

## Research Article

## Genetic Informed Trees (GIT\*): Path planning via reinforced genetic programming heuristics

Liding Zhang<sup>a</sup>, Kuanqi Cai<sup>b,\*</sup>, Zhenshan Bing<sup>a,\*</sup>, Chaoqun Wang<sup>c</sup>, Alois Knoll<sup>a</sup><sup>a</sup> School of Computation, Information and Technology (CIT), Technical University of Munich, Munich 85748, Germany<sup>b</sup> Munich Institute of Robotics and Machine Intelligence (MIRMI), Technical University of Munich, Munich 80992, Germany<sup>c</sup> School of Control Science and Engineering, Shandong University, Jinan 250100, China

## ARTICLE INFO

## Article history:

Received 11 March 2025

Revised 4 May 2025

Accepted 6 May 2025

Available online 20 May 2025

## Keywords:

Genetic algorithm

Reinforced genetic programming

Generative heuristics

Optimal path planning

## ABSTRACT

Optimal path planning involves finding a feasible state sequence between a start and a goal that optimizes an objective. This process relies on heuristic functions to guide the search direction. While a robust function can improve search efficiency and solution quality, current methods often overlook available environmental data and simplify the function structure due to the complexity of information relationships. This study introduces Genetic Informed Trees (GIT\*), which improves upon Effort Informed Trees (EIT\*) by integrating a wider array of environmental data, such as repulsive forces from obstacles and the dynamic importance of vertices, to refine heuristic functions for better guidance. Furthermore, we integrated reinforced genetic programming (RGP), which combines genetic programming with reward system feedback to mutate genotype-generative heuristic functions for GIT\*. RGP leverages a multitude of data types, thereby improving computational efficiency and solution quality within a set timeframe. Comparative analyses demonstrate that GIT\* surpasses existing single-query, sampling-based planners in problems ranging from  $\mathbb{R}^4$  to  $\mathbb{R}^{16}$  and was tested on a real-world mobile manipulation task. A video showcasing our experimental results is available at [https://youtu.be/URjXbc\\_BiYg](https://youtu.be/URjXbc_BiYg).

© 2025 The Author(s). Published by Elsevier B.V. on behalf of Shandong University. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

## 1. Introduction

Motion planning is a fundamental challenge in robotic automation, involving the determination of a sequence of valid states that guide a robot from a starting point to a desired goal while avoiding obstacles [1–3]. Many algorithms have been proposed to address this problem, such as the A\* algorithm [4], artificial potential field (APF) algorithm [5], and sampling-based algorithms [6]. The A\* algorithm's performance declines with higher dimensionality, while the APF algorithm often converges to local minima. Sampling-based algorithms have gained popularity due to their efficient exploration of the state space [7]. However, they often require significant time to find the optimal solution. In multi-dimensional environments, such as autonomous vehicles and robot manipulators, it is essential to compute an efficient path to conserve power [8,9].

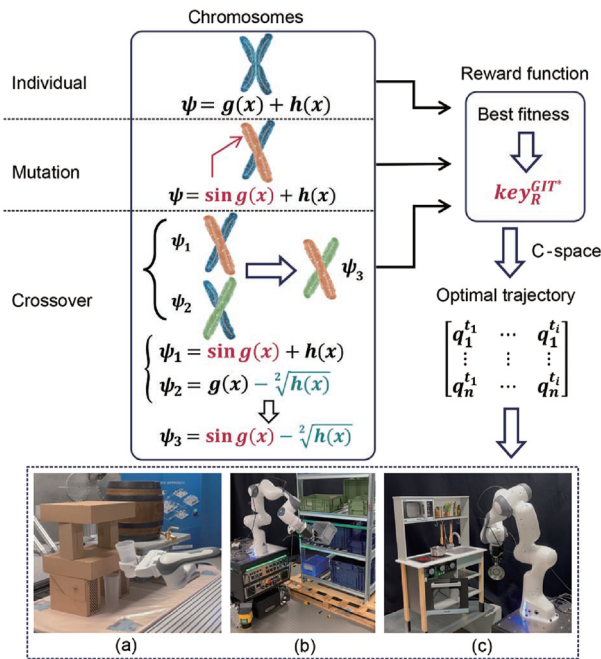
The motivation for this paper is to improve the convergence rate and find successful solutions faster with lower initial solution costs based on the genetic-based generation of heuristics. Sampling-based algorithms like Rapidly-exploring Random

Trees (RRT) [10], Probabilistic Roadmaps (PRM) [11], and variant algorithms of RRTs [12,13] have been widely used for recent path planning work and have demonstrated effectiveness in practical applications. However, these algorithms' performance fluctuates greatly in different environments. On the other hand, in optimization algorithm research, a combined method known as reinforced genetic programming (RGP) [14] is proposed. We introduced *genotype-generative heuristic* (G-heuristic) functions based on RGP for optimal edge evaluation, incorporating a fitness reward function to facilitate autonomous learning and adjustment of exploration strategies based on environmental feedback. This approach integrates the genetic algorithm (GA) [15] to assess bio-inspired chromosome behavior (e.g., *crossover*, *mutation*, *reproduction*) for integration with sampling-based planners. However, the G-heuristic cannot be directly applied to robot path planning because it does not consider environmental constraints (e.g., *obstacle avoidance*) or robustness across various scenarios. Therefore, the G-heuristic must be trained across different benchmarking datasets using reward feedback for robustness (see Fig. 1).

Inspired by RGP technology, this paper presents the Genetic Informed Trees (GIT\*) algorithm, which generates a heuristic function using problem-specific information via RGP. This heuristic enhances efficiency by minimizing expanded vertices. GIT\*

\* Corresponding authors.

E-mail addresses: [kuanqi.cai@iit.it](mailto:kuanqi.cai@iit.it) (K. Cai), [zhenshan.bing@tum.de](mailto:zhenshan.bing@tum.de), [kayle.ckq@gmail.com](mailto:kayle.ckq@gmail.com) (A. Knoll).



**Fig. 1.** GIT\* utilizes a population of G-heuristics (i.e.,  $\psi_1, \psi_2, \dots$ ) with chromosome behaviors. The example of G-heuristics is illustrated above, which are trained using a reward function over multiple generations through RGP. The best G-heuristic is employed for pathfinding guidance. GIT\*'s performance is evaluated in (a) beer barrel, (b) shelf, and (c) kitchen model scenarios.

uses invalid samples within obstacles and start/goal points to create an APF, incorporating obstacle shapes and locations, and tracks sample visit frequency to account for the dynamic importance of states. The G-heuristics represent a symbolic regression problem tackled by RGP. It involves evolving nonlinear expressions to refine the heuristic. As shown in Fig. 2, G-heuristic enables GIT\* to find the initial solution quickly and then expand. GIT\* incorporates additional graph search techniques, such as truncation and inflation, to balance exploitation and exploration, dynamically modified using RGP. GIT\* has shown improvements over state-of-the-art (SOTA) methods in time to find the initial solution, initial solution quality, and final solution quality in both generalized simulation benchmarks and real-world experiments.

The contributions of this paper are summarized as follows:

- (1) An efficient optimal genotype-generative heuristic function based on reinforced genetic programming, trained with a dataset from the random problem domain.
- (2) A novel sampling-based path planning algorithm, GIT\*, integrates the trained genotype-generative heuristic function to rapidly obtain high-quality solutions.
- (3) Demonstrating the effectiveness of GIT\* across various dimensional environments and optimization objectives.

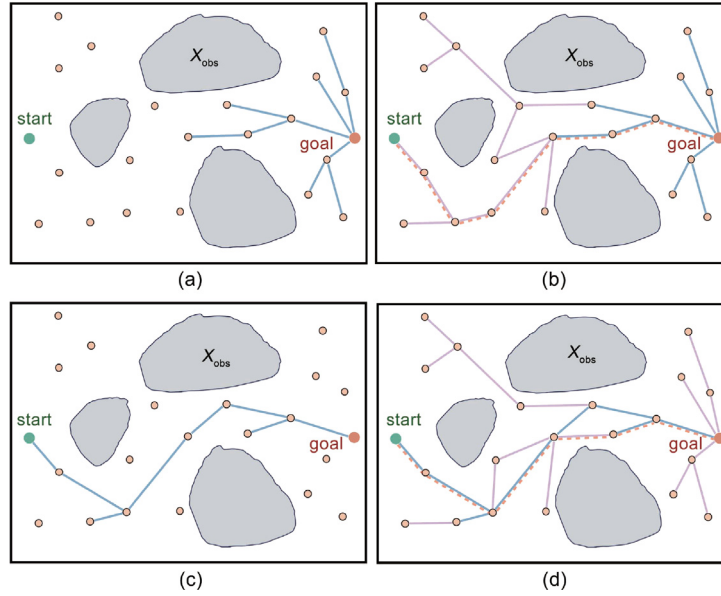
## 2. Related work

Heuristic functions enhance path planning by estimating the cost to the goal state, which is crucial in high-dimensional spaces. Informed planners leveraging heuristics outperform their uninformed counterparts [16,17]. Effective heuristics should be both accurate and computationally efficient; however, balancing these properties remains challenging [18]. This work employs G-heuristics to improve path planning efficiency and solution quality.

Graph-based algorithms like Dijkstra's [19] aim to find the shortest path by exhaustively exploring all possible routes within

a discrete graph. In contrast, the A\* algorithm [4] improves computational efficiency by using a heuristic to guide the search. The Anytime Repairing A\* (ARA\*) [20] ensures that solutions remain progressively optimal and provide valid paths at any given moment. However, when applied to continuous spaces, these methods require discretization, which may either lead to sparse grids that yield suboptimal paths or dense grids that suffer from high computational demands due to the *curse of dimensionality* [21]. RRT-Connect [22] extends the RRT framework by growing two trees: one from the start state and the other from the goal, using heuristic-guided planning to accelerate path convergence. Kinematic Constrained Bi-directional RRT (KB-RRT) [23] proposes an efficient branch pruning strategy and considers robot kinematic contain. Cost heuristics in tree growth are demonstrated by Heuristically-guided RRT (hRRT) [24] and Generalized Bidirectional RRT (GBRRT) [25]. hRRT uses a priori heuristics for exploration within RRT's Voronoi regions, while GBRRT, a bidirectional RRT variant, employs reverse tree-computed heuristics to guide the forward tree.

RRT\* [6] improves on RRT by incrementally rewiring the tree to achieve asymptotic optimality. Quick-RRT\* [26] rewires with the ancestor node, which can increase the speed of convergence to the optimal solution. Informed RRT\* [27] improves further by using elliptical informed sampling [16], thus accelerating optimal path convergence. To enhance planning efficiency, researchers have explored learning-based approaches [28]. Motion Planning Networks (MPNet) [29] eliminate explicit trajectory optimization by directly mapping sensory inputs to motion sequences. Neural RRT\* [30] and Neural Informed RRT\* [31] leverage neural networks to guide sampling, reducing redundant exploration. In addition to learning-based planners, optimization-based motion planning methods have also received considerable attention. These methods formulate the planning problem as a continuous optimization problem, often resulting in smooth and dynamically feasible trajectories. Notable examples include CHOMP [32], STOMP [33], and more recent advances based on sequential convex optimization such as TrajOpt [34], which improves planning efficiency by leveraging convex collision checking. Furthermore, Marcucci et al. [35] proposed a framework that combines convex optimization with obstacle avoidance guarantees, demonstrating strong theoretical properties and promising empirical performance. Batch Informed Trees (BIT\*) [36,37] uses A\* on a *random geometric graph* (RGG) formed by random samples, improving approximation as samples increase. While BIT\* efficiently refines its search, it still has limitations in using problem-specific information. Advanced BIT\* (ABIT\*) [38] enhances BIT\* by introducing inflated and truncated factors, balancing the exploration and exploitation. Flexible Informed Trees (FIT\*) [39] uses flexible batch sizes for fast initial solution. Adaptively Informed Trees (AIT\*) [18, 40] and Effort Informed Trees (EIT\*) [18] further enhance efficiency with bidirectional search strategies. EIT\* uses adaptive sparse collision checks, reducing expensive collision detections. The forward and reverse trees inform each other, sharing complementary information to optimize the search. However, these optimization-based methods produce high-quality trajectories, they are often sensitive to initialization and may require good prior guesses or warm starts. State-of-the-art planners did not leverage the information from the invalid sample/nearest neighbor [41] in the planning domain. Our approach differs because it integrates APF and dynamic importance for further guidance. GIT\* focuses on efficient and scalable exploration in high-dimensional configuration spaces with genotype-generative heuristic functions for pathfinding. If trajectory smoothness is permitted, it can be combined with trajectory optimization as a post-processing step.



