Genetic Algorithms for Offline Path Planning

IANIS BOUGDAL-LAMBERT, 'ianisbl'
GAEL COLAS, 'colasg

This paper studies the use of a genetic algorithm (GA) for the problem of offline path planning on a 2D map. The goal is to find the optimal feasible path to move from a starting position to a target location across a flat map of a terrain with impenetrable obstacles. Maps with variable difficulties were generated. The feasible paths must be evaluated according to two criteria : their length and their difficulty. In a first approach, only the length of the paths was considered in a Single-Objective minimization process. Then the initial algorithm was extended to optimize under the two criteria (MOGA), in a Pareto optimality search. Finally, the performance of the algorithms was compared to the performance of state-of-the-art Optimal Control tools.

Additional Key Words and Phrases: genetic algorithm, path planning, multi objective

ACM Reference Format:
Ianis Bougdal-Lambert and Gael Colas. 2018. Genetic Algorithms for Offline Path Planning. 1, 1 (June 2018), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Path planning is a corner stone of Robotic Autonomy, as a robot has to be able to evolve in its environment to be considered truly autonomous. Specifically, an autonomous robot must be able to find collision-free paths to go from its initial location to a target location. Furthermore, to be called "intelligent", the robot must choose paths that are optimized according to a set of predefined metrics. A natural set of metrics for an autonomous robot could be minimizing both the travel time and the energy required to travel along the path.

In the special case of offline path planning, the map is assumed to be fixed and well known. The optimal path can then be solved offline without the need for the robot to explore the map at each step.

This project aims at using genetic algorithms in order to solve the problem of offline optimal path planning for autonomous mobile robots. The approach was inspired by an article from Castillo et al [Castillo et al. 2007].

2 PROBLEM SETUP

The problem of optimal path planning on a discrete space can be expressed as follows.

Given a discrete space described by the set of cells $\mathcal{X}$, an obstacle region described by the set of cells $\mathcal{X}_{obs}$, an initial cell x_{init} and a final cell x_{end}, the goal is to find the optimal feasible path that goes from x_{init} to x_{end}.

A path is composed of a list of consecutive cells from $\mathcal{X}$. It is defined as feasible if it stays into the space and if it does not collide with obstacles.

Authors' addresses: Ianis Bougdal-Lambert, 'ianisbl'; Gael Colas, 'colasg.

In summary, a path $\mathcal{P} = \{x_0, x_1, ..., x_N\}$ of consecutive cells is feasible if and only if :

$$\begin{cases} x_0 = x_{init} \\ x_N = x_{end} \\ \mathcal{P} \subset \mathcal{X} \\ \mathcal{P} \cap \mathcal{X}_{obs} = \emptyset \end{cases}$$

A path is considered as optimal for the single-objective (SO) version if it is the shortest feasible path ; and as optimal for the multi-objective (MO) version if it is Pareto optimal.

3 MODELING

3.1 Map representation

We define our space as a square $[0, n-1]^2$ of size n. We discretize it using regular intervals to yield the following grid :

$$\mathcal{G} = \{(i,j) | 0 \leq i,j \leq n-1\}$$

The map is represented by an occupancy grid $occ \in \mathbb{R}^{n*n}$ where:

- SO : $occ[i,j] = \begin{cases} 1 & \text{if there is an obstacle in cell } (i,j) \\ 0 & \text{otherwise} \end{cases}$
- MO : $\begin{array}{rcl} occ[i,j] & = & \text{difficulty of cell} \\ 0 \leq occ[i,j] & \leq & 1 \end{array}$

The starting point is located in the $(0,0)$ cell and the end point is located in the $(n-1, n-1)$ cell.

In the framework described in introduction, minimizing the length of the path (with the "length" being the number of cells crossed) is equivalent to minimizing the total travel time of the robot. It is assumed that crossing a cell always takes the same time. Minimizing the difficulty of the path can be seen as minimizing the "energy" consumed by the robot to cross this region.

