Assisted Counterexample-Guided Inductive Optimization for Robot Path Planning

Mengze Li and Lucas Cordeiro
Department of Computer Science, University of Manchester, Manchester, UK

***Abstract*—This paper presents and evaluates a novel offline mobile robot path planning algorithm based on the Assisted Counterexample-Guided Inductive Optimization (ACEGIO) technique. ACEGIO employs a technique to assist the Counterexample-Guided Inductive Optimization (CEGIO) technique by requesting counterexamples from Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solvers to improve its efficiency and effectiveness. In particular, Gradient Descent (GD) technique is implemented as an auxiliary technique to CEGIO. GD-Assisted Counterexample-Guided Inductive optimization (ACEGIO-GD) has been successfully applied to obtain two-dimensional paths for autonomous mobile robots using off-the-shelf SAT and SMT solvers. Experimental results demonstrate that the ACEGIO-based path planning algorithm has substantial improvements in efficiency and effectiveness compared to the traditional CEGIO-based path planning algorithm, which allows generating the optimal paths for autonomous mobile robots with much less execution time. If compared to other traditional path planning optimization techniques (e.g., GA and PSO), the execution time of the proposed algorithm is relatively high, whereas its performance is stable, reliable and robust.**

I. Introduction

The last several decades have seen a growing trend towards automated services. Mobile robots are replacing humans in repeated and monotonous work, such as transforming shelves within warehouses [1]. The growing demand for autonomous mobile robots is why autonomous navigation, especially path planning, has a pivotal role in developing mobile robots [2]. Path planning is a computational problem to find a valid path from an initial location to the desired destination that avoids all possible obstacles in the motion space [3].

The existing literature provides a series of algorithms and solutions to path planning problems [4, 5, 6, 7, 8]. Araujo et al. [4] considered the path planning problem as an optimization problem. Here, a decision variable represents a given path, and the cost function appears as a specific criterion whose value needs to be optimized. An optimal path is achieved if the value of specific criteria such as distance and execution time is optimal. In addition, the robot movement must follow the optimal path, consisting of a sequence of points. To solve this optimization problem, algorithms, such as A* algorithm [9], genetic algorithm (GA) [5], particle swarm optimization (PSO) [6], ant colony optimization [7], gravitational search algorithm (GSA) [8], can be employed. Due to the efficiency of these optimization methods, they can be applied to online path planning mode, which allows path planning to be performed during the robot movement. However, these methods have an uncertain possibility that global optimality of the robot path cannot be achieved [4].

Araujo et al. [10, 11] show that Counterexample Guided Inductive Optimization Algorithm (CEGIO) ensures global optimality of various non-trivial functions classes, with better effectiveness than traditional optimization technique. In terms of the established description about CEGIO [4, 11], the authors employ Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) [10]. The main idea of CEGIO is that it is executed iteratively by SAT/SMT solvers to generate counterexamples. These counterexamples are employed to update both decision variables and the cost function and guide global optimality. However, based on the experimental results provided in prior work [4], CEGIO is only applied to offline path planning mode due to high time consumption on CEGIO-based path planning applications which thus requires path planning to be performed before robot movement [12]. To improve the CEGIO efficiency, there exist two main approaches: develop more efficient optimization algorithms on top of SAT or SMT solvers [13] or employ a more efficient optimization algorithm to assist CEGIO. The second method is applied to this paper.
Contributions. This paper describes and evaluates a novel offline mobile robot path planning algorithm based on Assisted Counterexample-Guided Inductive Optimization (ACEGIO) technique. ACEGIO employs an auxiliary technique to assist the Counterexample-Guided Inductive Optimization (CEGIO) technique by reducing times of requesting counterexamples from SAT and SMT solvers to improve the efficiency and effectiveness of the CEGIO technique considerably. Therefore, the main original contributions of this paper are as follows. First, develop ACEGIO algorithm with Gradient Descent as the auxiliary algorithm to generate optimal two-dimensional paths for autonomous mobile robots in a predefined movement environment. Second, evaluate the proposed ACEGIO-based path planning algorithm and compare it with CEGIO-based path planning algorithm, which demonstrates that the efficiency and effectiveness of the CEGIO-based path planning algorithm are substantially improved by employing GD technique. Lastly, identify the proposed algorithm is more stable, reliable and robust, if compared to traditional path planning algorithms based on GA and PSO.

II. Background

A. Optimization Problems

Optimization problems and global optimal solutions can be defined as follows:

Definition 1. *An optimization problem consists of finding the global optimal solution from all feasible solutions [14].*

Optimization problems consist of minimizing the cost function. If optimization problems need to maximize the cost function, then minimize the negation of the cost function [14]. General optimization problems can be written as [14]

$$\begin{array}{ll} \underset{\mathrm{x}}{\text{minimize}} & \mathrm{f(x)} \\ \text{subject to} & \mathrm{g_i(x)} \leq 0, \mathrm{i} = 0, \ldots, \mathrm{m}, \\ & \mathrm{h_j(x)} = 0, \mathrm{j} = 0, \ldots, \mathrm{p} \end{array} \tag{1}$$

where $\mathrm{f(x)}$ is the objective function known as cost function, which needs to be minimized over a vector x composed by n variables known as decision variables. $g_i(x) \leq 0$ and $h_j(x) = 0$ are inequality constraints and equality constraints, respectively.

Definition 2. *A vector* x^* *such that* $g_i(x^*) \leq 0$ *and* $\mathrm{h_j}(\mathrm{x}^*) = 0$ *is a global optimal solution of objective function* $\mathrm{f(x)}$ *iff for* $\forall \mathrm{x}$ *such that* $g_i(x) \leq 0$ *and* $h_j(x) = 0$, $f(x^*) \leq f(x)$.

B. Path Planning for Mobile Robots

As one step in mobile robot navigation, efficient path planning algorithms considerably contribute to safe and effective mobile robot navigation. Depending on environment nature, robot's knowledge about the environment and completeness, path planning can be classified as static or dynamic, global or local, exact or heuristic, respectively [15]. The present work focuses on static, global, exact path planning algorithms in a bi-dimensional environment with the following properties: (1) the environment is fixed, where the starting position, the target position, obstacles and map information are unvarying; (2) the mobile robot has a priori knowledge about the environment; and (3) the algorithm finds an optimal path if one exists or proves no feasible solution exists.

