Lazy Search

Sanjiban Choudhury

TAs: Matthew Rockett, Gilwoo Lee, Matt Schmittle
High-order bit

Expansion of a search wavefront from start to goal

Dijkstra  A*  Weighted A*

Courtesy wikipedia
High-order bit

Expansion of a search wavefront from start to goal

Dijkstra  A*  Weighted A*

Courtesy wikipedia
What do we want?

1. Search to systematically reason over the space of paths

2. Find a (near)-optimal path quickly

   (minimize planning effort)
Best First Search

(Explore the graph by expanding/processing promising nodes)

<table>
<thead>
<tr>
<th>Element (Node)</th>
<th>Priority Value (f-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node S</td>
<td>f(S)</td>
</tr>
</tbody>
</table>
Best First Search

(Explore the graph by expanding/processing promising nodes)

<table>
<thead>
<tr>
<th>Element (Node)</th>
<th>Priority Value (f-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node S</td>
<td>f(S)</td>
</tr>
<tr>
<td>Node A</td>
<td>f(A)</td>
</tr>
<tr>
<td>Node C</td>
<td>f(C)</td>
</tr>
</tbody>
</table>
Different $f(s)$ leads to different algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$f(s)$</th>
<th>Optimality</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Djikstra</td>
<td>$f(s) = g(s)$</td>
<td>Yes</td>
<td>Poor</td>
</tr>
<tr>
<td>A*</td>
<td>$f(s) = g(s) + h(s)$</td>
<td>Yes* (if $h(s)$ is admissible)</td>
<td>Good* (better if $h(s)$ is consistent)</td>
</tr>
</tbody>
</table>
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal.
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

- External knowledge about metrics
  - Euclidean distance admissible because triangle inequality
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

- External knowledge about metrics
  - Euclidean distance admissible because triangle inequality

- Solving a relaxation of the problem (which is easier)
  - Ignore obstacles, ignore dynamics
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

Goal is to come up with cheap-to-compute estimates

- External knowledge about metrics
  - Euclidean distance admissible because triangle inequality

- Solving a relaxation of the problem (which is easier)
  - Ignore obstacles, ignore dynamics

- Statistical knowledge of the problem
  - Keep arms tucked in, don’t drive headfirst into a wall
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal.
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal

But another way to think of heuristics is as a ranking function
What is a heuristic $h(s)$?

So far, we have been thinking of heuristics as an estimate of cost-to-go to goal.

But another way to think of heuristics is as a ranking function.

Even if estimates are off, as long as a heuristic is ranking good states better than bad states, it’s useful.
The case for inadmissible search
The case for inadmissible search

Let’s say you have a $h(s)$ that is very very admissible
The case for inadmissible search

Let’s say you have a $h(s)$ that is very very admissible

But it is actually good at ranking good vs bad states
The case for inadmissible search

Let’s say you have a $h(s)$ that is very very admissible

But it is actually good at ranking good vs bad states

$f(s) = g(s) + h(s)$ might not be best at ranking states. Why?
The case for inadmissible search

Let’s say you have a $h(s)$ that is very very admissible

But it is actually good at ranking good vs bad states

$f(s) = g(s) + h(s)$ might not be best at ranking states. Why?

$f(s) = g(s) + 10h(s)$ might be much better. Why?
Side-effects of inflating heuristics

\[ f(s) = g(s) + h(s) \]

\[ f(s) = g(s) + 5h(s) \]
Side-effects of inflating heuristics

\[ f(s) = g(s) + h(s) \]

\[ f(s) = g(s) + 5h(s) \]
Can we bound the solution quality?

\[ \epsilon \geq 1 \]
Can we bound the solution quality?

\[ g(s) + \epsilon h(s) \quad \epsilon \geq 1 \]

results in

\[ \text{cost(solution)} \leq \epsilon \text{ cost(solution)} \]
• Dijkstra’s: expands states in the order of $f = g$ values

What are the states expanded?

$S_{\text{start}}$ ...

