Artificial Intelligence
Informed Optimal Search

Andres Mendez-Vazquez

January 23, 2019

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance

2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm

3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

What is an Heuristic? [1]

Heuristic

- It is possible to use domain-dependent knowledge to capture information about the problem

Updating Function

We have the following Cost function

$$f : V \longrightarrow \mathbb{R} \text{ with } f = g + h$$

- $h(n)$ is an estimate (lower bound) of the remaining costs from n to a goal, the *heuristic function*

Cinvestav

Updating Function

We have the following Cost function

$$f : V \longrightarrow \mathbb{R} \text{ with } f = g + h$$

Where

- V is the state space of the search

- $h(n)$ is an estimate (lower bound) of the remaining costs from n to a goal, the *heuristic function*

Updating Function

We have the following Cost function

$$f : V \longrightarrow \mathbb{R} \text{ with } f = g + h$$

Where

- V is the state space of the search

$f(u) = g(u) + h(u)$

- $g(u)$ is the weight of the (current optimal) path from s to u.
- $h(u)$ is an estimate (lower bound) of the remaining costs from u to a goal, *the heuristic function*.

Cinvestav

Updating Function

We have the following Cost function

$$f : V \longrightarrow \mathbb{R} \text{ with } f = g + h$$

Where

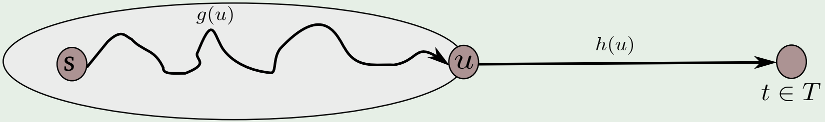
- V is the state space of the search

$f(u) = g(u) + h(u)$

- $g(u)$ is the weight of the (current optimal) path from s to u.
- $h(u)$ is an estimate (lower bound) of the remaining costs from u to a goal, ***the heuristic function***.

Graphically, we have

We have then



Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

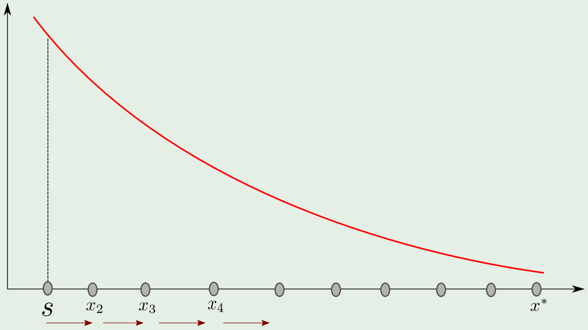
Formal Definition

Definition

- Given the weighted state space problem, $G = (V, E, s, T, w)$.
  - A **heuristic** h is a node evaluation function, mapping $h : V \to \mathbb{R}^+$.

Example quite simplified!!!

