Mobile Manipulator Path Planning by A Genetic Algorithm

Min Zhao, Nirwan Ansari, and Edwin S. H. Hou*
Center for Communications and Signal Processing
Electrical and Computer Engineering Department
New Jersey Institute of Technology
University Heights
Newark, New Jersey

Received May 9, 1992; accepted February 25, 1993;
revised June 1, 1993

This article addresses the path-planning problem for a mobile manipulator system that is used to perform a sequence of tasks specified by locations and minimum oriented force capabilities. The problem is to find an optimal sequence of base positions and manipulator configurations for performing a sequence of tasks given a series of task specifications. The formulation of the problem is nonlinear. The feasible regions for the problem are nonconvex and unconnected. Genetic algorithms applied to such problems appear to be very promising while traditional optimization methods cause difficulties. Computer simulations are carried out on a three-degrees-of-freedom manipulator mounted on a two-degrees-of-freedom mobile base to search for the near optimal path-planning solution for performing the sequence of tasks.

ここでは、位置と最少配向力の能力によって特性を与えられるタスク・シーケンスの実行に使われる、モービル・マニピュレータ・システム用の経路計画問題について説明する。この問題では、一連のタスク特性が与えられているタスクのシーケンスを実行するために必要な、ベース・ポジションとマニピュレータ配置の最適シーケンスを求める。この問題に適しているのは、凹凸が無く、連結されていない領域である。このような領域では、従来の最適化方法では障害が発生するが、ここで提案するアルゴリズムは良好な結果が期待できる。2d.o.f. のモービル・ベースにマウントされた 3d.o.f. のマニピュレータに対して、コンピュータ・シミュレーションを行い、タスク・シーケンスの実行に必要な最適経路計画の近似解を求めている。

*To whom all correspondence should be addressed.

Journal of Robotic Systems 11(3), 143–153 (1994)

CCC 0741-2223/94/030143-011

1. INTRODUCTION

The mobile manipulator path-planning problem is to find base trajectories for performing a sequence of tasks. Here, path is defined as the space curve that the base moves on, and the path-planning task is to search, define, and describe some optimal path for a mobile manipulator. This idea, first proposed and studied by Carriker and his colleagues,[1,2] is motivated by the following two considerations:

1. Manipulator mobility: In mobile manipulator applications, a common task description involves acquiring, moving, and depositing material from one point to another over a large space. Thus, mobility is required to accomplish the task, which could possibly not be accomplished by a stationary manipulator, and it is desirable to plan optimal point-to-point trajectories.
2. Manipulator redundancy: Although a stationary robot with 6 degrees of freedom (dof) is sufficient to achieve translational and rotational end-effector motion with reference to the base in any arbitrary direction within the workspace of the robot, the dexterous workspace is only a small part of the whole workspace.[3] This restriction greatly limits the usefulness of a manipulator.

By mounting a manipulator on a mobile base, one can introduce a mobile manipulator system. While redundancy introduced by extra manipulator joints can increase system flexibility within a fixed workspace,[3–8] redundancy introduced by base mobility can be used to increase the useful workspace of the system and can increase flexibility. Thus, rather than using the redundancy to achieve optimal manipulator configurations for each task, one can use it to solve a more global optimization problem by finding the optimal base trajectories for performing a sequence of tasks while guaranteeing the manipulator to be within a feasible region to perform each task.

The mobile manipulator path-planning problem is formulated as a nonlinear and nonpolynomial optimization problem. The feasible regions, in which the mobile base can be located to perform a task without violating the physical constraints of the manipulator, are unconnected and nonconvex.[2] Solutions to the problem obtained by any gradient descent method depend on the initial conditions. The optimal solution will be obtained only if the initial guess is within a feasible convex region surrounding the optimal solution. Otherwise, the search will encounter the constraints, and produce a nonoptimal solution. To avoid this difficulty, one can make a number of initial guesses, and pass each guess to the numerical method. However, the number of initial guesses grows rapidly with the number of tasks. An improvement[1] is to generate new initial points based on the results of previous executions of the numerical optimization algorith.

In Carriker et al.,[2] simulated annealing is introduced as a search method for the near global optimal solution to this problem. Simulated annealing[9,10] is a stochastic computational technique derived from statistical mechanics for finding near globally minimum cost solutions to large optimization problems. Carriker et al. implemented a version of the basic simulated annealing algorithm for the mobile manipulator path-planning problem, using a constant Markov chain length at each temperature and a constant cooling rate. Simulated annealing technique is robust for many complicated optimization problems. However, it imposes limitations on the cost function of the problem, and it is computationally intensive to find satisfactory near optimal results for many problems.[2]

In this article, we have developed another robust technique, a genetic algorithm, to solve the path-planning problem for the mobile manipulator system used by Carriker et al.[1,2]

2. PROBLEM FORMULATION

The problem we try to solve in this article is to find an optimal sequence of base positions and manipulator configurations for performing a sequence of tasks, given a sequence of task specifications characterized by the required end-effector locations in the three-dimensional space and minimum force capabilities in specified directions.

First, the position of the manipulator end-effector can be described by:

$$X_{tip} = X_{base} + X_{man}(\Phi) \tag{1}$$

where X_{tip} is a three-dimensional vector specifying the location of a task with respect to a global Cartesian reference frame, X_{base} is the location of the mobile base with respect to the global frame, and $X_{man}(\Phi)$ is a vector function that maps the manipulator configuration vector Φ into the end-effector position relative to a Cartesian reference frame fixed on

the mobile base. It can be assumed in general that the components of the manipulator joint position are constrained by independent upper and lower bounds specified by the vectors Φ, $\overline{\Phi}$. Thus, the constraints can be described by:

$$\underline{\Phi} \leq \Phi \leq \overline{\Phi} \tag{2}$$

where the vector inequalities are applied componentwise.

