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.

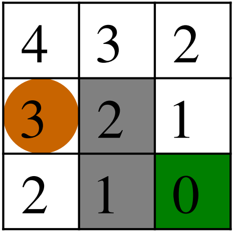
Agent-centered Search

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

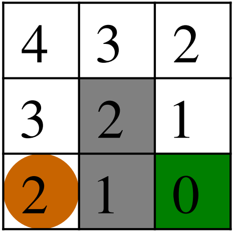
The LRTA* Algorithm

Learning Real-Time A*

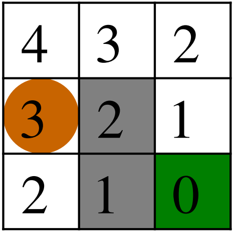
- Local A*-like search around the agent
- Move towards the best state in the local region



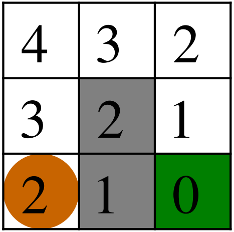
Learning Real Time A*



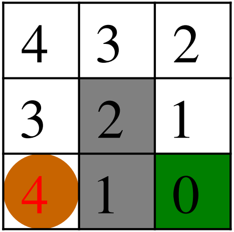
Learning Real Time A*



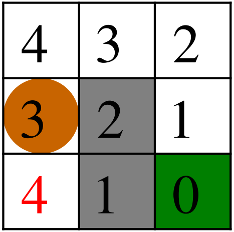
Learning Real Time A*



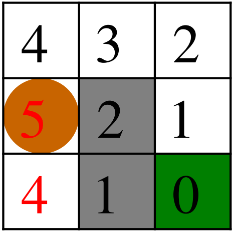
Learning Real Time A*



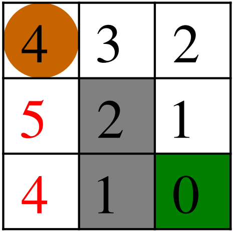
Learning Real Time A*



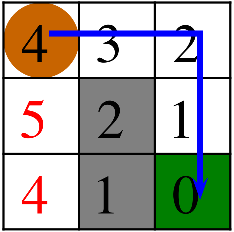
Learning Real Time A*



Learning Real Time A*

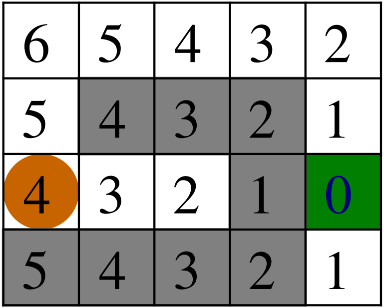


Learning Real Time A*



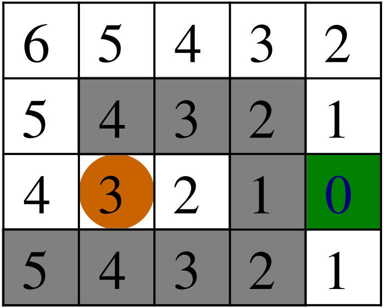
Learning Real Time A*

Heuristic learning (à la LRTA*)



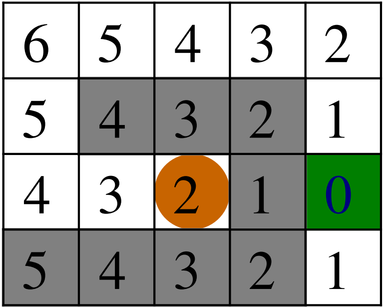
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



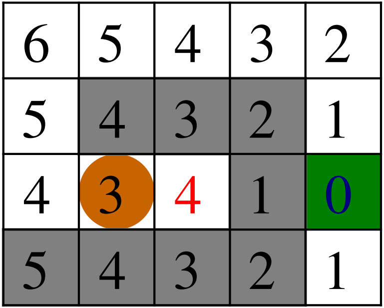
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)

6	5	4	3	2
5	4	3	2	1
4	3	4	1	0
5	4	3	2	1

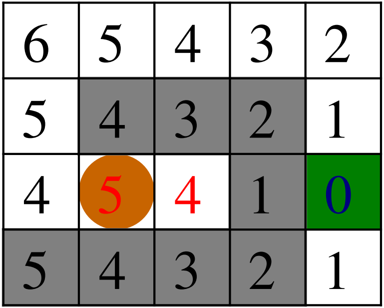
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



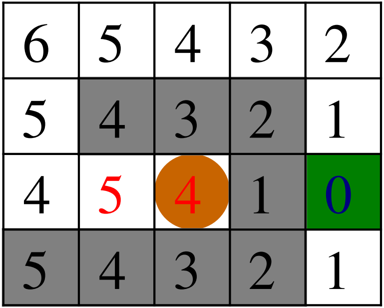
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



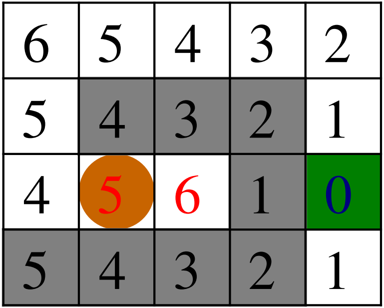
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



