A genetic algorithm for nonholonomic motion planning

Gorkem Erinc
School of Engineering and Science
International University Bremen – Germany

Stefano Carpin
School of Engineering
University of California, Merced – USA

***Abstract*— The paper presents a genetic algorithm to find and optimize solutions for nonholonomic motion planning problems. Mainly focusing on mobile robots, the algorithm uses present randomized algorithms to come up with suboptimal paths and iteratively optimizes them according to a fitness function which includes domain specific knowledge. The major advantages of this method include being an any-time algorithm, and improving the quality of the solution throughout the evolutionary process. An extensive experimental analysis comparing our results with state of the art algorithms outline the effectiveness of the proposed methodology.**

***Index Terms*— nonholonomic motion planning, genetic algorithms, mobile robots**

I. Introduction

Since the publication of Laumond's seminal paper on non-holonomic motion planning [1], there has been an ever increasing amount of research devoted to the subject. The grounds of this vein of research are found in the fact that most robots indeed involve nonholonomic constraints. This is specially true for wheeled mobile robots. However, even the basic motion planning is well known for being a problem exhibiting intrinsic high complexity (*NP-HARD*) [2]. Nonholonomic motion planning is therefore not amenable to easy to compute solutions. As outlined in section III, along the years different solutions were proposed. Some are able to solve the problem very quickly, but without any guarantee for what concerns *path quality*. Of course, having the possibility to choose between multiple solutions, we would prefer solutions featuring shorter or safer paths. Other solutions instead produce solutions that can be very close to the shorter one, but, not surprisingly, their time requirements dramatically grow when closer approximations are sought. Realizing that motion planning can be seen as an optimization problem, in this paper we propose a genetic algorithm for motion planning of car-like vehicles in static environments that aims to position itself between these two extremes. The presented algorithm iteratively refines a set of candidate solutions according to a fitness function taking into account not only path length, but also other criteria, an aspect often not addressed in other motion planning algorithms. A set of four domain specific genetic operators to evolve the population over time has been developed. Finally, it is worth outlining that while genetic algorithms were already used to study holonomic motion planning problems, to the best of our knowledge, the application to nonholonomic problems is scarce. The paper is organized as follows. Section II formalizes the problem as well as the vehicle model used throughout the paper. Related literature relevant for the current paper is presented in section III. Section IV is the core of the paper, where the different components of the genetic algorithm are presented. Comparative experimental results are illustrated in section V, and conclusions are offered in the closing section VI.

II. Problem formulation

We shortly formalize the nonholonomic motion planning problem. The interested reader is referred to the book by LaValle [3] for a comprehensive presentation of the topic. We mainly follow the notation used therein. Let $\mathcal{W}$ be the *workspace* where the robot $\mathcal{A}$ moves, and let $\mathcal{O} \subset \mathcal{W}$ be the obstacle region in the workspace. Associated with $\mathcal{A}$ there exists a configuration space $\mathcal{C}$. Indicating with $\mathcal{A}(q)$, $q \in \mathcal{C}$, the subset of $\mathcal{W}$ occupied by $\mathcal{A}$ when it is at configuration q, we define $\mathcal{C}_{obs}$ and C_{free} as follows:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$$

Given $q_{init}, q_{goal} \in \mathcal{C}_{free}$, the *piano movers problem*, i.e. the holonomic motion planning problem, asks to determine a continuous path $p : [0, T] \rightarrow \mathcal{C}_{free}$ such that $p(0) = q_{init}$ and $p(T) = q_{goal}$. The fact that p cannot enter $\mathcal{C}_{obs}$ is a so called *global constraint*. In the *nonholonomic* motion planning problem, a set of additional non integrable constraints in the form

$$g(q, \dot{q}) \bowtie 0 \tag{1}$$

have to be satisfied for each q along the path, where $\bowtie$ can be $=, <, \leq, >$ or $\geq$. These constraints are *local*, and state that the robot cannot achieve arbitrary velocities, because they must always satisfy relationship 1. Assuming that at each configuration $q \in C_{free}$ a set of inputs $U(q)$ can be applied to the robot, local constraints can also be expressed in parametric form, i.e.