No Information



$$f(x^*) = \sum_{i=1}^{n} w(x_i, x_{i+1}) = n$$

Figure: The states are uniform no information $h(u) = 0$

Example

More Information

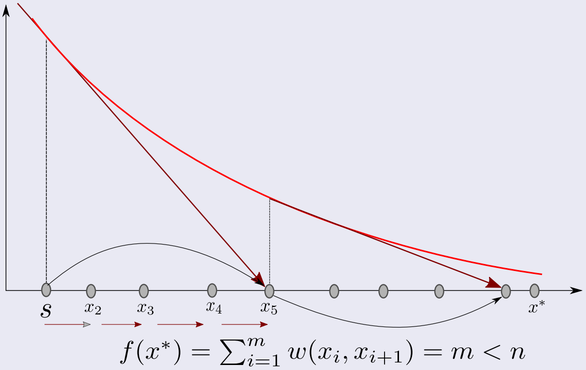


Figure: Some Information

Formal Definition

Total Information - Follow the heuristic

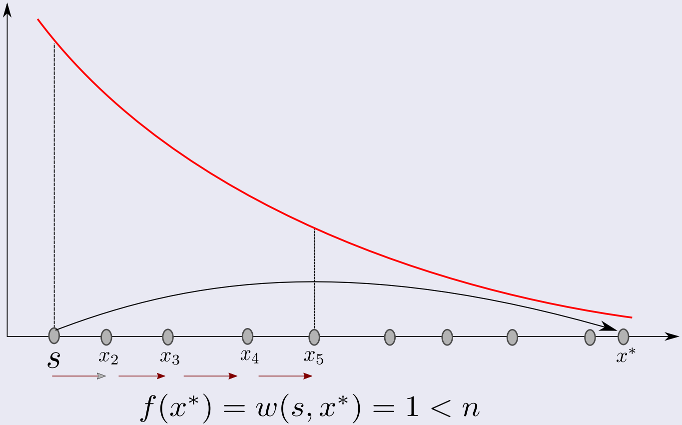


Figure: Total information

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Desirable Properties of a Heuristic

Definition 1.8

An estimate h is an **admissible** heuristic if it is a lower bound for the optimal solution costs; that is, $h(u) \leq \delta(u, T)$ for all $u \in V$.

Example

Tile Game

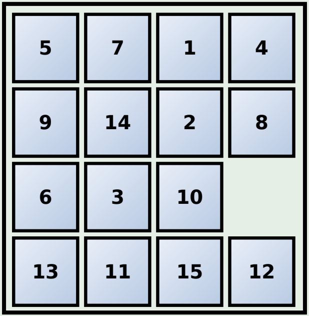


Figure: A game where the player can move tiles Up, Down, Left and Right to an empty spot

Example

Movements in the Tile Game

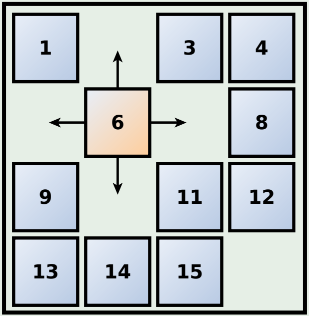


Figure: A game where the player can move tiles Up, Down, Left and Right to an empty spot

Example

Goal State of the Tile Game

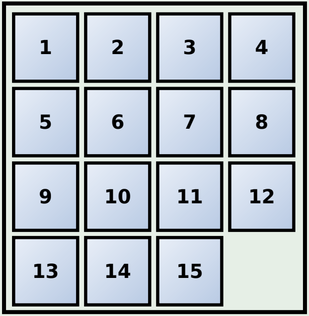


Figure: Goal State

Examples of Admissible Heuristics for the Tile Game

Hamming Distance

- The Hamming distance is the total number of misplaced tiles.

Using the Manhattan distance

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_1 = \sum_{i=1}^{n} |x_i - y_i| \quad (1)$$

With $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Examples of Admissible Heuristics for the Tile Game

Hamming Distance

- The Hamming distance is the total number of misplaced tiles.

Using the Manhattan distance

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_1 = \sum_{i=1}^{n} |x_i - y_i| \quad (1)$$

With $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Examples of Admissible Heuristics for the Tile Game

Thus, we have the following heuristic

$$h(v) = \sum_{i \in v} d\left(tile_i \text{ position}, \text{correct position of } tile_i\right) \quad (2)$$

Where

$$d\left(tile_i, \text{correct position of } tile_i\right) = \left|x_1^{(i)} - y_1^{(i)}\right| + \left|x_2^{(i)} - y_2^{(i)}\right| \quad (3)$$

With

- $tile_i$ position $= (x_1, x_2)^t \in \mathbb{N}^2$
- correct position of $tile_i = \left(y_1^{(i)}, y_2^{(i)}\right)^t \in \mathbb{N}^2$

Cinvestav

Examples of Admissible Heuristics for the Tile Game

Thus, we have the following heuristic

$$h(v) = \sum_{i \in v} d\left(tile_i \text{ position}, \text{correct position of } tile_i\right) \quad (2)$$