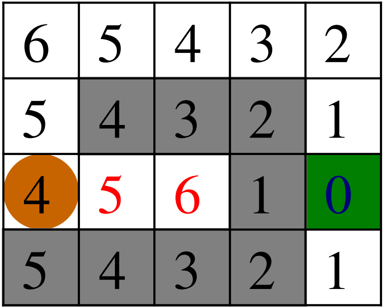
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



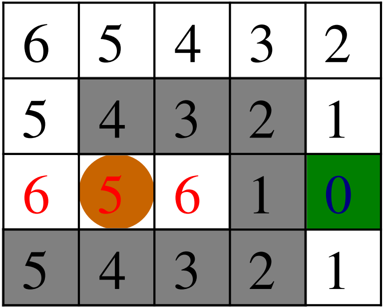
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



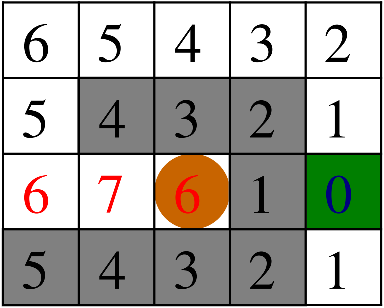
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



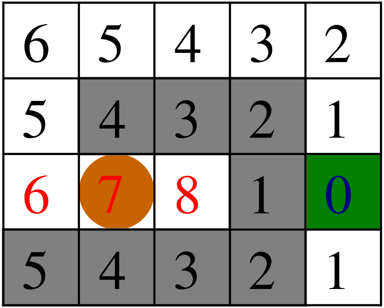
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



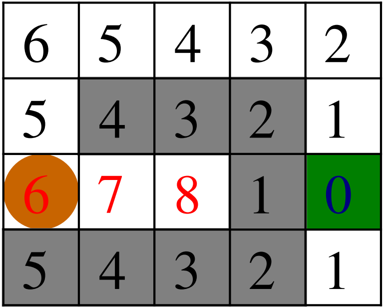
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



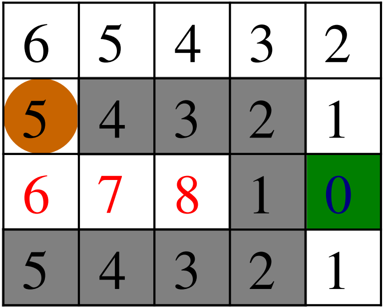
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



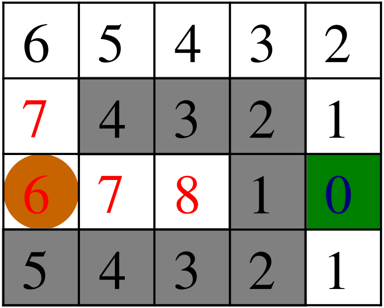
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



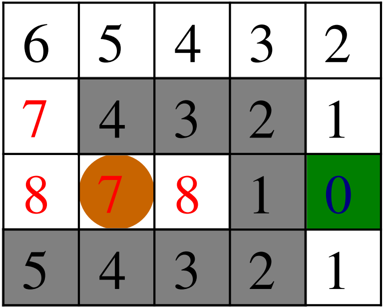
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



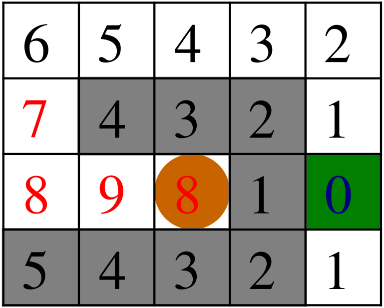
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



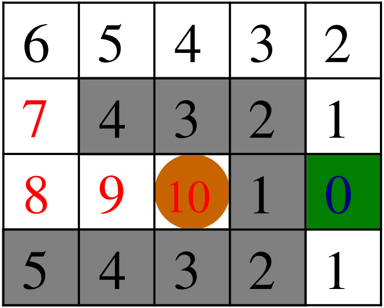
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



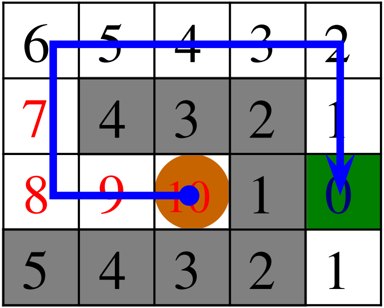
Path finding in an unknown environment
(w/ free-space assumption)

Heuristic learning (à la LRTA*)



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.

2.

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

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.

The Ideal Tree

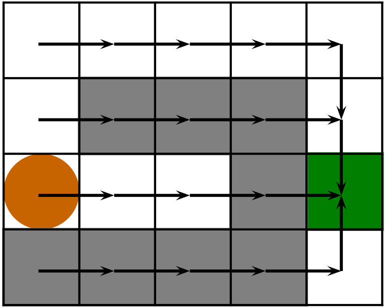
Definition (Ideal Tree)

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.

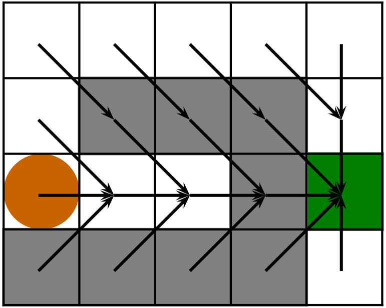
In practice:

$$parent(s) = \operatorname*{argmin}_{u:(s,u)\in E(G_M)} c(s,u) + h(u)$$

The Ideal Tree



The Ideal Tree



FRIT

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

$s \leftarrow s_0$ `// 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.