$$\dot{q} = f(q, u).$$

The meaning is the same, i.e. the velocity cannot be freely chosen, but it is rather constrained by the configuration and the set of inputs available at that configuration. One of the simplest examples of nonholonomic system is the car. Its configuration space is $\mathbb{R}^2 \times SO(2)$, i.e. its configuration is $q = (x, y, \theta)$ with $\theta \in [0, 2\pi)$. The car cannot move

sideways, but in the short time interval it always moves along the direction of the rear wheels. This constraint can be expressed as

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0$$

which is a non integrable instance of equation 1. Let L be the distance between the front and the rear wheels. The same constraint can be expressed in parametric form assuming that two inputs u_s, speed, and u_ϕ, steering, can be applied.

$$\begin{aligned} \dot{x} &= u_s\cos\theta \\ \dot{y} &= u_s\sin\theta \\ \dot{\theta} &= \frac{u_s}{L}\tan u_\phi \end{aligned}$$

By restricting $u_s \in \{-1, 1\}$ and $u_\phi \in \{\phi_{max}, \phi_{min}\}$ we obtain the famous Reeds and Shepp car. By disallowing the input $u_s = -1$ the Dubins car is obtained. This is the model that will be used from now on.

III. Related work

A significant amount of research has been devoted to non-holonomic motion planning. Due to space constraints, we here briefly report only about the methods relevant for the sequel of this paper. A solution to the nonholonomic problem was proposed by Barraquand and Latombe [4]. Their approach, also called Forward Dynamic Programming (FDP), is based on dynamic programming. In order to speed up the process, the configuration space is preliminary divided into cells whose shape is a parallelepiped. Initially every cell is labeled as *free*. The algorithm starts by creating a tree rooted at the starting configuration q_{init}. Leaves in the tree are additionally stored in a priority queue Q sorted according to the *cost-to-come*. At every iteration the algorithm extracts the leaf with the lowest cost from Q. Let this configuration be q_{min}. All applicable inputs are applied to q_{min} in order to create its descendants. Each of the descendent configurations is validated to determine whether it should be inserted in Q or not. A configuration not in $\mathcal{C}_{free}$ or lying in a cell marked as *visited* is discarded. Otherwise, after its cost is calculated it is inserted in Q and the corresponding cell label is changed to *visited*. This process is iterated until the cell containing q_{goal} receives the *visited* label. Fine grained cell subdivisions lead to solutions approximating better and better the best solution. Clearly, the finer a subdivision, the higher the time needed to explore it. The Rapidly-exploring Random Trees (RRT) algorithm introduced by LaValle [5] also builds a tree $\mathcal{T}$ rooted at the starting configuration q_{init}. At every iteration a random configuration q_{new} is generated and the closest node q_{near} in $\mathcal{T}$ is determined. By applying suitable inputs, $\mathcal{T}$ is extended from q_{near} towards q_{new}. The extension of a branch terminates when an obstacle is hit, or after a fixed integration time. By combining two trees into a bidirectional search schema, RRT is able to solve challenging motion planning problems and is practically one of the fastest planners available. One drawback is that the determined path is in no way optimal with respect to path length, clearance, or any other criteria.

For what concerns genetic algorithms for motion planning problems, Michalewicz, Xiao et al. [6][7] developed a planing/navigation algorithm for a holonomic robot moving in 2D that shows the power of evolutionary computation for motion planning tasks. The chromosomes are paths consisting of one or more line segments. Each chromosome includes the initial and goal configurations, and additional intermediate nodes. Each node consists of x and y coordinates and a Boolean state variable that indicates whether the node and the line segments initiating from that node are feasible. The number of nodes in a chromosome is randomly generated in the initialization process. The diversity of the population is sustained by eight operators. Notably, the firing probabilities of the operators are not fixed but tuned during the evolutionary process. Moreover, in the papers by Hocaoglu and Anderson [8][9] another novel approach to multidimensional path planning for holonomic mobile robots is demonstrated. The most notable characteristic of this approach is its path representation. In contrast to most path planners which use a fixed resolution, they use a multiresolution representation to cope with a variety of different environments. Not all the applications on evolutionary computation in motion planning focus on mobile robots. Nikolos et al. [10] demonstrate an evolutionary approach able to solve the planning problem for an aerial vehicle in both known and unknown environments. The path planner generates the solutions as curved paths in a 3D terrain environment by using B-Splines.

