UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

LUCAS DA SILVA ASSIS

CGPlan: A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm.

CGPlan: Um planejamento de rotas construtivo e escalável para agentes móveis baseado no algoritmo genético compacto.

Goiânia
2017

sistema de bibliotecas ufg

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[x] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação

Nome completo do autor: Lucas da Silva Assis

Título do trabalho: **CGPlan: A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm.**

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [x] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Lucas da Silva Assis

Assinatura do (a) autor (a)

Data: 30 / 01 / 2017

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

LUCAS DA SILVA ASSIS

CGPlan: A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm.

CGPlan: Um planejamento de rotas construtivo e escalável para agentes móveis baseado no algoritmo genético compacto.

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Programa de Pós-Graduação em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Anderson da Silva Soares

Co-Orientador: Prof. Dr. Gustavo Teodoro Laureano

Goiânia
2017

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

da Silva Assis, Lucas
CGPlan : A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm [manuscrito] : CGPlan: Um planejamento de rotas construtivo e escalável para agentes móveis baseado no algoritimo genético compacto. / Lucas da Silva Assis. - 2017.
74 f.: il.

Orientador: Prof. Dr. Anderson da Silva Soares; co-orientador Dr. Gustavo Teodoro Laureano.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2017.
Bibliografia. Apêndice.
Inclui tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Robótica Móvel. 2. Planejamento de Rotas. 3. Computação Evolutiva. 4. Algorítimo Genético Compacto. I. da Silva Soares, Anderson, orient. II. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFG

ATA Nº 02/2017

ATA DA SESSÃO DE JULGAMENTO DA DISSERTAÇÃO DE MESTRADO DE LUCAS DA SILVA ASSIS

Aos dezesseis dias do mês de fevereiro de dois mil e dezessete, às catorze horas, na sala 150 do Instituto de Informática da Universidade Federal de Goiás, Campus Samambaia, reuniu-se a banca examinadora designada na forma regimental pela Coordenação do Curso para julgar a dissertação de mestrado intitulada **"CGPlan: A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm"**, apresentada pelo aluno Lucas da Silva Assis como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência da Computação, área de concentração Ciência da Computação. A banca examinadora foi presidida pelo orientador do trabalho de dissertação, Professor Doutor Anderson da Silva Soares (INF/UFG), tendo como membros os Professores Doutores Gustavo Teodoro Laureano (INF/UFG – coorientador), Celso Gonçalves Camilo Júnior (INF/UFG) e Fernando Santos Osório (ICMC/USP). Aberta a sessão, o candidato expôs seu trabalho. Em seguida, o aluno foi arguido pelos membros da banca e:

(X) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, sem restrições.

( ) não tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **reprovação** do candidato.

Os trabalhos foram encerrados às 17:10 horas. Nos termos do Regulamento Geral dos Cursos de Pós-Graduação desta Universidade, lavrou-se a presente ata que, lida e julgada conforme, segue assinada pelos membros da banca examinadora.

Prof. Dr. Anderson da Silva Soares Anderson da Silva Soares

Prof. Dr. Gustavo Teodoro Laureano Gustavo Teodoro Laureano

Prof. Dr. Celso Gonçalves Camilo Júnior Celso Camilo

Prof. Dr. Fernando Santos Osório Fernando S. Osório

Lucas da Silva Assis

Engenheiro Eletricista pela EMC - Universidade Federal de Goiás. Durante a graduação co-fundou o Núcleo de Robótica Pequi Mecânico - UFG e tornou-se coordenador da equipe de futebol de robôs (categoria IEEE Very Small Size Soccer). Durante o Mestrado foi bolsista CAPES e continuou a pesquisa com a temática de futebol de robôs, com o foco na movimentação e geração de trajetórias para agentes móveis.

Este trabalho é dedicado às várias crianças adultas que, quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Agradeço primeiramente a Deus pela minha vida, pelas oportunidades e por permitir que meus sonhos se concretizassem.

Agradeço também ao meu orientador Dr. Anderson Soares e ao meu coorientador Dr. Gustavo Teodoro pela confiança, pelo conhecimento compartilhado e pela indispensável orientação no desenvolvimento deste trabalho. Agradeço também à CAPES, pela bolsa de mestrado concedida que permitiu a execução desta pesquisa.

Agradeço a toda a minha família, especialmente a meus pais, Orlando Dionízio e Sávia Cristina pela educação, amor e incentivo durante toda a minha vida, o seu apoio e a ajuda tornaram possível a realização desse trabalho.

Agradeço minha namorada Ludmille, melhor amiga e companheira de todas as horas, pelo carinho e confiança, pela compreensão nos momentos de dificuldade, pelos momentos felizes que vivemos juntos e por ter sempre me apoiado e motivado a atingir meus objetivos.

E por último, mas não menos importante, agradeço à todos os amigos e colegas de trabalho do Núcleo de Robótica Pequi Mecânico da UFG pela inspiração e constante incentivo para desempenhar o meu melhor nesse trabalho. Agradeço especialmente aos amigos Vinícius Oliveira e Vinícius Araújo pelo companheirismo e ajuda nos diversos desafios durante esta jornada no mestrado.

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

Alan Turing,
Computing machinery and intelligence.

Resumo

ASSIS, LUCAS DA SILVA. **CGPlan: A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm.**. Goiânia, 2017. 74p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

O planejamento de rotas é um recurso importante para agentes móveis, permitindo-lhes encontrar caminhos ideais entre os pontos desejados. Neste contexto, caminhos ideais podem ser entendidos como trajetórias que melhor atingem um objetivo, minimizando a distância percorrida ou o tempo gasto, por exemplo. As técnicas tradicionais tendem a considerar um modelo global do ambiente, no entanto, os problemas reais de planejamento de rotas usualmente estão no âmbito de ambientes desconhecidos ou parcialmente desconhecidos. Portanto, aplicações como essas geralmente são restritas a abordagens subótimas que planejam um caminho inicial baseado em informações conhecidas e, em seguida, modificam o caminho localmente ou até planejando novamente todo o caminho à medida que o agente descobre novos obstáculos ou características do ambiente. Sendo assim, mesmo as estratégias tradicionais de planejamento de caminhos sendo amplamente utilizadas em ambientes parcialmente conhecidos, suas soluções subótimas se tornam ainda piores quando o tamanho ou a resolução da representação do ambiente aumentam.
Por isso, neste trabalho apresentamos o CGPlan (Constructive Genetic Planning), uma nova abordagem evolutiva baseada no Algoritmo Genético Compacto (cGA) que almeja um planejamento eficiente de caminho em ambientes conhecidos e desconhecidos. O CGPlan foi avaliado em ambientes simulados com crescente complexidade e comparado a técnicas comuns utilizadas para o planejamento do caminho, como o A*, o algoritmo BUG2, o RRT (Rapidly-Exploring Random Tree) e o planejamento evolutivo do caminho usando clássico Algoritmo Genético. Os resultados mostraram uma grande eficiência da proposta e indicam uma nova abordagem confiável para o planejamento de rotas de agentes móveis com poder computacional limitado e restrições em tempo real no hardware.

Palavras–chave

Planejamento de Rotas Local, Algoritmo Genético Compacto, Agentes Móveis.

Abstract

ASSIS, LUCAS DA SILVA. **CGPlan: A Scalable Constructive Path Planning for Mobile Agents based on the Compact Genetic Algorithm.**. Goiânia, 2017. 74p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Path Planning is an important feature for autonomous mobile agents, allowing them to find optimal paths between desired points. These optimal paths can be understood as trajectories that best achieves an objective, e.g. minimizing the distance travelled or the time spent. Most of usual path planning techniques assumes a complete and accurate environment model to generate optimal paths. But many of the real world problems are in the scope of Local Path Planning, i.e. working with partially known or unknown environments. Therefore, these applications are usually restricted to sub-optimal approaches which plan an initial path based on known information and then modifying the path locally or re-planning the entire path as the agent discovers new obstacles or environment features. Even though traditional path planning strategies have been widely used in partially known environments, their sub-optimal solutions becomes even worse when the size or resolution of the environment's representation scale up.
Thus, in this work we present the CGPlan (Constructive Genetic Planning), a new evolutionary approach based on the Compact Genetic Algorithm (cGA) that pursue efficient path planning in known and unknown environments. The CGPlan was evaluated in simulated environments with increasing complexity and compared with common techniques used for path planning, such as the A^*, the BUG2 algorithm, the RRT (Rapidly-Exploring Random Tree) and the evolutionary path planning based on classic Genetic Algorithm. The results shown a great efficient of the proposal and thus indicate a new reliable approach for path planning of mobile agents with limited computational power and real-time constraints on on-board hardware.

Keywords

Local Path Planning, Compact Genetic Algorithm, Mobile Agents.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Contextualization

Machines that can operate autonomously is a concept that dates back to classical times, but research into the functionality and potential uses of robots did not grow substantially until the 20th century. Product of these researches, the first modern robots appeared in industrial manufacturing – as simple fixed machines capable of manufacturing tasks which allowed production without the need for human assistance. Robotics has also achieved its first economic success in this field, as robot arms, or manipulators, became a 2 billion dollar industry in the early 2000s. Since this, robotic applications have rapidly grown from purely industrial manufacturing to service and household assistants, reaching 3.6 million units of service robots and 1.6 million units of industrial robots worldwide, with a total market value of 11.1 billion dollars in 2015 [23].

Even before the advent of affordable mobile robots, the path planning field was heavily explored due to its applicability in the area of industrial manipulator robotics. This fundamental robotic task was first used as a means of planning collision-free movements for complex bodies from a beginning to a goal position among a collection of static obstacles.

Static at a specific position in the assembly line, an industrial manipulator can use path planning techniques to move with great speed and accuracy in order to perform repetitive tasks such as spot welding and painting. In the electronics industry, manipulators are also used to place surface-mounted components with superhuman precision, making modern smartphones and laptop computers possible.

Although a relatively simple approach, the path planning problem turned out to be computationally hard, becoming even worse when the environment's map is scaled up to represent larger scenarios [28]. Modern algorithms have been fairly successful in addressing hard instances of the basic geometric problem and a lot of effort is devoted to extend their capabilities to more challenging instances, in which computational complexity and map scalability continues to be great challenges to overcome.

Interestingly, the path planning problem for a manipulator is usually far more complex than that of a common mobile robot operating in a simple environment. Therefore, although inspired by the earlier techniques developed for manipulators, the path planning algorithms used by mobile robots tend to be simpler approximations owing to the greatly reduced degrees of freedom. Furthermore, industrial robots often operate at the fastest possible speed because of the economic impact of high throughput on a factory line. So, the dynamics and not just the kinematics of their motions are significant, further complicating path planning and execution. In contrast, a number of mobile robots operate at such low speeds that dynamics are rarely considered during path planning, further simplifying the mobile robot instantiation of the problem.

1.2 Motivation

In the upcoming subsections the topics that inspired the development of this work are discussed.

1.2.1 Path planning efficiency.

Path planning efficiency for autonomous navigation is still an open challenge in robotics. In the last decades, the efforts to reach an intelligent motion behavior lead to various strategies both in literature and industry: the deterministic solution, based on heuristic graph search; the reactive solution, based on environment sensor data; and the sampling-based solution, representing the solutions to the path planning with a road map of sampled configurations [5]. Recently, new evolutionary approaches have also reached the spotlight, proving to be competitive with the already renowned solutions [18] [20].

Unfortunately, the main complications concerning path planning efficiency are the difficulty to directly compute C_{obs} and C_{free} from a real environment, as the dimensionality of a real C-space often reaches high values. In terms of computational complexity, the path planning problem of a rigid body in a 2D environment was shown to be PSPACE-hard by Reif [28], and a series of polynomial time algorithms for problems with fixed dimension suggested an exponential scalability with the dimensionality of the C-space [31] [32].

Certain path planning problems can become even harder, for example, planning under uncertainty in a 3D polyhedral environment was shown to be NEXPTIME-hard [4], as the hardest problems in NEXPTIME are believed to require doubly-exponential time to solve. In this work the term scalability refers to how a certain technique can handle the increase in dimensionality or size of the C-space.

1.2.2 Planning and reacting behaviours.

Planning and reacting are often viewed as contrary behaviours or even opposites by the artificial intelligence community [34]. However, when applied to mobile agents, combining these two behaviours can greatly enhance the agent's ability to navigate in real environments.

The navigation problem is usually solved by executing a previously established course of action (or plan) in order to reach its goal position. Unfortunately, during execution the agent can encounter unforeseen events (e.g., obstacles or other agents), forcing it to react in such a way as to still reach the goal. Without reacting, the planning effort will likely be incomplete, as the robot will diverge from its path and could never physically reach its goal. Without planning, the reacting effort usually cannot solely guide the robot through the environment, possibly never reaching the desired goal.

1.2.3 Planning with limited knowledge.

The autonomous navigation of a mobile agent relies heavily on its ability to balance planning and reacting behaviours. In some cases, the agent has a complete knowledge of the environment and can solely plan its path based on it, not needing a reactive behavior. Unfortunately, in most applications the only available information cover the task's goal and little or no information are available about the environment, forcing most real-world mobile agents to gather information about the environment and react while moving.

Traditional approaches to the path planning problem tend to struggle with this lack of knowledge, compelling applications with limited information to resort on sub-optimal solutions. This problem becomes even worse when a path is planned entirely before navigation begin, disregarding the possibility of unforeseen obstacles. This approach lead to expensive computational cost when reaction is needed, as modifications to the original path must be made.

1.2.4 The Constructive Genetic Planning - CGPlan.

Based on the topics discussed above, an efficient and scalable approach is desirable when dealing with restricted information, especially when a higher resolution or dimension in the representation of the environment is required. Within the context of mobile agents, the planning also needs to be computationally viable and be completed within a reasonable time frame, since the agent must often perform its task with a low performance on-board equipment.

With that in mind, the reacting and planning behaviour are merged in this work, creating a technique that plan short trajectories aiming at the goal while still reacting to the local environment around the agent. Aiming at a relative inexpensive optimization, these trajectories are found using a Compact Genetic Algorithm search, and the collection of these short trajectories forms, in a constructive fashion, the final path between the starting and goal position, hence the name Constructive Genetic Planning or CGPlan in short.

This technique is especially designed to navigate in unknown environments but it is competitive with traditional planning strategies in completely known environments. Most importantly, it can also achieve the standards listed above, generating optimal or near-optimal paths with low resources consumption using high resolution representations of the environment.

1.3 The organization of this work.

We start by clarifying the path planning problem and its conventional approaches in chapter 2. We then discuss the choice of a compact genetic algorithm for efficient optimization, eventually reaching the proposed planning strategy in chapter 3. In this chapter we then provide a full description on the path construction, the evaluation process and the solution to the local minimum used in our proposal. Furthermore in chapter 4, computer simulations are used to compare the proposed technique with four well-stabilised path planning approaches, addressing the differences and similarities both in terms of solution quality and performance. Finally, experiments with scalability show the behaviour of each strategy when exposed to increasing on the resolution of the environment's representation.

CHAPTER 2

Path Planning

This chapter starts with the definition behind the path planning problem and its various classifications in section 2.1, ranging from available knowledge from the environment through the choice of its representation. In section 2.2 a discussion on the related works in the literature is present, focusing on the four techniques chosen to compose a comparison group for the proposed CGPlan.

2.1 The Path Planning Problem

Path planning is an important task for autonomous mobile agents that lets them find a path between points, obeying restrictions imposed by environment and requirements from the application [16]. These paths can achieve any specific requirement, therefore, algorithms designed to find these paths are important not only in robotics, but also in network routing, video games, and gene sequencing. Path planning applications are also often placed in two categories based on the previous knowledge of the environment [33]. Global path planning, also known as offline path planning, occurs in environments where complete information about stationary obstacles and moving obstacles is known in advance. When this option is not available, the mobile agent gathers information as it moves through the environment. This is known as online or local path planning.

2.1.1 Configuration Space

Modern path planning of manipulator robots, and even for most mobile robots, is usually done using a representation of the workspace called configuration space. Let A be a complete description of the geometry of a robot and W be a workspace populated with obstacles, with $W = \mathbb{R}^N$, in which $N = 2$ or $N = 3$. The configuration space, or C-space, is defined as the set of all possible configurations, representing all transformations that can be applied to a robot given its kinematics.

The path planning goal is to find a collision-free path for A to move from an initial position and orientation to a goal position and orientation. To achieve that, a complete

specification of the location of every point on the robot geometry, i.e a configuration q, must be provided. Supposing that A has k degrees of freedom, every state or configuration possible of A can be described with k real values: $q_1, q_2, \ldots, q_k$.

Likewise, the closed set $A(q) \subset W$ denote the set of points that can be occupied by the robot when at configuration $q \in C$, and the closed set $O \subset W$ is used to represent the obstacle points in the workspace. The subspace solely composed by obstacles in the C-space, C_{obs}, can be defined as expressed in equation 2-1.

$$C_{obs} = \{q \in C \mid A(q) \cap O \neq \emptyset\}[16] \tag{2-1}$$

Since O and $A(q)$ are closed sets in W, the obstacle region is a closed set in C. Therefore, the set of configurations that avoid collision can be defined as $C_{free} = C \setminus C_{obs}$, and is called the free space. This representation is a useful approach to abstract planning problems in a unified way, mapping a robot with complex kinematics in the workspace to a single k-dimensional point in the C-space. Several common robots and their respective C-Space representation can be seen in table 2.1.

However, in common mobile applications the robot can often be simplified to a single point in the workspace, reducing the configuration space to a 2D representation with just x and y axes. This simplification make the configuration space looks essentially identical to a 2D version of the workspace, with obstacles inflated by the robot's radius. In this work, the path planning is treated as a general problem to be solved in the configuration space described above, disconnected from the kinematics of the robot that will perform them. Therefore, the robot is also referred to with a general term - the agent.

Table 2.1: *Some common robot configurations and their respective C-Space representation. [14]*

Type of Robot	C-Space Representation
Mobile robot translating in the plane.	$\mathbb{R}^2$
Mobile robot translating and rotating in the plane.	$SE(2)$ **or** $\mathbb{R}^2 \times S^1$
Rigid body translating in the three-space.	$\mathbb{R}^3$
A Spacecraft.	$SE(3)$ **or** $\mathbb{R}^3 \times SO(3)$
An n-joint revolute arm.	T^n
A planar mobile robot with an attached n-joint arm	$SE(2) \times T^n$

Note that in table 2.1 the notations $SO(n)$, $SE(n)$ and S^1, represents the set of all $n \times n$ rotation matrices, the set of all $n \times n$ homogeneous transformation matrices and the set of all 2D joint angles respectively. In the other hand, the notation T^n represents the set of all 2D joint angles, S^1, multiplied n times (e.g $T^2 = S^1 \times S^1$).

