15

Genetic Algorithm Based Automation Methods for Route Optimization Problems

G. Andal Jayalakshmi
Intel,
Malaysia

1. Introduction

Genetic Algorithms (GA) are robust search techniques that have emerged to be effective for a variety of search and optimization problems. The primary goal of this chapter is to explore various Genetic Algorithm (GA) based automation methods for solving route optimization problems. Three real world problems: Traveling Salesman, Mobile Robot Path-Planning and VLSI global routing are considered here for discussion. All the three problems are *Non-deterministic Polynomial (NP)-complete* problems and require a heuristic algorithm to produce acceptable solutions in a reasonable time.

The basic principles of GAs were first laid down by *Holland*. GAs work with a population of individuals each representing a solution to the problem. The individuals are assigned a fitness value based on the solution's quality and the highly fit individuals are given more opportunities in the reproduction. The reproduction process generates the next generation of individuals by two distinct processes named 'crossover' and 'mutation'. The new individuals generated by crossover share some features from their parents and resemble the current generation whereas the individuals generated by mutation produces new characters, which are different from their parents. The probability of crossover operation is usually very high compared to the probability of mutation operation due to the nature of their operations. The reproduction process is carried out until the population is converged which usually takes hundreds of iterations for complex real world problems. The time taken for convergence is dependent on the problem and it is the progression towards increasing uniformity among the individuals of a population.

A standard GA described by *Goldberg* uses binary encoding for representing the individuals, one-point crossover for reproduction which exchanges two consecutive sets of genes and random mutation which randomly alters the selected gene. The probability for applying crossover typically ranges from 0.6 to 1.0 and the probability for mutation operation is typically 0.001. Generally crossover enables the rapid exploration of the search space and mutation provides a small amount of random search to ensure that no point in the search space is given zero probability of being examined.

Traditional GAs are generally not the successful optimization algorithms for a particular domain as they blindly try to optimize without applying the domain knowledge. *L.Davis* states in the "Handbook of Genetic Algorithms", that the "Traditional genetic algorithms

although robust are generally not the most successful optimization algorithms on any particular domain". *Davis* argues that the hybridization will result in superior methods. Hybridizing the genetic algorithm with the optimization method for a particular problem will result in a method which is better than the traditional GA and the particular optimization method. In fact this will produce a more superior method than any of the individual methods. The standard GA can be improved by introducing variations at every level of the GA component including the encoding techniques, the reproduction mechanisms, population initialization techniques, adaptation of genetic parameters and the evolution of the individuals. These thoughts have resulted in a class of genetic algorithms named '*Hybrid Genetic Algorithms*' (HGA). These are the customized genetic algorithms to fit the traditional or simple GA to the problem rather than to fit the problem to the requirements of a genetic algorithm. The HGAs use real valued encodings as opposed to binary encodings and employs recombination operators that may be domain specific.

We will explore further on the use of HGA by discussing a solution to the *Traveling Sales Person* (TSP) problem. The TSP problem has been a typical target for many approaches to combinatorial optimization, including classical local optimization techniques as well as many of the more recent variants on local optimization, such as Simulated Annealing, Tabu Search, Neural Networks, and Genetic Algorithms. This problem is a classic example of *Non deterministic Polynomial hard* problem and is therefore impossible to search for an optimal solution for realistic sizes of *N*. The HGA that is described here is as proposed by *Jayalakshmi* et. al. which combines a variant of an already existing crossover operator with a set of new heuristics. One of the heuristics is for generating the initial population and the other two are applied to the offspring either obtained by crossover or by shuffling. The heuristics applied to the offspring are greedy in nature and hence the method includes proper amount of randomness to prevent getting stuck up at local optimum.

While the hybrid GAs exploit the domain knowledge, in many realistic situations, a priori knowledge of the problem may not be available. In such cases, it is fortunately possible to dynamically adapt aspects of the genetic algorithm's processing to anticipate the environment and improve the solution quality. These are the '*Adaptive GAs*' which are distinguished by their dynamic manipulation of selected parameters or operators during the course of evolving a problem solution. In this chapter we will see an adaptive GA solution to the *mobile robot path planning* problem which generates collision free paths for mobile robots. The problem of generating collision-free paths has attracted considerable interest during the past years. Recently a great deal of research has been done in the area of motion planning for mobile robots as discussed by Choset et al. Traditional planners often assume that the environment is perfectly known and search for the optimal path. On the other hand on-line planners are often purely reactive and do not try to optimize a path. There are also approaches combining offline planers 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. The developments in the field of *Evolutionary Computation* (EC) have inspired the emergence of EC-based path planners. However traditional EC-based planners have not incorporated the domain knowledge and were not adaptive and reactive to the changing environments. Recent research has offered EC-based planners for dynamic environments.