Where

$$d\left(tile_i, \text{correct position of } tile_i\right) = \left|x_1^{(i)} - y_1^{(i)}\right| + \left|x_2^{(i)} - y_2^{(i)}\right| \quad (3)$$

With

- $tile_i$ position $= (x_1, x_2)^t \in \mathbb{N}^2$
- correct position of $tile_i = \left(y_1^{(i)}, y_2^{(i)}\right)^t \in \mathbb{N}^2$

Cinvestav

Examples of Admissible Heuristics for the Tile Game

Thus, we have the following heuristic

$$h(v) = \sum_{i \in v} d(tile_i \text{ position}, \text{correct position of } tile_i) \tag{2}$$

Where

$$d(tile_i, \text{correct position of } tile_i) = \left| x_1^{(i)} - y_1^{(i)} \right| + \left| x_2^{(i)} - y_2^{(i)} \right| \tag{3}$$

With

- $tile_i$ position $= (x_1, x_2)^t \in \mathbb{N}^2$
- correct position of $tile_i = \left(y_1^{(i)}, y_2^{(i)} \right)^t \in \mathbb{N}^2$

Cinvestav

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Desirable Properties of a Heuristic

Definition 1.9 (Consistency and Monotonicity)

- Let $G = (V, E, s, T, w)$ be a weighted state space problem graph.

Cinvestav

Desirable Properties of a Heuristic

Definition 1.9 (Consistency and Monotonicity)

- Let $G = (V, E, s, T, w)$ be a weighted state space problem graph.
  1. A goal estimate h is a **consistent heuristic** if $h(u) \leq h(v) + w(u, v)$ for all edges $e = (u, v) \in E$.
  2. Let $(u_0, \ldots, u_k)$ be any path, $g(u_i)$ be the path cost of $(u_0, \ldots, u_i)$, and define $f(u_i) = g(u_i) + h(u_i)$.
     1. A goal estimate h is a **monotone heuristic** if $f(u_i) \leq f(u_j)$ for all $i < j$, $0 \leq i, j \leq k$.

Cinvestav

Desirable Properties of a Heuristic

Definition 1.9 (Consistency and Monotonicity)

- Let $G = (V, E, s, T, w)$ be a weighted state space problem graph.
  1. A goal estimate h is a **consistent heuristic** if $h(u) \leq h(v) + w(u, v)$ for all edges $e = (u, v) \in E$.
  2. Let $(u_0, ..., u_k)$ be any path, $g(u_i)$ be the path cost of $(u_0, ..., u_i)$, and define $f(u_i) = g(u_i) + h(u_i)$.
  3. A goal estimate h is a **monotone heuristic** if $f(u_i) \leq f(u_j)$ for all $i < j$, $0 \leq i, j \leq k$.

Desirable Properties of a Heuristic

Definition 1.9 (Consistency and Monotonicity)

- Let $G = (V, E, s, T, w)$ be a weighted state space problem graph.
  1. A goal estimate h is a **consistent heuristic** if $h(u) \leq h(v) + w(u, v)$ for all edges $e = (u, v) \in E$.
  2. Let $(u_0, ..., u_k)$ be any path, $g(u_i)$ be the path cost of $(u_0, ..., u_i)$, and define $f(u_i) = g(u_i) + h(u_i)$.
     1. A goal estimate h is a **monotone heuristic** if $f(u_i) \leq f(u_j)$ for all $i < j$, $0 \leq i, j \leq k$.

Cinvestav

Equivalence between Consistency and Monotonicity

Theorem 1.1 (Equivalence between Consistency and Monotonicity)

- A heuristic is consistent if and only if it is monotone.

Proof

- For two subsequent states u_{i-1} and u_i on a path $(u_0, u_1, \ldots, u_k)$

Equivalence between Consistency and Monotonicity

Theorem 1.1 (Equivalence between Consistency and Monotonicity)

- A heuristic is consistent if and only if it is monotone.

