Adaptive Evolutionary Planner/Navigator for Mobile Robots

Jing Xiao, *Member, IEEE*, Zbigniew Michalewicz,
Lixin Zhang, *Associate Member, IEEE*, 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 evolutionary algorithm which 1) accommodates different optimization criteria and changes in these criteria, 2) incorporates various types of problem-specific domain knowledge, and 3) enables good tradeoffs 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 toward the goal.**

***Index Terms*—Evolutionary computation, mobile robots, navigation, path planning.**

I. Introduction

THE mobile robot path planning problem is typically formulated as follows [22]: given a robot and a description of an environment, plan a path between two specified locations which is collision-free and satisfies certain optimization criteria. Although a great deal of research has been performed to further a solution to 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 environment is perfectly known and try to search for the optimal path based on some fixed criteria (most commonly, the shortest path) which is usually costly (see [22] and [9] for surveys). On-line planners, on the other hand, are often purely reactive and do not try to optimize a path (e.g., [1], [3], [4], [12], [13]). 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 [5], [16], [23]. Such approaches, however, suffer from the same inflexibility as the traditional off-line planners. The many advantages of evolutionary computation have inspired the emergence of evolutionary computation (EC)-based path planners. Early planners, however, often used standard evolutionary algorithms (e.g., [17], [18], [24]) without being empowered by more domain-specific knowledge. In addition, they often assumed discrete search maps derived from known environments, and thus, they were also inflexible, like many traditional planners, and were not adaptive to changes or uncertainties. More recently, EC-based planners have been offered to deal with dynamic environments with parallel implementation [2] and to create diversity in paths [7].

There is still a need, however, 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 the 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 adapt to diversities and changes in optimization goals, environments, and computing resources.

The evolutionary planner/navigator (EP/N) described in this paper combines the concept of evolutionary computation with problem-specific chromosome structures and operators [14]. 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 itself been an ever-living "evolution" process: major effort was focused on operators and fitness evaluation [20], [15] and more recently on system performance and self-tuning [21]. In this paper, we focus on the adaptability of the EP/N system, particularly with respect to the operator probabilities and the on-line (real time) navigation process.

The paper is organized as follows. Section II introduces the evolutionary algorithm of the EP/N in detail. Section III

Manuscript received September 3, 1996; revised December 24, 1996. This work was supported in part by Grant 8T11C 010 10 from the Polish State Committee for Scientific Research.

J. Xiao is with Department of Computer Science, University of North Carolina, Charlotte, NC 28223 USA (e-mail: xiao@uncc.edu).

Z. Michalewicz is with Department of Computer Science, University of North Carolina, Charlotte, NC 28223 USA and Institute of Computer Science, Polish Academy of Sciences, 01-237 Warsaw, Poland (e-mail: zbyszek@uncc.edu).

L. Zhang was with Department of Computer Science, University of North Carolina, Charlotte, NC 28223 USA. He is now with Datastream System Inc., Greenville, SC 29605 USA (e-mail: zhangl@dstm.com).

K. Trojanowski is with Institute of Computer Science, Polish Academy of Sciences, 01-237 Warsaw, Poland (e-mail: trojanow@ipipan.waw.pl).

Publisher Item Identifier S 1089-778X(97)03421-8.

```
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_j with probability p_j
    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
```

Fig. 1. A high-level description of the structure of the adaptive evolutionary planner/navigator algorithm.

describes the self-tuning capabilities of the EP/N. Section IV presents a set of off-line experiments performed on the EP/N which demonstrate its adaptability to diverse environments. Section V discusses the on-line process and presents simulation results of the on-line navigation on a few environments. Section VI concludes the paper and discusses further research issues.

II. Description of the EP/N Algorithm

As introduced in Section I, 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 Fig. 1.

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 II-A for details on the chromosome structure). Each chromosome is evaluated (Section II-B), and the algorithm enters the evolutionary loop (while statement). An operator (Section II-C) is selected on the basis of some probability distribution (Section III); 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 two 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 process terminates after some number of generations, which can be fixed either by the user or determined dynamically by the program itself, and the best chromosome represents the near-optimum path found.

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 result 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 II-A). 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;
2) continuation of the evolution process in search of further path improvements, taking into account the 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 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 V.

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 on the on-line process only, where the evolution can start from a randomly generated initial population of paths (Section II-A).

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. The current forms of these components are the results of numerous redesigns, modifications, and improvements. These are by no means, however, the only way to implement the EP/N approach, and the current components can still be further improved upon (see Section VI).

Fig. 2. A linked list chromosome representing a path. Each node contains x and y coordinates of a point together with a state variable b, which provides information on feasibility of the point and the following path segment. The point (x_1, y_1) is the starting point, and the point (x_n, y_n) is the goal point.

A. Chromosomes and Initialization

In the EP/N algorithm, a chromosome represents a path, which consists of straight-line segments, as the sequence of nodes with the first node indicating the starting point of the first segment, followed by (a varied number of) intermediate nodes representing the knot points (i.e., intersection points) between segments, and the last node indicating the ending point of the last segment, which is the goal point (Fig. 2). Each node, apart from the pointer to the next node, consists of the x and y coordinates of the point and a state variable b, providing information such as whether or not 1) the point is feasible (i.e., outside obstacles) and 2) the path segment connecting the point to the next 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.