$s \leftarrow s_0$ `// 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:

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:

  Follow:

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:

Follow:

$s \leftarrow parent(s)$ `// 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.

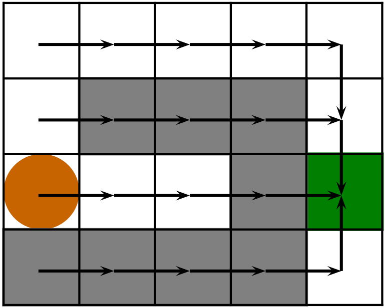
$s \leftarrow s_0$ `// Set the current state to` s_0

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:**
- $s \leftarrow parent(s)$ `// Move the agent to the parent of` s

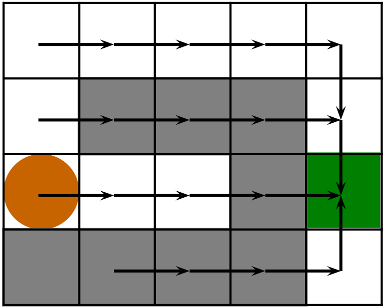
FRIT by example

Observe Follow Reconnect



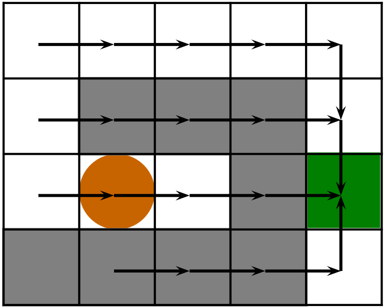
FRIT by example

Observe Follow Reconnect



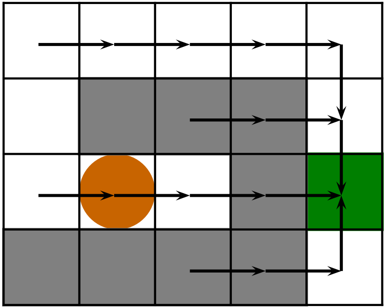
FRIT by example

Observe **Follow** Reconnect



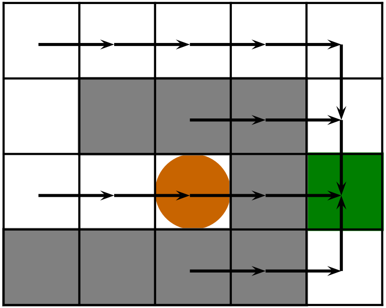
FRIT by example

Observe Follow Reconnect



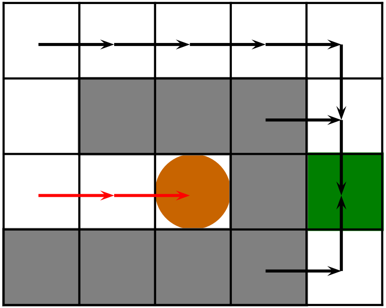
FRIT by example

Observe **Follow** Reconnect



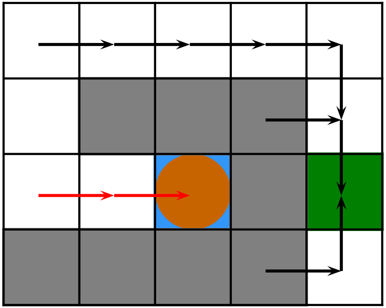
FRIT by example

Observe Follow Reconnect