2.1.2 Environment representation

In addition to knowledge of the environment, the choice of environment representation (i.e the map), is also crucial to solve the path planning problem. The representation impacts directly on the performance and resources consumption of path planning methods, since it is responsible for the processing and storage of information collected from the environment. Hence, three fundamental relationships must be understood when choosing a particular map [34]:

1. The size of the map must appropriately match the precision with which the agent needs to achieve its goals.
2. The type of the map and the features represented must match the precision and data types returned by the agent's sensors.
3. And more importantly for low performance agents, the resolution of the map representation has direct impact on the computational complexity of reasoning about mapping, localization, and navigation.

Reasoning with these relationships, two main approaches to environment representation emerged over time: continuous and discrete.

- The continuous approximation is one method for decomposition and representation of the environment. The position of environmental features (e.g, obstacles, doors, etc.) can be marked in continuous space. Generally, mobile agent implementations use continuous maps only in 2D representations, as further increase in dimensionality can result in high computational complexity, as seen in subsection 1.2.1. A common approach is to combine the exactness of a continuous representation with the compactness of the closed-world assumption. This means that the representation will specify all objects in the environment, and that any area in the environment that is devoid of objects will not be represented. Thus, the total storage needed in this representation is proportional to the density of objects in the environment, and a sparse environment can be represented by a low-memory structure. In summary, the continuous map representation has the potential for high accuracy and expressiveness of the environment as well as the agent position within that environment but the map can be computationally costly if not sparse [34].
- In a discrete approximation, the environment is sub-divided into equal areas (e.g., a grid or hexagonal) or differing areas (e.g., rooms in a building). Discrete maps are know to perfectly fit a graph representation, where the every area of the environment corresponds to a vertex (also known as a node), which is connected by edges, so an agent can navigate by traversing the graph. Computationally, this graph can be stored as an adjacency, incidence list, or matrix, being a very inexpensive

representation, growing with the size and resolution of the sub-divisions of the environment. However, if a combination of high resolution and large environment is present, this approach can quickly become untenable [34]. Assuming a discrete approximation, an optimal path planning solution could be the *shortest* path from one vertex to another through a connected graph. The term *shortest* here refers to the minimum cumulative edge cost, which could be physical distance, time spent on travel, or any other important metric for a particular application.

2.2 Related Works

Various possible strategies for local and global path planning have been used effectively [22]. Classic techniques such as A^* [7], the Bug2 algorithm [3], RRT [15] and different heuristic approaches using well-known techniques such as particle swarm optimization (PSO) [29], artificial neural networks (ANN) [26], fuzzy logic (FL) [1], and genetic algorithms (GA) [13] have been described as effective and reliable for path planning. Among these, four well renowned techniques were chosen to serve as comparison to the proposed CGPlan. A brief relation on the main characteristics of each technique can be found in table 2.2 and a detailed description to these techniques can be found in subsections 2.2.1 through 2.2.4

Table 2.2: *Comparison on the chosen techniques.*

	Technique Features	Known Advantages	Known Disadvantages
A*	• Global path planning. • Heuristic distance consideration.	• Optimal path generation. • Simple implementation.	• Restricted trajectories within grids or graphs. • Bad scalability with map resolution.
Bug2	• Local path planning. • Deterministic path planning.	• Low resource consumption. • Very simple implementation.	• Can output an inefficient path in some obstacle configurations.
RRT [12]	• Global path planning. • Auxiliary path smoothing technique.	• Efficient search of non-convex spaces. • Reasonable resource management.	• Struggles with narrow passages. • Optimal path is not guaranteed.
GA [11]	• Global path planning. • Fixed-length binary chromosome. • Basic genetic operators.	• Can solve complex environments. • Near-optimal path generation.	• Very time consuming. • Bad scalability with map resolution.

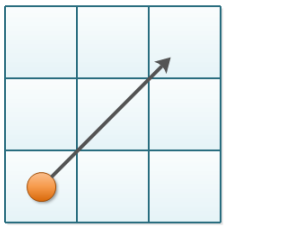
2.2.1 The A* Path Planning

One of the earliest and simplest path planning algorithms is known as the Dijkstra's algorithm. Starting from the initial vertex of a graph, the algorithm marks all direct neighbors with its respective distance cost. It then proceeds from the neighbor with the lowest cost, repeating the process to all of its adjacent vertexes. Once the algorithm reaches the goal, it terminates and the agent can follow the edges pointing towards the lowest edge cost. One well known improvement to the Dijkstra's algorithm can be done by adding an estimated cost of movement, usually with a heuristic distance function. This improved algorithm is known as A* path planning. The A* is a best-first search, with each cell being evaluated by the fitness function:

$$F(v) = H(v) + G(v) \tag{2-2}$$

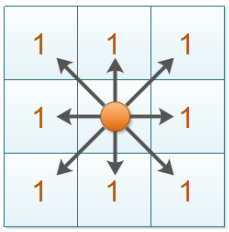
where H(v) is any heuristic distance function, e.g. Manhattan, Euclidean or Chebyshev of a cell to the goal state and G(v) is the distance cost, a representation of the path's length from the initial state to the current cell. The algorithm proceeds to evaluate each adjacent cell of its current state and chooses the cell with the lowest value of F(v) as the next one in the path sequence.

The heuristic function H(v) must be a computationally easy estimate of the distance between each node and the goal, as it will be calculated at least once for every node before reaching the goal. The implementation of the H score can vary depending on the properties of the graph being searched, with the most common being the heuristics illustrated in figure 2.1:



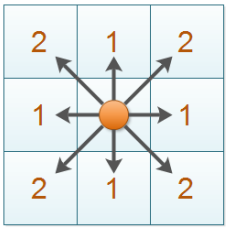
$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

(a) The Euclidean distance function.



$\max(|x_1 - x_2|, |y_1 - y_2|)$

(b) The Chebyshev distance function.



$|x_1 - x_2| + |y_1 - y_2|$

(c) The Manhattan distance function.

Figure 2.1: *Common distance heuristics used in the A**

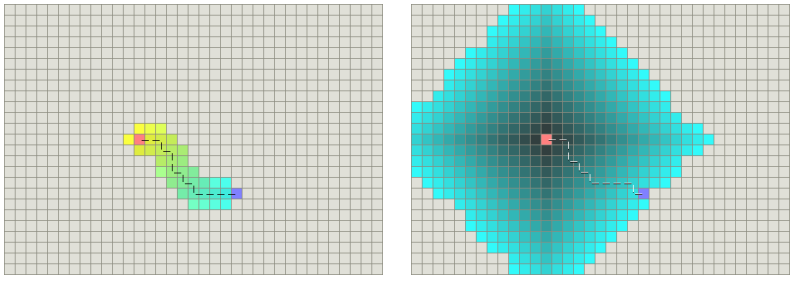
The classic implementation of the A* path planning uses two lists to manage the search between nodes. The open list, usually contains nodes that have been visited but not yet expanded (meaning that its successors have not been explored yet). This is also known as the list of pending tasks. The closed list, however, consists on nodes that have been visited and expanded (successors have been explored already and included in the open list, if needed). The pseudo-code showed in 2.1 further explains the use of these two data structures in the implementation of the A*.

Algorithm 2.1: Pseudo-code of the A*

Let the goal node be denoted by *node_goal*, the source node be denoted by *node_start*, the open list be denoted by *OPEN* and the closed list be denoted by *CLOSED*.

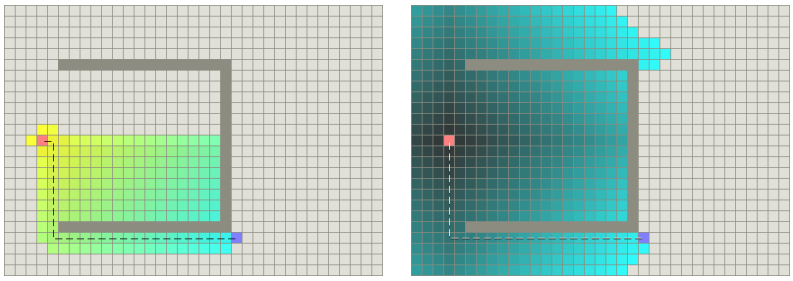
```
/* Push the node_start in the OPEN list                                  */
OPEN ← node_start
/* While the open list is not empty                                      */
while !OPEN.empty() do
    /* Take the node with the lowest fitness from the open list          */
    node_current = OPEN.pull_lowest_node()
    CLOSED ← node_current
    node_current_fitness = H(node_start, node_current) + G(node_current)
    /* Check if node_current is the goal node                            */
    if node_current = node_goal then
        return reconstruct_path(PATH)
    end
    /* For each successor node do the following                          */
    for i=1 to node_current.get_successor_total() do
        /* Get the successor fitness                                     */
        node_successor = node_current.get_successor_node(i)
        node_successor_fitness =
          G(node_current) + H(node_current, node_successor)
        /* Do not evaluated previously expanded nodes.                   */
        if !CLOSED.contains(node_successor) then
            if !OPEN.contains(node_successor) then
                /* Push the node_successor in the OPEN list for
                   further expansion                                     */
                OPEN ← [node_successor , node_successor_fitness]
            end
            if node_successor_fitness <= G(node_successor) then
                /* This path is the best until now. Record it.           */
                PATH ← [node_current, node_successor]
            end
        end
    end
end
```

An illustration on the solution to planning in different environments obtained with the A* and Dijkstra's algorithm can be seen in figure 2.2. In this figure, the pink square represents the starting point, while the blue square represents the goal for the path planning. Sub-figures (b) and (d) shows the Dijkstra's solution to two different environments, while the teal color shows the ones explored by the algorithm with lighter tones being the most recent. In the other hand, the sub-figures (a) and (c) shows the A*'s solution for the same environments with the yellow representing the heuristic distance function and teal representing the distance cost for a given tile.



(a) A*'s solution to a simple planning problem.

(b) Dijkstra's solution to a simple planning problem.



(c) A* solution to a planning problem with obstacles.

(d) Dijkstra's solution to a planning problem with obstacles

Figure 2.2: *An Illustration on the A* and Dijkstra solutions to different path planning problems.[25]*

In summary, the A* is a discrete global approach that combines the advantages of uniform-cost and greedy searches using a fitness function [7]. The A* path planning is also easily modifiable, as different task requirements can be added as heuristic functions to the fitness, i.e. energy consumption or path safety. However, the computational time can scale exponentially with increase in the number of cells to be processed [24]. The A* was chosen for comparisons due to its well known solution quality and robustness, been a good parallel to the proposed CGPlan.

2.2.2 The Bug Path Planning

Bug is a family of algorithms that solve planning problems using ideas that are related in many ways to the maze exploration. This class of algorithms is based on the assumption that the agent is placed into an unknown 2D environment that may contain any finite number of bounded obstacles. These obstacles could be composed by polygons, smooth curves, or any combination of curved and linear parts [16].

The agent must know its position and orientation and should be able to perform two simple types of sensing, these being :

- A goal sensing, indicating the current Euclidean distance to the goal and the relative direction to the goal.
- A local visibility sensing, providing the shape and distance from obstacles within a small radius from the robot.

The goal sensing encodes the robot and goal position in a polar coordinate system in which the goal is the origin. Therefore, unique coordinates can be assigned to any position already visited. This feature allows the agent to incrementally trace out obstacle boundaries that it has already traversed. In the other hand, the local visibility sensing provides just enough information to allow a wall-following behaviour; as the range of the sensing is usually very short implying that the robot cannot learn any other information about the environment.

These path planning approaches requires minimum sensing input and have a simple algorithm that circumnavigate obstacles in order to reach the goal, being able to work in global or local planning applications. Currently, there are many types of Bug algorithms, the most commonly used and referred to in mobile robot path planning being the Bug1 and Bug2, DistBug, VisBug, and TangentBug [3]. The DistBug, VisBug, and TangentBug are the later evolutions to this family of algorithms, following the same strategy as the Bug2 and incorporating range sensors to the usual Bug sensing configuration in order to improve its solutions. Despite their differences, the variations of Bug algorithms show the same effort toward shorter path planning, shorter timing, simpler algorithms, and better performance.

The Bug1 strategy

The Bug1 algorithm was proposed by [19] as a simple and effective solution to path planning problem. It is the first Bug's family algorithm and its solution to the planning problem is illustrated in figure 2.3. Its execution usually follows as described below:

1. Move toward the goal until an obstacle or the goal is encountered. If the goal is reached, then stop.
2. Turn left and follow the entire perimeter of the contacted obstacle. Once the full perimeter has been visited, then return to the point at which the goal was closest, and go to Step 1.

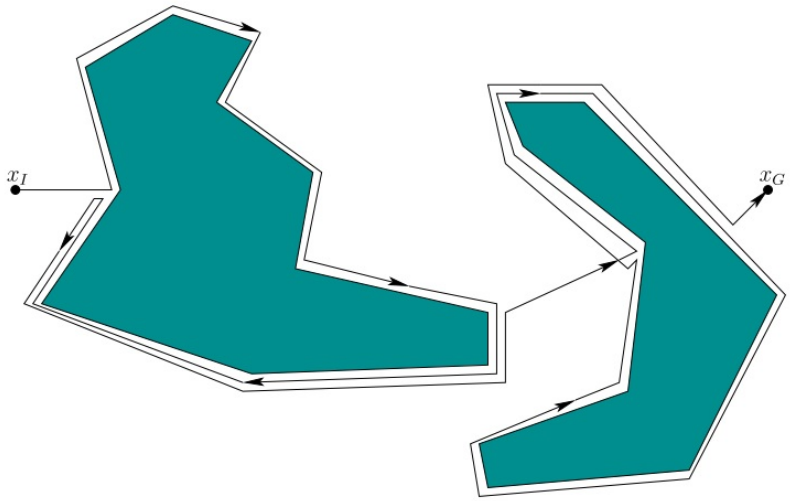


Figure 2.3: *An Illustration of the Bug1 Algorithm. [16]*

In this strategy, finding a point at which the goal sensing repeats its output indicates that the entire perimeter has been traversed. The worst case is conceptually simple to understand, as the total distance travelled by the robot is no greater than D_{max} , as expressed in equation 2-3;

$$D_{max} = d + (3/2)\sum_{i=1}^{M} p_i \tag{2-3}$$

In which d is the Euclidean distance from the initial position to the goal position, p is the perimeter of the i^{th} obstacle, and M is the number of obstacles. This means that the boundary of each obstacle is followed no more than 3/2 times. This bound relies on the fact that the robot can always recall the shortest path from which it needs to leave.

The Bug2 strategy

The Bug2 is an evolution, to some extent, from the Bug1. In this strategy, the agent always attempts to move along a line that connects the initial (x_I) and goal positions (x_G), this line is also referred as m-line. The execution of this strategy usually follows as described bellow:

1. Move toward the goal until an obstacle or the goal is encountered. If the goal is reached, then stop.
2. Follow the perimeter of an obstacle until the m-line is reached. Once the m-line is reached, then move towards the goal and return to Step 1.

The Bug2 solution to the same planning problem solved by Bug1 in figure 2.3 is illustrated in figure 2.4.

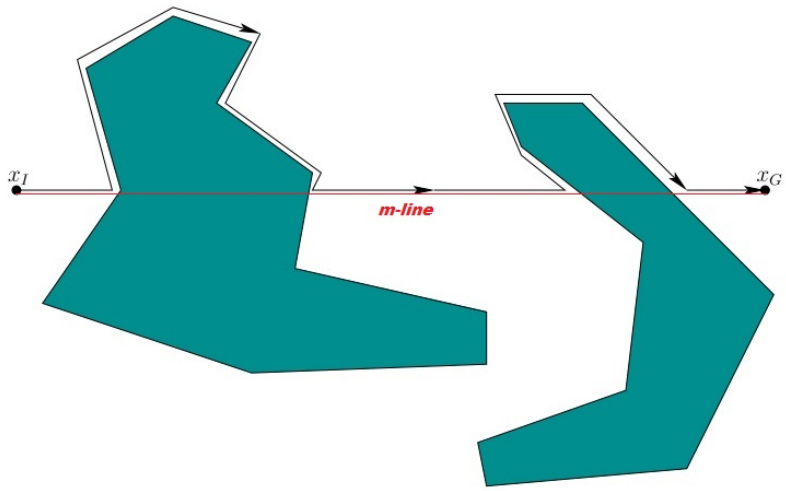


Figure 2.4: *An Illustration of the Bug2 Algorithm. [16]*

As was expressed so far, it is possible that this strategy takes infinite cycles to complete the planning. Therefore, a small modification is needed. The agent must remembers the distance to the goal from the last point at which it departed from the boundary, and only departs from the boundary again if the candidate point is closer to the goal. This is applied iteratively until the goal is reached or it is deemed to be impossible. For the Bug2 strategy, the total distance traveled is no more than D_{max}, as expressed in equation 2-4;

$$D_{max} = d + (1/2)\sum_{i=1}^{M} n_i p_i \tag{2-4}$$

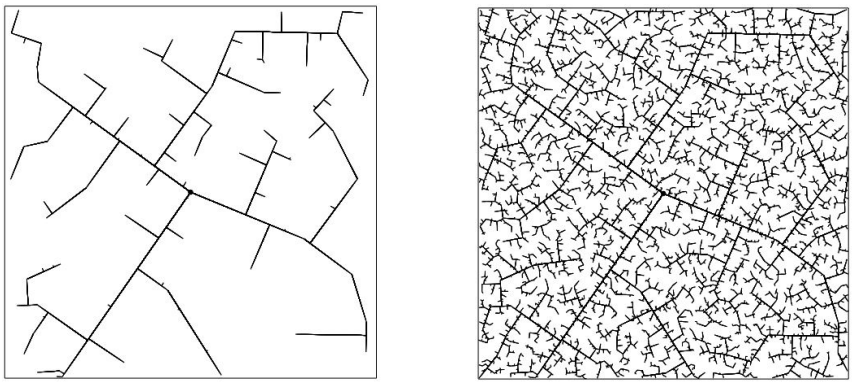
In which d is the Euclidean distance from the initial position to the goal position, p is the perimeter of the i^{th} obstacle, M is the number of obstacles and n is the number of times the *ith* obstacle crosses the m-line.

As seen in the previous subsections, the Bug1 can be considered overcautious, yet effective, meanwhile Bug2 can be inefficient in some cases, such as local loops and crossings, but usually generates better optimized paths when compared to Bug1 [3]. The Bug2 strategy was chosen for comparisons due to its unique, yet simple, circumnavigation

behaviour that works with either global or local path planning problems and unlike the Bug1, outputs more optimized paths. The classic Bug2 was also chosen in the detriment of the DistBug, VisBug, and TangentBug, as being the simplest implementation that still maintains the core characteristics wanted for comparison, as described earlier.

2.2.3 The Rapidly-exploring Random Tree - RRT

The Rapidly-Exploring Dense Tree (RDT) is one of the many incremental sampling and searching approaches that usually yields good performance in practice without any parameter tuning [16]. The core idea behind these approaches is to incrementally advance a tree towards the largest free portions of the environment, also known as Voronoi regions, using collision-free branches. Through enough iterations, the tree should densely cover all the free space in the environment ultimately finding the optimal path between two points; an example of the RDT's coverage in different iterations can be seen in figure 2.5.