IV. The genetic algorithm

This section describes the different components needed to implement the proposed genetic algorithm. The reader is referred to [11] for a comprehensive introduction to the subject.

A. Encoding

Genetic algorithms evolve a population of candidate solutions, called *chromosomes* over time. The *encoding* process establishes how solutions are represented in chromosomes. It is important to observe that in the studied motion planning problem candidate solutions do not necessarily lie entirely in $\mathcal{C}_{free}$, but can intersect obstacles. We extended the encoding defined in [7] into a structure where a chromosome represents a path as a linked list of nodes. In genetic algorithms jargon nodes are called *genes*. Each gene includes the following information: a configuration $q = (x, y, \theta)$ and two action variables (u_s, u_ϕ). The action variables are such that if n_i and n_{i+1} are two successive nodes, then the configuration q_{i+1} associated with n_{i+1} is obtained from q_i applying the inputs stored in n_i. Finally, a boolean variable b_i carrying the information about the *feasibility* of the node is included in the structure. The feasibility of the node is determined by two factors: 1) $q \in \mathcal{C}_{free}$ 2) the path connecting the current node to the next one lies entirely in $\mathcal{C}_{free}$. A solution is said to be feasible when all its nodes are feasible, and infeasible otherwise. The resulting encoding can be seen in Figure 1.

Fig. 1. The structure of a chromosome representing a path.

B. Creation of the initial population

An initial population in which each chromosome corresponds to a path has to be created to be evolved iteratively. The paths will be recombined and mutated during the evolution process. Hence, during the evolution either the local constraints are satisfied all the time, while global constraints are abandoned, or both constraints can be ignored. Satisfying both constraints is an option, but then the algorithm would be so restrained that it could not create enough variations during the evolution. Our approach is to start with paths satisfying the nonholonomic constraints and assure that during the evolution no path violating them is ever generated. In order to create the initial chromosomes the RRT algorithm is used. This planner is well suited because it indeed generates paths complying with the nonholonomic constraints. However, in order to make the computation faster and to explore a vaster spectrum of candidate solutions, the algorithm is run without the collision checker, i.e. configurations lying in $\mathcal{C}_{obs}$ are not discarded. The planner therefore creates solutions that in general cross obstacles. In the following, whenever we say *generating a path between two points* we mean running the RRT algorithm without collision checking.

C. Genetic operators

Genetic operators are used to generate new offspring and evolve the solution population. The success of an operator mainly depends on the integration of the domain specific knowledge into the design of the operator. As stated in Michalewicz's book [11], empirical studies show that problem representations closer to the natural representation of the problem do better than the others. With this idea in mind we propose four mutation operators in addition to the one-point crossover operator, that evolve chromosomes into possibly better ones by manipulating intermediate nodes. Please note that the defined genetic operators may result in feasible or unfeasible chromosomes which eventually increase the diversity in the population. A discussion about the firing probabilities and their effectiveness will be provided in section V.

1) Crossover: The crossover operator takes two feasible or unfeasible chromosomes and produces two new ones. The two paths are cut at random *switch points* and are recombined as shown in figure 2. Since different solutions in general do not feature the same number of nodes, and due to the fact that paths must be continuous and compatible with the kinematic constraints, parts cannot be seamlessly attached to each other. Junctions are rather obtained by running the bidirectional RRT planner in $\mathcal{C}_{free}$.

2) Mutation-1: This operator is applicable to both feasible and unfeasible paths and attempts an optimization in path length. It selects a first random node from a given path.

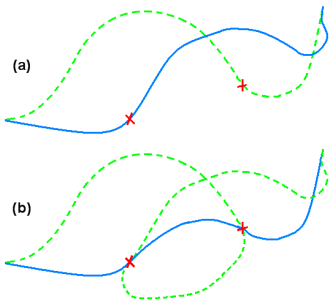

Fig. 2. (a) Two chromosomes before applying the crossover operator are shown where the crosses present the switching points. (b) The emergence of new chromosomes after the crossover. New subpaths between the crosses are obtained running the RRT algorithm.

