts from several (two or more) individuals. The produced offspring replaces the worst individual in the population. Thus, in this steady-state evolutionary system, the populations $P(t+1)$ and $P(t)$ differ by a single individual.

The two Boolean variables *known_path* and *online* are used to achieve maximum flexibility. If *known_path* is true, it means that the EP/N does not have to create the initial population $P(0)$ of chromosomes (which represent paths) from scratch. Instead, it can input a population of paths as the initial generation, which could be the results of previous planning and/or navigation or obtained from a priori knowledge of the task (i.e., the paths to accomplish the task), and so on. Otherwise, the EP/N needs to generate an initial population (Section 2.1). The value of *online* indicates the working mode of the EP/N. If *online* is false, the algorithm is run off-line, characterized by evolution of paths (chromosomes) based on only known information of an environment. If *online* is true, the algorithm is run in real time to guide a robot's movement based on both known and newly sensed information of the environment.

The on-line EP/N runs two processes in parallel:

1. navigation of the robot along the current best path while sensing the environment to detect unknown objects, and

2. continuation of the evolution process in search for further path improvements, taking into account new location of the robot and newly sensed objects (if any).

The two processes are related in the following way: while the robot moves along the current best path p_c, the best new path p emerged from the evolution process is checked every n generations for feasibility: if p is feasible, the robot starts moving along p; otherwise the robot continues to move along p_c while the evolution process also continues. Note that during such on-line navigation, the starting location of each path (chromosome) in a population is constantly updated to reflect the current location of the robot as it moves. By letting the robot to follow the *current* best path from the continuing evolution, the EP/N is able to constantly improve the robot motion between the current location of the robot and the goal, even if the robot is not approaching any obstacles. A discovery of a new obstacle during the navigation process results in changes in fitness values for all paths in the current population. The on-line process is further detailed in Section 5.

The flexible EP/N algorithm (as the two Boolean variables *known_path* and *online* indicate) allows an off-line planning process and an on-line navigation process to be nicely concatenated: the final generation of paths found by the off-line planning can be input to the on-line process as the initial population, a basis to start navigation and further evolution. On the other hand, if the environment is totally unknown beforehand, planning will depend the on-line process only, where the evolution can start from a randomly generated initial population of paths (Section 2.1).

The following subsections describe the important components of the EP/N algorithm: (1) the chromosome structure and initialization process, (2) the evaluation function, and (3) the operators used.

2.1 Chromosomes and initialization

In the EP/N algorithm, a chromosome represents a path, consisting of straight-line segments, as the sequence of knot points (i.e., intersections between two segments) or nodes on the path (Figure 2). 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, providing information such as whether (1) the knot point is feasible (i.e., outside obstacles) and whether (2) the path segment connecting the knot point to the next knot point is feasible (i.e., without intersecting obstacles). Thus, a path (or chromosome) can be either feasible or infeasible. A feasible path is collision-free, i.e., has only feasible nodes and path segments.

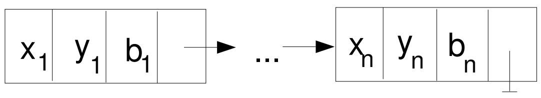

Figure 2: A chromosome representing a path

A path (or chromosome) can have a varied number of nodes. An initial population of chromosomes can be randomly generated such that each chromosome has a random number of nodes and randomly-generated coordinates for each node. Chromosomes are then evaluated and selected (based on fitness) for parenthood for one of eight operators (also selected on the basis of some probability distribution—see Section 3) for a possible improvement. This sequence of application of the chosen operator to one or two selected parents, evaluation of offspring, and the replacement of the worst individual by the generated offspring, corresponds to a single generation in the evolutionary process. The process terminates after some number of generations, which can be either fixed by the user or determined dynamically by the program itself, and the best chromosome represents the near-optimum path found.

2.2 Evaluation

The evaluation function of a chromosome p measures the cost of a path represented by p. Since a path can be either feasible (i.e., collision-free) or infeasible, we adopt two separate evaluation functions, $eval_f$ and $eval_i$, to handle the feasible and infeasible cases, respectively. For feasible paths, our current $eval_f$ is designed to accommodate three different optimization goals: shortness, smoothness, and clearness (i.e., away from obstacles) of a path. Specifically, $eval_f$ is a linear combination of these three factors:

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

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 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.

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

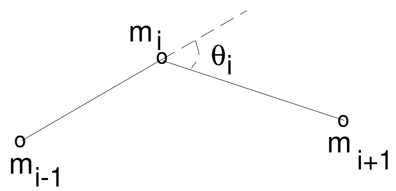

Figure 3: θ_i at each node m_i

For infeasible paths, our design of $eval_i$ takes into account several factors: 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 infeasible segments, the total lengths of feasible and infeasible segments, and so on, as detailed in [18]. In ranking all paths, we assume that the worst feasible path is better (or fitter) than the best infeasible path.

2.3 Operators

The current version of EP/N uses eight types of operators to evolve chromosomes into possibly better ones. These operators are sufficient to generate an *arbitrary* path, but may not all be applicable or needed in all situations. The application of each operator is controlled by a probability. Now we introduce these eight operators, which are also illustrated in Figure 4:

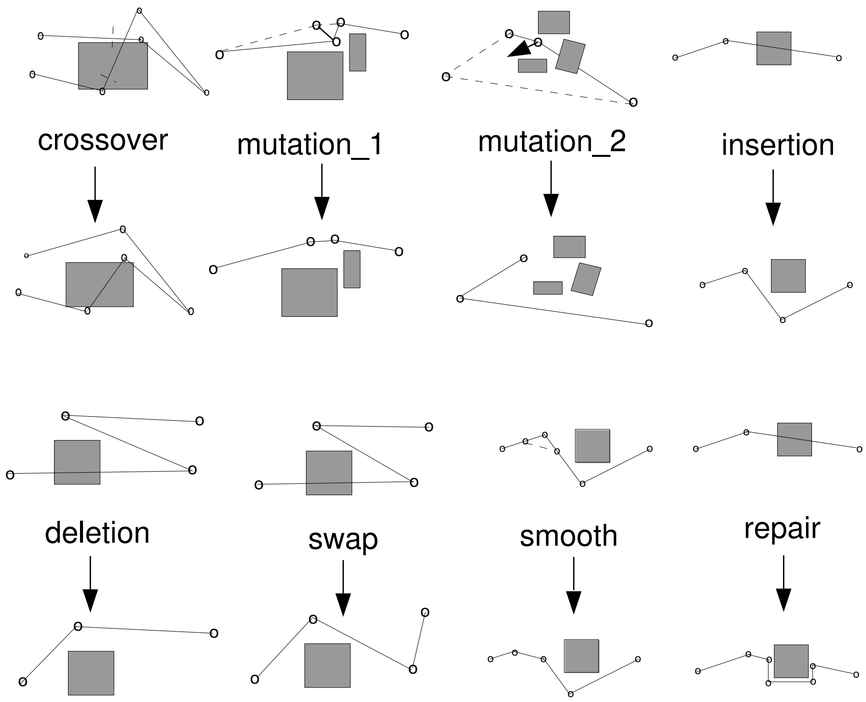

Figure 4: The roles of the operators

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.

Mutate_1: it is used for fine tuning node coordinates in a feasible path for shape adjustment. The operator randomly adjusts node coordinates within some local clearance of the path so that the path remains feasible afterwards.

Mutate_2: it is used for large random change of node coordinates in a path, which can be either feasible or infeasible.

Insert-Delete: it operates on an infeasible path by inserting randomly generated new nodes into infeasible path segments and deleting infeasible nodes (i.e., path nodes that are inside obstacles).

Delete: it deletes nodes from a path, which can be either feasible or infeasible. If the path is infeasible, the deletion is done randomly. Otherwise, the operator decides whether a node should definitely be deleted based on some heuristic knowledge, and if a node is not definitely deletable, its deletion will be random.

Swap: it swaps the coordinates of randomly selected adjacent nodes in a path, which can be either feasible or infeasible.

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. The nodes with sharper turns are more likely to be selected.

Repair: it repairs a randomly selected infeasible segment in a path by "pulling" the segment around its intersecting obstacles.

It can be seen that except for the purely random operators **Crossover** and **Mutate_2**, all the other operators (which are varied forms of mutations) are designed with some heuristic knowledge to make them more effective. Note that since most knowledge needed is available from the evaluation of path fitness (see the previous subsection), the operators mostly use the knowledge with little extra computation.

The firing probability p_i $(i = 1, ..., 8)$ of each operator governs the contribution or role of the operator to the whole evolution process. Clearly different values of these probabilities affect the overall performance of the EP/N.

3 Performance and probability self-tuning

Different values of the operator probabilities affect the overall performance of the EP/N. As the EP/N uses many operators, how to determine their probabilities properly is not a trivial matter, especially since proper values could very much depend on environmental characteristics and specific constraints imposed on a task. In this section, we describe how to enable the EP/N self-adapt these probabilities to achieve the best results by a systematic method.

3.1 Operator performance index

We evaluate the performance of an operator taking into account three essential aspects:

1. its effectiveness in improving the fitness of a path,
2. its operation time (or time cost), and
3. its operation side effect to future generations.

While the first two aspects are self-explanatory, the third aspect refers to the fact that five operators, i.e., **crossover**, **insert-delete**, **delete**, **smooth**, and **repair**, tend to change the number of nodes (or the length) of a chromosome after their application.

Since the length of a chromosome affects both the processing time and the storage space needed by the chromosome — the more nodes are in the chromosome, the more space and time (i.e., evaluation time and often operation time) are needed, if an operator alters the number of nodes in a chromosome, the effect will be felt in future processing. Such effect can be either positive or negative on the *processing cost* of future generations[1], depending on if the operator reduces or increases the number of nodes in the chromosome. Note that including the last two aspects in evaluating operators is particularly useful when constraints on operation resources (i.e., time and space) for the EP/N are stringent.

Now we describe the performance measures in detail. The three aspects are first measured individually and then combined to form a compound performance index for an operator. Since the role of an operator often varies in different stages of an evolution process (e.g., some operators apply only to infeasible paths while some only apply to feasible ones), each aspect is measured as a function of generation interval $[T_1, T_2]$, where T_1 and T_2 are the starting and ending generations of the interval. For an operator i, $i = 1, ..., 8$,

- its effectiveness in improving the fitness of a path is measured by the ratio $e_i(T_1, T_2)$ between the number of times it improves a path and the total number of times it is applied;
- its operation time $t_i(T_1, T_2)$ is measured as the average time per its operation;
- its operation side effect $s_i(T_1, T_2)$ is measured as the average time cost of all operators on the average change of nodes by the operator i:

  $$s_i(T_1, T_2) = \frac{\delta n_i \cdot t(\bar{n})}{\bar{n}}$$

  where δn_i is the average change in the number of nodes of a chromosome by operator i per its operation during the generations in $[T_1, T_2]$, such that δn_i is negative if the number of nodes is decreased on average and is positive otherwise, $\bar{n}$ is the average number of nodes in a chromosome over the generations in $[T_1, T_2]$, and $t(\bar{n})$ is the weighted average operation time (on an average chromosome) of all operators during $[T_1, T_2]$:

  $$t(\bar{n}) = \sum_{i=1}^{8} \frac{m_i}{T_2 - T_1} \cdot t_i(T_1, T_2)$$

  where m_i is the number of times (i.e., generations) the operator i is applied during $[T_1, T_2]$.

[1]It is important to differentiate the processing cost (in terms of time and space) and the fitness of a chromosome; the latter is often improved as the chromosome has more nodes (such as after the operator **repair** or **smooth** is applied).

Note that the formulation of $s_i(T_1, T_2)$ takes into account the fact that the node number change in a chromosome by operator i has an effect on *any* future operation, *not* necessarily by the same operator i.

