Robotica (1998) volume 16, pp. 575–588. Printed in the United Kingdom © 1998 Cambridge University Press

Path planning of a mobile robot using genetic heuristics

Andreas C. Nearchou

Mechanical Engineering Department, University of Patras, 26 110 Patras, Greece
Tel.: + + 3 -61 997.590. Fax: + + 3 -61 997.700. Email: nearchou@upatras.gr

(Received in Final Form: December 11, 1997)

SUMMARY

A genetic algorithm for the path planning problem of a mobile robot which is moving and picking up loads on its way is presented. Assuming a findpath problem in a graph, the proposed algorithm determines a near-optimal path solution using a bit-string encoding of selected graph vertices. Several simulation results of specific task-oriented variants of the basic path planning problem using the proposed genetic algorithm are provided. The results obtained are compared with ones yielded by hill-climbing and simulated annealing techniques, showing a higher or at least equally well performance for the genetic algorithm.

KEYWORDS: Path planning; Mobile robot; Genetic heuristics; Load picking.

1. INTRODUCTION

Path planning is widely recognized as a fundamental problem and one of the most complex in the domain of Robotics.[1] Given a description of the robot's environment, a start position and a desired destination, the problem is to find a continuous, collision-free motion for the robot from the start to the goal position. Even in its simplest form, the problem is characterized as NP-complete and PSPACE-hard. Its complexity increases exponentially with the dimension of the robot's configuration space.[2]

A fundamental tool for formulating and solving path planning problem is the configuration space.[3] The central idea behind this approach is the representation of the position and orientation of the robot as a single point in its configuration space (C-space) by building geometric objects, called configuration space obstacles (C-obstacles).

Many approaches have been proposed by the research community to solve the path planning problem using C-space. The most extensively studied so far is to reduce the problem to a shortest-path problem in a graph: (i) the visibility graph approach[4,5] is based on the construction of an undirected graph whose vertices are the initial and the goal configuration of the robot and the vertices of the C-obstacles. The problem of path planning is thereby converted into a search of the graph for a path (usually the shortest) between the initial and the final configuration. (ii) Retraction method[6,7] uses a Voronoi diagram to solve the path planning problem. The edges of the Voronoi diagram represent paths that are equidistant from the closest pair of obstacles, and its vertices are points where three or more such paths meet. A solution to the problem is found by searching this graph for shortest path. (iii) The decomposition of the robot's free-space into simple regions, called cells, and the construction of a non-directed graph, called connectivity graph, is one of the oldest path planning approaches.[8] This graph represents the adjacency relation between the cells. Its vertices are the generated cells and two cells are connected with a link if they are adjacent.

After the construction of the suitable graph, the path planning process is completed by searching this graph for a sequence of vertices which connect the start and goal position. Usually, the Dijkstra's graph search algorithm,[9] or the A* algorithm[10] are used. Since the computational time of the searching process increases exponentially with the number of the vertices in the graph, the implementation of these algorithms for problems with large number of vertices in a conventional serial computer may be not effective. Furthermore, in real-world applications, robot path planning must additionally satisfy special, task-oriented criteria, such as minimizing the time of travel, maximizing load carrying operations, etc.

Recent advances in robotics and machine intelligence have led to the application of modern heuristics, such as the genetic algorithms (GAs), to solve the path planning problem. Davidor[11] proposed a special GA for optimized robot trajectories. The main characteristics of his algorithm is the use of dynamic chromosomes' structures and a modified crossover operator called analogous crossover. Each chromosome in the population represents a different robot trajectory, i.e. a sequence of successive arm-configurations. The goal of the proposed GA is to minimize the accumulative deviation between the actual and the desired path. Hoi-Shan Lin et al.[12] presented an evolutionary algorithm for the path planning problem of mobile robots in environments which may contain unknown obstacles. Their algorithm first determines an optimal global path from the start to the destination, and then performs a process of repairing this path, by handling possible collisions with known or unknown obstacles. In a recent work,[13] the author of the current paper used a simple GA to solve the path planning problem of redundant articulated robots. The objective of the proposed GA was the minimization of the end-effector's positional error from the desired destination, while keeping the links of the robot in a safe distance from the obstacles. Assuming a predefined map with knot points, Shibata and Fukuda[14] proposed a GA with fuzzy critic to determine an optimal path for a mobile robot which is moving and picking up loads on its way.

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

This paper presents a different genetic-based approach for solving the path planning problem of a robot which during its motion may be ordered to pick and carry loads to the destination. Assuming a known map of the robot's environment consisting of knot points, which are given in a graph-vertices format, the proposed GA searches this map for an optimal path between the start and the destination points satisfying the additional task requirements.

The main characteristics of the proposed GA consist of the following: (1) bit-string encoding of selected graph-vertices is used. (2) The elimination of the non-feasible solutions is performed without using penalty terms in the evaluation function. (3) Sigma-truncation scaling of the fitness function is used. (4) The individual's chromosomes are selected for reproduction according to the Tournament strategy. (5) New populations of chromosomes are generated using three genetic operators: a uniform crossover, a bit-mutation and a bit-inversion.

The performance of the proposed GA is tested on a large number of experiments, and compared to that of three other heuristics: an iterated hill-climbing, an iterated stochastic hill-climbing, and the simulated annealing.

The remainder of the paper is organized as follows: Section 2 defines and formulates the problem. Section 3 presents and analyzes the proposed GA solution. Section 4 demonstrates and discusses the efficiency of the proposed solution through simulation experiments. In the same section a comparison between the GA's performance and that of the three heuristic techniques is provided. Finally, the contribution of the paper is summarized in section 5.

2. FORMULATION OF THE PATH PLANNING PROBLEM

The problem's terminology used in this paper is as in reference 8. Let a robot, called A, be a single point moving in its two dimensional C-space among polygonal C-obstacles. Also assume that in the preprocessing phase, the map, called G, representing the possible free paths for A, has been constructed and given as a network of one-dimensional curves. Path planning consists of connecting the start and goal configurations to points in G, and searching G for the shortest path between these points.

Several types of G exist in the literature, such as the visibility graph, the connectivity graph, the Voronoi diagram, etc. Without loss of generality, this work assumes that $G=(V, E)$ is a visibility graph,[4] i.e. a non-directed graph whose vertices V are the start and goal robot's configurations, and all the C-obstacle vertices. The links E of G are straight line segments connecting pair of vertices that do not cut the interior of a C-obstacle region.

Further, it is assumed that A not only moves from start to goal, but also performs a pick-and-carry-loads operation in its way. The loads given as pure positive quantities are located on the knot points, i.e. on the vertices of G.

This paper deals with the following types of robot's tasks:

(a) Move as fast as possible without carrying any loads on the way.
(b) Move by picking-and-carrying on the way as many loads as possible without considering the time of travel.
(c) Move fast picking-and-carrying on the way as many loads as possible.
(d) Move fast picking-and-carrying on the way as many loads as possible. Do not carry more loads than a permitted upper limit.

According to the above assumptions, the problem can be formally stated as follows:

The path planning problem in a graph (PPPG): Given a connected, unidirected graph $G=(V, E)$, the start and goal vertices Vs, Vg, respectively, in $G(Vs, Vg \in V)$. A positive edge-cost function c: $E \in \Re^+$. A positive vertex-profit function l: $V \in \Re^+$. Compute a sub-graph $G'=(V', E')$ of G, where $V' \subseteq V$, $E' \subseteq E$, and Vs, $Vg \in V'$, such that:

- Task-1: $c(G')$ is minimal,
- Task-2: $l(G')$ is maximal,
- Task-3: $c(G')$ is minimal and $l(G')$ is maximal,
- Task-4: $c(G')$ is minimal and $l(G')$ is maximal but less than an upper permitted limit (Lmax) for the loads.

