Reconnection with the Ideal Tree
A New Approach to Real-Time Search

León Illanes

Department of Computer Science
School of Engineering
Pontificia Universidad Católica de Chile
Santiago, Chile

January 10, 2014
Agent-centered Search

- Search in initially unknown environments.
Agent-centered Search

- Search in initially unknown environments.
- Search in dynamic environments.
Agent-centered Search

- Search in initially unknown environments.
- Search in dynamic environments.
- Real-time search.
Agent-centered Search

- Search in initially unknown environments.
- Search in dynamic environments.
- Real-time search.
The LRTA* Algorithm

Learning Real-Time A*

- Local A*-like search around the agent
- Move towards the best state in the local region
Learning Real Time A*
Learning Real Time A*
<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Learning Real Time A*

León Illanes  
Reconnection with the Ideal Tree
Learning Real Time A*
Agent-centered Search

Issue: Heuristic Depressions

Learning Real Time A*
<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Learning** Real Time A*
Learning Real Time A*
Reconnection with the Ideal Tree

León Illanes

Learning Real Time A*
Learning Real Time A*
### Heuristics Learning (à la LRTA*)

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (with free-space assumption)
### Heuristic learning (à la LRTA*)

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (w/ free-space assumption)
### Heuristic learning (à la LRTA*)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

<table>
<thead>
<tr>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
### Heuristic learning (à la LRTA*)

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (w/ free-space assumption)
### Heuristic learning (à la LRTA*)

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
Heuristic learning (à la LRTA*)

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td><strong>0</strong></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment
(w/ free-space assumption)
Heuristic learning (à la LRTA*)

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment (w/ free-space assumption)
Heuristic learning (à la LRTA*)

Path finding in an unknown environment
(w/ free-space assumption)
How do we avoid erratic movements?

- More lookahead
- More learning
- Pruning states
How do we avoid erratic movements?

- More lookahead
- More learning
- Pruning states

We asked ourselves: Anything simpler?
Design principles

1. Avoid expensive computation

2. Exploit the heuristic
Design principles

1. Avoid expensive computation
   - Sorting
   - Learning

2.
Design principles

1. Avoid expensive computation
   - Sorting
   - Learning

2. Exploit the heuristic
The FRIT Algorithm

Follow and Reconnect with the Ideal Tree

León Illanes

Reconnection with the Ideal Tree
**Definition (Ideal Tree)**

For a problem graph $G$ with goal $g$ and free-space assumption graph $G_M$, we define an Ideal Tree to be any spanning tree for $G_M$ rooted at $g$. 

León Illanes

Reconnection with the Ideal Tree
The Ideal Tree

Definition (Ideal Tree)

For a problem graph \( G \) with goal \( g \) and free-space assumption graph \( G_M \), we define an Ideal Tree to be any spanning tree for \( G_M \) rooted at \( g \).

In practice:

\[
parent(s) = \arg\min_{u: (s, u) \in E(G_M)} c(s, u) + h(u)
\]
The Ideal Tree

Reconnection with the Ideal Tree
The Ideal Tree

Follow and Reconnect
FRIT and Real-Time Search
Properties

León Illanes
Reconnection with the Ideal Tree
**FRIT**

**Input:** Given the free-space assumption graph $G_M$, a goal $g$, and a starting node $s_0$.

$s \leftarrow s_0$ \hspace{1cm} // Set the current state to $s_0$

while $s \neq g$ do


FRIT

**Input:** Given the free-space assumption graph $G_M$, a goal $g$, and a starting node $s_0$.

$s \leftarrow s_0$ // Set the current state to $s_0$

**while** $s \neq g$ **do**

- Observe the environment around $s$ and remove non-existent arcs from $G_M$.
**FRIT**

**Input**: Given the free-space assumption graph \( G_M \), a goal \( g \), and a starting node \( s_0 \).

\[
s \leftarrow s_0 \quad \text{// Set the current state to } s_0
\]

while \( s \neq g \) do

- Observe the environment around \( s \) and remove non-existent arcs from \( G_M \).
- if \( s \) has no parent node then
  - **Reconnect:**

León Illanes

Reconnection with the Ideal Tree
FRIT

**Input:** Given the free-space assumption graph $G_M$, a goal $g$, and a starting node $s_0$.

$s ← s_0$  // Set the current state to $s_0$

while $s ≠ g$ do

Observe the environment around $s$ and remove non-existent arcs from $G_M$.