(a) RRT after 45 iterations. [16] (b) RRT after 2345 iterations. [16]

Figure 2.5: *An Illustration of the RRT's coverage.*

The configuration space C defined in subsection 2.1 is infinite, yet RDTs work by considering at most a finite number of points (i.e samples) from C. The key idea behind this strategy is to exploit advances in collision detection algorithms that were designed to compute whether a single configuration is collision-free. Given this simple primitive, the algorithm samples different configurations to construct a data structure that stores 1D C-space curves, which represent collision-free paths. In this way, RDTs do not access the C_{obs} directly but only through the collision detector and the constructed data structure. Using this level of abstraction, the planners are applicable to a wide range of problems by tailoring the collision detector to specific robots and applications

RDTs can sample the C-space probabilistically or deterministically. Either way, the requirement is usually that a dense sequence, α, of samples is obtained. In this context,

denseness denotes the ability of a given sample sequence to get arbitrarily close to every single element of C, assuming $C \subseteq \mathbb{R}^n$. Mathematically, suppose that $C = [0,1]$ and let $I \subset [0,1]$ be an interval of length e. If k samples are chosen independently at random, the probability that none of them falls into I is $(1-e)^k$. As k approaches infinity, this probability converges to zero meaning that the probability of any nonzero-length interval in $[0,1]$ to not contain points converges to zero. In other words, the number of samples tends to infinity, the samples get arbitrarily close to every point in C.

For probabilistic sampling, this denseness (with probability one) ensures probabilistic completeness for the planning algorithm, in which a solution is likely to be found as long as the sampling sequence is large enough. For deterministic sampling, it ensures resolution completeness, which means that if a solution exists, the algorithm is guaranteed to find it; otherwise, it may search indefinitely.

The RRT is a special case of a Rapidly-Exploring Dense Tree, which uses a random sampling strategy. The basic idea behind it is to randomly sample the C-space and then induce a Voronoi bias in the exploration process by selecting for expansion the point in the tree that is closest to a given sample in each iteration. Therefore, the probability that a vertex is chosen is proportional to the volume of its Voronoi region. The The original RRT [15] was also introduced with a step size parameter, restricting the maximum size of its exploring branches and greatly simplifying the implementation of this approach.

Let α denote a random sequence of samples that are present in the configuration space $C = [0,1]$, while the i^{th} sample being represented by $\alpha(i)$. Based on that, the RRT can be expressed as a topological graph represented by an ordered pair $G = (V,E)$, comprising a set V of vertexes together with a set E of edges with $S \subset C_{free}$ indicating the set of all points reached by G. Since each $e \in E$ is a path, the *swath* S of the search graph is the union of all possible known paths and is formally denoted as expressed in equation 2-5.

$$S = \bigcup_{e \in E} e([0,1]). \tag{2-5}$$

With this mathematical formulation in mind, a pseudo-code of the RRT can be done as expressed in algorithm 2.2.

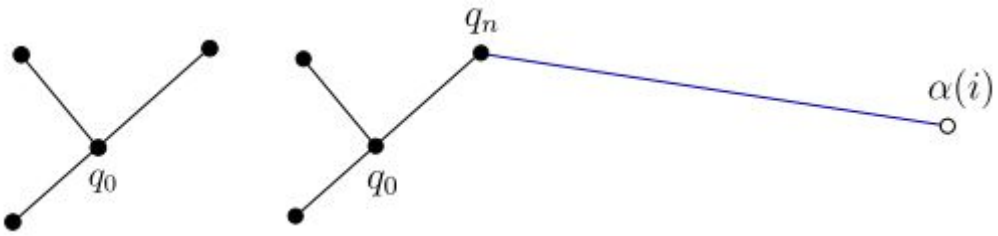
Algorithm 2.2: Pseudo-code of the RRT

Let G be the representation of the graph structure used in the RRT, q_0 be the starting point of the planning problem and K be the sampling sequence size.

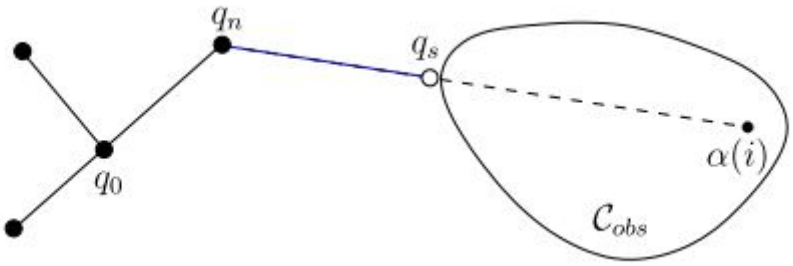
```
/* Initialize the tree with the starting position.                    */
G.init(q_0)
/* Repeat the expansion until the max iteration is reached            */
for i=1 to K do
    /* Find the nearest edge relative to the sampling α(i) that is present
       in S                                                           */
    q_n ← NEAREST(S, α(i))
    /* Find the nearest new edge possible within the boundary of C_free,
       along the direction toward α(i), if no collision is detected the
       output is α(i)                                                 */
    q_s ← STOPPING_CONFIGURATION(q_n, α(i))
    /* As long as the edges q_s ≠ q_n, the new edge can be added to the RRT
       alongside its vertex.                                          */
    if q_s ≠ q_n then
        /* Add the new vertex                                        */
        G.add_vertex(α(i))
        /* Add the sampled edge connected to the nearest edge        */
        G.add_edge(q_n, α(i))
    end
end
```

Supposing that a RRT has been constructed so far using the pseudocode provided in algorithm 2.2, an step-by-step of its expansion strategy can be seen in figure 2.6. In detail, 2.6(a) represents an arbitrary iteration on the exploring tree trying to expand towards a goal. In this step, the strategy is to use the *NEAREST* function to find the nearest edge in S relative to the sampling position, followed by the *STOPPING_CONFIGURATION* function in order to discover a new possible edge based on the $\alpha(i)$ sample.

In the absence of obstacles, the new edge should overlap with the sampled position, as indicated by 2.6(b). On the order hand, if an obstacle is detected in the way, the new edge should be in an intermediary position towards $\alpha(i)$, usually defined by a "collision avoidance range" chosen beforehand, as illustrated in figure 2.6(c). In the other hand, the overall behaviour of the RRT with the presence of obstacles in the workspace can be seen in figure 2.7, being possible to see the expansion towards the objective while still maintaining a certain distance from obstacles.

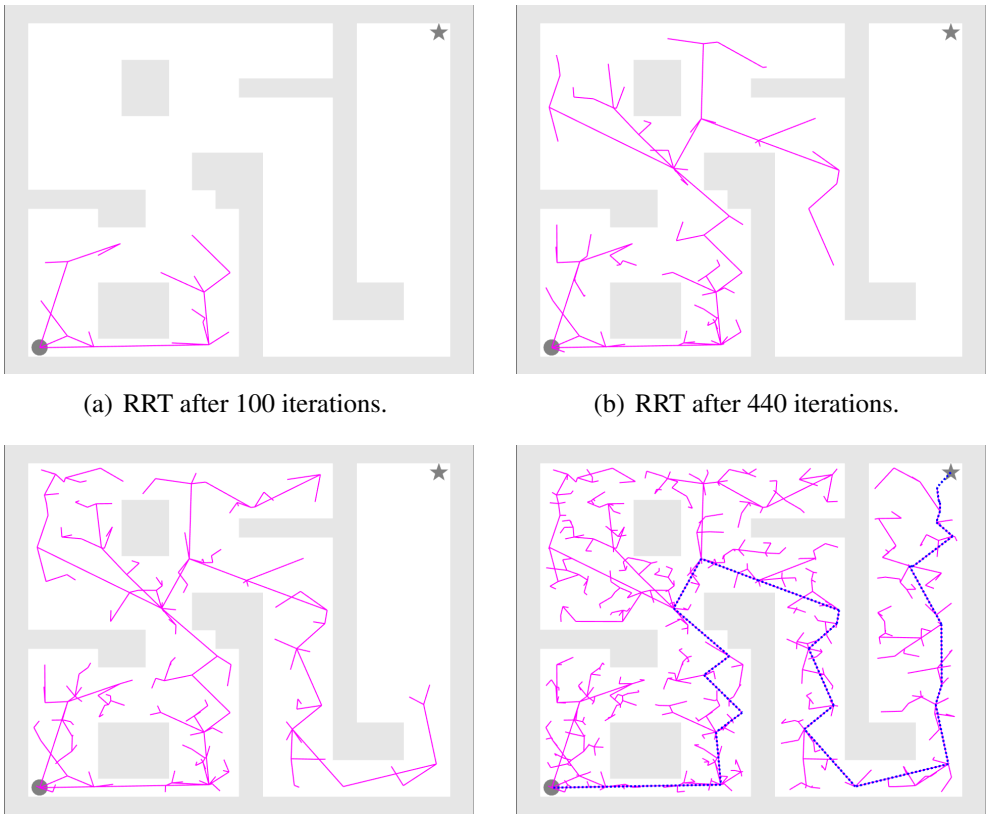


(a) Initial Tree. (b) Branch expansion without obstacles.



(c) Branch expansion with obstacles present.

Figure 2.6: *An Illustration of the RRT's expansion strategy.*



(a) RRT after 100 iterations. (b) RRT after 440 iterations.

(c) RRT after 662 iterations. (d) Final path after 1421 iterations.

Figure 2.7: *An Illustration of the RRT's behaviour in the presence of obstacles.*

The classic RRT approach described in this subsection is widely used in global path planning problems that uses continuous representation. Although being an efficient alternative for high dimensional maps, it commonly struggles with narrow passages and obstacle dense environments, due to the biased search towards large Voronoi regions. Overall, the RRT shows good performances in real-world applications with the use of continuous space representation.

2.2.4 The GA Path Planning

Genetic algorithms (GAs) have been widely explored as solutions to the global path planning problem with discrete representations of the environment [27]. The GA's search and optimization are inspired by the principles of natural selection, proposed on Darwin's theory about evolution (the survival of the fittest), and as such, in GA based approaches a group of candidate solutions goes trough natural selection and genetic operations to find the ones with better fitness.

The figure 2.8 shows an illustration on different possible solutions considered by a GA for the same map while searching for the fittest path. GAs were first created by John Holland while studying the phenomenon of adaptation as it occurs in nature and trying to develop ways in which the mechanisms of natural adaptation might be imported into computer systems [8].

Holland presented the genetic algorithm as an abstraction of biological evolution and gave a theoretical framework for adaptation under it. His solution to computational adaptation starts from one population of "chromosomes" (e.g., sets of ones and zeros, or "bits") and ends in a new population by using "natural selection", in the means of a fitness function, together with the genetic inspired operators of crossover, mutation, and selection.

Each chromosome consists of "genes" (e.g., bits), with each gene being an instance of a particular "allele" (e.g., 0 or 1). The selection operator chooses the chromosomes in the population that will be allowed to reproduce, with fitter chromosomes being most likely to produce offspring than less fit ones. The crossover operator exchanges sub parts of two chromosomes, mimicking biological recombination between two single chromosome organisms; while the mutation operator randomly changes the allele values of some locations in the chromosome.

Holland's introduction of a population based algorithm was a major innovation. Moreover, it was the first to attempt to put computational evolution on a firm theoretical base [8]. Until recently this theoretical foundation, based on the notion of "schemas," was the basis of almost all subsequent theoretical work on genetic algorithms.

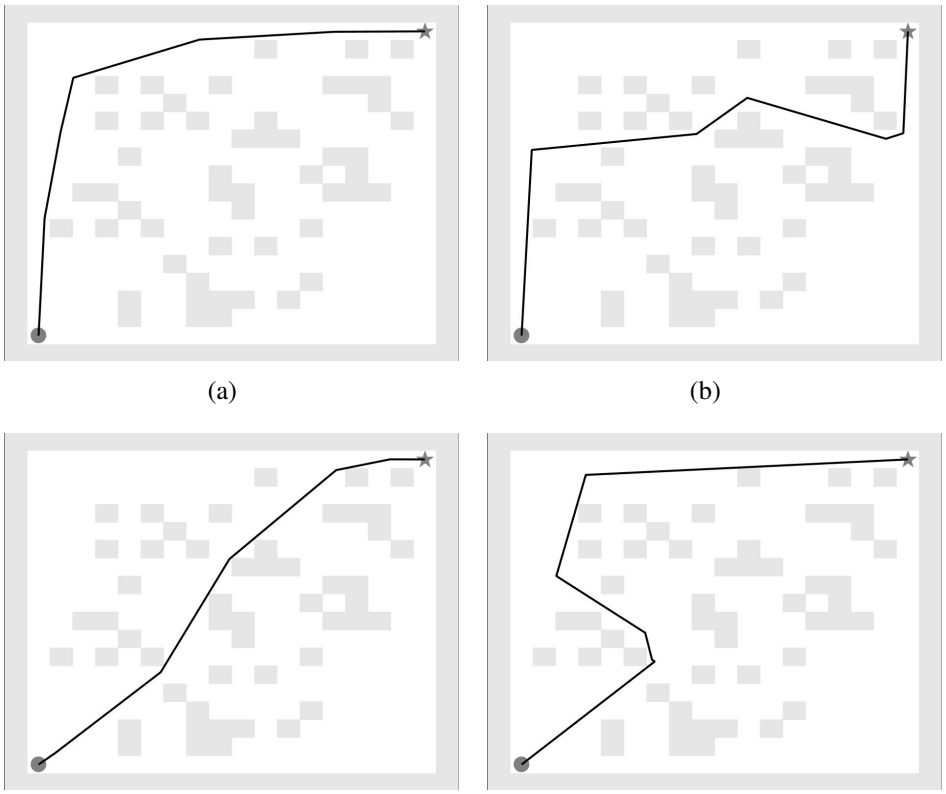


Figure 2.8: *Different chromosomes evaluated by the GA.*

In order to use GAs for path planning applications the path planning paradigm defined in this chapter must be modeled as an optimization problem. To achieve this, a objective function and optimization variables must be formally expressed. The common formulation is defined as follows: let a path be characterized by a fixed number of points in the environment, with each point being connected to the next by a straight line and the last being connected to the goal, as illustrated in figure 2.9.

The objective function is usually a minimization of the combined length of this path with an additive penalty if any part of the path lies inside the obstacle, with it being proportional to the length inside the obstacle. The locations of each of these fixed number of points (in both x and y axis positions) are the optimization variables (i.e genes) present in the chromosome, while the number of points is commonly referred as the chromosome size. The respective chromosome to the path shown in figure 2.9 is illustrated in table 2.3.

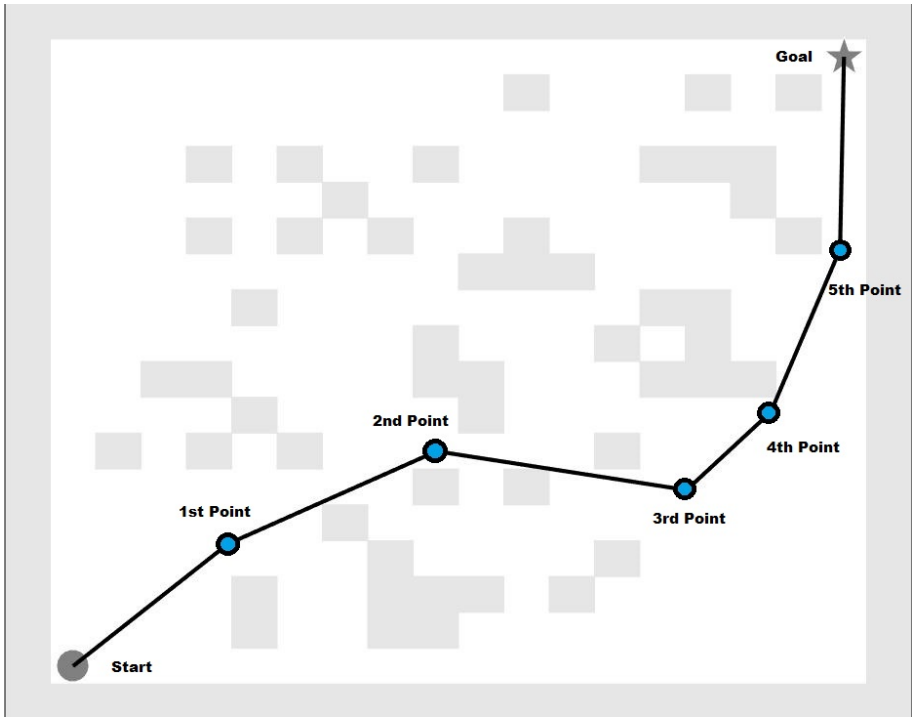


Figure 2.9: *Common path representation used by GAs.*

Table 2.3: *Chromosome representation for the path illustrated in figure 2.9*

	1º Gene	2º Gene	3º Gene	4º Gene	5º Gene
x (cm)	**24.59**	**47.58**	**74.97**	**84.10**	**92.05**
y (cm)	**24.25**	**37.26**	**32.05**	**42.74**	**65.34**

The total number of points is usually an algorithm parameter and should be equal to the maximum number of turns the agent is expected to make in the environment. Setting this number too high can result in large computational requirements and if slow processing time is not allowed, random results may be the output. Furthermore, a large chromosome in simple scenarios will result in useless turns and high path length, while a small one may not give enough flexibility to the algorithm to model the optimal path, thus resulting in collision-prone paths.

The inherently parallel and high quality search are clearly the main advantages of the Genetic Algorithm when opposed to other search heuristics [30]. However, some important characteristics of GAs can also translate into difficulties when applied to the path planning problem. The initial population may include impossible paths, granting low genetic diversion to feasible paths and delaying the convergence [30]. Likewise, large chromosome sizes can result in impracticable paths due to time requirements, further diminishing the efficiency of this GA path planning approach.

Many researchers tried to take advantage of the GA's optimization ability to compute optimal paths in the context of path planning [35, 10, 9, 17]. However, most of them still employs computationally expensive approaches, using standard path planning representation, fixed-length binary chromosomes, basic genetic operators, with few modifications made on the GA canonical structure in order to improve GA based path planning.

For example, despite presenting optimal paths, the technique proposed in Thomaz et al [35] suffers from several problems, being computationally expensive, requiring large memory space when dealing with dynamic or large environments, and being very time consuming, making it impractical for mobile agents applications.

In other research, Ismail et al [10] propose a GA based path planning that allow more degrees of freedom on movement, but only on low complexity maps, a restriction that cannot be employed in real unstructured environments. And lastly, Hu et al [9] make good improvements with a variable-length chromosome paired with knowledge based operators, but their approach requires domain knowledge to find feasible solutions for the initial population, adding another step to an already complex approach, compromising performance.