**Fig. 2.** Four snapshots show EIT\* and GIT\* exploration strategies in reverse search. GIT\* employs G-heuristics, while EIT\* maintains a linear combination heuristic. Yellow points indicate valid samples in obstacle-free areas. The blue line represents the reverse tree at time  $t_n$ , the pink line represents it at time  $t_{n+1}$ , and the dashed red lines are optimized reverse edges.

### 2.1. Genetic-based path planning method

Genetic sampling-based algorithms utilize genetic operations like crossover and mutation to generate candidate solutions in the problem space. Hybridizing-RRT [42] uses a hybrid path generation scheme that combines RRT with an island parallel GA to efficiently find  $G^3$ -continuous  $\eta^3$ -spline paths that optimize path length and curvature. This approach leverages RRT injections to maintain genetic diversity and prevent premature convergence in complex map scenarios. Genetic-RRT [43] uses GA to optimize paths planned by RRTs. This approach retains multiple optimal solutions, increasing the likelihood of finding an asymptotically optimal path with more iterations. However, genetic-based algorithms typically use GA to optimize path length objectives, often overlooking edge evaluation during exploration, resulting in a challenging achievement of rapid convergence of the initial solution. Our method enhances search efficiency by employing RGP to create G-heuristics.

### 2.2. Applications of symbolic regression

Symbolic regression, a popular application of GA [44], discovers mathematical expressions that accurately represent datasets without presupposing a specific mathematical form. Unlike traditional regression models that require predefined functional relationships, symbolic regression explores all possible expressions, uncovering complex data relationships, nonlinear interactions, and dynamic patterns [45]. Inspired by symbolic regression, our work incorporates genetic programming to generate heuristic functions within the GIT\* algorithm for path planning. This integration allows GIT\* to leverage a broader range of data, improving computational efficiency and solution quality.

The Open Motion Planning Library (OMPL) [46] is commonly used in benchmarking motion planning algorithms. It provides a comprehensive framework and tools for researchers to evaluate algorithms. GIT\* is integrated into the OMPL framework, the Planner-Arena benchmark database [47], and Planner Developer Tools (PDT) [48].

### 3. Problem formulation

We define the optimal planning problem according to the definition provided in [6] and consider the symbolic regression problem defined in [49] as a tool to address optimal planning.

**Problem Definition 1 (Optimal Planning).** Consider a path planning problem with the  $n$ th dimensional state space  $X \subseteq \mathbb{R}^n$ . Let  $X_{\text{obs}} \subset X$  represent states in collision with obstacles, and  $X_{\text{free}} = \text{cl}(X \setminus X_{\text{obs}})$  denote the resulting permissible states, where  $\text{cl}(\cdot)$  represents the *closure* of a set. The initial/start state is denoted by  $\mathbf{x}_{\text{start}} \in X_{\text{free}}$ , and the set of desired final/goal states is  $X_{\text{goal}} \subset X_{\text{free}}$ . A sequence of states  $\sigma : [0, 1] \mapsto X$  forms a continuous map (i.e., a collision-free, feasible path), and  $\Sigma$  represents the set of all nontrivial paths.

The optimal solution, represented as the queue vector  $\sigma^*$ , corresponds to the path that minimizes a selected scalar cost function  $s : \Sigma \mapsto \mathbb{R}_{\geq 0}$ . This path connects the initial state  $\mathbf{x}_{\text{start}}$  to any goal state  $\mathbf{x}_{\text{goal}} \in X_{\text{goal}}$  through the free space:

$$\sigma^* = \arg \min_{\sigma \in \Sigma} \{s(\sigma) \mid \sigma(0) = \mathbf{x}_{\text{start}}, \sigma(1) \in X_{\text{goal}}, \forall t \in [0, 1], \sigma(t) \in X_{\text{free}}\} \quad (1)$$

where  $\mathbb{R}_{\geq 0}$  denotes non-negative real numbers. The cost of the optimal path is  $s^*$ , and  $t$  is the timestep of the exploration.

Considering a discrete set of states,  $X_{\text{samples}} \subset X$ , as a graph where edges are determined algorithmically by a transition function, we can describe its properties using a probabilistic model implicit dense RGGs when these states are randomly sampled, i.e.,  $X_{\text{samples}} = \{\mathbf{x} \sim \mathcal{U}(X)\}$ , as discussed in [50].

The characteristics of the anytime almost-surely sampling-based planner with the definition are provided in [16].

**Problem Definition 2 (Symbolic Regression).** Symbolic regression seeks to find a mathematical expression that best fits a given dataset. The goal is to minimize the error between the predicted output  $\hat{y}$  and the actual output  $y$  over a dataset  $\mathcal{D}$  [51]. The dataset  $\mathcal{D}$  consists of a finite set of input-output pairs  $(\rho, y)$ , where  $\rho = (\rho_1, \rho_2, \dots, \rho_n)$  represents the  $n$ -dimensional input vector, and  $y$  is the corresponding scalar output. These pairs are typically

obtained from historical data, experimental measurements, or simulated observations of a physical system. The dataset  $\mathcal{D}$  can be expressed as

$$\mathcal{D} = \{(\rho_i, y_i)\}_{i=1}^N \quad (2)$$

where  $N$  is the total number of data points in the dataset,  $\rho_i$  is the  $i$ th input vector, and  $y_i$  is the corresponding output value for  $\rho_i$ . The dataset  $\mathcal{D}$  is either sampled from an underlying system or generated from empirical data sources, and it serves as the basis for finding the symbolic expression  $\psi$  that minimizes the error between predicted and actual outputs. The symbolic regression process involves searching for a symbolic expression  $\psi$  that minimizes the error over the dataset. This can be formulated as an optimization problem where the objective is to minimize the sum of squared errors between the predicted outputs  $\hat{y}$  and the actual outputs  $y$ . The fitness function  $Fitness(\cdot)$  quantifies this error and is defined as

$$Fitness(\psi) = \sum_{i=1}^N (y_i - \hat{y}(\rho_i, \psi))^2 \quad (3)$$

where  $\psi$  represents the symbolic expression,  $\hat{y}(\rho_i, \psi)$  is the predicted output generated by the symbolic expression  $\psi$  for the input  $\rho_i$ , and  $y_i$  is the actual output corresponding to  $\rho_i$  in the dataset  $\mathcal{D}$ . The objective of symbolic regression is to find the expression  $\psi^*$  that minimizes the fitness function:

$$\psi^* = \arg \min_{\psi} Fitness(\psi) \quad (4)$$

which in this context, the fitness function measures how well a given symbolic expression  $\psi$  fits the dataset  $\mathcal{D}$ . By minimizing the fitness function, we aim to find the symbolic expression that best fits the data, as discussed in [49].

## 4. Algorithm

This section explains how to use the RGP to learn heuristic functions from the benchmark dataset. Then, the learned G-heuristics are then applied in the  $GIT^*$  to achieve fast and high-quality path planning. Finally, we prove that  $GIT^*$  guarantees probabilistic completeness and asymptotic optimality.

### 4.1. Notation

The state space of the planning problem is denoted by  $X \subseteq \mathbb{R}^n$ , where  $n \in \mathbb{N}$ . The start point is represented by  $\mathbf{x}_{start} \in X$ , and the goals are denoted by  $X_{goal} \subset X$ . The sampled states are denoted by  $X_{sampled}$ . The forward and reverse search trees are represented by  $\mathcal{T}_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$  and  $\mathcal{T}_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ , respectively. The vertices in these trees, denoted by  $V_{\mathcal{F}}$  and  $V_{\mathcal{R}}$ , correspond to valid states. The edges in the forward tree,  $E_{\mathcal{F}} \subseteq V_{\mathcal{F}} \times V_{\mathcal{F}}$ , represent valid connections between states, while the edges in the reverse tree,  $E_{\mathcal{R}} \subseteq V_{\mathcal{R}} \times V_{\mathcal{R}}$ , may traverse invalid regions of the problem domain. An edge comprises a source state,  $\mathbf{x}_s$ , and a target state,  $\mathbf{x}_t$ , denoted as  $(\mathbf{x}_s, \mathbf{x}_t)$ . The true connection cost between two states in *configuration space* ( $\mathcal{C}$ -space) is represented by the function  $c : X \times X \rightarrow [0, \infty)$ .

Let  $A$  be a set and let  $B, C$  be subsets of  $A$ . The notation  $B \stackrel{\pm}{\leftarrow} C$  is used to denote  $B \leftarrow B \cup C$  and  $B \stackrel{-}{\leftarrow} C$  is used to denote  $B \leftarrow B \setminus C$ .

*$GIT^*$ -specific Notation:* Let  $\Theta$  be the space of all path planning problems and  $\mathcal{E}$  be the space of all path planning algorithms. The dataset consisting of  $k$  path planning problems is represented as  $D_{benchmark}^k = \{\theta_1, \theta_2, \dots, \theta_k\}$ . The function  $\Phi : \mathcal{E} \times \Theta \rightarrow \{(m_1, v_1), (m_2, v_2), \dots, (m_k, v_k)\}$  quantifies the expected performance of running an algorithm  $\xi \in \mathcal{E}$  on a path planning

problem  $\theta \in \Theta$  one hundred times, with the performance measured by  $k$  distinct indicators. The elements  $(m_i, v_i)$  belong to a set  $M \times \mathcal{V}$ , where  $M$  is the set of all possible metrics and  $\mathcal{V}$  is the set of all possible values.

In the genetic programming process, an individual is denoted as  $\psi \in \mathcal{E}$ . The individual corresponding to the heuristic function of  $EIT^*$  is defined as  $\psi_{EIT^*}$ . The individuals generated in the same iteration form a population  $\mathcal{P}$ , with size denoted as  $\mathcal{O}$ . The probabilities of mutation and crossover, the two types of genetic operations, are denoted as  $p_m$  and  $p_c$  respectively. We define the fitness loss function  $\phi : \mathcal{E} \rightarrow [0, \infty)$ , which quantitatively evaluates the performance of individuals on the dataset  $D_{benchmark}^k$ . The evaluated fitness value of an individual is denoted as  $\rho_{\psi} := \phi(\psi)$ . The algorithm obtained by substituting the heuristic function in  $EIT^*$  with the individual  $\psi_i$  is denoted by  $GIT_{\psi_i}^*$ . The reward function to assess the improvement of algorithm performance is denoted as  $\chi : M \times \mathcal{V} \rightarrow [0, \infty)$ . The function  $U : X \rightarrow [0, \infty)$  provides the magnitude of potential energy of a state in APF.

### 4.2. Reinforced genetic programming (RGP)

This subsection introduces RGP and its adaptation to improve the heuristic function in sampling-based path planning. RGP uses a reward function to evaluate candidate models on unlabeled data, enabling model evolution.