The cost c_j of an edge E_j in G' corresponds to the Euclidean distance between the vertices (V_{j-1}, V_j) joined by E_j. Therefore, the total path cost is equal to

$$Path_{Cost} = \sum_j c(E_j). \quad (1)$$

The profit l_j corresponds to the load founded in V_j, and thus, the total load carried is equal to

$$Load_{Carried} = \sum_j l(V_j). \quad (2)$$

3. THE PROPOSED GENETIC ALGORITHM SOLUTION

The purpose of this paper is not to outperform or to find better solutions than other existing heuristic approaches to the PPPG. Its purpose is to show how genetic algorithms may be effectively applied to this problem and to describe how to do so. This section starts with a brief overview of GAs, and then presents and analyzes the proposed GA solution to the PPPG. Finally, it briefly describes three other heuristic techniques tested in this work.

3.1 Genetic algorithms: a brief overview

Genetic algorithms (GAs) are formed of a directed random search, which emulates the process of genetic evolution found in nature to perform artificial evolution. They were developed by John Holland[15] in the early 1970s and since then have been successfully applied to numerous large and complex search space problems.[16,17]

In nature, organisms have certain characteristics that affect their ability to survive and reproduce. These characteristics are contained in their genes. Natural selection ensures that genes from a strong idividual are presented in

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

greater numbers in the next generation than those from a weak individual. Over a number of generations, the fittest individuals (in the environment in which they live) have the highest probability of survival and tend to increase in numbers, while the less fit individuals tend to die out. This is the Darwin's principle of survival-of-the-fittest and constitutes the basic idea behind GAs.

Holland[15] showed that potential solutions to a search problem can be encoded using simple representations (bit strings), mirroring the role played by the genes. A number of such strings or individuals form the population for the GA. The quality of an individual is defined through a measure called fitness. As the GA evolves, a set of basic genetic operators are iteratively applied on the entire population to produce new populations with better individuals. After a number of generations, this process should lead to a population of highly fit individuals, i.e to structures which represent good solutions to the physical search problem to be solved. The fittest individual of the "final" generation should be an optimal, or a near optimal solution to the physical problem.

Traditionally, a GA starts evolving by generating (usually in a random way) an initial population of chromosomes. Then, the value of a function called fitness function is evaluated for each chromosome of the population. After this, a set of genetic operators (selection, crossover and mutation) are used in succession to create a new population of chromosomes for the next generation. The process of evaluation and creation of new successive generations is repeated until the satisfaction of a convenient termination condition. The basic structure of any GA is given below in a Pascal-like format:

```
procedure GA
begin
   gen←0;
   initialize Pop(gen);
   evaluate Pop(gen);
   while not termination-condition do
      gen←gen + 1;
      select Pop(gen) from Pop(gen − 1);
      crossover Pop(gen);
      mutate Pop(gen);
      evaluate Pop(gen);
   endwhile
end
```

The major components of any GA which can significantly affect its performance are the following:[17]

3.1.1 **The representation mechanism:** a means of encoding solutions to the physical search problem as artificial chromosomes. This mechanism consists of choosing a way to represent a solution to the real problem as a finite-length string over a specific alphabet. This string is called a chromosome. Binary coding, i.e. strings of bits (0, 1), is the most commonly used representation in today's GAs. However, other codings have proved to be more effective for specific problems and under certain circumstances.

3.1.2 **The initial population:** a way of creating an initial population of such chromosomes. Choosing the initial population of chromosomes in a GA is usually done randomly, with the goal of selecting solutions from all over the search space. The speed with which a GA moves from a random to a well-adapted population is a good measure of performance. Another way of choosing the initial population, is the direct initialization, when some aspects of a good solution are known in advance. Although this method may drastically reduce the computation time required by the GA, it can lead to problems of premature convergence, a case where the GA falls into a local optimum.

3.1.3 **The evaluation mechanism:** a means of evaluating the "fitness" of each chromosome. This mechanism consists of evaluating a function called fitness function, the value of which denotes how good solution to the real problem a specific chromosome represents. This evaluation is very significant since the fitness' value defines the ability of the corresponding chromosome to survive and reproduce in the next generation. Furthermore, the fitness function plays the role of the environment in which during the evolution of the GA the chromosomes must be adapted. In an optimization problem, the fitness function corresponds to the objective function which must be optimized.

3.1.4 **The operators:** a set of genetic operators applied on the population. There are three basic forms of genetic operators in most GAs: selection, crossover and mutation. Each one of these operators emulates a corresponding process found in biological evolution. Selection is the operator which emulates the process of natural selection or the "survival-of-the-fittest". Individual's chromosomes are copied from one generation to the next according to their relative fitness value in respect to the total population's fitness. Therefore, chromosomes with higher fitness value have greater chance of contributing one or more offspring to the subsequent generations. The traditional method implementing selection is the proportionate selection or roulette wheel.[16] Crossover follows selection and is a recombination operator that works on a pair of old chromosomes randomly selected according to a specified probability (typical in the range between 0.6 and 1.0). The traditional view, is that crossover is the most important part of a GA for rapidly exploring a search space. Due to this fact, several forms of crossover have been proposed in the literature, e.g. single-point, two-point, cyclic, uniform crossover, etc. Mutation is applied to each child individual chromosome after crossover. It randomly alters the value of each gene of a chromosome with a small probability (typical equal to 0.001). For chromosomes encoded as binary strings, mutation results to a change of the digital value "1" to a "0" value and vice versa.

3.1.5 **Control parameters:** appropriate settings on a set of control parameters. A number of parameters which control how the various elements (described above) of a GA combine and operate must also be defined. This is very important because each different combination of the same elements has its own characteristics in the GA performance. According to Grefenstette's experimental work,[18] the most important control parameters are: population size, crossover

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

and mutation rates, generation gap, selection strategy, fitness scaling.

3.2 The selected chromosome and its interpretation

The key point in designing a GA to find solutions to a given problem is the suitable syntax of the chromosomes in the population together with their interpretation. For the PPPG, the chromosome selected for use is a string of bits of a fixed length representing a set of selected graph vertices. Each bit in the string corresponds to a specific vertex in the graph. The left most bit in the string corresponds to the start vertex, while the right most bit corresponds to the destination vertex in the graph. If a specific bit is set, the associated graph vertex is selected. The set of the selected vertices specifies a path for the robot between the start and the destination vertices under the assumption that each vertex can be passed only once or not at all in the path. We are interested only for feasible baths, i.e. for paths that do not cross or cut any C-obstacles region.

Specifically, for a graph with N vertices the start vertex has the label 1 and the destination vertex the number N. The other vertices are numbered in the range [2 . . . N − 1].

Initially, the chromosomes are generated as follows:

(i) Randomly choose an integer k in the range [2 . . . N − 1].
(ii) Randomly select k integers in the range [2 . . . N − 1]. These integers correspond to string positions, i.e. to vertices' labels. Duplications are not allowed.
(iii) For each one of the k positions generate a bit value by flipping a fair coin.
(iv) If a specific bit is set then the corresponding vertex is selected for use in the path.

For example, the 7-bit string **1 0 0 1 1 0 1** represents a unique path in a 7-vertices graph passing through the vertices 1-4-5-7, where 1 denotes the start vertex and 7 the destination vertex in the graph. It must also be underlined that a chromosome's structure may lead to a not feasible path i.e. to a path that cuts the interior of a C-obstacle region.

3.3 Fitness evaluation

By far the selection of the convenient fitness function's form is the next important aspect in designing a GA for a specific problem. Based on the problem's formulation described in section 2, each one of the desired robot's tasks correspond to a different constrained optimization problem. The fitness function (also known as the scoring function of a chromosome's solution), corresponds to the objective function of the constrained optimization problem we want to optimize.