The solution to the *mobile robot path planning* problem discussed here is as proposed by *Jayalakshmi* et. al, which incorporates domain knowledge through domain specific operators

and uses an initialization heuristics to convert infeasible paths into feasible ones. The fitness of the solution is measured based on the number of fragments, acute edges and the angle between the turns in the path. The algorithm plans the path for the current environment and the robot travels in that direction. If an obstacle is found in its path, the robot senses the presence of the obstacle before the critical time to avoid collision and calls the path planner algorithm again to find the new path from that point onwards.

The CHC genetic algorithm proposed by *Eshelman* has emerged as an alternative to resolve the perennial problem with simple GAs which is the premature convergence. The simple GA allows a sub-optimal individual to take over a population resulting in every individual being extremely alike and thus causing premature convergence. The consequence of premature convergence is a population which does not contain sufficient genetic diversity to evolve further. The CHC genetic algorithm uses crossover using generational elitist selection, heterogeneous recombination by incest prevention and cataclysmic mutation to restart the search when the population starts to converge. The CHC GA has a very aggressive search by using monotonic search through survival of the best and offset the aggressiveness of the search by using highly disruptive operators such as uniform crossover.

In this chapter we will also explore a solution to the *VLSI global routing problem* using CHC GA. One of the most important VLSI Design Approaches is the Macro Cell design. Macro cells are large, irregularly sized parameterized circuit modules that are generated by a silicon compiler as per a designer's selected parameters. Usually the physical design process for macro cells is divided into Floor Planning/Placement, Global Routing and Detailed Routing. Floor Planning/Placement constructs a layout indicating the position of the macro cells. The placement is then followed by routing, which is the process of determining the connection pattern for each net to minimize the overall routing area. Before the global routing process begins, a routing graph is extracted from the given placement and routing is done based on this graph. Computing a global route for a net corresponds to finding a corresponding path in the routing graph. Each edge represents a routing channel and the vertex is the intersection of the two channels. First the vertices that represent the terminal of the net are added to the routing graph and then the shortest route for the net is found. Both the placement and routing problems are known to be NP-complete. Thus it is impossible to find optimal solutions in practice and various heuristics are used to obtain a near optimal solution. There has been a lot of work on optimization for routing, including Simulated Annealing algorithms and Genetic Algorithms.

The details of the genetic algorithm solutions to each of these problems are described in the following sections.

2. Design of a hybrid GA for TSP

A heuristic approach employs some domain knowledge in providing a solution to the problem. A good heuristics can be devised provided one has the knowledge of the problem being solved. In cases, where there is no knowledge of the problem, it is best to use a more general heuristic, often called a meta-heuristic. Meta-heuristics are sometimes also called black-box optimization algorithms or simply, general-purpose optimization algorithms. Coding complex data structures by simple lists of bits or real values leads to the problem that has no one-to-one correspondence between these lists and the problem instances. Hence

problem knowledge is necessary either to repair operators to deal with invalid solutions or to design special operators tailored to the problem.

Of the present evolutionary algorithms, hybrid genetic algorithms have received increasing attention and investigation in recent years. This is because of the reason that the hybrid GAs combine the global explorative power of conventional GAs with the local exploitation behaviours of deterministic optimization methods. The hybrid GAs usually outperform the conventional GAs or deterministic methods in practice. To hybridize the genetic algorithm technique and the current algorithm, the following three principles are suggested by *Davis*:

- Use the current algorithm's encoding technique in the hybrid algorithm. This guarantees that the domain expertise embodied in the encoding used by the current algorithm will be preserved.
- Hybridize where possible by incorporating the positive features of the current method in the hybrid algorithm.
- Adapt the genetic operators by creating new crossover and mutation operators for the new type of encoding by analogy with bit string crossover and mutation operators. Incorporate domain based heuristics on operators as well.

Theoretical work as well as practical experience demonstrates the importance to progress from fixed, rigid schemes of genetic algorithms towards a problem specific processing of optimization problems.

This section explores how a HGA is used to solve the TSP problem. The TSP is probably the most studied optimization problems of all times. In the *Travelling Sales Person* problem, given a set $\{c_1 , c_2 , \ldots , c_n\}$ of cities, the goal is to find an ordering π of the cities that minimizes the quantity

$$\Sigma\, d(c_{\pi(i)},c_{\pi(i+1)})+d(c_{\pi(n)},c_{\pi(1)}) \quad 1\leq i\leq n\text{-}1$$

Where d (c_i,c_j) is the distance associated with each pair of distinct cities $<c_i,c_j>$. This quantity is referred to as the *tour_length*, since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permutation, returning at the end, to the initial city. The *Euclidean Travelling Sales Person* problem involves finding the shortest *Hamiltonian Path* or Cycle in a graph of *N* cities. The distance between the two cities is just the *Euclidean* distance between them.