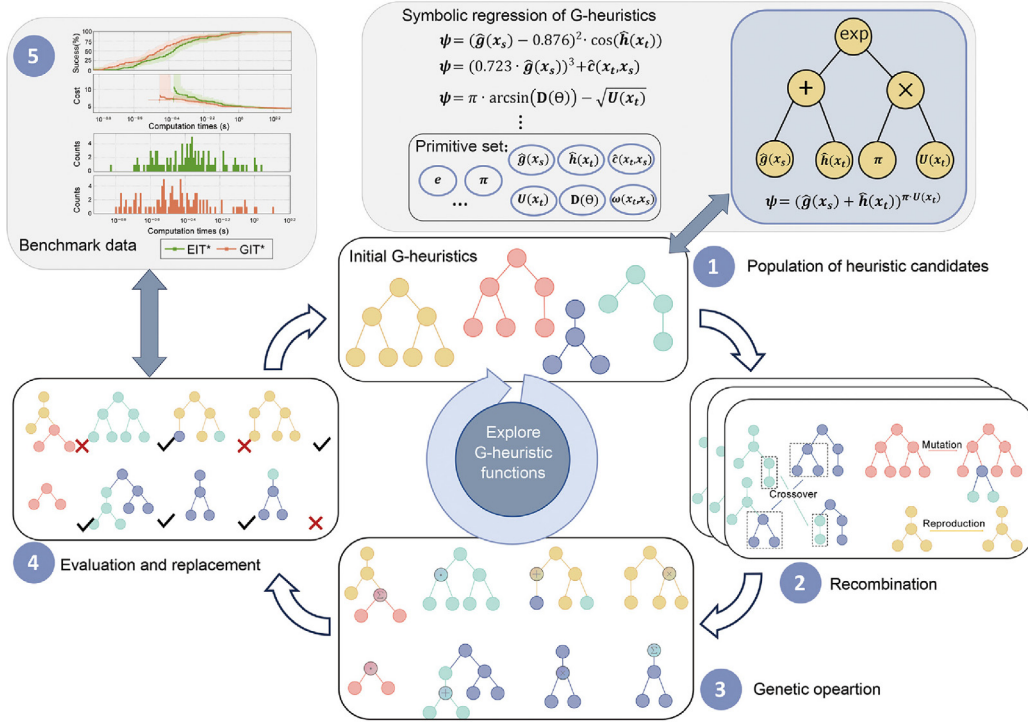
As shown in Fig. 3, RGP continues the traditional GP iterative evolutionary process. Initially, a primitive set is established to generate individuals and populations in the evolutionary cycle. This set includes essential components for individual generations. An algorithm outlines the rules for assembling individuals from these components. Multiple individuals created using the primitive set form a population of candidate solutions. Each individual  $\psi_i$  represents a heuristic function and corresponds to a new algorithm  $GIT_{\psi_i}^*$ . The performance of this new algorithm is assessed using a Reinforced Fitness Evaluation Function, which compares the fitness of  $GIT_{\psi_i}^*$  with the baseline  $EIT^*$  algorithm.

Based on fitness values, exceptional individuals from the previous generation are chosen for the next, preserving superior genetic segments and removing inferior ones. This iterative process involves selecting parents, performing crossover and mutation to introduce new genetic segments, and evaluating the new population's fitness. Crossover mixes genetic material between parents, creating offspring with diverse traits, while mutation introduces random changes for unique variations. The process continues iteratively until a termination condition, such as a specific number of generations or satisfactory fitness, is met. The best-performing individual  $\psi^*$  is then selected as the genotype-generative heuristic function for  $GIT^*$ . The pseudocode for this process is illustrated in Alg. 1.

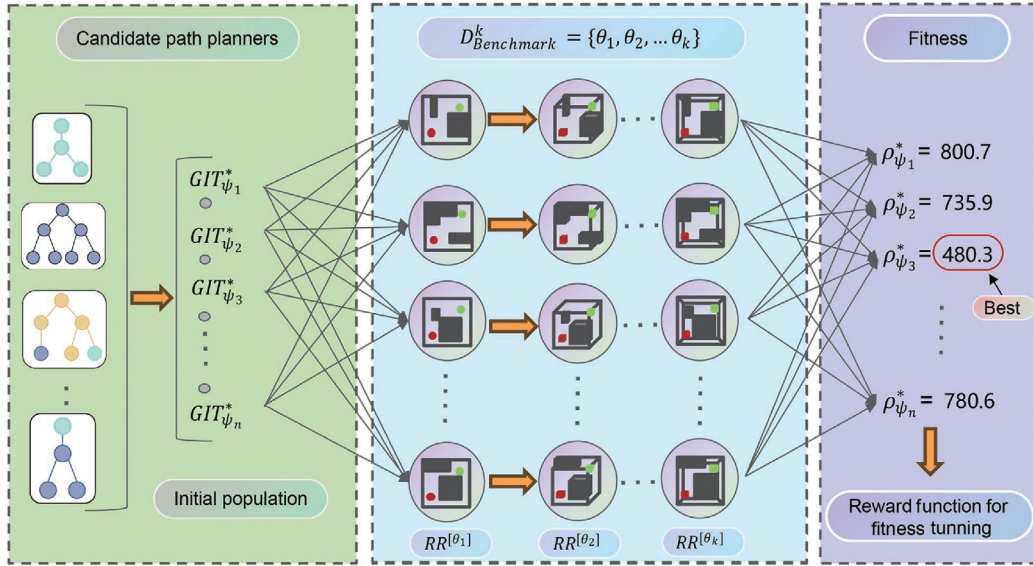
Unlike traditional GP, RGP employs a reward function to assess the fitness of individuals, known as the Reinforced Fitness Evaluation Function. In traditional GP, the dataset  $D$  comprises input data  $x_i \in X$  and corresponding label data  $l_i \in L$ . The objective is for each individual's model,  $\psi_i$ , to simulate the mapping from inputs to labels,  $\phi : X \rightarrow L$ , minimizing the discrepancy between predicted outputs and actual labels to optimize model performance (see Fig. 4).

However, in our path planning problem, only the environment and problem description are provided as input data without any labels. We adopted an incentive-based approach to evaluate an individual's fitness using our unlabeled dataset  $D_{benchmark}$ . The objective is to identify an individual  $\psi^*$  from the set  $\mathcal{E}$  to replace the heuristic of  $EIT^*$  and maximize the performance improvement over the  $EIT^*$  baseline, as measured by the loss function  $\phi$ . This is intended to optimize the effectiveness of the algorithm by adjusting its G-heuristic:

$$\psi^* = \arg \min_{\psi_i \in \mathcal{E}} \phi(\psi_i, D_{benchmark}) \quad (5)$$



**Fig. 3.** Overview of the Reinforced Genetic Programming (RGP) process for path planning. The process initiates with the collection of path planning scenarios as benchmark data, followed by the generation of expression trees acting as G-heuristics. These trees represent individuals' heuristics in the genetic process, undergoing various genetic operations such as mutation, crossover, and reproduction to evolve a new population of heuristic candidates. The cycle concludes with the evaluation and replacement of individuals, continuously iterating to enhance algorithmic performance.



**Fig. 4.** Illustration of the evaluation and fitness assignment process for individuals in the RGP. Each individual  $\psi_i$  represents an expression that serves as a heuristic to guide the search, forming a new planner  $GIT^*_{\psi_i}$ . These planners are benchmarked across multi-dimensional and multi-scenario problems. They are scored based on a designed reward function, and the resulting score is the fitness value of the individual.

$$\phi(\psi_i, D_{\text{benchmark}}) = \chi(\Phi(\psi_i, D_{\text{benchmark}}), \Phi(\psi_{\text{EIT}^*}, D_{\text{benchmark}})) \quad (6)$$

For problem description input  $\theta_i$ , we use the existing algorithm EIT\* performance as a control group, assessed over a set number of trials. We then compare this with the performance of a new algorithm  $GIT^*_{\psi_i}$ , generated by replacing EIT\*'s heuristic function with individual  $\psi_i$ , tested under identical conditions. If  $GIT^*_{\psi_i}$  outperforms EIT\* on any performance metric, the reward function  $\chi$  decreases the fitness score proportionally to the degree of improvement. Conversely, if  $GIT^*_{\psi_i}$  performs worse than EIT\*

on any metric,  $\chi$  increases the fitness score accordingly. This method ensures that a lower fitness score indicates the superior performance of an individual compared to EIT\* within the dataset  $D_{\text{benchmark}}$ .

To illustrate how the fitness of an individual  $\rho_{\psi_i}$  is assessed, consider the following example. Table 1 presents the performance results of EIT\* and  $GIT^*_{\psi_i}$ . These two algorithms were tested 100 times on the Random Rectangle problems across different dimensions, with time limits for each run (unsuccessful runs were

**Algorithm 1:** Reinforced genetic programming (RGP)

---

**Input** : Population size  $\mathcal{O}$ , mutation rate  $p_m$ , crossover rate  $p_c$

**Output:** Best found individual  $\psi^*$

- 1  $\mathcal{P} \leftarrow \text{initializePopulation}(\mathcal{O})$
- 2 **while not** terminateCondition **do**
- 3    $\rho_\psi \leftarrow \text{evaluateFitness}(\mathcal{P})$
- 4    $\mathcal{P}_{new} \leftarrow \emptyset$  // Initialize new population
- 5   **for**  $i \leftarrow 1$  **to**  $\mathcal{O}$  **do**
- 6      $\psi_{parent1} \leftarrow \text{select}(\mathcal{P}, \rho_\psi)$
- 7      $\psi_{parent2} \leftarrow \text{select}(\mathcal{P}, \rho_\psi)$
- 8      $\psi_{child} \leftarrow \text{crossover}(\psi_{parent1}, \psi_{parent2}, p_c)$
- 9     **if** random()  $< p_m$  **then**
- 10       $\psi_{child} \leftarrow \text{mutate}(\psi_{child})$
- 11      $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \psi_{child}$
- 12    $\mathcal{P} \leftarrow \mathcal{P}_{new}$
- 13    $\psi^* \leftarrow \text{bestIndividual}(\mathcal{P})$
- 14 **return**  $\psi^*$

---

considered as infinite costs). Ten critical metrics were evaluated, reflecting the algorithm's performance in terms of *time* to find the initial solution, the *cost of the initial solution*, the *cost of the optimal solution* within the time limit, and the final success rate over 100 runs of finding solutions.

When assessing the fitness  $\rho$  of an individual  $\psi_i$ , these metrics must be taken into consideration, and the corresponding weights for each metric should be set according to the specific application context. Below, we present the **reward function system** and rules used in our subsequent experiments:

(1) **Initial score:** The initial score for an individual is 800, ensuring the final computed fitness is greater than 0.

(2) **Weights of metrics:** To determine the specific weights  $w[m^i]$  for each metric  $m^i$ , the weighting depends on the specific application scenario and requirements. The weights for the metrics are provided in the **weights** row of Table 1.

(3) **Base score for each metric:** For each metric  $m^i$ , a base score  $s_{base}$  is assigned based on whether  $GIT^*$  outperforms  $EIT^*$  and the magnitude of the difference. First, it is assessed whether  $GIT^*$ 's value  $v_{GIT^*}^i$  outperforms  $EIT^*$ 's value  $v_{EIT^*}^i$ . If  $GIT^*$  is superior, a fixed score  $\delta$  is subtracted; otherwise, it is added. To quantify the degree of superiority, this score is multiplied by a coefficient  $\alpha$ , calculated as the ratio of the difference between  $v_{GIT^*}^i$  and  $v_{EIT^*}^i$ :

$$\alpha = \frac{v_{GIT^*}^i - v_{EIT^*}^i}{v_{EIT^*}^i} \quad (7)$$

The base score for the metric  $m^i$  is then:

$$s_{base}[m^i] = \delta + \delta \times \alpha \quad (8)$$

(4) **Handling infinity as a special case:** Some metrics may be infinite if solutions are not found in time. If both  $EIT^*$  and  $GIT^*$  record infinity for a metric,  $s_{base}$  is set to 0. If only one does,  $s_{base}$  is  $2 \times \delta$ .

(5) **Bonus for significant success rate enhancement:** Given the importance of the success rate, substantial differences between  $GIT^*$  and  $EIT^*$  in this metric should impact the overall fitness evaluation. If the difference  $v_{GIT^*}^{success} - v_{EIT^*}^{success}$  exceeds 5% but is less than 15%, a bonus  $s_{bonus}[m^{success}] = \delta$  is applied. For differences exceeding 15%,  $s_{bonus}[m^{success}] = 2 \times \delta$ .