Definition 3. *An optimal path consists of a sequence of points consecutively and sequentially connected by straight segments. The optimal path is valid for robots to move from a starting to a target position; it minimizes the cost function related to that path, such as energy consumption, distance [4].*

According to Definition 3 and the above properties, the proposed path planning algorithm is to find an optimal path, consisting of a set of straight segments that formed by points, from the source to the destination, which meets the path specification (obstacle avoidance) and minimizes the distance as the principal objective in the predefined environment.

C. Bounded Model Checking (BMC)

Prior work [11, 16] has provided definitions and descriptions of BMC applied to path planning [4]. Given a finite-state model of a system and a formal property, BMC systematically explores and checks a subset of states within a given bound [17]. A counterexample describes the execution path from the initial system state to property violation state [17]. The counterexample contains diagnostic information that plays a vital role in debugging. BMC converts the system to a verification condition (VC) and checks the negation of a given property. The VC is satisfiable *iff* the property is violated in a reachable state; in this case, a counterexample is generated. If it is unsatisfiable, there is no error state within the bound.

D. Counterexample Guided Inductive Optimization (CEGIO)

CEGIO is a search-based optimization algorithm that executes successive verifications to reduce the optimization domain and achieve global optimality. During the CEGIO optimization process, counterexamples extracted from SAT/SMT solvers are employed. They are used to update domain boundaries and the optimal candidates, thereby constraining the state-space search and guiding it towards global optimality. Additionally, Araujo et al. [18] demonstrate that CEGIO has a solid ability to find the global minima of optimization problems. The study also presents three variants of CEGIO, which are the Generalized CEGIO (CEGIO-G), the Simplified CEGIO (CEGIO-S), and the Fast CEGIO (CEGIO-F), respectively [11, 18]. CEGIO-F is the fastest among these three variants, while it can only be applied to convex functions [19].

E. Assisted Counterexample Guided Inductive Optimization

Assisted Counterexample Guided Inductive Optimization (ACEGIO) combines CEGIO with an auxiliary algorithm, which significantly improves the optimization efficiency without restricting the range of problems CEGIO can be applied to. The selected auxiliary algorithm must have fast optimization speed, while it probably cannot achieve global optimality. In contrast, CEGIO can ensure global optimality while it requires huge execution time and computational memory. ACEGIO relies on the advantages of both auxiliary algorithm and CEGIO to preserve the optimization ability and application range of CEGIO and improve the efficiency by reducing times of the most time-consuming process in CEGIO, which is requesting counterexamples from SAT and SMT solvers. CEGIO only uses counterexamples to update domain boundaries and optimal candidates, while both auxiliary algorithm and counterexamples are employed in ACEGIO.

In particular, Gradient Descent (GD), an efficient optimization technique for optimizing convex functions [20], can be applied as an auxiliary algorithm to assist CEGIO. Given a differentiable cost function, GD can find the local minimum [21]. The basic idea of the general GD algorithm is to iteratively move towards the opposite direction of the function gradient at the current position, which is the direction of steepest descent at the current point. Precisely, the GD algorithm can be pictured as a hiker who wants to climb down a mountain (cost function) into a valley (local minimum). The direction of the steepest descent and the hiker's leg length determine each step at the current position. Each iteration of GD Optimization computes the cost function's gradient and moves in the opposite direction with a step. Thus, the hiker iteratively takes steps till reaching the valley (local minimum).

III. ACEGIO-BASED PATH PLANNING ALGORITHM

Here, ACEGIO-GD based path planning is applied to generate a series of points on a path in a predefined environment with static obstacles. This method consists of two steps: (1) formulate the path planning problem as an optimization problem (i.e., model the environment, and static obstacles as constraints, set the cost function) (2) apply ACEGIO-GD to find the optimal path that satisfies the constraints.

A. Optimization problem formulation

In order to solve the path planning problem by the CEGIO algorithm, it is necessary to formulate the path planning problem as an optimization problem. Thus, its cost function and constraints need to be defined. Previous research [4] has proposed the following definitions (Definition 4, Definition 5) about cost function and constraints for path planning problems.

Definition 4. *Cost function: Define the starting position (S) and target position (T) as* $\mathrm{S} = \mathrm{P}_1$ *and* $\mathrm{T} = \mathrm{P}_\mathrm{n}$*, respectively. The objective is to find a decision variable matrix,* $\mathbf{L} = [P_1, P_2, \ldots, P_{n-1}, P_n]$*, which minimizes the cost function J(L). J(L) is the distance function to calculate the total distance of the path, also J(L) is the cost function of the path planning problem in this project.*

The cost function is defined as:

$$J(\mathrm{L}) = \sum_{\mathrm{i}=1}^{\mathrm{n}-1} \|\mathrm{P}_{\mathrm{i}+1} - \mathrm{P}_\mathrm{i}\|_2 \,, \tag{2}$$

where n is the number of points (including the starting position and the target position) that form the path. A smooth trajectory will be achieved if n is infinite [4].

Definition 5. *Constraints: Each straight segment on the path must not intercept any obstacle and must be within the predefined environment.*

According to Definition 1 and Eq. (2), the optimization problem for path planning can be written as:

$$\begin{array}{cc} \min_\mathrm{L} & \mathrm{J(L)}, \\ & \mathrm{p}_{\mathrm{i}\lambda}(\mathrm{L}) \notin \mathbb{O} \\ \text{s.t.} & \mathrm{p}_{\mathrm{i}\lambda}(\mathrm{L}) \in \mathbb{E} \\ & \mathrm{i} = 1, \ldots, \mathrm{n}-1, \end{array} \tag{3}$$

where $\mathbb{O}$ and $\mathbb{E}$ represent obstacles and points within environment limits, resp.; n is the number of points that compose the path; and $\mathrm{p}_{\mathrm{i}\lambda}(\mathrm{L})$ is all points to the i-th path straight segment defined by vector L. Each $\mathrm{p}_{\mathrm{i}\lambda}(\mathrm{L})$ point is defined as [4]:

$$\mathrm{p}_{\mathrm{i}\lambda}(\mathrm{L}) = (1-\lambda)\mathrm{P}_\mathrm{i} + \lambda \mathrm{P}_{\mathrm{i}+1}, \forall \lambda \in [0,1]. \tag{4}$$