if $s$ has no parent node then

Reconnect:

Follow:
Input: Given the free-space assumption graph $G_M$, a goal $g$, and a starting node $s_0$.

\[
s \leftarrow s_0
\]

\[
// \text{Set the current state to } s_0
\]

\[
\text{while } s \neq g \text{ do}
\]

Observe the environment around $s$ and remove non-existent arcs from $G_M$.

if $s$ has no parent node then

Reconnect:

Follow:

\[
s \leftarrow \text{parent}(s)
\]

// Move the agent to the parent of $s$
**FRIT**

**Input:** Given the free-space assumption graph $G_M$, a goal $g$, and a starting node $s_0$.

$$s \leftarrow s_0$$ // Set the current state to $s_0$

**while** $s \neq g$ **do**

- Observe the environment around $s$ and remove non-existent arcs from $G_M$.

**if** $s$ **has no parent node** **then**

  **Reconnect:**
  - Locally search around $s$ to find any state $s'$ connected to $g$.
  - Update the Ideal Tree to include the path from $s$ to $s'$.

**Follow:**

$$s \leftarrow \text{parent}(s)$$ // Move the agent to the parent of $s$
FRIT by example

Observe   Follow   Reconnect

Reconnection with the Ideal Tree
FRIT by example

Observe Follow Reconnect

Diagram of a grid with arrows indicating the process of Observe, Follow, and Reconnect.
FRIT by example

Observe Follow Reconnect
FRIT by example

Observe Follow Reconnect
FRIT by example

Observe  Follow  Reconnect
FRIT by example

Observe  Follow  Reconnect
FRIT by example

Observe  Follow  Reconnect
FRIT by example

Observe Follow Reconnect

Reconnection with the Ideal Tree
FRIT by example

Observe  Follow  Reconnect

Reconnection with the Ideal Tree
FRIT by example

Observe  Follow  Reconnect
FRIT by example

Observe   Follow   Reconnect

Reconnection with the Ideal Tree
FRIT by example

Observe  Follow  Reconnect
FRIT by example

Observe  Follow  Reconnect
A better example
A better example

León Illanes
Reconnection with the Ideal Tree
A better example
A better example
A better example
A better example
A better example
A better example
A better example

León Illanes
Reconnection with the Ideal Tree
A better example
A better example
A better example
A better example
A better example
A better example
A better example
A better example
A better example
Video!
As described, FRIT is not a Real-Time Search Algorithm. We need a bound on the amount of states visited while reconnecting.
The Real-Time Property

- As described, FRIT is not a Real-Time Search Algorithm.
- We need a bound on the amount of states visited while reconnecting.
- What to do when the bound is surpassed?
Two approaches

1. Standard FRIT: Do nothing. [RIBH13, RIBH14]
2. FRIT RT: Use a Real-Time Search Algorithm for Reconnection. [RIBH14]
Two approaches

1. Standard FRIT: Do nothing... [RIBH13, RIBH14]
Two approaches

1. Standard FRIT: Do nothing... [RIBH13, RIBH14]
2. FRIT_{RT}: Use a Real-Time Search Algorithm for Reconnection. [RIBH14]
Complexity

- **Follow** is $O(1)$
Complexity

- **Follow** is $O(1)$
- **Reconnect** can be $O(|V|)$
Complexity

- **Follow** is $O(1)$
- **Reconnect** can be $O(|V|)$

**Reconnect can be $O(|V|)$**.

Using BFS as the local search algorithm, we check at most $|V|$ nodes to see if they are connected to the goal. This check can be done as a recursive function with no side effects and can thus be memoized, ensuring that for each reconnection search we do at most $|V|$ comparisons.
Complexity

- **Follow** is $O(1)$
- **Reconnect** can be $O(|V|)$

**Reconnect** can be $O(|V|)$.

Using BFS as the local search algorithm, we check at most $|V|$ nodes to see if they are connected to the goal. This check can be done as a recursive function with no side effects and can thus be memoized, ensuring that for each reconnection search we do at most $|V|$ comparisons.