Lastly, the GA path planning by R. Kala [11] employs the standard evolutionary solution to the path planning problem, using fixed-length binary chromosomes, basic genetic operators and a continuous representation of the environment. However, this basic approach clearly shows the advantages and weaknesses inherent to this type of solution, with the benefit of being an open-software application of the GA path planning, facilitating the reproduction of his work. Therefore, this approach was chosen to compose the comparison group.

CHAPTER 3

The Proposal

This chapter starts with an explanation on the compact genetic algorithm and its real encoded variation in section 3.1. It is then presented in section 3.3 the CGPlan's proposal, going through the specific features designed for the path planning application.

3.1 The Compact Genetic Algorithm

Analyzing the growth and decay of a particular gene in the population of a GA as a one-dimensional random walk, Harik et al. [6] observed that as the GA progresses, genes fight with their competitors and their number in the population increase or decrease depending on the GA's operators.

With a different understanding of the role of the GA's parameters and operators, Harik et al [6] proposed a minimalist view of the population and creates an algorithm that mimics the behavior of conventional GAs by evolving a Probability Vector (PV) that describes the hypothetical distribution of each gene in a population of solutions.

Named Compact Genetic Algorithm (cGA), it is designed to iteratively process the PV with updating mechanisms that mimic the typical selection and recombination operations performed in a standard GA until a stopping criteria is achieved. A pseudocode of the classic cGA proposed by Harik et al. [6] can be seen in algorithm 3.1.

The main strength of the cGA is the significant reduction of memory requirements, as it needs to store only the PV instead of an entire population of solutions. This feature makes cGAs particularly suitable for memory-constrained applications [6] such as the path planning problem.

A further description of the PV can be found in subsection 3.1.1 and the operators present in the cGA and its relation to the classic GA operators are explored in the subsections 3.1.2 and 3.1.3, note that although used in the classic GA, a mutation operator was not included in the classic cGA [6]. The number of individuals generated, the number of individuals to update from, the stopping criterion, and the rate of the probability vector's change are all parameters of the cGA.

3.1.1 The Probability Vector - PV

In the classic implementation of the cGA by Harik et al. [6], the probability vector's role is to record the proportion of ones (and consequently zeroes) at each gene position in the virtual population. These proportions are initially set to 0.5, expressing an equal occurrence of zeros and ones, and move towards one of them as the updates are made.

The aim of this update is to orientate the probability vector toward the fittest of the generated solutions. This update step also has a constant size of $\frac{1}{n}$. Therefore, while the classic GA needs to store n bits for each gene position, the cGA only needs to keep a proportion of ones (and zeros), being a finite set of $(n+1)$ numbers that can be stored with $log_2(n+1)$ bits.

3.1.2 Selection

The selection operator aims to choose better individuals for reproduction on a given population. But unfortunately, this does not always benefit the better genes, as genes are always evaluated within the context of a larger individual. For example, consider the onemax problem, where the fitness evaluation of an individual depends solely on the frequency of genes with value 1. Given the two individuals a and b in table 3.1 :

Table 3.1: *Individuals for the onemax example.*

Individual	Chromosome	Fitness
a	1011	3
b	0101	2

The selection operator will analyze the fitness function and declare individual a to be more likely to reproduce. However, at the gene level a decision error is made on the second position. Being that, the selection operator incorrectly prefers the schema *0* to *1* in the second position. Fortunately the role of the population in GAs is to buffer against a finite number of such decision errors.

Now consider a steady-state binary tournament selection that chooses two individuals randomly from the population and keeps two copies of the better one. With this selection, considering a population of size n, the proportion of the winning gene will always increase by $1/n$. For instance, in the onemax example the proportion of 1's will increase by $1/n$ at gene positions 1 and 3, and the proportion of 0's will also increase by $1/n$ at gene position 2. At gene position 4, the proportion will remain the same, as both chromosomes have this gene present. This experiment suggests that an update rule increasing a gene's proportion by $1/n$ can simulate a steady-state binary tournament selection of a GA with a population of size n.

With that in mind, Harik et al. [6] proposed a similar update rule for the classic cGA, aiming to select better individuals without the inherit decision error in the gene level caused by the usual selection operator used in GAs. This update is triggered when the best individual of a given generation is chosen, iterating each position of the winning chromosome and updating the corresponding position in the PV with $(+\frac{1}{n})$, in the case of a gene one, and with $(-\frac{1}{n})$, otherwise.

Algorithm 3.1: Pseudocode of the cGA

Let l be the length of the probability vector and n be the update rate.

```
   /* Initialize the probability vector with probability 0.5      */
1  for i=1 to l do
2  |  PV[i]=0.5
3  end
   /* Generate a random solution based on PV                      */
4  father ← generate(PV)
   /* Repeat until PV is fully converged                          */
5  while NotConverged(PV) do
6  |  son ← generate(PV)
   |  /* Evaluate the solutions                                   */
7  |  [fatherFit, sonFit] ← fitness(father, son)
   |  /* Define the best (winner) and worse (loser) solutions.   */
8  |  if fatherFit >= sonFit then
9  |  |  winner ← father
10 |  |  loser ← son
11 |  else
12 |  |  winner ← son
13 |  |  loser ← father
14 |  end
   |  /* Update the probability vector towards the winner.       */
15 |  for i=1 to l do
16 |  |  if winner[i]==1 then
17 |  |  |  PV[i]+=(1/n)
18 |  |  else
19 |  |  |  PV[i]-=(1/n)
20 |  |  end
21 |  |  father ← winner
22 |  end
23 end
```

3.1.3 Crossover

The role of crossover in the classic GA is to combine schemes from fit solutions in order to create better individuals. However, when repeatedly used in a population, it eventually leads to a decorrelation of it's genes, making it possible to be compactly represented as a probability vector of genes [6]. Further analysis showed that in this probability vector state it is possible to generate individuals with decorrelated genes, without the need of crossover [6]. Therefore, the generation of individuals from the PV in the cGA can be seen as a shortcut to the eventual aim of usual crossover operators of the GA.

3.2 The Real-encoded Compact Genetic Algorithm.

Real-value encoding has been widely used in evolutionary algorithms (including GAs) to improve the quality of solutions for problems in the real numbers scope [21], as binary encoding and its conversion into real solutions tends to increase the granularity in real numbers applications.

Therefore, to avoid additional computational cost related to the binary-to-float conversions and also increase the quality of solution, Mininno et al proposed a variant of the cGA that works directly with real-valued chromosomes, using a PV that stores the mean and the standard deviation of the distribution of each gene in the hypothetical population. New update rules are also introduced in order to evolve the PV in a way that mimics the binary-coded cGA, which in turn, emulates the behavior of the standard GA. This new variant of the cGA was named real-encoded compact genetic algorithm, or rcGA in short.

In the context of search-based path planning, the use of real-encoded path representation brings various benefits to the application, since the final solution will not be quantized nor restricted in fixed directions as the binary approach. This feature is especially useful when using the continuous representation of the environment, as the final solution can be refined even further to produce an optimal path. Therefore, the Real-coded Compact Genetic Algorithm (rcGA) combines a reasonable time and resources consumption with the ability to deliver optimized real-valued solutions, being a promising search approach on the path planning problem especially for applications involving low performance mobile agents.

3.2.1 The real-value representation in the rcGA.

As seen previously in section 3.1, a single gene in the cGA is expressed with a scalar between [0,1], it being the probability of finding a "0" or a "1" in this single gene.

As the rcGA uses real-valued genes, the distribution of a single gene in the hypothetical population must be described by a Probability Density Function (PDF) defined on the normalized interval [-1,+1]. Assuming that the distribution of the i^{th} gene can be described with a Gaussian PDF with mean x_i and standard deviation σ_i. More precisely, since the Gaussian PDF is defined in $(-\infty, +\infty)$, a "Gaussian-shaped" PDF defined between [-1,+1] is used. The height of this PDF is also normalized so that its area is equal to one.

Let us consider a minimization problem in a m-dimensional hyper-rectangle, with m being the number of parameters. Without loss of generality, let us assume that these parameters are also normalized so that each search interval is contained in [-1,+1]. Therefore, in the rcGA the PV becomes a $m \times 2$ matrix specifying the two parameters of the PDF for each gene. Thus, the PV can be formally defined as

$$\mathbf{PV}^{(k)} = [\mathbf{x}^{(k)} \quad \sigma^{(k)}] \tag{3-1}$$

where $\mathbf{x}^{(k)} = [\mathbf{x}[1]^{(k)}\ \mathbf{x}[2]^{(k)}\ ...\ \mathbf{x}[m]^{(k)}]$ is the vector of mean values, $\sigma^{(k)} = [\sigma[1]^{(k)}\ \sigma[2]^{(k)}\ ...\ \sigma[m]^{(k)}]$ is the vector of standard deviations, and k being the iteration index. Therefore, it is possible to express the update function for the mean x of a gene g in an iteration k with the following equation:

$$\mathbf{x}[g]^{(k+1)} = \mathbf{x}[g]^{(k)} + \frac{1}{n}.(\mathbf{winner}[g] - \mathbf{loser}[g]) \tag{3-2}$$

where **winner** and **loser** are the chromosomes with best and worst fitness in the given iteration k. Similarly, it is possible to express the update function for the standard deviation σ of a gene g in an iteration k with the following equations:

$$\mathbf{V}[g]^{(k)} = (\sigma[g]^{(k)})^2 + (\mathbf{x}[g]^{(k)})^2 - (\mathbf{x}[g]^{(k+1)})^2 + \frac{1}{n}.((\mathbf{winner}[g])^2 - (\mathbf{loser}[g]]^2) \tag{3-3}$$

$$\sigma[g]^{(k+1)} = \begin{cases} \sqrt{\mathbf{V}[g]^{(k)}}, & \text{if} \quad \mathbf{V}[g]^{(k)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

With these modifications in mind, the pseudocode of the rcGA is defined in algorithm 3.2.

Algorithm 3.2: Pseudocode of the rcGA

Using the definitions presented in section 3.2 and n being the virtual population size that regulates the update step, the rcGA goes as follows:

```
/* Initialize the probability vector with initial mean x_ini and
   standard deviation σ_ini                                        */
```
for *i=1 to l* **do**
  PV[i][1]=x_{ini}
  PV[i][2]=σ_{ini}
end
```
/* Generate a random solution based on PV                          */
```
father ← generate(**PV**)
```
/* Repeat until PV is fully converged                              */
```
while *NotConverged(***PV***)* **do**
  son ← generate(**PV**)
  ```
  /* Evaluate the solutions                                        */
  ```
  $[fatherFit, sonFit] \leftarrow fitness(father, son)$
  ```
  /* Define the best (winner) and worse (loser) solutions.         */
  ```
  if $fatherFit >= sonFit$ **then**
    $winner \leftarrow father$
    $loser \leftarrow son$
  else
    $winner \leftarrow son$
    $loser \leftarrow father$
  end
  ```
  /* Update PV towards the winner.                                 */
  ```
  update(**PV**, winner, loser)
  father ← winner
end
Function *update(***PV***, winner, loser)*
  for *i=1 to l* **do**
    $old\_x = \mathbf{x}[i]$
    $\mathbf{x}[i] = \mathbf{x}[i] + \frac{1}{n}.(winner[i] - loser[i])$
    $V = \sigma[i]^2 + (old\_x)^2 - (\mathbf{x}[i])^2 + \frac{1}{n}.(\mathbf{winner}[i]^2 - \mathbf{loser}[i]]^2)$
    $\sigma[i] = sqrt(max(0, V))$
  end
  PV = [**x** σ]
end

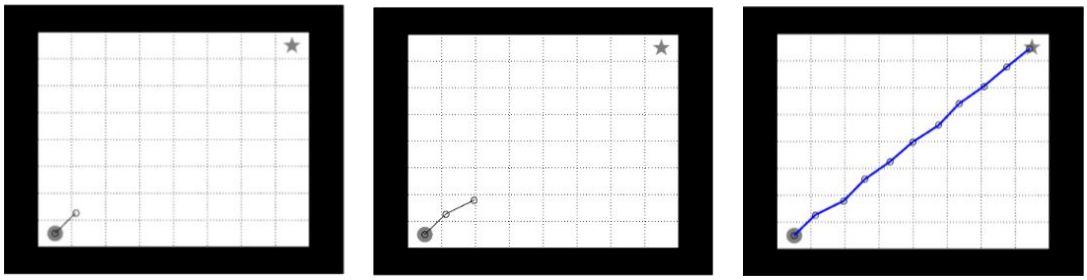
3.3 The CGPlan

As stated by Hu and Yang [9], path representation is indeed a key issue for the efficient use of any evolutionary path planning. Hence, its proposed a new representation of individuals for the CGPlan, based on concepts of the local path planning paradigm.

In order to incorporate the reaction and planning behaviours within an evolutionary technique, it is proposed a constructive approach on the assembly of the optimal path. It starts with the rcGA searching for optimal partial trajectories that solve the path planning on a local scope, with this stage being named the rcGA cycle. In this case, the chromosome expresses a straight partial trajectory of the agent towards the objective, restricted by a maximum length, while the PV represents a hypothetical population of these chromosomes.

After a cycle is over (i.e the maximum generation count is reached), the rcGA outputs the fittest chromosome, namely, the best partial trajectory found in that scope. This partial trajectory is then applied to the agent, updating its position. This process continues until the objective is reached, with the collection of the obtained trajectories forming the final path between the starting position and the objective.

These partial trajectories are not restricted in directions nor quantized within grids, different from the usual discrete approach with GA path planning. This representation can also reach solutions similar to the variable-length chromosome, without the resource management problems, producing a high quality path with efficient resource consumption that can be used in either local or global path planning problems. A detailed illustration of the algorithm's constructive output during each cycle can be seen in figure 3.1.



(a) First cycle. (b) Second cycle. (c) Path found after 10 cycles.

Figure 3.1: *Illustration of the CGPlan's output at differing cycles.*

3.3.1 Evaluation

The CGPlan uses a distance transform (also known as distance map) of the environment to evaluate possible trajectories. This transform is continuously calculated

using gathered environment and obstacle data and outputs a pixel grid in which each pixel stores its distance to the nearest obstacle. Formally, the distance transform can be expressed as follows:

Let $\Gamma = \{0,...,n-1\} \times \{0,...,m-1\}$ be a two-dimensional discrete map, and $f : \Gamma \to \mathbb{R}$ be a function describing the obstacles. The two-dimensional distance transform of f under the squared Euclidean distance, DT, is given by equation 3-4.

$$DT_f(x,y) = \min_{x'y'}((x-x')^2 + (y-y')^2 + f(x',y')). \tag{3-4}$$

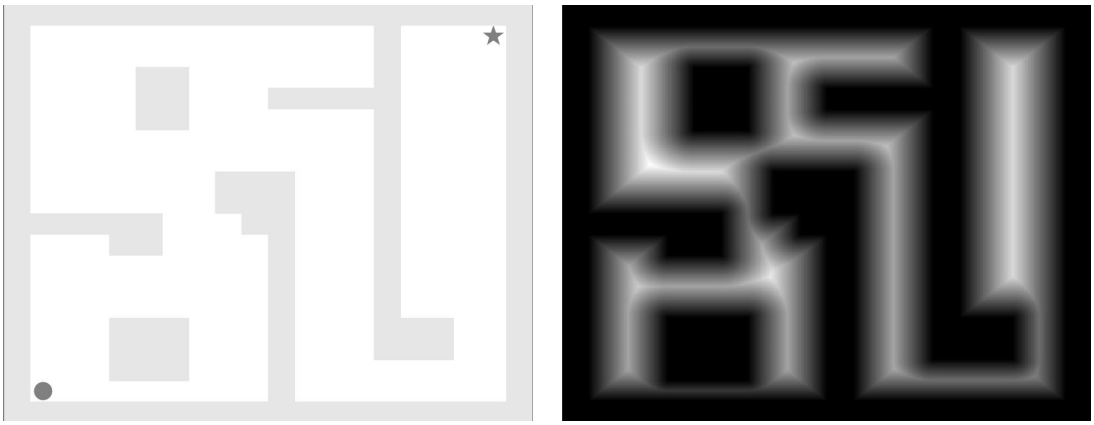
The first term of equation 3-4 does not depend on y'. Therefore, we can rewrite it as expressed in equation 3-6.

$$DT_f(x,y) = \min_{x'}((x-x')^2 + \min_{y'}((y-y')^2 + f(x',y'))), \tag{3-5}$$

$$DT_f(x,y) = \min_{x'}((x-x')^2 + DT_{f|x'}(y)). \tag{3-6}$$

Where $DT_{f|x'}(y)$ is a one-dimensional distance transform of f restricted to the column indexed by x'. Therefore, a two-dimensional distance transform can be computed by fist evaluating a one-dimensional distance transform along each column of the map, and another on-dimensional distance transform along each row of the result. This argument can be extended to arbitrary dimensions, resulting in the composition of transforms along each dimension of the map. This generalized multidimensional transform runs in $O(dN)$ time, where d is the dimension of the map and N is the overall number of discrete cells.

Although the repetitive use of this technique can become expensive computationally, in this approach it is executed only when new obstacles or features are detected by the mobile agent, not harming our resource management goal. A visual representation of the distance transform can be found in Figure 3.2.



(a) Original map. (b) Distance transform representation.

Figure 3.2: *Comparison of the original map and a visual gray-scale representation of the distance transform. A darker pixel means closer to an obstacle in this representation.*

To work along the constructive path representation, the algorithm must be able to evaluate each partial trajectory individually, numerically expressing its solution quality and distinguish feasible and infeasible ones. The evaluation function tailored to perform this task is presented in equation 3-7:

$$fitness = \frac{(m+1)}{(m+2)} + \frac{k}{(k+d)} \tag{3-7}$$

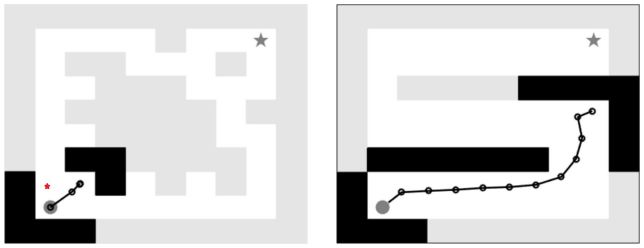
Where m is obtained from the distance map and represents the minimum distance between the trajectory and the closest obstacle, d is the Euclidean distance between the last point of the evaluated trajectory and the objective and k is a constant weight to adjust the bias towards the goal.

A partial trajectory with a high m will drive the agent far from the obstacles, while a $m = 0$ indicates an imminent collision. Meanwhile, the d value becomes smaller if a trajectory manages to guide the agent closer to the goal and increasing as the agent diverge from the goal. On the other hand, the weight k regulate the relation between obstacle avoidance and final path length, with a high value representing a cautious trajectory.

Therefore, it can be noticed that the best paths will be expressed by the highest values given by this fitness function, while still penalizing infeasible segments (i.e $m = 0 \rightarrow$ a lower fitness) and ensuring a non-null fitness that allow their data to still update the probability vector, thus encouraging genetic diversity.