A path (or chromosome) can have a varied number of intermediate nodes. An initial population of chromosomes can be randomly generated such that each chromosome has a random number of intermediate nodes and randomly generated coordinates for each intermediate node.[1]

B. Evaluation

The evaluation function of a chromosome measures the cost of the path p it represents. Since p can be either feasible (i.e., collision-free) or infeasible, we adopt two separate evaluation functions, $eval_f$ and $eval_i$, to handle these cases, respectively. For feasible paths, $eval_f$ is designed to accommodate three different optimization goals: 1) minimize distance traveled, 2) maintain a smooth trajectory, and 3) satisfy the clearance requirements (the robot should not approach the obstacles too closely). We have selected a linear combination of these three factors[2]

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

as a formula for calculating $eval_f$, 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 points 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 path points m_{i-1} and m_i and the line segment connecting points m_i and m_{i+1}.

- $clear(p) = \max_{i=1}^{n-1} c_i$, 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. When the distance between a path segment and the closest obstacle is smaller than τ, the penalty grows exponentially to discourage such close encounters strongly. The function $clear(p)$ is defined as the maximum of c_i's to make sure that if a certain segment of a path is dangerously close to an obstacle, i.e., within distance τ, then the path is penalized strongly even if all other path segments are safe.

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

For infeasible paths, our design of $eval_i$ takes into account several factors: 1) the number of intersections of a path with obstacles, 2) the depth of intersection (i.e., how deep a path cuts through obstacles), 3) the ratio between the numbers of feasible and infeasible segments, 4) the total lengths of feasible and infeasible segments, and so on, as detailed in [20]. To determine the worst path in the whole population we assume that the worst feasible path is better (or fitter) than the best infeasible path.

C. 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 a path of an arbitrary shape, but each may not be applicable or needed in a given situation. The application of each operator is probabilistic. Note that all operators only change the intermediate nodes of a chromosome. Now we introduce these eight operators, which are also illustrated in Fig. 3.

Crossover 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 numbers of nodes in the two parent paths.

Mutate_1 is used for fine-tuning node coordinates in a feasible path for shape adjustment. Given a path, the operator randomly selects[3] its intermediate nodes for adjusting their coordinates within some local clearance of the path so that the path subsequently remains feasible.

Mutate_2 is used for imposing a large random change node coordinates in a path, which can be either feasible or infeasible. Given a path, the operator randomly selects an intermediate node and changes the coordinates of this node randomly.

Insert–Delete operates on an infeasible path by inserting randomly generated new nodes into infeasible path segments

[1] Note that all paths have the same starting point and goal point. Thus, paths only differ because of different intermediate (i.e., knot) points.

[2] The linear combination is very simple and consequently allows the investigation on significance of all components involved in the expression.

[3] Here and in the rest of the descriptions of operators, "random selection" means selection with equal probability for all outcomes.

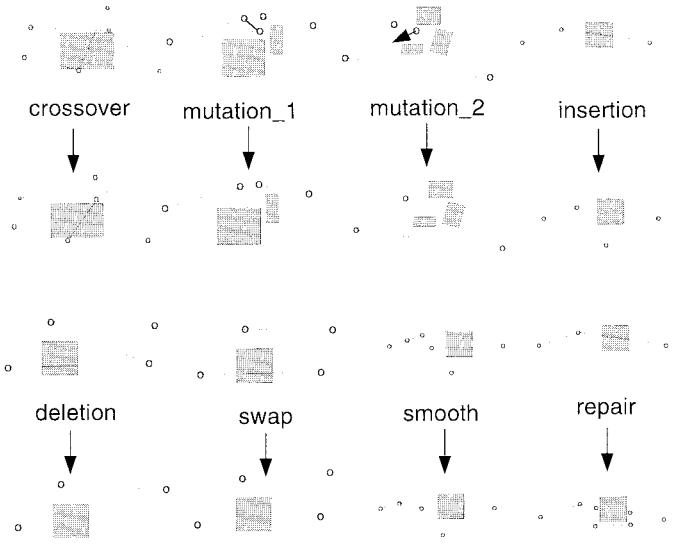


Fig. 3. The roles of the operators. The upper part of each of the eight pairs of diagrams represents a subpath (or two subpaths in the case of crossover) before the operator is applied, whereas the lower diagram shows a possible outcome after application of the operator.

and deleting infeasible nodes (i.e., nodes that are inside obstacles).

Delete removes nodes from a path, which can be either feasible or infeasible. If the path is infeasible, nodes for deletion are selected randomly in the chromosome. Otherwise, the operator decides whether or not a node should be deleted based on some heuristic knowledge. In the case where there is no knowledge supporting the deletion of a node, its selection for deletion is decided randomly with a small probability.

Swap exchanges the coordinates of selected adjacent nodes in a chromosome to eliminate two consecutive sharp turns (Fig. 3). The path can be either feasible or infeasible. The probability for selecting a node n_i and the next node n_{i+1} is proportional to the sharpness of the two turns (measured by angles between the path segments) at the two nodes.

Smooth 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 fixes 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 for this problem. Note that since most knowledge needed is available from the evaluation of path cost (see the previous subsection), the operators mostly use the knowledge with little extra computation.

