Incremental Planning

Sanjiban Choudhury

TAs: Matthew Rockett, Gilwoo Lee, Matt Schmittle
General framework for motion planning

Create a graph
General framework for motion planning

Create a graph

Search the graph
General framework for motion planning

Create a graph

Search the graph

Interleave
General framework for motion planning

Any planning algorithm

Create graph    Search graph    Interleave
General framework for motion planning

Any planning algorithm

Create graph    Search graph    Interleave
General framework for motion planning

Any planning algorithm

Create graph  Search graph  Interleave

RRT*-XYZ
General framework for motion planning

Any planning algorithm

Create graph

Search graph

Interleave

RRT*-XYZ

e.g. fancy random sampler

×
e.g. fancy heuristic

×
e.g. fancy way of densifying
General framework for motion planning

Any planning algorithm

Create graph

Search graph

Interleave

RRT*-XYZ

e.g. fancy random sampler

\times

e.g. fancy heuristic

\times

e.g. fancy way of densifying

What's the best we can do?

What's the best we can do?

What's the best we can do?
Today’s discussion

1. Why would we want to interleave?

2. How do we search when we interleave?
   (repairing search)

3. How do we improve graphs when we interleave?
   (incremental sampling)

4. Putting it all together
Today’s discussion

1. Why would we want to interleave?

2. How do we search when we interleave?

   (repairing search)

3. How do we improve graphs when we interleave?

   (incremental sampling)

4. Putting it all together
Anytime Planning
Anytime Planning

Quickly get a feasible path. Improve if you have more time.
Anytime Planning

Quickly get a feasible path. Improve if you have more time.
Planning as Inference
A Bayesian Approach to Edge Evaluation

First Set of Provably Near Bayes-Optimal Planning Algorithms

[NIPS’17, ISRR’17, IJCAI’18]
A Bayesian Approach to Edge Evaluation

First Set of Provably Near Bayes-Optimal Planning Algorithms

[NIPS’17, ISRR’17, IJCAI’18]
Today’s discussion

1. Why would we want to interleave?

2. How do we search when we interleave?
   (repairing search)

3. How do we improve graphs when we interleave?
   (incremental sampling)

4. Putting it all together
Interleaving implies new vertices / edges appear

Do we always have to replan whenever the graph changes?
What’s true about $g(s)$ values after search?

the cost $c(s_1, s_{\text{goal}})$ of an edge from $s_1$ to $s_{\text{goal}}$
Vertices are locally consistent

\[ g(s) = \min_{s' \in \text{pred}(s)} (g(s') + c(s', s)) \]
What happens when we introduce a new edge?

The cost $c(s_1, s_{goal})$ of an edge from $s_1$ to $s_{goal}$
Why Reinforcement Learning played a big role in developing planning ... (obv. the reverse is true)
Value iteration on graphs

\[ V(s) \]
Value iteration step

\[ g^+(s) = \min_{s' \in \text{pred}(s)} \left( g(s') + c(s', s) \right) \]

Do this for all states!
Value iteration on graphs

\[
g^+(s) = \min_{s' \in \text{pred}(s)} (g(s') + c(s', s))
\]

\[
V(s) \quad \Rightarrow \quad V^+(s)
\]
Does this converge?

\[ V^1(s) \quad \rightarrow \quad V^2(s) \quad \rightarrow \quad \cdots \quad \rightarrow \quad V^n(s) \]
Does this converge?

\[ V^1(s) \rightarrow V^2(s) \rightarrow \ldots \rightarrow V^n(s) \]

Yes!

Value iteration is a contraction
Does this converge?

\[ g = 0 \]

\[ g = \infty \]

\[ \text{S}_{\text{start}} \quad 1 \quad \text{S}_2 \quad 2 \quad \text{S}_1 \quad 2 \quad \text{S}_{\text{goal}} \quad g = \infty \]

\[ \text{S}_4 \quad 3 \quad \text{S}_3 \quad 1 \quad \text{S}_{\text{goal}} \quad g = \infty \]

\[ g = \infty \]

\[ g = \infty \]
Does this converge?

\[ g = 0 \]

\[ g = -3 \quad \rightarrow \quad g = 1 \]

\[ S_{\text{start}} \quad 1 \quad S_2 \quad 2 \quad S_1 \quad 2 \quad S_{\text{goal}} \]

\[ g = 5 \quad \rightarrow \quad g = 13 \]

\[ g = 15 \]
Asynchronous value iteration

What if we didn’t update for ALL the states?