$S_{\text{goal}}$
Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

$S_{\text{start}}$  $\rightarrow$  $S_{\text{goal}}$

Courtesy Max Likhachev
Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values
Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

for large problems this results in A* quickly running out of memory (memory: $O(n)$)

Courtesy Max Likhachev
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g + \epsilon h$ values, $\epsilon > 1 = \text{bias towards states that are closer to goal}$

$S_{\text{start}} \rightarrow$  

$S_{\text{goal}}$

Courtesy Max Likhachev
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g + \epsilon h$ values, $\epsilon > 1$ = bias towards states that are closer to goal

Courtesy Max Likhachev
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

solution is always $\varepsilon$-suboptimal:
$\text{cost(solution)} \leq \varepsilon \cdot \text{cost(optimal solution)}$

Courtesy Max Likhachev
Effect of the Heuristic Function

$\epsilon = 2.5$

$\epsilon = 1.5$

$\epsilon = 1.0$ (optimal search)

Courtesy Max Likhachev
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of \( f = g + \varepsilon h \) values, \( \varepsilon > 1 \) = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over \( 10^{26} \) states

planning with ARA* (anytime version of weighted A*)

Courtesy Max Likhachev
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of \( f = g + \epsilon h \) values, \( \epsilon > 1 \) = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over \( 10^{26} \) states

planning with ARA* (anytime version of weighted A*)

Courtesy Max Likhachev
Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

20DOF simulated robotic arm
state-space size: over $10^{26}$ states

planning with ARA* (anytime version of weighted A*)

Courtesy Max Likhachev
Effect of the Heuristic Function

- planning in 8D ($<x,y>$ for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds

Uses R* - A randomized version of weighted A*

Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza

Courtesy Max Likhachev
Effect of the Heuristic Function

- planning in 8D \((<x,y>\) for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds

Uses R* - A randomized version of weighted A*

Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza
Effect of the Heuristic Function

- planning in 8D ($<x,y>$ for each foothold)
- heuristic is Euclidean distance from the center of the body to the goal location
- cost of edges based on kinematic stability of the robot and quality of footholds

Uses R* - A randomized version of weighted A*

Joint work between Max Likhachev, Subhrajit Bhattacharya, Joh Bohren, Sachin Chitta, Daniel D. Lee, Aleksandr Kushleyev, and Paul Vernaza
But is the **number of expansions** really what we want to minimize in motion planning?

What is the most expensive step?
Edge evaluation is expensive

Check if helicopter intersects with tower

Check if manipulator intersects with table

(Schluman et al. ’14)
Edge evaluation **dominates** planning time

Hauser, Kris., Lazy collision checking in asymptotically-optimal motion planning. *ICRA* 2015
Let’s revisit Best First Search

<table>
<thead>
<tr>
<th>Element (Node)</th>
<th>Priority Value (f-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node S</td>
<td>f(S)</td>
</tr>
</tbody>
</table>

Node S connected to A, B, C, and G in a graph.
Let’s revisit Best First Search

<table>
<thead>
<tr>
<th>Element (Node)</th>
<th>Priority Value (f-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node S</td>
<td>f(S)</td>
</tr>
<tr>
<td>Node A</td>
<td>f(A)</td>
</tr>
<tr>
<td>Node C</td>
<td>f(C)</td>
</tr>
</tbody>
</table>

![Graph diagram](image)
What if we never use C? Wasted collision check!

<table>
<thead>
<tr>
<th>Element (Node)</th>
<th>Priority Value (f-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node S</td>
<td>(f(S))</td>
</tr>
<tr>
<td>Node A</td>
<td>(f(A))</td>
</tr>
<tr>
<td>Node C</td>
<td>(f(C))</td>
</tr>
</tbody>
</table>

![Diagram](image)
The provable virtue of laziness:

Take the thing that’s expensive (collision checking) and procrastinate as long as possible till you have to evaluate it!
Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014
Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:
Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:

1. When expanding a node, don’t collision check edge to successors
   (be optimistic and assume the edge will be valid)
Lazy (weighted) A*

Cohen, Phillips, and Likhachev 2014

Key Idea:

1. When expanding a node, don’t collision check edge to successors
   (be optimistic and assume the edge will be valid)

2. When expanding a node, collision check the edge to parent
   (expansion means this node is good and worth the effort)
Lazy (weighted) A* 
Cohen, Phillips, and Likhachev 2014

Key Idea:

1. When expanding a node, don’t collision check edge to successors  
   (be optimistic and assume the edge will be valid)

2. When expanding a node, collision check the edge to parent  
   (expansion means this node is good and worth the effort)

3. Important: OPEN list will have multiple copies of a node  
   (multiple candidate parents since we haven’t collision check)
Lazy A*