After defining the cost function and constraints, movement environment ($\mathbb{E}$) and static obstacles ($\mathbb{O}$) need to be encoded. In this paper, the environment of movement is modelled in a two-dimensional Cartesian system as rectangle with lower and upper boundaries. Each point on path must be within this rectangle. As for obstacles ($\mathbb{O}$), one circle is used to model one obstacle, such that a centre of circle is the geometric centre of a physical obstacle, radius of circle is the largest distance from the centre to edge of the physical obstacle. This ensures all the points formed physical obstacles are surrounded by the corresponding circles. Therefore, for each obstacle, constraints $\mathrm{p}_{\mathrm{i}\lambda}(\mathrm{L}) \notin \mathbb{O}$ such that $\mathrm{i} = 1, \ldots, \mathrm{n}-1$ can be written as:

$$(\mathrm{x}_{\mathrm{i}\lambda} - \mathrm{x}_0)^2 + (\mathrm{y}_{\mathrm{i}\lambda} - \mathrm{y}_0)^2 \geq (\mathrm{r} + \sigma)^2, \tag{5}$$

where $p_{i\lambda} = (x_{i\lambda}, y_{i\lambda})$, and σ is a safety margin, $(\mathrm{x}_0, \mathrm{y}_0)$ is the center of an obstacle, r is the radius of the obstacle.

B. ACEGIO-GD

Previous research has demonstrated that the path planning cost function (Eq. (2)) is convex. Thus ACEGIO-GD, which is applied to solve the path planning problem, is developed based on CEGIO-F due to its efficiency.

Algorithm 1 presents steps of ACEGIO-GD. ACEGIO-GD takes the cost function $\mathrm{f}(\mathbf{x})$, the space for constraint set Ω, a desired precision ϵ and the Gradient Descent function $\mathrm{G}(\mathbf{x})$as input. The output includes the optimal decision variable vector $\mathbf{x}^*$, and optimal cost function value $\mathrm{f}(\mathbf{x}^*)$. In order to generate more optimised candidates and eventually achieve global optimal solution, the core of ACEGIO-GD is to iteratively verify if literal l_{optimal} is satisfied for current optimal candidate $\mathrm{f}\left(\mathbf{x}^{(\mathrm{i}-1)}\right)$. The literal l_{optimal} is described as:

$$l_{\text{optimal}} \Leftrightarrow f\left(\mathbf{x}^{(\mathrm{i})}\right) \geq \mathrm{f}\left(\mathbf{x}^{(\mathrm{i}-1)}\right). \tag{6}$$

The directives *ASSUME* sets the constraints and *ASSERT* verifies the specific property. If the verification in line 9 of Algorithm 1 fails, the new decision variables vector$\mathbf{x}^*\left(\mathbf{x}^{(\mathrm{i})}\right)$ and new optimal candidate $\mathrm{f}(\mathbf{x}^*)\left(\mathrm{f}\left(\mathbf{x}^{(\mathrm{i})}\right)\right)$ are generated either by extracting from counterexamples or by applying Gradient Descent. The new optimal candidate is smaller than previous optimal candidate $\mathrm{f}\left(\mathbf{x}^{(\mathrm{i}-1)}\right)$ and closer to global optimality. Otherwise, if l_{ontimal} is satisfiable, $\mathbf{x}^*$ and $\mathrm{f}(\mathbf{x}^*)$ would not be updated and would remain $\mathbf{x}^{(\mathrm{i}-1)}$ and $f\left(x^{(i-1)}\right)$, respectively. As a result, in each iteration, either a new optimal candidate, which is closer to global optimality is found, or the state-space is updated. After substantial times of iterations, the optimal value and optimal solution are generated.

The space for constraint set is defined as Ω^k, such that $\mathrm{k} = \log \mathrm{p}$, where k is the number of decimal places of the points coordinate values (i.e., if p is 1, then k is 0 and coordinates are considered as integers). Precision p is initialized with value 1 and is updated by multiplying p by 10. Finally, ACEGIO-GD ends if precision p reaches the desired precision η or the global optimal solution is found.

The difference between CEGIO-F and ACEGIO-GD is that ACEGIO-GD additionally employs Gradient Descent to calculate optimal candidates iteratively. For a given precision in ACEGIO-GD, requesting counterexamples only happens once when it reaches the inner do-while loop, resulting in an optimal candidate. Then GD is iteratively used to achieve new optimal candidates closer to the global optimal solution for the given precision. Since the execution speed of employing GD is much faster than extracting counterexamples from SAT or SMT solvers, contrary to CEGIO-F, ACEGIO-GD dramatically reduces the time consumption.

C. ACEGIO-based Path Planning Algorithm

Algorithm 2 summarised the process of applying ACEGIO-GD to the path planning problem. Similar to ACEGIO-GD, if the verification in line 11 fails, a new decision variable vector $\boldsymbol{L}^*$ and a new optimal candidate $\mathrm{J}(\boldsymbol{L}^*)$ are generated either by extracting from counterexamples or by applying GD. In this case, a more optimized path can be generated. If J_{optimal} is satisfiable, $\boldsymbol{L}^*$ and $\mathrm{J}(\boldsymbol{L}^*)$ would not be updated and would remain $\boldsymbol{L}^{(\mathrm{i}-1)}$ and $\mathrm{J}\left(\boldsymbol{L}^{(i-1)}\right)$, respectively. As a result, either a new optimal candidate path closer to global optimality is found in each iteration, or the state-space is updated.