The overall performance of an operator i is measured by the following *performance index* $I_i(T_1, T_2)$, which combines the three aspects of performance:

$$I_i(T_1, T_2) = \frac{e_i(T_1, T_2) + c}{t_i(T_1, T_2) + s_i(T_1, T_2)}$$

where $c \geq 0$ is a small constant. Note that greater value of I_i means better performance. In addition, when $s_i(T_1, T_2)$ is negative (i.e., when δn_i is negative), it contributes positively to I_i, which can be shown to be non-negative.

The operator performance index I_i has great significance because of the following:

- It can be automatically computed by the EP/N since it is based on statistics which the EP/N can accumulate during its run. Thus, it can be used by the EP/N for automatic determination of the operator probability p_i, defined as:

$$p_i = \frac{I_i}{\sum_{i=1}^{8} I_i} \tag{1}$$

- Since I_i is a function of generation interval $[T_1, T_2]$, for different generation intervals, the EP/N can compute different I_i's and accordingly different p_i's. That is, a look-up table that maps different generation intervals to different operator probabilities can be built by the EP/N automatically. Next, the operator probabilities can be changed during different stages of evolution to achieve greater effectiveness and efficiency.

- More importantly, the EP/N can be self-adaptive as follows. Let δT be a sufficiently small number of generations. Assign all initial operator probabilities randomly (for example, uniformly). After the first δT generations, use the computed $I_i(0, \delta T)$, $i = 1, ..., 8$, to compute new probabilities $p_i(I_i)$ and use the new probabilities in the next δT generations. Afterwards, compute the next $I_i(\delta T, 2\delta T)$ and again reset the probabilities accordingly. Repeat the procedure until the whole evolution process terminates.

3.2 System performance measures

We use the following measures to evaluate the performance of the whole EP/N system over $[0, T]$ generations:

- **effectiveness index** — in terms of the average path cost avg_T or the best path cost $best_T$ in the population of the final generation T.

- **efficiency index** — in terms of the product of the average path cost avg_T or the best path cost $best_T$ in the final generation T and the total time t_T spent over $[0, T]$ generations: $avg_T \times t_T$ or $best_T \times t_T$.

Clearly *smaller* values of both indices mean *better* effectiveness and *better* efficiency respectively. To improve system performance is to reduce the values of those indices.

Note that the above measures are not necessarily optimal ways of measuring the system performance. One may also take into account factors such as how quickly infeasible paths are evolved into feasible ones (or the percentage of the feasible paths in each generation) and the diversity of feasible paths, and so on, depending need.

3.3 Adapting operator probabilities

Based on the procedure of computing the operator performance indices I_i's in the EP/N, we have used the following method to enable the EP/N self-adapt its operator probabilities at run-time. First, divide the total number of generations T into several equal intervals such that the number of intervals determine the frequency of adaptation. To begin the run, let the EP/N assign equal probabilities to all operators initially. Then, after the first interval of generations, the EP/N computes the corresponding I_i's and probabilities p_i's (by equation (1)) and resets the operator probabilities to the newly computed p_i's to run the next interval of generations. At the end of interval 2, the EP/N again resets the operator probabilities based on the I_i's corresponding to that interval and uses the new probabilities to run interval 3, and so on. Thus, adaptiveness is achieved by applying the probabilities computed based on the operator performance in generation interval n to the next interval $n+1$.

Our experiments showed (Section 4) that comparing to running the EP/N with equal operator probabilities or other manually determined fixed operator probabilities, running the system with adaptive operator probabilities enhanced both the effectiveness and efficiency of the system for diverse tasks.

4 Off-line experiments and results

We have implemented the EP/N for polygonal obstacles and run the EP/N off-line on different tasks in diverse environments to test its self-tuning ability and overall performance. Figure 5 shows six different tasks in six different environments, where for each task, a near-optimal path obtained by the EP/N is displayed.

Tables 1 shows the gain on both system effectiveness and efficiency when the EP/N self-adapted its operator probabilities against the case when the EP/N used fixed, equal operator probabilities for each environment shown in Figure 5, in a total $T = 400$ generations. Specifically, T was divided into 4 intervals of 100 generations each, i.e., the frequency of adaptation is 4. The population size was set to be 30. Each result was the average of repeatedly running the EP/N under the same condition 100 times.[2] As expected, values of both system effectiveness and efficiency indices are *reduced*, which means *better* effectiveness and efficiency was achieved in most of the cases.

[2]Note that [19] presented some results averaged over 30 times. We later find that 30 runs are not enough statistically and 100 runs are more reasonable.

