Improved A-search guided tree construction for kinodynamic planning

Wang, Y.

TR2019-029 June 12, 2019

Abstract

With node selection being directed by a heuristic cost [1]–[3], A-search guided tree (AGT) is constructed on-thefly and enables fast kinodynamic planning. This work presents two variants of AGT to improve computation efficiency. An improved AGT (i-AGT) biases node expansion through prioritizing control actions, an analogy of prioritizing nodes. Focusing on node selection, a bi-directional AGT (BAGT) introduces a second tree originated from the goal in order to offer a better heuristic cost of the first tree. Effectiveness of BAGT pivots on the fact that the second tree encodes obstacles information near the goal. Case study demonstrates that i-AGT consistently reduces the complexity of the tree and improves computation efficiency; and BAGT works largely but not always, particularly with no benefit observed for simple cases.

IEEE International Conference on Robotics and Automation (ICRA)

Improved A-search guided tree construction for kinodynamic planning

Yebin Wang

Abstract— **With node selection being directed by a heuristic cost [1]–[3], A-search guided tree (AGT) is constructed on-the-fly and enables fast kinodynamic planning. This work presents two variants of AGT to improve computation efficiency. An improved AGT (i-AGT) biases node expansion through prioritizing control actions, an analogy of prioritizing nodes. Focusing on node selection, a bi-directional AGT (BAGT) introduces a second tree originated from the goal in order to offer a better heuristic cost of the first tree. Effectiveness of BAGT pivots on the fact that the second tree encodes obstacles information near the goal. Case study demonstrates that i-AGT consistently reduces the complexity of the tree and improves computation efficiency; and BAGT works largely but not always, particularly with no benefit observed for simple cases.**

I. Introduction

Path planning arises in numerous applications such as autonomous vehicles [4] and robotics [5]. Established results include graph-based A* [6]–[8] and D* [5], [9]; navigation function and potential field [10]; sampling-based algorithms such as probabilistic roadmaps (PRM) [11], expansive-space trees [12], rapidly-exploring random trees (RRT) [13], optimal variants RRT* and PRM* [14], particle RRT [15], and anytime RRT [16], [17].

Graph-based approaches search a pre-defined graph which approximates the configuration space of a robot. The graph consists of uniformly distributed nodes and pre-defined edges. During real-time search, all or portion of the nodes and edges are tested to acquire a spare representation of collision free configuration space. A* and D* achieve resolution-completeness, with optimality guarantee under certain circumstances [18]. Since the complexity of the pre-defined graph grows exponentially along with the dimension of configuration space, these approaches become practically infeasible for high dimensional systems.

Sampling-based approaches overcome the curse of dimensionality by constructing a graph on-the-fly, where nodes are added by testing randomly drawn configurations. Their effectiveness, as pointed out in [4], [19], relies on how quick a graph can grow towards a goal configuration. Biased sampling schemes are intensively investigated to improve efficiency, e.g. visibility-based sampling [20], quasi-randomized sampling [21], heuristically-guided [22], reachability-guided [23], environment-guided [24], and waypoint-guided [25]. Commonly used sampling-based algorithms have been shown to work well in high dimensional robotic applications and possess theoretical guarantees such as probabilistic completeness [11], with exceptions [26]. A key restriction is that the resultant path could be quite sub-optimal as well as random.

Y. Wang is with Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA (email: yebinwang@ieee.org).

When human and robot share the environment, the lack of deterministic guarantee raises concerns to a certain extent. Prevailing work leverage both sampling-based and graph-search techniques, e.g. quasi-randomized RRT [21], Hybrid A* [1], Anytime D* [2], [3]. Take A-search guided tree (AGT), equivalently Hybrid A*, as an example. It selects and expands nodes by applying pre-defined control actions (motion primitives) until the tree is in proximity to a goal. Node selection is deterministically guided by a heuristic cost. AGT differs from RRT by its deterministic tree construction and optimality guarantee. Key issues are two-fold: it incurs a large memory to store pre-defined state lattice; a node is unnecessarily expanded by all primitives.

Similarly to probabilistic approaches adopting uniform sampling, AGT expands a node in all directions and leads to a tree containing unnecessary nodes. Work [27] proposes to only add child nodes that haves lower heuristic costs than the parent. Mimicking greedy search, it trades optimality for computation efficiency. This work presents an improved A-search guided tree (i-AGT) to balance optimality and computation efficiency. The i-AGT conducts selective expansion for a node: control actions are prioritized at each node and only a subset of control actions with the highest priority will be applied at one time. The node will not be selected for expansion only if all actions have been applied. Achieving resolution completeness as AGT, i-AGT produces a smaller tree and performs faster. However, its heuristic cost typically overestimates the true value, which implies sub-optimality. Similar idea has been explored in [27].

This work makes a second contribution by proposing a bi-directional A-search guided tree (BAGT). BAGT provides an alternative answer to the key question that A* and D* encounter: how the heuristic cost should be defined to guide node selection efficiently? This question is notoriously tricky because the heuristic cost consists of: a known arrival cost and an unknown cost-to-go. Depending on the environment, obstacles, and system dynamics, the latter is difficult to reckon [28]. BAGT proposes to construct two trees simultaneously: a start tree and a goal tree rooted at an initial configuration and a goal configuration, respectively. The goal tree explores obstacles close to the goal, and establishes arrival costs of its nodes, which can help to better estimate the cost-to-go for nodes on the start tree. Introducing the goal tree seemingly incurs remarkable overhead; and the complexity of both trees should be commensurate with that of the AGT. Surprisingly, BAGT works well for many scenarios, especially if there are obstacles near the goal.