For a given *n* and *p*, if it is not possible to find a new optimal candidate, number of points *n* is updated by adding 1 to *n*. If $\neg \mathrm{J}_{\text{optimal}}$ is not consecutively satisfiable, then update precision *p* by multiplying *p* by 10. And the number of decimal places of the points coordinate values *k* is updated by adding 1 to

Algorithm 1: ACEGIO-GD

```
Input  : Cost function $f(\mathbf{x})$, the space for constraint set $\Omega$, and a desired
         precision $\eta$ and a Gradient Descent function $G(\mathbf{x})$
Output: The optimal decision variable vector $\mathbf{x}^*$ and the optimal cost
         function value $f(\mathbf{x}^*)$
1  Initialise $f\left(x^{(0)}\right)$ randomly and $i = 1$;
2  Initialise the precision variables with $p = 1$ and $k = 0$;
3  Declare the auxiliary variables $x$ as non-deterministic integer variables;
4  while $p \leq \eta$ do
5      Define bounds for $x$ with the ASSUME directive, such that $\mathbf{x} \in \Omega^k$;
6      Describe a model for $f(\boldsymbol{x})$;
7      do
8          Constrain $f\left(x^{(i)}\right) < f\left(x^{(i-1)}\right)$ with the ASSUME directive;
9          Verify the satisfiability of $l_{\text{optimal}}$ given by Eq. (6). with the
             ASSERT directive;
10         if $\neg l_{optimal}$ is satisfiable then
11             Update $x^* = x^{(i)}$ and $f(x^*) = f\left(x^{(i)}\right)$ based on the
                 counterexample;
12             Do $i = i + 1$;
13             Update $x^i = G\left(x^{(i-1)}\right)$ and
                 $f\left(x^i\right) = f\left(G\left(x^{(i-1)}\right)\right)$;
14         end
15     while $\neg l_{optimal}$ is satisfiable;
16     Do $k = k + 1$, $p = p * 10$;
17     Update the set $\Omega^k$;
18     Update the precision variable, $p$;
19     Set $x^i$ as non-deterministic integer variables;
20 end
21 $x^* = x^{(i-1)}$ and $f\left(x^{(i-1)}\right)$;
22 return $x^*$ and $f(x^*)$;
```

k, since $k = log\ p$. Therefore, the precision is increased by adding one decimal place in the coordinate values.

Additionally, ACEGIO-GD takes Gradient Descent function $G(\mathbf{L})$ as input. Gradient Descent algorithm is an optimization algorithm capable of finding local optimality of optimization problems whose cost function is differentiable. The basic idea is to move towards the direction of the steepest descent iteratively. Gradient Descent is defined as follows.

Definition 6. *For a multi-variable function* $\mathrm{F(x)}$ *which is differentiable in its whole domain. For every point* $\mathbf{x_n}$ *of the function,* $\mathrm{F}(\mathbf{x}_\mathrm{n})$ *decreases fastest if one moves from* $\mathbf{x_n}$ *against the gradient of* F *at* $\mathbf{x_n}$*, which is* $-\nabla\mathrm{F}(\mathbf{x}_\mathrm{n})$.

For every point $\mathbf{x_n}$ of the function, if $\mathbf{x}_{\mathrm{n}+1} = \mathbf{x}_\mathrm{n} - \gamma_\mathrm{n}\nabla\mathrm{F}(\mathbf{x}_\mathrm{n}), \mathrm{n} \geq 0$, then

$$\mathrm{F}(\mathbf{x}_\mathrm{n}) \geq \mathrm{F}(\mathbf{x}_{\mathrm{n}+1}), \tag{7}$$

where γ_n is the value of step size, which is allowed to change at every iteration, $-\nabla F(\mathbf{x}_n)$ is the steepest direction of function $\mathrm{F(x)}$ at current point $\mathbf{x_n}$. According to Eq. (3), the path planning cost function can be written as:

$$\begin{aligned} &J(x_1, y_1, x_2, y_2, \ldots, x_n, y_n) = \\ &\sum_{i=1}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}, \end{aligned} \tag{8}$$

where i is the index of points, $(\mathrm{x_i}, \mathrm{y_i})$ is point on path, n is the number of points on the path. Assume every point on the path $(\mathrm{x_i}, \mathrm{y_i}) \in \mathbb{R}^2$ thus the cost function of the path planning problem can be considered as a differentiable function with 2n variables. Combining Definition 6 and following

$$\mathbf{L}^{(j+1)} = \mathrm{G}\left(\mathbf{L}^{(\mathrm{j})}\right). \tag{9}$$

Algorithm 2: ACEGIO-based Path Planning Algorithm

```
Input  : Cost function $J(\mathbf{L})$, a set of obstacles constraints $\mathbb{O}$, a set of
         environment constraints $\mathbb{E}$, which define $\Omega$ and a desired precision $\eta$,
         and a Gradient Descent function $G(\mathbf{L})$
Output: The optimal path $\mathbf{L}^*$ and the optimal cost function value $J(\mathbf{L}^*)$
1  Initialise $J\left(\boldsymbol{L}^{(0)}\right)$ randomly;
2  Initialise precision variable with $p = 1, k = 0\ i = 1$;
3  Initialise precision variable with $n = 1$;
4  Declare decision variables vector $\boldsymbol{L}^i$ as non-deterministic integer variables;
5  while $k \leq \eta$ do
6      Define upper and lower limits of $\boldsymbol{L}$ with directive ASSUME, such as
         $L \in \Omega^k$;
7      Describe the objective function model $J(\boldsymbol{L})$;
8      do
9          do
10             Define the constraint $J\left(\boldsymbol{L}^{(i)}\right) < J\left(\boldsymbol{L}^{(i-1)}\right)$ with
                 directive ASSUME;
11             Verify the satisfiability of $J_{\text{optimal}}$ given by Eq. (6) with
                 directive ASSERT;
12             if $\neg l_{optimal}$ is satisfiable then
13                 Update $\boldsymbol{L}^* = \boldsymbol{L}^{(i)}$, $J(\boldsymbol{L}^*) = J\left(\boldsymbol{L}^{(i)}\right)$ based on the
                     counterexample;
14                 Do $i = i + 1$;
15                 Update $\boldsymbol{L}^{(i)} = G\left(\boldsymbol{L}^{(i-1)}\right)$, and
                     $J\left(\boldsymbol{L}^{(i)}\right) = J\left(\mathrm{G}\left(\boldsymbol{L}^{(i-1)}\right)\right)$;
16             end
17         while $\neg J_{optimal}$ is satisfiable;
18         if $\neg J_{optimal}$ is not consecutively satisfiable then
19             break ;
20         else
21             Update the number of points, $n$;
22         end
23     while TRUE;
24     Do $k = k + 1$;
25     Update the set $\Omega^k$;
26     Update the precision variable, $p$;
27     Set $\boldsymbol{L}^i$ as non-deterministic integer variables;
28 end
29 $\boldsymbol{L}^* = \boldsymbol{L}^{(i-1)}, J(\boldsymbol{L}^*) = J\left(\boldsymbol{L}^{(i-1)}\right)$;
30 return $\boldsymbol{L}^* J(\boldsymbol{L}^*)$;
```

$\mathbf{L}(\mathrm{j}+1)$ can be written as:

$$\mathbf{L}^{(\mathrm{j}+1)} = \mathrm{G}\left(\mathbf{L}^{(\mathrm{j})}\right) = \mathbf{L}^{(\mathrm{j})} - \gamma_\mathrm{j}\nabla J\left(\mathbf{L}^{(\mathrm{j})}\right), \tag{10}$$

where $\mathbf{L} = [\mathrm{x_1}, \mathrm{y_1}, \mathrm{x_2}, \mathrm{y_2}, \ldots, \mathrm{x_n}, \mathrm{y_n}]$, j is the index of iteration in Gradient Descent, $\mathbf{L}^{(\mathrm{j}+1)}$ is the new decision variable vector generated by Gradient Descent.