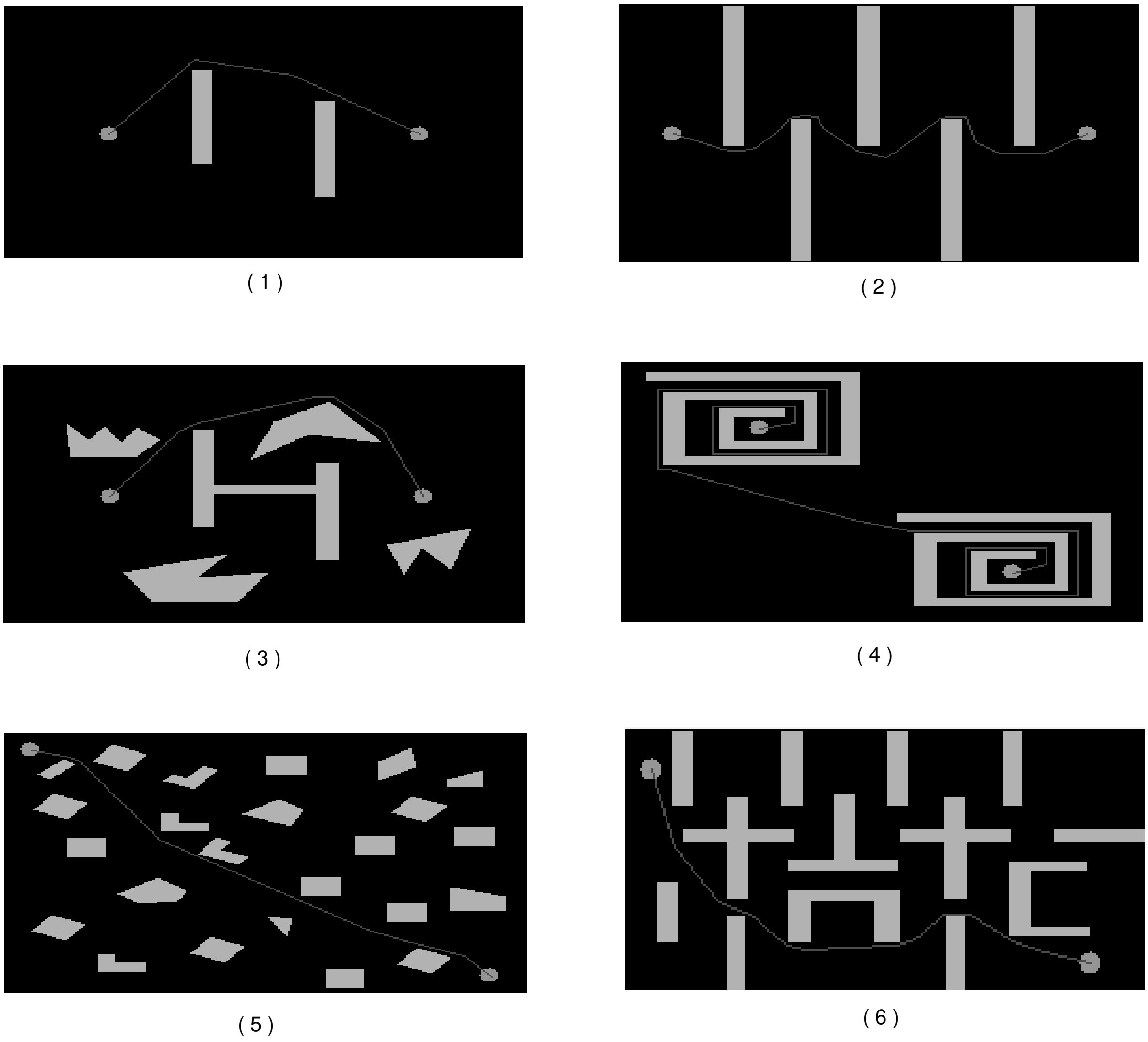

Figure 5: Task environments and typical paths found by EP/N

Table 2 shows, for each environment and the task shown in Figure 5, the average time of running the EP/N with adaptive probabilities for $T = 600$ generations over 20 runs on a Silicon Graphics station. The population size was set to 30 for each case, and the coefficients w_d, w_s, w_c, a, τ in the evaluation function $eval_f(p)$ were selected as 1.0, 1.0, 1.0, 7.0, 10, respectively. For the environments 1–5, the best paths found are as shown in Figure 5. For the task in the environment 6, we provide snapshots at six different states of evolution as shown in Figure 6, where two-thirds of the population are displayed. The best path found at the last state ($T = 1000$), i.e., the best of the paths from Figure 6 (for $T = 1000$), is shown in Figure 5.

Comparing to the results obtained by running the EP/N with fixed, manually de-

Table 1: Average gain (over 100 runs) on system performance against the case with equal operator probabilities in $T = 400$ generations

env	effectiveness		efficiency	
	avg_T%change	$best_T$ %change	$avg_T \times t_T$ %change	$best_T \times t_T$ %change
1	-1.77%	-1.17%	-1.20%	-0.60%
2	-4.03%	-0.39%	-1.14%	2.61%
3	-4.59%	-2.79%	-12.76%	-11.11%
4	-4.80%	-3.43%	-4.90%	-3.52%
5	-2.28%	-1.41%	-6.30%	-5.46%
6	-5.48%	-4.96%	-7.88%	-7.39%

Table 2: Average running time (over 20 runs) for $T = 600$ on a Silicon Graphics station. The time for the simplest environment 1 is for $T = 200$

environment	1	2	3	4	5	6
time (sec)	0.17	1.45	3.33	5.46	7.25	7.61

termined operator probabilities [18], the results from running the EP/N with adaptive probabilities (as presented here) show significant improvement of the system performance. The advantages of letting the system self-adapt are obvious.

5 On-line navigation

We have implemented a simulation program for the on-line navigation of the EP/N (see Figure 1) with the following assumptions:

- There are two kinds of obstacles in the environment: known and unknown. In the current implementation they are assumed to be static.
- There is a range of view of the robot (described by R, parameter of the robot). If, because of robot's motion, an unknown obstacle is located in the range of the view, the obstacle becomes known and is marked in the robot's map of the environment. In the current stage of simulation, for simplicity, we further assume that once an obstacle is inside the robot's range of view, it becomes known totally. Of course, to be more realistic, this assumption can easily be revised as only the part of the object boundary that is inside the robot's range of view is known. However, as our current focus is on testing the robot's adaptation capability to the discovery of unknowns in an environment, it does not matter much at present how such discovery is actually done.