BAGT is motivated by notable observations made in prior art, e.g. [1], [3], [28]: it is crucial and effective to encode obstacle information into the estimated cost-to-go. Work [1], [3] decompose the estimated cost-to-go into two parts: a dynamic heuristic related to system, and a collision heuristic induced by obstacles. This treatment suffers a fundamental limitation: cost-to-go is not a simple combination of two heuristics. How to harness two heuristics is non-obvious. Additional shortcoming is: the collision heuristic is unnecessarily constructed over entire configuration space. Oppositely, BAGT exploits arrival costs of nodes on the goal tree to estimate cost-to-go of nodes on the start tree. Since the arrival costs account for (incomplete) obstacles and system dynamics, BAGT partially circumvents the fundamental limitation. Since the goal tree is a sparse representation of configuration space, BAGT is overall efficient. A key limitation of BAGT is that the goal tree might provide misleading information. How to detect and avoid this pitfall is interesting. It is also noteworthy that BAGT resorts to bidirectional search idea which has been extensively exploited in [**?**], [29], [30].

This paper is organized as follows. Section II presents a path planning problem and AGT. Section III offers i-AGT and performance analysis. BAGT is described in Section IV. Simulation results are included in Section V to verify the proposed algorithms. Section VI completes this paper with conclusion and future work.

II. Preliminary Results

A. Path Planning Problem

Consider a robot with the following dynamics

$$\dot{X} = f(X) + g(X)u, \tag{1}$$

where $X \in \mathcal{X} \subset \mathbb{R}^{n_x}$ is state, $u \in \mathcal{U} \subset \mathbb{R}^m$ the control, f a smooth vector field or the drift, and $g = [g_1^\top, \cdots, g_m^\top]^\top$ with g_i a smooth vector field. A *configuration* of system (1) is a complete specification of the position of every points of that system. The *configuration space* $\mathcal{C} \subset \mathbb{R}^{n_c}$ is a compact set representing all possible configurations of the system. A collision-free configuration space $\mathcal{C}_{free}$ is the set of configurations at which the robot has no intersection with obstacles in the environment. Denote the collision configuration space $\mathcal{C}_{obs} = \mathcal{C} \backslash \mathcal{C}_{free}$. An *admissible trajectory* $\mathcal{X}_t$ is a solution of system (1) with given initial and final conditions and $u \in \mathcal{U}$. An *admissible path* $\mathcal{P}_t$ is the image of an admissible trajectory on the configuration space $\mathcal{C}$. For brevity, an admissible path, if additionally collision-free, is termed a *feasible path*. The state space typically has a higher dimension than configuration space. Whenever system (1) represents its kinematics, $n_x = n_c$ and $\mathcal{C} = \mathcal{X}$, which is assumed in this work.

Example 2.1: Consider a front wheel drive vehicle. Its kinematics are modeled as [31]

$$\begin{aligned}\dot{x} &= \cos(\theta)u_1 \\ \dot{y} &= \sin(\theta)u_1 \\ \dot{\theta} &= u_2 u_1/R,\end{aligned} \tag{2}$$

where (x, y) is the coordinates of the midpoint A of the rear wheels, θ the vehicle orientation, u_1 is the velocity along the car orientation, u_2 is the steering control, and R is the minimum turning radius. System state space $X = (x, y, \theta)^\top$ coincides with the configuration space, i.e., $\mathcal{C} = \mathcal{X} \subset \mathbb{R}^3$.

Problem 2.2: Given an initial configuration $X_0 \in \mathcal{C}_{free}$, a goal configuration $X_f \in \mathcal{C}_{free}$, and system (1), find a feasible path $\mathcal{P}_t$ which

(I) starts at X_0 and ends at X_f, while satisfying (1); and
(II) lies in the collision-free configuration space $\mathcal{C}_{free}$.

Let $J(\cdot)$ be a cost function that assigns to each non-trivial path a non-negative cost. Optimal path planning is to find a feasible path $\mathcal{P}_t^* : [0, 1] \to \mathcal{C}_{free}$ that minimizes $J(\cdot)$.

Notation: Tree $\mathcal{T}$ is a union of a node set $\mathcal{V} \subset \mathcal{C}_{free}$ and an edge set $\mathcal{E}$, i.e., $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. Without causing confusion, node and configuration are used interchangeably below. An edge $E(X_i, X_j) \in \mathcal{E}$ represents a feasible path between X_i and X_j. A start tree $\mathcal{T}_s$ and a goal tree $\mathcal{T}_g$ has X_0 and X_f as its root node, respectively. For a finite set $\mathcal{V}$, $|\mathcal{V}|$ denotes the number of its elements. Let $\mathcal{A}$ denote a finite set of control actions $a_k \in \mathcal{U}$, and $\mathcal{I}$ denote a finite set with its element ΔT_k being bounded real.

B. AGT Algorithm

AGT, described by Algorithms 1-2, tries to construct a start tree $\mathcal{T}_A$, which reaches a neighbor $\mathcal{B}_\epsilon(X_f)$ of X_f with $\mathcal{B}_\epsilon(X_f) \triangleq \{X | d(X, X_f) \le \epsilon, \forall X \in \mathcal{X}\}$. Specifically, $d(\cdot, \cdot)$ is a distance function, e.g. a weighted 2-norm: $\|X_i - X_j\|_P = ((X_i - X_j)^\top P (X_i - X_j))^{1/2}, \forall X_i, X_j \in \mathcal{C}$. Similar to A*, each node X is assigned a key value through a heuristic cost function $F(\cdot)$

$$F(X) = g(X_0, X) + h(X, X_f), \tag{3}$$

where $g(X_0, X)$ represents the arrival cost, or g-value, from X_0 to X, and $h(X, X_f)$ denotes the estimated cost-to-go, or h-value, from X to X_f. F-value (3) for node X is an estimated cost of a potential path from X_0 to X_f while passing through node X. AGT maintains a priority queue Q_A, which contains nodes to be expanded. All nodes in Q_A are ordered according to their F-values.

Remark 2.3: The cost-to-go from X to X_f depends on spatial locations of X and X_f, system dynamics, and obstacles. Accordingly, $h(X, X_f)$ admits forms such as weighted p-norm, the length of a Reeds-Shepp path [31], the length of continuous-curvature paths [32], [33], a combination of the Reeds-Shepp path length and a collision heuristic [1]. In both AGT and i-AGT, $h(X, X_f)$ is defined as the length of a Reeds-Shepp path in obstacle-free environment.