Proof

- For two subsequent states u_{i-1} and u_i on a path $(u_0, u_1, ..., u_k)$

Cinvestav

Proof

We have

$$
\begin{aligned}
f(u_i) &= g(u_i) + h(u_i) \\
&= g(u_{i-1}) + w(u_{i-1}, u_i) + h(u_i) \\
&\geq g(u_{i-1}) + h(u_{i-1}) \\
&= f(u_{i-1})
\end{aligned}
$$

Proof

We have

$$
\begin{aligned}
f\left(u_i\right) &= g\left(u_i\right) + h\left(u_i\right) \\
&= g\left(u_{i-1}\right) + w\left(u_{i-1}, u_i\right) + h\left(u_i\right) \\
&\geq g\left(u_{i-1}\right) + h\left(u_{i-1}\right) \\
&= f\left(u_{i-1}\right)
\end{aligned}
$$

Proof

We have

$$
\begin{aligned}
f(u_i) &= g(u_i) + h(u_i) \\
&= g(u_{i-1}) + w(u_{i-1}, u_i) + h(u_i) \\
&\geq g(u_{i-1}) + h(u_{i-1}) \\
&= f(u_{i-1})
\end{aligned}
$$

Proof

We have

$$\begin{aligned} f(u_i) &= g(u_i) + h(u_i) \\ &= g(u_{i-1}) + w(u_{i-1}, u_i) + h(u_i) \\ &\geq g(u_{i-1}) + h(u_{i-1}) \\ &= f(u_{i-1}) \end{aligned}$$

Consistent Estimates are Admissible

Theorem 1.2 (Consistency and Admissibility)

Consistent **estimates are admissible.**

Consistent Estimates are Admissible

Theorem 1.2 (Consistency and Admissibility)

Consistent **estimates are admissible.**

Proof

- if h is consistent we have that $h(u) - h(v) \leq w(u, v)$ for all $(u, v) \in E$
- Let $p = (v_0, ..., v_k)$ be any path from $u = v_0$ to $t = v_k$

Consistent Estimates are Admissible

Theorem 1.2 (Consistency and Admissibility)

Consistent **estimates are admissible.**

Proof

- if h is consistent we have that $h(u) - h(v) \leq w(u,v)$ for all $(u,v) \in E$
- Let $p = (v_0, ..., v_k)$ be any path from $u = v_0$ to $t = v_k$

Proof

We have

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

$$\geq \sum_{i=1}^{k-1} \left(h(v_i) - h(v_{i+1})\right)$$

$$= h(u) - h(v)$$

$$= h(u)$$

Proof

We have

$$
\begin{aligned}
w(p) &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) \\
&\geq \sum_{i=1}^{k-1} \left(h(v_i) - h(v_{i+1})\right) \\
&= h(u) - h(v) \\
&= h(u)
\end{aligned}
$$

Proof

We have

$$
\begin{aligned}
w(p) &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) \\
&\geq \sum_{i=1}^{k-1} \left(h(v_i) - h(v_{i+1})\right) \\
&= h(u) - h(v) \\
&= h(u)
\end{aligned}
$$

Proof

We have

$$\begin{aligned} w(p) &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) \\ &\geq \sum_{i=1}^{k-1} \left(h(v_i) - h(v_{i+1})\right) \\ &= h(u) - h(v) \\ &= h(u) \end{aligned}$$

Proof

This is also true in the important case of p being optimal

$$h(u) \leq \delta(u, T) \tag{4}$$

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Dominance

In Heuristics

Given h_1, h_2 admissible heuristics. If $h_1(n) \leq h_2(n)$, then h_2 dominates h_1.

Given that we want

h an **admissible** heuristic such that $h(u) \leq \delta(u, T)$ for all $u \in V$.