For each variable x" or y", the partial derivate can be written as:

$$\begin{aligned} \nabla\mathrm{J}(\mathrm{x_i}) =& \frac{\mathrm{x_i} - \mathrm{x_{i-1}}}{\sqrt{(\mathrm{x_i} - \mathrm{x_{i-1}})^2 + (\mathrm{y_i} - \mathrm{y_{i-1}})^2}} + \\ & \frac{\mathrm{x_i} - \mathrm{x_{i+1}}}{\sqrt{(\mathrm{x_i} - \mathrm{x_{i+1}})^2 + (\mathrm{y_i} - \mathrm{y_{i+1}})^2}} \\ \nabla\mathrm{J}(\mathrm{y_i}) =& \frac{\mathrm{y_i} - \mathrm{y_{i-1}}}{\sqrt{(\mathrm{x_i} - \mathrm{x_{i-1}})^2 + (\mathrm{y_i} - \mathrm{y_{i-1}})^2}} + \\ & \frac{\mathrm{y_i} - \mathrm{y_{i+1}}}{\sqrt{(\mathrm{x_i} - \mathrm{x_{i+1}})^2 + (\mathrm{y_i} - \mathrm{y_{i+1}})^2}} \end{aligned} \tag{11}$$

Gradient Descent G of the cost function J is defined as above equations. A new decision variable vector $\mathbf{L}^{(\mathrm{j}+1)}$ can be achieved by Eq. (10) and Eq. (11). Since every point $(\mathrm{x_i}, \mathrm{y_i})$ on the path is assumed in the range of $\mathbb{R}^2$, it is necessary to make the new calculated $\mathrm{x_i}$ and $\mathrm{y_i}$ keep k decimal places for a given precision p. Then a new optimal candidate based on the new decision variables can be generated by Gradient Descent.

The number of points on the path depends on the algorithm's efficiency and the number of obstacles that impact the algorithm's performance. As the number of points increases, the time and space complexity substantially increase, resulting in a significant execution time.

IV. Experimental Evaluation

A. Experimental Objectives and Description

This section evaluates the ACEGIO-based path planning (cf. Section III). There are two criteria for identifying the success of ACEGIO-based path planning algorithm: *effectiveness* and *efficiency*. Three experimental goals are presented:

EG1 **(effectiveness)** Does ACEGIO-based path planning algorithm generate the global optimal path or a path close to the global optimal path?;

EG2 **(efficiency)** Dooes ACEGIO-based path planning algorithm generate paths closer to the global optimality with less execution time comparing to CEGIO-based path planning?; and

EG3 **(state-of-the-art)** how does ACEGIO-based path planning algorithm compare to other state-of-the-art approaches?

To evaluate the effectiveness (EG1), all the experiments with different environment settings aim to generate a set of points that composed the global optimal path or a path close to the global optimal path from starting position to target position by ACEGIO-based path planning algorithm. As for the efficiency (EG2), the CEGIO-based path planning algorithm and ACEGIO-GD-based path planning algorithm must be evaluated via experiments with the same environment setting. The experimental results (e.g., time consumption, points on the path, distance) need to be evaluated and compared. Besides, to answer EG3, ACEGIO-based path planning algorithms need to be compared with various state-of-the-art path planning approaches in the same experimental setting. Therefore, Genetic Algorithm based path planning and Particle Swarm Optimization based path planning algorithm are selected and evaluated in this section. In summary, the path planning algorithms based on CEGIO, ACEGIO, Genetic Algorithm, Particle Swarm Optimization are labeled as Algorithms A, B, C, and D, respectively.

Figure 1 presents two environment settings that were designed with a meter as the measurement unit. The shape of both settings is square, whose side length is 10. The starting point (labeled as S in Figure 1) and a target point (labeled as T in Figure 1) in both settings are (1, 1) and (9, 9), respectively. The only difference is the obstacles. In the first setting (setting 1 in Figure 1), one obstacle is applied whose center (labeled as O in Figure 1) is (5,5) and radius is 2.5. In Setting 2, four obstacles are applied, centered in (4,3), (8,7), (2,8), (8,3) with 1.5, 1, 1, 0.5 as their radius, respectively. Besides, the safety margin in both settings is 0.5. Figure 1 shows the obstacles with the blue line and safety margin with the dotted red line.

For Algorithm A, B, C, and D, each of them is applied to both settings to answer EG1 and EG3. For each setting, Algorithm A and B are applied to compare their efficiency (EG2) and performance (EG1) in the same environment.

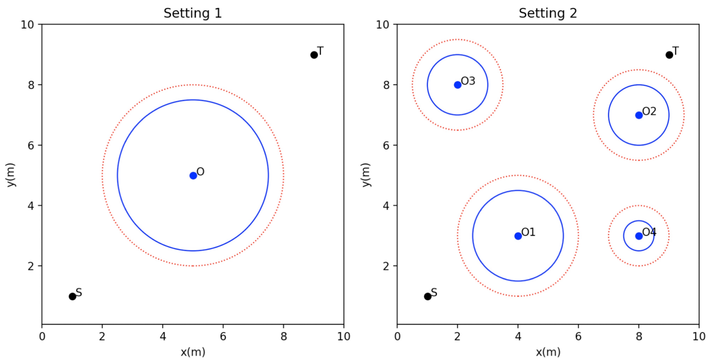

Fig. 1: Experimental environment settings.