The three-dimensional vector force F at the end-effector is related to the manipulator joint torques τ through the manipulator Jacobian matrix, $J(\Phi)$.[2] This relationship is expressed by:

$$\tau = J(\Phi)^T F \tag{3}$$

where τ is a vector of which each vector component is the torque exerted on its respective joint.

There are independent bounds on the components of the torque vector given by:

$$\underline{\tau} \leq \tau \leq \overline{\tau} \tag{4}$$

where $\underline{\tau}$ and $\overline{\tau}$ are the lower and upper bounds of τ, respectively.

The sequence of tasks to be performed is specified by two sets of vectors $\{X_{\text{tip},1}, X_{\text{tip},2}, \cdots, X_{\text{tip},n}\}$ and $\{F_1, F_2, \cdots, F_n\}$ that specify the location and force vectors, respectively, required to perform a sequence of n operations or tasks in the workspace. These sets of vectors impose constraints on the sequence of manipulator configurations and base locations that are feasible for performing the sequence of tasks. For example, to perform the ith task, the specification indicates that the end-effector must be at location $X_{\text{tip},i}$, with the capability to apply a force of F_i. These specifications for the ith task impose constraints on the base position $X_{\text{base},i}$ and joint configuration Φ_i expressed by the following equality and inequality constraints.

$$X_{\text{tip},i} = X_{\text{base},i} + X_{\text{man}}(\Phi_i) \tag{5}$$

$$\tau_i = J(\Phi_i)^T F_i \text{ and} \tag{6}$$

$$\underline{\tau} \leq \tau_i \leq \overline{\tau} \tag{7}$$

Provided the above constraints are satisfied for each of the n tasks, there will, in general, be a range of base positions and manipulator joint configurations that satisfy each of the task specifications. To choose the most desirable positions and configurations for each of the tasks, we introduce a cost function that in some sense measures the energy or time required to move the system through the complete sequence of tasks. In general,[11] the cost function is expressed as:

$$C = \alpha C_{\text{base}} + \beta C_{\text{man}} \tag{8}$$

with:

$$C_{\text{base}} = \sum_{i=1}^{n} \delta_b(X_{\text{base},i-1}, X_{\text{base},i}) \tag{9}$$

$$C_{\text{man}} = \sum_{i=1} \delta_m(\Phi_{i-1}, \Phi_i) \tag{10}$$

where δ_b and δ_m are the generalized distance functions representing the relative cost of moving the base and manipulators, respectively, between two points. Thus C_{base} is the cost for moving the base through the complete sequence of n tasks. Likewise, C_{man} is the cost of having the manipulator taken into various configurations in completing the n tasks. α and β are weight factors for the cost of moving the base and the cost of changing the configuration of the manipulator, respectively.

In short, the problem is defined as follows: given a task set S, and a mobile manipulator system, determine a path for the system such that all the tasks in the set can be performed with the minimum cost C:

$$\min\{C\} = \min\{\alpha C_{\text{base}} + \beta C_{\text{man}}\} \tag{11}$$

subject to:

$$X_{\text{tip},i} = X_{\text{base},i} + X_{\text{man}}(\Phi_i), \forall i, \tag{12}$$

$$\tau_i = J(\Phi_i)^T F_i, \forall i, \tag{13}$$

with the following manipulator joint position and torque constraints:

$$\underline{\Phi} \leq \Phi_i \leq \overline{\Phi}, \forall i, \tag{14}$$

$$\underline{\tau} \leq \tau_i \leq \overline{\tau}, \forall i. \tag{15}$$

3. THE GENETIC ALGORITHM APPROACH

Since the publication of John Holland's *Adaptation in Natural and Artificial Systems*,[12] there has been a surge of research efforts in genetic algorithms. Genetic al-

gorithms are robust stochastic searching algorithms for various optimization problems. This class of methods is based on the mechanics of natural selection and natural genetics that combines the notion of survival of the fittest, random and yet structured search, and parallel evaluation of the nodes in the search space. A genetic algorithm consists of a string representation (*genes*) of the node in the search space, a set of genetic operators for generating new search nodes, a fitness function to evaluate the search nodes, and a stochastic assignment to control the genetic operators.

Genetic algorithms have been successfully applied to solve many complicated problems,[13,14] such as those of the design of a communication network, the very large scale integration (VLSI) circuit layout, the Traveling Salesman Problem, image classification, and pattern recognition. One of the most important questions pertains to the applicability of genetic algorithms, i.e., what type of problem is well-fitted for the genetic algorithms. Davis[13] provides an overview on the problem domain that is suitable for the application of genetic algorithms: the domain should contain multiple extrema; the domain should exhibit some epistasis[13] (an epistasis is the inhibition of one part of a solution by the action of another); the domain should exhibit detectable regularity that can be encoded into strings. The mobile manipulator path planning is a nonlinear optimization problem with nonlinear, nonconvex constraints and unconnected feasible regions. As the characteristics of the mobile manipulator path-planning problem fits the applicability of genetic algorithms, we develop this kind of algorithm to solve the problem.