III. Performance and Probability Tuning

The firing probability p_i $(i = 1, \cdots, 8)$ of each operator governs the contribution or role of the operator to the whole evolution process. Different values of these probabilities affect the overall performance of the EP/N. As the system uses many operators, properly determining their probabilities is not a trivial matter, particularly 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 to adapt these probabilities to achieve the best results by a systematic method.

A. 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. Note that the length of a chromosome affects both the processing time and the storage space needed by the chromosome—the more nodes that are in the chromosome, the more space and time (i.e., evaluation time and often operation time) are needed. So, if an operator alters the number of nodes in a chromosome, the effect will be felt in future processing. Such an effect can be either positive or negative on the processing cost of future generations,[4] 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 chromosomes 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, $i = 1, \cdots, 8$, we have the following.

- 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

[4] 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).

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 $[T_1, T_2]$.

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 nonnegative.

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 that 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

$$p_i = \frac{I_i}{\sum_{j=1}^{8} I_j}. \tag{1}$$

- 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 lookup 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 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 $I_i(0, \delta T)$, $i = 1, \cdots, 8$ 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. Clearly, the method can be refined further by incorporating a concept of "sliding window" with a horizon δT, such that the probabilities are updated every cycle based on the last δT generations. In such a case, however, the computational effort for recalculating all probabilities is much higher. Moreover, the EP/N is a steady-state evolutionary system where only one operator is applied within a single generation, so the effort of recalculating all probabilities every generation would not pay off.

B. Adapting Operator Probabilities

Based on the procedure of computing the operator performance indexes I_i's in the EP/N, we have used the following method to enable the EP/N to 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 (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 IV) that compared 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.

C. System Performance Measures

We use the following measures to evaluate the performance of the 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 indexes mean *better* effectiveness and *better* efficiency, respectively. To improve system performance is to reduce the values of those indexes.

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), the diversity of feasible paths, and so on, depending on the need.

IV. 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 tuning ability and overall performance. Fig. 4 shows six sample tasks in six different environments where, for each task, a near-optimal path obtained by the adaptive EP/N is displayed. Note that the same values of the EP/N parameters were used for all tasks, except the operator probabilities, which were adapted (with adaptation occurred every 100 generations). Different complexities of the environments were reflected by the different T generations of evolution needed to obtain the near-optimal results displayed. However, $T = 600$ are usually sufficient to achieve very good results in all cases, and $T = 400$ are usually sufficient to create feasible paths of reasonable shape in all cases. Fig. 5 shows