3.3.2 Avoiding local minima

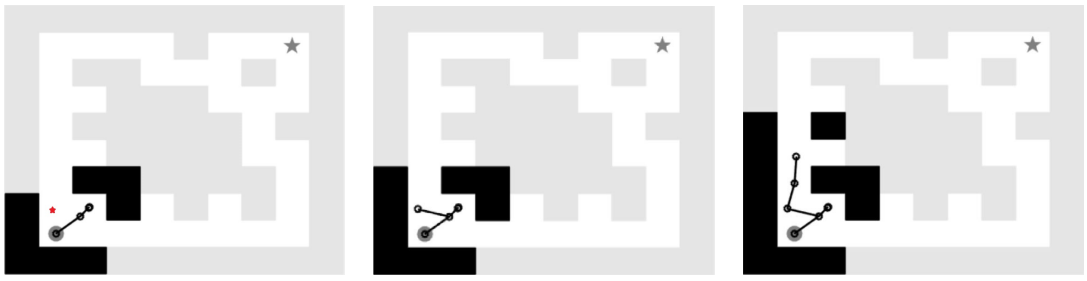
As a local planning strategy, the proposed algorithm acts as a greedy optimization procedure, becoming vulnerable to local minimum on the evaluation function. This situation usually occur when the agent runs into a dead end (e.g. inside a corner or U-shaped obstacle), or into an obstacle bigger then the maximum length of a partial trajectory (e.g in front of a wall-like obstacle). These can be easily detected by monitoring the trajectory size between cycles, as the agent tends to "orbit" the local minimum with a trajectory length close to zero. An illustration of these scenarios can be seen in figure 3.3.



(a) Corner obstacle. (b) Wall-like obstacle.

Figure 3.3: *Representation of possible local minimum situations.*

The solution found was to impose a restriction in the local minimum search surface and then set a new temporary objective to redirect the agent. The new objective is selected in order to circumnavigate the obstacle causing the minimum and it expires when reached, allowing the CGPlan to advance towards the original goal once more. This modular strategy was designed to fit our constructive representation, allowing the agent to avoid local minimum while maintaining the partial trajectories approach. An illustration of the proposed local minimum solution can be seen in figure 3.4.



(a) Local minimum detected, choosing new objective. (b) Temporary objective reached, returning objective to goal. (c) Continue normal behaviour.

Figure 3.4: *Illustration of the CGPlan's strategy to overcome local minimum.*

A simplified version of the algorithm using all the features discussed in the sections above is presented in Algorithm 3.3.

Algorithm 3.3: Pseudocode of the CGPlan

Let **PV** be the population's probability vector, N be the max generation count and C be the current point on the constructive path.

$START$ and END are respectively the starting and goal coordinates.

```
/* Initialize path on starting coordinates.                    */
```

$C \leftarrow START$

while $C \neq END$ **do**

  $T \leftarrow 0$

  $objective \leftarrow END$

  if $New\_Obstacle\_Found()$ **then**

    ```
    /* If new information is found, update distance map.        */
    ```

    $DistanceMap \leftarrow DistanceTransform(Obstacles)$

  end

  if $LocalMinimumFound()$ **then**

    ```
    /* If local minimum found, generate new objective.         */
    ```

    $objective \leftarrow AvoidLocalMinimum(C, DistanceMap)$

  end

  ```
  /* Initialize the probability vector with initial values.    */
  ```

  $initialize(\mathbf{PV})$

  ```
  /* Generate a random solution based on PV                    */
  ```

  $father \leftarrow generate(\mathbf{PV})$

  while $T < N$ **do**

    $son \leftarrow generate(\mathbf{PV})$

    ```
    /* Evaluate solutions using the distance map.              */
    ```

    $(fatherFit, sonFit) \leftarrow evaluate(father, son, DistanceMap, objective)$

    ```
    /* Define the best (winner) and worse (loser) solutions.   */
    ```

    $(winner, loser) \leftarrow tournament(fatherFit, sonFit)$

    ```
    /* Update the probability vector towards the winner.       */
    ```

    $update(\mathbf{PV}, winner, loser)$

    $father \leftarrow winner$

    $T \leftarrow T + 1$

  end

  ```
  /* Update the constructive path with the winning trajectory */
  ```

  $C \leftarrow C + winner$

end

CHAPTER 4

Experiments and Results

The four established approaches of intelligent navigation discussed in chapter 2 were compared in terms of performance, quality of solution, and scalability against the proposed CGPlan. These techniques follow different strategies but were able to effectively guide a mobile agent towards an objective in related works.

4.1 The simulation setup

All these path planning approaches were implemented and tested on the same simulated environment, created using Matlab R2015b in a system with a Intel Core i5-4670k 3.4GHz and 16GB RAM. More information on the simulation suite can be found in appendix B. About the techniques, the A* and Bug2 were implemented in their classic form whereas the GA and RRT were implementations by Rahul Kala, found in [11] and [12], respectively. Furthermore, other relevant information on the implementations of each technique are:

- The implemented A* uses a euclidean distance function.
- The implemented BUG2 always prefer to circumnavigate an obstacle by its right side.
- The GA path planning from R. Kala uses a real-valued chromosome in witch each gene represents a sequential point in the path. The chromosome have a fixed length of 5, meaning that the path generated will always be formed by 5 points. This implementation also uses a stochastic uniform selection operator, a standard Gaussian mutation operator and a binary mask crossover operator. Another parameters of this GA are: population size = 50, number of generations = 10 and mutation rate = 0.01.
- The RRT path planning from R. Kala is a usual implementation of the RRT using a maximum sample size of 10000 with a threshold of 5cm from the goal to prematurely end the planning.

- The CGPlan's parameters used in its implementation were : 800 generations per cycle, an update rate of $\frac{1}{50}$ and a weight $k = 150$ in the fitness function.

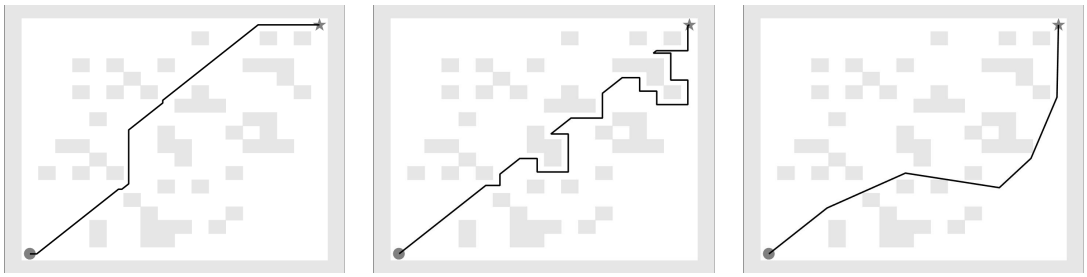
The environment was assembled in a 100x100cm simulated area, using 0.1x0.1cm cells for the discrete representation. The obstacle detection was standardized for all techniques, discretizing the obstacles into the cells used for the discrete representation and these cells were then represented with absolute coordinates in the continuous representation. The time consumption of each technique is measure from its start to the output of the final path, including the smoothing technique if present. On the other hand, the path length is calculated after the output, using the points in the final path returned by each technique.

4.2 Performance and quality comparison

The performance and quality of solutions were compared in two scenarios, random scattered obstacles and indoor-like configurations. The objective of these tests was to find the strengths and flaws of the CGPlan opposed to renowned solutions.

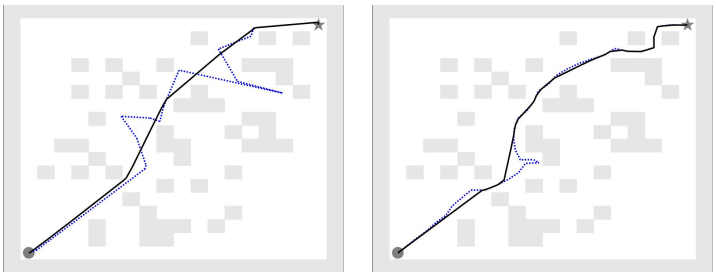
4.2.1 First scenario - Environment with random scattered obstacles

The first scenario was composed by randomly scattering obstacles, filling 20% of the environment. This setting aims to measure the ability of a path planning strategy to navigate over environments with multiple possible paths and different geometries of obstacles. The RRT and the CGPlan also counted with a post-processing path smoothing technique to refine the final path, therefore, the smoothed path is represented with a solid line while the raw path is represented with a dotted line. After 30 executions, the best paths found in this scenario by each of the techniques can be seen in Figures 4.1 and 4.2.



(a) A* final path. (b) Bug2 final path. (c) GA best path found.

Figure 4.1: *Best solutions found by A* (a), Bug2 (b), and GA (c) after 30 executions.*



(a) RRT best path found. (b) CGPlan best path found.

Figure 4.2: *Best solutions found by RRT (a), and CGPlan (b) after 30 executions.*

After 30 executions of each technique on the first scenario a chart on the solution quality can be made, regarding the final path length. A box plot illustrating this comparison can be found in Figure 4.3.

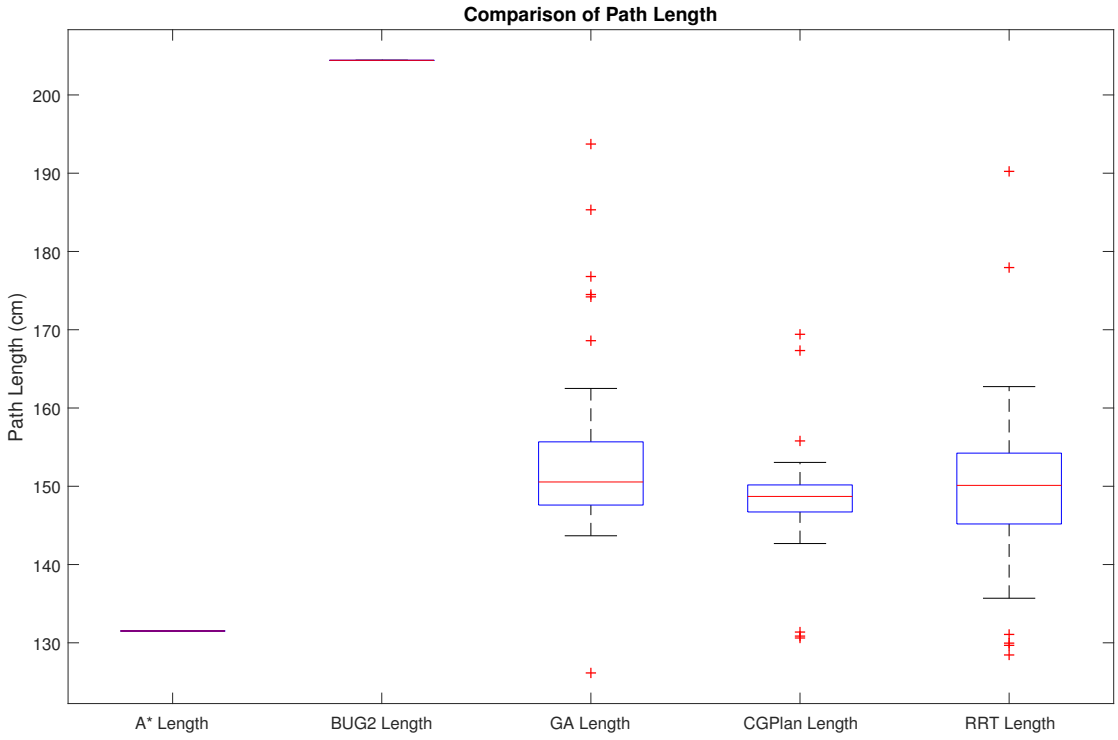


Figure 4.3: *Overall results for path length comparison after 30 runs.*

In this experiment the A* was able to reach the optimal solution, regarding path length, while the BUG2 presented the worst solution quality of the comparison group. In between these two results, the GA, the CGPlan and the RRT presented reasonable solutions, with the CGPlan having the lower average path length and standard deviation,

being the most consistent of the three. Another chart about the solutions can now be made regarding the time consumption of each technique. Trying to improve the understanding on the time consumption, the chart in Figure 4.4 is shown in logarithm scale due to large differences between the time consumption of these techniques.

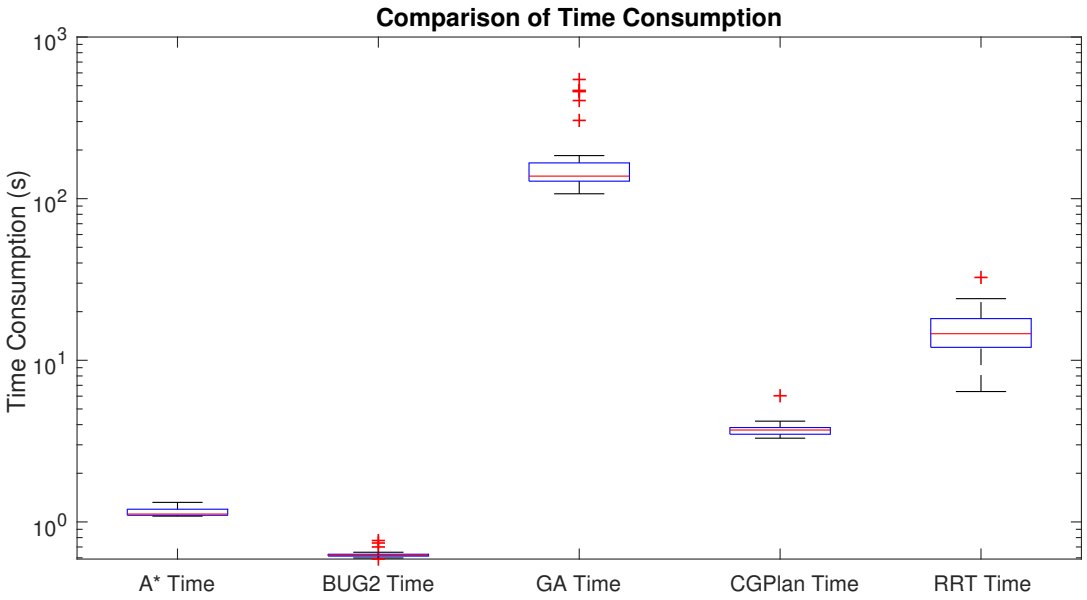


Figure 4.4: *Overall results for time consumption after 30 runs (logarithm scale).*

In this comparison of time consumption it is possible to notice an outstanding performance of the BUG2 algorithm, as expected for a reactionary technique. In contrast, the GA reached an impracticable consumption for applications with time requirements as the path planning of mobile agents. On the other hand, the A*, the CGPlan and the RRT presented a reasonable performance, with the A* and CGPlan requiring a time frame in the single digits. Again, the CGPlan showed to be competitive, and even better at some cases, then the commonly used path planning approaches.

In order to support the results obtained in this first scenario, a serial test was proposed. The comparison group was executed 30 times in 10 different random generated maps with 20% infill, except for the GA that was executed 10 times on each map, due to time constraints. The overall data gathered in this serial test was compressed in a chart for time and another for path length and can be seen in figures 4.6 and 4.5, respectively.

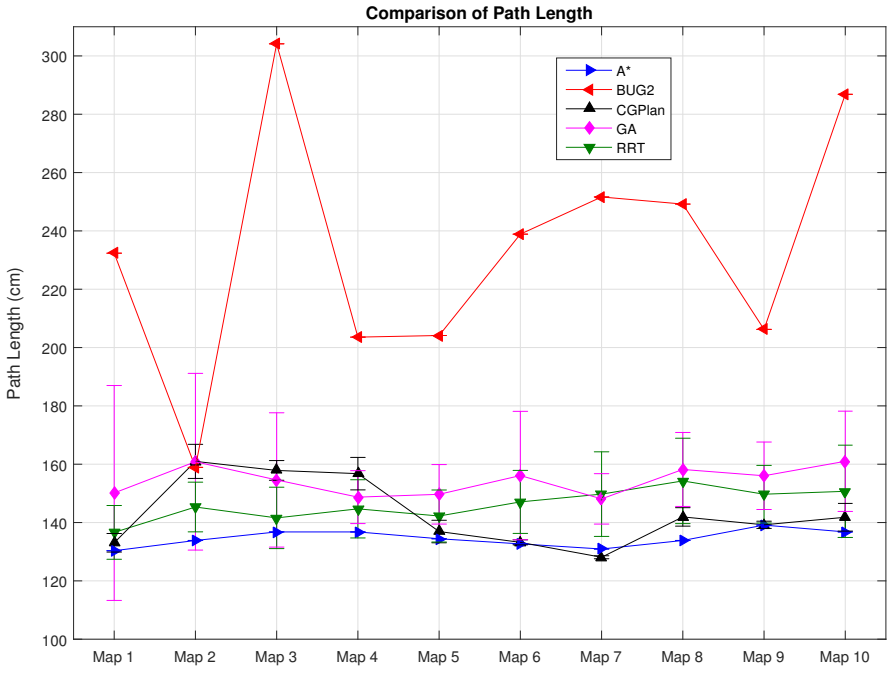


Figure 4.5: *Overall results for final path's length after 30 executions in 10 different maps.*

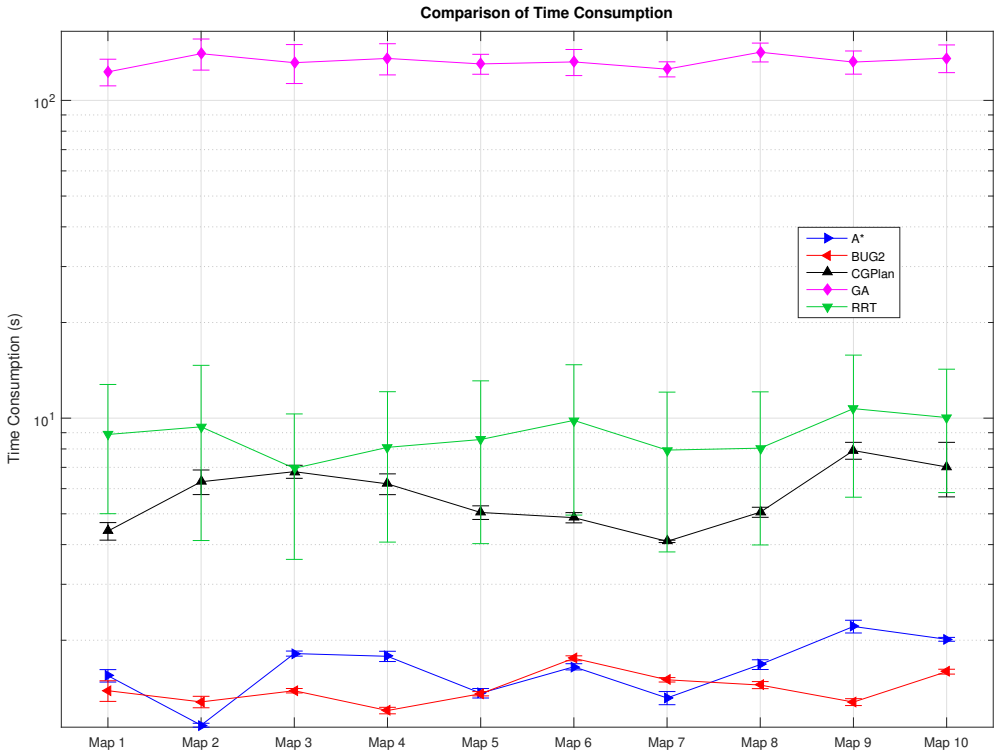


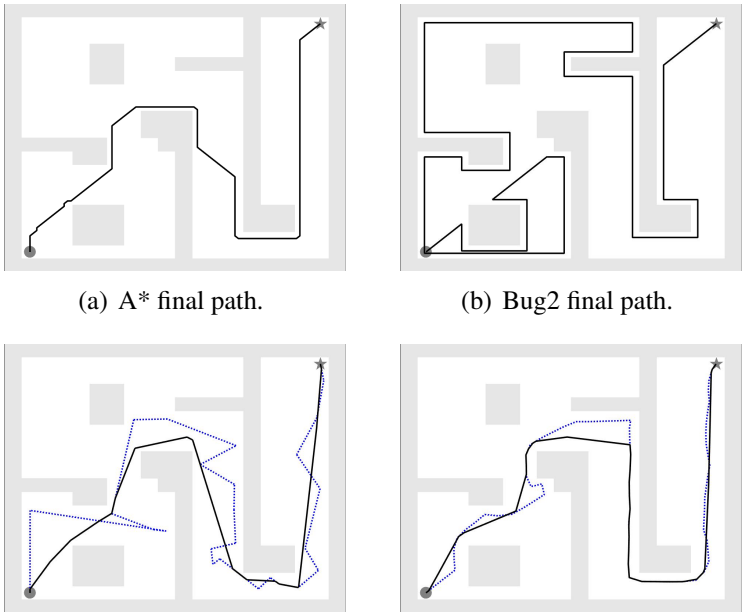
Figure 4.6: *Overall results for time consumption after 30 executions in 10 different maps (logarithm scale).*

With these results in mind, it can be noted that the A* and BUG2 present a

low time consumption in this scenario, whereas the CGPlan and RRT present a moderate consumption, with CGPlan being the lower of the two. The GA approach on the other hand, demonstrated huge time consumption as was reported in related works in the literature [35, 10, 9, 17] proving again to be an extremely inefficient approach to the path planning problem, showing sub-optimal solutions and being the most time intensive technique in the first scenario. Therefore, the GA was discarded for tests in the second scenario, as its executions were delaying the advance of the study.