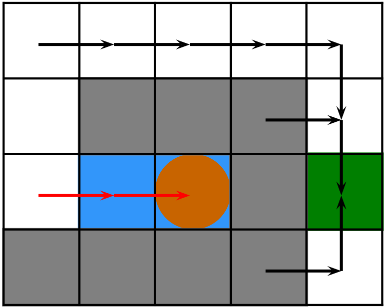
FRIT by example

Observe Follow **Reconnect**



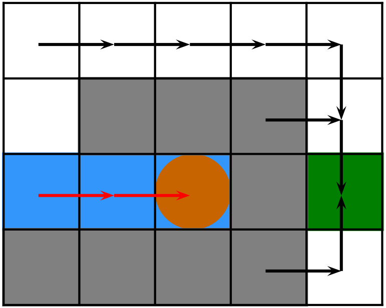
FRIT by example

Observe Follow **Reconnect**



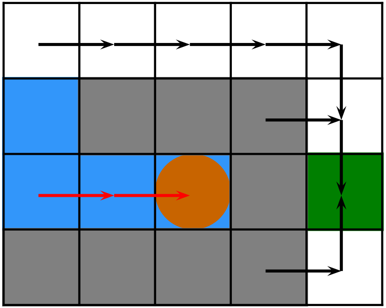
FRIT by example

Observe Follow **Reconnect**



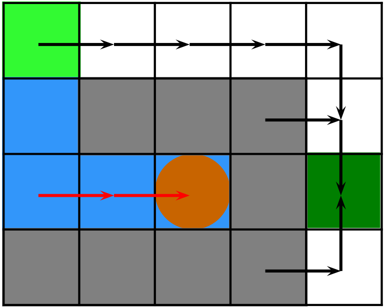
FRIT by example

Observe Follow **Reconnect**



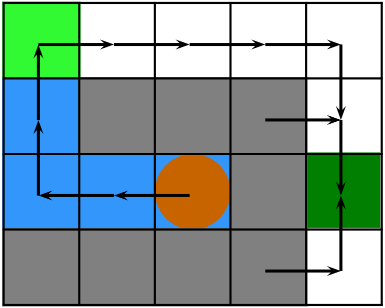
FRIT by example

Observe Follow **Reconnect**



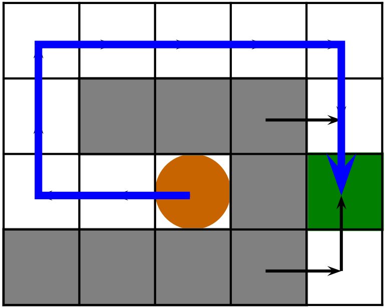
FRIT by example

Observe Follow **Reconnect**

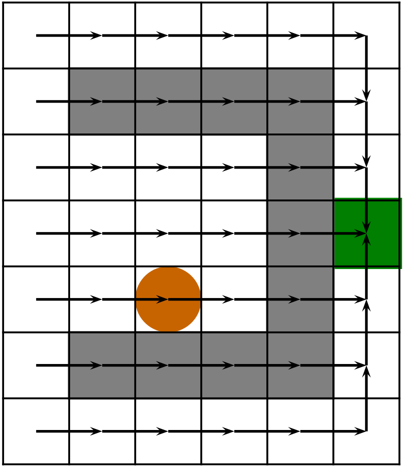


FRIT by example

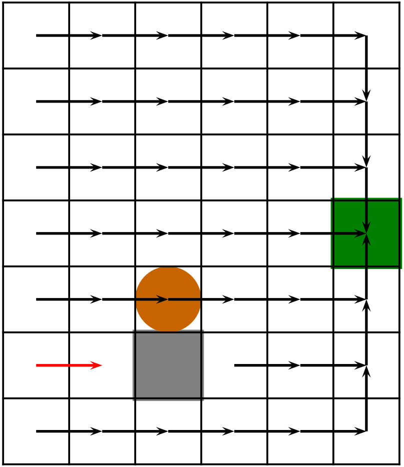
Observe **Follow** Reconnect



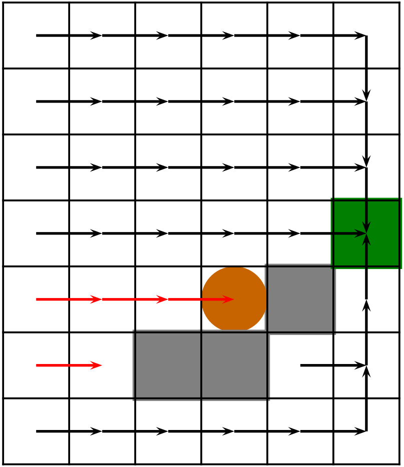
A better example



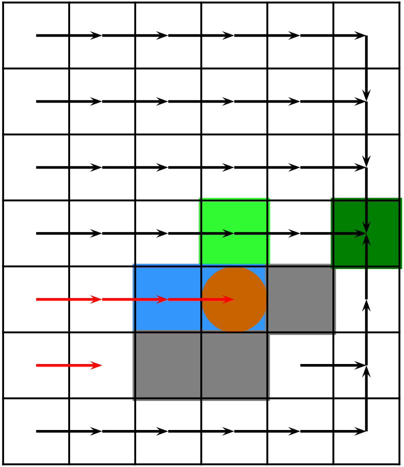
A better example



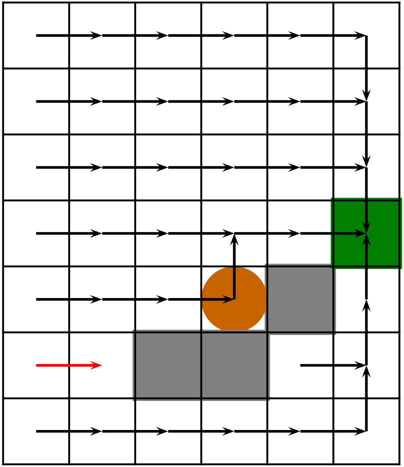
A better example



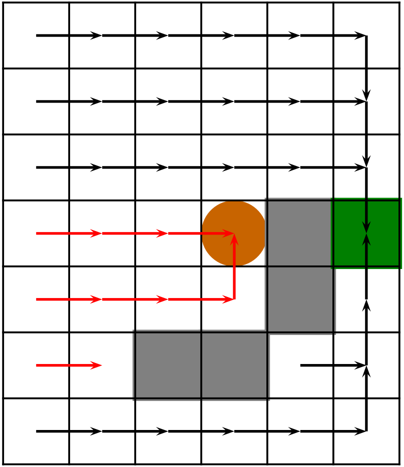
A better example



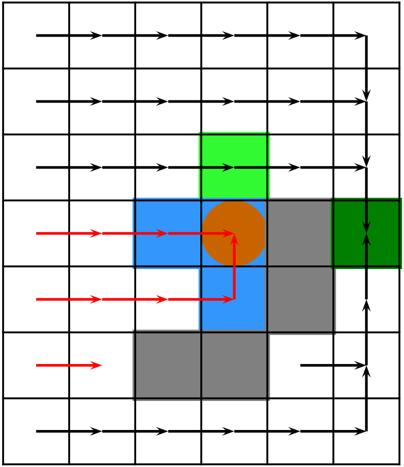
A better example



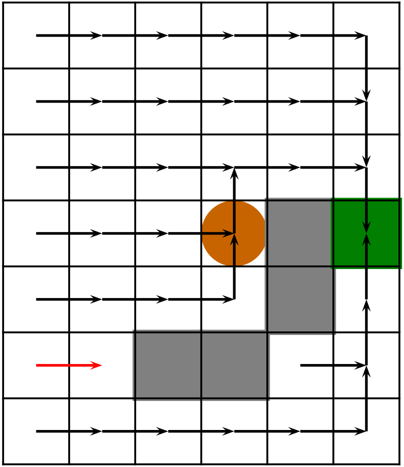
A better example



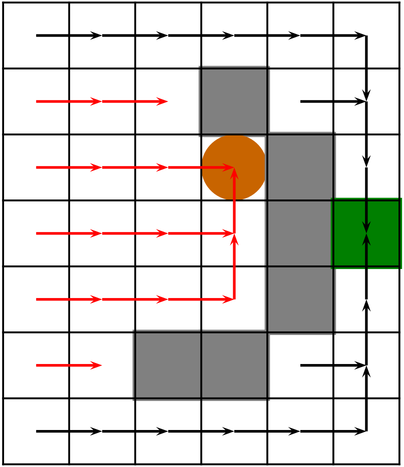