Then, by parsing the chromosome it makes a list of genes that are located in a local neighborhood whose shape is a ball of radius ε centered on that node and depicted as region A in Figure 3. After randomly selecting a second node from this list, all the nodes in between the first and the second are deleted. An intermediate path segment is created by a bidirectional RRT connecting these two nodes. The new generated path segment replaces the old segment.

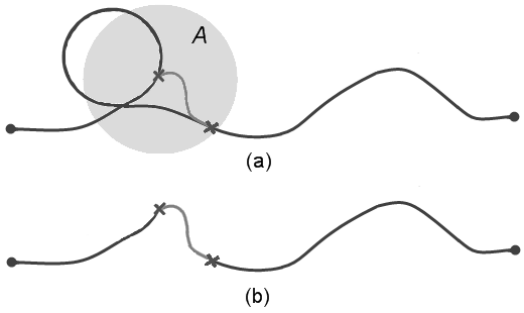

Fig. 3. (a) The upper curve is the chromosome before the application of Mutation-1. Inside the ball shaped neighborhood centered on a randomly chosen point represented by the upper cross, another point, the lower cross, is selected and a path segment connecting these two is generated. (b) The resulting chromosome is shown.

3) Mutation-2: This operator is like Mutation-1, but the location of the second node is randomly selected without being bounded to any region except the borders of the configuration space.

4) Mutation-3: This operator is applied only to chromosomes encoding unfeasible solutions. It first identifies a subpath C crossing an obstacle. Let B_1 be a set of consecutive nodes of predefined length preceding C and B_2 be a set of equal size of consecutive nodes following C. The length of B_1 and B_2 are chosen according to the dimensions of the environment and the vehicle. The operator randomly selects two genes, $p_1 \in B_1$ and $p_2 \in B_2$. Then, a random-walk starting from p_1 and consisting of a preset number of steps is generated. At each step a randomly chosen input is applied for a fixed time interval. At last a path segment is generated by running a bidirectional RRT initiating from the last point of the random-walk and ending at p_2 and concatenated to the random-walk segment. The mutation

process finishes replacing the original segment between p_1 and p_2 with the resulting path segment. The process is depicted in Figure 4.

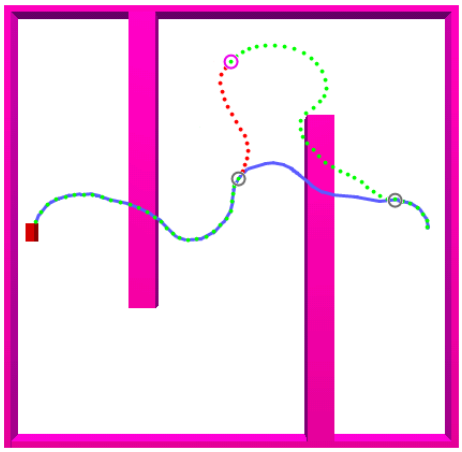

Fig. 4. A sample application of random-walk mutation. The small circle on the left is the point in the path where the random-walk starts, while the upper circle is where it ends. The final point in the random walk is then connected to the node indicated by the small circle on the right.