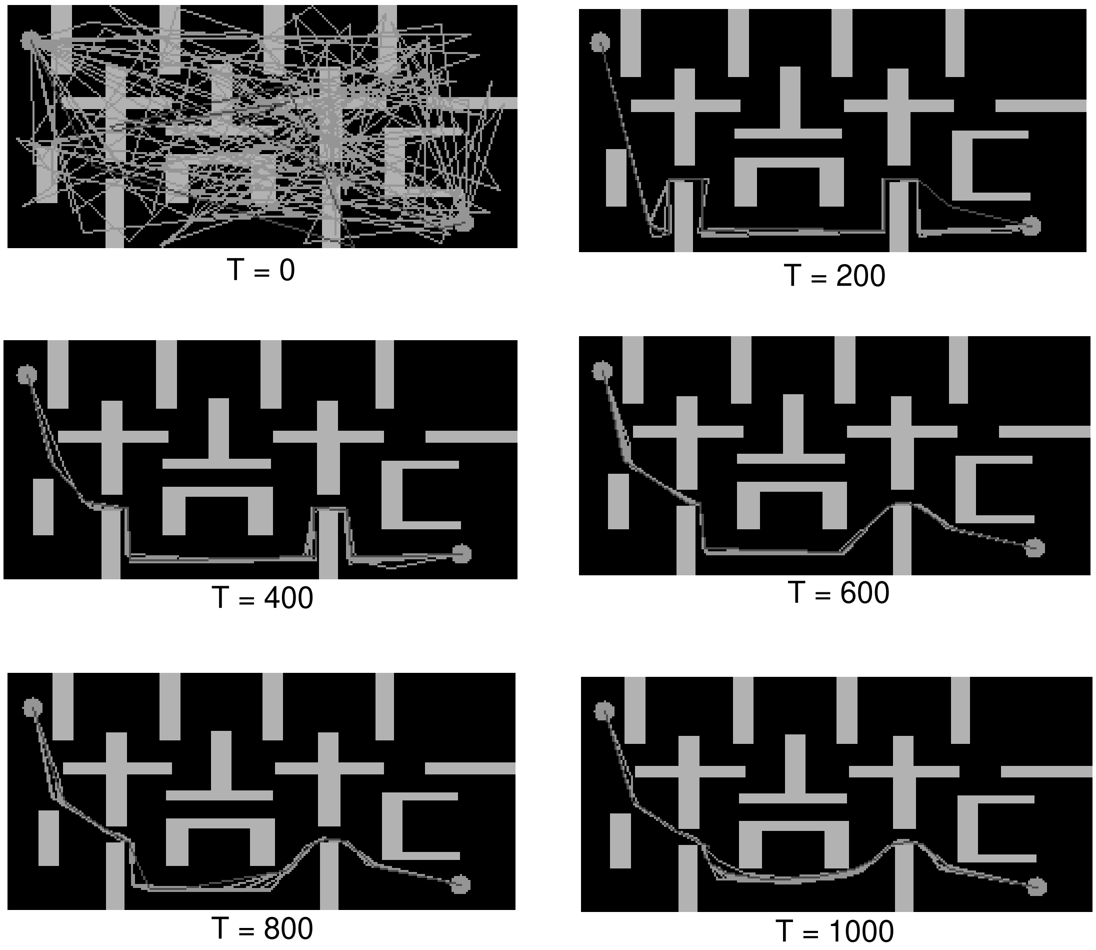

Figure 6: Snapshots of evolution

In the on-line navigation, the result of evolution is checked after every n generations (see the second "if" statement in the procedure of Figure 1) to provide the robot with the current best feasible path. The robot moves along such a path in steps, and we use a parameter k_{max} to denote the maximum length of the robot's step. k_{max} is less than R. If a segment of the path currently being followed by the robot has a length shorter than k_{max}, the robot will cover it in one step to reach the next knot point of the path. Otherwise, the robot will move along the segment in more than one step, and during the process, it may also change course if a better path (i.e., the next best path) becomes available from the evolution process. The evolution process runs in parallel with the robot's motion.

The implication of sensing a previously unknown obstacle is a change in the fitness values of the current path population. As such new obstacle is added to the robot's map of environment, it may change the subsequent evaluation results of all paths.

This is a very sensitive moment for the evolution process: some of the (feasible) paths in the population may become infeasible and their cost can grow up in a significant way. Usually, the current best path may not be the best any longer and a new best path may need to be found.

5.1 An example

The actions of the robot guided by the on-line navigation process are illustrated by the following example. Figure 7 displays robot's environment. The start point is in the left bottom corner and the goal point is close to the right top corner of the rectangle. There are eight obstacles in total; two of them are unknown (marked by their contours only).

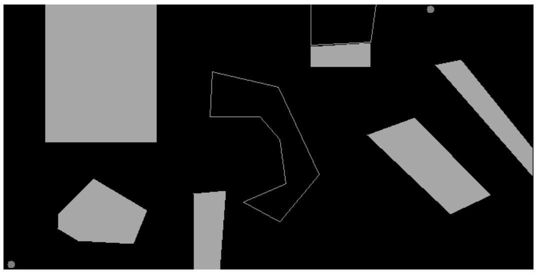

Figure 7: The environment

During the initialization process, a set of randomly generated paths is developed; very likely none of these initial paths is feasible (Figure 8 displays the best path in the initial population).

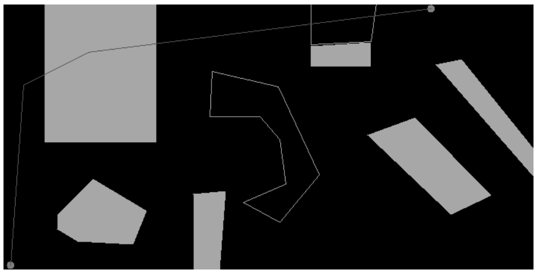

Figure 8: The best path in the initial population

Soon after discovery of the first feasible path, the first step ahead is made. Note that this path of the robot is optimized only based on known obstacles so that it does not have to be really feasible (Figure 9). Note also, that the remaining parts of the paths are continuingly optimized: the current best path in Figure 9 is different from that in Figure 10.

When a previously unknown obstacle is in the robot's range of view, it becomes known and the robot marks its presence on the map of the environment (Figures 10

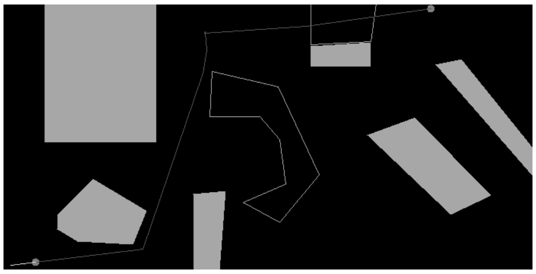

Figure 9: The first feasible path

and 11). At this moment, all paths in the current population are re-evaluated, and as the evolution process continues, the robot eventually follows a newly emerged best feasible path (Figure 12).