To solve the manipulator path-planning problem by a genetic algorithm, we need to choose a coding scheme to encode the parameters of the problem into genetic strings. In this problem we can code either the base position (X_{base}) or the manipulator configuration (Φ). For detectable regularity and efficiency, manipulator configuration Φ is chosen to be coded. Each element $\theta_i (i = 1, \ldots, j-1, j+1, \ldots, m)$ of the manipulator configuration vector Φ, except one element θ_j, is coded into a binary string. θ_j can be expressed in terms of θ_i, for $i \neq j$, by the constraint of eq. (12). The genetic string is formed by concatenating the codes of the element $\theta_i (i = 1, \ldots, j = 1, j+1, \ldots, m)$.

Genetic algorithms use the fitness value of each string of the current generation to decide if and how many copies of the string should be passed to the next generation. The larger the fitness value of one string is, the higher probability of the string being chosen for the next generation. To reflect this characteristic of genetic algorithms, a fitness function is always designed in such a way that a "good" string scores a large fitness value and a "bad" one scores a small fitness value. The fitness value can never be negative. In our genetic algorithm the fitness function is defined by

$$f = C_m - C + \frac{C_m}{2}(1 - I), \tag{16}$$

where C_m is any positive real number not less than the maximum cost C (see eq. (8)). Because finding the feasible maxima is as difficult as finding the feasible minimia, C_m is used here instead of the maximum of C to enforce the non-negativity of the fitness function. I is an indicator function, which is defined by:

$$I = \begin{cases} 0 & \text{if } \underline{\tau} \leq \tau_i \leq \bar{\tau}, \quad \text{for all } i, \text{ i.e., for all tasks} \\ 1 & \text{otherwise} \end{cases}$$

$\frac{C_m}{2}(1 - I)$ is used to induce a large fitness value when the constraint of eq. (15) is satisfied. The indicator function is used as a penalty method instead of the quadratic function suggested by Goldberg.[15] Therefore, violation is penalized regardless of the severity of the violation because if constraints are violated, no matter how severe they are, the manipulator cannot function. Thus by encoding the manipulator configuration parameters into genetic binary strings and by incorporating a moderate violation penalty into the fitness function, the search result is guaranteed to produce feasible solutions.

Although C_m can take on infinitely many values to ensure the non-negativity of the fitness function, it should not be arbitrarily large. A too-large value of C_m will diminish greatly the relative significance between a "good" and a "bad" string. On the other hand, a too small value of C_m will introduce a big difference in the fitness value between a "good" and a "bad" string. This will cause the takeover of the population by a superstring, which is not expected in the early iteration of the genetic algorithm. A constant is added or subtracted from the fitness function in each generation such that the string that has the least fitness value in the population is moved to a fixed reference value.

As the population is converging, competition among population members is less severe, and thus the simulation tends to wander. In this case, we use a tangent function to accentuate the relative signifi-

cance among population members to reward the good strings more. That is, the fitness function is first normalized to the range $[0, \frac{\pi}{2}]$, with the small fitness value being mapped to 0, and the $\frac{3C_m}{2}$ being mapped to $\frac{\pi}{2}$. Assuming f is normalized, then the modified form of the fitness function is defined by:

$$f' = \tan(f). \tag{17}$$

During the reproduction of the genetic algorithm, the probability of a string in a population being selected for further genetic operations is proportional to its fitness value. However, as pointed out by De Jong,[16] such a selection process will cause a stochastic error, i.e., there is a chance that a good string may not be selected for mating. The selection scheme is a high-variance process with a fair amount of scatter between expected and actual numbers of copies. To reduce the stochastic error associated with the selection, we implement the so-called stochastic remainder sampling without replacement. It works as follows: expected individual count values are calculated as *pselect* × n, where *pselect* is the normalized fitness value of the individual and n is the population size. The integer part of the product is the number of copies of the individual being reproduced. The fractional part of the product is treated as a probability of having another copy of the individual being reproduced.

Three genetic parameters have significant effects on the performance of the genetic algorithms. These parameters are: n (population size), p_c (crossover probability), and p_m (mutation probability). As a reasonable compromise between the speed to reach the acceptable result (on-line performance) and convergence (off-line performance), we choose n as 50, p_c as 0.8, and p_m as $\frac{1}{n}$, in our genetic programming.

The procedure is summarized below:

1. Use a finite length of binary string to represent each θ_i (for $i \neq j$), and form the genetic string by concatenating each of the string representative. Choose a population size n, a crossover probability p_c, and a mutation probability p_m.
2. Form an initial population of n strings through successive flips of an unbiased coin.
3. Reproduction:
   a. Calculate the fitness value of each string.
   b. Generate a new population of n strings by stochastic remainder sampling without replacement, which makes a number of copies of each string in present generation according to its fitness value.
4. Crossover: randomly pair the newly reproduced strings. For each pair of selected strings, with a probability of p_c, randomly select a position for a pair of strings and swap parts of the strings from the selected position to the last position to form a pair of new strings.
5. Mutation: with a small probability, p_m, randomly alternate the value of a string position.
6. Go back to step 3 (next generation), until a specified number of iterations is reached.

In the following simulations, each genetic string is encoded by 80 bits. A population size of 100 strings, $p_c = 0.6$ and $p_m = 0.02$ are used.