GAs are essentially unconstrained search procedures within the given representation space. The traditional GA formulation for constrained optimization problems is through the use of penalty functions.[16] However, as Davis observed in reference 17, though the evaluation function may be well defined, there is no accepted methodology for combining it with the penalty. To overcome this problem, a different function formulation is used in this work.

For the PPPG, each chromosome represents a unique path in the graph. Taking into account equation (1), the total path cost of a chromosome's structure is evaluated using the following relation:

$$\mathrm{Path}_{\mathrm{Cost}} = \begin{cases} \sum_j c(E_j), & \text{If feasible path} \\ \text{A real maximum}, & \text{Otherwise} \end{cases} \tag{3}$$

For the pick-and-carry loads operation the following relation is used:

$$\mathrm{Load}_{\mathrm{Carried}} = \begin{cases} \sum_j l(V_j), & \text{If (feasible path)} \wedge \left(\sum_j l(V_j) < \mathrm{Lmax} \right) \\ 0, & \text{Otherwise} \end{cases} \tag{4}$$

Therefore, for the robot's task-1, the fitness function is defined as:

$$fitness = \frac{1}{Path_{Cost}} \tag{5}$$

For robot's task-2, fitness function is defined as:

$$fitness = \mathrm{Load}_{\mathrm{Carried}} \tag{6}$$

For robot's task-3 and task-4, fitness is computed by:

$$fitness = \frac{\mathrm{Load}_{\mathrm{Carried}}}{Path_{Cost}} \tag{7}$$

For each one of the desired robot's tasks, the objective of the proposed GA is to maximize the associated fitness function. This results to a minimum value for the path cost (robot's tasks: 1, 3, and 4), and to a maximum value for the total load carried by the robot (robot's tasks: 2, 3, and 4). The above formulations of the fitness function progressively eliminate the appearance of non-feasible solutions in the population by reinforcing the survival and reproduction of the feasible solutions. A solution is feasible if it corresponds to a valid path, and if the total weight of the loads on the vertices of this path do not exceed the upper permitted limit Lmax (case of task-4).

3.4 Genetic operators and control parameters

Four genetic operators were used in the proposed GA: selection, crossover, mutation, and inversion.

Selection: The individual chromosomes are selected

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

based on the binary Tournament selection strategy.[19] According to this strategy, two chromosomes are picked at random from the population and that with the higher fitness value is copied into a mating pool (i.e. it survives and reproduces its structure into the new population). This process is repeated until the mating pool is full.

Crossover: A uniform crossover operator[20] was used. This operator works as follows: two parent chromosomes are selected based on the crossover probability. For the pair of the selected parents a template or mask chromosome is randomly generated. The bit-value at each position of the template specifies the bit-value of the corresponding position of the child chromosome. Specifically, where there is "1" in the template, the corresponding gene from the first parent passes its value to the child, otherwise the second parent passes its bit-value to the child. The process is repeated with the parents exchanged to produce the second child. Therefore, offspring contain a mixture of genes from each parent.

Mutation: This operator is the traditional bit-mutation operator described in sub-section 3.1. It randomly alters each gene with a small probability.

Inversion: Inversion is a reordering operator applied on the bits of a single chromosome. It works by reversing the order of genes between two randomly chosen positions within the selected chromosome. Exhaustive experiments for the PPPG show that the use of this operator significantly increases the performance of the proposed GA.

The last critical aspect in designing a GA is the selection of the suitable settings for the GA's control parameters. Unfortunately, there is no formal way to define the appropriate parameters' settings. Traditionally, this is achieved experimentally. The final settings used to test the proposed GA for the PPPG are (experimentally determined) as follows: population size = 100, total number of generation = 50, crossover rate = 0.6, mutation rate = 0.0333, inversion rate = 0.1, sigma scaling factor c = 1.0, generation gap = 1.0. A description of each one of the control parameters is given below:

Population size: Determines how many chromosomes, i.e. how much genetic material, are available during the genetic search. A too small population size decreases the ability of the GA to adequately cover the search space. A too large population size significantly increases the time needed by the GA to evaluate the chromosomes and thus results in an ineffective search.

Crossover rate: Specifies the frequency with which the crossover operator is applied to the individual's chromosomes in a new generation. A too low crossover rate causes the introduction of fewer new individual's into the population and may lead to search stagnation since the process of reproduction tends to dominate. A too high rate leads to a very fast exploration of the search space but the GA's performance may be degraded as strong individuals are discarded very fast before reproducing their structure.

Mutation rate: Specifies the probability that a gene's value of a newly created chromosome will be changed. Mutation governs the introduction of new unexplored areas in the search. A high mutation rate increases the diversity in the population but introduces excessive randomness in the search. Conversely, a too low mutation rate reduces the diversity and leads to sub-optimal solution.

Generation gap: This parameter specifies the proportion of the individuals in the population which are replaced by the offspring in each generation. Usually, a generation gap of one is use, i.e. the whole population is replaced in each generation.

Scaling: This is a method to avoid the problem of premature convergence in a GA. Scaling is used to maintain good levels of competition throughout the search process. In the absence of scaling, few, relatively "good" chromosomes but not optimal (called super-individuals) could dominate the population very early in the process, leading the GA to premature convergence. In this paper a sigma-truncation technique[16] was used:

$$\text{fitness}_{\text{scaled}} = \text{fitness}_{\text{raw}} - (\text{fitness}_{\text{average}} - \text{c}.\sigma) \qquad (8)$$

where c a small integer chosen in the range [1 . . . 5] and σ is the population's standard deviation. Possible negative results, i.e. ($\text{fitness}_{\text{scaled}} < 0$) are set to zero.

3.5 Heuristic algorithms comparison

The performance of the proposed GA was compared to that of three well-known heuristics: an iterated hill-climbing (HC), an iterated stochastic hill-climbing (SHC), and simulated annealing (SA). The methodology used to implement these algorithms is as in references 21. The basic characteristic of this implementation is the use of bit-vector representation of the problem's space solution which is then mapped to a unique real solution of the physical problem. Below, is given the pseudo-code for each one of the algorithms ina Pascal-like format. For a detail discussion on various versions of these algorithms, the interested reader is referred to reference 21.

```
procedure iterated HC
begin
  I←1;
  while (I≤MaxIterations) do
    select randomly a current string Xc;
    evaluate Xc;
    b←1;
    while (b≤Nbits) do
      generate a new string Xn by flipping
            the b-th bit of string Xc;
      evaluate Xn;
      if f(Xc) < f(Xn) then Xc←Xn;
      b←b + 1;
    endwhile
    I←I + 1;
  endwhile
end
```

```
procedure iterated SHC
begin
  select randomly a current string Xc;
  evaluate Xc;
  I←1;
```

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

```
    while (I≤MaxIterations) do
      b←1;
      while (b≤Nbits) do
        generate a new string Xn by flipping
              the b-th bit of string Xc;
        evaluate Xn;
        if random < e^((f(Xn) − f(Xc))/T) then Xc←Xn;
        b←b+1;
      endwhile
      I←I+1;
    endwhile
  end
```

```
procedure SA
begin
  T←Tmax;
  while (T≥min) do
    select randomly a current string Xc;
    evaluate Xc;
    b←1;
    while (b≤Nbits) do
      generate a new string Xn by flipping
            the b-th bit of string Xc;
      evaluate Xn;
      if f(Xc) < f(Xn) then Xc←Xn
```