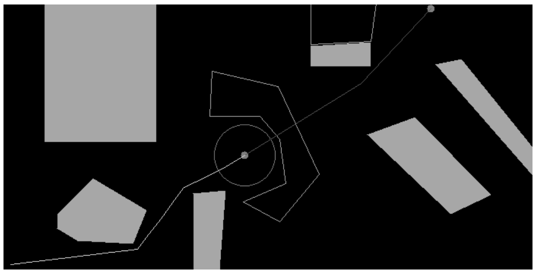

Figure 10: Before sensing the first unknown obstacle

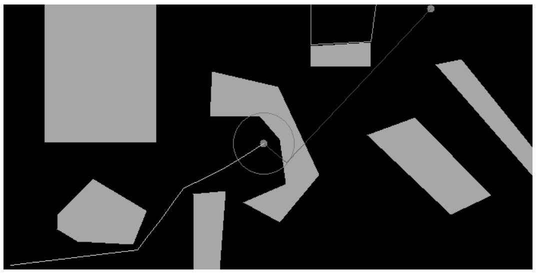

Figure 11: Sensing and re-evaluation

As mentioned earlier, as the robot proceeds, the remaining parts of the paths are continuingly optimized: the current best path in Figure 12 is different from that in Figure 13. When the second unknown obstacle is discovered, again, all paths in the population are re-evaluated, and the robot adjusts its motion accordingly by following a newly emerged best path (Figure 14).

The actual path traversed by the robot is displayed in Figure 15 (note again the improvements in the final segments of the path from those displayed in Figure 14). Clearly, this path is far from being optimal in comparison with an "ideal" path which

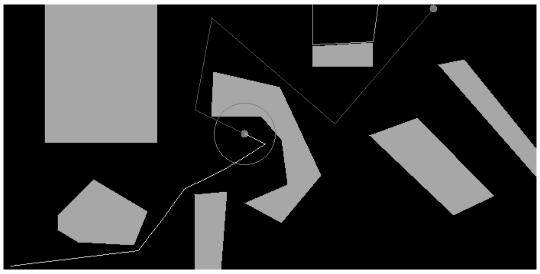

Figure 12: The best feasible path after sensing the first unknown obstacle

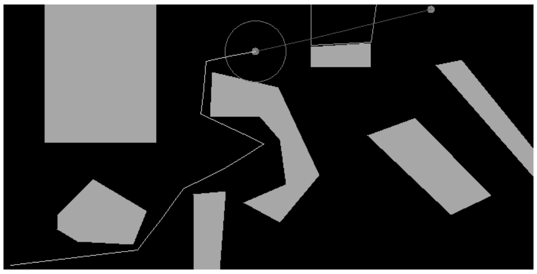

Figure 13: Before sensing the second unknown obstacle

would emerge if all obstacles in the environment were known beforehand (Figure 16). However, it is unfair to simply compare such a real path traversed as the result of on-line dealing with unknown obstacles to the "ideal" one.

5.2 Analysis of performance

To evaluate the quality of a real path, a more reasonable approach is to divide the path into so-called fragments: the cut point between fragments is the location where the robot sensed a new obstacle. There are as many cut points as the number of new obstacles sensed during the robot's movement (therefore the number of fragments, f, is by one greater than the number of cut points). Then each fragment is compared

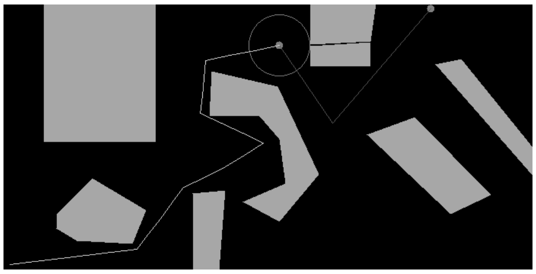

Figure 14: After sensing the second unknown obstacle

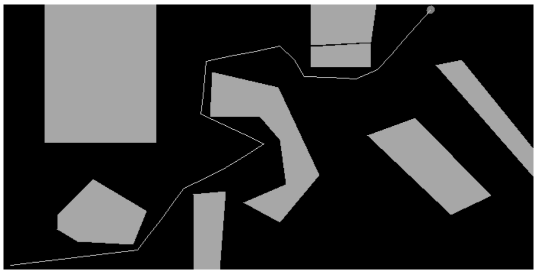

Figure 15: The robot's real path

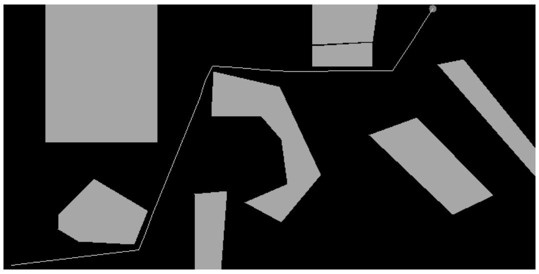

Figure 16: The robot's "ideal" path for the completely known environment

to an ideal path generated to connect the fragment's start and goal locations[3], which results in a relative error e_i in path cost for the segment.

Knowing the merit of each fragment (measured by e_i), we can measure the error E of the real path with the formula:

$$E = \frac{\sum_{i=1}^{f} e_i \cdot g_i}{N},$$

where:

f – number of fragments,
e_i – relative error of the i-th fragment (in path cost),
g_i – the number of generations passed during the traversal of the i-th fragment, and
$N = \sum_{j=1}^{f} g_j$, i.e., N represents the total number of generations.