B. Experimental Setup

The path planning experiments were conducted on a 2.3 GHz OCTA Intel Core i9 processor with 16GB of RAM, running macOS Catalina 10.15.6 64-bits. For each experiment, the execution time, which is the average of five executions, is measured in seconds based on CPU time. The maximum execution time for settings 1 and 2 is two days. Memory consumption is not restricted. ESBMC 6.4.0 [22] is selected as a verification tool since it was already applied to verify a wide range of applications [23, 24], and Boolector 3.0 [25] is chosen as an SMT solver for executing Algorithm A and B. As for applying Algorithm C and D, Matlab_R2021a is employed.

C. Experimental Results

This section presents experimental results of applying Algorithm A and Algorithm B to both Setting 1 and Setting 2, which aims to answer EG1 and EG2. Figure 2 presents the paths composed by sequences of points in Setting 1 and Setting 2, which are generated by Algorithm A and Algorithm B. For both settings, red paths in Figure 2 were obtained by Algorithm A, and the blue dotted paths were obtained by Algorithm B. For Setting 1, a path composed of five points (including starting point and target point) was generated by Algorithm A, and a path with nine points was generated by Algorithm B. As for Setting 2, a path with four points and a path with six points was obtained by Algorithm A and Algorithm B, respectively. Both algorithms in both settings suffered the pre-set timeout.

Figure 3 shows the reduction trend (optimization level) of cost function values which were obtained by Algorithm A and Algorithm B during the pre-set execution time. The above figure in Figure 3 presents the decrease of the cost function value with execution time increasing in Setting 1, and the below figure illustrates the trend in Setting 2. The execution time on the horizontal axis represents the time of optimizing the path planning problem, and the cost function value (total distance of generated path) is shown in the vertical axis. The dots on the lines (red line and blue line) represent new optimal

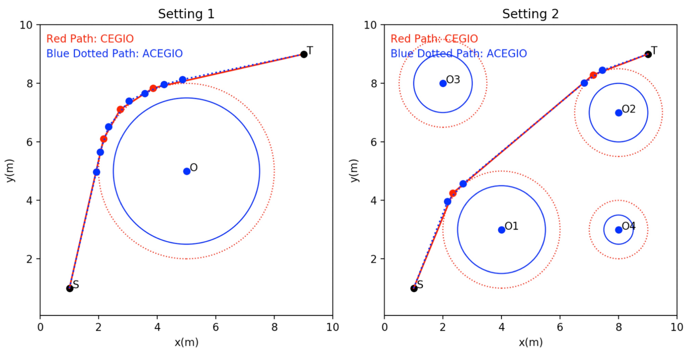

Fig. 2: Paths generated by Algorithm A and Algorithm B.

candidates generated by the path planning algorithm at that time. The red line represents the decrease of cost function values obtained by Algorithm A. The blue line shows the changes of cost function values generated by Algorithm B during the optimization process.

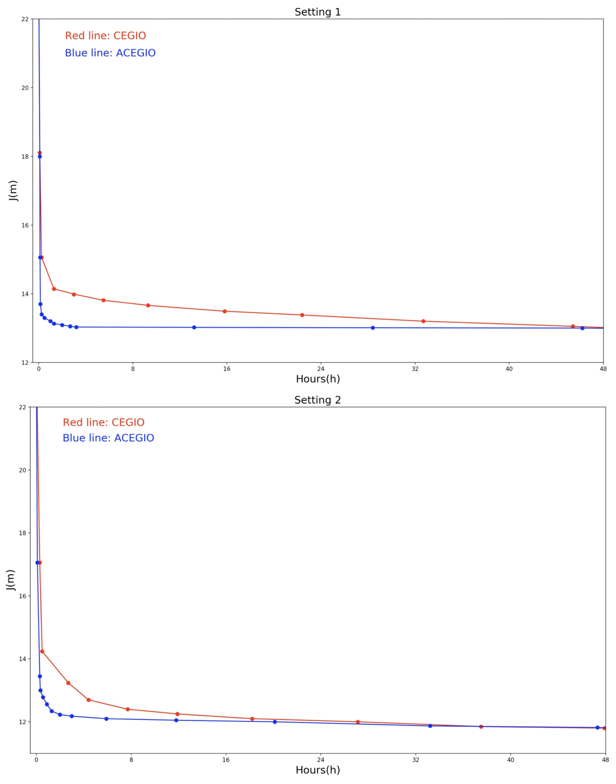

Fig. 3: Trend of cost function value during optimization.

According to the results achieved by Algorithm A and B, all solutions in both scenarios are not the global optimal solutions because of the pre-set timeout. However, they are close to global optimality. The cost function value continuously decreases and converges to the global optimality.

For both settings, the red line is above the blue line. With the assistance of Gradient Descent algorithm, the cost function value generated by Algorithm B (blue line) decreases more sharply and converges more quickly to the optimal solution. Therefore, the ACEGIO-based path planning algorithm can get solutions closer to the global optimal solution with much less execution time than the paths generated by the CEGIO-based path planning algorithm. Therefore, ACEGIO-based path planning algorithm does significantly improve the efficiency of finding a more optimized solution, which answers EG2. In addition, Algorithm B has successfully generated paths extremely close to global optimum within the timeout and the blue line converges to the global optimality. Therefore, the effectiveness of the ACEGIO-based path planning algorithm has also been proved, which answers EG1.

EG1 **(effectiveness)** The experimental results have identified that the ACEGIO-based path planning algorithm can generate satisfactory paths (i.e., paths extremely close to the global optimal path) and solve the optimal path planning problem.

EG2 **(efficiency)** These experiments have confirmed that the ACEGIO-based path planning algorithm has generated more optimized paths with much less execution time, which considerably improves the efficiency of obtaining optimal path for the optimal path planning problem.

D. Comparison to other state-of-the-art methods

Comparison among ACEGIO-based path planning (Algorithm B), GA-based path planning algorithm (Algorithm C) and PSO-based path planning algorithm (Algorithm D) are presented in this section, which aims to answer EG3.

The paths generated by the Algorithm C for Setting 1 and Setting 2 are presented in Figure 4 and Figure 5, respectively. In Figure 4 and Figure 5, the blue circles are the edges of obstacles, and the red circles are the edges of the obstacle safety margin. The achieved red paths starting from the yellow square (starting position) to the green star (target position) are composed of a series of blue points. Figure 4 and Figure 5 both contain three experimental results of three n=4 cases (four points on the path including the starting point and target point), three n=5 cases and three n=6 cases.