(6) **Calculation of total score:** The total score run on path planning problem  $\theta$  is equal to the sum of all metrics' basic scores

**Algorithm 2:** Genetic informed trees (GIT\*)

---

**Input** : Start point  $\mathbf{x}_{start}$ , goal region  $X_{goal}$ , best individual  $\psi^*$ , optimal inflation/truncation factors  $\varepsilon_{infl}^*$ ,  $\varepsilon_{trunc}^*$

**Output:** Feasible path  $\mathcal{T}_F$

- 1  $X_{sampled} \leftarrow \{\mathbf{x}_{goal}\}, E_F \leftarrow \emptyset, \mathcal{T}_F = (V_F, E_F)$
- 2  $\text{key}_{\mathcal{R}}^{GIT^*} \leftarrow \text{bestKey}(\psi^*)$
- 3 **while not** terminateCondition() **do**
- 4    $X_{sampled} \leftarrow X_{sampled} \cup \text{sample}()$
- 5    $\mathcal{T}_R \leftarrow \text{reverseSearch}(\text{key}_{\mathcal{R}}^{GIT^*}, \varepsilon_{infl}^*, \varepsilon_{trunc}^*)$
- 6   **while** couldImproveForwardSearch( $\mathcal{T}_R$ ) **do**
- 7      $E_F \leftarrow \text{forwardSearch}(\mathcal{T}_R, \varepsilon_{trunc}^*)$
- 8     **if** pathFound( $E_F$ ) **then**
- 9        $\mathcal{T}_F \leftarrow \mathcal{T}_R$
- 10    **else**
- 11      $\mathcal{T}_R \leftarrow \text{updateReverseSearch}()$
- 12     $\mathcal{T}_F \leftarrow \text{prune}(\mathcal{T}_F)$
- 13 **return**  $\mathcal{T}_F$

---

**Algorithm 3:** GIT\*: Potential energy

---

- 1  $U[\mathbf{x}] \leftarrow \emptyset$
- 2  $U_{attr}[\mathbf{x}] \leftarrow \text{calcAttractiveEnergy}(\mathbf{x}_{start}, \mathbf{x})$   
// Eq. (17)
- 3 **foreach**  $\mathbf{x}_{invalid} \in X_{invalid}$  **do**
- 4    $U_{rep}[\mathbf{x}] \leftarrow \text{calcRepulsiveEnergy}(\mathbf{x}_{invalid}, \mathbf{x})$   
// Eq. (15)
- 5  $U[\mathbf{x}] \leftarrow U_{rep}[\mathbf{x}] + U_{attr}[\mathbf{x}]$
- 6 **return**  $U[\mathbf{x}]$

---

and bonuses, each multiplied by their respective weights. The total score is expressed as:

$$s_{total}^\theta = \sum_{i=1}^n (s_{base}[m^i] + s_{bonus}[m^i]) \cdot w[m^i] \quad (9)$$

The above rules evaluate an individual's total score within a specific problem context. To measure generalizability, we use randomly generated problem descriptions as a dataset  $D_{benchmark}$ . The average total scores within this dataset are included in the fitness calculation. To ensure stability across problems, we include the variance of total scores. Lastly, we consider the number of nodes as a complexity measure to avoid overfitting from complex expressions. The final fitness calculation formula is as follows:

$$\rho_\psi = \overline{s_{total}} + c_1 \sigma_{s_{total}}^2 + c_2 |\psi| \quad (10)$$

where  $\psi$  denotes the individual.  $\overline{s_{total}}$  represents the mean of the total scores across each problem definition in the benchmark.  $\sigma_{s_{total}}^2$  is the variance of the total scores, multiplied by the coefficient  $c_1$ .  $|\psi|$  signifies the size of the individual  $\psi$ , multiplied by the coefficient  $c_2$ .

During practical training, techniques can reduce unnecessary computations. A segmented system can evaluate an individual without testing the complete benchmark. The benchmark's scenarios are divided into segments with increasing difficulty. If the fitness in the first  $i$  segments is significantly lower than a baseline, it indicates the algorithm performs worse than the expected ideal threshold ( $EIT^*$ ). Consequently, the fitness score can be directly assessed and recorded as  $L_\psi$  without testing the entire benchmark, as such an individual is likely to be quickly eliminated in the evolutionary process.

**Table 1**  
Example performance results for **randomly generated** environment.

Weights $w[m^l]$	$t_{init}^{min}$ 1.0	$t_{init}^{med}$ 3.5	$t_{init}^{max}$ 0.5	$c_{init}^{min}$ 1.0	$c_{init}^{med}$ 2.5	$c_{init}^{max}$ 1.0	$c_{final}^{min}$ 1.0	$c_{final}^{med}$ 2.5	$c_{final}^{max}$ 1.0	Success 3.0
EIT*	0.19	$\infty$	$\infty$	2.5	$\infty$	$\infty$	2.5	$\infty$	$\infty$	0.48
GIT* $\psi_i$	0.16	0.39	$\infty$	2.34	5.05	$\infty$	2.33	3.53	$\infty$	0.72

### 4.3. Genetic Informed Trees (GIT\*)

In Section 4.2, we use the RGP to evaluate the best individual  $\psi^*$  of the generated population. In this subsection, the evaluated best individual  $\psi^*$  is utilized in the GIT\* to guide robot path planning, allowing the robot to rapidly converge on the initial solution while maintaining path quality.

Problem-specific information falls into three categories: search tree information  $g(\mathbf{x})$ , heuristic information  $\hat{h}(\mathbf{x})$ , and environmental information (e.g., dimensionality  $D(\theta)$  and obstacle details). GIT\* uses the RGP to generate evolving individuals that combine these information types into complex expressions. These expressions are integrated into the EIT\* heuristic function to form new algorithms,  $GIT_{\psi^*}^*$ , with the optimal GIT\* algorithm being selected based on performance:

$$\psi^* := \arg \min_{\psi_i} \rho_{\psi_i} \quad (11)$$

$$GIT^* := GIT_{\psi^*}^* \quad (12)$$

When GIT\* trains its heuristic function using RGP, information is stored in the primitive set to generate individuals. The search tree-related information includes  $g(\mathbf{x}_s)$ , while prior heuristic information includes  $\hat{h}(\mathbf{x}_t)$  and  $\hat{c}(\mathbf{x}_s, \mathbf{x}_t)$ .  $\bar{e}(\mathbf{x}_s)$  estimates the effort to find and validate a path from  $\mathbf{x}_s$  to the goal, whereas  $\bar{e}(\mathbf{x}_s, \mathbf{x}_t)$  estimates the computational effort required to find and validate a path between states, while  $\bar{d}(\mathbf{x}_t)$  estimates the effort from  $\mathbf{x}_t$  to the start. Environmental information comprises not only  $D(\theta)$  but also two variables that record information about obstacles and the dynamic importance of states. According to the GA model, after natural selection, the **winner G-heuristic** function generated by RGP can be equivalently represented by  $\text{key}_{\mathcal{R}}^{GIT^*}$ , which extracts the next edge from the reverse queue:

$$\text{key}_{\mathcal{R}}^{GIT^*}(\mathbf{x}_s, \mathbf{x}_t) := \begin{cases} (\hat{g}(\mathbf{x}_t) - \pi) \times \frac{\log(1+|U[\mathbf{x}_t]-U[\mathbf{x}_s]|)}{1+w_{dyn}[\mathbf{x}_t]} \\ \sqrt{\bar{e}(\mathbf{x}_s) + \bar{e}(\mathbf{x}_s, \mathbf{x}_t)} \times \log(\bar{d}(\mathbf{x}_t)) \end{cases} \quad (13)$$

where  $U[\mathbf{x}_t]$  refers to the potential energy of the current state in an artificial potential field, and  $w_{dyn}$  refers to dynamic importance, represented by the number of times the current state has been visited. The following will detail how these variables are obtained.

#### 4.3.1. Potential field variable $U[\mathbf{x}_t]$

Understanding obstacle characteristics like shapes, numbers, and locations is crucial for guiding the search tree to either circumvent obstacles for quicker solutions or approach them to reduce costs. However, these characteristics are often unknown beforehand.

GIT\* approximates the environment by sampling points in the  $C$ -space to acquire information about obstacles, denoted as  $X_{obs}$ . Those randomly sampled points undergo a validity check (e.g., collision detection) to determine if they are inside obstacles. Invalid points, denoted as  $\mathbf{x}_{invalid}$ , indicate locations within obstacles, gradually outlining their shapes and locations as sampling increases. GIT\* also employs the APF method to conceptualize the navigation space as a force field where obstacles generate repulsive forces, and targets generate attractive forces (Alg. 3, line 2).  $\mathbf{x}_{invalid}$  and  $\mathbf{x}_{goal}$  generate repulsive and attractive forces with target state  $\mathbf{x}_t$ , respectively, and the potential field is dynamically

#### Algorithm 4: GIT\*: Dynamic importance

```

1  $w_{dyn}[\mathbf{x}_t] \leftarrow \emptyset$ 
2 foreach  $\mathbf{x}_{neighbor} \in \text{neighbors}(\mathbf{x}_t)$  do
3   if  $\text{inReverseTree}(\mathbf{x}_{neighbor}, \mathbf{x}_t)$ 
4      $w_{dyn}[\mathbf{x}_t] \leftarrow w_{dyn}[\mathbf{x}_t] + 1$ 
5 return  $w_{dyn}[\mathbf{x}_t]$ 

```

adjusted based on the obstacle data. The calculated data is then utilized in the primitive set as candidates for RGP to generate G-heuristic individuals.

- **Repulsive force:** Generated around invalid samples, these forces prevent entry into these areas. The magnitude of the repulsive force is:

$$F_{rep}(q) := \begin{cases} -\frac{k_r \cdot q \cdot q_{obs}}{r^2} & \text{if } r \leq \rho_0 \\ 0 & \text{otherwise} \end{cases}, \quad (14)$$

where  $k_r$  is a proportionality constant,  $q$  is the charge equivalent of the path planner,  $q_{obs}$  is the charge equivalent of the obstacle,  $r$  is the distance between the path planner and the obstacle, and  $\rho_0$  is the threshold distance beyond which the force is not exerted.

- **Repulsive potential energy:** The potential energy is:

$$U_{rep}(q) := \begin{cases} -\frac{k_r \cdot q \cdot q_{obs}}{r} & \text{if } r \leq \rho_0 \\ 0 & \text{otherwise} \end{cases}, \quad (15)$$

- **Attractive force:** Produced by the target, these forces guide the path planner towards the target, navigating around repulsive regions. The magnitude of the attractive force is:

$$F_{attr}(q) := \frac{k_a \cdot q \cdot q_{goal}}{r^2} \quad (16)$$

where  $k_a$  is another proportionality constant,  $q$  is the charge equivalent of the path planner,  $q_{goal}$  is the charge equivalent of the target, and  $r$  is the distance between the path planner and the target.

- **Attractive potential energy:** The potential energy is:

$$U_{attr}(q) := \frac{k_a \cdot q \cdot q_{goal}}{r} \quad (17)$$

The potential energy in the APF can be calculated using these formulae, recording information about obstacles and incorporating it into the primitive set to construct the heuristic function. As potential energy increases, indicating proximity to obstacles, the heuristic function's value increases, reducing the likelihood of state selection. When  $U[\mathbf{x}_t]$  is high, indicating frequent visits, the heuristic function's value decreases, increasing the likelihood of exploration. As  $\hat{g}(\mathbf{x}_t)$  increases, indicating greater distance from the start, the heuristic function's value increases, making the node less likely to be searched.

#### 4.3.2. Dynamic importance variable $w_{dyn}[\mathbf{x}_t]$

In incremental asymptotically sampling-based planners like GIT\*, certain sample points in  $C$ -space are frequently visited, often in nearest neighbor areas (e.g., path rewire) of path planning.