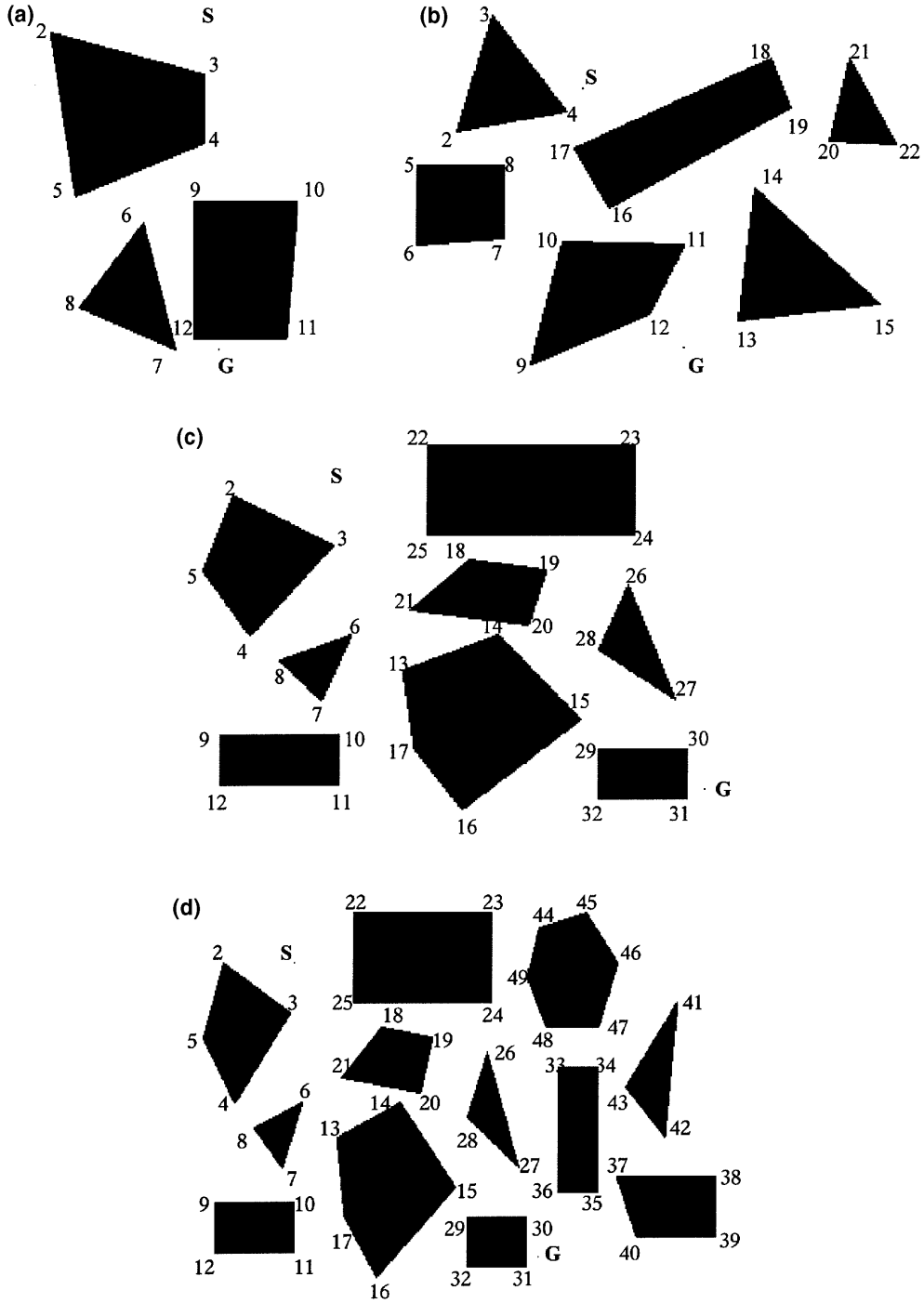

Fig. 1. The robot's environments. Four different aspects of the PPPG with (a) 13-vertices, (b) 23-vertices, (c) 33-vertices, and (d) 50-vertices graphs.

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

Table I. The weight of the loads at each vertex for the 13, 23, and 33-vertices graphs, respectively.

13-VERTICES GRAPH

Vertex	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Load	0	2	1	5	0	1	2	2	2	1	7	1	0

23-VERTICES GRAPH

Vertex	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Load	0	4	1	0	0	1	1	4	5	1	6	10	1
Vertex	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**			
Load	1	1	1	1	10	8	7	1	1	0			

33-VERTICES GRAPH

Vertex	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Load	0	6	7	7	6	8	4	6	0	0	4	1	0
Vertex	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
Load	5	9	10	8	6	7	9	0	4	0	6	4	8
Vertex	**27**	**28**	**29**	**30**	**31**	**32**	**33**						
Load	3	2	1	7	2	7	0						

```
    else if random < e^(f(Xn) − f(Xc))/T then Xc←Xn;
    b←b + 1;
  endwhile
  T←r.T;
endwhile
end
```

f is the evaluation or fitness function, **random** is a function that returns a random number in the range [0, 1) and **Nbits** is the total number of bits in the string, and depends on the size of the PPPG. The HC procedure is a simple version of a steepest ascent hillclimbing algorithm. The algorithm runs for **MaxIterations** iterations. In each iteration it randomly selects a current string solution (called Xc) to the problem. Then, it generates **Nbits** neighbours (called Xn's); each new neighbour Xn competes with Xc, and if it has a larger fitness value it becomes the new current string.

Similarly, the SHC procedure runs **MaxIterations** iterations. In each iteration a new neighbour string (from the **Nbits** in total) is selected to replace the current solution with a probability $e^{(f(X_n) - f(X_c))/T}$. In SHC there is not any "restart" operation, i.e. after accepting a new solution Xn, the algorithm continues searching the solution space from that solution and does not return back to the beginning of enumeration. The parameter **T** (**T** > 0) in the iterated SHC procedure was defined as equal to 10.

In the SA procedure, the starting temperature (**Tmax**) was defined as equal to 10,000, the minimum temperature **Tmin** was defined as equal to 0.1, and the factor **r** ($0 \leq \mathbf{r} < 1$) representing the temperature decay rate was defined as equal to 0.9999. Each time the temperature T does not drop below Tmin, the algorithm selectes randomly a new string Xc to become the new current solution. The algorithm terminates when the system becomes "frozen" (T < Tmin); in this case, typically no more changes are accepted.

Table II. The case of a 13-vertices PPPG. (a) Evaluations performed by each algorithm before converging to the global optimum. (b) Best solutions evaluated for each one of the robot's tasks. Lmax = 15.

13-VERTICES GRAPH

Algorithm	Task-1	Task-2	Task-3	Task-4
HC	923	507	4,745	1,638
SHC	5,639	4,004	6,461	10,530
SA	122,148	517,738	1,000,970	799,591
GA	600	1,500	1,200	2,000

13-VERTICES GRAPH

Tasks	Path cost	Total load	Generated path
1	65.681	9	1-3-4-9-12-13
2	267.76	22	1-2-3-4-6-7-9-10-11-12-13
3	109.49	16	1-3-4-9-10-11-13
4	97.32	14	1-3-4-10-11-13

4. EXPERIMENTAL RESULTS AND DISCUSSION

The effectiveness of the proposed GA for the PPPG was examined through multiple experiments carried out on four different problems. These problems consisted of 13, 23, 33, and 50 vertices graphs. The problems are shown in Figure 1. For each one of these problems the GA was run to optimize each time the tasks described in section 2. The performance of the proposed GA was compared to that of the HC, SHC, and SA heuristics described in sub-section 3.5. The loads located at each vertex in the 13, 23, and 33-vertices graphs are given in Table I.

The performance of each algorithm is measured (a) by the number of evaluations performed until the convergence to the optimum, or to a near-optimum solution, and (b) by the fact of how good the generated best solution to the problem was.