In a *symmetric* TSP, the distances satisfy $d(c_i,c_j) = d(c_j,c_i)$ for $1\leq i, j\leq N$. The symmetric traveling salesman problem has many applications, including VLSI chip fabrication X-ray crystallography and many other. It is NP-hard and so any algorithm for finding optimal tours must have a worst-case running time that grows faster than any polynomial. This leaves researchers with two alternatives: either look for heuristics that merely find *near*-optimal tours, but do so quickly, or attempt to develop optimization algorithms that work well on 'real-world' rather than worst-case instances. Because of its simplicity and applicability the TSP has for decades served as an initial proving ground for new ideas related to both these alternatives.

2.1 The hybrid GA solution

The HGA proposed by *Jayalakshmi* et al. to solve the TSP problem use heuristics for initialization of population and improvement of offspring produced by crossover. The

Initialization Heuristics algorithm is used to initialize a part of the population and the remaining part of the population is initialized randomly. The offspring is obtained by crossover between two parents selected randomly. The tour improvement heuristics: *RemoveSharp* and *LocalOpt* are used to bring the offspring to a local minimum. If cost of the tour of the offspring thus obtained is less than the cost of the tour of any one of the parents then the parent with higher cost is removed from the population and the offspring is added to the population. If the cost of the tour of the offspring is greater than that of both of its parents then it is discarded. For shuffling, a random number is generated within one and if it is less than the specified probability of the shuffling operator, a tour is randomly selected and is removed from the population. Its sequence is randomized and then added to the population.

2.1.1 The crossover operator

The initial city is chosen from one of the two parent tours. This is the current city and all the occurrences of this city are removed from the edge map. If the current city has entries in its edgelist then the city with the shortest edge is included in the tour, and this becomes the current city. Any ties are broken randomly. This is repeated until there are no remaining cities. An example is given below:

Let the distance matrix be

0	10	4	15	5	20
10	0	5	25	5	10
4	5	0	13	6	2
15	25	13	0	6	10
5	5	6	6	0	20
20	10	2	10	20	0

Let the *genotype* p_1 be equal to (2,3,4,5,0,1) which encodes the TSP tour (2,3,4,5,0,1,2) and p_2 be equal to (2,3,1,4,0,5) which encodes the TSP tour(2,3,1,4,0,5,2). The combined edge map M_{12} contains the combined edge relationships from both the parents. The first gene value in p1 i.e. 2 is added to the child c1. Then the gene value 2 is removed from the edge map. The combined edge map before and after are given below:

Gene value	**Edge map (p_1)**	**Edge map (p_2)**	**Combined Edge map M_{12} (before)**	**Combined Edge map M_{12} (after)**
0	5,1	4,5	1,4,5	1,4,5
1	0,2	3,4	0,2,3,4	0,3,4
2	3,1	3,5	1,3,5	1,3,5
3	2,4	2,1	1,2,4	1,4
4	3,5	1,0	0,1,3,5	0,1,3,5
5	4,0	0,2	0,2,4	0,4

Table 1. Combined edge map

Now |E(2)| = 3 therefore an edge j is chosen such that j ε E(2) and |<2,j,>| is minimum and is added to the child c_1. In this example j is 5. Now j is removed from the edge map, and the same procedure is followed until the child c_1 is filled with all the genes. For the example the child c_1 will become (2,5,0,4,1,3).

2.1.2 The Initialization Heuristics

The *Initialization Heuristics* (IH) initializes the population based on a greedy algorithm which arranges the cities depending on their x and y coordinates. The tours are represented in linked-lists. First an initial list is obtained in the input order which is the Input List. The linked-list that is obtained after applying the *Initialization Heuristics* is the "Output List". During the process of applying the *Initialization Heuristics* all the cities in the "Input List" will be moved one by one to the "Output List". Four cities are selected, first one with largest x-coordinate value, second one with least x-coordinate value, third one with largest y-coordinate and fourth one with least y-coordinate value. These are moved from the "Input List" to the "Output List". The sequence of the four cities in the Output List is changed based on minimum cost. The elements in the Input List are randomized and the head element is inserted into the Output List at a position where the increase in the cost of the tour is minimal. This process is repeated until all the elements in the Input List are moved to the Output List.

Figure 1(a) shows a 8-city problem. Figure 1(b) shows the Boundary Tour formed from four extreme cities. Figure 2 (a), (b), (c) & (d) shows the four possible tours that can be formed when city 'E' is moved to the "Output List". It is obvious from the figures that the Tour in Figure 2(a) will result in minimum increase in the cost of the tour in the "Output List". Similarly other cities will be moved one by one to the "Output List".