In the beginning of AGT, both $\mathcal{T}_A$ and Q_A have one element X_0. `InitializePrimitives` pre-computes motion primitives $\mathcal{M}$. Variable K sets the maximum number of iterations. At the kth iteration, Q_A.`Pop` retrieves node X_{best} with the lowest key value. If $X_{\text{best}} \in \mathcal{X} \backslash \mathcal{B}_\epsilon(X_f)$, node X_{best} will be expanded to grow the tree (lines 8-9); otherwise, the tree construction stops and returns success. If $\mathcal{T}_A$ fails to reach $\mathcal{B}_\epsilon(X_f)$ within K iterations, AGT returns failure.

Algorithm 1: AGT

```
1  input $X_0, K, \epsilon, \delta, \mathcal{A}, \mathcal{I}$;
2  $\mathcal{T}_A \leftarrow (X_0, \emptyset), Q_A \leftarrow X_0$;
3  $\mathcal{M} \leftarrow$ InitializePrimitives($\mathcal{A}, \mathcal{I}$);
4  $k \leftarrow 1$, $flag \leftarrow$ false;
5  while $k \leq K$ and not $flag$ do
6      $k \leftarrow k + 1$;
7      $X_{\text{best}} = Q_A$.Pop where $F(X_{\text{best}}) \leq F(X), \forall X \in Q_A$;
8      if $d(X_{\text{best}}, X_f) > \epsilon$ then
9          Expand($\mathcal{T}_A, Q_A, X_{\text{best}}$);
10     else
11         $flag \leftarrow$ true ;
12 return $(\mathcal{T}_A, flag)$;
```

Always starting from the origin $X = \mathbf{0}$, a motion primitive $MP_k \in \mathcal{M}$ is obtained by applying a control action $a_k \in \mathcal{A}$ to system (1) for a period of ΔT_k. Each control action could have a distinctive $\Delta T_k \in \mathcal{I}$. Given a system, a primitive is uniquely defined by a tuple $(a_k, \Delta T_k)$. Expand, given in Algorithm 2, conducts expansion of node X_{best} according to motion primitives in $\mathcal{M}$. Parameter δ is introduced to restrict the density of nodes. Given X_{best} and MP_k, a new configuration $X_k \in \mathcal{P}_k$ and an admissible path $\mathcal{P}_k$ from X_{best} to X_k are obtained. As long as $\mathcal{P}_k$ is collision-free and X_k is δ-distant away from tree $\mathcal{T}_A$, i.e.,

$$\min_{X \in \mathcal{V}_A} d(X_k, X) \geq \delta,$$

node X_k and edge $E(X_{\text{best}}, X_k)$ are added to $\mathcal{T}_A$. Node X_k is pushed into Q_A for future expansion.

Algorithm 2: Expand in AGT

```
1  input $\mathcal{T}_A, Q_A, X_{\text{best}}$;
2  $(\mathcal{V}_A, \mathcal{E}_A) \leftarrow \mathcal{T}_A$;
3  $k \leftarrow 1$;
4  while $k \leq |\mathcal{M}|$ do
5      $(X_k, \mathcal{P}_k) =$ Simulate($X_{\text{best}}, MP_k$);
6      if $\min_{X \in \mathcal{V}_A} d(X_k, X) \geq \delta$ and CollisionFree($\mathcal{P}_k$)
       then
7          $\mathcal{V}_A \leftarrow \mathcal{V}_A \bigcup \{X_k\}, \mathcal{E}_A \leftarrow \mathcal{E}_A \bigcup E(X_{\text{best}}, X_k)$;
8          $Q_A$.Push($X_k$);
9  $\mathcal{T}_A \leftarrow (\mathcal{V}_A, \mathcal{E}_A)$;
```

AGT is similar to Hybrid A [1] and Anytime D [3]. All three expand a tree according to motion primitives; and node selection is guided by the heuristic cost (3). Nodes of the tree might form a non-uniform distribution over $\mathcal{C}_{free}$. That is to say, the sparsity of the tree is not guaranteed. Work [1], [3] utilize pre-defined state lattices to ensure the uniform distribution of nodes. This treatment entails a large memory to store state lattices. Slightly different, AGT enforces the uniform density by checking whether any new configuration X_{new} is δ-distant from the tree. Because the check is done online, AGT is subject to loss of computation efficiency, meanwhile requiring less memory.

Key parameters of AGT are $\epsilon, \delta, \mathcal{I}$, and $\mathcal{A}$. All need to be tuned so that AGT behaves decently. The search for a path essentially boils down to find a sequence of primitives which steers the robot into $\mathcal{B}_\epsilon(X_f)$. Intuitively, the smaller ϵ is, the more challenging and time-consuming the path search will be. Parameter δ affects the feasibility and computation efficiency. Roughly speaking, δ defines the resolution of $\mathcal{C}$, and is related to resolution-completeness. Feasibility-wise, the smaller δ is, the better. However, a smaller δ typically leads to a larger tree. Practically, its amplitude is lower-bounded by accumulated errors result from localization and path following control systems. Similarly, $\mathcal{I}$ and $\mathcal{A}$ are related to resolution-completeness as well as computation efficiency. Given $\delta, \mathcal{I}$ and $\mathcal{A}$, one can determine primitives $(a_k, \Delta T_k)$ by ensuring that all resultant configurations maintain a δ-distance from each other.

III. Improved A-Search Guided Tree

In AGT, each node gets at most one chance to expand. This poses a noticeable limitation: all motion primitives have been applied during node expansion. Recalling why AGT outperforms breadth first search by sophisticated node selection, the whole idea of applying all primitives during node expansion is apparently not necessary and inefficient. i-AGT is proposed to weaken the limitation, where a node can be expanded multiple times, and its expansion is biased by prioritizing motion primitives. i-AGT is strongly incentivized by scenarios such as city driving or parking, where vehicle paths can be classified and each class corresponds to a limited number of motion primitives.