The execution speed of using GA for solving the path planning problem is extremely fast. Setting the iteration times to 100000, each experiment was completed in a few minutes. However, a few problems occurred while applying GA to the path planning problem. Firstly, according to the results in Figure 4 and Figure 5, comparing to the path generated by ACEGIO-based path planning algorithm, the paths generated by GA-based path planning algorithm are farther from the global optimality. Secondly, with more chromosomes (more points on the path), the performance of GA is worse and unstable. Besides, there is uncertain that the cost function value converges differently and sometimes even fails to converge to the global optimality. Overall, the GA-based path planning algorithm is much faster, while the ACEGIO-based path planning algorithm is more effective than GA, and its performance is more stable than GA.

The PSO-based path planning algorithm generates paths for Setting 1 and 2 as illustrated in Fig. 6. The half left of Fig. 6

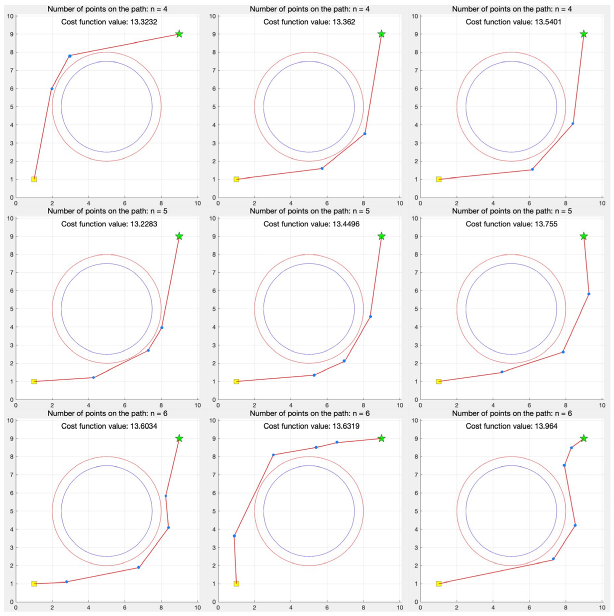

Fig. 4: Paths generated by Algorithm C in Setting 1.

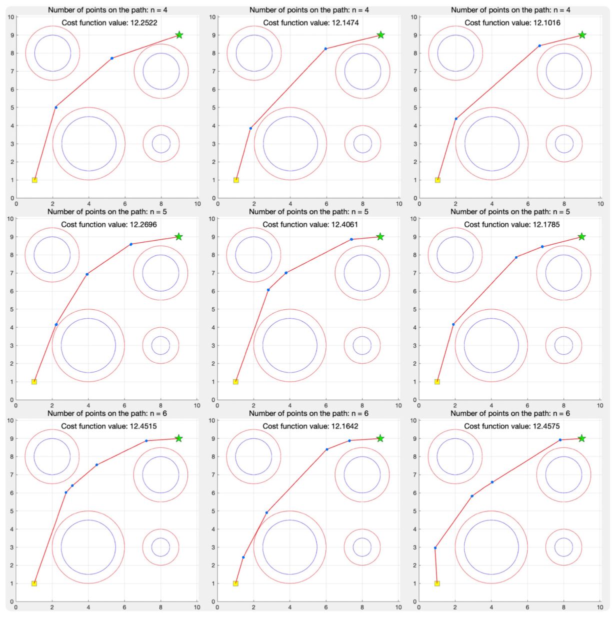

Fig. 5: Paths generated by Algorithm C in Setting 2.

contains two results of $n = 4$, two $n = 6$, and two $n = 10$ cases for Setting 1. The half right of Fig. 6 has the same empirical points for Setting 2. In Fig. 6, subfigures in column (a) and column (c) present the paths generated by PSO-based path planning algorithm with appropriate parameters, while subfigures in column (b) and column (d) show the paths achieved with inappropriate parameters setting.

According to the experimental results, with appropriate parameters, paths extremely close to the global optimality could be achieved by PSO-based path planning algorithm. Besides repeating the same experiments 100 times, the cost function value always converged to the global optimality,

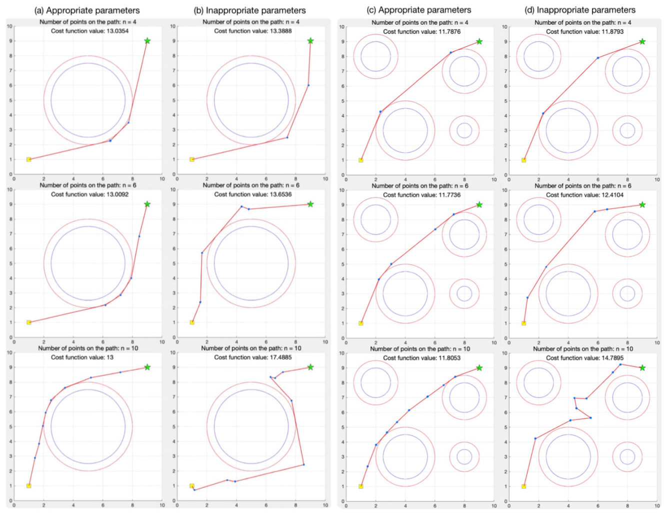

Fig. 6: Paths generated by Algorithm D.

which illustrates the performance of PSO-based path planning is stable and reliable. However, with inappropriate parameters, paths farther from the global optimal path were achieved. This is because the parameters have more obvious impacts while the number of points on the path increasing. In addition, because parameters of PSO (such as the number of particles, acceleration coefficients, inertia weight, neighborhood size) can influence the performance, and these parameters differ while problems change. Therefore, tuning these problem-dependent parameters is an essential step while employing PSO. Comparing to ACEGIO-based path planning, the only parameter that needs to be tuned in ACEGIO-GD is the step length of Gradient Descent, which is much easier to tune.

EG3 (state-of-the-art) Those comparisons between ACEGIO-based path planning algorithm and GA-based path planning algorithm have shown that the performance of the ACEGIO-based path planning algorithm is more stable and reliable. If compared to PSO-based path planning algorithm, the parameter of the proposed algorithm has less impacts on its performance, which makes it robust and easy to employ.