4. COMPUTER SIMULATION RESULTS

The system we consider is a 3 dof arm mounted on a 2 dof mobile base as shown in Figure 1. The joints with angles θ_1, θ_2, θ_3 are referred to as the base, shoulder, and elbow, respectively. The forward kinematic equations for the system are given by:

$$X_{tip} = X_{base} + \cos(\theta_1)(l_2\cos(\theta_2) - l_3\cos(\theta_2 + \theta_3)) \tag{18}$$

$$Y_{tip} = Y_{base} + \sin(\theta_1)(l_2\cos(\theta_2) - l_3\cos(\theta_2 + \theta_3)) \tag{19}$$

$$Z_{tip} = Z_{base} + l_1 + l_2\sin(\theta_2) - l_3\sin(\theta_2 + \theta_3)) \tag{20}$$

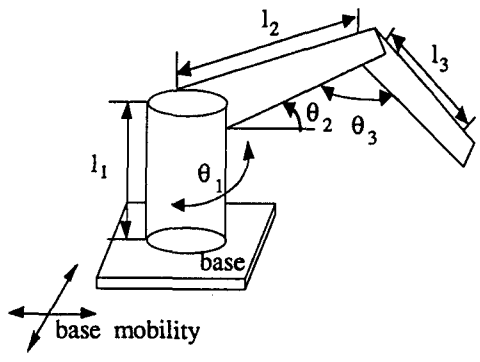


Figure 1. A mobile manipulator system with a 3-dof arm mounted on a 2-dof mobile base.

The manipulator Jacobian matrix[4] is thus:

$$J = \begin{bmatrix} -S_1(l_2C_2 - l_3C_{23}) & C_1(-l_2S_2 + l_3S_{23}) & C_1l_3C_{23} \\ C_1(l_2C_2 - l_3C_{23}) & S_1(-l_2S_2 + l_3C_{23}) & S_1l_3S_{23} \\ 0 & l_2C_2 - l_3C_{23} & -l_3C_{23} \end{bmatrix} \tag{21}$$

where $C_i = \cos(\theta_i)$, $S_i = \sin(\theta_i)$, $C_{ij} = \cos(\theta_i + \theta_j)$, $S_{ij} = \sin(\theta_i + \theta_j)$. For a given end-effector force capability F, the manipulator joint torque vector is given by:

$$\tau = J(\Phi)^T F$$

With the mobile manipulator system described above, we performed the following simulations:

1. First, the mobile manipulator system is used to perform two tasks with the locations and minimum oriented force capabilities shown below:

$$X_{\text{tip},1} = [0.0, 0.0, 2.8]^T$$
$$F_1 = 2.25\,\mathbf{i} + 2.25\,\mathbf{j}$$
$$X_{\text{tip},2} = [3.0, 3.0, 2.1]^T$$
$$F_2 = 2.25\,\mathbf{j} - 2.25\,\mathbf{k}$$

Note that the second subscript of a variable is referred to a task. The cost function is defined as the squares of the Euclidean distance between the two locations:

$$C = (X_{\text{base},2} - X_{\text{base},1})^2 + (Y_{\text{base},2} - Y_{\text{base},1})^2$$

The lengths of the manipulator links are

$$l_1 = 1.0 \quad l_2 = 1.5 \quad l_3 = 0.75$$

The torques for all three joint actuators are bounded above by five. That is, $\bar{\tau} = [5, 5, 5]^T$.

Using a genetic string of 80 bits to encode the four parameters of the problem, two parameters (θ_1 and θ_2) for each task, each parameter is thus represented by 20 bits. Note also that having encoded θ_1 and θ_2 for each task, θ_3 can be obtained by the constraints defined by eqs. (18)–(20).

Each graph in Figure 2 shows the fitness value of the best string in each generation versus the generation, for a type of fitness function. In Figure 2a the simulation tends to wander because there is not much difference on the fitness values between "good" strings and "bad" ones. In Figure 2b the fitness function is modified by a tangent function from the initial generation until the end, to accentuate the differences among population members. Note that in this case a superstring takes over the population in the early stage. In Figure 2c the fitness function is modified by a tangent function from the 200th generation after the genetic approach has ex-

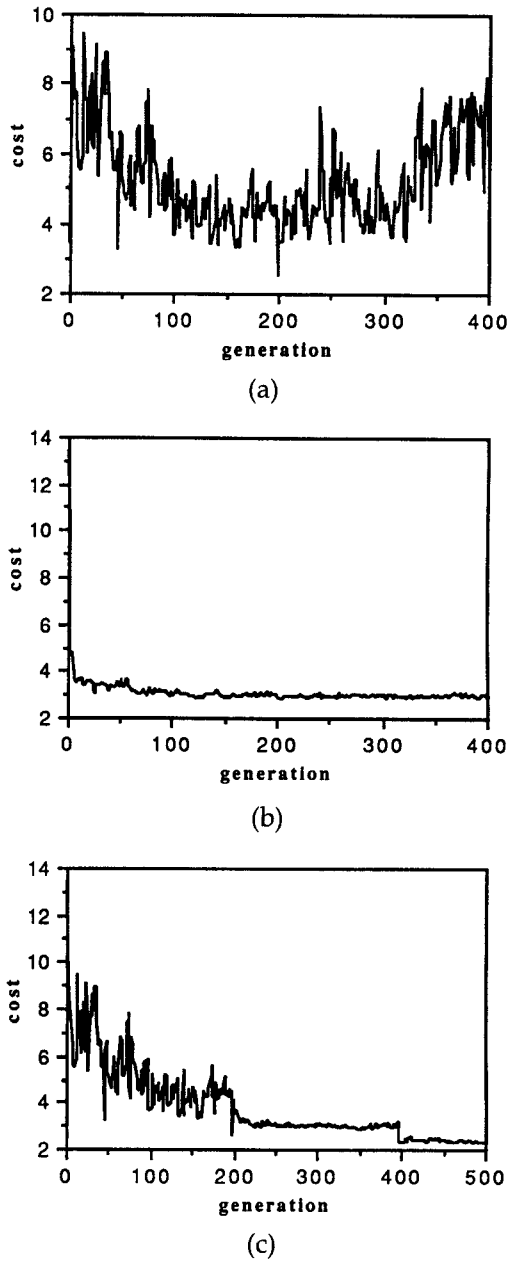


Figure 2. The effect of fitness function. The cost versus generation plots using (a) the fitness function defined by eq. (16) throughout the simulation; (b) the modified fitness function defined by eq. (17) throughout the simulation; and (c) the fitness function defined by eq. (16) till the 200th generation and then the modified fitness function defined by eq. (17) for the rest of the simulation.