Fig. 1. (a) Input cities and (b) boundary tour formed by four extreme cities

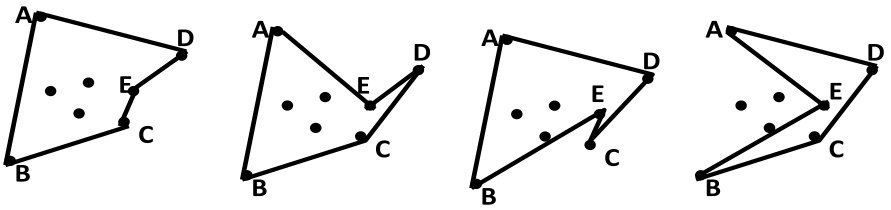
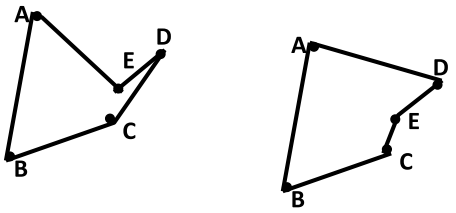


Fig. 2. Various possible tours, which can be formed by moving city 'E' to the output list

2.1.3 The *Removesharp* heuristics

The *RemoveSharp* algorithm removes sharp increase in the tour cost due to a city, which is badly positioned. It rearranges the sequence of a tour by considering the nearest cities of a badly positioned city such that the *tour_cost* is reduced. A list containing the nearest *m* cities to a selected city is created. The selected city from the tour is removed and a tour with *N-1* cities is formed. Now the selected city is reinserted in the tour either before or after any one of the cities in the list previously constructed with m nearest cities and the cost of the new tour length is calculated for each case. The sequence, which produces the least cost, is selected. This is repeated for each city in the tour.

An example is given below:



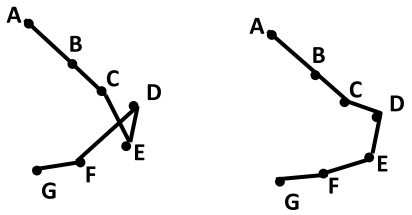
(a) (b)

Fig. 3. (a) A tour with a badly positioned city and (b) The tour after *RemoveSharp* is applied

In Figure 3(a) the city E is in between the cities A and D, while it is obvious that the nearest cities to it are city C and B. *RemoveSharp* will move city E between the cities C and D, resulting in a decrease in the tour cost as shown in Figure 3(b).

2.1.4 The local heuristics

The heuristics finds a locally optimal tour for a set of cities, by rearranging them in all possible orders. The *LocalOpt* algorithm will select *q* consecutive cities $(S_{p+0}, S_{p+1}, \ldots\ldots, S_{p+q-1})$ from the tour and it arranges cities $S_{p+1}, S_{p+2}, \ldots\ldots, S_{p+q-2}$ in such a way that the distance is minimum between the cities S_{p+0} and S_{p+q-1} by searching all possible arrangements. The value of *p* varies from *0* to *n-q*, where *n* is the number of cities. In Figure 4(a) it is quite clear that the distance between the cities A and G can be reduced if some rearrangements are made in the sequence of the cities between them. *LocalOpt* will make all possible rearrangements and replace them to the sequence as shown in Figure 4(b).



(a) (b)

Fig. 4. (a) A bad tour and (b) The tour after *LocalOpt* is applied

2.2 Results

The results for the HGA solution for 3 standard TSP problems are compared with the results for GA and SA solutions. The best integer *tour_length* and the best real *tour_length* (in parenthesis) are tabulated below in Table 2. NA represents "Not Available".

The heuristics with which Hybrid GA is compared here are GA and SA as reported by *Whitley. D* et al. and TSPLIB. The difference between integer and real tour length is that in the first case distances are measured by integer numbers, while in the second case by floating point approximations of real numbers. In TSPLIB website only Eil51 and Eil76 are available which have an additional city to Eil50 and Eil75 respectively.

Problem name	**HGA**	**GA**	**SA**
Eil50 **50-city problem**	426 (428.871) {for Eil51}	428 (NA)	443 (NA)
Eil75 **75-city problem**	538 (544.36) {for Eil76}	545 (NA)	580 (NA)
KroA100 **100-city problem**	21282 (21285.44)	21761 (NA)	NA (NA)

Table 2. Comparison of HGA with other heuristics on geometric instances of the symmetric TSP.

3. Design of an adaptive GA for mobile robots

Adaptive Genetic algorithms dynamically manipulate selected parameters or operators during the course of evolving a problem solution. Adaptive GAs are advantageous over SGAs in that they are more reactive to unanticipated characteristics of the problem and can dynamically acquire information about the problem characteristics and exploit them. As described by *Davis*, adaptive GAs can be categorized based on the level at which the adaptive parameters operate. *Population-level* techniques dynamically adjust parameters that are global to the entire population. *Individual-level* adaptive methods modify a particular individual within the population depending on how it is affected by the mutation operators. *Component-level* adaptive GAs dynamically alter the individual components depending on how each individual will be manipulated independently from each other. The operator probabilities play a major role in determining the solution quality and the convergence rate. Since the range of potential applications of genetic algorithms is infinite, it is difficult to measure the goodness of the parameter values. This suggested the idea of adapting the operator probabilities during the evolution of the GA.