A better example



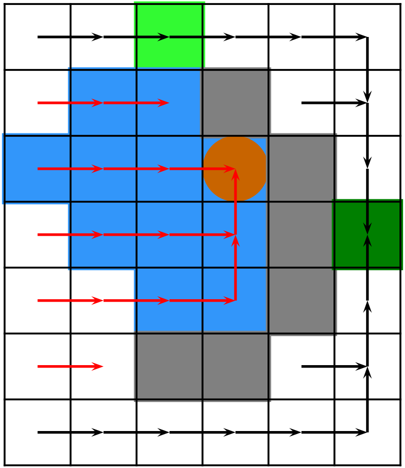
A better example



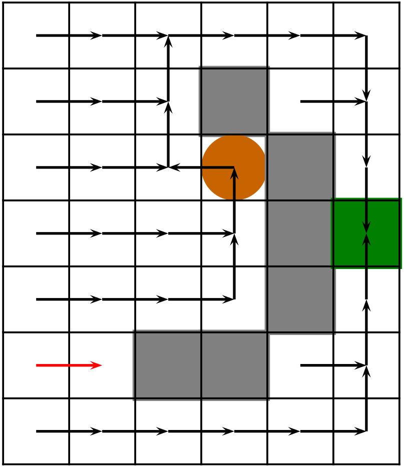
A better example



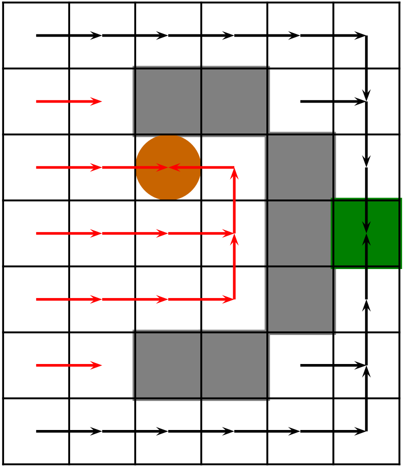
A better example



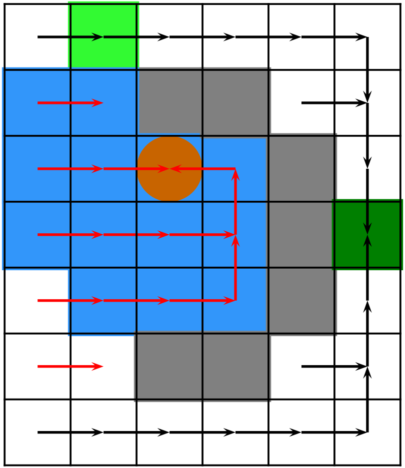
A better example



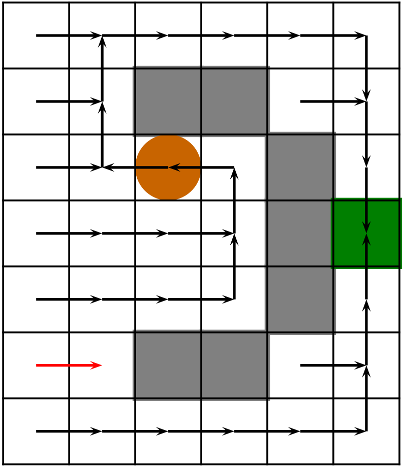
A better example



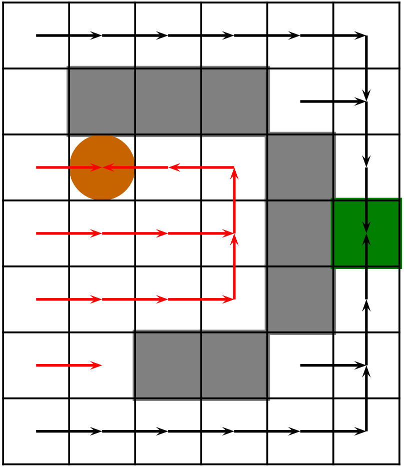
A better example



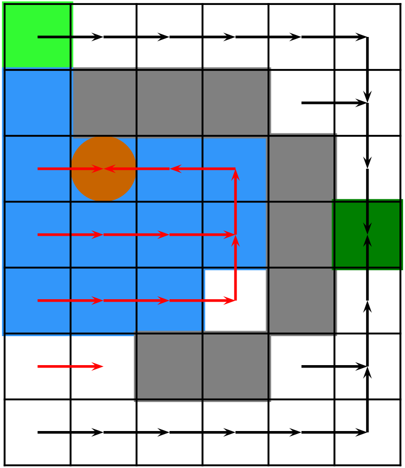
A better example



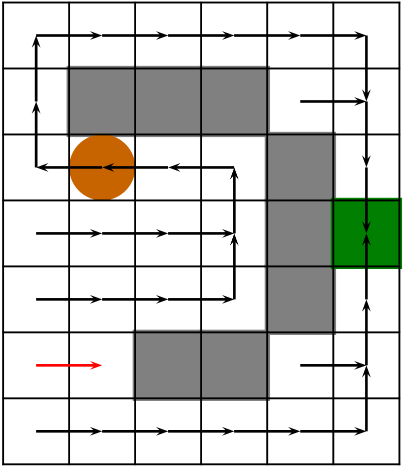
A better example



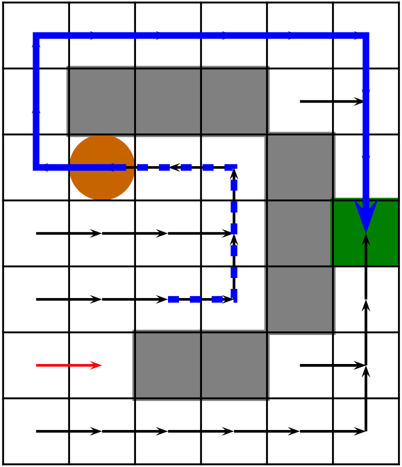
A better example



A better example



A better example



Video!

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.

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.
2.

Two approaches

1. Standard FRIT: Do nothing... [RIBH13, RIBH14]
2.

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 $O(|V|)$

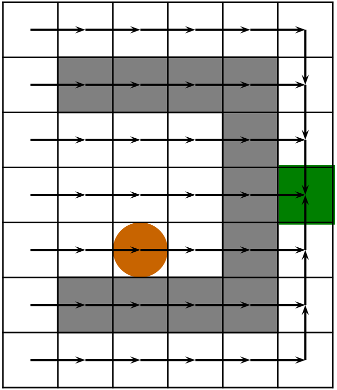
Reconnect can be $O(|V|)$.

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 correcteness 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}.