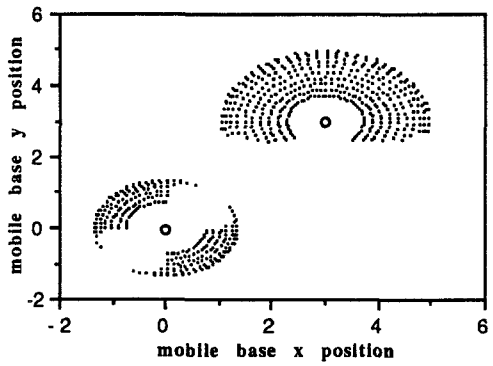


Figure 3. The feasible regions for the two task problem.

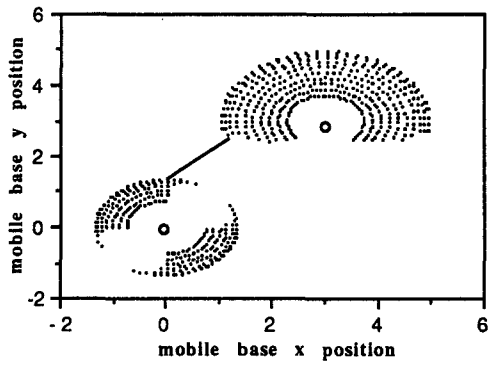
plored a considerable number of strings, and a better result is achieved. Throughout the simulation we feel strongly that the performance of the genetic algorithm greatly depends on the proper formulation of the fitness function. Because the cost function of the mobile manipulator path planning is well defined by eq. (11), the fitness function used for the genetic algorithm must be closely related to the cost function. Only slight modification is made to the cost function, as demonstrated in Figure 2, to improve the convergence and the solution. As explained above, as the solution is taking shape, a tangent function is introduced at the 200th generation (in the middle) to accentuate the differences among population members to further explore the solution space. However, many other functions can be applied equally.

The feasible regions in which the mobile base can be located to perform the tasks mentioned in this problem are determined exhaustively, and are indicated by dotted areas as shown in Figure 3. The locations where the tasks are to be performed, i.e., the locations of the end-effector, are indicated by small circles. In general, the base can be located on an annulus region to perform the task without the torque constraints. A part of this annulus region becomes invalid when the torque constraints are applied, as shown in the figure.

For example, for the first task, two regions are missing along the diagonal line. This is because most of the desired force vector must be generated by the shoulder joint of the manipulator, thus exceeding the shoulder torque constraint.

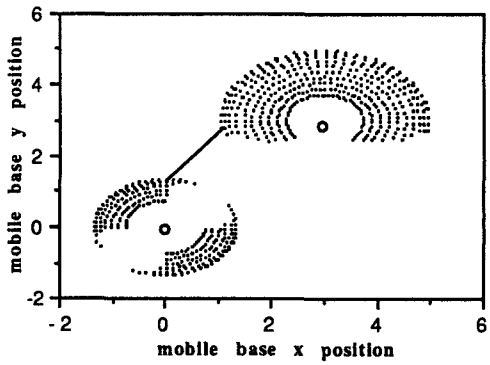
For the second task, the force specification is in the positive y direction and negative z direction. If the base is in an unfeasible area of the annulus region, most of the desired force must be generated by the shoulder joint, thus exceeding the shoulder torque constraint. If the base is in the feasible area of the annulus region, the desired force can be generated by both the joints of the shoulder and the elbow.

The effectiveness and efficiency of the genetic algorithms are shown in Figures 4a–4c. Figure 4a shows that satisfactory results are reached after 50 generations. The genetic algorithm can easily reach



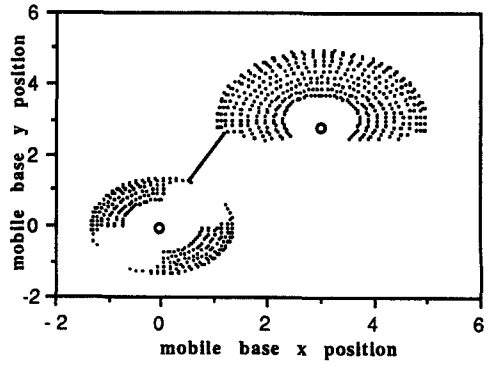
$x_1 = -0.006$, $y_1 = 1.339$, $x_2 = 1.147$, $y_2 = 2.655$; cost = 3.06

(a)



$x_1 = -0.015$, $y_1 = 1.350$, $x_2 = 1.079$, $y_2 = 2.653$; cost = 2.88

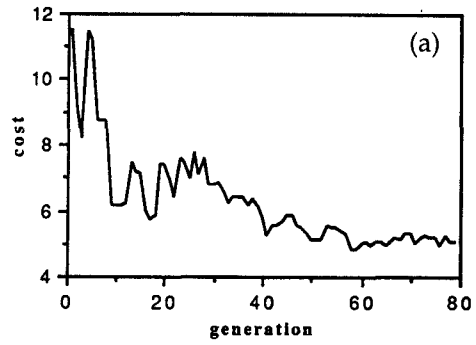
(b)