The path planner proposed by *Jayalakshmi* et al. incorporates domain knowledge in the algorithm through domain specific operators. An initialization heuristics is used to convert infeasible paths into feasible ones. The fitness is measured based on the number of fragments, acute edges and the angle between the turns in the path. The algorithm plans the path for the current environment and the robot travels in that direction. If an obstacle is found in its path, the robot senses the presence of the obstacle before the critical time to avoid collision and calls the path planner algorithm again to find the new path from that point onwards. The following sections describe the solution.

3.1 The adaptive GA solution

The path planner algorithm design has four major phases. The first phase is the design of the *Initialization Heuristics* (IH), which includes the *Backtrack* and *Change_Y* operators. The algorithm initializes the population randomly and then repairs the population by applying these two operators. This leaves the initial population free of any infeasible solutions and reduces the search region considerably. The heuristics is discussed in detail in a later section.

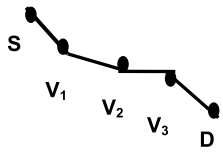
The second phase is the design of domain specific genetic operators *End_New*, *Mid_New*, *Shake* and *Adjacent* to change the characteristics of the path. The operators tune the path generated by removing the sharp edges, inserting adjacent segments in the path and introducing new vertices. The operators are discussed in detail in section 3.1.4.

The third phase is the design of the objective function which is designed to include the smoothness factors of the path in calculating the fitness value. This ensures that the path is not only optimal in length but also smooth without any sharp turns. The objective function is discussed in detail in section 3.1.5. The fourth phase of the algorithm is the design of the adaptive rules to evolve the operator probabilities. The operator probabilities are adapted based on the smoothness factors. The adaptive rules are discussed in section 3.1.6. The binary tournament selection scheme is used to select a parent chromosome and the reproduction is carried out by refining the paths in the previous generation using the domain specific genetic operators described in section 3.1.4. The complete algorithm is given below:

```
Adaptive_Path_Planner()
Begin
    Initialize the population using the heuristics
    Calculate the objective function and evaluate the population
    While not convergence do
    Begin
        Repeat
            Select the parent using binary tournament selection
            Apply the domain specific genetic operators with the optimal
            probability and produce offspring
            Replace the parent with the offspring
        Until the next generation is filled
        Evaluate the population based on the objective function
        Tune the operator probability
    End
End
```

3.1.1 The chromosome structure

A chromosome in a population represents a feasible/infeasible path for the robot to reach the goal location. It consists of the starting point followed by the intersection points of the line segments of the path, and finally the goal position or the ending point. A path can have a varied number of intermediate nodes. And hence the length of the chromosomes will be a variable. An initial population of chromosomes is randomly generated such that it has a random number of intermediate edges with random coordinates. A sample path and its representation are given below:



S, V_1, V_2, V_3, D

Chromosome

Fig. 5. Example path and its equivalent chromosome

3.1.2 The selection scheme

The binary tournament selection scheme is used to select a parent chromosome. In binary tournament selection, pairs of individuals are picked at random from the population; whichever has the higher fitness is copied into a mating pool. This is repeated until the mating pool is full. In this, the better individual wins the tournament with probability p, where $0.5 < p < 1$. By adjusting tournament size or win probability, the selection pressure can be made arbitrarily large or small.

3.1.3 The initialization heuristics

Each chromosome shall represent a feasible or an infeasible path. A random initialization generally leads to a large number of infeasible paths and hence a heuristics is used to convert infeasible paths to feasible ones. The heuristics involve two operators: *Change_Y* and *Backtrack*. The *Change_y* operator is used to change the value of Y coordinate of the vertex which when included leads to an infeasible path. This makes the robot go up a step to take a different path so that the collision with the obstacle is avoided. The *Backtrack* operator is used to take a different path when the path already taken by the robot leads to an obstacle. It allows the robot to go back by two steps in the original path and take up a new path. It is designed to go back two steps because when the robot realizes an obstacle, it will be very nearer to the obstacle and going back one step may not lead to a feasible path.

3.1.4 The genetic operators

The traditional genetic operators Mutation and Crossover cannot be used as such here, hence they are given new forms to accommodate the requirements of the problem to be solved. The operators are designed having in mind the nature inspired actions, a person shall take to avoid collisions with obstacles. The operators are described below:

Mid_New: A vertex is chosen randomly and the edges connecting it to the previous and the next vertices are altered so that any steep increase in the path can be eliminated. The mid points of the edges are considered recursively until a feasible path is found. This helps to take up a closer but safer path around the obstacle

End_New: A vertex is chosen randomly and is removed from the path provided the resultant path is feasible. This operator helps to reduce the number of fragments in a path.

Shake: This operator chooses a vertex randomly and changes its Y coordinate by either adding or subtracting a constant value. This works like a mutation operator.

Adjacent: This operator changes the original path by interchanging a segment of a path with a parallel segment by either adding or subtracting a constant value with the Y coordinate of each vertex in that segment.