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

In the following discussion, the term evaluation means the computation of the fitness or scoring function of a specific chromosome. Thus, for the proposed GA we have 100 evaluations per generation, and in total 5,000 (for 50 generations) evaluations after its termination. For the case of the iterated HC and the iterated SHC **Nbits** evaluations are done in each new iteration. Therefore, the algorithms can proceed up to **MaxIterations**·**Nbits** evaluations. **MaxIterations** was defined to be equal to 50,000. For the case of the SA technique, each time a new string Xc is randomly selected and evaluated corresponds to an iteration. The iterations proceed until the temperature T becomes less than

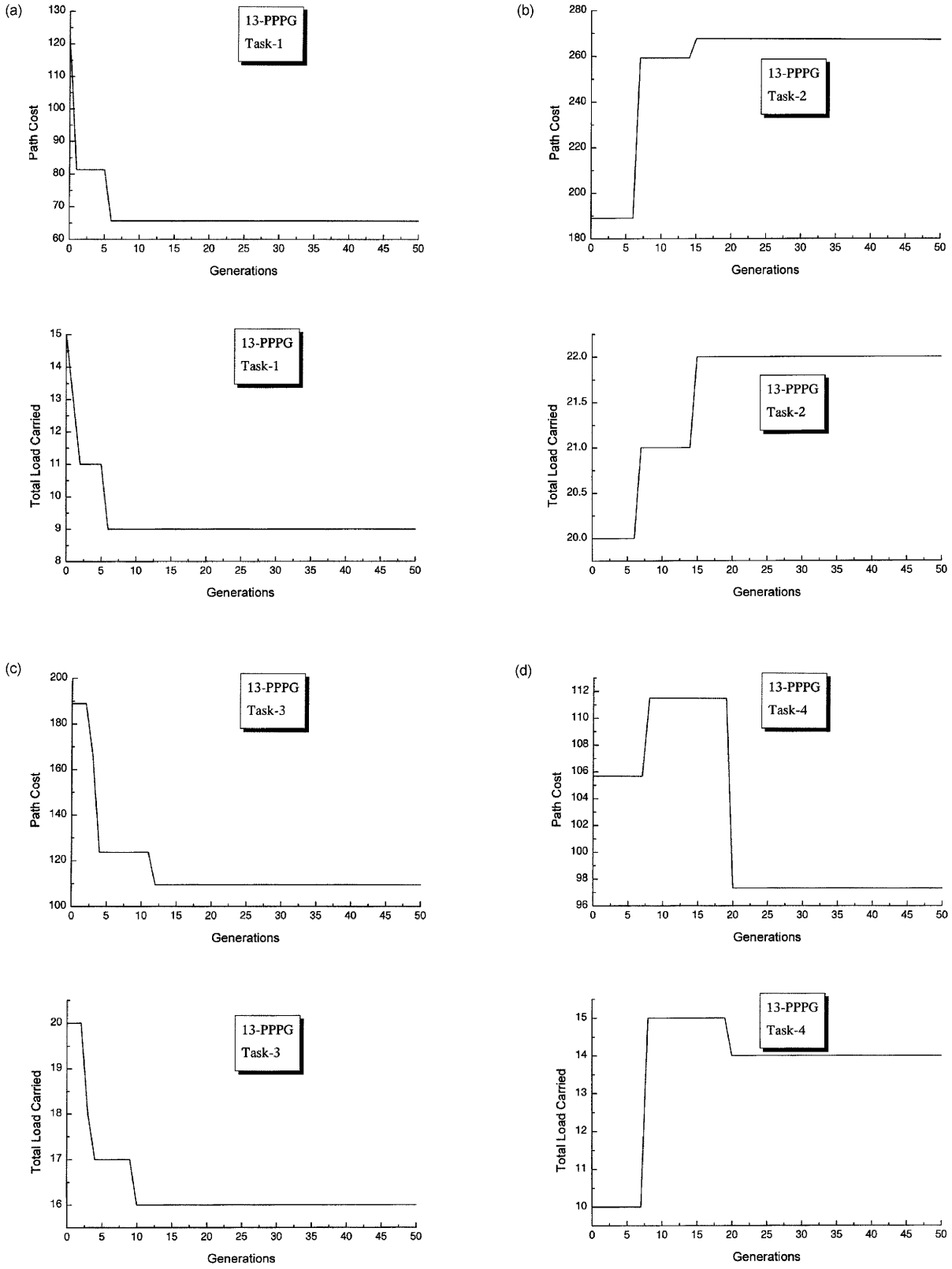

Fig. 2. The case of a 13-vertices PPPG. Evolution of the path cost and total load carried over the successive generations of the proposed GA for each one of the four robot's tasks. Lmax = 15 (for robot's task-4).

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

Tmin. Similarly, in each iteration we have **Nbits** evaluations. Recall that the parameter **Nbits** corresponds to the length of the chromosome (bit-vector) being used, which in fact depends on the size of the specific PPPG. Therefore, **Nbits** was defined equal to 13 for the 13-vertices PPPG, equal to 23 for the 23-vertices PPPG, etc.

Starting from the 13-vertices graph, Table II summarizes the experimental results of the four heuristics for this problem. Specifically, Table II(a) displays the number of evaluations performed by the algorithms for each different task, before converging to the "global" optimum solution. The generated best solutions for each one of the four tasks

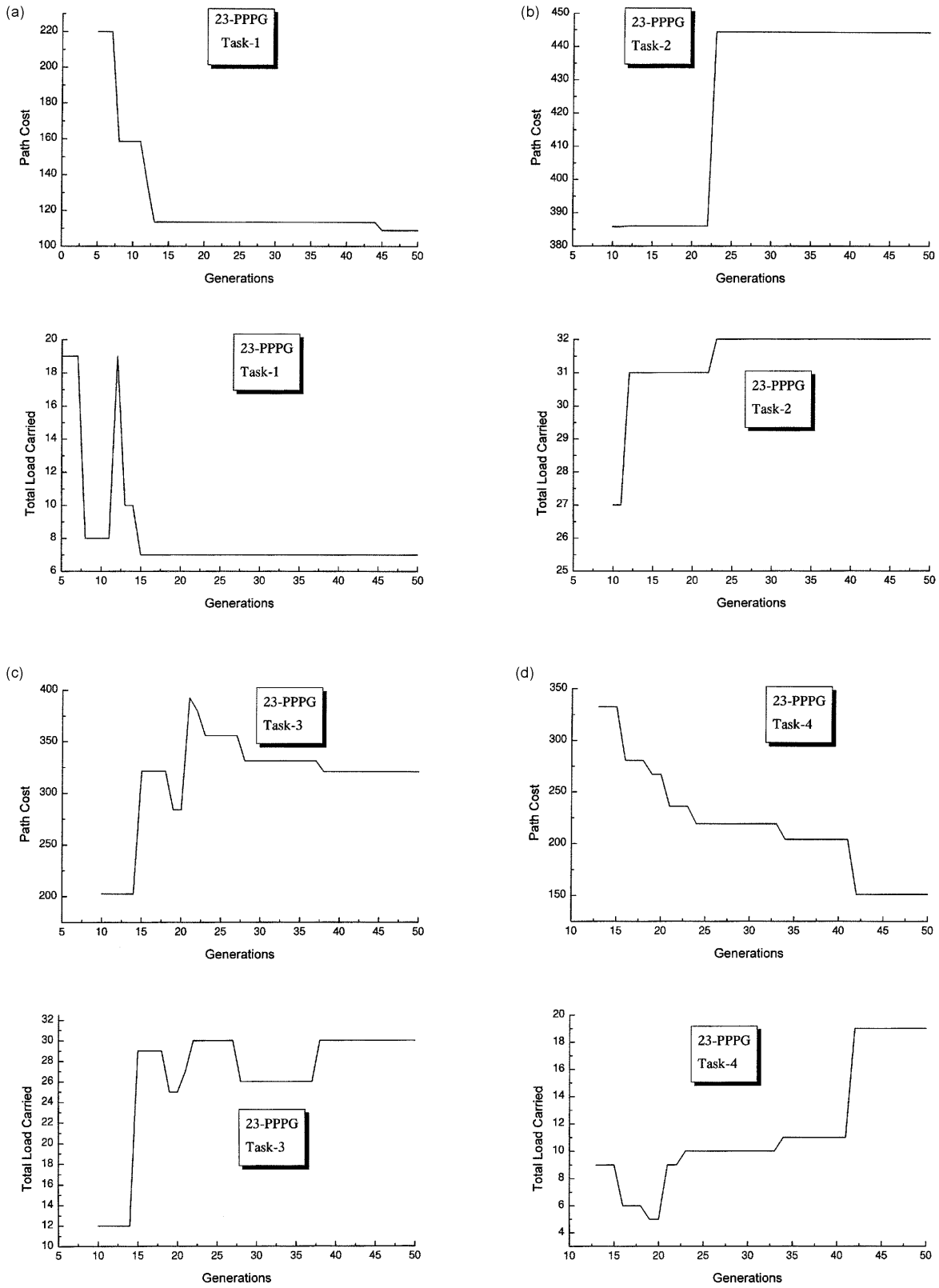

Fig. 3. The case of a 23-vertices PPPG. Evolution of the path cost and total load carried over the successive generations of the proposed GA for each one of the four robot's taks. Lmax = 20 (for robot's task-4).