3.2 Path representation

In the chosen representation, a path is described by $(n-1)$ pairs of direction/distance actions. Each action tells in which direction the robot moves from the current cell and how many cells are crossed along this direction

The problem was restricted to 2 different types of paths:

- x-monotone path: every action increases the x-coordinate of the path by exactly 1.
- y-monotone path: every action increases the y-coordinate of the path by exactly 1.

The monotony guarantees that we reach the final column or row after exactly (n-1) actions for x-monotone paths and y-monotone paths respectively. Using this representation, it is possible to store the path as a binary array $bina$ of fixed length.

The length of this array is :

$$m = 1 + (n-1) * (2 + 1 + 1 + log_2(n))$$

The first bit $bina[0]$ indicates whether the path is x-monotone ($bina[0] = 0$) or y-monotone ($bina[0] = 1$). The remaining bits code

Table 1. Coding of a direction/distance pair for x-monotone paths

First 2 bits	Remaining bits	Direction
00	Number of cells crossed (signed integer)	Vertical
01	Ignored	Diagonal-Up
10	Ignored	Horizontal
11	Ignored	Diagonal-Down

Table 2. Coding of a direction/distance pair for y-monotone paths

First 2 bits	Remaining bits	Direction
00	Number of cells crossed (signed integer)	Horizontal
01	Ignored	Diagonal-Right
10	Ignored	Vertical
11	Ignored	Diagonal-Left

the $(n-1)$ actions according to the correspondence explained in Table 1 for x-monotone paths, and Table 2 for y-monotone paths.

This representation is inspired from the work in [Sugihara and Smith 1999].

This choice of model presents the advantage of providing a fixed length binary correspondence with every feasible path. Furthermore, it is perfectly suited for the use of a Genetic Algorithm: specifically, mutation and crossover can be implemented exactly as explained in the book "Algorithm for Optimization" [Mykel J. Kochenderfer 2018].

4 GA ARCHITECTURE

4.1 High-level description

A high-level description of the algorithm implemented for this project is given in Algorithm 1. It follows the classic Genetic Algorithm steps, with an additional stage: the "Repair Process", specific to the category of problem presented in this report.

More details about the choices that were made for the implementation are given in the following subsections.

4.2 Initialization

We initialize a population of Path individuals of fixed size p.

$(p-2)$ of them as initialized randomly. Specifically, their binary representation is randomly generated with every bit uniformly sampled in $\{0, 1\}$.

As proposed in [Castillo et al. 2007], the last two paths are respectively x- and y-monotone straight paths. They have been represented on a occupancy grid on Figure 1 along with one random path.

For the MOGA, a empty Pareto filter is also initialized.

4.3 Selection

4.3.1 Selection strategy. The Roulette Wheel Selection method was implemented.

It is a random selection process where the probability of a parent being selected from the population is proportional to its fitness.

ALGORITHM 1: Genetic Algorithm for Path Planning

Input: Occupancy grid occ ; Number of generations n_{gen} ; Size of the population p
Output: Binary array $bina$ coding for an Optimal Feasible Path from cell $(0, 0)$ to cell $(n-1, n-1)$
Initialization population of size p;
for $i \leftarrow 1$ **to** n_{gen} **do**
  generation = $\emptyset$;
  for $k \leftarrow 1$ **to** p **do**
    $parent_1, parent_2 \leftarrow$ SELECTION(population, occ);
    $individual \leftarrow$ CROSSOVER($parent_1, parent_2$);
    $mutated \leftarrow$ MUTATION($individual$);
    $repaired \leftarrow$ REPAIR($mutated, occ$);
    ADD(generation, $repaired$);
  end
  population $\leftarrow$ REPLACE(generation);
end
return $individual_{best}$

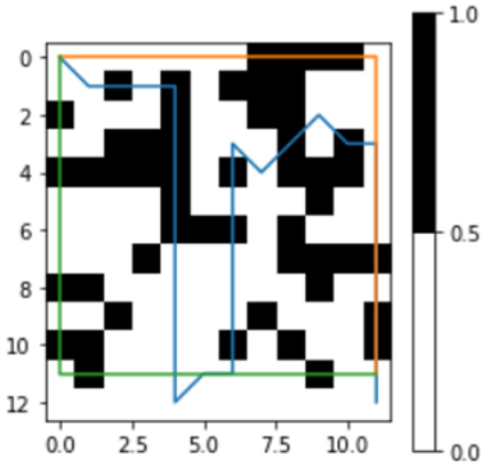

Fig. 1. Initialization of a random path (blue), a x-monotone straight path (orange) and a y-monotone straight path (green)