**Algorithm 5:** GIT\*: Nearest neighbors

---

```

1  $X_{neighbors} \leftarrow \text{nearest}(\mathbf{x}_t)$ 
2  $X_{neighbors} \stackrel{+}{\leftarrow} \{\text{parent}(\mathbf{x}_t) \cup \text{children}(\mathbf{x}_t) \setminus X_{neighbors}\}$ 
3  $X_{neighbors} \stackrel{-}{\leftarrow} \{\mathbf{x}_s \in X_{neighbors} | (\mathbf{x}_s, \mathbf{x}_t) \in E_{invalid}\}$ 
4 return  $X_{neighbors}$ 

```

---

These samples may lie along essential routes between start and end states, serve as conduits connecting regions, and could be located in narrow corridor areas. Thus, frequently visited samples in the free space prior to neighboring areas guide the search into explore-worthy regions, which improves search efficiency. GIT\* tracks the number of visits to each sample point, capturing its dynamic importance (Alg. 4 and 5), and navigates to higher importance states. These strategies help GIT\* search more efficiently during the path optimization phase. Similar to the APF discussed in Section 4.3.1, the number of visits (i.e., dynamic importance) to a state is included in RGP's primitive set to generate G-heuristics.

Formally, the dynamic importance of a state  $\mathbf{x}_t$ , denoted as  $w_{dyn}[\mathbf{x}_t]$ , is calculated as follows:

$$w_{dyn}[\mathbf{x}_t] := \sum_{\mathbf{x}_{neighbor} \in X_{neighbors}(\mathbf{x}_t)} \mathbb{I}(\mathbf{x}_{neighbor} \in \mathcal{T}_{\mathcal{R}}(X_{neighbors}(\mathbf{x}_t))) \quad (18)$$

where  $\mathbb{I}(\cdot)$  is the indicator function that equals 1 if the condition is true and 0 otherwise.

Each time a sample point appears in the nearest neighbors of the reverse tree queue, the dynamic importance of the corresponding state is incremented by 1, emphasizing frequently visited states for path optimization. Furthermore, the inflation factor speeds up the search by biasing the goal, resulting in rapid initial solutions. The truncation factor optimizes the search by stopping it when the solution quality is satisfied.

#### 4.3.3. Inflation and truncation factor function

The traditional inflation and truncation factor update strategy is a user-adjustable parameter that can be tailored to specific application scenarios and requirements. However, this strategy lacks flexibility as it requires manual adjustments in each scenario to achieve optimal performance. The updated function for the inflation factor derived from this training session is:

$$\varepsilon_{infl}^* = 1.0 + \frac{\log(D(\theta)) + \sqrt{D(\theta)}}{\sqrt{N_{samples}} + \log(N_{samples}) + 1} \quad (19)$$

where  $D(\theta)$  is the dimensionality of the path planning problem  $\theta$ , and  $N_{samples}$  is the current number of samples taken.

As the problem's dimensionality increases, this expression's value also increases, biasing the search towards rapidly finding feasible solutions based on heuristics rather than ensuring the lowest cost solution. In higher-dimensional spaces, fewer obstacles relative to the overall space decrease the probability of blocking the path, enhancing the success rate and reducing the time to find initial solutions. As the number of samples increases, the value decreases, leading GIT\* to focus on low-cost solutions after several sampling batches, aligning with practical requirements.

The updated function for the truncation factor is derived through the RGP process, where candidate expressions are generated from a predefined primitive set and selected based on their fitness performance, which is defined as follows:

$$\varepsilon_{trunc}^* = 1.0 + \frac{3\pi}{N_{samples}} \quad (20)$$

**Table 2**

Parameter settings for the training.

Parameter name	Value or description
Population size	1500
Number of generations	100
Selection method	tournament selection
Crossover rate	0.8
Mutation rate	0.1
Maximum tree depth	4
Tournament size	5
Crossover type	Subtree crossover
Mutation type	Point mutation

where  $N_{samples}$  represents the current number of samples taken.

As  $N_{samples}$  increases, the value decreases, indicating a tendency to exploit the current approximation rather than explore new ones. This is suitable for the later stages of the search when  $N_{samples}$  is large.

## 5. Analysis

In this section, firstly, we provide the convergency analysis and asymptotical time to prove the feasibility of the proposed algorithm. In addition, we explain the reason that GIT\* consumes less time complexity, and we also verify the advantage of RGP from mathematics.

### 5.1. Reinforced genetic programming training analysis

Due to the randomness of the RGP algorithm and variability in training parameters, results from each RGP instance are unique. Practical applications need to consider specific objectives, use cases, datasets, and time constraints for parameter settings. Table 2 details the chosen parameters: high population size enhances diversity but raises computational cost, 1500 size was chosen for optimal performance with our equipment; 100 generations provide a balance between solution quality and overfitting; a crossover rate of 0.8 promotes exploration without excessive disruption; a mutation rate of 0.1 maintains diversity and prevents premature convergence; a maximum tree depth of 4 avoids overfitting and underfitting; and a tournament size of 5 balances selection pressure and diversity. The fitness variation across generations is shown in Fig. 5.

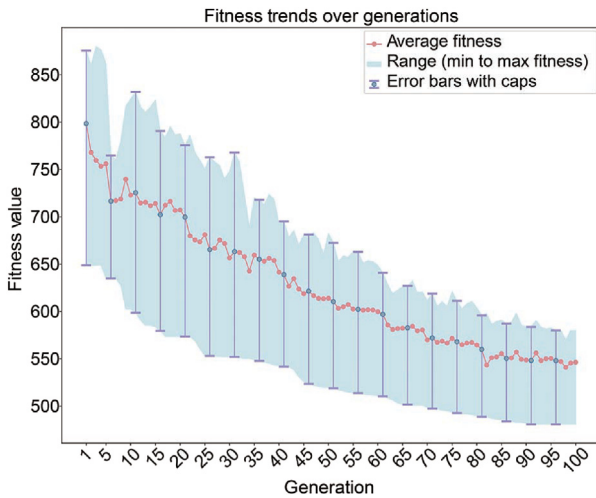
### 5.2. Proof of convergence in genetic programming

Research has explored the convergence properties of genetic programming (GP) for symbolic regression [52]. The global optimum in symbolic regression problem refers to the best possible individual that achieves the minimum or maximum fitness of the objective function across the entire state space [53]. Convergence to the global optimum implies generating solutions where the global optimum emerges as a limit. This study adopts a probabilistic interpretation. Rudolph [44] modeled genetic programming using a Markov Chain framework and demonstrated convergence when the population retains the best solution. The natural selection, crossover, and mutation processes in GP mimic biological evolution.

Let  $\mathcal{P}(t)$  be the population at time  $t$ , and  $\psi^*$  be the global optimum. GP maintains a diverse population  $\mathcal{P}(t)$  over generations to escape local optima:

$$\mathcal{P}(t+1) := \text{select}(\text{crossover}(\text{mutate}(\mathcal{P}(t)))) \quad (21)$$

This helps GP escape local optima, unlike greedy search methods, which may converge quickly to local optima.



**Fig. 5.** Evolution of fitness over generations during genetic programming training. The light purple error bar represents the maximum/minimum fitness value within each generation, while the light orange line depicts the average fitness value across individuals within each generation. The horizontal axis represents the generations, and the vertical axis represents the fitness values.

Let  $Z_t$  denote a sequence of random variables representing the best fitness within a population at step  $t$ . The convergence property of genetic programming, which preserves the best solution in the population, can then be formalized as:

$$\lim_{t \rightarrow \infty} \mathbb{P}(Z_t = \psi^*) := 1 \quad (22)$$

where  $\psi^*$  represents the global optimum. This expression indicates that the probability of the best fitness  $Z_t$  equating to the global optimum  $\psi^*$  approaches unity as the number of iterations steps  $t$  approaches infinity.

Through mutation and crossover, GP maintains diversity and explores the search space effectively. Selection mechanisms favor individuals with higher fitness, leading to gradual improvement. Consequently, the probability of finding the global optimum  $\psi^*$  increases with each iteration.

### 5.3. Probabilistic completeness and asymptotic optimality

Most informed tree-based path planning algorithms have been proven to be probabilistically complete and asymptotically optimal, and GIT\* can also guarantee these two properties. GIT\* utilizes uniform sampling strategies. As the number of iterations  $n$  approaches infinity, the entire state space will be explored, satisfying the following equation:

$$\lim_{n \rightarrow \infty} \mathbb{P}(\{V_{\mathcal{F}} \cup V_{\mathcal{R}}\} \cap X_{\text{goal}}) \neq \emptyset = 1 \quad (23)$$

which means that if there is a feasible path, it must be found by the GIT\*. Therefore, the probabilistic completeness of the optimal path planner is guaranteed.

The GIT\* implements the same Choose Parent and Rewire strategies as the EIT\*. It means that if the rewiring radius  $r(q)$  in Choose Parent and Rewire processes satisfies:

$$r(q) > \eta \left( 2 \left( 1 + \frac{1}{d} \right) \left( \frac{\lambda(X_f)}{\zeta_d} \right) \left( \frac{\log(q)}{q} \right) \right)^{\frac{1}{d}} \quad (24)$$

here,  $q$  denotes the number of sampled states in the informed set,  $\eta > 1$  is a tuning parameter,  $\lambda(\cdot)$  denotes the Lebesgue measure, and  $d$  is the dimensionality of the workspace,  $\lambda(X_f)$  is the Lebesgue measure of informed set  $X_f$  and  $\zeta_d$  is the volume of

unit ball in current workspace. In reference to Lemma 56, 71 and 72 in [6], the following equation holds:

$$\mathbb{P}(\limsup_{q \rightarrow \infty} \min_{\sigma \in \Sigma_q} \{c(\sigma)\} = c^*) = 1 \quad (25)$$

where  $q$  is the number of samples,  $\Sigma_q \subset \Sigma$  is the set of valid paths from the start to the goal found by the planner from those samples,  $c : \Sigma \rightarrow [0, \infty)$  is the cost function, and  $c^*$  is the optimal solution cost. It indicates that the GIT\* can find an optimal path, if it exists, as the number of iterations go to infinity. Therefore, the asymptotic optimality is guaranteed.

## 6. Experiments

In this paper, we utilize the Planner Developer Tools (PDT) [48] and MoveIt [54] to benchmark motion planner behaviors. GIT\* was tested against SOTA algorithms in both simulated random scenarios (Fig. 6) and real-world manipulation problems (Fig. 8). The comparison involved several versions of RRT-Connect, Informed RRT\*, BIT\*, AIT\*, ABIT\*, and EIT\* sourced from the Open Motion Planning Library (OMPL) [46]. The evaluations were conducted on a computer with an Intel i7 3.90 GHz processor and 32 GB of LPDDR3 3200 MHz memory. These comparisons were carried out in simulated environments of dimensions  $\mathbb{R}^4$  and  $\mathbb{R}^8$ . The primary objective for the planners was to minimize path length (cost). The RGG constant  $\eta$  was uniformly set to 1.001, and the rewiring factor was set to 1.2 for all planners.