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

are displayed in Table II(b). The upper limit Lmax for the loads (case of task-4) was defined as equal to 15. Taking into account the speed of convergence (number of evaluations performed before converging to the optimum), we see that the iterated HC and the proposed GA have the highest performance, with the first being slightly better.

Figure 2 displays the cost of the generated best path with the associated total load carried during the evolution of the proposed GA for each one of the four tasks.

Figure 3 concerns the results generated by the proposed GA for the second PPPG. Here the graph has 23-vertices. We can see from the figure the path cost of the best solution (maximum fitness) produced in each generation for each one of the four robot's tasks; accordingly, we can find the corresponding total load carried by the robot for the best solution in each generation of the GA. Note that only the feasible solutions are displayed in the figure, a fact that explains why the curves for each task do not start from the initial generation. The best (maximum) fitness value over the successive generations of the GA for the third and fourth robot's tasks, is shown in Figure 4. For the case of the task-4, Lmax (i.e. the upper permitted limit for the loads), was defined as equal to 20.

Table III summarizes the full comparative results for the 23-vertices PPPG obtained by the four heuristics for each

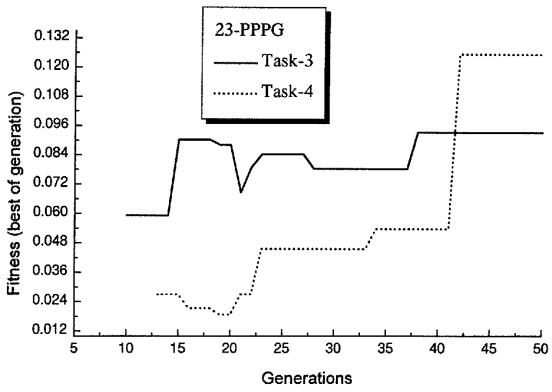

Fig. 4. Transition of the best (maximum) fitness evaluated at each generation of the GA for the tasks 3 and 4 on the 23-vertices PPPG.

one of the robot's tasks. The information shown in each task-table include: the number of evaluations performed by each algorithm until convergence to the near-optimum solution, the path cost, the total load carried by the robot, and the actual generated path corresponding to the best solution. As we can see from these task-tables, the proposed GA and the SA technique gave the best results, with the first being slightly better and obviously faster.

Similarly, the complete comparative simulation results

Table III. Comparative simulation results on the 23-vertices PPPG for the four robot's tasks. Lmax = 20.

(a)

23-VERTICES GRAPH (Task-1)

Algorithm	Evaluations	Path cost	Total load	Generated path
HC	151,317	123.331	5	1-9-23
SHC	221,812	232.740	23	1-4-8-10-11-12-13-14-23
SA	1,433,107	109.091	7	1-4-10-11-23
GA	4,500	109.091	7	1-4-10-11-23

23-VERTICES GRAPH (Task-2)

Algorithm	Evaluations	Path cost	Total load	Generated path
HC	58,305	392.949	27	1-3-4-6-7-9-10-11-12-13-15-23
SHC	60,812	392,949	27	1-3-4-6-7-9-10-11-12-13-15-23
SA	1,504,223	391.297	30	1-3-4-6-7-8-9-10-11-12-13-23
GA	2,200	444.459	32	1-3-4-6-7-8-9-10-11-12-13-14-15-23

(b)

23-VERTICES GRAPH (Task-3)

Algorithm	Evaluations	Path cost	Total load	Generated path
HC	96,738	232.782	25	1-3-6-7-8-10-11-12-13-23
SHC	110,055	352,912	30	1-3-6-7-8-9-10-11-12-14-23
SA	506,874	321.297	30	1-3-4-6-7-8-9-10-11-12-13-23
GA	3,800	321.297	30	1-3-4-6-7-8-9-10-11-12-13-23

23-VERTICES GRAPH (Task-4)

Algorithm	Evaluations	Path cost	Total load	Generated path
HC	88,251	231.994	12	1-4-7-8-9-10-14-23
SHC	87,446	231,994	12	1-4-7-8-9-10-14-23
SA	1,645,604	151.223	19	1-4-7-10-11-12-13-23
GA	4,200	151.223	19	1-4-7-10-11-12-13-23

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

Table IV. Comparative simulation results on the 33-vertices PPPG beween the proposed GA and the SA technique. Lmax = 35.

(a)
33-VERTICES GRAPH (Task-1)

Algorithm	Evaluations	Path cost	Total load	Generated path
SA	2,290,728	131.952	40	1-3-13-14-20-26-28-30-31-33
GA	4,100	90.608	22	1-18-19-28-30-33

33-VERTICES GRAPH (Task-2)

Algorithm	Evaluations	Path cost	Total load	General path
SA	2,384,976	425.055	77	1-6-7-9-10-13-14-15-19-20-21-22-23-24-25-26-27-29-30-31-33
GA	4,900	368.254	86	1-3-4-6-7-8-10-11-13-14-15-19-20-21-22-23-24-27-30-33

(b)
33-VERTICES GRAPH (Task-3)

Algorithm	Evaluations	Path cost	Total load	Generated path
SA	2,252,085	145.361	46	1-6-18-19-24-26-28-30-31-33
GA	4,900	141.209	54	1-3-6-18-19-24-26-27-30-31-33

33-VERTICES GRAPH (Task-4)

Algorithm	Evaluations	Path cost	Total load	General path
SA	2,345,112	111.714	34	1-18-19-20-28-29-30-31-33
GA	4,900	103.067	34	1-18-19-20-27-30-31-33

for the 33-vertices PPPG are shown in Table IV. The characteristic of this table is the absence of the HC and SHC techniques. The reason was the poor performance of these two heuristics compared to that of the GA and SA techniques. The hill-climbing based techniques fail to generate comparative "good" solutions in most of the experiments; a characteristic which is caused due to their inherently local scope of search, and due to the large problem's search space.

Figure 5 shows the path cost and the total load carried by the robot, respectively, over the successive GA's generations for the 33-vertices PPPG. These values correspond to the best solution (maximum fitness) found in each generation of the GA. Accordingly, Figure 6 illustrates the transition of the best (maximum) fitness value evaluated at each generation of the GA for the robot's tasks 3 and 4. Recall that, only the feasible solutions are displayed in the figures. thus, for robot's task-3, the first (generation's best) feasible solution was achieved at the 36th generation. This means that during the first 35 generations, no feasible solution was estimated. For the robot's task-4, the first feasible solution was produced rather more quickly, actually in the 31st generation. Finally, for both tasks, the algorithm was converged to the near-optimal solution at the 49th generation.