4.2.2 Second scenario - Environment with indoor-like configuration

The second scenario is composed of obstacles arranged to simulate an indoor-like configuration. This setting aims to measure the ability of the chosen path planning strategies to navigate in this common environment for mobile agents. RRT and the proposed approach were again tested the aid of a post-processing path smoothing technique, with the smoothed path being represented by a solid line and the raw path represented by a dotted line. The best solutions found in this scenario after 30 executions by each of the techniques can be seen in Figure 4.7



(a) A* final path. (b) Bug2 final path.

(c) RRT best path found. (d) CGPlan best path found.

Figure 4.7: *The best solutions proposed by the chosen techniques after 30 executions.*

Again, after 30 executions of each technique on the scenario an observation regarding path length was made. A box plot illustrating this comparison can be found in figure 4.8.

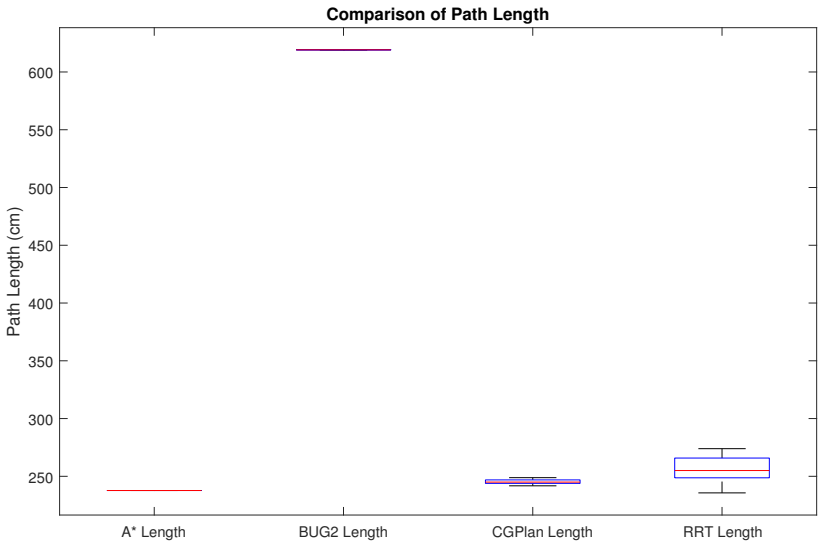


Figure 4.8: *Comparison of solution length after 30 executions.*

Analysing the chart in figure 4.8 it can be seen that, again, the A* approach reaches the shortest length between the compared techniques and the BUG2 shows the longest solution to the problem. Right next to the A* solution, the CGPlan and RRT approaches show similar results, with the CGPlan path planning being, again, the technique with the lower average length and with the least difference between the first and fourth quartiles, therefore being, comparatively, the more consistent approach between the two. Using the data gathered among 30 executions on this second scenario another observation regarding the time consumption of these techniques can be made. A box plot illustrating the results on this comparison can be found in figure 4.9.

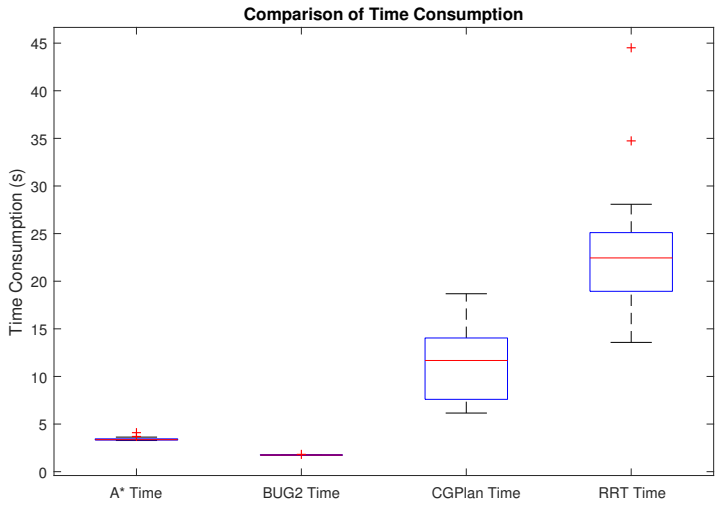


Figure 4.9: *Comparison of time consumption after 30 executions.*

Again, as shown in Figure 4.9, A* and BUG2 showed little consumption while the CGPlan and RRT showed moderate consumption of time in this scenario, with the proposed approach being the lower of the two.

4.3 Experiments Analysis.

After the analysis of the data extracted from the tests in scenario one and two, the following considerations for each technique can be made:

- The A* found the optimal solution with little time consumption, being the best global technique for either scenario;
- The BUG2 represented a trade-off, being a local technique with outstanding time efficiency but proposing poor solutions;
- The GA path planning had poor performance, presenting reasonable solutions regarding path length but, unfortunately, being the most time expensive solution. The time cost grew in an unbridled manner, forcing the removal of the technique from the tests to preserve the continuity of the study in reasonable time.
- RRT proved again to be another reliable global technique for path planning, presenting near-optimal solutions with moderate time consumption. The main issue with this approach is the inherit dependence of path smoothing techniques, as the random construction of the solution tree provides choppy paths.
- Finally, the proposed technique showed that it can reach solutions with near-optimal quality within a reasonable time span in both scenarios and even though being a local technique it outperformed the widely used global approach RRT.

4.4 Scalability Comparison

Based on the results obtained in the tests in scenarios 1 and 2, the best overall three techniques, A*, RRT, and CGPlan were chosen for a new round of tests for scalability. As the GA and BUG2 presented performance and solution quality problems, respectively.

By definition, scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to accommodate such growth [2]. In the context of path planning, scalability often describes the capability of a technique to handle environments of greater resolution, size, or dimension than usual.

The next tests were designed to measure the time consumption of each technique in the same environment (with the map presented in the second scenario) with four different resolutions. The indexes 1 through 4 represent the configurations:

1. Resolution: 1000x1000 - 100x100cm environment with 0.100x0.100cm pixels.
2. Resolution: 2000x2000 - 100x100cm environment with 0.050x0.050cm pixels
3. Resolution: 3000x3000 - 100x100cm environment with 0.033x0.033cm pixels
4. Resolution: 4000x4000 - 100x100cm environment with 0.020x0.020cm pixels

4.4.1 A* Scalability

A* showed promising results in the previous tests, presenting an optimal solution with the least time consumption, but this technique is well known for its bad scalability with resolution. The chart in Figure 4.10 presents the measured time consumption of the A* after 30 executions on each resolution.

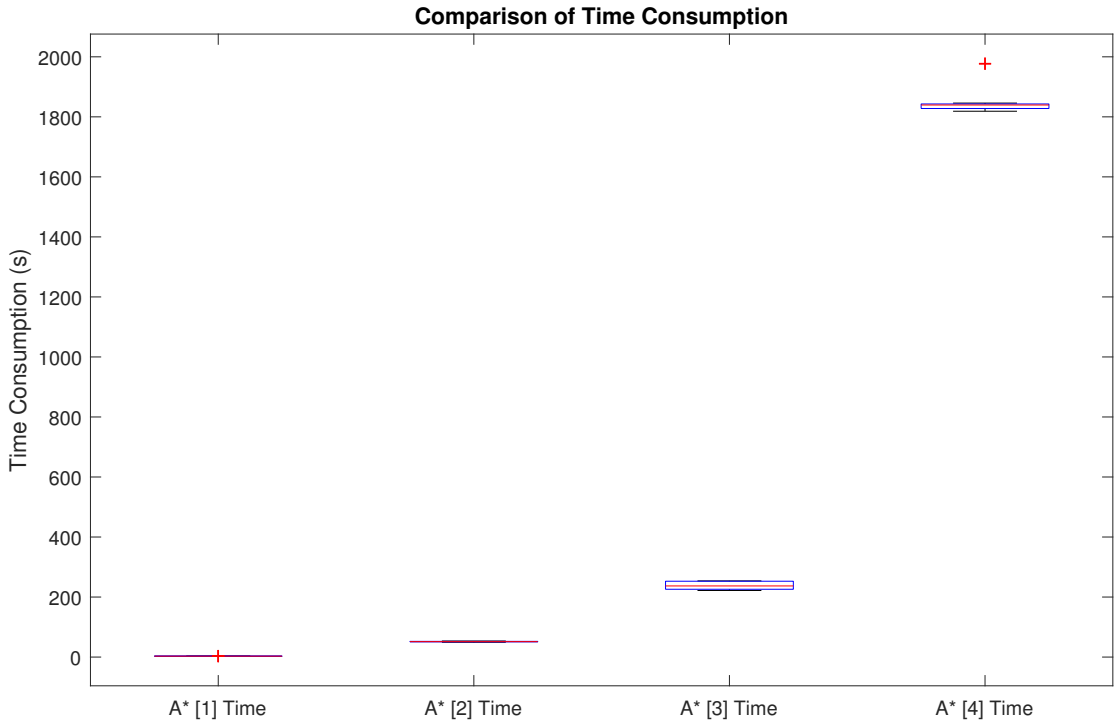


Figure 4.10: *Comparison of the A*'s time consumption after 30 executions on each resolution.*

As can be seen, raising the resolution of the environment can cause the time consumption to exponential increase, reaching infeasible values. Therefore, A * is not suitable for high-resolution applications when time efficiency is required, although presenting extremely good results in low-resolution applications.

4.4.2 RRT Scalability

The RRT also showed good results in the previous tests, presenting a near optimal solution with reasonable time consumption and, as a sampling-based algorithm,

it is expected to work well for high-resolution environments, because unlike grid-based algorithms, its running time is not (explicitly) exponentially dependent on the resolution. The chart in Figure 4.11 presents the time consumption after 30 executions on each resolution.

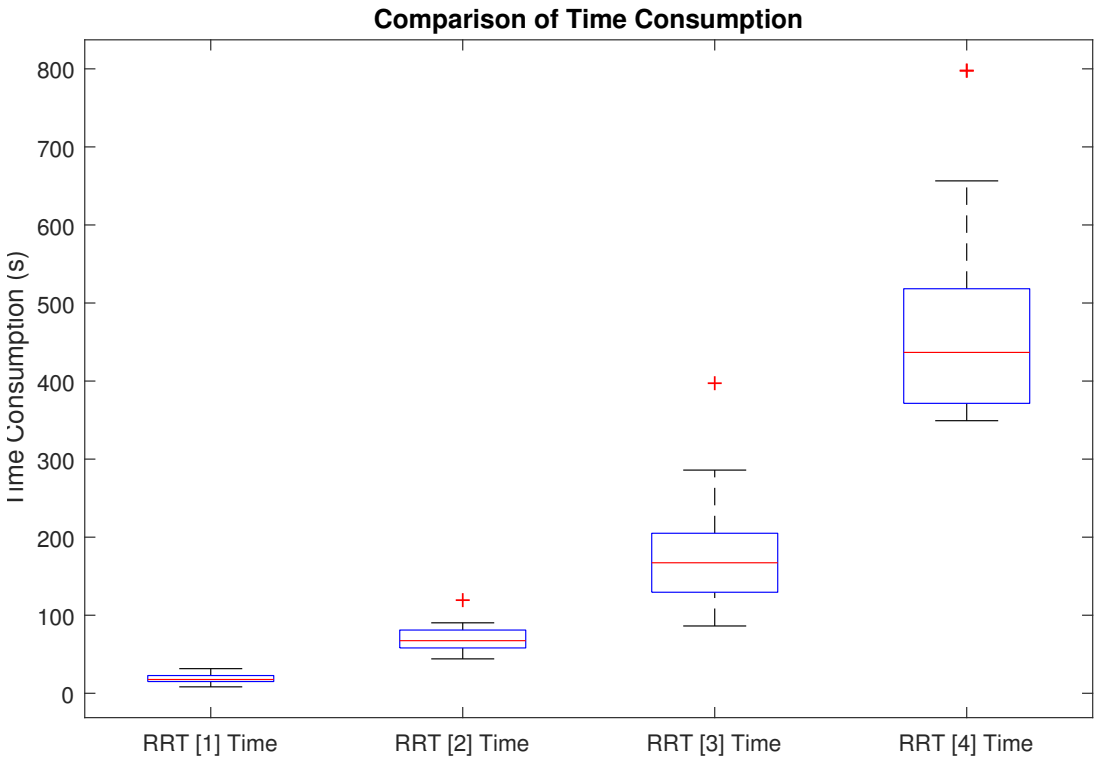


Figure 4.11: *Comparison of RRT's time consumption after 30 executions on each resolution.*

As can be seen, although presenting better scalability than A*, RRT still increases its time consumption considerably on higher resolutions. This phenomena can be explained by analysing the collision detection strategy of the RRT, witch scales poorly with the substantial increase of obstacles triggered by a incremental change of resolution.

4.4.3 CGPlan Scalability

At last, the CGPlan demonstrated good results on previous tests, presenting a near optimal solution within reasonable time consumption. Scalability to larger resolutions are expected to be achieved due to the constructive search technique, which decomposes the global search into a series of smaller successive searches, significantly reducing computation. The chart in Figure 4.12 presents the time consumption after 30 executions on each resolution.

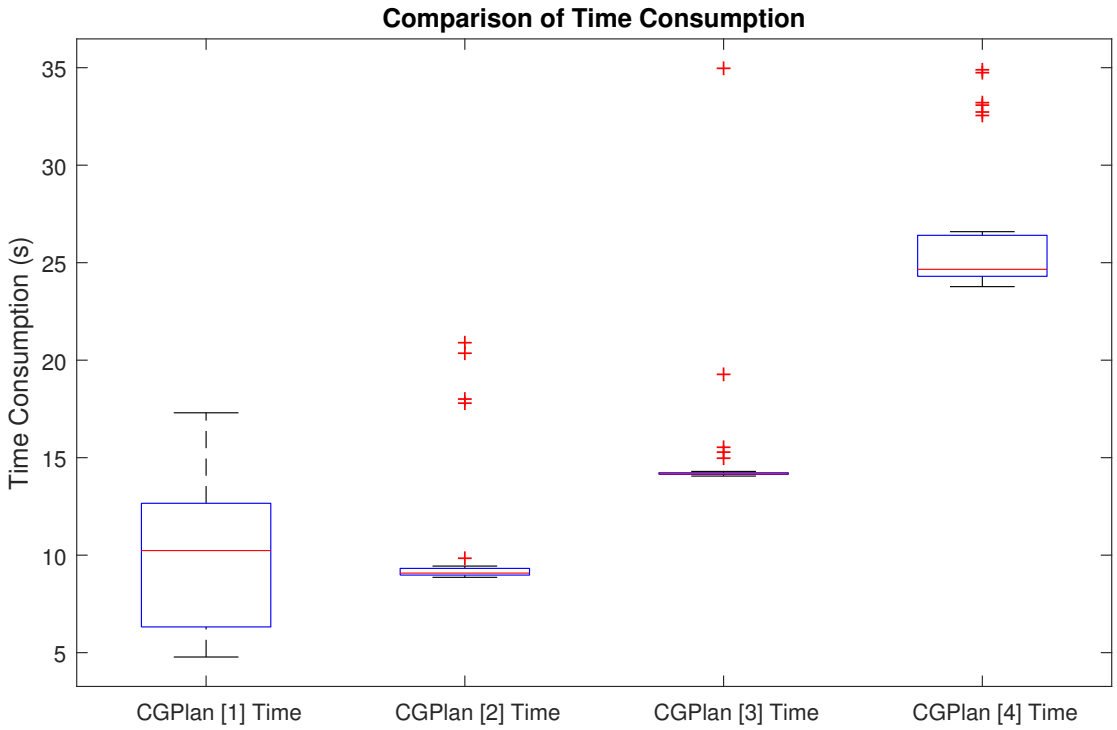


Figure 4.12: *Comparison of CGPlan's time consumption after 30 executions on each resolution.*

As can be seen, the increase in environment resolution slightly affects the time consumption of the proposed approach, reaching approximately 1.35% and 5.8% of the time consumed on the highest resolution by A* and RRT, respectively. Interestingly, the increase of resolution between the first and second configuration resulted in a better performance of the algorithm. This is due the fact that a higher resolution of the distance map ends up improving the selection of better trajectories, slightly lowering the average time consumption and improving its consistency.