4.3.2 Single objective fitness. When only the length of the paths is being minimized.

The fitness of a *feasible* path x is given by :

$$f_1(x) = n^2 - n_{cell}$$

Where n_{cell} is the number of cells crossed on the path to get from the start to the end.

If the path is infeasible because it intersects with obstacles, its fitness is penalized as follows :

$$f_1(x) = \frac{n^2 - n_{cell}}{20 n_{coll}}$$

Where n_{coll} is the number of collisions with obstacles on the path.

This collision penalty allows to "stratify" the infeasible paths according to their number of collisions while still ranking them according to their lengths. It is especially useful in the first generations of the GA where there are no feasible paths yet.

Finally if the path is infeasible because it goes out of bounds or does not end in the end cell, a very low fitness is attributed to the path :

$$f_1(x) = 1$$

4.3.3 Multi objective fitness. When the difficulty of the paths is also being minimized, a second fitness function is added:

$$f_2(x) = n^2 - \sum_{(i,j)\in x} occ[i,j]$$

Where $\sum_{(i,j)\in x} occ[i,j]$ is the total difficulty of the path (the sum of the difficulties of every cell crossed along the path).

Infeasible paths are penalized like previously:

$$\begin{aligned} f_2(x) &= \frac{n^2 - \sum_{(i,j)\in x} occ[i,j]}{n_{coll}} && \text{for paths that include collisions} \\ f_2(x) &= 1 && \text{for out of bounds or non-terminated paths} \end{aligned}$$

4.4 Crossover

The Single Point Crossover method was implemented: the crossover point is chosen uniformly at random between 0 and $(m-1)$.

4.5 Mutation

Bitwise Flip mutation was implemented: each bit of an individual's binary array can be flipped (0 becomes 1 and reciprocally) with probability μ.

4.6 Repair Process

This is a step specific to this family of problems. Indeed, after one Crossover and one Mutation steps, a resulting individual is unlikely to correspond to a feasible path.

Thus an attempt is made to fix this infeasible path into a feasible one, according to the repair mechanism described by the flow chart Figure 2. The goal of this Lamarckian process is to speed up convergence.

This flow chart was presented in [Castillo et al. 2007] but specific subrepair routines were implemented for this project:

- *Out of Bounds repair* The actions that cause the path to go out of bounds are replaced by one horizontal or vertical step for x- and y-monotone path respectively ;
- *Non-Terminated repair* The last action is changed so that the path ends at the end cell. The repair process fills out the remaining cells in the last column or the last row for x- and y-monotone path respectively, in order to close the path ;
- *Collision repair* Starting from the beginning cell, the repair process tries to avoid the collision by changing the previous action. Then the next action is changed to fill the missing cell and reconnect with the rest of the path. The path is rebuilt and checked for new collisions. The repair process iterates on the collisions until they are all avoided or until the path cannot be repaired any further.

The repair mechanism looks into all the infeasible paths of the new generation and establishes the causes of their non-validity. First, it applies the "Out of Bounds" repair, because this process can cause both "Non-Terminated" and "Collision" problems. Then it fixes the "Non-Terminated" problems (this process can also cause new collisions). Finally, it tries to avoid the collisions on the paths.

Not all the paths can be completely repaired. Still, the infeasible paths are re-injected into the new population in order to maintain a fixed size population. But their fitness i penalized according to the process explained in 4.3.