Note that g_i's depend on the following parameters: n (the number of generations between robot's steps), k_{max} (the robot's step length), and R (the robot's range of view). As the result, longer fragments usually correspond to larger values of g_i's, and consequently smaller values of e_i's. This is why we define the error E as a weighted average of e_i's, with the corresponding weight being g_i/N. Note also that if the robot

[3]Such an ideal path can be generated by running the EP/N off-line in the same environment but with all obstacles known for a sufficient number of generations.

Table 3: Result of experiments for $n = 5$; the resulting number of generations $N = 463.7$ and error $E = 0.118076$

frag.	**real cost**	**ideal cost**	g_i	e_i	$(e_i \cdot g_i)/N$
1	681.1	603.1	200.1	0.129331	0.055810
2	1059.3	951.0	198.9	0.113880	0.048847
3	715.8	653.0	64.7	0.096172	0.013419

Table 4: Result of experiments for $n = 10$; the resulting number of generations $N = 592.8$ and error $E = 0.081996$

frag.	**real cost**	**ideal cost**	g_i	e_i	$(e_i \cdot g_i)/N$
1	655.4	603.2	251.9	0.086538	0.036773
2	1031.5	953.3	223.8	0.082031	0.030969
3	699.9	652.8	117.1	0.072151	0.014254

encounters no unknown obstacle, then there is only one fragment: the entire real path, and E is simply the relative error of the real path traversed against an ideal path (which can be the result of evolution over a larger number of generations).

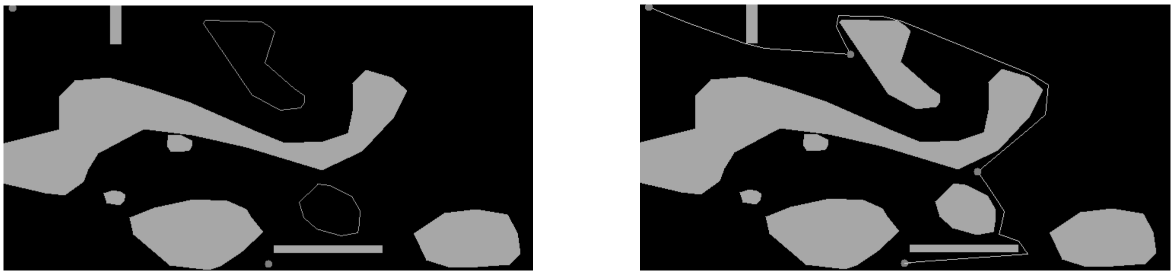

Figure 17: Experimental environment; three fragments of a path are clearly marked

We now discuss how E is related to n, the number of generations between the robot's steps. Tables 3 – 6 show results of experiments (average over 100 runs) for different n's. The other parameters used in these experiments were: population size 70, the robot's single step $k_{max} = 40$, and the range of view $R = 45$. The environment used is displayed in Figure 17, whose dimensions were 400×500.

Results reported in Tables 3–6 confirm a basic intuition: the total error E decreases with the growth of n. That is, the more generations between robot's steps are, the better precision (in terms of the path quality) can be achieved. Also, the relative errors e_i's are smaller for later fragments, i.e., fragments closer to the goal. This is because these fragments are subject to more optimization (in terms of larger number of generations) in the on-going evolutionary process.

Table 5: Result of experiments for $n = 15$; the resulting number of generations $N = 753.9$ and error $E = 0.071032$

frag.	**real cost**	**ideal cost**	g_i	e_i	$(e_i \cdot g_i)/N$
1	649.1	603.0	303.2	0.076451	0.030747
2	1017.4	952.2	333.4	0.068473	0.030281
3	695.2	653.2	117.3	0.064299	0.010004

Table 6: Result of experiments for $n = 20$; the resulting number of generations $N = 960.3$; error $E = 0.064764$

frag.	**real cost**	**ideal cost**	g_i	e_i	$(e_i \cdot g_i)/N$
1	650.5	603.1	341.1	0.078594	0.027917
2	1009.7	952.7	455.9	0.059830	0.028404
3	685.0	652.6	163.3	0.049648	0.008443

On the other hand, note that in this specific implementation of on-line navigation, n is coupled with the step length k_{max} of the robot, in that n generations of evolution must complete before the robot proceeds to the next step. For a fixed k_{max}, this implies that a very large n *may* cause a slower movement of the robot, and therefore a longer time for path traversal.

Now let us discuss the effect of n on the time of navigation. As confirmed by the experimental results (shown in Tables 3–6), a larger n leads to a larger total number of generations N, which means a longer total time of evolution t_N. Since the evolution process and the robot's movement are done in parallel, the total time of path traversal t is the maximum of t_N and the total time of robot's movement t_M:

$$t = \max(t_N, t_M),$$

where t_M is a function of path quality (length and smoothness) and the robot's velocity. From off-line running the EP/N (Section 4), we know that even for a very complex environment, t_N for a reasonable N (e.g., in the range of 600–1000, as in the Tables 3–6) is usually in the order of a few seconds, which should be well below the time required for a robot to physically traverse a normal indoor or outdoor environment. That is, $t_N \leq t_M$ can usually hold for a reasonably large N as a result of a comfortably large n. In other words, n can be sufficiently large without affecting the time of traversal. Since a larger n can result in a path of better quality, which often means a smoother path of shorter length, a larger n may actually reduce the robot's time of traversal.

In summary, the parallelism between path evolution and path traversal in the adaptive EP/N is shown to be very advantageous in achieving both high effectiveness and high efficiency in real-time navigation of a robot, especially when the environment is only partially known.

6 Conclusions and Future Research

The EP/N represents a promising new approach in robot path planning taking great advantage of evolutionary computation. The adaptive EP/N presented in this paper is particularly suitable for dealing with the diversities, changes, and unknowns in an environment gracefully with both high effectiveness and efficiency. Results from off-line planning with adaptive probabilities of genetic operators and simulation of on-line navigation are presented, which confirm the flexible nature and advantage of such an evolutionary system. We are currently implementing the on-line process of the EP/N on a real robot (Khepera robot). We expect to see more results from the real-time experiments.