3.1.5 The evaluation function

The fitness of a chromosome is calculated based on the following factors:

- Length
- Feasibility
- Number of acute edges
- Number of bends
- Number of fragments

Two objective functions are designed; one based on length and the other on smoothness. The objective function *Simple_Obj* calculates the fitness of a path on the basis of the length of the path and its feasibility. The objective function *Smooth_Obj* calculates the fitness based on the smoothness of the path. The objective functions are described in the following sections.

3.1.5.1 The objective function based on length and feasibility

The fitness is calculated based on the length and feasibility. The feasibility is measured by checking whether the next node on the path generated so far leads to a collision with any of the obstacles. The length_constant is a large integer value which when divides the path length will give high fitness value for the path with minimal length.

$$Fitness\ F(path) = \left(\frac{Kl}{total\ length\ (path)}\right) + Kf * Feasibility(path)$$

Where K_l is the length_constant (a large integer value) and K_f is the feasibility_constant (an integer value).

$$\text{total\_length(path)} = \Sigma\, \text{length}(i, i+1),\ 1 \le i < n, n = \text{No. of nodes}$$

$$feasibility(path) = \begin{cases} 1, & for\ a\ feasible\ path \\ -1, & for\ an\ infeasible\ path \end{cases}$$

3.1.5.2 The objective function based on smoothness

The objective function *Simple_Obj* discussed above takes into account only the path length and the feasibility factor whereas the *Smooth_Obj* takes into account the other factors such as

the number of acute increases/decreases, turns with angle 90° and the number of fragments in the path. The new objective function *Smooth_Obj* is given below:

$$Fitness\ F(path) = \left(\frac{Kl}{total\_length(path)}\right) + Ks * smoothness(path)$$

$$Smoothness(path) = \left(\frac{50 * SA}{100}\right) + \left(\frac{40 * SB}{100}\right) + \left(\frac{10 * SF}{100}\right)$$

where

$$SA = 1 - \frac{Acute\_Count * 100}{NF}$$

$$SB = 1 - \frac{Bend\_Count * 100}{NF}$$

$$SF = \frac{Ideal\ Fragements\ * 100}{NF}$$

And NF is the number of Fragments

The *Acute_Count* is the number of sharp increases / decreases in the path, the *Bend_Count* is the number of rectangular turns and the *Ideal_Fragments* is the minimum number of fragments of all the paths that occur in a generation. Since the number of acute edges affects the smoothness of the path largely, its contribution is 50% in the calculation of the fitness value. The number of bends is given 40% share and the fragments 10% share in the calculation of the fitness value.

3.2 The dynamic environment

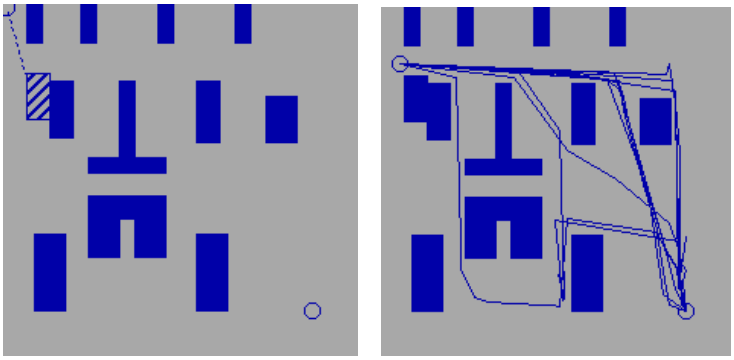
The robot travels in an environment where the obstacles may get introduced suddenly in the path planned by the robot for travelling. In such situations the robot has to decide at every step whether to take up the already planned path or a new path. The robot is assumed to travel at a fixed speed and has a sensor in it to detect the existence of any obstacle. When an obstacle is recognized by the robot it calls the dynamic path planner algorithm and plans its path from that position onwards. The sensor is assumed to sense the existence of any obstacle before the robot reaches the region and the path planner returns the path within the critical time. Now the new path is taken up from that position and the same procedure is repeated until the robot has reached the destination.

3.3 Results

A sample output for the adaptive GA solution is given below. The Robot is assumed to travel in a dynamic environment of dimension (5,5) to (400,400) with different kinds of obstacles placed randomly. A dynamic environment is created by adding new obstacles on the path planned by the robot.

The path planning algorithms for dynamic environments are computationally intensive and hence will take longer time to converge for an environment with non-rectilinear obstacles.

Faster genetic operators and multi-threading methods might help to speed up the path planning process in these environments.



(a) (b)

Fig. 6. (a) A dynamic obstacle in the planned path and (b) the final generation of paths from the point of intervention

4. Design of a Hybrid CHC GA for VLSI routing

The Minimum Rectilinear Steiner Tree (MRST) problem arises in VLSI global routing and wiring estimation where we seek low-cost topologies to connect the pins of signal nets. The Steiner tree algorithm is the essential part of a global routing algorithm. It has been an active field of research in the recent past. This section presents a Hybrid CHC (HCHC) genetic algorithm for global routing. The *Minimum Rectilinear Steiner Tree* problem is to construct a tree that connects all the n points given in the Euclidean plane. If the edges in this tree are to be selected from all possible edges that are from the complete graph on the points, it is the familiar problem of finding a spanning tree in an undirected graph. If the edges of the tree must be horizontal and vertical, the additional points where the edges meet are called the Steiner points, and the resulting tree is a Rectilinear Steiner Tree [RST]. A shortest such tree on a set of given points is a minimum rectilinear Steiner tree.