CHAPTER 5

Conclusion and Future Works

5.1 Conclusion

This work investigated the use of a constructive genetic planning (CGPlan) for solving the path planning problem with less resource consumption through different map resolutions, aiming at usage in mobile agents. The proposed technique was compared against four well known approaches and through a series of tests some conclusions about performance, solution quality, and scalability can be made.

As seen, discrete algorithms (such as A*) can effectively solve low-resolution environments, overlaying a grid on the environment and searching for the optimal path. Unfortunately, scaling this planning strategy into high-resolution environments has proved to be computationally intractable, especially when constant changes on the environment are detected. In turn, reactive approaches (such as Bug1 and Bug2) solve the planning problem circumnavigating obstacles, being scalable and extremely time efficient, but despite its performance, the solution is not guaranteed to be the minimum path and in some cases can become impracticable.

In contrast, sampling-Based algorithms such as RRT can solve the planning problem in continuous environment with reasonable time and presenting near optimal solutions. This class of algorithms is also known to be better than grid-based approaches at higher resolution, as running time is not (explicitly) exponentially dependent on the resolution of the environment. However, despite showing a lower rise in time consumption, RRT struggled in higher resolutions, increasing its time consumption considerably and reaching infeasible time on the highest resolution.

The CGPlan, on the other hand, is a new evolutionary approach based on the Compact Genetic Algorithm (cGA) that pursue efficient path planning in known and unknown environments. This technique was especially designed for local applications with restricted information but showed to be competitive in common global path planning problems, unlike most path planning approaches using the classic GA.

The main strength of this technique relies on the cGA's economic search paired with a constructive representation of the path, providing near-optimal solutions within a

reasonable time frame. Through simulations, the CGPlan also proved to be better suited for higher resolutions than commonly used techniques such as the A* and RRT path planning, while still maintaining competitive performance in low resolutions.

We hope that this interesting feature and the mild increase in time consumption when scaled to higher resolutions places the proposed technique in a favorable position among modern path planning approaches.

5.2 Future Works

This work showed the restrictions and advantages a new evolutionary local path planning technique, the CGPlan. In future works, we would like to extend the CGPlan to higher-dimension problems, such as aerial vehicles, robotic manipulators and more complex mobile robots due to the scalable performance presented in this work. Further improvements on the environment representation can also be done, including time-variant obstacles and probabilistic prediction to support dynamic applications, thus enabling the use of this technique in common real-world applications.

Future works can also explore on different rcGA operators and structures in order to improve the performance of the algorithm, since more than 80% of the overall time is consumed by the rcGA cycles.

Bibliography

[1] ANTONELLI, G.; CHIAVERINI, S.; FUSCO, G. **A fuzzy-logic-based approach for mobile robot path tracking**. *IEEE Transactions on Fuzzy Systems*, 15(2):211–221, 2007.

[2] BONDI, A. B. **Characteristics of scalability and their impact on performance**. In: *Proceedings of the 2nd international workshop on Software and performance*, p. 195–203. ACM, 2000.

[3] BUNIYAMIN, N.; NGAH, W. W.; WAN NGAH, W. A. J.; SARIFF, N.; MOHAMAD, Z. **A simple local path planning algorithm for autonomous mobile robots**. *International journal of systems applications, Engineering & development*, 5(2):151–159, 2011.

[4] CANNY, J.; REIF, J. **New lower bound techniques for robot motion planning problems**. In: *Foundations of Computer Science, 1987., 28th Annual Symposium on*, p. 49–60. IEEE, 1987.

[5] FU, L.; SUN, D.; RILETT, L. R. **Heuristic shortest path algorithms for transportation applications: state of the art**. *Computers & Operations Research*, 33(11):3324–3343, 2006.

[6] HARIK, G. R.; LOBO, F. G.; GOLDBERG, D. E. **The compact genetic algorithm**. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.

[7] HOANG, V.-D.; HERNÁNDEZ, D. C.; HARIYONO, J.; JO, K.-H. **Global Path Planning for Unmanned Ground Vehicle based on Road Map Images**. p. 82–87, 2014.

[8] HOLLAND, J. H. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.** U Michigan Press, 1975.

[9] HU, Y.; YANG, S. X. **A knowledge based genetic algorithm for path planning of a mobile robot**. In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, p. 4350–4355. IEEE, 2004.

[10] ISMAIL, A.; SHETA, A.; AL-WESHAH, M. **A mobile robot path planning using genetic algorithm in static environment**. *Journal of Computer Science*, 4(4):341–344, 2008.

[11] KALA, R. **Code for Robot Path Planning using Genetic Algorithms**. p. 2–5, 2014.

[12] KALA, R. **Code for Robot Path Planning using Rapid Exploring Random Trees**. p. 2–5, 2014.

[13] KALA, R.; SHUKLA, A.; TIWARI, R.; RUNGTA, S.; JANGHEL, R. **Mobile Robot Navigation Control in Moving Obstacle Environment Using Genetic Algorithm, Artificial Neural Networks and A* Algorithm**. *2009 WRI World Congress on Computer Science and Information Engineering*, p. 705–713, 2009.

[14] KAVRAKI, L. E.; LAVALLE, S. M. **Motion planning**. In: *Springer handbook of robotics*, p. 139–162. Springer, 2016.

[15] LAVALLE, S. M. **Rapidly-Exploring Random Trees: A New Tool for Path Planning**. *Techreport*, 11, 1998.

[16] LAVALLE, S. M. **Planning Algorithms**. 2006.

[17] LI, Q.; ZHANG, W.; YIN, Y.; WANG, Z.; LIU, G. **An improved genetic algorithm of optimum path planning for mobile robots**. In: *Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on*, volume 2, p. 637–642. IEEE, 2006.

[18] LI, S.; DING, M.; CAI, C.; JIANG, L. **Efficient path planning method based on genetic algorithm combining path network**. In: *Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on*, p. 194–197. IEEE, 2010.

[19] LUMELSKY, V. J.; STEPANOV, A. A. **Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape**. *Algorithmica*, 2(1-4):403–430, 1987.

[20] MAHMOUDZADEH, S.; POWERS, D.; YAZDANI, A. **Differential evolution for efficient auv path planning in time variant uncertain underwater environment**. *arXiv preprint arXiv:1604.02523*, 2016.

[21] MININNO, E.; CUPERTINO, F.; NASO, D. **Real-valued compact genetic algorithms for embedded microcontroller optimization**. *IEEE Transactions on Evolutionary Computation*, 12(2):203–219, 2008.

[22] MOHANTY, P.; PARHI, D. **Controlling the Motion of an Autonomous Mobile Robot Using Various Techniques: a Review**. *Journal of Advanced Mechanical Engineering*, p. 24–39, 2013.

[23] OF ROBOTICS (IFR), I. F. **IFR World Robotics 2016**. http://www.ifr.org/, 2017. [Online; accessed January-2017].

[24] PALA, M.; ERAGHI, N. O.; LÓPEZ-COLINO, F.; SANCHEZ, A.; DE CASTRO, A.; GARRIDO, J. **Hctnav: A path planning algorithm for low-cost autonomous robot navigation in indoor environments**. *ISPRS International Journal of Geo-Information*, 2(3):729–748, 2013.

[25] PATEL, A. **A* Comparisons in game programming.** http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html, 2017. [Online; accessed January-2017].

[26] QU, H.; YANG, S. X.; WILLMS, A. R.; YI, Z. **Real-time robot path planning based on a modified pulse-coupled neural network model**. *Neural Networks, IEEE Transactions on*, 20(11):1724–1739, 2009.

[27] RAJA, P.; PUGAZHENTHI, S. **Optimal path planning of mobile robots: A review**. *International Journal of the Physical Sciences*, 7(9):1314–1320, 2012.

[28] REIF, J. H. **Complexity of the mover's problem and generalizations**. In: *Foundations of Computer Science, 1979., 20th Annual Symposium on*, p. 421–427. IEEE, 1979.

[29] ROBERGE, V.; TARBOUCHI, M.; LABONTÉ, G. **Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning**. *Industrial Informatics, IEEE Transactions on*, 9(1):132–141, 2013.

[30] SAMADI, M.; OTHMAN, M. F. **Global Path Planning for Autonomous Mobile Robot Using Genetic Algorithm**. *2013 International Conference on Signal-Image Technology & Internet-Based Systems*, p. 726–730, 2013.

[31] SCHWARTZ, J. T.; SHARIR, M. **On the "piano movers'" problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers**. *Communications on pure and applied mathematics*, 36(3):345–398, 1983.

[32] SCHWARTZ, J. T.; SHARIR, M. **On the piano movers' problem: V. the case of a rod moving in three-dimensional space amidst polyhedral obstacles**. *Communications on Pure and Applied Mathematics*, 37(6):815–848, 1984.

[33] SEDIGHI, K. H.; ASHENAYI, K.; MANIKAS, T. W.; WAINWRIGHT, R. L.; TAI, H.-M. **Autonomous local path planning for a mobile robot using a genetic algorithm**. In: *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, p. 1338–1345. IEEE, 2004.

[34] SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots**, volume 23. 2004.

[35] THOMAZ, C. E.; PACHECO, M. A. C.; VELLASCO, M. M. B. **Mobile robot path planning using genetic algorithms**. In: *Foundations and Tools for Neural Modeling*, p. 671–679. Springer, 1999.

APPENDIX A

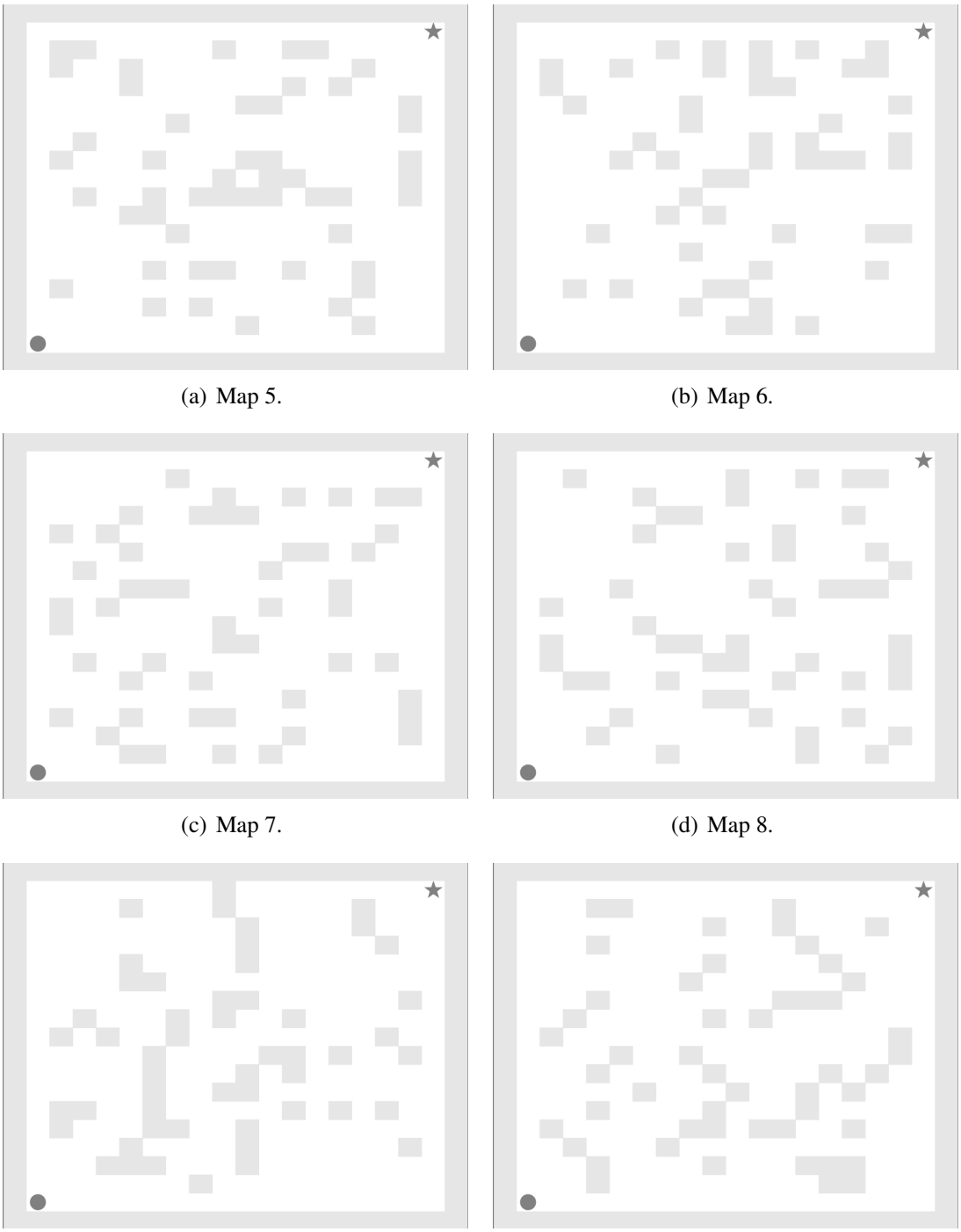
Random generated maps with 20% infill.

The 10 different maps generated for the tests in section 4.2.1 are displayed in this appendix for transparency.

(a) Map 1. (b) Map 2.

(c) Map 3. (d) Map 4.

Figure A.1: *Randomly generated maps used in the comparison.*



(a) Map 5. (b) Map 6.

(c) Map 7. (d) Map 8.

(e) Map 9. (f) Map 10.

Figure A.2: *Randomly generated maps used in the comparison.*

APPENDIX B

The path planning simulator.

During the development of this work a simulator was created using the Matlab environment. This united all tests into a single platform and provided the automation on serial tests and the creation of a data analysis tool. In figure B.1 the main window of the developed simulator can be seen.

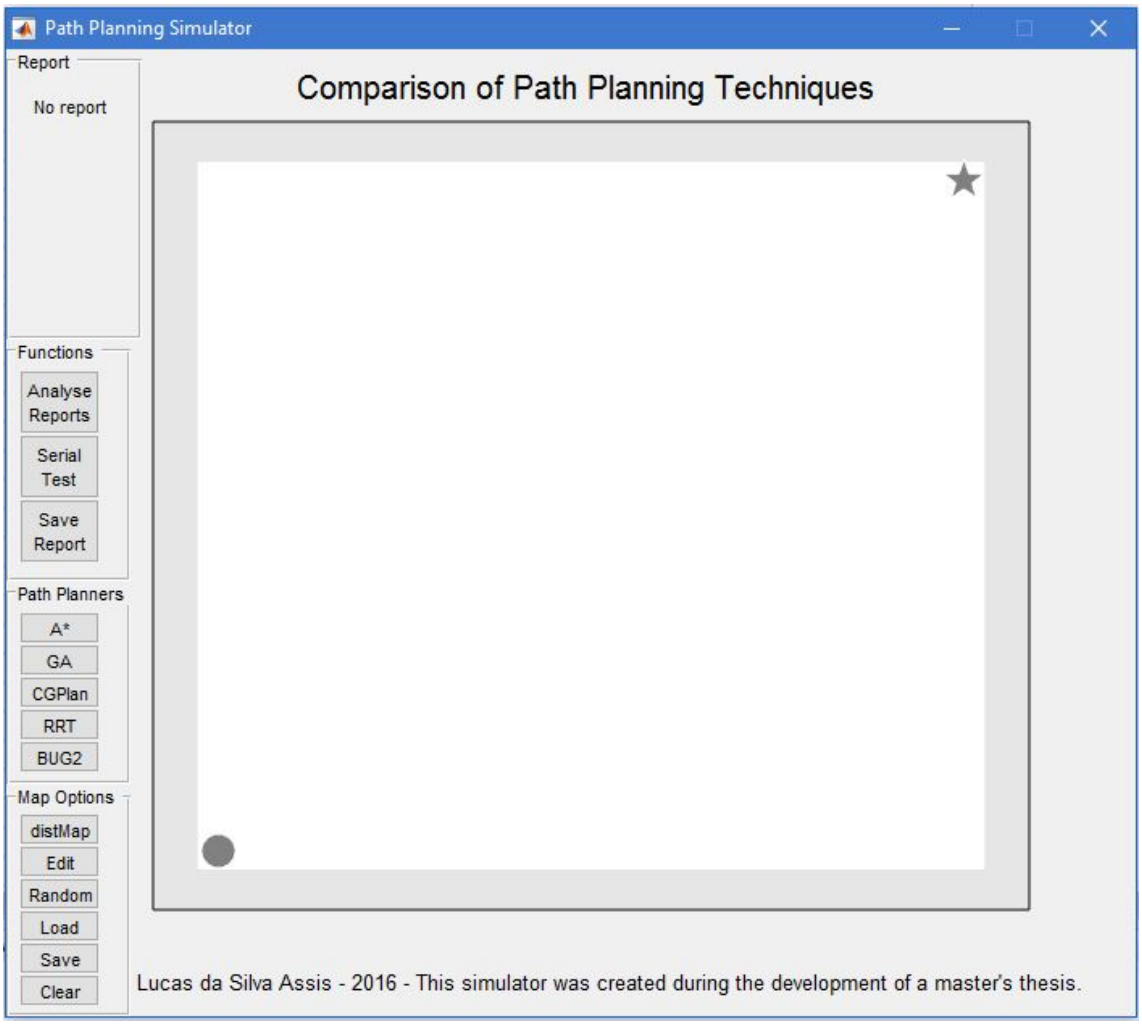


Figure B.1: *The main window of the simulator.*

The standard map have obstacles present only on its borders, but it is possible to create personalized maps using the *Edit* button. When triggered, this function places squared obstacles using a pre-defined grid. To stop the obstacle placement the *Edit* button must be pressed again. An example of this functionality can be seen in figure B.2. It is also possible to change the starting position (circle) and the final position (star) of the agent by clicking in its respectively symbol and then clicking in the desired position.