The EP/N also exposes many interesting challenges of general importance to evolutionary computation. One of the most important is how to make an evolutionary system self-adaptive. The adaptive EP/N uses an automatic mechanism to measure performances of its genetic operators and self-adapt the operator probabilities accordingly. The general nature of the strategy makes it applicable to other evolutionary systems as well. The on-line adaptiveness of the EP/N to changes/unknowns in an environment provides a good example of a real-time adaptive system based on evolutionary computation. An important issue of future research is how to make the EP/N capable of adapting other system parameters. Several such parameters may be of particular interests. One is the parameter deciding how frequently operator probabilities should be adjusted or adapted (i.e., the generation interval $[T_1, T_2]$'s in Section 3). Parameters n and k_{max} in the on-line process (Section 5) as well as the velocity of an robot are crucial to determine how frequently the robot should adjust its path during on-line navigation and how to balance the quality of path and the time of traversal. In the current implementation, n and k_{max} are coupled (Section 5.2), but they can be also implemented as two independent parameters so that their relations are mainly affected by specific environment/task characteristics. In general, for any parameter whose best value may vary for different tasks or environments, making it self-adaptive could be desirable.

It may also be desirable to further incorporate domain knowledge in important components/processes of the EP/N to enhance its performance. Although in the current EP/N, we have incorporated domain knowledge in both fitness evaluation and operators, there are other components/processes, such as the initialization process, which may benefit from more 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).

Another important issue is to improve the organization of the EP/N to stress learning for on-line navigation. The system may be extended by some sort of memory where "valuable" paths or segments of paths discovered earlier can be stored. It could also be interesting to study other forms of "memory", such as one based on multi-chromosome structures with a dominance function [7] or one employing machine learning techniques.

References

[1] Arkin, R.C., "Motor Schema-based Mobile Robot Navigation", *Int. J. Robotics Research*, pp.92–112, Aug. 1989.

[2] Bessiere, P., Ahuactzin, J.-M., Talbi, A.-G., and Mazer E., "The "Ariadne's Clew" Algorithm: Global Planning with Local Methods", Proceedings of 1993 IEEE-IROS International Conference on Intelligent Robots and Systems, Yokohama, Japan, Sept. 1993.

[3] Borenstein, J., and Koren, Y., "The Vector Field Histogram — Fast Obstacle Avoidance for Mobile Robots", *IEEE Trans. Robotics and Automation*, 7(3), pp.278-287, June 1991.

[4] Brooks, R.A., "A Robust Layered Control System for A Mobile Robot", *IEEE Journal of Robotics and Automation*, vol.2, pp.14–23, 1986.

[5] Hocaoglu, C, and Sanderson, A.C., "Planning Multi-Paths using Speciation in Genetic Algorithms", Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan, pp.378–383, May 1996.

[6] Foux, G., Heymann, M., Bruckstein, A., "Two-Dimensional Robot Navigation Among Unknown Stationary Polygonal Obstacles", *IEEE Transactions on Robotics and Automation*, vol.9, pp.96–102, 1993.

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

[8] Khatib O., "Real-Time Obstacles Avoidance for Manipulators and Mobile Robots", *International Journal of Robotics Research*, vol.5, pp.90–98, 1986.

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

[10] 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.

[11] Lin, H.-S., Xiao, J., and Michalewicz, Z., "Evolutionary Algorithm for Path Planning in Mobile Robot Robot Environment," Proc. IEEE Int. Conf. Evolutionary Computation, Orlando, Florida, June 1994, pp. 211-216.

[12] Lumelsky, V.J., "A Comparative Study on the Path Length Performance of Maze-Searching and Robot Motion Planning Algorithms", *IEEE Trans. Robotics and Automation*, 7(1), pp.57–66, Feb. 1991.

[13] Lumelsky, V.J., and Stepanov, A.A., "Path Planning Strategies for a Point Mobile Automaton Moving amidst Unknown Obstacles of Arbitrary Shape", *Algorithmica*, vol.2, pp.403–430, 1987.

[14] Michalewicz, Z. and Xiao, J., "Evaluation of Paths in Evolutionary Planner/Navigator", Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, Japan, May 30–31, 1995, pp.45–52.

[15] Oommen, J.B., Iyengar, S.S., Rao, N.S.V., and Kashyap, R.L., "Robot Navigation in Unknown Terrains Using Visibility Graphs: Part I: The Disjoint Convex Obstacle Case", *IEEE J. Robotics and Automation*, RA-3, pp.672–681, 1987.

[16] Page, W.C., McDonnell, J.R., and Anderson, B., "An Evolutionary Programming Approach to Multi-Dimensional Path Planning", Proceedings of the First Annual Conference on Evolutionary Programming, D. Fogel and J.W. Atmar (Editors), Evolutionary Programming Society, San Diego, 1992, pp.63–70.

[17] Shibata, T., and Fukuda, T., "Robot Motion Planning by Genetic Algorithm with Fuzzy Critic", Proceedings of the 8th IEEE International Symposium on Intelligent Control, Chicago, August 25-27, 1993.

[18] Xiao, J., "Evolutionary Planner/Navigator in a Mobile Robot Environment," to appear in the *Handbook of Evolutionary Computation*, (T. Bäck, D. Fogel, and Z. Michalewicz, Editors), Oxford University Press and Institute of Physics Publishing.

[19] Xiao, J., Michalewicz, Z., and Zhang, L., "Evolutionary Planner/Navigator: Operator Performance and Self-Tuning," Proc. IEEE Int. Conf. Evolutionary Computation, Nagoya, Japan, May 1996, pp. 366–371.

[20] 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.

[21] Zelinsky, A., "A Mobile Robot Exploration Algorithm", *IEEE Transactions on Robotics and Automation*, vol.8, pp.707–717, 1992.

[22] Zhao, M., Ansari, N., and Hou, E., "Mobile Manipulator Path Planning by a Genetic Algorithm", Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.681–688, Raleigh, N.C., July 7-10, 1992.