A. i-AGT Algorithm

Basic idea of i-AGT pivots on a concept 'mode' which is associated to a subset of motion primitives $M_i \subset \mathcal{M}$. Assume that the set $\mathcal{M}$ is partitioned into m subsets, i.e.,

$$\mathcal{M} = \bigcup_{1 \leq k \leq m} M_k$$
$$M_i \cap M_j = \emptyset, \ \forall 1 \leq i \neq j \leq m.$$

Given a node X, mode M_i has a priority $p_X^{M_i}$ for $1 \leq i \leq m$; if primitives in M_i have not been applied in the expansion of X, we say the corresponding mode is untried at X.

Remark 3.1: As a special case, the set $\mathcal{M}$ can be split into $|\mathcal{M}|$ subsets, where each subset corresponds to one primitive. To simplify the presentation, pseudo-code and discussions below presume that $|M_i| = 1$ for $1 \leq i \leq m$. The special case reduces i-AGT to greedy search, which is good in practice. Number of modes and associated primitives deserve a careful design to balance exploration and exploitation.

Algorithm 3 details Expand of i-AGT. During the expansion of X_{best}, a current mode, denoted by M_c, is first determined by GetCurrentMode. Particularly, GetCurrentMode enumerates untried modes and returns the mode which has the highest priority. Then, X_{best} is expanded by applying primitive M_c, which gives X_k and $\mathcal{P}_k$. If X_k is δ-distant away from $\mathcal{T}_A$, and $\mathcal{P}_k$ is collision free, then

(I) UpdatePriority updates priority $P_{X_{\text{best}}}^{M_c}$ according to $F(X_{\text{best}}) - F(X_k)$;

(II) InheritPriority initializes the priority of all modes of X_k as follows

$$p^{M_i}_{X_k} = p^{M_i}_{X_{\text{best}}}, \quad 1 \le i \le m.$$

(III) node X_k and edge $E(X_{\text{best}}, X_k)$ are added to $\mathcal{T}_A$;

(IV) node X_k is inserted into Q_A.

```
Algorithm 3: Expand in i-AGT
1  input $\mathcal{T}_A, Q_A, X_{\text{best}}$;
2  $(\mathcal{V}_A, \mathcal{E}_A) \leftarrow \mathcal{T}_A$;
3  $M_c \leftarrow$ GetCurrentMode($X_{\text{best}}$);
4  $(X_k, \mathcal{P}_k) =$ Simulate($X_{\text{best}}, M_c$);
5  if $\min_{X \in \mathcal{V}_A} d(X_k, X) \ge \delta$ and CollisionFree($\mathcal{P}_k$) then
6      UpdatePriority($X_{\text{best}}$);
7      InheritPriority($X_k, X_{\text{best}}$);
8      $\mathcal{V}_A \leftarrow \mathcal{V}_A \bigcup \{X_k\}, \mathcal{E}_A \leftarrow \mathcal{E}_A \bigcup E(X_{\text{best}}, X_k)$;
9      $Q_A$.Push($X_k$);
10 $\mathcal{T}_A \leftarrow (\mathcal{V}_A, \mathcal{E}_A)$;
```

i-AGT contains two key steps: UpdatePriority and InheritPriority. The former can update $p^{M_c}_{X_{\text{best}}}$ in a probabilistic or deterministic manner. With a focus on predictable path planning, deterministic rules are adopted here. As an example, UpdatePriority implements the following rules.

(I) If $F(X_k) < F(X_{\text{best}})$, the priority $p^{M_c}_{X_{\text{best}}}$ is set to 1;

(II) If $F(X_k) \ge F(X_{\text{best}})$, then $p^{M_c}_{X_{\text{best}}}$ is reduced. Specifically, $p^{M_c}_{X_{\text{best}}}$ is updated according to

$$p^{M_i}_{X_{\text{best}}} = p^{M_i}_{X_{\text{best}}} - \alpha(F(X_k) - F(X_{\text{best}})),$$

where $\alpha > 0$. In an extreme case, $p^{M_i}_{X_{\text{best}}} = 0$.

Remark 3.2: The aforementioned rules render priority $p^{M_i}_{X_{\text{best}}}$ a dynamic process. We ought to ensure $p^{M_i}_{X_{\text{best}}}$ is bounded. This is true if the number of iterations is bounded. As an alternative, one can always saturate priority variables to ensure boundedness.

It is worth mentioning that all modes for X_0 have the same pre-defined priority p_0, e.g. $p_0 = 1$. This means all primitives in $\mathcal{M}$ will be applied to expand X_0.

Properties of UpdatePriority certainly impacts computation efficiency and completeness of i-AGT. The efficacy of i-AGT is contingent on: whether the right node and mode can be determined. We roughly analyze how fast i-AGT can be versus AGT, by considering an idea case: UpdatePriority perfectly captures the priority of modes. Suppose that

(I) AGT $\mathcal{T}_{AGT}$ includes a path $\mathcal{P}^*_t$ which contains nodes $\{X^*_1, \ldots, X^*_N\}$ with $X^*_N \in \mathcal{B}_\epsilon(X_f)$;

(II) $h(X_k, X_f)$ is an exact cost-to-go. This implies that both AGT and i-AGT will select nodes $\{X^*_1, \ldots, X^*_N\}$ sequentially without expanding any other nodes;

(III) each subset M_i contains the same number of primitives: $|M_i| = |\mathcal{M}|/m$.

For the ideal case, i-AGT yields a tree containing as many nodes as one-mth of $\mathcal{T}_{AGT}$. Argument follows. Each node in $\mathcal{T}_{AGT}$ is expanded by applying $\mathcal{M}$ and thus has $|\mathcal{M}|$ children. $\mathcal{T}_{AGT}$ contains $|M| \times N$ nodes. For i-AGT, provided that UpdatePriority exactly knows the priority of modes, GetCurrentMode returns the correct mode of node X^*_k. One shows, by induction, that for $1 \le k \le N$, i-AGT sequentially

(I) selects node X^*_k as AGT does;

(II) expands X^*_k according to the best mode, and thus each node contains $|\mathcal{M}|/m$ children including X^*_{k+1};

(III) yields a tree $\mathcal{T}_{iAGT}$ contains $|\mathcal{M}| \times N/m$ nodes.