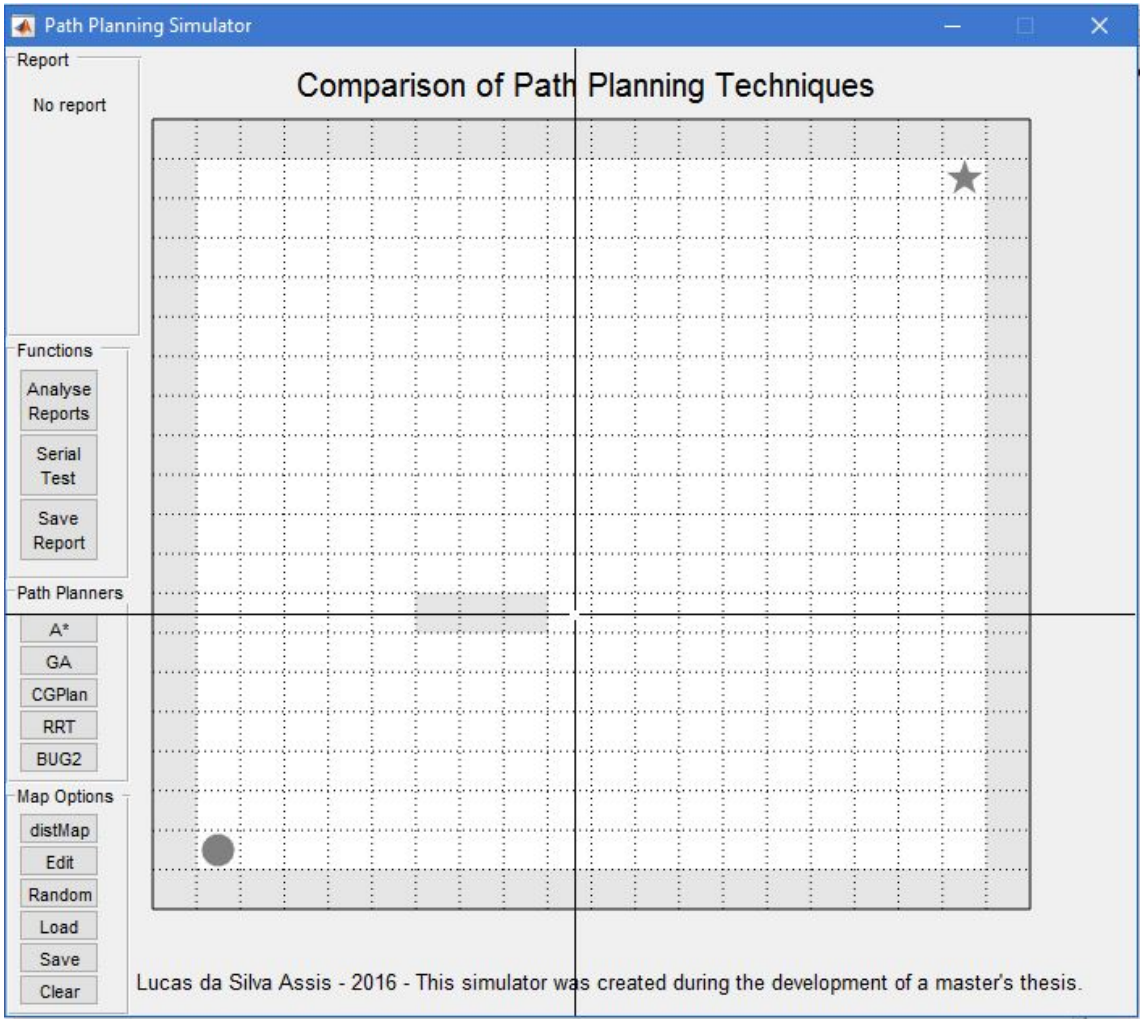


Figure B.2: *Editing a map in the simulator.*

In the Map Options section it is also possible to return to the standard map using the *Clear* button, calculate the Distance Transform of the current map using the *distMap* button, generate a random map given a infill percentage using the *Random* button, save the current map into a *.mat* file and load previously created maps. The load function also provides a list of the *.map* files present in the save folder. This functionality can be seen in figure B.3

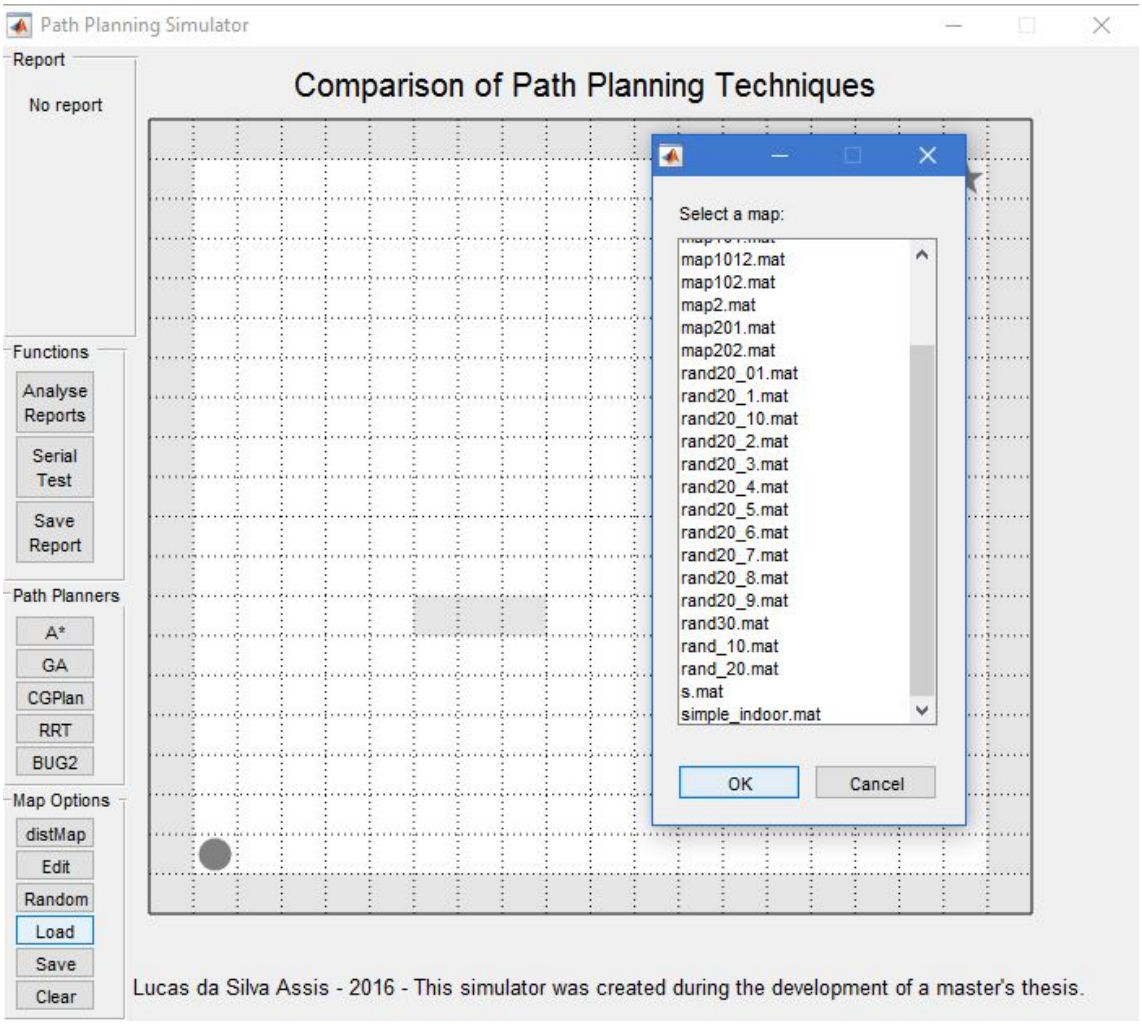


Figure B.3: *Loading a map into the simulator.*

After creating or loading a map is its possible to use any technique in the Path Planners section to find a possible path between the starting position (circle) and the final position (star). After choosing your technique, it is possible to see a report on performance and path quality in the Report section. This can be notice in figure B.4, when the CGPlan is tested in a complex indoor environment. The units are always in centimeters (for path length) and seconds (for time consumption).

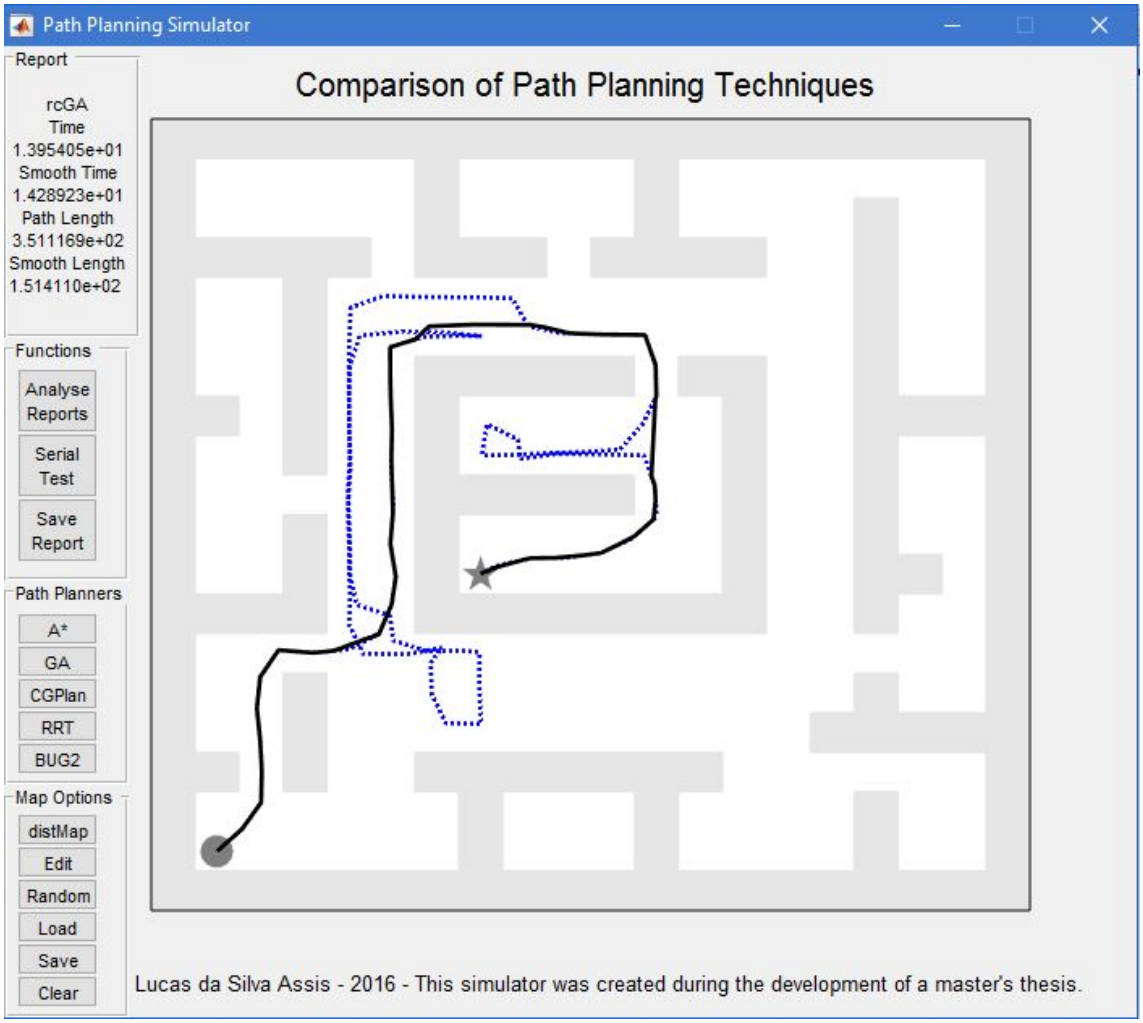


Figure B.4: *Report after a path planning execution.*

In this simulator you are also able to automate successive tests of a same technique in the current map. This feature is triggered with the *Serial Test* button and it is possible to choose from a list of path planing techniques, to input the number of desired executions and the test name. The test name is then used to create a folder with images representing the final solution of each execution as well as a *.txt* file with the report on each execution. Figure B.5 illustrate the creation of a serial test.

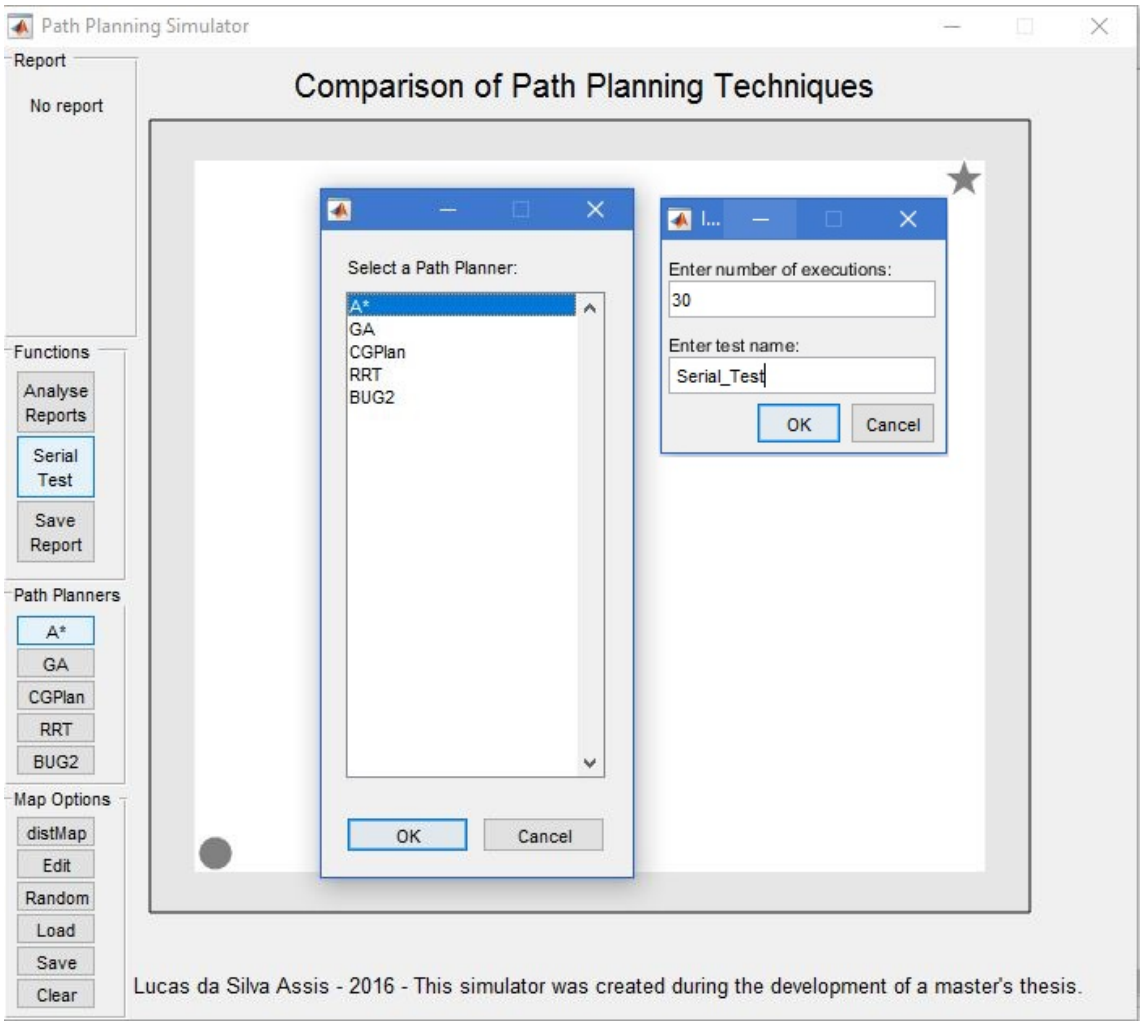


Figure B.5: *The serial test feature.*

After a serial test it is possible to analyse the gathered data and create charts using the *Analyse Reports* button. This function opens a new window in witch is possible to import the raw reports in *.txt* files, select the variable to analyse and plot either as a Box plot or a Error Line plot. This feature can be seen in figure B.6.

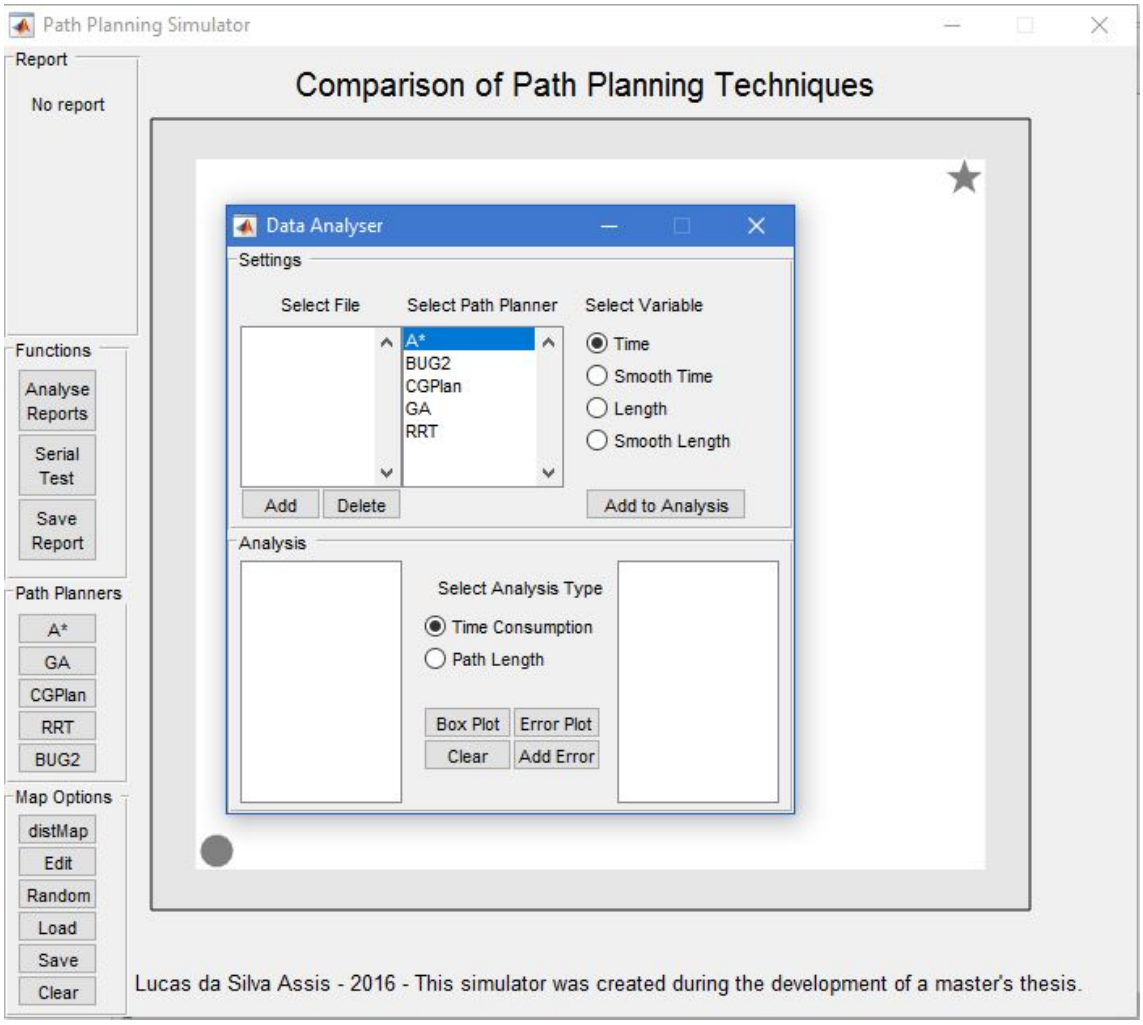


Figure B.6: *Data Analysis Tool.*

This simulator was of great importance for the development of the work, enabling the testing of different path planning techniques in the same environment and the automated analysis of the gathered data.