Dominance

In Heuristics

Given h_1, h_2 admissible heuristics. If $h_1(n) \leq h_2(n)$, then h_2 dominates h_1.

Given that we want

h an **admissible** heuristic such that $h(u) \leq \delta(u, T)$ for all $u \in V$.

Better Lower Approximation

Thus

Given the dominance and admissibility:

$$h_1(n) \leq h_2(n) \leq \delta(u, T) \quad (5)$$

Therefore

We have a better approximation to the real solution using the heuristic h_2 than h_1 .

Drawback

This has a problem!!! If the problem is NP-Complete!!!

- Thus, the calculation of h_2 may be more expansive than the calculation of h_1.

Better Lower Approximation

Thus

Given the dominance and admissibility:

$$h_1(n) \leq h_2(n) \leq \delta(u, T) \tag{5}$$

Therefore

We have a better approximation to the real solution using the heuristic h_2 than h_1 .

Drawback

This has a problem!!! If the problem is NP-Complete!!!

- Thus, the calculation of h_2 may be more expansive than the calculation of h_1.

Better Lower Approximation

Thus

Given the dominance and admissibility:

$$h_1(n) \leq h_2(n) \leq \delta(u, T) \tag{5}$$

Therefore

We have a better approximation to the real solution using the heuristic h_2 than h_1 .

Drawback

This has a problem!!! If the problem is NP-Complete!!!

- Thus, the calculation of h_2 may be more expansive than the calculation of h_1.

Cinvestav

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

The most prominent heuristic search algorithm is A*.

This algorithm uses the estimate

$$f(u) = g(u) + h(u) \tag{6}$$

That requires

- A way to keep a priority!!

Thus

1. *Open* a **MIN priority queue**.
2. *Closed* is a set

Cinvestav

The most prominent heuristic search algorithm is A*.

This algorithm uses the estimate

$$f(u) = g(u) + h(u) \tag{6}$$

That requires

- A way to keep a priority!!

Thus

1. *Open* a **MIN priority queue**.
2. *Closed* is a set

The most prominent heuristic search algorithm is A*.

This algorithm uses the estimate

$$f(u) = g(u) + h(u) \tag{6}$$

That requires

- A way to keep a priority!!

Thus

1. $Open$ a **MIN priority queue**.
2. $Closed$ is a set

Outline

1 Informed Optimal Search
- What is an Heuristic?
- Formal Definition of a Heuristic
- Desirable Properties of a Heuristic
  - Consistency and Monotonicity
  - Dominance

2 A* Algorithm
- The Heuristic A*
- Pseudo-Code
- Consistency of A*
- Admissibility in A*
- Lemma Toward Admissibility of A*
- Expansion of Different Strategies
- Optimality in A*
- Iterative-Deepening for A*
- A*: Re-weighting Edges
- Dealing with the problem
- Best-First Searches
  - Algorithm
- Greedy Best First Search
- Greedy Best-First Search Vs. A* Algorithm

3 Limits in Heuristics
- Where do heuristics come from?
- Abstraction Transformations and Valtortas's Theorem

Pseudo-Code

Procedure A*

Input: Implicit graph with start node s, weight function w, heuristic h, function $Expand$ and Predicate $Goal$

Output: Optimal path from s to $t \in T$, or $\emptyset$.

1. $Closed = \emptyset$
2. **Insert**$(Open, s)$
3. $f(s) = h(s)$
4. **while** $(Open \neq \emptyset)$
5. $\quad u =$ **remove MIN**$_{f(u)}(Open)$
6. $\quad Closed = Closed \cup \{u\}$
7. $\quad$**if** $(Goal(u))$ **return** $Path(u)$
8. $\quad$**else** $Succ(u) = Expand(u)$
9. $\quad\quad$**for each** v **in** $Succ(u)$
10. $\quad\quad\quad Improve(u, v)$
11. **return** $\emptyset$

Pseudo-Code

Procedure A*

Input: Implicit graph with start node s, weight function w, heuristic h, function $Expand$ and Predicate $Goal$

Output: Optimal path from s to $t \in T$, or $\emptyset$.