5) Mutation-4: The last mutation operator is also to be applied to unfeasible paths and inspired from TangentBug [12], which is a local sensor-based obstacle avoidance algorithm. Our algorithm mimics the movements of a bug by following the contours of the obstacles until it is able to move directly towards the target. It uses a virtual finite range sensor called *Antenna* instead of a contact sensor used in traditional Bug algorithms. In this mutation operator regions B_1 and B_2 are defined as for Mutation-3. In addition, the set containing the last n genes in B_1 is classified as deadzone, D_1. A symmetric definition applies for defining the set $D_2 \subset B_2$. The operator selects one gene from $B_1 \setminus D_1$ and one from $B_2 \setminus D_2$. With equal probability it then randomly decides from which one it starts. Before initiating the movement, first the Antenna is created by applying each input for a fixed number of times. After each application of each input the resulting configuration is checked for collision. Figure 5 depicts how the antenna looks like. In case of a collision, the extension of that arc is stopped. If there is no collision in any nodes of the extended antenna, then it means there is a direct visibility of the goal point in the local perspective so the robot can move towards it. In this way, after trying all possible inputs, the one that causes the robot to move toward the goal point is selected and applied. On the other hand, if a collision occurs during one of the extensions of the arcs, then the robot realizes that an obstacle is in the range. Thus, it tries to get away from it by applying the input resulting in the maximum deviation from the direction of the obstacle. In other words, if one of the left arcs of the antenna hits the obstacle, then the rightmost input is applied. If the obstacle is right ahead, then one of the extreme inputs is selected with a 0.5 probability. The movement procedure is carried out for a preset number of times. At the end of the local navigation the end point is connected to the rest of the solution by running a bidirectional RRT in between. Figure 6 illustrates an example of application.

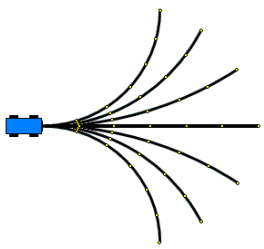

Fig. 5. A schematic representation of the *antenna* virtual sensor used to implement the TangentBug algorithm.

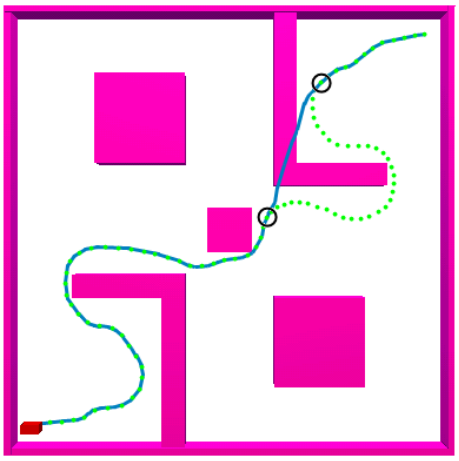

Fig. 6. A sample application of a successful Bug mutation where small circles demostrate the limits of the path segment altered by the mutation.

D. Fitness function

The role of the fitness function is to rank the chromosomes, i.e. candidate solutions to the given motion planning problem. Within the fitness function it is then possible to embed different quality criteria. The fitness of a solution is defined as the reciprocal of the *cost*, which is the weighted sum of four contributions:

$$cost(s) = \sum_{i=1}^{4} c_i cost_i(s) \tag{2}$$

Each of the four terms accounts for one quantity to be minimized. The first one computes the *path length*, i.e.

$$cost_1(s) = \sum_{i=1}^{k} RKI(u_s(i), u_\phi(i)) \tag{3}$$

where k is the number of nodes composing the given chromosome and RKI denotes the length of the arc obtained applying a Runge-Kutta integration. The second term accounts for *safety*. Due to unavoidable execution errors, it is preferable to avoid paths with small clearance from obstacles. Let $d(q)$ be the distance between the robot at configuration q

and the closest obstacle. The safety penalty at configuration q is defined as follows:

$$a(q) = \begin{cases} 0 & \text{if } d(q) \geq d_s \\ 1/d(q) & \text{else if } d_s > d(q) > d_l \\ 1/d_l & \text{otherwise} \end{cases} \quad (4)$$

where d_s and d_l are two suitable constants depending on the robot. The overall safety cost is taken by summing up the safety penalty accrued by all configurations composing a candidate solution:

$$cost_2(s) = \sum_{i=1}^{k} a(q(i)) \quad (5)$$

The third aspect is *smoothness*. Sharp turns are less preferable to smooth ones, because they imply a lower velocity in order to be executed. A cumulative cost is defined as follows:

$$cost_3(s) = \sum_{i=1}^{k} |u_\phi(i)| \quad (6)$$

Finally, since candidate solutions crossing obstacles are allowed during the evolutionary process, a *feasibility* cost is added. The penalty for infeasible solutions measures the effort necessary to make the path feasible. In order to calculate this measure, a visibility graph of the environment is created in the initialization process of genetic algorithm to be used through all the iterations. The effort is then measured by using Dijkstra's algorithm to compute the length of the shortest path in the visibility graph shortcutting a subpath crossing an obstacle. Figure 7 shows an example of the outcome of this procedure. The summation of this measure for all obstacles intersected by the path symbolizes the minimum effort to change the path in order to make it steer around the obstacle as if it was a holonomic path. The greater this cumulative measure is, the more difficult it is to make the path feasible.