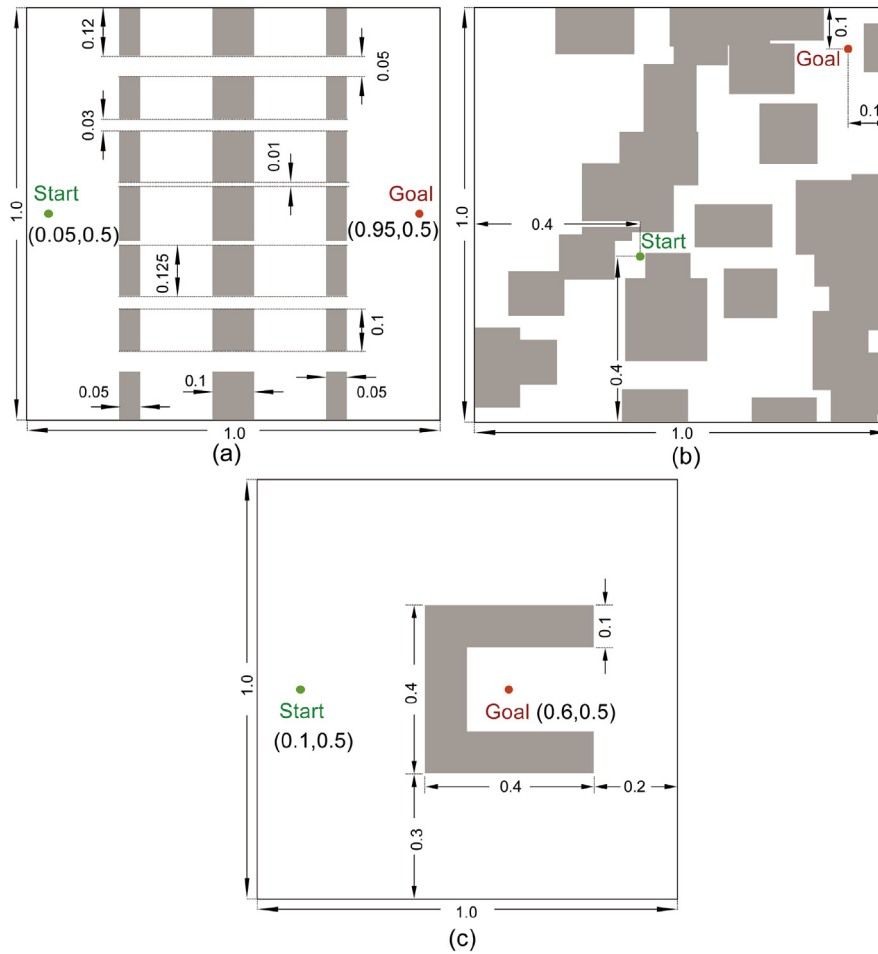
In the case of RRT-based algorithms, a goal bias of 5% was employed, and the maximum edge lengths were determined based on the dimensionality of the space. All batched algorithms utilized a batch size of 100. BIT\*, AIT\*, ABIT\*, and EIT\* maintained a linear combination heuristic function of Euclidean distance and effort, respectively. GIT\* utilized optimal G-heuristic (Eq. (13)) to extract the next edge from the reverse queue, which was selected based on the fitness of RGP.

### 6.1. Simulation experimental tasks

The planners were tested across three distinct benchmarks in two domains:  $\mathbb{R}^4$  and  $\mathbb{R}^8$ . In the first scenario, a constrained environment resembling a dividing wall with several narrow gaps was simulated, allowing valid paths in multiple general directions for non-intersecting solutions (Fig. 6(a)). Each planner underwent 100 runs, with computation time for each anytime asymptotically optimal planner shown in the labels, using varying random seeds. The overall success rates and median path lengths for all planners are depicted in Fig. 7(a) and 7(b). It can be seen that GIT\* quickly finds the initial solution in both dimensions with minimal time, whereas EIT\* requires more time to find the initial solution.

In the second test scenario, random widths were assigned to axis-aligned hyperrectangles, generated arbitrarily within the  $C$ -space (Fig. 6(b)). Random rectangle problems were created for each dimension of the  $C$ -space, with each planner undergoing 100 runs for every instance. Fig. 7c and 7d illustrate the proposed method has the highest success rates and lowest median path costs within the computation time compared with other planners. This indicates that GIT\* can recognize promising regions via environmental information (e.g., APF) where feasible paths likely lie, thereby biasing the sampling process towards these regions. As a result, GIT\* outperformed and can quickly find an initial solution.

The last test problem consisted of a hollow, axis-aligned hyperrectangle enclosing the goal state, configured such that even in higher dimensions, the goal can only be reached through the face of the hyperrectangle farthest from the start state (Fig. 6(c)). This problem is challenging for GIT\* because there are many invalid edges close to the root of the reverse search tree, often requiring



**Fig. 6.** The 2D representation of the simulated planning problems in Section 6. (a) Dividing walls. (b) Random rectangles. (c) Goal enclosure. The state space, denoted as  $X \subset \mathbb{R}^p$ , is constrained within a hypercube with one width for both problem instances. Specifically, we conducted ten distinct instantiations of the random rectangles experiment and the outcomes are showcased in Fig. 7.

large parts to be repaired (Figs. 7(e)–(f)). From the figure, the GIT\* achieves the best performance in finding the initial solution and converging to the optimal solution compared with the SOTA planner.

As observed in Table 3, there is a median initial time improvement across varied benchmark scenarios, correlating with dimensionality. For instance, in the DW –  $\mathbb{R}^4$  scenario, GIT\* exhibits a lower initial median time (i.e., median value over 100 trials) of 0.0201s compared to 0.0252s for EIT\* and 0.1299s for AIT\*. This trend is consistent across other scenarios, such as RR –  $\mathbb{R}^4$  and GE –  $\mathbb{R}^4$ , where GIT\* consistently shows reduced initial median times.

In the GE –  $\mathbb{R}^8$  scenario, GIT\* demonstrates an initial median time of 0.0512s, compared to 0.0941 for EIT\* and 0.3834s for AIT\*. This indicates an improvement in initial convergence time of approximately 45.59% compared to EIT\*.

Overall, Table 3 highlights the advantages of GIT\* in achieving lower initial median times compared to SOTA, thereby enhancing the efficiency of path planning algorithms.

## 6.2. Real-world path planning tasks

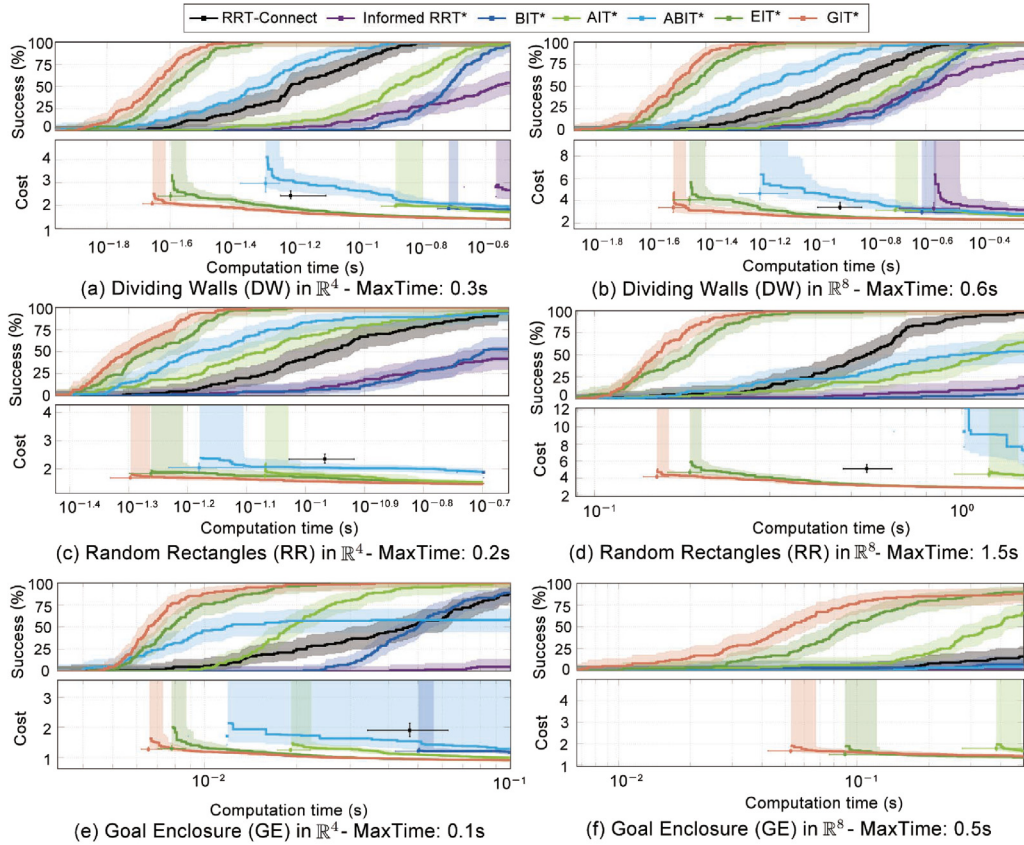
To evaluate the algorithm’s performance in real-world scenarios, three numerical experiments are conducted on a single-arm manipulator and mobile manipulator (DARKO) to demonstrate the efficiency and extensibility of GIT\* compared with three SOTA path planning algorithms: Batch Informed Trees (BIT\*) [36,37],

Adaptively Informed Trees (AIT\*) [18,40], and Effort Informed Trees (EIT\*) [18].

We compare GIT\* with AIT\* and EIT\* in single-arm manipulator environments to evaluate their performance in converging to the optimal solution cost and success rate over 30 runs. The first environment (*Beer Barrel*) consists of simple cup holder obstacles. The second and third environments (*Shelf* and *Kitchen*) are confined to the DARKO robot and cluttered with narrow spaces. A collision-free path connecting the start state to the goal region is required. GIT\* demonstrated its effective G-heuristic during multiple experimental tasks (Fig. 8). The detailed behavior of real-world experiments can be viewed in the accompanying video.

### 6.2.1. Beer barrel cup placement task

Fig. 8a showcases the start and goal configuration of the cup placement task. In this task, we utilize a single robotic manipulator to grab a beer cup and place it under the beer tap of the beer barrel keg while avoiding obstacles. The following graph illustrates the performance of AIT\*, EIT\*, and GIT\* in terms of solution cost and success rate. All planners were given 1.0 s to address the beer barrel cup placement problem. Over the course of 30 trials, GIT\* achieved a 100% success rate with a median solution cost of 13.8972. EIT\* had a success rate of 96.67% with a median solution cost of 15.1332. AIT\* was 93.33% successful, with a median solution cost of 19.2183.



**Fig. 7.** Detailed experimental results from Section 6.1 are presented above. MaxTime is the planner's maximum allotted planning time. (a) and (b) depict test benchmark dividing walls outcomes in  $\mathbb{R}^4$  and  $\mathbb{R}^8$ , respectively. Panel (c) showcases random rectangle experiments in  $\mathbb{R}^4$ , while panels (d) demonstrate in  $\mathbb{R}^8$ . Panel (e) and (f) present goal enclosure experiments in  $\mathbb{R}^4$  and  $\mathbb{R}^8$ . In the cost plots, boxes represent solution cost and time, with lines showing cost progression for optimal planners (unsuccessful runs have infinite cost). Error bars provide nonparametric 99% confidence intervals for solution cost and time.