B. Completeness

As shown below, completeness of i-AGT requires that all modes be visited if necessary. This property is related to GetCurrentMode and boundedness of priority. Recall GetCurrentMode locates the mode with the highest priority among untried modes. Any mode can be visited as long as priority is finite, no matter how low its priority is. Without loss of generality, this property is assumed in analysis.

Proposition 3.3: The i-AGT is resolution-complete if and only if AGT is resolution-complete.

Proof: Proof is omitted due to space limitation. ■

Remark 3.4: Let i-AGT uses (3) where $h(X_k, X_f)$ offers a lower bound estimate of a cost-to-go toward X_f from X_k. Then i-AGT is reduced to AGT, because

$$\begin{aligned} F(X_{\text{best}}) &= g(X_0, X_{\text{best}}) + h(X_{\text{best}}, X_f) \\ &\le g(X_0, X_{\text{best}}) + c(X_{\text{best}}, X_k) + h(X_k, X_f) \\ &= F(X_k), \end{aligned}$$

and Expand will apply all primitives in $\mathcal{M}$ to X_{best}. In other words, i-AGT necessitates the use of an inflated heuristic cost to exhibit computational benefits: $\rho h(X_k, X_f)$ with $\rho > 1$. This also implies i-AGT is sub-optimal. By using $\rho h(X_k, X_f)$, anytime A and D prioritize nodes with a lower heuristic cost. Analogously, i-AGT runs in anytime fashion by prioritizing primitives having higher priority.

IV. Bi-Directional A-Search Guided Tree

This section investigates means to better estimate the cost-to-go, which is challenging but critical to computation efficiency. We propose BAGT which concurrently constructs a goal tree $\mathcal{T}_g$ in addition to a start tree $\mathcal{T}_s$. Exploring environment near the goal, $\mathcal{T}_g$ produces arrival costs which allow a better estimate of the cost-to-go for nodes in $\mathcal{T}_s$.

A. BAGT Algorithm

Algorithm 4 describes the main flow of BAGT. It constructs a start tree $\mathcal{T}_s$ arriving at $\mathcal{B}_\epsilon(X_f)$ (lines 9-10). A distinguishable feature is that a goal tree $\mathcal{T}_g$ is constructed along with $\mathcal{T}_s$ (lines 14-15). Unlike D*, where $\mathcal{T}_g$ manages to touch $\mathcal{B}_\epsilon(X_0)$, here $\mathcal{T}_g$ helps the construction of $\mathcal{T}_s$ via finding out obstacles close to X_f and better estimating the cost-to-go of nodes on $\mathcal{T}_s$. Therefore, the construction of $\mathcal{T}_g$ will stop (line 13) if it is close to $\mathcal{T}_s$. Variable $flag_g$ is true if $\mathcal{T}_g$ and $\mathcal{T}_s$ are close.

Key feature of BAGT is the way to compute the heuristic cost (3), specifically the estimated cost-to-go. As shown in Algorithm 5, Expand works on node $X_{\text{best}} \in \mathcal{T}_s$, which means $\mathcal{T}_A = \mathcal{T}_s, \mathcal{T}_B = \mathcal{T}_g$. The expansion leads to an admissible node X_k, which is checked against all nodes on $\mathcal{T}_B$. If it is within μ-distance from one node in $\mathcal{T}_B$ (line

Algorithm 4: BAGT

```
1  input $X_0, K, \epsilon, \delta, \mathcal{A}, \mathcal{I}, \mu, \gamma$;
2  $\mathcal{T}_s \leftarrow (X_0, \emptyset), Q_s \leftarrow X_0$;
3  $\mathcal{T}_g \leftarrow (X_f, \emptyset), Q_g \leftarrow X_f$;
4  $\mathcal{M} \leftarrow$ InitializePrimitives$(\mathcal{A}, \mathcal{I})$;
5  $k \leftarrow 1$, $flag \leftarrow$ false, $flag_g \leftarrow$ false ;
6  while $k \leq K$ and not $flag$ do
7      $k \leftarrow k + 1$;
8      $X_{\text{bests}} = Q_s$.Pop where $F(X_{\text{bests}}) \leq F(X), \forall X \in Q_s$;
9      if $d(X_{\text{bests}}, X_f) > \epsilon$ then
10         $flag_g \leftarrow$ Expand$(\mathcal{T}_s, Q_s, X_{\text{bests}}, \mathcal{T}_g)$;
11     else
12         $flag \leftarrow$ true ;
13     if not $(flag_g$ or $flag)$ then
14         $X_{\text{bestg}} = Q_g$.Pop where
           $F(X_{\text{bestg}}) \leq F(X), \forall X \in Q_g$;
15         $flag_g \leftarrow$ Expand$(\mathcal{T}_g, Q_g, X_{\text{bestg}}, \mathcal{T}_s)$;
16 return $(\mathcal{T}_s, flag)$;
```

7), $flag_g$ is set true. Procedure `UpdateCost` calculates the heuristic cost $F(X_k)$ as follows. It first finds all nodes on $\mathcal{T}_B$ which are close to X_k, i.e.,

$$\mathcal{X}_{\text{near}} \triangleq \{X | d(X, X_k) \leq \gamma, \forall X \in \mathcal{V}_B\},$$

where γ is a tuning parameter. If $\mathcal{X}_{\text{near}}$ is not empty, then

$$h(X_k, X_f) = \min_{X_i \in \mathcal{X}_{\text{near}}} \{h(X_k, X_i) + g(X_f, X_i)\},$$

where $g(X_f, X_i)$ is the arrival cost from X_f to X_i, inferred from the goal tree. If $\mathcal{X}_{\text{near}}$ is empty, then

$$h(X_k, X_f) = h(X_k, X_{\text{nearest}}) + g(X_f, X_{\text{nearest}})\},$$

where $X_{\text{nearest}} = \arg\min_{X \in \mathcal{V}_B} d(X_k, X)$.