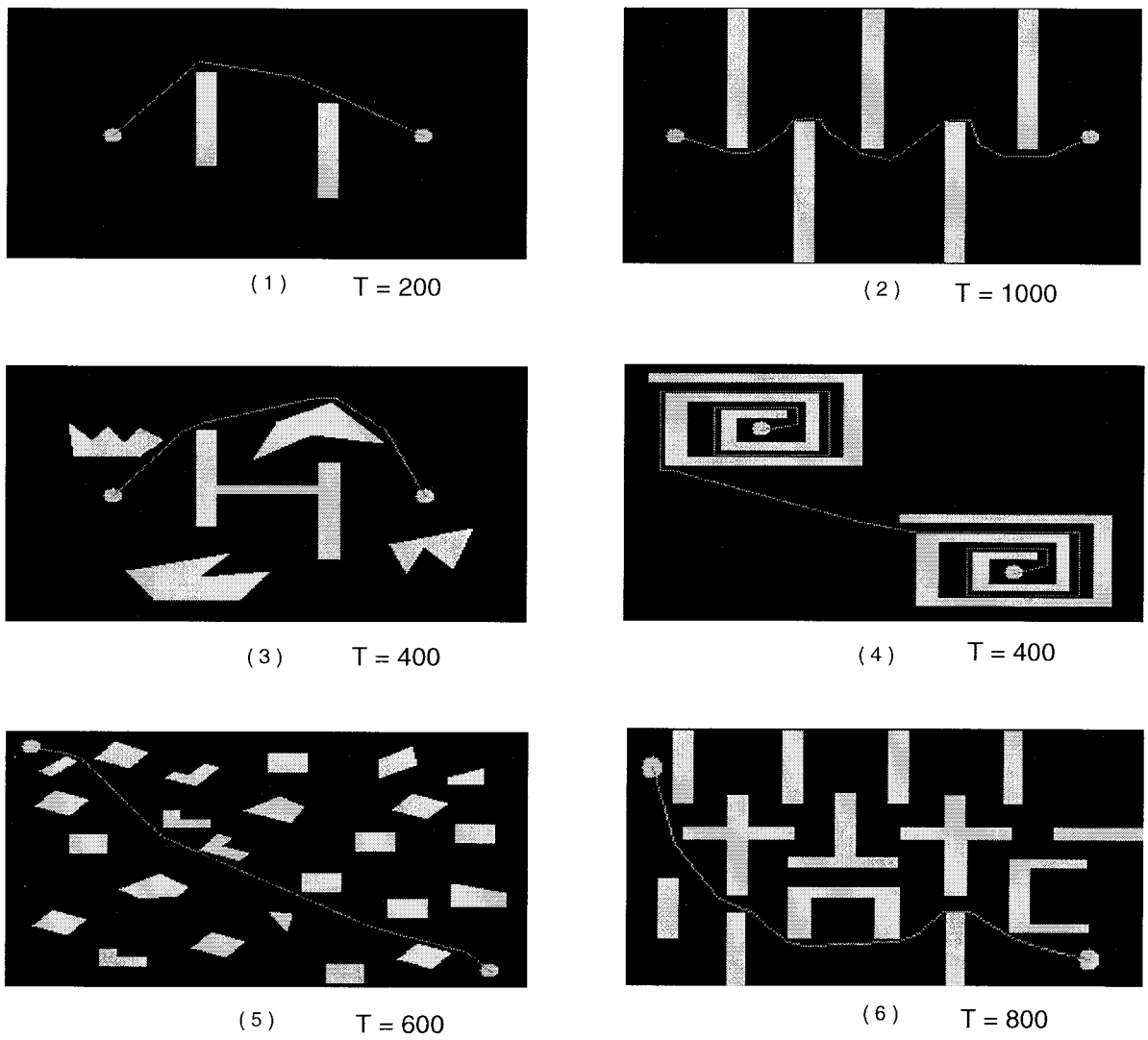


Fig. 4. Six tasks in six different environments selected for experiments and near-optimal paths found by the EP/N (adaptive version) in T generations (as indicated). Other parameters of the EP/N were set to be the same for all six environments: The population size was set to be 30, and the coefficients 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, and 10, respectively.

the snapshots of one evolution process of paths for the task in Environment 6 in $T = 400$ generations. Very reasonable paths were already formed even with only 400 generations.

Table I shows, for $T = 400$ generations and each task shown in Fig. 4, 1) the average time t_T of running the EP/N with adapted probabilities over 100 repeated runs[5] and 2) the average gain on both system effectiveness and efficiency when the EP/N adapted operator probabilities against the case when the EP/N used fixed, equal operator probabilities over 100 repeated runs.[6] As expected, values of both system effectiveness and efficiency indexes are *reduced*, which means *better* effectiveness and efficiency was achieved in most of the cases.

Compared to the results obtained by running the EP/N with fixed, manually determined operator probabilities [20], the results with adaptive probabilities (as presented here) show significant improvement of the system performance.

V. On-Line Navigation

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

- Obstacles in the environment are either known or unknown. In the current implementation they are assumed to be static.
- The robot has a range of view (described by R, parameter of the robot). If, due to the 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 (i.e., its complete dimensionality is given). Of course, to be more realistic, this assumption can easily be revised

[5] All runs were made on a Silicon Graphics station.

[6] Note that [21] presented some results averaged over 30 runs. We later find that 30 runs are not enough statistically and 100 runs are more reasonable.