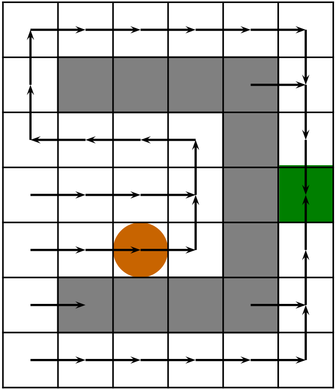
Convergence

FRIT immediately converges to a suboptimal solution



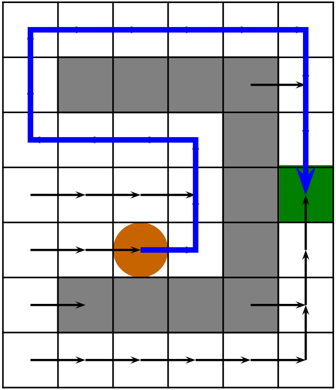
Convergence

FRIT immediately converges to a suboptimal solution

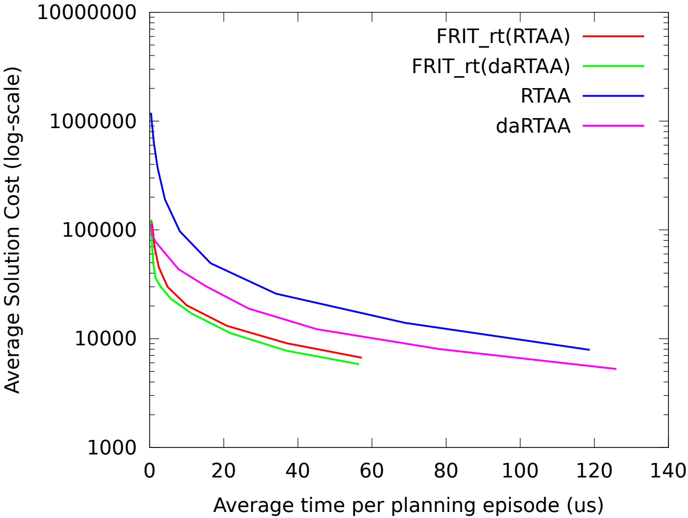


Convergence

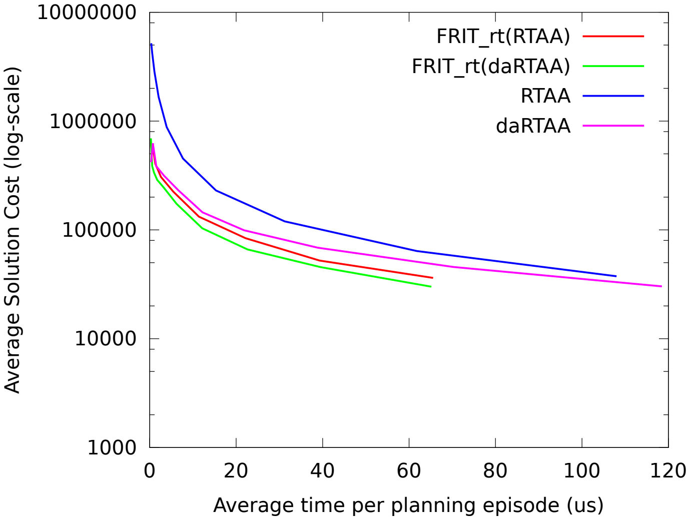
FRIT immediately converges to a suboptimal solution



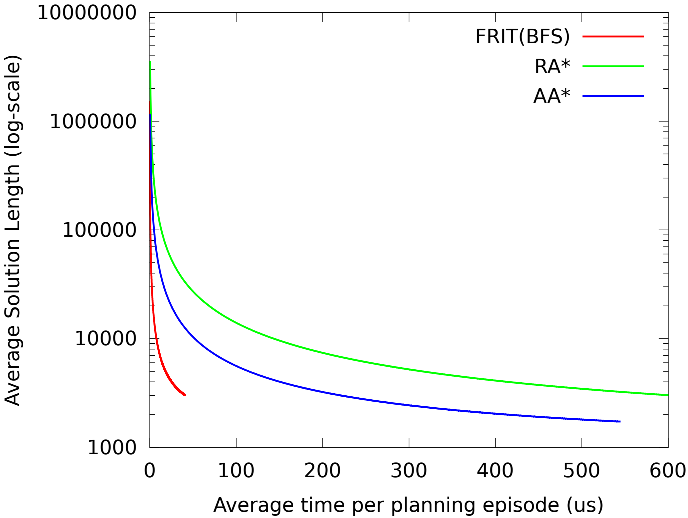
Games: $FRIT_{RT}$ halves daRTAA*'s solutions



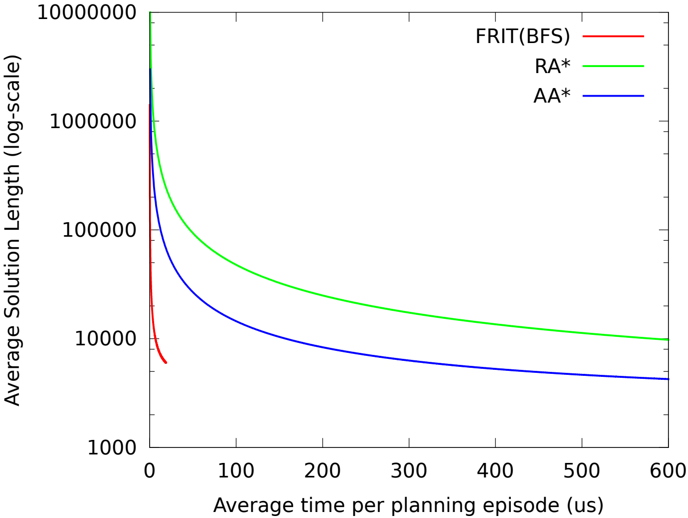
Mazes: Similar tendencies



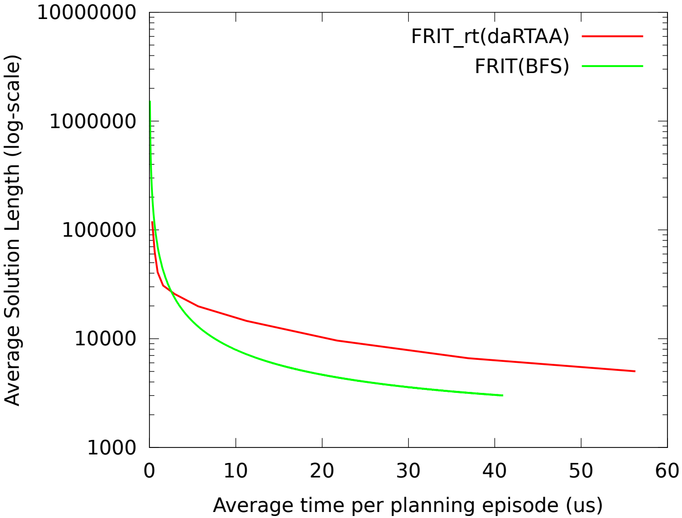
Games: FRIT dominates for very small t



Mazes: Again, similar tendencies



FRIT(BFS) obtains better solutions



Future work

Other applications

- Optimizing for pathfinding in grids [RIB14]
- Moving-target search
- Dense graphs (e.g.: Airport networks)

Summary

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

- Nicolás Rivera, León Illanes, and Jorge A. Baier, *Real-time pathfinding in unknown terrain via reconnection with an ideal tree*, Proceedings of the 14th Ibero-American Conference on Artificial Intelligence (IBERAMIA) (Santiago, Chile), November 2014, **To appear**.
- Nicolas Rivera, Leon Illanes, Jorge A. Baier, and Carlos Hernández, *Reconnecting with the ideal tree: An alternative to heuristic learning in real-time search*, Proceedings of the 6th Symposium on Combinatorial Search (SoCS), 2013.
- Nicolás Rivera, León Illanes, Jorge A. Baier, and Carlos Hernández, *Reconnection with the ideal tree: A new approach to real-time search*, Journal of Artificial Intelligence Research **50** (2014).