\[
g^+(s) = \min_{s' \in \text{pred}(s)} (g(s') + c(s', s))
\]

\[
\begin{bmatrix}
g_1 \\
g_2 \\
g_3 \\
g_4 \\
g_5 \\
\end{bmatrix}
\]

\[
V(s)
\]

\[
\begin{bmatrix}
g_1^+ \\
g_2^+ \\
g_3^+ \\
g_4^+ \\
g_5^+ \\
\end{bmatrix}
\]

\[
V^+(s)
\]
Asynchronous value iteration

What if we didn’t update for ALL the states?

\[ g^+(s) = \min_{s' \in \text{pred}(s)} (g(s') + c(s', s)) \]

What if we did this for a RANDOM SUBSET of states?
Asynchronous value iteration

What if we didn’t update for ALL the states?

\[ g^+(s) = \min_{s' \in \text{pred}(s)} (g(s') + c(s', s)) \]

What if we did this for a RANDOM SUBSET of states?

Does this converge?
Asynchronous value iteration

What if we didn’t update for ALL the states?

\[ g^+(s) = \min_{s' \in \text{pred}(s)} (g(s') + c(s', s)) \]

What if we did this for a RANDOM SUBSET of states?

Does this converge?

YES
Does this converge?

\[ g = 0 \]

\[ g = -3 \rightarrow S_2 \]

\[ g = 1 \rightarrow S_1 \]

\[ g = 15 \rightarrow S_{goal} \]

\[ g = 5 \rightarrow S_4 \]

\[ g = 13 \rightarrow S_3 \]
Back to our problem …

What happens if you run asynchronous value iteration?
Key Idea

Run asynchronous value iteration in an organized way

LPA* (Koenig and Likhachev)
How general is this idea?
How general is this idea?

How many ways can a graph change?
How general is this idea?

How many ways can a graph change?

New edges / vertices appear
How general is this idea?

How many ways can a graph change?

New edges / vertices appear

Cost of edges increase (lazy evaluation)
How general is this idea?

How many ways can a graph change?

New edges / vertices appear

Cost of edges increase (lazy evaluation)

Cost of edges increase/decrease (approximation tech)
How general is this idea?

How many ways can a graph change?

- New edges / vertices appear

- Cost of edges increase (lazy evaluation)

- Cost of edges increase/decrease (approximation tech)

- F-value of nodes change (dynamic heuristic)
How general is this idea?

How many ways can a graph change?

New edges / vertices appear

Cost of edges increase (lazy evaluation)

Cost of edges increase/decrease (approximation tech)

F-value of nodes change (dynamic heuristic)

What about planning across iterations?
How general is this idea?

How many ways can a graph change?

New edges / vertices appear

Cost of edges increase (lazy evaluation)

Cost of edges increase/decrease (approximation tech)

F-value of nodes change (dynamic heuristic)

What about planning across iterations?

New obstacles appear/disappear - cost of edges increase/decrease
Anytime Repairing A* (ARA*)

https://www.youtube.com/watch?v=rZHtHJlJa2w
Anytime Repairing A* (ARA*)

https://www.youtube.com/watch?v=rZHTHJlJa2w
D*-Lite
D*-Lite
Today’s discussion

1. Why would we want to interleave?

2. How do we search when we interleave?
   (repairing search)

3. How do we improve graphs when we interleave?
   (incremental sampling)

4. Putting it all together
What is incremental sampling?
Probabilistic Roadmaps were batch
Rapidly Exploring Dense Tree (RDT)

LaValle, 1998

SIMPLE_RDT($q_0$)
1 $G$.init($q_0$);
2 for $i = 1$ to $k$ do
3 $G$.add_vertex($\alpha(i)$);
4 $q_n \leftarrow$ NEAREST($S(G), \alpha(i)$);
5 $G$.add_edge($q_n, \alpha(i)$);

(a) (b)

Figure 5.16: The basic algorithm for constructing RDTs (which includes RRTs as a special case) when there are no obstacles. It requires the assumption that the problem is free of obstacles or boundary obstructions. It is assumed that the goal is to get as close as possible to every configuration, as long as they are dense with probability one. Random sequences that induce a nonuniform or avert bias are also available (as a special case) when there are no obstacles. It requires the assumption that the sample points reached by the sample using the algorithm in Figure 5.16. (b) A new edge is added that connects $q_n$ to $\alpha(i)$.