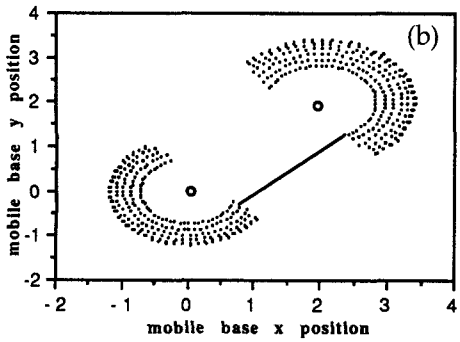


$x_1 = 0.497$, $y_1 = 1.230$, $x_2 = 1.079$, $y_2 = 2.691$; cost = 2.28

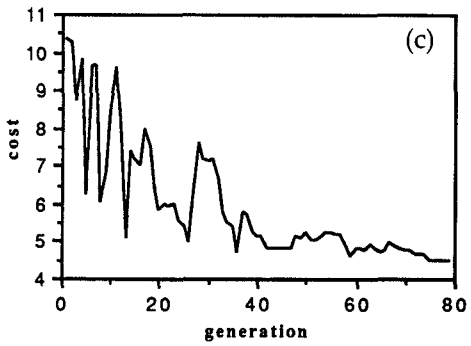
(c)

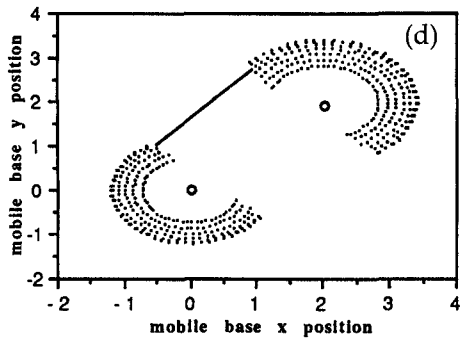
Figure 4. The solutions obtained by the genetic algorithm for the two task problem when the (a) 50th, (b) 300th, and (c) 400th generation is reached, respectively.



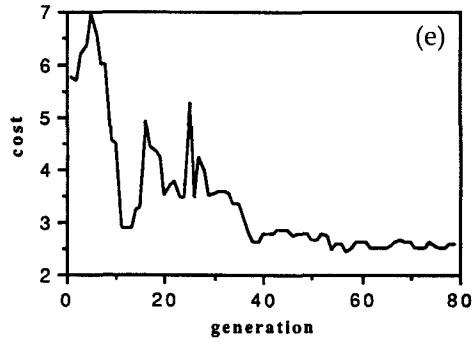


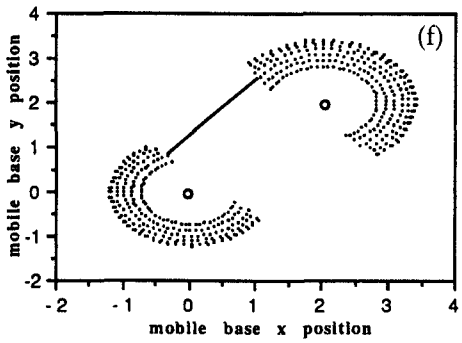
$x_1 = 0.835$, $y_1 = -0.398$, $x_2 = 2.248$, $y_2 = 1.284$; cost = 4.83



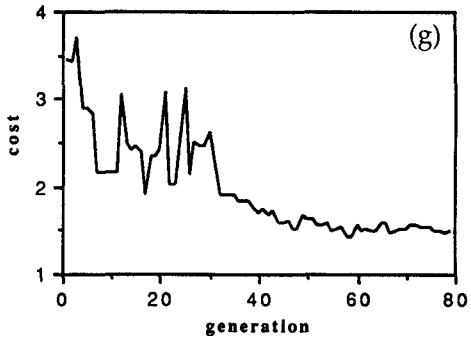


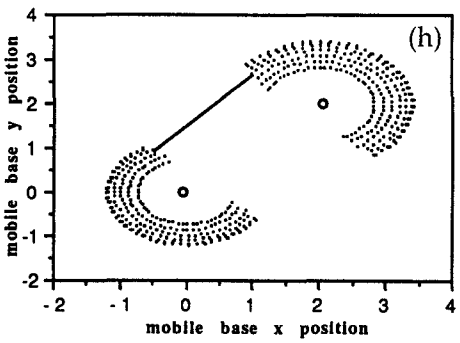
$x_1 = -0.547$, $y_1 = 0.955$, $x_2 = 1.083$, $y_2 = 2.476$; cost = 4.48





$x_1 = -0.408$, $y_1 = 0.894$, $x_2 = 1.063$, $y_2 = 2.509$; cost = 2.39





$x_1 = -0.433$, $y_1 = 0.903$, $x_2 = 1.180$, $y_2 = 2.352$; cost = 1.41

Figure 5. The cost versus generation plots corresponding to $\alpha = 1, 0.9, 0.5$ and 0.3 are shown in (a), (c), (e) and (g), respectively. Correspondingly the solutions obtained by the genetic algorithm are shown in (b), (d), (f) and (h), respectively. The dotted areas indicate the feasible regions.

the near optimal in this nonlinear, nonconvex constraint problem with unconnected feasible regions. Then there is no significant change in cost until the 400th generation is reached as shown in Figure 4c, in which case a much lower cost is yielded.

$$X_{\text{tip},1} = [0.0, 0.0, 2.9]^T$$
$$F_1 = 2\,\mathbf{i} + 2\,\mathbf{j} + 3\,\mathbf{k}$$
$$X_{\text{tip},2} = [2.0, 2.0, 2.75]^T$$
$$F_2 = 2\,\mathbf{i} + 2\,\mathbf{j} - 3\,\mathbf{k}$$