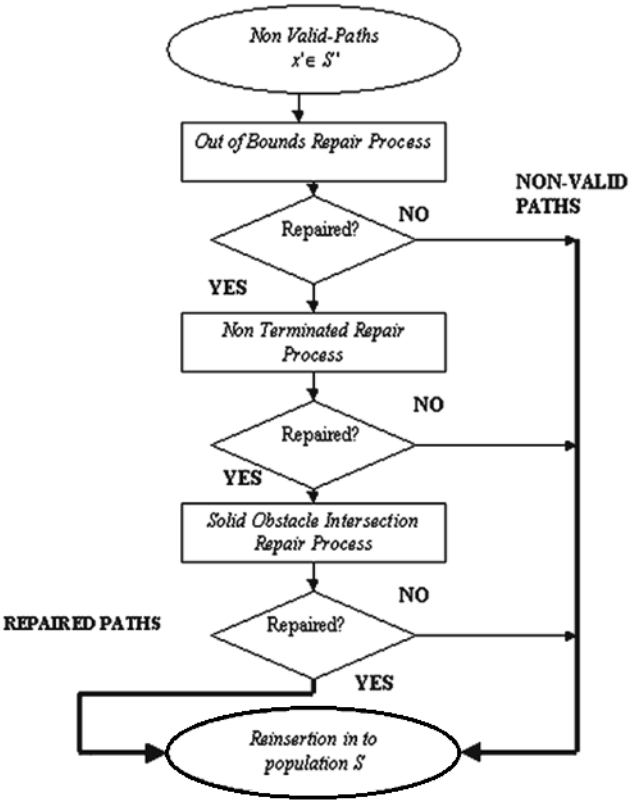

Fig. 2. Flow chart of the path repair mechanism [Castillo et al. 2007]

5 RESULTS

5.1 Single Objective

To evaluate the SOGA, its performance was compared to the results of a random algorithm. The Random Algorithm samples uniformly a new population of individuals at each generation (see 4.2). The infeasible paths are fixed using the repair mechanism, and finally the best individual seen over all the generations is returned.

The goal of these simulations was to identify if the Genetic processes were useful or if the repair mechanism was doing all the work.

Then our SOGA was compared with the state-of-the-art algorithm in Optimal Control for 2D Path Planning : A^*. A^* is an improvement to the well-known Dijkstra algorithm. It is a fast deterministic algorithm that guarantees to return the optimal path if there is one. For more information about A^*, one can refer to the lecture notes from the "AA274 : Robotic Autonomy" class[Pavone 2018]. A^* was used as a benchmark for our simulations.

Three different maps of size $n = 24$ were considered, represented on Figure 3: a "random map", a "narrow valley" map and a "maze" map. For the random map, each cell could randomly be an obstacle with probability 0.35. This probability yields complicated but feasible maps most of the time. On these maps are represented the best individuals found by the Random Algorithm, the Genetic Algorithm implemented for this project and A^* in green, blue and red respectively.

The results are averaged over 100 runs and are summarized in Table 3.

For the "random" map and the "maze" map, Figure 4 and Figure 5 display the evolution of the best individual's length with respect to the number of generations, for respectively the Random Algorithm and the Genetic Algorithm.

On Figure 4, it can be observed that the Genetic Algorithm does a better job at finding a feasible path. It needs around 5 generations to find a feasible path, whereas the Random Algorithm need around 10 random populations initializations before being able to fix an individual into a feasible path. Moreover it decreases the length of the best individual a lot quicker than the random algorithm: the GA decreases the length of the initial feasible path by 5 cells in around 20 generations where the Random Algorithm needs 150 generations. However after the first 50 generations, we can see that the GA does not improve the quality of the best individual by more than 1 cell. Hence, the remaining generations are a waste of time.

This is even clearer on Figure 5. Indeed, for the more complicated "maze" map, the Random Algorithm struggles to find a feasible path, even after 200 random population initializations. Only 20% of the simulations resulted in a feasible path. With the GA however, 85% of the simulations resulted in a feasible path and on average, the GA feasible paths were 10 cells shorter than the Random Algorithm ones.

In summary, the Genetic Algorithm tools do help to find a feasible path quicker than random sampling with the repair mechanism, while the output paths are of better quality overall. Still, the improvement is not very impressive.