Additionally, we prove correctness and completeness for both FRIT and FRIT$_{RT}$, while giving an explicit upper bound of $\frac{(|V|+1)^2}{4}$ moves for FRIT and $O(|V|^3)$ moves for FRIT$_{RT}$. 
FRIT immediately converges to a suboptimal solution
FRIT immediately converges to a suboptimal solution.
FRIT immediately converges to a suboptimal solution.
Games: $\text{FRIT}_{\text{RT}}$ halves daRTAA*'s solutions

```
<table>
<thead>
<tr>
<th></th>
<th>FRIT rt(RTAA)</th>
<th>FRIT rt(daRTAA)</th>
<th>RTAA</th>
<th>daRTAA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Mazes: Similar tendencies

![Graph showing average solution cost and time per planning episode for different algorithms: FRIT rt(RTAA), FRIT rt(daRTAA), RTAA, and daRTAA. The x-axis represents average time per planning episode (us), while the y-axis shows average solution cost in log-scale. The graph compares the performance of these algorithms across different maze sizes, illustrating similar tendencies in solution cost and time.](image-url)
Games: FRIT dominates for very small $t$
Mazes: Again, similar tendencies
FRIT(BFS) obtains better solutions

The graph below compares the average solution length (log-scale) against the average time per planning episode (us) for two approaches: FRIT_rt(daRTAA) and FRIT(BFS).

- FRIT_rt(daRTAA) shows a higher solution length but a shorter average time compared to FRIT(BFS).
- FRIT(BFS) has a lower solution length and a longer average time.

This indicates that FRIT(BFS) is more efficient in terms of solution quality for larger planning episodes.
Other applications

- Optimizing for pathfinding in grids [RIB14]
- Moving-target search
- Dense graphs (e.g.: Airport networks)
We presented a family of real-time search algorithms which:

- Are easy to implement
- Avoid expensive computations
- Converge to suboptimal solutions in the second trial
- Significantly outperform standard real-time search algorithms when time constraints are tight
Bibliography


Reconnection with the Ideal Tree
A New Approach to Real-Time Search

León Illanes

Department of Computer Science
School of Engineering
Pontificia Universidad Católica de Chile
Santiago, Chile

January 10, 2014
<table>
<thead>
<tr>
<th>$k$</th>
<th>$k$ Avg. Its</th>
<th>$k$ Time/ep ($\mu$s)</th>
<th>$k$ No moves (%)</th>
<th>$k$ Avg. Its</th>
<th>$k$ Time/ep ($\mu$s)</th>
<th>$k$ No moves (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1508631</td>
<td>0.0430</td>
<td>99.80</td>
<td>1144680</td>
<td>0.4152</td>
<td>99.84</td>
</tr>
<tr>
<td>5</td>
<td>303483</td>
<td>0.2148</td>
<td>99.01</td>
<td>229967</td>
<td>2.0727</td>
<td>99.25</td>
</tr>
<tr>
<td>10</td>
<td>152858</td>
<td>0.4283</td>
<td>98.03</td>
<td>115628</td>
<td>4.1376</td>
<td>98.51</td>
</tr>
<tr>
<td>50</td>
<td>32401</td>
<td>2.0940</td>
<td>90.71</td>
<td>24156</td>
<td>20.378</td>
<td>92.86</td>
</tr>
<tr>
<td>100</td>
<td>17370</td>
<td>4.0678</td>
<td>82.67</td>
<td>12723</td>
<td>40.004</td>
<td>86.44</td>
</tr>
<tr>
<td>500</td>
<td>5449</td>
<td>16.115</td>
<td>44.74</td>
<td>3607</td>
<td>172.41</td>
<td>52.15</td>
</tr>
<tr>
<td>1000</td>
<td>4035</td>
<td>24.840</td>
<td>25.38</td>
<td>2583</td>
<td>274.35</td>
<td>33.20</td>
</tr>
<tr>
<td>5000</td>
<td>3073</td>
<td>39.316</td>
<td>2.046</td>
<td>1854</td>
<td>474.29</td>
<td>6.904</td>
</tr>
<tr>
<td>10000</td>
<td>3026</td>
<td>40.487</td>
<td>0.501</td>
<td>1775</td>
<td>514.88</td>
<td>2.764</td>
</tr>
<tr>
<td>50000</td>
<td>3011</td>
<td>40.851</td>
<td>0.030</td>
<td>1728</td>
<td>524.55</td>
<td>0.117</td>
</tr>
<tr>
<td>100000</td>
<td>3011</td>
<td>40.869</td>
<td>0.007</td>
<td>1726</td>
<td>543.66</td>
<td>0.014</td>
</tr>
</tbody>
</table>
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component
Memoizing the tree component