Taking into account the stochastic behaviour of the proposed GA a more difficult class of PPPG was examined. This class of problems concerns the path planning on a graph with 50-vertices (see Figure 1(d)). To be more accurate, we run the GA and the SA technique for 1,000 different experiments randomly selected on this graph for the robot's task 2, 3, and 4. In each one of the experiments a load (an integer number in the range [0 . . . 7]) at each vertex of the graph was randomly generated.

Figure 7 displays the generated (by the GA and the SA technique) average load carried over every 20 experiments. The experimental results concern the task-2 of the robot, i.e. move safely from start to destination by picking and carrying as many as possible weight of loads. Therefore, the fitness function was estimated using equation (6). As we can see from the figure, the performance of the proposed GA (solid curve) is higher or at least the same as that of the SA. Further, the GA's speed of convergence is substantially better than that of the SA technique.

Figure 8 shows the comparative results over the 1,000 experiments for the robot's tasks-3. The fitness function was evaluated using equation (7). Figure 8(a) plots the average value of the best fitness functions estimated every 20 experiments. Accordingly, Figure 8(b) shows the associated mean cost of the generated best paths (mean value over every 20 experiments).

Finally, Figure 9 illustrates the comparative results for the robot's task-4. The upper permitted limit for the loads (Lmax) was defined as equal to 40. Again, the values plotted in Figures 9(a) to 9(c) correspond to the average values of the best solutions over every 20 experiments. From these figures the strength of the GA vs. the SA technique for the specific PPPG is given.

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

5. CONCLUSIONS

A genetic algorithm (GA) was designed and used to solve a path planning problem in a graph for a mobile robot which during its motion performs a pick-and-carry loads operation. The proposed GA was used to optimize several task-oriented criteria. Simulation results demonstrates the effectiveness of the algorithm.

The performance of the proposed GA was compared to that of an iterated hill-climbing, an iterated stochastic hill-climbing, and a simulated annealing technique. The

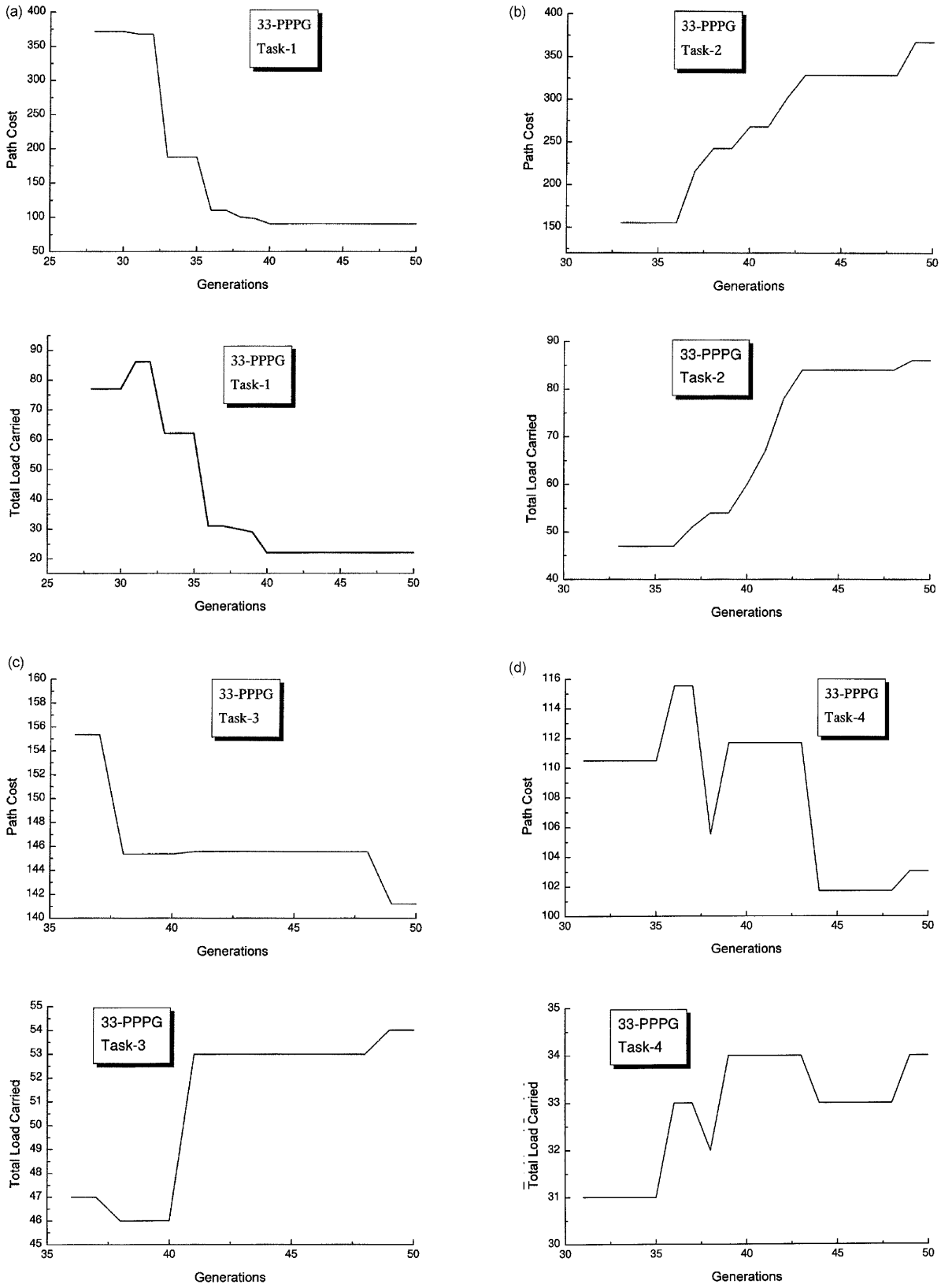

Fig. 5. The case of a 33-vertices PPPG. Evolution of the path cost and total load carried over the successive generations of the proposed GA for each one of the four robot's tasks. Lmax = 35 (for robot's task-4).

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

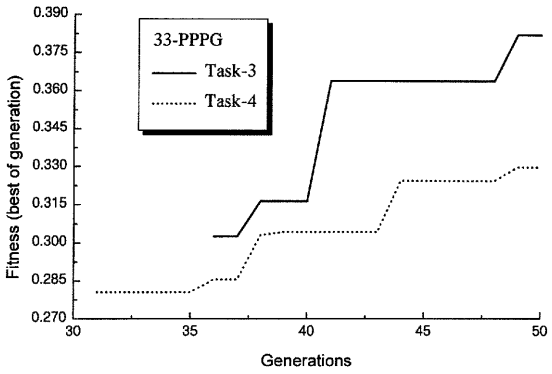

Fig. 6. Transition of the best (maximum) fitness evaluated at each generation of the GA for the tasks 3 and 4 on the 33-vertices PPPG.

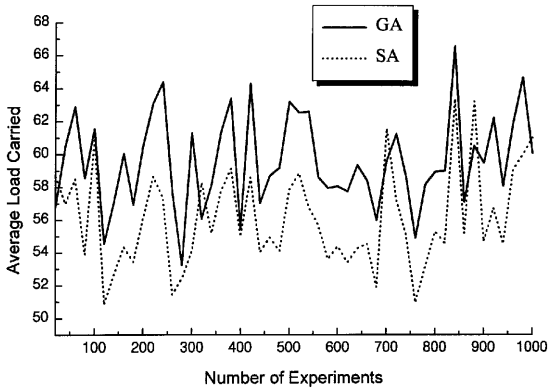

Fig. 7. The case of a 50-vertices PPPG. Comparison between the proposed GA and the SA over 1000 random experiments concerning the robot's task-2.