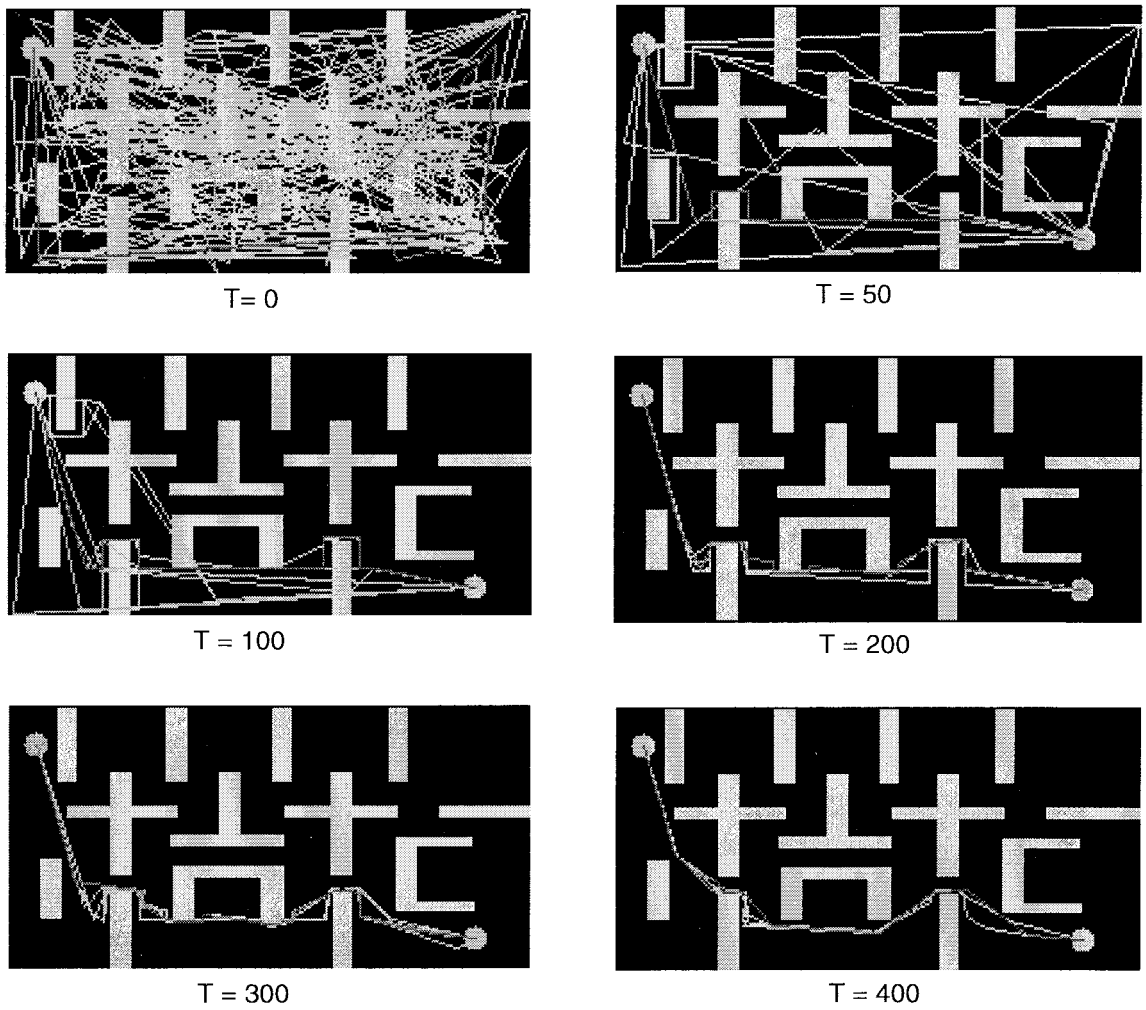


Fig. 5. Snapshots of path evolution at different generations (T) for Environment 6, where two-thirds of population are shown and the EP/N is under the same condition as in Fig. 4.

TABLE I
AVERAGE TIME AND GAIN (OVER 100 RUNS) ON SYSTEM PERFORMANCE AGAINST THE CASE WITH EQUAL OPERATOR PROBABILITIES IN $T = 400$ GENERATIONS. THE "EFFECTIVENESS" AND "EFFICIENCY" COLUMNS REPORT THE PERCENTAGE GAINS ON THE EFFECTIVENESS INDEX AND THE EFFICIENCY INDEX DEFINED IN SECTION III-C, RESPECTIVELY

env	t_T	effectiveness		efficiency	
	(sec)	avg_T%change	$best_T$ %change	$avg_T \times t_T$ %change	$best_T \times t_T$ %change
1	0.24	-1.77%	-1.17%	-1.20%	-0.60%
2	0.64	-4.03%	-0.39%	-1.14%	2.61%
3	1.19	-4.59%	-2.79%	-12.76%	-11.11%
4	1.96	-4.80%	-3.43%	-4.90%	-3.52%
5	2.83	-2.28%	-1.41%	-6.30%	-5.46%
6	2.63	-5.48%	-4.96%	-7.88%	-7.39%

as only the part of the object boundary that is inside the robot's range of view is known. As our current focus is on testing the robot's adaptation capability to the discovery of unknowns in an environment, however, it does not matter much at present how such discovery is actually accomplished.

In the on-line navigation, the result of evolution is checked after every n generations (see the second "if" statement in the procedure of Fig. 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 a new obstacle is added to the robot's map of the 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 increase significantly. The previous best path may no longer be best, and a new best path may need to be found.

As mentioned in Section II, during such on-line navigation, the starting location of each path in a population is constantly updated to reflect the current location of the robot as it moves. Given a path to be updated, this process involves the application of an additional operator, short-cut, which connects the current position of the robot to the furthest knot point of the path (provided, of course, that the resulting path is feasible).

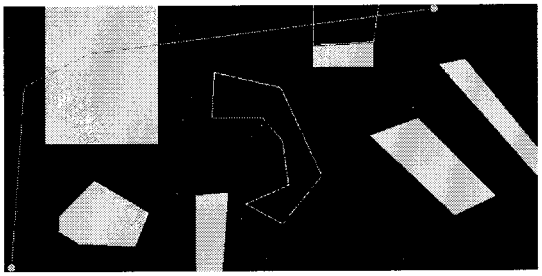


Fig. 6. The best path in the initial population. The shaded obstacles are known. The outlined obstacles are unknown. The initial best path crosses two obstacles, one known and one unknown.

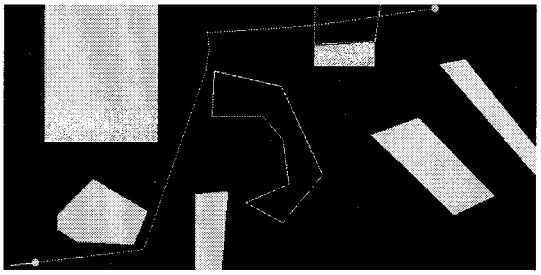


Fig. 7. The first feasible path. The robot makes its initial move upon generating the first feasible solution given known obstacles.

A. An Example

The actions of the robot guided by the on-line navigation process are illustrated by the following example. Fig. 6 displays the robot's environment. The starting 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). During initialization, a set of randomly generated paths is developed; very likely none of these initial paths will be feasible (Fig. 6 displays 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 based only on known obstacles so that it does not have to be truly feasible (Fig. 7): it might intersect an unknown obstacle. Note also that the remaining parts of the paths are continuously optimized; the current best path in Fig. 7 is different from that in Fig. 8.

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 (Figs. 8 and 9). At this moment, all paths in the current population are reevaluated, and as the evolution process continues, the robot eventually follows a newly emerged best feasible path (Fig. 10), which is subsequently improved (Fig. 11).

When the second unknown obstacle is discovered, again, all paths in the population are reevaluated, and the robot adjusts its motion accordingly by following a newly emerged best path (Fig. 12).