Reconnection with the Ideal Tree

A New Approach to Real-Time Search

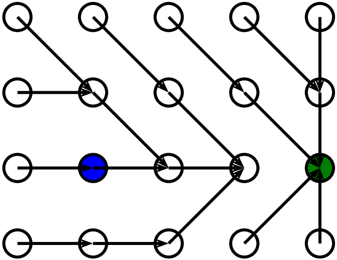
León Illanes

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

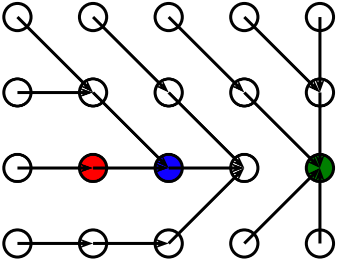
January 10, 2014

	FRIT(BFS)			AA*		
k	Avg. Its	Time/ep (μs)	No moves (%)	Avg. Its	Time/ep (μs)	No moves (%)
1	1508631	0.0430	99.80	1144680	0.4152	99.84
5	303483	0.2148	99.01	229967	2.0727	99.25
10	152858	0.4283	98.03	115628	4.1376	98.51
50	32401	2.0940	90.71	24156	20.378	92.86
100	17370	4.0678	82.67	12723	40.004	86.44
500	5449	16.115	44.74	3607	172.41	52.15
1000	4035	24.840	25.38	2583	274.35	33.20
5000	3073	39.316	2.046	1854	474.29	6.904
10000	3026	40.487	0.501	1775	514.88	2.764
50000	3011	40.851	0.030	1728	524.55	0.117
100000	3011	40.869	0.007	1726	543.66	0.014

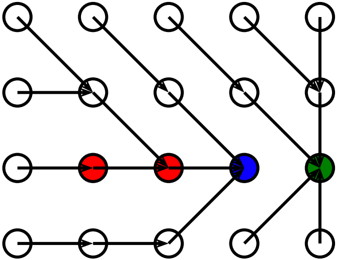
Memoizing the tree component



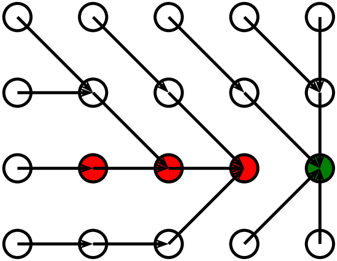
Memoizing the tree component



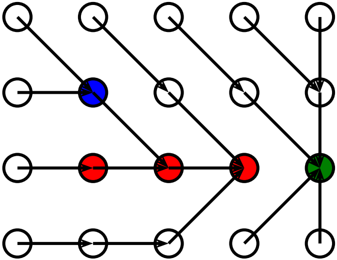
Memoizing the tree component



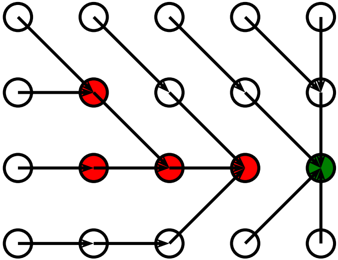
Memoizing the tree component



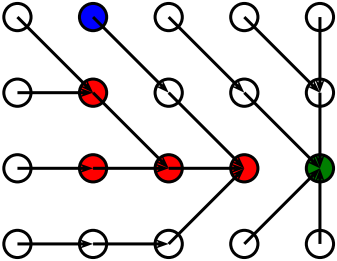
Memoizing the tree component



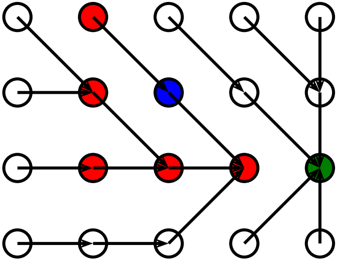
Memoizing the tree component



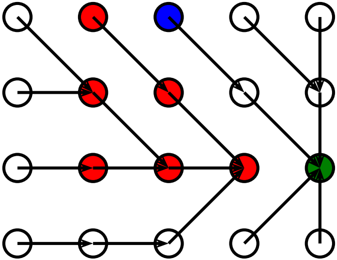
Memoizing the tree component



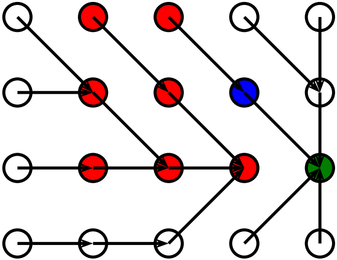
Memoizing the tree component



Memoizing the tree component



Memoizing the tree component



Memoizing the tree component