The length of manipulator links and the torque constraints of manipulator joints are the same as in the first problem. Similar to the previous simulation, each parameter (θ_i) is encoded by 20 bits. The cost function is defined to reflect the trade-off between base motion and manipulator motion:

$$\begin{aligned} C = \alpha\{&(X_{\text{base},2} - X_{\text{base},1})^2 \\ &+ (Y_{\text{base},2} - Y_{\text{base},1})^2\} \\ &+\beta\{(l_2 + l_3)^2 \times (\theta_{1,2} - \theta_{1,1})^2 \\ &+ l_2^2 \times (\theta_{2,2} - \theta_{2,1})^2 \\ &+ l_3^2 \times (\theta_{3,2} - \theta_{3,1})^2\} \end{aligned} \tag{22}$$

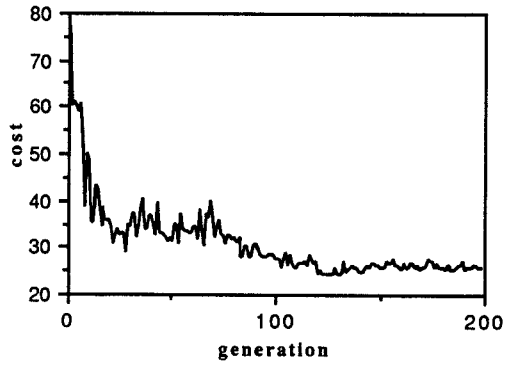
where α and β are weight factors for the cost of moving the base and the cost of changing the configuration of the manipulator, respectively. Here β is chosen as $1 - \alpha$.

When α equals 1, C is the squared Euclidean distance between the two locations. The fitness value of the best string in each generation versus the generation plots with different α values are shown in Figures 5a, 5c, 5e, and 5g, respectively. Correspondingly the solutions obtained by the genetic algorithm with different α values are shown in Figures 5b, 5d, 5f, and 5h, respectively. The feasible regions are indicated by dotted areas.

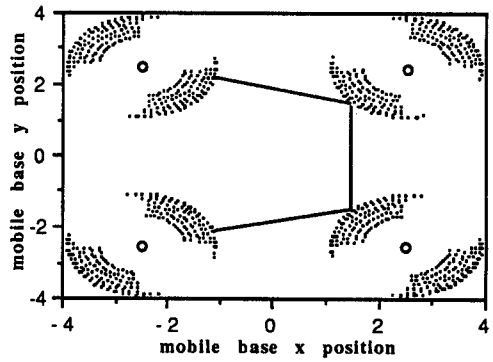
3. Consider a four-task problem defined as follows:

$$X_{\text{tip},1} = [-2.5, 2.5, 2.75]^T$$
$$F_1 = 2.5\,\mathbf{i} + 2.5\,\mathbf{j}$$
$$X_{\text{tip},2} = [2.5, 2.5, 2.75]^T$$
$$F_2 = 2.5\,\mathbf{i} - 2.5\,\mathbf{j}$$
$$X_{\text{tip},3} = [2.5, -2.5, 2.75]^T$$
$$F_3 = 2.5\,\mathbf{i} + 2.5\,\mathbf{j}$$
$$X_{\text{tip},4} = [-2.5, -2.5, 2.75]^T$$
$$F_4 = 2.5\,\mathbf{i} + 2.5\,\mathbf{j}$$

The length of the manipulator links and the torque constraints for the manipulator joints are the same as the former two problems. The cost is defined as the sum of the squares of the Euclidean distances that the base traverses as the mobile manipulator performs the sequence of the tasks. Here, there are altogether eight parameters, two (θ_1, θ_2) for each task, each parameter thus encoded by 10 bits with a genetic string of 80 strings. If the mobile manipulator is only required to perform the four tasks without repetition, the cost versus generation plot and the solution are shown in Figures 6a and 6b, respectively.



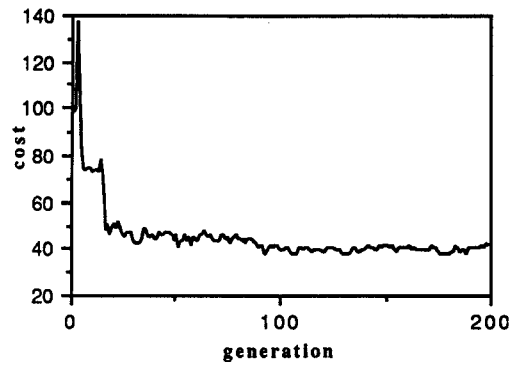
(a)



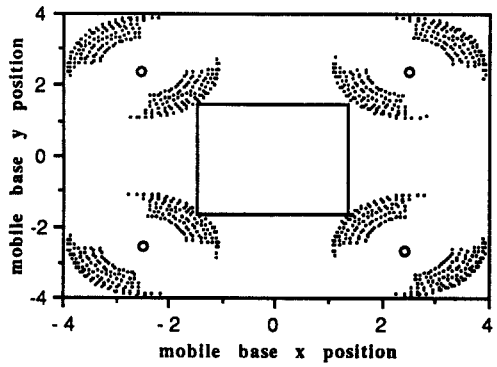
$x_1 = -1.15$, $y_1 = 2.07$, $x_2 = 1.50$, $y_2 = 1.50$, $x_3 = 1.51$, $y_3 = -1.50$, $x_4 = -1.11$, $y_4 = -2.27$; cost = 23.95

(b)

Figure 6. Results for the four task problem without repetition: (a) cost versus generation plot; (b) the solution. The dotted areas indicate the feasible regions.