Algorithm 5: Expand in BAGT

```
1  input $\mathcal{T}_A, Q_A, X_{\text{best}}, \mathcal{M}, \mathcal{T}_B$;
2  $(\mathcal{V}_A, \mathcal{E}_A) \leftarrow \mathcal{T}_A$, $(\mathcal{V}_B, \mathcal{E}_B) \leftarrow \mathcal{T}_B$;
3  $k \leftarrow 1, flag_g \leftarrow$ false;
4  while $k \leq |\mathcal{M}|$ do
5      $(X_k, \mathcal{P}_k) =$ Simulate$(X_{\text{best}}, MP_k)$;
6      if $\min_{X \in \mathcal{V}_A} d(X_k, X) \geq \delta$ and CollisionFree$(\mathcal{P}_k)$
       then
7          if $\min_{X \in \mathcal{V}_B} d(X_k, X) \leq \mu$ then
8              $flag_g \leftarrow$ true;
9          $X_k$.UpdateCost;
10         $\mathcal{V}_A \leftarrow \mathcal{V}_A \bigcup \{X_k\}, \mathcal{E}_A \leftarrow \mathcal{E}_A \bigcup E(X_{\text{best}}, X_k)$;
11         $Q_A$.Push$(X_k)$;
12 $\mathcal{T}_A \leftarrow (\mathcal{V}_A, \mathcal{E}_A)$;
13 return $flag_g$;
```

`UpdateCost` is depicted in Fig. 1, which includes a start tree in olive, a goal tree in black, a circular obstacle in gray, and many dots representing nodes. Suppose that expansion of $X_{\text{best}} \in \mathcal{T}_s$ gives two admissible child nodes: X_{k1}, X_{k2}. X_{k1}.`UpdateCost` draws around X_{k1} a green circle of radius γ and identifies that $\mathcal{X}_{\text{near}}$ contains one element $X_g \in \mathcal{T}_g$. The heuristic cost of X_{k1} is:

$$F_{BAGT}(X_{k1}) = g(X_0, X_{k1}) + h(X_{k1}, X_g) + g(X_f, X_g).$$

After identifying X_g based on the magenta circle, X_{k2}.`UpdateCost` computes the heuristic cost as follows

$$F_{BAGT}(X_{k2}) = g(X_0, X_{k2}) + h(X_{k2}, X_g) + g(X_f, X_g).$$

Oppositely, AGT uses the heuristic costs

$$F_{AGT}(X_{k1}) = g(X_0, X_{k1}) + h(X_{k1}, X_f)$$
$$F_{AGT}(X_{k2}) = g(X_0, X_{k2}) + h(X_{k2}, X_f),$$

where $h(X_{k1}, X_f), h(X_{k2}, X_f)$ do not account for the obstacle. If $g(X_f, X_g)$ contains information about the obstacle, $F_{BAGT}(X_{k2})$ and $F_{BAGT}(X_{k1})$ intuitively offer better estimates than $F_{AGT}(X_{k2})$ and $F_{AGT}(X_{k1})$. Consistently, tree $\mathcal{T}_s$ will grow toward $\mathcal{T}_g$; and AGT likely grows $\mathcal{T}_s$ by adding nodes in blue (toward the obstacle). Noticing that $F_{BAGT}(X) > F_{AGT}(X)$ if $h(X_k, X_f)$ underestimates the cost-to-go, and we know BAGT is sub-optimal.

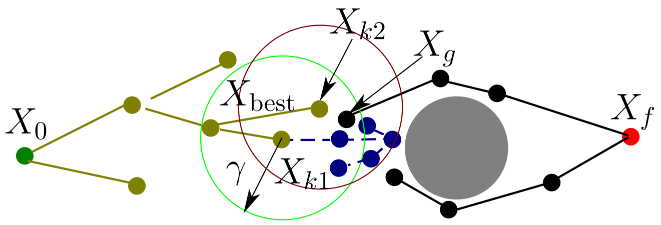


Fig. 1. Schematics to estimate cost-to-go

V. Case Studies

Three algorithms, AGT, i-AGT and BAGT, are coded to solve Problem 2.2 with system kinematics (2). Simulation is conducted in Matlab®2016b.

TABLE I
Computation Time

Alg.	Case 1	Case 4	Case 5	Case 6	Case 12	Case 14
AGT	0.69	5.11	0.17	3.74	1.12	1.39
i-AGT	0.46	1.82	0.04	1.89	0.65	0.60
BAGT	0.62	0.77	0.18	0.61	0.88	2.45

As shown in Figs. 2-4, all test cases in simulation are motivated by moving a vehicle inside parking lots. Particularly, the vehicle starts at the configuration represented by the black box, and moves into the configuration denoted by the red box. All parking lots and obstacles are abstracted as rectangles. The vehicle has a length of 4.85m, width 1.81m, and minimum turning radius 4.13m.

All algorithms use the same parameter values: $\Delta T = 0.175\text{s}, \epsilon = 2, \delta = 0.04, \rho = 1.25$. BAGT has parameters: $\mu = \gamma = 5$. Take $\mathcal{A} = V \times S$ with $V = \{\pm 1\}$ and $S = \{\pm 1, \pm 0.5, 0\}$, where S and V is the action set of steering angle and longitudinal velocity, respectively. The set $\mathcal{A}$ and ΔT induces 10 motion primitives. For i-AGT, these motion primitives are split into two modes:

(I) forward mode M_1 (ΔT is omitted for simplicity):

$$\{(1, 1), (1, 0.5), (1, 0), (1, -0.5), (1, -1)\};$$

(II) backward mode M_2 (ΔT is omitted for simplicity):

$$\{(-1,1),(-1,0.5),(-1,0),(-1,-0.5),(-1,-1)\}.$$

At X_0, take $p_{X_0}^{M_1} = p_{X_0}^{M_2} = 1$. Since only two modes are involved, $p_X^{M_k}$ for $k = \{1,2\}$ take binary values $\{0,1\}$.