**Table 3**  
Benchmarks evaluation comparison (Fig. 7).

Benchmark	Adaptively informed trees			Effort informed trees			Genetic informed trees			$t_{init}^{med} \uparrow\uparrow$ (%)
	$t_{init}^{med}$	$c_{init}^{med}$	$c_{final}^{med}$	$t_{init}^{med}$	$c_{init}^{med}$	$c_{final}^{med}$	$t_{init}^{med}$	$c_{init}^{med}$	$c_{final}^{med}$	
DW - $\mathbb{R}^4$	<b>0.1299</b>	1.9571	1.7151	<b>0.0252</b>	2.4051	1.3693	<b>0.0201</b>	2.0619	1.3634	<b>84.53/20.23</b>
DW - $\mathbb{R}^8$	<b>0.1947</b>	3.1492	2.6388	<b>0.0357</b>	4.0910	2.2892	<b>0.0279</b>	3.3791	2.3109	<b>85.67/21.84</b>
RR - $\mathbb{R}^4$	<b>0.0853</b>	1.7570	1.5282	<b>0.0587</b>	1.8392	1.4715	<b>0.0472</b>	1.6874	1.4595	<b>44.67/19.59</b>
RR - $\mathbb{R}^8$	<b>1.1843</b>	4.4697	4.3599	<b>0.1889</b>	4.6789	2.8588	<b>0.1429</b>	4.1716	2.8450	<b>87.93/24.35</b>
GE - $\mathbb{R}^4$	<b>0.0191</b>	1.2457	0.9900	<b>0.0082</b>	1.2909	0.9126	<b>0.0064</b>	1.2678	0.9083	<b>66.49/21.95</b>
GE - $\mathbb{R}^8$	<b>0.3834</b>	1.7854	1.6605	<b>0.0941</b>	1.4970	1.4086	<b>0.0512</b>	1.6634	1.3636	<b>86.64/45.59</b>

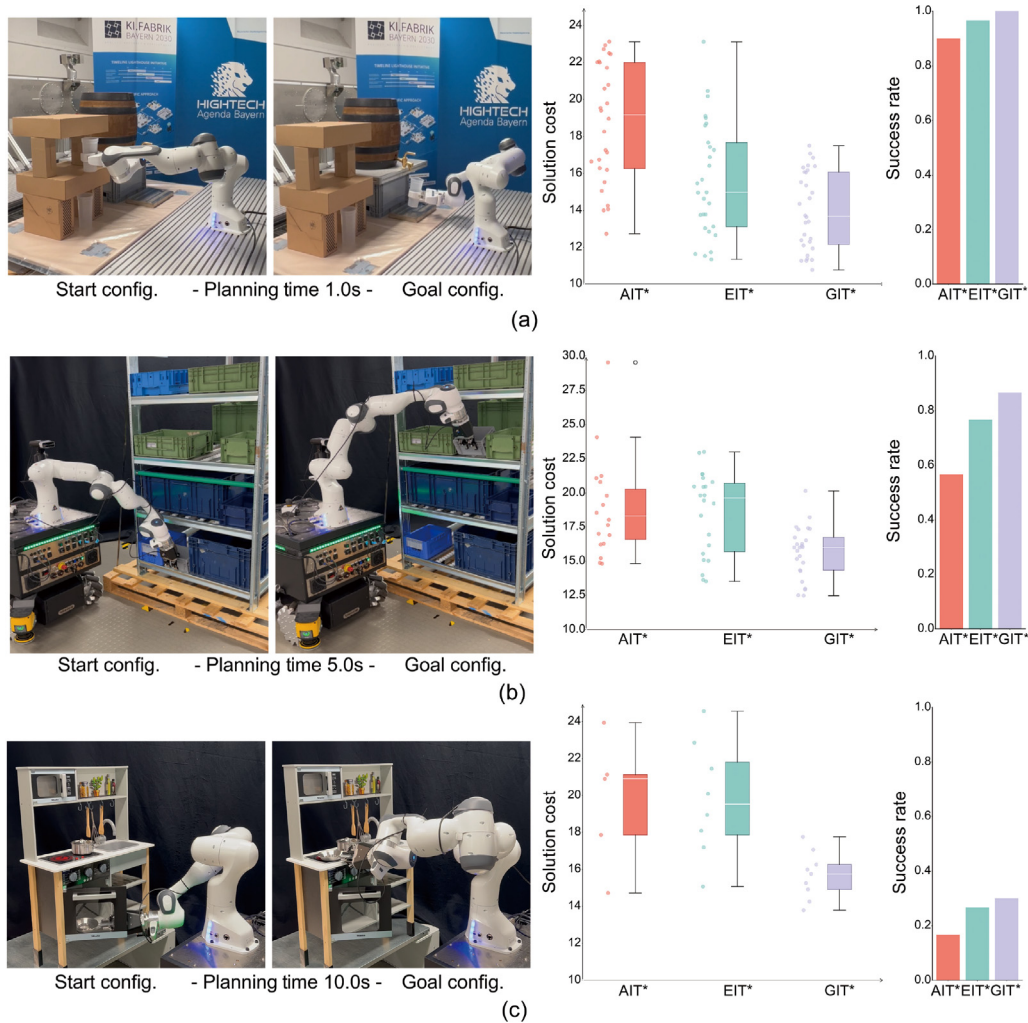
### 6.2.2. Industry shelf container rearrangement task

The initial and final configurations for the shelf task are depicted in Fig. 8b. This task involves extracting an industry-standard container from a position between two other boxes on the lower bottom layer and repositioning it on the third layer of the shelf, again between two containers. Due to component standardization, the challenge lies in the precise insertion of industry containers into narrow spaces. The task aims to place the industry-standard container between two larger containers on the shelf, with a tolerance scope of  $\leq 5$  mm, making the planning of a collision-free feasible path particularly difficult. Each planner was allocated 5.0 s to solve this confined, limited space pull-out and insertion problem. Across 30 trials, GIT\* achieved an 86.67% success rate with a median solution cost of 15.9745. EIT\* had a 76.67% success rate with a median solution cost of 19.1045. AIT\* managed a 56.67% success rate with a median solution cost of 18.2672.

### 6.2.3. Kitchen model pan cooking task

For the third task, we utilized the DARKO robot positioned in front of a kitchen model. The start and goal configurations are illustrated in Fig. 8c. This task is particularly challenging as the manipulator must navigate the geometric shape of the pan within a cluttered oven while also avoiding collisions between the base robot and the kitchen shelves. The complexity is further heightened by the need for precise movements in a confined space. Each planner was allotted 10.0 s to solve this kitchen pan reallocation problem. Over the course of 30 trials, GIT\* achieved a 30% success rate with a median solution cost of 15.8860. EIT\* had a success rate of 26.67% with a median solution cost of 19.2746. AIT\* managed a 16.67% success rate with a median solution cost of 20.9824.

In short, compared with the AIT\* and the EIT\*, the GIT\* achieves the best performance in finding the initial solution and converging to the optimal solution.



**Fig. 8.** Detailed experimental results from Section 6.2 are summarized above. (a) illustrates the *beer barrel* ENV, highlighting the start and goal configurations along with the solution cost and success rate. (b) depicts the *industry shelf* ENV, showing the initial and final positions for extracting and placing an industry-standard container. (c) presents the *kitchen* ENV, focusing on the DARKO robot's performance. In the cost box plots, boxes indicate the solution cost per planner, while lines represent the mean cost progression for an optimal planner (unsuccessful runs are assigned an infinite cost).

### 6.3. Discussion

In our approach, we compare the optimized individuals with EIT\* using various performance metrics, such as solution cost and initial solution time, which are weighted to provide a comprehensive evaluation. When multiple individuals outperform the baseline, we select the one with the highest fitness score. A self-adaptive mechanism is naturally incorporated through the genetic algorithm, as superior individuals evolve through selection, crossover, and mutation in subsequent cycles.

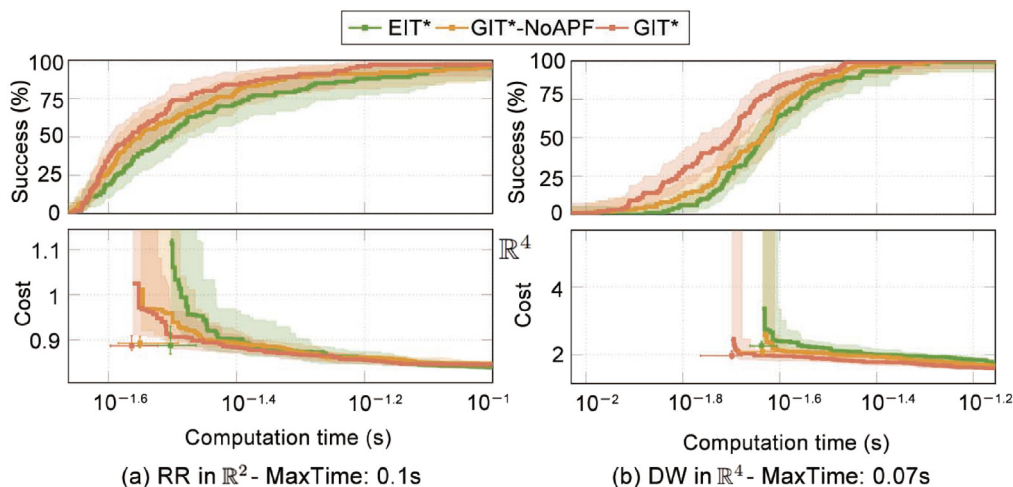
#### 6.3.1. Comparison with SOTA planner

To showcase the advantages of GIT\*, we compared its performance with AIT\* and EIT\* using success rate and solution cost metrics in three real-world tasks (Fig. 8): placing cups on beer barrel faucets, rearranging industrial containers on shelves, and cooking pans in a kitchen model, and six simulation tasks (Fig. 6) across multi-dimensions with randomly generated seeds.

From the experiment results, we observe that EIT\* performs much better than AIT\* in both simulation environments (Table 3) and real-world scenarios. However, GIT\* outperforms SOTA planners due to its use of problem-specific environmental information via RGP and the integration of the G-heuristic. As shown in Fig. 7(a), (c), and (e)), in low-dimensional problem domains, the

initial solution finding time and cost show minimal improvement. In high-dimensional domains, the linear combination heuristic struggles to guide the search efficiently, as shown in Fig. 7(b), (d), and (f). Furthermore, in the first real-world environment, GIT\* outperformed EIT\* by 3.33% in success rate and reduced the solution cost by approximately 8.17%. Compared to AIT\*, GIT\* improved the success rate by 6.67% and reduced the solution cost by approximately 27.68%, as shown in Fig. 8(a). In the second real-world experiment, the benchmark results show that the G-heuristic can enhance solving cluttered tasks, achieving the highest success rate and the lowest solution cost among the evaluated planners. GIT\* outperformed EIT\* by 13.04% in success rate and reduced the solution cost by approximately 16.38%. Compared to AIT\*, GIT\* improved the success rate by 52.94% and reduced the solution cost by approximately 12.56%, as shown in Fig. 8(b). In the third real-world experiment, GIT\* outperformed EIT\* by 12.5% in success rate and reduced solution cost by about 17.58%. Compared to AIT\*, GIT\* improved success rate by 80% and reduced solution cost by approximately 24.32%, as shown in Fig. 8(c). These results highlight the effectiveness of the G-heuristic in narrow environments to prevent obstacle avoidance in the kitchen model.

From the discussion, one may conclude that using RGP to train an optimal G-heuristic across all benchmarks can improve the



**Fig. 9.** The detailed ablation study results from Section 6.3.2 are summarized. The cost plots illustrate both solution cost and time, with lines representing the optimal cost progression and error bars indicating 99% confidence intervals.

initial convergence rate and initial path length. Furthermore, GIT\* can utilize environmental information to search via more promising regions (i.e., APF and dynamic importance), which accelerates the path-planning initial finding process. GIT\* achieved the highest success rate and lowest solution cost among tested SOTA planners, emphasizing its potential for real-world applications.

### 6.3.2. Ablation studies

To evaluate the individual contributions of the proposed planner, we conducted an ablation study focusing on the artificial potential field (GIT\*-NoAPF) in two representative scenarios (Fig. 9): random rectangles (RR)- $\mathbb{R}^2$  and dividing walls (DW)- $\mathbb{R}^4$ . In the RR scenario, which contains few narrow passages, the APF offers limited performance improvement. In contrast, in the DW scenario—characterized by numerous narrow passages—the APF provides clear benefits. The proposed APF-based planner leverages the repulsive forces generated by obstacles to quickly identify optimal paths through narrow gaps. Notably, GIT\*-NoAPF performs comparably to the baseline EIT\* in this setting, highlighting the APF's effectiveness in navigating complex and cluttered environments. Overall, these results emphasize the APF's pivotal role in enhancing planner performance, particularly in dense environments with narrow passages.

### 6.3.3. Limitations and future work

While GIT\* demonstrates superior performance, it has certain limitations. The current implementation is tailored to specific tasks with predefined start and goal configurations, which limits its adaptability to more dynamic and variable environments. Future work could enhance GIT\*'s generalization capabilities by integrating neural network-driven approaches to learn from diverse human demonstrations, thus improving its flexibility across different tasks and environments. This would align with recent advancements in neural network-based path planning, further enhancing GIT\*'s robustness and versatility. Moreover, future research could explore aspects such as computational efficiency and performance under various environmental conditions to provide a more balanced comparison with traditional methods. Additionally, considerations regarding human acceptability, risk-awareness, and safety could be integrated when planning trajectories for robots, helping to further refine the practical applicability of GIT\*. This expanded analysis would help highlight both the strengths and potential limitations of the proposed approach in a broader context.