1. $Closed = \emptyset$
2. **Insert**$(Open, s)$
3. $f(s) = h(s)$
4. **while** $(Open \neq \emptyset)$
5. $\quad u$ **= remove MIN**$_{f(u)}(Open)$
6. $\quad Closed = Closed \cup \{u\}$
7. $\quad$ **if** $(Goal(u))$ **return** $Path(u)$
8. $\quad$ **else** $Succ(u) = Expand(u)$
9. $\qquad$ **for each** v **in** $Succ(u)$
10. $\qquad\quad Improve(u, v)$
11. **return** $\emptyset$

Pseudo-Code

Procedure A*

Input: Implicit graph with start node s, weight function w, heuristic h, function $Expand$ and Predicate $Goal$

Output: Optimal path from s to $t \in T$, or $\emptyset$.

$Closed = \emptyset$

Insert$(Open, s)$

$f(s) = h(s)$

while $(Open \neq \emptyset)$

  u **= remove MIN**$_{f(u)}$ $(Open)$

  $Closed = Closed \cup \{u\}$

  if $(Goal(u))$ **return** $Path(u)$

  else $Succ(u) = Expand(u)$

    for each v **in** $Succ(u)$

      $Improve(u, v)$

return $\emptyset$

Pseudo-Code

Procedure A*

Input: Implicit graph with start node s, weight function w, heuristic h, function $Expand$ and Predicate $Goal$

Output: Optimal path from s to $t \in T$, or $\emptyset$.

```
Closed = ∅
Insert(Open, s)
f(s) = h(s)
while (Open ≠ ∅)
        u = remove MIN_{f(u)} (Open)
        Closed = Closed ∪ {u}
        if (Goal(u)) return Path(u)
        else Succ(u) = Expand(u)
                for each v in Succ(u)
                        Improve(u, v)
return ∅
```

Procedure Improve

Procedure Improve

Input: Node u and v, v successor of u

Effects: Update parent of v, $f(v)$, $Open$ and $Closed$

1. **if** $v \in Open$ $\Rightarrow$**Node generated but not expanded**
2. **if** $(g(u,v) + w(u,v) < g(v))$
3. $parent(v) = u$
4. $f(v) = g(u) + w(u,v) + h(v)$
5. else if $v \in Closed$ $\Rightarrow$Node already expanded
6. if $(g(u,v) + w(u,v) < g(v))$
7. $parent(v) = u$
8. $f(v) = g(u) + w(u,v) + h(v)$
9. $Closed = Closed - \{v\}$
10. Insert$(Open, v)$
11. else $\Rightarrow$Node not seen before
12. $parent(v) = u$
13. Initialize $f(v) = g(u) + w(u,v) + h(v)$
14. Insert$(Open, v)$ with $f(v)$

Procedure Improve

Procedure Improve

Input: Node u and v, v successor of u

Effects: Update parent of v, $f(v)$, $Open$ and $Closed$

1. **if** $v \in Open$ ⇒**Node generated but not expanded**
2. **if** $(g(u,v) + w(u,v) < g(v))$
3. $parent(v) = u$
4. $f(v) = g(u) + w(u,v) + h(v)$
5. **else if** $v \in Closed$ ⇒**Node already expanded**
6. **if** $(g(u,v) + w(u,v) < g(v))$
7. $parent(v) = u$
8. $f(v) = g(u) + w(u,v) + h(v)$
9. $Closed = Closed - \{v\}$
10. **Insert**$(Open, v)$
11. else ⇒Node not seen before
12. $parent(v) = u$
13. Initialize $f(v) = g(u) + w(u,v) + h(v)$
14. Insert$(Open, v)$ with $f(v)$