Cohen, Phillips, and Likhachev 2014

Non lazy A*

while($s_{goal}$ is not expanded)
  remove $s$ with the smallest
  $[f(s) = g(s) + h(s)]$ from OPEN;
  insert $s$ into CLOSED;
  for every successor $s'$ of $s$ such
    that $s'$ not in CLOSED
    if edge $(s,s')$ in collision
      $c(s,s') = \infty$
    if $g(s') > g(s) + c(s,s')$
      $g(s') = g(s) + c(s,s')$;
      insert $s'$ into OPEN;
Lazy A*

Cohen, Phillips, and Likhachev 2014

Non lazy A*

while($s_{goal}$ is not expanded)

remove $s$ with the smallest $[f(s) = g(s)+h(s)]$ from OPEN;

insert $s$ into CLOSED;

for every successor $s'$ of $s$ such that $s'$ not in CLOSED

if $edge \ (s, s')$ in collision

$c(s, s') = \infty$

if $g(s') > g(s) + c(s, s')$

$g(s') = g(s) + c(s, s')$;

insert $s'$ into OPEN;

Lazy A*

while($s_{goal}$ is not expanded)

remove $s$ with the smallest $[f(s) = g(s)+h(s)]$ from OPEN;

if $s$ is in CLOSED

continue;

if $edge(parent(s), s)$ in collision

continue;

insert $s$ into CLOSED;

for every successor $s'$ of $s$ such that $s'$ not in CLOSED

no collision checking of edge

if $g(s') > g(s) + c(s, s')$

$g(s') = g(s) + c(s, s')$;

insert $s'$ into OPEN; // multiple copies
A*

Let’s say S-A is in collision and true shortest path is S-B-A-G
Let’s say S-A is in collision and true shortest path is S-B-A-G

A* will collision check all N+2 edges!
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G

Let's set $f(s) = g(s)$

<table>
<thead>
<tr>
<th>OPEN</th>
<th>CLOSED</th>
<th>CollChecked</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = 1$</td>
<td>B (from S)</td>
<td>S</td>
</tr>
<tr>
<td>$f = 2$</td>
<td>A (from S)</td>
<td></td>
</tr>
<tr>
<td>$f = 1000$</td>
<td>X (from S)</td>
<td></td>
</tr>
</tbody>
</table>

N edges all in collision
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G

We have TWO copies of A in OPEN!

<table>
<thead>
<tr>
<th>OPEN</th>
<th>CLOSED</th>
<th>CollCheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>f = 2</td>
<td>A (from S)</td>
<td>S</td>
</tr>
<tr>
<td>f = 3</td>
<td>A (from B)</td>
<td>B</td>
</tr>
</tbody>
</table>

N edges all in collision
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G

We have TWO copies of A in OPEN!

N edges all in collision
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G.
Yet another algorithm to remember....!?!?!
Best First Search explains it all

SIT BACK 'N RELAX

WE GOT THIS
Best First Search as a Meta++ algorithm

<table>
<thead>
<tr>
<th>Element</th>
<th>Ranking value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element A</td>
<td>f(A)</td>
</tr>
<tr>
<td>Element B</td>
<td>f(B)</td>
</tr>
</tbody>
</table>

POP best element

Process Element

Done?

Update Queue

N   Y
If we want to minimize the number of vertices expanded, then elements of the queue should be candidate vertices to expand!
If we want to minimize the number of edges evaluated, then

......
If we want to minimize the number of edges evaluated, then

......

elements of the queue should be candidate edges to evaluate!
Best First Search as a Meta++ algorithm

<table>
<thead>
<tr>
<th>Element</th>
<th>Ranking value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element A</td>
<td>f(A)</td>
</tr>
<tr>
<td>Element B</td>
<td>f(B)</td>
</tr>
</tbody>
</table>

- POP best element
- Process Element
- Done?
  - Y
  - N
- Update Queue
How do we define a queue over edges???

**OPEN list of A***

<table>
<thead>
<tr>
<th>Element (Vertex)</th>
<th>Value (f-value of path through vertex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex A</td>
<td>( f(A) = g(A) + h(A) )</td>
</tr>
<tr>
<td>Vertex B</td>
<td>( f(B) = g(B) + h(B) )</td>
</tr>
</tbody>
</table>
How do we define a queue over edges???