(a)



$x_1 = -1.49$, $y_1 = 1.52$, $x_2 = 1.36$, $y_2 = 1.67$, $x_3 = 1.47$,

$y_3 = -1.53$, $x_4 = -1.52$, $y_4 = -1.61$; cost = 37.15

(b)

Figure 7. Results for the four task problem with repetition: (a) cost vesus generation plot, (b) the solution. The dotted areas indicate the feasible regions.

4. The mobile manipulator is required to perform the four tasks repeatedly; the cost versus generation plot and the solution obtained by the genetic algorithm are shown in Figures 7a and 7b, respectively. Again, the dotted areas indicate the feasible regions.

5. COMPARISON

Table I compares the performance of exhaustive search and the proposed genetic algorithm. The performance measures compared are the CPU time used on a SUN 4–C75 and the cost of the optimal solution found. The four simulations described in section 4 were tested with exhaustive search using, respectively, 8, 16, and 24 bit encoding of the joint angles. For all four cases, the solutions found by the exhaustive search method with 8 bit encoding are inadequate as their cost are much higher than the solutions found by the genetic algorithm. Although the cost of the solutions found by the exhaustive search algorithm with 24 bit encoding are comparable to those found by the genetic algorithm, but the CPU time needed by the exhaustive search algorithm is roughly two orders of magnitude greater. One of the reasons that the genetic algorithm found better solutions is that it uses a total of 80 bits to encode the parameters. However, the run-time for the exhaustive search is exponential in terms of the number of bits used to encode the parameters. The estimated CPU time required by an exhaustive search using 80 bits to encode the parameters is 10^{17} times more than the 24 bit case. Therefore, the genetic algorithm is an effective and efficient way to obtain a near optimal solution to the mobile manipulator path-planning problem.

Table 1. Comparisons of the CPU time and cost of optimal solution for exhaustive search and the proposed genetic algorithm.

	Exhaustive Search						*Genetic Algorithm*	
	String = 8 Bits		*String = 16 Bits*		*String = 24 Bits*			
Simulations	*CPU Time**	*Optimal Cost*	*CPU Time**	*Optimal Cost*	*CPU Time**	*Optimal Cost*	*CPU Time*†*	*Cost of Best Solution*
1	0.1	3.8	17.6	2.96	4580	2.31	≤10.9	2.28
2	0.12	31.82	20.8	12.89	4490	12.71	≤2.14	2.39
3	0.07	34.43	19.8	34.43	4931	24.37	≤5.89	23.96
4	0.07	59.43	19.6	59.43	4780	36.0	≤5.7	37.15

* CPU time in seconds on SUN 4/C75.

† The CPU time shown are for 400 generations: Simulation 1—solution found within 400 generations (α = 0.5); Simulation 2—solution found within 80 generations (α = 0.5); Simulation 3—solution found within 200 generations (α = 1); Simulation 4—solution found within 200 generations (α = 1).

6. CONCLUSION

In this article, we have developed a genetic algorithm to search for an optimal base trajectory for a mobile manipulator to perform a sequence of tasks. The mobile manipulator path-planning problem is a constrained optimization problem. The feasible regions are nonconvex and unconnected. The problem is multi-modal, and traditional methods such as gradient descent methods, which are highly sensitive to initial guesses, are likely to be trapped in local minima. We have demonstrated the effectiveness and efficiency of our algorithm to reach a near optimal solution to the mobile manipulator path planning problem through various simulations.

REFERENCES

1. W. Carriker, P. Khosla, and B. Krogh, "An approach for coordinating mobility and manipulation," in *IEEE International Conference on System Engineering*, Dayton, OH, 1989, pp. 59–63.
2. W. Carriker, P. Khosla, and B. Krogh, "The use of simulated annealing to solve the mobile manipulator path planning problem," in *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, pp. 204–209.
3. R. Duley and J. Luh, "Redundant robot control for higher flexibility," in *Proceedings 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, pp. 1066–1071.
4. T. Yoshikawa, "Analysis and control of robot manipulators with redundancy," in *Robotics Research: The First Int. Symp.*, MIT Press, Cambridge, MA, 1984, pp. 753–748.
5. S. Lee, "Dual redundant arm configuration with task-oriented dual arm manipulability," *IEEE Transactions on Robotics and Automation*, 5(1), 78–97, 1989.
6. S. Chiu, "Control of redundant manipulators for task compatibility," in *Proceedings 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, pp. 1718–1724.
7. S. Chiu, "Task compatibility of manipulator postures," *The International Journal of Robotics Research*, 7(5), 13–21, 1988.
8. L. Kelmar and P. Khosia, "Automatic generation of kinematics for a reconfigurable modular manipulator system," in *Proceedings 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1988, pp. 663–668.
9. S. Kirkpatrick, C. Gelett, and M. Vecchi, "Optimization by simulated annealing," *Science*, May, 671–680, 1983.
10. P. J. M. Van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Boston, 1987.
11. D. McCloy and D. M. J. Harris, *Robotics: An Introduction*, Halsted Press, New York, 1986.
12. J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.
13. L. Davis, *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.
14. N. Ansari, M. Chen, and E. S. Hou, "Point pattern matching by a genetic algorithm," in *Proc. IECON 90, 16th Conf. of IEEE Industrial Electronics Society*, Pacific Grove, CA, 1990, pp. 1233–1238.
15. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learing*, Addison-Wesley, Reading, MA, 1989.
16. K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph. D. dissertation, University of Michigan, Dissertation Abstracts International 36(10), 5140B. (University Microfilms No. 76-9381), 1975.