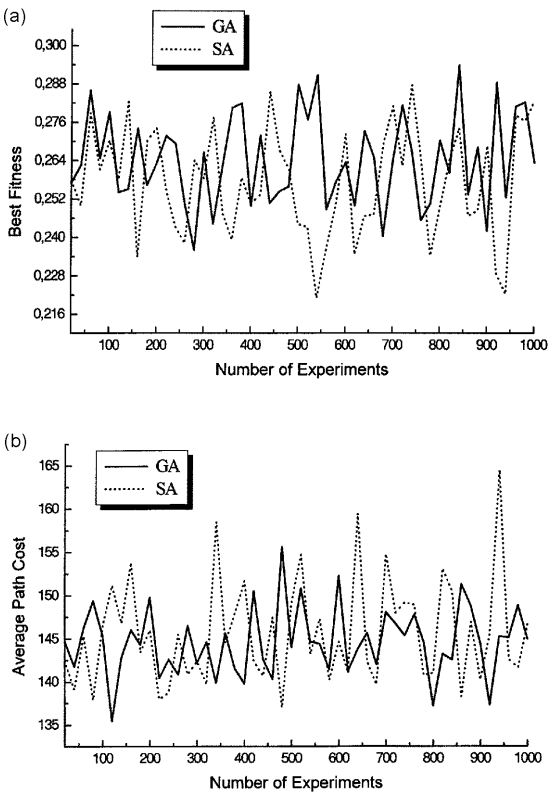

Fig. 8. Comparative results (on the 50-vertices PPPG) between the GA and the SA technique for robot's task-3. The average value of (a) the best (maximum) fitness and (b) the path cost, estimated every 20 experiments.

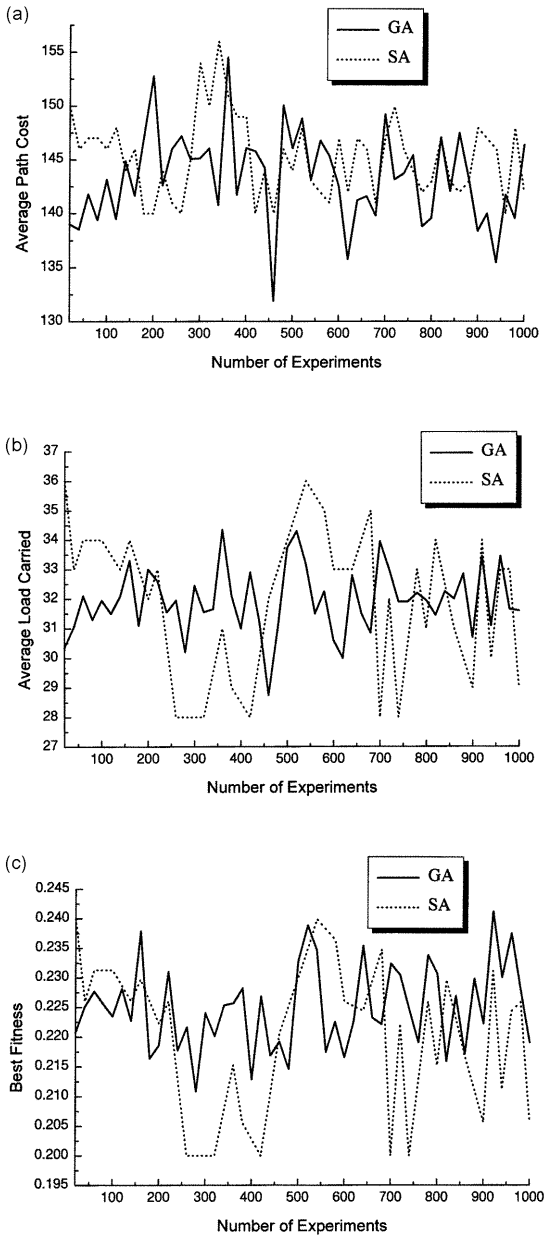

Fig. 9. GA vs. SA for robot's task-4 (Lmax = 40). Evaluations over 1000 random experiments carried out on the 50-vertices PPPG. The average value (estimated every 20 experiments) of: (a) the cost of the generated path, (b) the total load carried, and (c) the best fitness.

algorithms' performance was measured by the number of evaluations performed until a near-optimum solution to the problem has been reached, and by the fact of how good the generated "best" solution to the problem was.

As the problem's search space becomes larger and more complex, the experimental comparisons show a much better performance for the proposed GA and the simulated annealing technique, than that of the hill-climbing based techniques. Further, in all the experiments GA appears to work better and substantially faster than a simulated annealing technique.

References

1. J.E. Hopcroft and D.B. Krafft, "The challenge of robotics for computer science", **In:** *Advances in Robotics, Vol. 1, Algo-*

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press

rithmic and Geometric Aspects of Robotics (Schwartz J.T. and Yap C.K., eds) (Eribaum, Hillsdale, New Jersey, 7–42, 1987).
2. J.T. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms", *Artificial Intelligence* **37,** 157–169 (1988).
3. T. Lozano Perez, "Spatial planning: a configuration space approach", *IEEE Trans. on Computers* **C-32,** No. 2, 108–120 (1983).
4. N.J. Nilsson, "A mobile automaton: an application of artificial intelligence techniques", *Proc. of the 1st Int. Joint Conf. on Artificial Intelligence*, Washington D.C. (1969) pp. 509–520.
5. H. Mitchell, "An algorithmic approach to some problems in terrain navigation", *Artificial Intelligence* **37,** 171–201 (1988).
6. C. O'Dunlaing and C.K. Yap, "A retraction method for planning the motion of a disc", *Journal of Algorithms* **6,** 104–111 (1982).
7. C. O'Dunlaing, M. Sharir and C.K. Yap, "Retraction: a new approach to motion planning", *Proc. of the 15th ACM Symp. on the Theory of Computing*, Boston (1983) pp. 207–220.
8. J-C. Latombe, Robot motion planning, Kluwer Academic Publ., 1991.
9. E. W. Dijkstra, "A note on two problems in connection with graphs", *Numerische Math.* **1,** 269–271 (1959).
10. P.E. Hart, N.J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Trans. on Systems, Man, and Cybernetics* **SMC-4,** No. 2, 100–107 (July, 1968).
11. Y. Davidor, *Genetic Algorithms and Robotics: a heuristic strategy for optimizaton* (World Scientific publishing, Singapore, 1991).
12. H-S. Lin, J. Xiao and Z. Michalewicz, "Evolutonary navigator for a mobile robot", *Proc. of the IEEE Conf. on Robotics and Automation*, San Diego (May, 1994) pp. 2199–2204.
13. A.C. Nearchou and N.A. Aspragathos, "A genetic path planning algorithm for redundant articulated robots", *Robotica* **15,** Part 2, 213–224 (1997).
14. T. Shibata and T. Fukuda, "Intelligent motion planning by genetic algorithm with fuzzy critic", *Proc. of the 8th IEEE Int. Symp. on Intelligent Control*, Chicago, Illinois (August, 1993) pp. 565–569.
15. J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, 1975).
16. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, Mass., 1989).
17. L. Davis, *Genetic Algorithms and Simulated Annealing* (Morgan Kaufmann Publishers, Los Altos, CA, 1987).
18. J.J. Grefenstette, "Optimization of control parameters for genetic algorithms", *IEEE Trans. on Systems, Man, and Cybernetics* **SMC-16,** No. 1, 122–128 (1986).
19. A. Brindle, "Genetic algorithms for function optimization", *Doctoral dissertation* (University of Alberta, Department of Computer Science, Edmonton, 1981).
20. G. Syswerda, "Uniform crossover in genetic algorithms", **In:** (J. David Schaffer, ed.) *Proc. of the Third Int. Conf. on Genetic Algorithms*, San Mateo, California 1989) pp. 2–9.
21. D.H. Ackley, "An empirical study of bit vector function optimization", *Genetic Algorithms and Simulated Annealing* (ed. Davis L.), (Morgan Kaufmann, 1987), Chapter 13, pp. 170–216.

https://doi.org/10.1017/S0263574798000289 Published online by Cambridge University Press