## 7. Conclusion

In this paper, we introduced the GIT\* algorithm, a novel path planning approach that leverages RGP to refine heuristic functions for enhanced guidance. By incorporating additional environmental data, such as repulsive forces from obstacles and the dynamic importance of vertices, GIT\* improves search efficiency and solution quality. The integration of RGP allows GIT\* to mutate genotype-generative heuristic functions (G-heuristic), adapting to various problem domains. Our comparative analyses demonstrate that GIT\* consistently outperforms existing single-query, sampling-based planners across different scenarios, including simulation benchmarks and real-world robot manipulation tasks. Optimal G-heuristic exhibits notable improvements over SOTA methods in terms of both success rate and solution cost, showcasing its robustness and adaptability, particularly in handling complex, cluttered environments with high precision and efficiency.

In conclusion, GIT\* leverages RGP to generate and evaluate G-heuristics, incorporating a fitness reward for training and adjustment of exploration strategies based on environmental feedback. This approach enhances rapid initial pathfinding and reduces solution costs. GIT\* shows promising potential for future research and applications in motion planning.

### CRedit authorship contribution statement

**Liding Zhang:** Conceptualization. **Kuanqi Cai:** Conceptualization. **Zhenshan Bing:** Writing – review & editing. **Chaoqun Wang:** Supervision. **Alois Knoll:** Visualization.

### Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

### Appendix A. Supplementary data

Supplementary material related to this article can be found online at <https://doi.org/10.1016/j.birob.2025.100237>.

## References

- [1] J.D. Gammell, M.P. Strub, Asymptotically optimal sampling-based motion planning methods, *Annu. Rev. Control. Robot. Auton. Syst.* 4 (2021) 295–318.
- [2] K. Cai, W. Chen, C. Wang, H. Zhang, M.Q.-H. Meng, Curiosity-based robot navigation under uncertainty in crowded environments, *IEEE Robot. Autom. Lett.* 8 (2) (2023) 800–807.
- [3] K. Cai, R. Laha, Y. Gong, L. Chen, L. Zhang, L.F. Figueredo, S. Haddadin, Demonstration to adaptation: A user-guided framework for sequential and real-time planning, in: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2024, pp. 9871–9878.
- [4] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.* 4 (2) (1968) 100–107.
- [5] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, *Int. J. Robot. Res.* 5 (1) (1986) 90–98.
- [6] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, *Int. J. Robot. Res.* 30 (7) (2011) 846–894.
- [7] K. Cai, W. Chen, D. Dugas, R. Siegart, J.J. Chung, Sampling-based path planning in highly dynamic and crowded pedestrian flow, *IEEE Trans. Intell. Transp. Syst.* 24 (12) (2023) 14732–14742.
- [8] L. Zhang, K. Cai, Z. Sun, Z. Bing, C. Wang, L. Figueredo, S. Haddadin, A. Knoll, Motion planning for robotics: A review for sampling-based planners, *Biomim. Intell. Robot.* 5 (1) (2025) 100207.
- [9] K. Cai, C. Wang, J. Cheng, C.W. De Silva, M.Q.-H. Meng, Mobile robot path planning in dynamic environments: A survey, *Instrumentation* (2020).
- [10] S.M. LaValle, J.J. Kuffner Jr., Randomized kinodynamic planning, *Int. J. Robot. Res.* 20 (5) (2001) 378–400.
- [11] L.E. Kavraki, P. Svestka, J.-C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Trans. Robot. Autom.* 12 (4) (1996) 566–580.
- [12] K. Cai, C. Wang, S. Song, H. Chen, M.Q.-H. Meng, Risk-aware path planning under uncertainty in dynamic environments, *J. Intell. Robot. Syst.* 101 (2021) 1–15.
- [13] K. Cai, W. Chen, C. Wang, S. Song, M.Q.-H. Meng, Human-aware path planning with improved virtual Doppler method in highly dynamic environments, *IEEE Trans. Autom. Sci. Eng.* 20 (2) (2023) 1304–1321.
- [14] K.L. Downing, Reinforced genetic programming, *Genet. Program. Evolvable Mach.* 2 (2001) 259–288.
- [15] K. Sastry, D. Goldberg, G. Kendall, Genetic algorithms, in: E.K. Burke, G. Kendall (Eds.), *Search Methodologies*, Springer, Boston, MA, 2005.
- [16] J.D. Gammell, T.D. Barfoot, S.S. Srinivasa, Informed sampling for asymptotically optimal path planning, *IEEE Trans. Robot.* 34 (4) (2018) 966–984.
- [17] L. Zhang, Y. Ling, Z. Bing, F. Wu, S. Haddadin, A. Knoll, Tree-based grafting approach for bidirectional motion planning with local subsets optimization, *IEEE Robot. Autom. Lett.* 10 (6) (2025) 5815–5822.
- [18] M.P. Strub, J.D. Gammell, Adaptively informed trees (AIT\*) and effort informed trees (EIT\*): Asymmetric bidirectional sampling-based path planning, *Int. J. Robot. Res.* 41 (4) (2022) 390–417.
- [19] E. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* 1 (1) (1959) 269–271.
- [20] M. Likhachev, G. Gordon, S. Thrun, ARA\*: Anytime A\* with provable bounds on sub-optimality, in: *Advances in Neural Information Processing Systems*, vol. 16, MIT Press, 2003.
- [21] R. Bellman, *Dynamic Programming*, Princeton Univ. Press Princeton, New Jersey, 1957.
- [22] J.J. Kuffner, S.M. LaValle, RRT-connect: An efficient approach to single-query path planning, in: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, Vol. 2, IEEE, 2000, pp. 995–1001.
- [23] J. Wang, B. Li, M.Q.-H. Meng, Kinematic constrained bi-directional RRT with efficient branch pruning for robot path planning, *Expert Syst. Appl.* 170 (2021) 114541.
- [24] C. Urmson, R. Simmons, Approaches for heuristically biasing RRT growth, in: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*(Cat. No. 03CH37453), Vol. 2, IEEE, 2003, pp. 1178–1183.
- [25] S. Nayak, M.W. Otte, Bidirectional sampling-based motion planning without two-point boundary value solution, *IEEE Trans. Robot.* 38 (6) (2022) 3636–3654.
- [26] I.-B. Jeong, S.-J. Lee, J.-H. Kim, Quick-RRT\*: Triangular inequality-based implementation of RRT\* with improved initial solution and convergence rate, *Expert Syst. Appl.* 123 (2019) 82–90.
- [27] J.D. Gammell, S.S. Srinivasa, T.D. Barfoot, Informed RRT\*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2014, pp. 2997–3004.
- [28] J. Wang, T. Zhang, N. Ma, Z. Li, H. Ma, F. Meng, M.Q. Meng, A survey of learning-based robot motion planning, *IET Cyber- Syst. Robot.* 3 (2021) 302–314.
- [29] A.H. Qureshi, Y. Miao, A. Simeonov, M.C. Yip, Motion planning networks: Bridging the gap between learning-based and classical motion planners, *IEEE Trans. Robot.* 37 (1) (2021) 48–66.
- [30] J. Wang, W. Chi, C. Li, C. Wang, M.Q.-H. Meng, Neural RRT\*: Learning-based optimal path planning, *IEEE Trans. Autom. Sci. Eng.* 17 (4) (2020) 1748–1758.
- [31] Z. Huang, H. Chen, J. Pohovey, K. Driggs-Campbell, Neural informed RRT\*: Learning-based path planning with point cloud state representations under admissible ellipsoidal constraints, in: 2024 IEEE International Conference on Robotics and Automation, ICRA, 2024, pp. 8742–8748.
- [32] M. Zucker, N. Ratliff, A.D. Dragan, M. Howard, M. Vecerik, T.F.Y. Wang, J.-P. van der Smagt, J.A. Bagnell, CHOMP: Covariant Hamiltonian optimization for motion planning, *Int. J. Robot. Res.* 32 (9–10) (2013) 1164–1193.
- [33] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in: 2011 IEEE International Conference on Robotics and Automation, ICRA, Shanghai, China, 2011, pp. 4569–4574.
- [34] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking, *Int. J. Robot. Res.* 33 (9) (2014) 1251–1270.
- [35] T. Marcucci, M. Petersen, D. von Wrangel, R. Tedrake, Motion planning around obstacles with convex optimization, *Sci. Robot.* 8 (84) (2023).
- [36] J.D. Gammell, S.S. Srinivasa, T.D. Barfoot, Batch informed trees (BIT\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs, in: 2015 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2015, pp. 3067–3074.
- [37] J.D. Gammell, T.D. Barfoot, S.S. Srinivasa, Batch informed trees (BIT\*): Informed asymptotically optimal anytime search, *Int. J. Robot. Res.* 39 (5) (2020) 543–567.
- [38] M.P. Špace0mmStrub, J.D. Gammell, Advanced BIT\*(ABIT\*): Sampling-based planning with advanced graph-search techniques, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2020, pp. 130–136.
- [39] L. Zhang, Z. Bing, K. Chen, L. Chen, K. Cai, Y. Zhang, F. Wu, P. Krumbholz, Z. Yuan, S. Haddadin, A. Knoll, Flexible informed trees (FIT\*): Adaptive batch-size approach in informed sampling-based path planning, in: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2024, pp. 3146–3152.
- [40] M.P. Strub, J.D. Gammell, Adaptively informed trees (AIT): Fast asymptotically optimal path planning through adaptive heuristics, in: 2020 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2020, pp. 3191–3198.
- [41] L. Zhang, Z. Bing, Y. Zhang, K. Cai, L. Chen, F. Wu, S. Haddadin, A. Knoll, Elliptical K-nearest neighbors: Path optimization via Coulomb's law and invalid vertices in C-space obstacles, in: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2024, pp. 12032–12039.
- [42] C.-H. Wei, J.-S. Liu, Hybridizing RRT and variable-length genetic algorithm for smooth path generation, in: 2011 IEEE International Conference on Robotics and Biomimetics, 2011, pp. 626–632.
- [43] X. Wang, Genetic RRT: Asymptotically optimal sampling-based path planning via optimization of genetic algorithm, *Highlights Sci. Eng. Technol.* 43 (2023) 215–222.
- [44] G. Rudolph, Convergence analysis of canonical genetic algorithms, *IEEE Trans. Neural Netw.* 5 (1) (1994) 96–101.
- [45] G. Rudolph, *Convergence Properties of Evolutionary Algorithms*, Verlag Dr. Kovač, 1997.
- [46] I.A. Sucas, M. Moll, L.E. Kavraki, The open motion planning library, *IEEE Robot. Autom. Mag.* 19 (4) (2012) 72–82.
- [47] M. Moll, I.A. Sucas, L.E. Kavraki, Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization, *IEEE Robot. Autom. Mag.* 22 (3) (2015) 96–102.
- [48] J.D. Gammell, M.P. Strub, V.N. Hartmann, Planner developer tools (PDT): Reproducible experiments and statistical analysis for developing and testing motion planners, in: *Proceedings of the Workshop on Evaluating Motion Planning Performance (EMPP)*, IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2022.
- [49] M. Virgolin, S.P. Pissis, Symbolic regression is NP-hard, *Trans. Mach. Learn. Res.* (2022).
- [50] M. Penrose, *Random geometric graphs*, vol. 5, OUP Oxford, 2003.

- [51] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
- [52] W.B. Langdon, R. Poli, Foundations of Genetic Programming, Springer Science & Business Media, 2013.
- [53] R. Horst, H. Tuy, Global Optimization: Deterministic Approaches, Springer, 1996.
- [54] M. Görner, R. Haschke, H. Ritter, J. Zhang, Moveit! task constructor for task-level motion planning, in: IEEE International Conference on Robotics and Automation, ICRA, 2019, pp. 190–196.