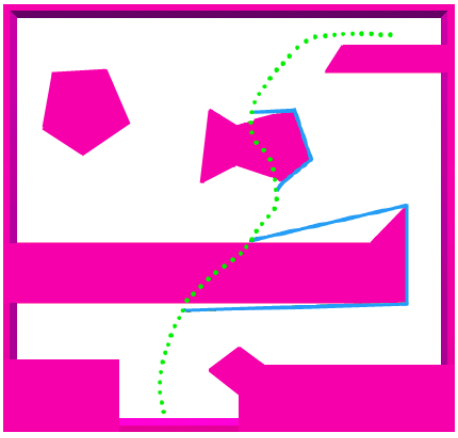

Fig. 7. The illustration of the infeasibility penalty calculation. The dashed path represents the path of the mobile robot, while the solid lines are the outcome of the visibility graph approach.

E. Termination condition

Genetic algorithms are iterative processes that need a terminating condition to stop the optimization process. After having tried different alternatives, we have found that the most effective one consists in identifying when the first feasible solution is found and then still continuing for a fixed number of iterations. An upper bound on the overall number of evolved generations is also introduced. Due to the *elitist* strategies clarified in the next subsection, the first valid solution found is unlikely to be discarded during these additional iterations, although in theory this could happen. The additional iterative cycles performed after the first solution was found aim to its further optimization.

F. The evolutionary process

We now have the elements to describe how the optimization process evolves. The algorithm initially creates a population p consisting of P elements. Every iteration evolves as follows, and starts assuming that chromosomes in p have been sorted according to their fitness. Let p' be the set consisting of the $T < P$ elements with the highest fitness (we call this process *truncation*). S chromosomes are selected from p' through a fitness based wheel selection process, and inserted in a set p''. S new chromosomes are then generated from the elements in p'' applying the genetic operators formerly described. The newly generated S chromosomes are then added to the initial population p. In order to keep the population at a constant size P, p is sorted according to the fitness and only the best P chromosomes are retained. Then a new iteration can start, unless the termination condition is met. Algorithm 1 provides the pseudocode for the overall optimization procedure. The *Select* step (line 12) implements an *elitist* strategy, i.e. it aims to retain the chromosomes exhibiting the best fitness in the next generation. Therefore, the average population fitness never decreases during the evolutionary process.

```
1:  p ← initializePopulation(P)
2:  calculateFitness(p)
3:  sort(p)
4:  while Termination Condition is NOT met do
5:      p' ← Truncate(p, T)
6:      p'' ← RouletteWheel(p', S)
7:      Crossover(p'')
8:      Mutations(p'')
9:      p''' ← p ∪ p''
10:     calculateFitness(p''')
11:     sort(p''')
12:     p ← Select(p''', P)
13: end while
14: bestSolution ← getBest(p)
15: return bestSolution
```

Algorithm 1: Algorithmic procedure

V. Experimental results

In this section we first examine the effectiveness of the proposed algorithm and contrast it with the RRT and FDP algorithms illustrated in section III. The algorithm has been coded within the *Motion Strategy Library* software [13], an open source package offering off-the-shelf implementations of the above algorithms, as well as a set of test environments. Tests were executed on a 2.2Ghz Pentium IV with 512 Mb of RAM running Linux. Displayed results for the presented genetic algorithm are the average of 100 independent runs. Due to space limitations, the whole result set is not presented. The interested reader is referred to [14] for a thorough discussion. Table I provides the default values for the parameters in the algorithm.