4.1 Construction of Minimum Spanning Tree

The Spanning tree algorithm presented here is based on the shortest path heuristic as described by *Ellis Horowitz* et al. A simple genetic algorithm is used for the construction of a Minimum Spanning Tree (MST), which is then used in the generation of the Steiner minimal tree. The spanning tree is generated by initializing the population with random solutions. The random solutions are then repaired using a repair heuristics. The offspring are generated by applying one point crossover and exchange mutation which exchanges edges in an individual. The exchange of edges may lead to a totally different tree, thus justifying the purpose of mutation. The new population is evolved and the same procedure is repeated until convergence.

The algorithm for the construction of the minimum spanning tree is given below.

Minimal_Spanning_Tree()

Begin

Initialize parameters: generation count, crossover and mutation probabilities

Initialize parent population randomly

Apply repair heuristics to the parent population

While termination condition not reached

Begin

Select parents based on the total length of the Spanning tree

Apply crossover and mutation

Evolve new population

Replace previous population by new population

End

End

The repair heuristics removes cycles and repeated edges from the population and makes it a set of feasible solutions. The vertices of the graph are stored in separate sets, so that it can be later combined whenever an edge is included in the final spanning tree. The *union* algorithm unions two sets containing V_i and V_j respectively when an edge E <V_i, V_j> is added to the final spanning tree. The *find* algorithm verifies whether a particular vertex belongs to a set. The *union* algorithm combines all vertices that are connected, in to a single set. When a particular edge is selected for addition into a partially constructed spanning tree, it is checked whether the vertices of that edge are already present in the same set using the *find* algorithm. If they are in the same set, then the inclusion of this edge will lead to a cycle. The repeated edges can be checked easily with the adjacency matrix.

4.2 Construction of minimum steiner tree

The Steiner tree problem can be defined as the subset of minimum spanning tree problem. In minimum spanning tree construction, a tree is constructed with vertices $V_1,V_2,\ldots V_n$ connected without loops at the lowest cost. In the Steiner tree problem, extra vertices are added besides the existing vertices $V_1,V_2,\ldots V_n$, to construct a lower cost tree connecting $V_1,V_2,\ldots.V_n$. The extra vertices are called the Steiner points. There are various heuristics available to construct a MRST, and most of them use MST as a starting point. The I-Steiner algorithm as discussed by *Kahng A.B* et al., constructs the MRST by evaluating all possible Steiner points for their impact on MST cost. The algorithm operates on a series of passes, in each pass the single Steiner point which provides the greatest improvement in spanning tree cost is selected and added to the set of demand points. Points are added until no further improvement can be obtained.

The heuristics used here for the construction of MRST is the "BOI" or "edge-removal technique" of *Borah*, *Owen* and *Irwin*. The algorithm constructs the Steiner tree through repeated modification of an initial spanning tree as discussed by Jayalakshmi et al. An edge

and a vertex pair that are close to each other in an MST are determined. For each vertex V_i, edge E_i pairing of the spanning tree, an optimal Steiner point is found to merge the endpoints of the edge E_j with vertex V_i. This will create a cycle, so the longest edge on this cycle is found and a decision is made about removing this edge from the cycle based on the cost. Among all possible eliminations, whichever leads to the lowest cost is removed and the tree is modified. The edges are removed and new connections are inserted until no improvements can be obtained. The resulting tree is the minimum Steiner tree. The approach has low complexity with performance comparable to that of I-Steiner. The algorithm for the construction of the minimum Steiner tree is given below:

```
Steiner_Tree()

Begin

        Build the routing graph G

        For each net do

        Begin

                Initialize weights for edges

                Find the minimum cost spanning tree T

                For each <vertex,edge> pair of the spanning tree

                Begin

                        Find the optimum Steiner point to connect this edge to the vertex
                at a suitable point

                        Find the longest edge on the generated cycle

                        Compute the cost of the modified tree and store the pair in a list
                if the cost is less than the MST

                End

                While the list is not empty do

                Begin

                        Remove the pair from the list which results in lowest cost

                        Re-compute the longest edge on the cycle and the cost of the tree

                        If the edges to be replaced are in the tree and the cost is less then
                modify the tree

                End

        End

End
```

4.3 Results

The HCHC solution for four standard test problems B1, B3 B6 and B9 from the problem sets of *J.E.Beasley* are given below in Table 3. A simple GA with one point crossover and exchange mutation is compared with the HCHC solution. In HCHC, Uniform crossover and External mutation are used for reproduction.