Figure 5.17: (a) Suppose inductively that this tree has been constructed so far. Let $\alpha(i)$ be a configuration such that $S(G \cup \alpha(i))$ is a metric space. Initially, $S(G \cup \alpha(i))$ is the image of the path between $q_0$ and $\alpha(i)$. This may possibly include a uniform, random sequence, which is the image of the path between $q_0$ and $\alpha(i)$.
Rapidly Exploring Dense Tree (RDT)

LaValle, 1998

**Algorithm 5.16**

SIMPLE_RDT($q_0$)

1. $\mathcal{G}$.init($q_0$);
2. for $i = 1$ to $k$ do
3. \hspace{1em} $\mathcal{G}$.add_vertex($\alpha(i)$);
4. \hspace{1em} $q_n \leftarrow$ NEAREST($S(\mathcal{G}), \alpha(i)$);
5. \hspace{1em} $\mathcal{G}$.add_edge($q_n, \alpha(i)$);

---

**Figure 5.18:** If the nearest point in $S$ lies in an edge, then the edge is split into two, and a new vertex is inserted into $\mathcal{G}$.
Rapidly Exploring Dense Tree (RDT)

LaValle, 1998

**Rapidly Exploring Dense Trees**

Figure 5.16: The basic algorithm for constructing RDTs (which includes RRTs as a special case) when there are no obstacles. It requires the availability of a dense sequence, \( \alpha \), and iteratively connects from \( \alpha(i) \) to the nearest point among all those reached by \( G \).

**5.5. The Exploration Algorithm**

Before explaining how to use these trees to solve a planning query, imagine that the goal is to get as close as possible to every configuration, starting from an initial configuration. The method works for any dense sequence. Once again, let \( \alpha \) denote an infinite, dense sequence of samples in \( C \). The \( i \)th sample is denoted by \( \alpha(i) \). This may possibly include a uniform, random sequence, which is only dense with probability one. Random sequences that induce a nonuniform bias are also acceptable, as long as they are dense with probability one.

An RDT is a topological graph, \( G(V, E) \). Let \( S \subset C \) free indicate the set of all points reached by \( G \). Since each \( e \in E \) is a path, this can be expressed as the swath, \( S \), of the graph, which is defined as \( S = \bigcup_{e \in E} e([0,1]) \).

In (5.40), \( e([0,1]) \subseteq C \) free is the image of the path \( e \).

The exploration algorithm is first explained in Figure 5.16 without any obstacles or boundary obstructions. It is assumed that \( C \) is a metric space. Initially, a vertex is made at \( q_0 \). For \( k \) iterations, a tree is iteratively grown by connecting \( q_n \leftarrow \text{NEAREST}(S(G), \alpha(i)) \); \( G \).add_edge(\( q_n \), \( \alpha(i) \));

---

**Figure 5.18**

If the nearest point in \( S \) lies in an edge, then the edge is split into two, and a new vertex is inserted into \( G \).

---

**Figure 5.19**

In the early iterations, the RRT quickly reaches the unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.
RDT with iterations

Figure 5.18: If the nearest point in $S$ lies in an edge, then the edge is split into two, and a new vertex is inserted into $G$.

Figure 5.19: In the early iterations, the RRT quickly reaches unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.

LaValle, 1998
RRT* (Karaman and Frazolli, 2010)
RRT* (Karaman and Frazolli, 2010)

At the $i^{th}$ iteration,

- end
RRT* (Karaman and Frazolli, 2010)

At the $i^{th}$ iteration,

- end

SAMPLE
RRT* (Karaman and Frazolli, 2010)

At the $i^{th}$ iteration,

- SAMPLE
- FIND BEST PARENT
RRT* (Karaman and Frazolli, 2010)

At the $i^{th}$ iteration,

- Sample
- Find best parent
- Rewire to children
RRT* is asynchronous Value Iteration!
Can we do better?
Informed RRT*

- RRT* is asymptotically optimal everywhere.
- This is unnecessary for single-query planning.

https://www.youtube.com/watch?v=nsf-5Mzwlu4&t=48s  J. Gammell, S. Srinivasa, T. Barfoot, 2014
Informed RRT*

- RRT* is asymptotically optimal everywhere.
- This is unnecessary for single-query planning.

https://www.youtube.com/watch?v=nsl-5MZfwu4&t=48s  
J. Gammell, S.Srinivasa, T.Barfoot, 2014
Today’s discussion

1. Why would we want to interleave?

2. How do we search when we interleave?
   (repairing search)

3. How do we improve graphs when we interleave?
   (incremental sampling)

4. Putting it all together
Batch Informed Trees

- BIT* uses **batches** of random samples to define an **implicit** random geometric graph (RGG).

- It then uses a **heuristic** to search the RGG in order of decreasing solution quality (e.g., A*).

https://www.youtube.com/watch?v=TQIoCC48gp4

J. Gammell, S. Srinivasa, T. Barfoot, 2015
Batch Informed Trees

- BIT* uses **batches** of random samples to define an **implicit** random geometric graph (RGG).

- It then uses a **heuristic** to search the RGG in order of decreasing solution quality (e.g., A*).