TABLE I
DEFAULT VALUES FOR THE PARAMETERS

P	T	S	w_1	w_2	w_3	w_4
10	8	4	0.5	0.01	0.25	5

A. Application of genetic operators

After the set p'' has been created through wheel selection from the set of fittest chromosomes, crossover and the four mutation operators are applied in order to generate the new offspring. Currently, we use fixed *firing probabilities*, i.e. the probability to apply operators is fixed throughout the entire evolutionary process. To be specific, crossover is applied with probability $p_{cross} = 0.6$, while the mutation operators have probability 0.2, 0.1, 0.1 and 0.4. These values were chosen after an extensive set of preliminary experiments. In addition to firing probabilities, the order in which genetic operators are applied is also important. We have experimented four different mutation schemas. Each of them starts by applying crossover with probability p_{cross}, but then differ in the application of mutation operators.

1) Mutation schema 1: each mutation operator is applied independently according to the given firing probabilities. The order of application is fixed (first Mutation-1, then Mutation-2 and so on).

2) Mutation schema 2: acts like schema 1, with the difference that after mutation took place, the resulting offspring replaces its input only if a fitness improvement is observed.

3) Mutation schema 3: like mutation schema 1, but the order is randomly chosen (each permutation has equal probability).

4) Mutation schema 4: in this case one operator is chosen randomly, with probability proportional to its firing probability, and then applied. Only one mutation takes place.

We have experimentally observed that the second mutation schema shows the best trade off in terms of path quality and computation time, therefore in the comparative evaluation with other motion planners this mutation schema is used.

B. Comparison with other motion planners

As stated in the introductory section, our goal is to obtain an algorithm that obtains solutions of increasing quality when more and more computation time is allotted. We have chosen to compare it with the RRT (specifically the *RRTExtExt* version) and with FDP, two algorithms offering antithetical characteristics, as outline before. For RRT we have gathered statistics in two ways. The first one runs the algorithm 100 times and computes the average path length, as well as the average executing time. The second one, indicated as *RRT-best* runs the algorithm 100 times and produces as output the best solution found. The time needed to run the algorithm 100 times is therefore charged to the best solution. For what concerns FDP, we have run it with different grid resolutions. FDP$-y$ indicates that FDP has been run on a grid obtained by dividing each dimension in y equal parts. For the proposed genetic algorithm, we run three different versions. The first one, indicated as GA-20 evolves 20 additional generations after the first feasible solution is found, while GA-50 and GA-100 continue for 50 and 100 additional generations. Figures 8 and 9 compare the path length and the execution time of the algorithms. It is evident that the proposed technique positions itself between RRT and FDP with respect to both computation time and path length.

VI. Conclusions

Inspired from work done on the integration of genetic algorithms with holonomic planning, the scientific challenge of using genetic algorithms for solving nonholonomic motion planning problems is addressed in this paper. We aimed to experimentally assess the applicability of genetic algorithms to this task, by recasting it as an optimization problem. With this purpose in mind, by incorporating domain specific knowledge into the fitness function and genetic operators, an algorithm focused on mobile robots has been developed. As presented in section V, the success of the algorithm is empirically proven on several problems in comparison to the existing planners which can solve nonholonomic motion planning problems so the target is achieved. In most of the four benchmark problems used to contrast the proposed genetic algorithm with the state of the art RRT and FDP algorithms, the algorithm shows a good tradeoff between path quality and computation time, thus reaching the goals we had fixed in the beginning. In the future we aim to extend the proposed technique to a broader class of systems, like the Reeds Shepp car, or vehicles featuring additional differential and nonholonomic constraints like bounded curvature derivatives and the alike.

References

[1] J.-P. Laumond, "Trajectories for mobile robots with kinematic and environment constraints," in *Proceedings International Conference on Intelligent Autonomous Systems*, 1986, pp. 346–354.

[2] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[4] J. Barraquand and J.-C. Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles," *Algorithmica*, vol. 10, pp. 121–155, 1993.