**OPEN list of A***

<table>
<thead>
<tr>
<th>Element (Vertex)</th>
<th>Value (f-value of path through vertex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex A</td>
<td>( f(A) = g(A) + h(A) )</td>
</tr>
<tr>
<td>Vertex B</td>
<td>( f(B) = g(B) + h(B) )</td>
</tr>
</tbody>
</table>

**OPEN list of Lazy A***

<table>
<thead>
<tr>
<th>Element (Edge)</th>
<th>Value (f-value of path through edge)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge (X,Y)</td>
<td>( f(X,Y) = g(X) + c(X,Y) + h(Y) )</td>
</tr>
<tr>
<td>Edge (P,Q)</td>
<td>( f(P,Q) = g(P) + c(P,Q) + h(Q) )</td>
</tr>
<tr>
<td>Edge (W,Y)</td>
<td>( f(W,Y) = g(W) + c(W,Y) + h(Y) )</td>
</tr>
</tbody>
</table>
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G
Lazy A*

Let’s say S-A is in collision and true shortest path is S-B-A-G

Let's consider a diagram with nodes S (start), A, B, and G (goal). The path S-B-A-G is marked in green. The algorithm works by maintaining two lists: OPEN and CLOSED. The OPEN list contains nodes that are candidates for the next step, while the CLOSED list contains nodes that have already been processed.

The function f(.) is defined as:

\[ f(\cdot) = g(S) + c(S,A) + h(A) \]

For node S, the cost to reach it is 0, the cost from S to A is 1, and the heuristic from A to G is 2, so the total cost is 3. Similarly,

\[ f(\cdot) = g(B) + c(B,A) + h(A) \]

For node B, the cost to reach it is 1, the cost from B to A is 1, and the heuristic from A to G is 2, so the total cost is 4.

The nodes are marked as:

- OPEN list:
  - S—A
  - B—A

- CLOSED list:
  - S
  - B

The diagram also shows an edge X—S in collision.
Lazy A* as BFS

<table>
<thead>
<tr>
<th>Element</th>
<th>Ranking value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element A</td>
<td>f(A)</td>
</tr>
<tr>
<td>Element B</td>
<td>f(B)</td>
</tr>
</tbody>
</table>

POP best element

Process Element

Collision Check Edge

Queue over edges

Add successors

Update Queue

Done?

N

Y
What is the laziest that we can be?
What is the laziest that we can be?

LazySP
(Lazy Shortest Path)

Dellin and Srinivasa, 2016

First Provably Edge-Optimal A*-like Search Algorithm
LazySP

Greedy Best-first Search over Paths

To find the shortest path, eliminate all shorter paths!
Lazy A* as BFS

<table>
<thead>
<tr>
<th>Element</th>
<th>Ranking value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element A</td>
<td>f(A)</td>
</tr>
<tr>
<td>Element B</td>
<td>f(B)</td>
</tr>
</tbody>
</table>

POP best element

Process Element

Collision Check Path

Done?

N

Y

Update Queue

Remove bad edges

Implicit queue over paths
LazySP

Optimism Under Uncertainty

Diagram:
- Graph, start, goal, lazy estimates
  - Lazy search for shortest path
  - Update the graph
  - Evaluate Path
    - Collision
    - Free
LazySP

Optimism Under Uncertainty

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

Graph, start, goal, lazy estimates

$p$

$p$

$p$
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

P

P
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

P

P
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

P

P

P
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

P

P

P
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Evaluate Path

Collision

Free

Graph, start, goal, lazy estimates
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

P

P
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Evaluate Path

Collision

Free

P

P
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Evaluate Path

Collision

Free

P

P

P
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

P

P
LazySP

Optimism Under Uncertainty

Lazy search for shortest path
Evaluate Path
Update the graph

Collision
Free

Graph, start, goal, lazy estimates
Comparison across environments

Number of Edges Evaluated

A*

LazySP
Comparison across environments

Number of Edges Evaluated

A*

LazySP
Comparison across environments

Number of Edges Evaluated

A*

LazySP
Comparison across environments

<table>
<thead>
<tr>
<th>Number of Edges Evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong>&lt;sup&gt;+&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>LazySP</strong></td>
</tr>
</tbody>
</table>

---

**Diagram:**

- **A**<sup>+</sup> algorithm results
- **LazySP** algorithm results

Note: The diagram visualizes the comparison across different environments for two algorithms, A<sup>+</sup> and LazySP, showing the number of edges evaluated.
Comparison across environments

Number of Edges Evaluated

A*

LazySP
Comparison across environments

Number of Edges Evaluated

A*

LazySP
Comparison across environments

Number of Edges Evaluated

A*

LazySP
Comparison across environments

Number of Edges Evaluated

A*  LazySP