As can be seen in Figure 3, the Genetic Algorithm succeeded in finding a better path than the Random Algorithm for the three different maps over the 100 runs, sometimes even the optimal one. However, it cannot compete with the A^* algorithm. As Table 3 shows, the GA finds a suboptimal path with on average as much as a 10% increase in length depending on the map. On the contrary, A^* always returns the optimal path and is not constrained to monotone paths. Moreover A^* is faster than the Genetic Algorithm. For a map of size $n = 24$, it takes around 1 minute for A^* to find the optimal solution whereas the GA need 10 minutes to run the 200 generations.

The GA suffers from the same kind of drawbacks as Grammatical evolution for expression optimization:

- It is impossible to evaluate the quality of a path from its binary representation. More specifically, it cannot be told if a individual is feasible without reconstructing the whole path ;
- The crossover step does not combine the quality of the two parents. The resulting individual can benefit from a "good" starting half given by its first parent, that is why the GA performs better than the Random Algorithm. However as the ending half given by its second parent does not begin where it initially began, most of the time this half has to be completely rebuilt by the Repair Mechanism. Hence, half of the interesting information from crossover is lost from one generation to another ;
- The mutation of an individual can either have no effect at all if it happens in a non-coding part (see Table 1), or completely change the path ;
- After one mutation and crossover step, the resulting individual is unlikely to be feasible. Most of the GA computation time is spent in the repair process where the path is reconstructed multiple times.

In summary, this path representation does not really benefit from the methods of Genetic Algorithm. Most of the resulting paths from one generation are unfeasible. And because reconstructing a path from its binary representation is expensive, this explains why the algorithm is slow.

5.2 Multi Objective

The MOGA is more relevant because it outputs an approximation of the Pareto Frontier. While the Optimal Control algorithm only outputs one optimal path for a linear combination of metrics, the MOGA gives a set of Pareto optimal paths the user can choose from.

The MOGA was tested on the map of size $n = 51$, Figure 6. This continuous map presents surfaces of variable elevations as well as peaks and holes that cannot be crossed. To run the MOGA on this map, the continuous map was pre-processed into a 2D occupancy grid. The peaks and the holes were converted into solid obstacles, then the elevation was scaled to be between 0 and 1. Finally the difficulty of a cell was computed as the average elevation over the corresponding region.

After running the MOGA, two specific paths were extracted from the Pareto filter, the "shortest" and the "easiest" one. They are plotted on the occupancy grid, respectively in red and in green.

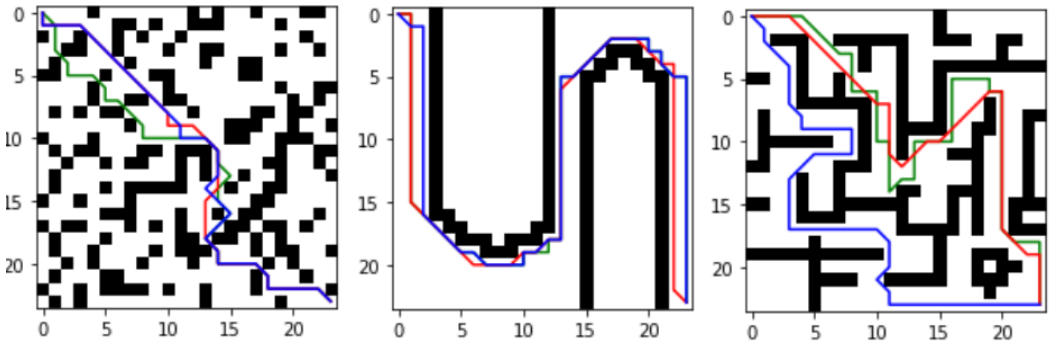

Fig. 3. The three different maps used, from left to right : "random", "narrow valley" and "maze. The best individuals returned by the Random Algorithm, the GA and A^* are plotted in green, blue and red respectively

Table 3. Comparison between the Genetic Algorithm and the Random Algorithm on the 3 different maps (in order : "random", "narrow valley" and "maze")