E. Threats to Validity

The proposed approach achieved satisfactory experimental results. Nonetheless, we discovered three threats to the validity of our assessment results: (1) memory consumption is not considered while comparing ACEGIO-based path planning algorithm with CEGIO-based path planning algorithm. Two criteria assess the comparison between these two algorithms: *how close are the derived paths to the global optimality? How much execution time does the algorithm spend?* The comparison results demonstrate the effectiveness and efficiency of the proposed algorithm. However, a lack of consideration of memory consumption may reduce the conclusion. (2) The proposed algorithm is evaluated in two environments. Theoretically, the proposed algorithm can be applied to all two-dimensional path planning problems described in Sections II and III. However, insufficient experimental settings could result in a lack of evidence for demonstrating the robustness of the proposed algorithm in practice. (3) Different tools are applied while comparing the ACEGIO-based path planning algorithm with other state-of-the-art approaches. This does not influence the conclusion in this study, but uniting the tools operating speed can make this study more accurate.

V. Conclusion

CEGIO-based path planning algorithm can provide optimal paths, but the cumulative execution time is high compared to other traditional optimization-based path planning algorithms. To overcome the CEGIO shortcoming, we proposed and evaluated a novel mobile robot path planning algorithm, which relies on the ACEGIO-GD algorithm to solve the optimal path planning problem. Experimental results show that ACEGIO-GD takes advantage of Gradient Descent's efficiency and CEGIO's optimization ability, allowing ACEGIO-GD to generate optimal paths with significantly shorter execution time than the original CEGIO-based algorithms. Compared to GA and PSO, the execution time of ACEGIO-based path planning algorithms is relatively slow. In particular, the given cost function always converges towards the global optimality while employing a GA-based path planning algorithm can not ensure the convergence, which indicates ACEGIO-based path planing is more stable and reliable than GA-based path planning algorithm. Our focus for future work consists of developing other auxiliary algorithms to assist CEGIO and exploring the best auxiliary algorithm for CEGIO to solve optimal path planning problems.

Acknowledgment

We acknowledge the contributions of Dumitru Chitoraga to the initial exploration of the ACEGIO approach. The work in this paper is partially funded by CNPq grant 432341/2018-8.

References

[1] N Vimal Kumar and C Selva Kumar. "Development of collision free path planning algorithm for warehouse mobile robot". In: *Procedia computer science* 133 (2018), pp. 456–463.

[2] W. Rahiman F. Gul and S. Nazli Alhady. "A comprehensive study for robot navigation techniques". In: *Cogent Eng.* (2019).

[3] Hui Liu. *Robot Systems for Rail Transit Applications*. Vol. 1. Elsevier, 2020, p. 418.

[4] Rodrigo F. Araujo et al. "Counter-example Guided Inductive Optimization Applied to Mobile Robots Path Planning". In: *SBR-LARS* (2017), pp. 1–6. DOI: https://doi.org/10.1109/SBR-LARS-R.2017.8215336.

[5] P. BAJPAI and M. Kumar. "Genetic Algorithm-an Approach to Solve Global Optimization Problems". In: *IJCSE* 1.3 (2010), pp. 199–206.

[6] N. BAYGIN et al. "PSO Based Path Planning Approach for Multi Service Robots in Dynamic Environments". In: *IDAP* (2018), pp. 1–5.

[7] R. Rashid et al. "Mobile robot path planning using Ant Colony Optimization". In: *ROMA* (2016), pp. 1–6.

[8] Constantin Purcaru et al. "Optimal robot path planning using gravitational search algorithm". In: *Int. J. Artif. Intell* 10.13 (2013), pp. 1–20.

[9] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 4th Edition*. 2020.

[10] Rodrigo Araújo et al. "SMT-based Verification Applied to Non-convex Optimization Problems". In: *SBESC* (2016), pp. 1–8.

[11] Rodrigo F. Araujo et al. "Counterexample guided inductive optimization based on satisfiability modulo theories". In: *Sci. Comput. Program.* 165 (2018), pp. 1–38. DOI: 10.1016/j.scico.2017.10.004.

[12] Zvi Shiller. "Off-line and on-line trajectory planning". In: *Motion and Operation Planning of Robotic Systems* (2015), pp. 29–62.

[13] Roberto Sebastiani and Patrick Trentin. "OptiMathSAT: A Tool for Optimization Modulo Theories". In: *J. Autom. Reason.* 64.3 (2020), pp. 423–460.

[14] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[15] Anis Koubaa et al. "Introduction to mobile robot path planning". In: *Robot Path Planning and Cooperation* 772 (2018), pp. 3–12.

[16] E. Clarke A. Biere A. Cimatti and Y. Zhu. "Symbolic Model Checking without BDDs". In: *TACAS* (1999), pp. 193–207.

[17] C. Baier and J. Katoen. "Principles of model checking". In: *MIT Press* (2008).

[18] Rodrigo F. Araujo et al. "Counterexample Guided Inductive Optimization". In: *arXiv:1704.03738 [cs.AI]* (2017), pp. 1–32. DOI: http://arxiv.org/abs/1704.03738.

[19] Higo F. Albuquerque et al. "OptCE: A Counterexample-Guided Inductive Optimization Solver". In: *SBMF*. Vol. 10623. Springer, 2017, pp. 125–141.

[20] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs.LG]* (2016).

[21] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv [cs.LG]: 1609.04747.

[22] Mikhail Y. R. Gadelha et al. "ESBMC: Scalable and Precise Test Generation based on the Floating-Point Theory - (Competition Contribution)". In: *23rd International Conference Fundamental Approaches to Software Engineering - FASE*. Vol. 12076. Lecture Notes in Computer Science. Springer, 2020, pp. 525–529.

[23] Phillipe A. Pereira et al. "SMT-based context-bounded model checking for CUDA programs". In: *Concurr. Comput. Pract. Exp.* 29.22 (2017).

[24] Felipe R. Monteiro et al. "Bounded model checking of C++ programs based on the Qt cross-platform framework". In: *Softw. Test. Verification Reliab.* 27.3 (2017).

[25] Aina Niemetz, Mathias Preiner, and Armin Biere. "Boolector 2.0". In: *J. Satisf. Boolean Model. Comput.* 9.1 (2014), pp. 53–58.