Algorithms are evaluated in terms of computation efficiency and path quality. The former is measured by computation time and complexity of trees, whereas the latter is quantified by path length. Numerous cases have been tested, which lead to similar conclusions. Results are summarized in Tables I-III and Figs. 2-4, while many details are left out, due to space limitation. Computation time and the node number of trees are recorded in Table I-II, indicating that i-AGT is significantly faster for all test cases, whereas BAGT is mostly effective but not always. Table III shows that both i-AGT and BAGT lead to slightly degraded paths. Figs. 2-4 plot trees produced by three algorithms. We draw several interesting observations

(I) i-AGT and AGT produce trees with similar (spatial) shape; the difference is that the former exhibits a sparser distribution. This is consistent with the fact that i-AGT applies less primitives during node expansion;

(II) BAGT produces a tree which is remarkably different from the other two, owing to distinctive mechanisms underlying algorithms. This implies promising synergy of integrating BAGT and i-AGT;

(III) BAGT performs worse than AGT for case 14, because initially the goal tree grows in the wrong direction (right turn). Consequently, its exploration produces erroneous information which confuses the start tree;

(IV) the benefits of BAGT are not compelling for simply cases (cases 1 and 5). This is understood due to the fact that the heuristic cost used in AGT offers good enough estimate, given simple environments.

TABLE II

NODE NUMBER IN THE TREE

Alg.	Case 1	Case 4	Case 5	Case 6	Case 12	Case 14
AGT	969	6339	505	5437	1476	2450
i-AGT1	537	2465	68	2570	592	950
BAGT	612	1325	379	988	1094	3191

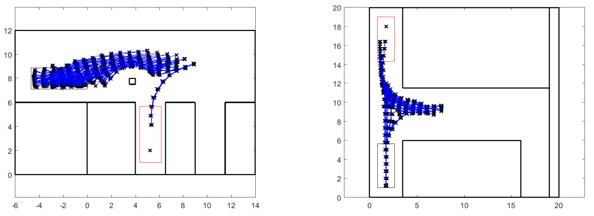
TABLE III

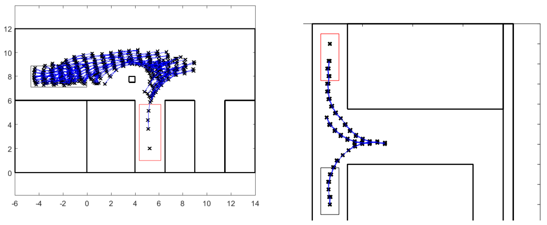
PATH LENGTH

Alg.	Case 1	Case 4	Case 5	Case 6	Case 12	Case 14
AGT	25	41	29	44	38	57
i-AGT	26	41	27	45	38	57
BAGT	26	43	31	44	39	57

VI. Conclusion and Future Work

This work proposed i-AGT and BAGT to fulfill fast kinodynamic planning. With prioritized motion primitives, i-AGT improves computation efficiency over AGT by microscopically biasing node expansion. Concentrating on node selection, BAGT provides an option to improve the estimated cost-to-go. Both algorithms exhibit preferably deterministic performance. Numerical simulation demonstrates their effectiveness. Future work includes: understand why BAGT fails, fuse i-AGT and BAGT for efficiency, improve path quality, and construct motion primitives to reduce complexity.


(a) Case 1: AGT (b) Case 5: AGT


(c) Case 1: i-AGT (d) Case 5: i-AGT

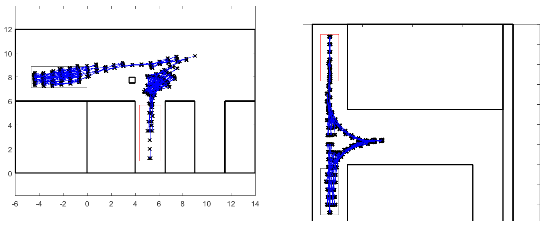

(e) Case 1: BAGT (f) Case 5: BAGT

Fig. 2. Tree complexity: cases 1 & 5

References

[1] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, B. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[3] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Robot Res.*, vol. 28, no. 8, pp. 933–945, 2009.

[4] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *Int. J. Robot Res.*, vol. 29, no. 5, pp. 485–501, 2010.

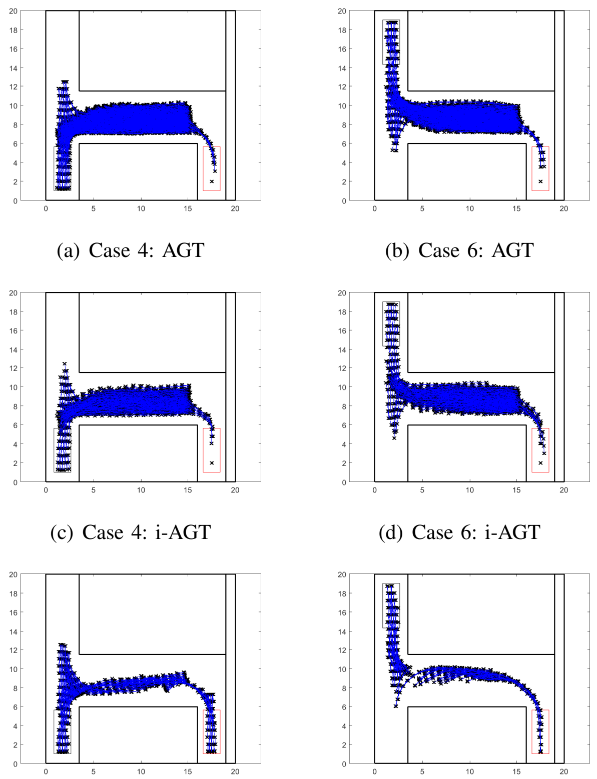
[5] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-93-20, 1993.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[7] M. Likhachev, G. Gordon, and S. Thrun, *ARA*: Anytime A* with probable bounds on sub-optimality*. MIT Press, 2003, ch. Advances in Neural Information Processing Systems.