Algorithm	**Number of generations (n_{gen})**	**Population size (p)**	**Length (n_{cell})**	**Optimality (%)**
Random Algorithm	200	100	40.64	88
Genetic Algorithm	200	100	39.52	91
A^*	1		36	100
Algorithm	**Number of generations (n_{gen})**	**Population size (p)**	**Length (n_{cell})**	**Optimality (%)**
Random Algorithm	200	100	76.2	90
Genetic Algorithm	200	100	74.45	93
A^*	1		69	100
Algorithm	**Number of generations (n_{gen})**	**Population size (p)**	**Length (n_{cell})**	**Optimality (%)**
Random Algorithm	100	100	61.0	70
Genetic Algorithm	100	100	51.3	84
A^*	1		43	100

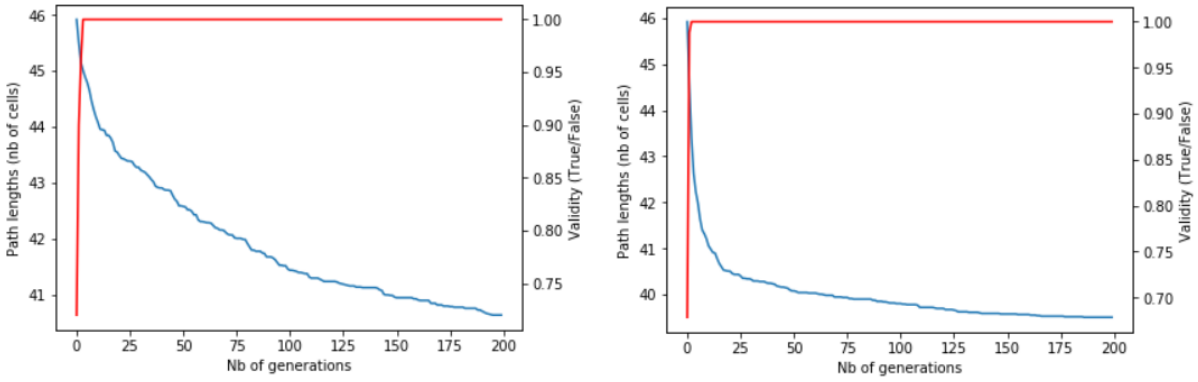

Fig. 4. Evolution on the "random" map of the best individual length (blue) and validity (red) wrt to the number of generations, for the Random Algorithm (left) and the Genetic Algorithm (right)

6 CONCLUSION

The individual representation described in [Sugihara and Smith 1999] seemed promising at first. It had the huge advantage of offering a fixed length binary representation for all the feasible paths. All the Genetic Algorithm methods could be implemented unaltered, with only the need for a Repair Mechanism. Unfortunately as it was seen in the previous section, this representation does not work well with these methods, as the crossover and mutation steps often lead to unfeasible paths. These paths are very different from their parents and need to be completely repaired. As the repair mechanism is very expensive, this leads to a slow algorithm, outperformed by combinatorial Optimal Control Algorithms. A comparison that was not made in [Castillo et al. 2007].

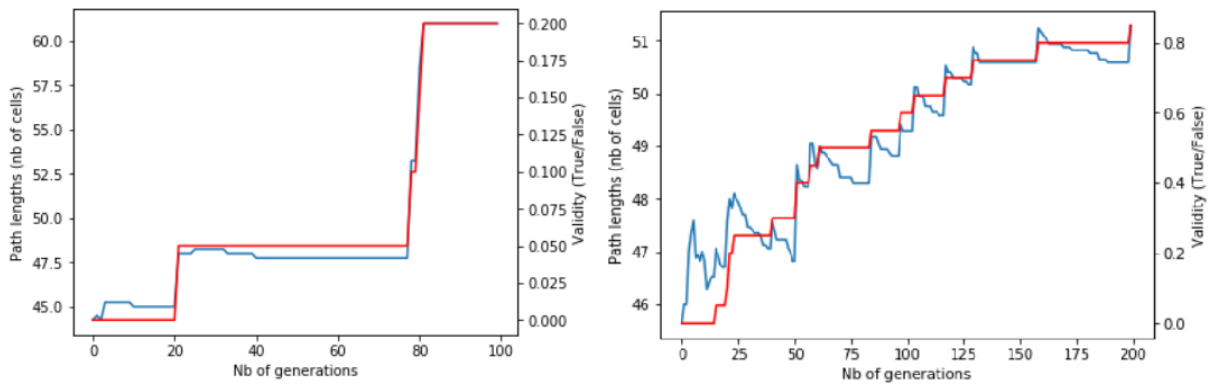

Fig. 5. Evolution on the "random" map of the best individual length (blue) and validity (red) wrt to the number of generations, for the Random Algorithm (left) and the Genetic Algorithm (right)