Test problem	Optimum	Solution		Error		Generations	
		SGA	HCHC	SGA	HCHC	SGA	HCHC
B1	82	187	95	105	13	150	200
B3	138	145	140	7	2	150	200
B6	122	128	125	6	3	400	250
B9	220	241	224	21	4	150	200

Table 3. The solutions obtained by SGA and HCHC for Beasley's test problems B1, B3,B6 and B9

For HCHC algorithm the maximum error is for the test problem B1 and for the rest of the problems the error is less than 6. And SGA has performed very poorly for B1 and B9. For the other problems, SGA has performed moderately well with error less than 10.

5. Summary

With Genetic Algorithms emerging as strong alternative to traditional optimization algorithms, in a wide variety of application areas, it is important to find the factors that influence the efficiency of the genetic algorithms. The simple GAs are found to be ineffective for most of the real world problems. Hence there arises the need for the customization of the traditional GAs. This chapter explored the variants of the simple genetic algorithm and their application to solve real world problems. TSP is a problem of a specific domain and required hybridization for quicker convergence. In particular the local search algorithm chosen has a determining influence on the final performance. The heuristics used were simple and easy to implement when compared to other algorithms. The solution to the mobile robot path planning problem explored the design of different operators and showed that the adaptation of operators has a significant impact in improving the solution quality. A hybrid CHC algorithm was used to solve the VLSI global routing problem. And this example showed that the simple GA could only find a sub optimal solution and could not go beyond certain values due to the lack of techniques that avoid premature convergence.

6. References

Beasley.J.E. (1989). An SST-Based Algorithm for the Steiner Problem in Graphs, Networks, Vol.19, pp.1-16, l989.

Borah.M, R. M. Owens & M. J. Irwin. (1994). An edge-based heuristic for Steiner routing, *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1563–1568, Dec. 1994.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., & Kavraki, L. E. (2005). *Principles of robot motion: theory, algorithms, and implementations*. Boston, MIT Press.

Davis.L, (Editor). (1991). "*Handbook of Genetic Algorithms* ", Van Nostrand Reinhold.

Ellis Horowitz, Sartaj Sahni & Sanguthevar Rajasekaran. (2007). *Computer Algorithms*, Silicon Pr.

Eshelman,L.J. (1991). *The CHC Adaptive Search Algorithm: How to have safe search when engaging in nontraditional genetic recombination*, Rawlins G.(Editor), Foundations of Genetic Algorithms, Morgan Kaufmann, pp.265-283.

Goldberg, David E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning,* Addison Wesley, ISBN 0201157675.

Goldberg, David E. (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, Addison-Wesley, Reading, MA.

Holland, John H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Hu.J & Sapatnekar.S.S. (2001). Performance driven global routing through gradual refinement, IEEE Int. Conf.on Comp Design, 2001, pp.481-483

Jayalakshmi.G.A, S. Sathiamoorthy & R. Rajaram. (2001). A Hybrid Genetic Algorithm – A New Approach to Solve Traveling Salesman Problem, *International Journal of Computational Engineering Science*, Volume: 2 Issue: 2 p.339 – 355.

Jayalakshmi, G.A, Prabhu. H, & Rajaram. R. (2003). An Adaptive Mobile Robot Path Planner For Dynamic Environments With Arbitrary-Shaped Obstacles, *International Journal of Computational Engineering Science*(2003),pp. 67-84

Jayalakshmi, G.A., Sowmyalakshmi.S & Rajaram.R (2003). A Hybrid CHC Genetic Algorithm For Macro Cell Global Routing, *Advances in Soft Computing* Engineering Design and Manufacturing Benitez, J.M.; Cordon, O.; Hoffmann, F.; Roy, R. (Eds.) pp. 343 – 350, Springer-Verlag, Aug 2003

Kahng.A.B and G.Robins. (1992). *A new class of iterative Steiner tree heuristics with good performance*, IEEE Trans on Computer Aided design of Integrated circuits and systems, 11(7), pp.893-902.

Li-Ying Wang, Jie Zhang & Hua Li. (2007). An improved Genetic Algorithm for TSP, *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics*, Hong Kong, 19-22 August 2007.

TSPLIB: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html retrievable as of 3rd Feb 2012

Whitley.D, Lunacek.M & Sokolov.A. (2006). Comparing the Niches of CMA-ES, CHC and Pattern Search Using Diverse Benchmarks, *Parallel Problem Solving from Nature Conference (PPSN 2006)*, Springer.

Whitley.D, T. Starkweather & D. Shaner .(1991). The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination. *The Handbook of Genetic Algorithms.* L. Davis, ed., pp: 350-372. Van Nostrand Reinhold.

Zhou.H. (2004). Efficient Steiner Tree Construction Based on Spanning Graphs, IEEE Transactions on Computer-Aided Design of Integrated Circuits And Systems, Vol. 23, No. 5, May 2004, pp:704-710.