[8] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

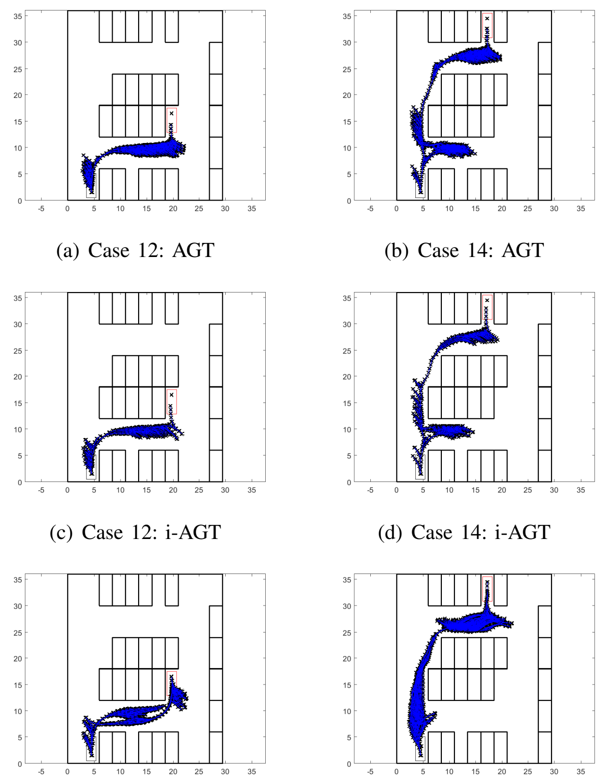
[9] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An anytime, replanning algorithm," in *Proc. 2005 ICAPS*, 2005, pp. 262–271.


(a) Case 4: AGT (b) Case 6: AGT

(c) Case 4: i-AGT (d) Case 6: i-AGT

(e) Case 4: BAGT (f) Case 6: BAGT

Fig. 3. Tree complexity: cases 4 & 6


(a) Case 12: AGT (b) Case 14: AGT

(c) Case 12: i-AGT (d) Case 14: i-AGT

(e) Case 12: BAGT (f) Case 14: BAGT

Fig. 4. Tree complexity: cases 12 & 14

[10] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *Int. J. Robot Res.*, vol. 5, no. 1, pp. 417–431, 1986.

[11] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[12] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *Int. J. Robot Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[13] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” *Int. J. Robot Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[14] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. J. Robot Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[15] N. A. Melchior and R. Simmons, “Particle RRT for path planning with uncertainty,” in *Proc. 2007 ICRA*, 2007, pp. 1617–1624.

[16] D. Ferguson and A. Stentz, “Anytime RRTs,” in *Proc. 2006 IROS*, 2006, pp. 5369–5375.

[17] J. van den Berg, D. Ferguson, and J. Kuffner, “Anytime path planning and replanning in dynamic environments,” in *Proc. 2006 ICRA*, 2006, pp. 2366–2371.

[18] B. R. Donald and P. Xavier, “Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds,” *Algorithmica*, vol. 14, pp. 443–479, 1995.

[19] H. M. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA: MIT Press, 2005.

[20] T. Siméon, J.-P. Laumond, and C. Nissoux, “Visibility-based probabilistic roadmaps for motion planning,” *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[21] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang, “Quasi-randomized path planning,” in *Proc. 2001 ICRA*, vol. 2, 2001, pp. 1481–1487.

[22] C. Urmson and R. Simmons, “Approaches for heuristically biasing RRT growth,” in *Proc. 2003 ICRA*, 2003, pp. 1178–1183.

[23] A. Shkolnik, M. Walter, and R. Tedrake, “Reachability-guided sampling for planning under differential constraints,” in *Proc. 2009 ICRA*, 2009, pp. 2859–2865.

[24] L. Jaillet, J. Hoffman, J. van den Berg, P. Abbeel, J. M. Porta, and K. Goldberg, “EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles,” in *Proc. 2011 IROS*, 2011, pp. 2646–2652.

[25] Y. Wang, D. K. Jha, and Y. Akemi, “A two-stage RRT path planner for automated parking,” in *Proc. of IEEE Conf. on Automation Science and Engineering*, 2017, pp. 496–502.

[26] T. Kunz and M. Stilman, “Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete,” in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 233–244.

[27] G. S. R. S. T. B. N. R. S. J. S. A. Felner, M. Goldenberg and R. Holte, “Partial-expansion A* with selective node expansion,” in *Proc. 26th AAAI Conf. on Artificial Intelligence*, 2012, pp. 471–477.

[28] E. Glassman and R. Tedrake, “A quadratic regulator-based heuristic for rapidly exploring state space,” in *Proc. 2010 ICRA*, 2010, pp. 5021–5028.

[29] F. Lslam, V. Narayanan, and M. Likhachev, “A*-connect: bounded suboptimal bidirectional heuristic search,” in *Proc. 2016 ICRA*, 2012, pp. 471–277.

[30] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, “Multi-heuristic A,” *Int. J. Robot Res.*, vol. 35, no. 1-3, pp. 224–243, 2016.

[31] J. A. Reeds and L. A. Shepp, “Optimal paths for a car that goes both forwards and backwards,” *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[32] T. Fraichard and A. Scheuer, “From Reeds and Shepp’s to continuous-curvature paths,” *IEEE Trans. Robot.*, vol. 20, no. 6, pp. 1025–1035, Dec. 2004.

[33] J. Dai and Y. Wang, “On existence conditions of a class of continuous curvature paths,” in *Proc. 36th Chinese Contr. Conf.*, 2017, pp. 6773–6780.