The actual path traversed by the robot is displayed in Fig. 13 (note again the improvements in the final segments of the path from those displayed in Fig. 12). This path is far from being optimal in comparison with an "ideal" path which would emerge if all obstacles in the environment were known *a priori* (Fig. 14). It is unfair, however, to simply compare such a real path traversed as the result of on-line handling of unknown obstacles to the "ideal" path.

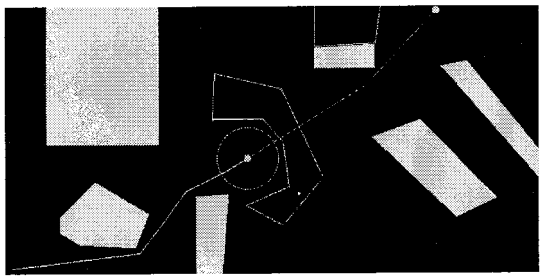


Fig. 8. Before sensing the first unknown obstacle. The robot has moved to a position where the outlined obstacle is still unknown. Note that the path has been optimized with respect to distance and smoothness, as compared to the path shown in Fig. 7.

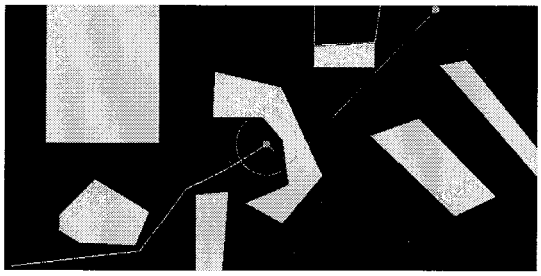


Fig. 9. Sensing and reevaluation. The robot senses the first unknown obstacle. Its planned path is no longer feasible.

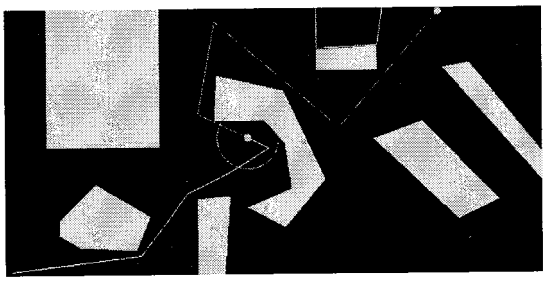


Fig. 10. The best feasible path after sensing the first unknown obstacle. The EP/N planner has invented a path around the newly discovered obstacle.

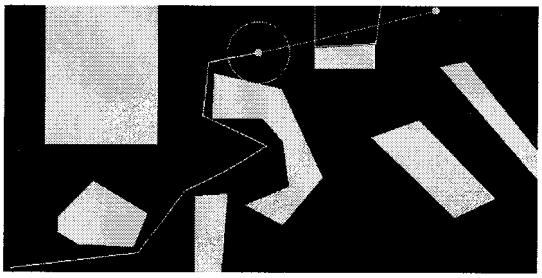


Fig. 11. Before sensing the second unknown obstacle. The robot has traversed around the first (previously) unknown obstacle and is approaching the second.

B. 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

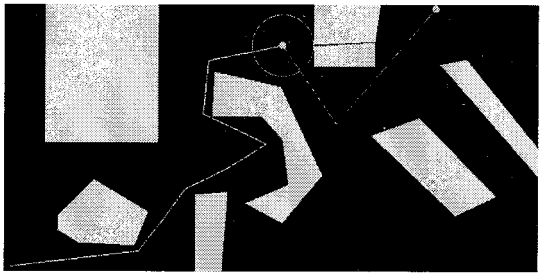


Fig. 12. After sensing the second unknown obstacle. The second unknown obstacle has come within sensor range and the EP/N has generated a path around it.

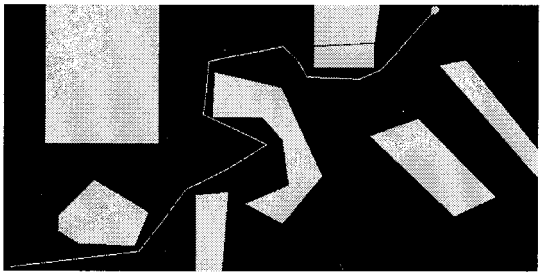


Fig. 13. The robot's real path as executed over time.

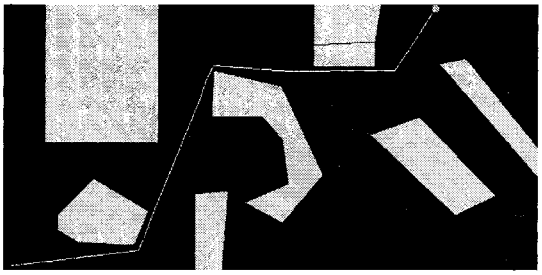


Fig. 14. The robot's "ideal" path for the completely known environment. The path differs from the path actually developed by the EP/N because two of the obstacles were unknown *a priori*.

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 to an ideal path generated to connect the fragment's start and goal locations,[7] 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 the number of fragments;
- e_i the relative error of the ith fragment (in path cost);
- g_i the number of generations passed during the traversal of the ith fragment;
- $N = \sum_{j=1}^{f} g_j$ i.e., N represents the total number of generations.