[5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

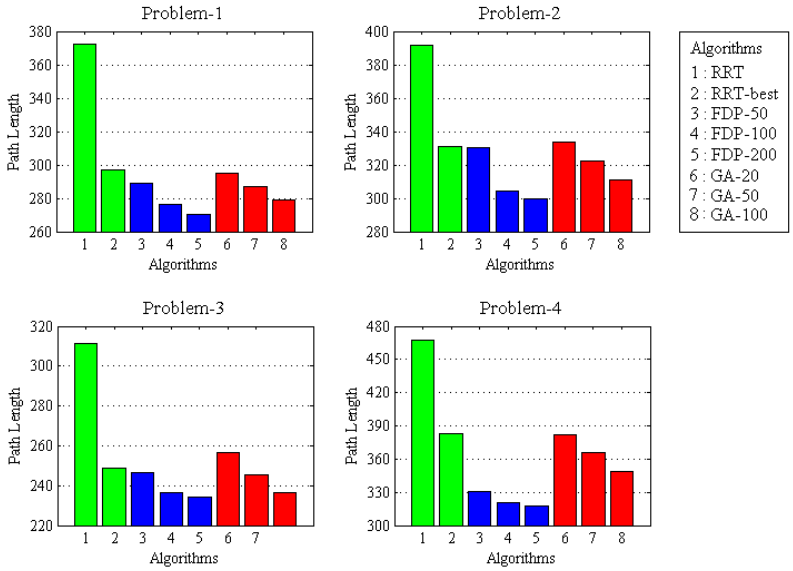

Fig. 8. Comparison of the output qualities of the existing methods with GA where the quality of a path is set to its path length.

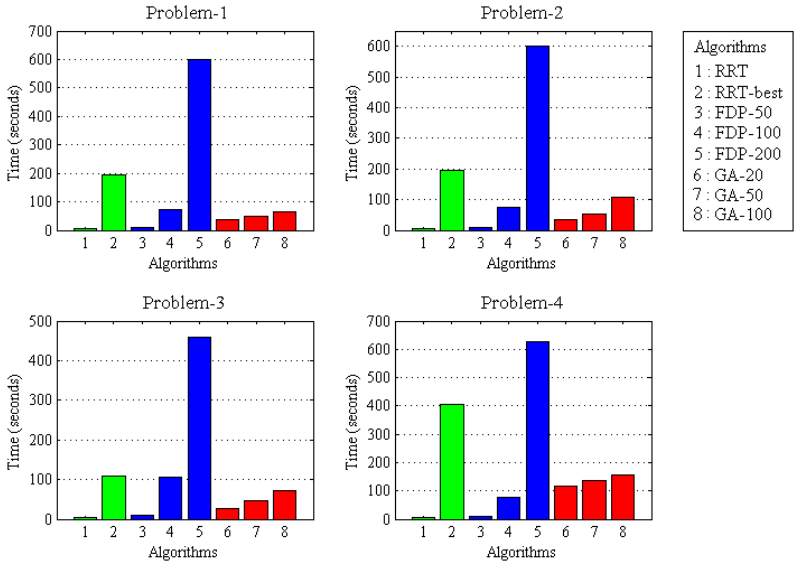

Fig. 9. Comparison of running times of all algorithms.

[6] H. S. Lin, J. Xiao, and Z. Michalewicz, "Evolutionary navigator for a mobile robot," in *Proceedings IEEE International Conference on Robotics and Automation*, 1994, pp. 2199–2204.

[7] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 18–28, April 1997.

[8] C. Hocaoglu and A. C. Sanderson, "Planning multiple paths with evolutionary speciation," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 169–191, June 2001.

[9] ——, "Multi-dimensional path planning using evolutionary computation," in *IEEE World Congress on Computational Intelligence*, May 1998, pp. 165–170.

[10] I. K. Nikolos, K. P. Valavanis, N. C. Tsourveloudis, and A. N. Kostaras, "Evolutionary algorithm based offline/online path planner for UAV navigation," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 33, no. 6, pp. 898–912, December 2003.

[11] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Springer Verlag, 1994.

[12] I. Kamon, E. Rivlin, and E. Rimon, "New range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings - IEEE International Conference on Robotics and Automation.*, vol. 1, 1996, pp. 429–435.

[13] S. LaValle, "Msl - the motion strategy library software, version 2.0," http://msl.cs.uiuc.edu.

[14] G. Erinc, "Nonholonomic motion planning with genetic algorithms for car-like robots," Master's thesis, International University Bremen, 2006.