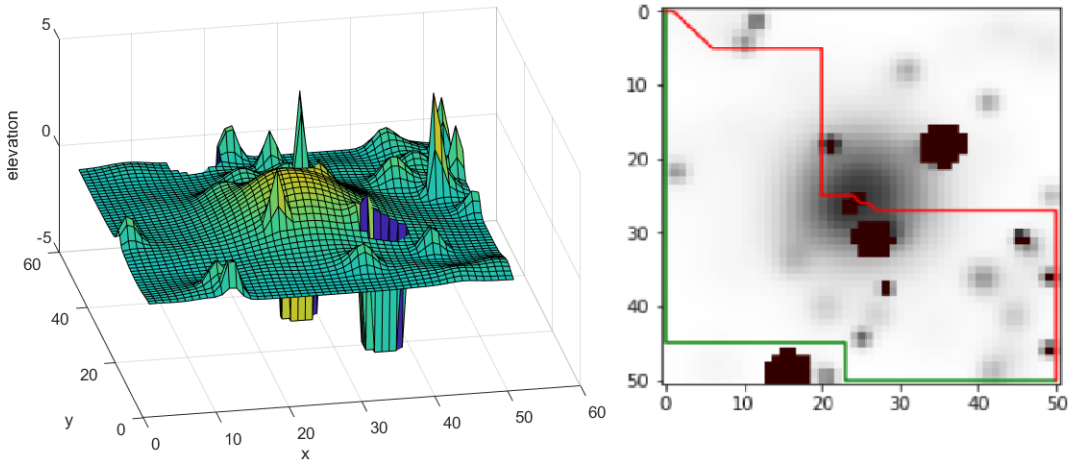

Fig. 6. Continuous 3D map of variable elevation (left) and its corresponding 2D occupancy grid (right)

The field of low-dimensional Path Planning (2D and 3D) is complete. There are already combinatorial algorithms such as A^* that are fast, reliable and are proven to be optimal. Moreover these algorithms do no need any monotony assumption, they consider the whole space of feasible paths. The Genetic Algorithm presented in this report is too slow and suboptimal to compete with these methods.

However there is still progress to be made for Path Planning in high-dimensional space, where for example A^* does not scale well in high dimensions and becomes too slow. For this category of problems, sampling methods are needed and a Genetic Algorithm could be used. However, the Genetic Algorithm implemented for this project cannot be used as is in high-dimensional spaces. Indeed, its representation is tailored for 2D maps and cannot be extended to higher dimensions without also suffering from the "curse of dimensionality". The length of the binary array would explode with the number of dimensions.

As Genetic Programming was proposed to solve expression optimization problems, there is a need for a clever individual Path representation that really benefits from the assets of Genetic Algorithm and works well in high-dimension.

REFERENCES

Oscar Castillo, Leonardo Trujillo, and Patricia Melin. 2007. Multiple Objective Genetic Algorithms for Path-planning Optimization in Autonomous Mobile Robots. *Soft Computing* 11, 3 (01 Feb 2007), 269–279. https://doi.org/10.1007/s00500-006-0068-4

Tim A. Wheeler Mykel J. Kochenderfer. 2018. *Algorithms for Optimization.* Section 5 'First-Order Methods'.

Marco Pavone. 2018. AA273 lecture slides. Lecture 15.

Kazuo Sugihara and John Smith. 1999. Genetic algorithms for adaptive planning of path and trajectory of a mobile robot in 2d terrains. (1999).

Received June 2018