[7] 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 II
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 III
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

TABLE IV
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

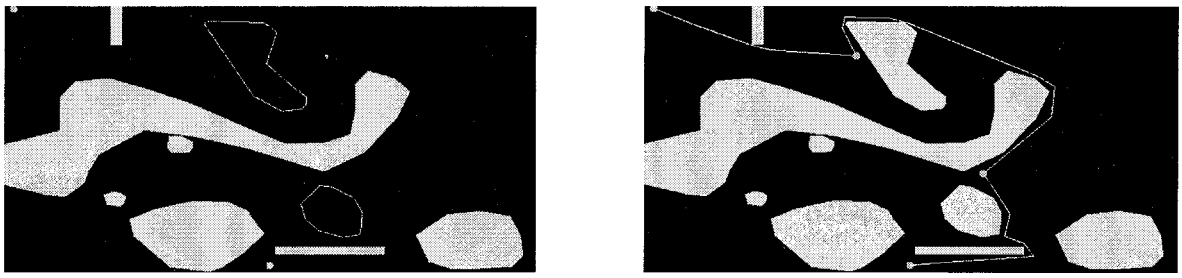
TABLE V
RESULT OF EXPERIMENTS FOR $n = 20$; THE RESULTING NUMBER OF GENERATIONS $N = 960.3$; ERROR $E = 0.064764$

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

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 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).

We now discuss how E is related to n, the number of generations between the robot's steps. Tables II–V show results of experiments using the environment shown in Fig. 15, with dimensions 400 $\times$ 500 (averaged over 100 trials) 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 $R = 45$.

Results reported in Tables II–V 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.



(a) (b)

Fig. 15. (a) Experimental environment and (b) three fragments of a path are clearly marked.

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 II–V), 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 performed in parallel, the total time of path traversal t is the maximum of t_N and the total time of the 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 running the EP/N off line (Section IV), 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 II–V) is usually in the order of a few seconds, which should be well below the time required for a current land-based 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 efficiency in real-time navigation of a robot, especially when the environment is only partially known.

VI. Conclusions

The adaptive EP/N presented in this paper is particularly suitable for dealing gracefully, effectively, and efficiently with the diversities, changes, and unknowns in an environment. Results from off-line planning with adaptive probabilities of genetic operators and simulation of on-line navigation confirm the flexible nature and robustness of such an evolutionary system. We are currently implementing the on-line process of the EP/N on a Khepera robot.

The EP/N also exposes many interesting challenges of general importance to evolutionary computation. One of the most significant is how to take the full advantage of adaptiveness of an evolutionary system and tune its various parameters during the execution of the system. The adaptive EP/N uses an automatic mechanism to measure performances of its genetic operators and adapt the operator probabilities accordingly. The general nature of the strategy makes it applicable to other evolutionary systems as well. 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 interest. One determines how frequently operator probabilities should be adjusted or adapted (i.e., the generation interval $[T_1, T_2]$'s in Section III). Parameters n and k_{max} in the on-line process (Section V), as well as the velocity of a 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 V-B), 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 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 we have incorporated domain knowledge in both fitness evaluation and operators, there are other components/processes, such as the initialization process, which may by improved as a result of greater 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 and/or 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 adding memory to store the knowledge from the robot's past exploration of the environment together with the knowledge from previous navigation tasks to facilitate more efficient and effective planning in the future. We have performed some preliminary studies in adding local memories to chromosomes for storing "valuable" paths or segments of paths discovered earlier [19]. The experiments demonstrate the potential of such an extension to the EP/N in improving planning effectiveness in partially known environments. It could also be interesting to study other forms of "memory," such as one based on multichromosome structures with a dominance function [6] or employing machine learning techniques.

ACKNOWLEDGMENT

The authors would like to thank D. Fogel and the anonymous referees for their constructive comments, which improved readability of the paper.

REFERENCES

[1] R. C. Arkin, "Motor schema-based mobile robot navigation," *Int. J. Robot. Res.*, vol. 8, no. 4, pp. 92–112, Aug. 1989.
[2] P. Bessiere, J.-M. Ahuactzin, A.-G. Talbi, and E. Mazer, "The 'Ariadne's Clew' algorithm: Global planning with local methods," in *Proc. 1993 IEEE-IROS Int. Conf. Intelligent Robots and Systems*, Yokohama, Japan, Sept. 1993.
[3] J. Borenstein and Y. Koren, "The vector field histogram—Fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Automat.*, vol. 7, no. 3, pp. 278–287, June 1991.
[4] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Trans. Robot. Automat.*, vol. 2, pp. 14–23, 1986.
[5] G. Foux, M. Heymann, and A. Bruckstein, "Two-dimensional robot navigation among unknown stationary polygonal obstacles," *IEEE Trans. Robot. Automat.*, vol. 9, pp. 96–102, 1993.
[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison Wesley, 1989.
[7] C. Hocaoğlu and A. C. Sanderson, "Planning multi-paths using speciation in genetic algorithms," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computation*, Nagoya, Japan, May 1996, pp. 378–383 .
[8] O. Khatib, "Real-time obstacles avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, pp. 90–98, 1986.
[9] J. C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.
[10] H.-S. Lin, J. Xiao, and Z. Michalewicz, "Evolutionary navigator for a mobile robot," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Diego, CA, May 1994, pp. 2199–2204.
[11] ——, "Evolutionary algorithm for path planning in mobile robot robot environment," in *Proc. First IEEE Int. Conf. Evolutionary Computation*, Orlando, FL, June 1994, pp. 211–216.
[12] V. J. Lumelsky, "A comparative study on the path length performance of maze-searching and robot motion planning algorithms," *IEEE Trans. Robot. Automat.*, vol. 7, no. 1, pp. 57–66, Feb. 1991.
[13] V. J. Lumelsky and A. A. Stepanov, "Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, pp. 403–430, 1987.
[14] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. New York: Springer-Verlag, 1996.
[15] Z. Michalewicz and J. Xiao, "Evaluation of paths in evolutionary planner/navigator," in *Proc. 1995 Int. Workshop on Biologically Inspired Evolutionary Systems*, Tokyo, Japan, May 1995, pp. 45–52.
[16] J. B. Oommen, S. S. Iyengar, N. S. V. Rao, and R. L. Kashyap, "Robot navigation in unknown terrains using visibility graphs: Part I: The disjoint convex obstacle case," *IEEE Trans. Robot. Automat.*, vol. RA-3, pp. 672–681, 1987.
[17] W. C. Page, J. R. McDonnell, and B. Anderson, "An evolutionary programming approach to multi-dimensional path planning," in *Proc. First Annu. Conf. Evolutionary Programming*, Evolutionary Programming Society, San Diego, CA, 1992, pp. 63–70.
[18] T. Shibata and T. Fukuda, "Intelligent motion planning by genetic algorithm with fuzzy critic," in *Proc. 8th IEEE Int. Symp. Intelligent Control*, Chicago, Aug. 25-27, 1993, pp. 565–570.
[19] K. Trojanowski, Z. Michalewicz, and J. Xiao, "Adding memory to an evolutionary planner/navigator," in *Proc. 4th IEEE Int. Conf. Evolutionary Computation*, Indianapolis, IN, 13-16 Apr. 1997.
[20] J. Xiao, "Evolutionary planner/navigator in a mobile robot environment," in *Handbook of Evolutionary Computation*, T. Bäck, D. Fogel, and Z. Michalewicz, Eds. New York: Oxford Univ. Press and Institute of Physics, 1997.
[21] J. Xiao, Z. Michalewicz, and L. Zhang, "Evolutionary planner/navigator: Operator performance and self-tuning," in *Proc. 3rd IEEE Int. Conf. Evolutionary Computation*, Nagoya, Japan, May 1996, pp. 366–371.
[22] C.-K. Yap, "Algorithmic motion planning," in the *Advances in Robotics, Vol. 1: Algorithmic and Geometric Aspects of Robotics*, J. T. Schwartz and C.-K. Yap, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 95–143.
[23] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Trans. Robot. Automat.*, vol. 8, pp. 707–717, 1992.
[24] M. Zhao, N. Ansari, and E. Hou, "Mobile manipulator path planning by a genetic algorithm," in *Proc. 1992 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Raleigh, NC, July 7-10, 1992, pp. 681–688.

Jing Xiao (S'88–M'90) received the B.S. degree in physics and electrical engineering from Beijing Normal University, Beijing, China, in 1982 and the M.S. and Ph.D. degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1984 and 1990, respectively.

Since 1990, she has been on the faculty of the Department of Computer Science at the University of North Carolina at Charlotte, where she is currently an Associate Professor. Her research interests include studying the effect of uncertainties on the performance of robotic tasks, sensor-based reasoning and planning strategies to deal with such effects, and evolutionary computation approach to planning problems.

Zbigniew Michalewicz received the M.Sc. degree from the Technical University of Warsaw, Poland, in 1974 and the Ph.D. degree from the Institute of Computer Science, Polish Academy of Sciences, Poland, in 1981.

He is Professor of Computer Science at the University of North Carolina at Charlotte. His current research interests are in the field of evolutionary computation. He has published a monograph (three editions) *Genetic Algorithms + Data Structures = Evolution Programs* (New York: Springer-Verlag, 1996)

Dr. Michalewicz was the General Chairman of the First IEEE International Conference on Evolutionary Computation, Orlando, FL, June, 1994. He is a member of many program committees and international conferences and has been an invited speaker for many international conferences. He is a member of the editorial board of the following publications: *Statistics and Computing, Evolutionary Computation, Journal of Heuristics,* IEEE TRANSACTIONS ON NEURAL NETWORKS, *Fundamenta Informaticae,* IEEE TRANSACTIONS ON EVOLUTIONARY COMPUATION, and *Journal of Advanced Comutational Intelligence.* He is an Editor-in-Chief of the *Handbook of Evolutionary Computation* and Chairman of the IEEE NNC Evolutionary Computation Technical Committee.

Lixin Zhang (S'95–A'95) received the B.E. and M.E. degrees in automatic control from Tsinghua University, P.R. China, in 1991 and 1993, respectively, and the M.S. degree in computer science from the University of North Carolina at Charlotte in 1996.

He is currently with Datastream System, Inc., Greenville, SC. His research interests include robot fine motion planning and path planning, computational geometry, and computer simulation and animation.

Krzysztof Trojanowski received the M.Sc. degree in computer science from the Warsaw Technical University, Poland, in 1994. He is currently pursuing the Ph.D. degree at the Institute of Computer Sciences, Polish Academy of Sciences, Warsaw, Poland, under the supervision of Prof. Z. Michalewicz.

His research interests include evolutionary computation techniques.