Procedure Improve

Procedure Improve

Input: Node u and v, v successor of u

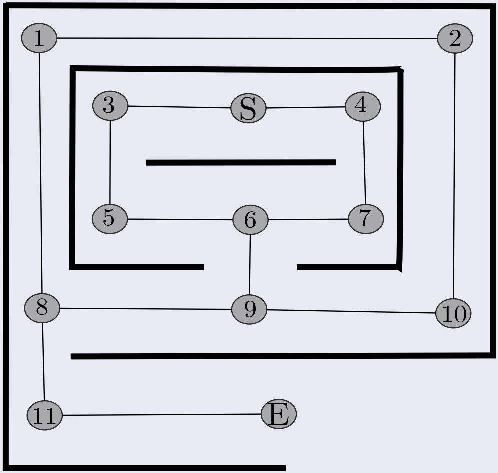
Effects: Update parent of v, $f(v)$, $Open$ and $Closed$

- **if** $v \in Open$ $\Rightarrow$**Node generated but not expanded**
  - **if** $(g(u,v) + w(u,v) < g(v))$
    - $parent(v) = u$
    - $f(v) = g(u) + w(u,v) + h(v)$
- **else if** $v \in Closed$ $\Rightarrow$**Node already expanded**
  - **if** $(g(u,v) + w(u,v) < g(v))$
    - $parent(v) = u$
    - $f(v) = g(u) + w(u,v) + h(v)$
    - $Closed = Closed - \{v\}$
    - **Insert**$(Open, v)$
- **else** $\Rightarrow$**Node not seen before**
  - $parent(v) = u$
  - **Initialize** $f(v) = g(u) + w(u,v) + h(v)$
  - **Insert**$(Open, v)$ **with** $f(v)$

Cinvestav

A* Example

We can use our previous example



Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance

2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm

3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Thus!!! We like consistency in A*

Theorem 2.9 (A* for Consistent Heuristics)

- Let h be consistent. If we set $f(s) = h(s)$ for the initial node s and update $f(v)$ with $f(u) + \widehat{w}(u, v)$, where $\widehat{w}(u, v) = h(v) - h(u) + w(u, v)$, instead of $f(u) + w(u, v)$, at each time a node $t \in T$ is selected, we have $f(t) = \delta(s, t)$.

Proof

First h is consistent

The, we have that $h(u) \leq h(v) + w(u, v)$

Therefore

We have the difference

$$\widehat{w}(u, v) = w(u, v) + h(v) - h(u) \geq 0 \quad (7)$$

Thus, given the

Given the recasting of A* as Disjkstra's Algorithm with weights $\widehat{w}(u, v) \geq 0$.

Proof

First h is consistent

The, we have that $h(u) \leq h(v) + w(u,v)$

Therefore

We have the difference

$$\widehat{w}(u,v) = w(u,v) + h(v) - h(u) \geq 0 \tag{7}$$

Thus, given the

Given the recasting of A* as Disjkstra's Algorithm with weights $\widehat{w}(u,v) \geq 0$.

Proof

First h is consistent

The, we have that $h(u) \leq h(v) + w(u,v)$

Therefore

We have the difference

$$\widehat{w}(u,v) = w(u,v) + h(v) - h(u) \geq 0 \quad (7)$$

Thus, given the

Given the recasting of A* as Disjkstra's Algorithm with weights $\widehat{w}(u,v) \geq 0$.

Cinvestav

Proof

We have that for a shortest path $\langle s = p_0, p_1, ..., u = p_n \rangle$ under $\widehat{w}$ with

$$f(p_1) = \widehat{w}(p_0, p_1) + h(s) \tag{8}$$

Thus

$$f(p_n) = \underbrace{\widehat{w}(p_n, p_{n-1}) + ... + \widehat{w}(p_2, p_1) + \widehat{w}(p_1, p_0)}_{\widehat{\delta}(s,u)} + h(s) \tag{9}$$

Given that once the shortest path is achieved, it does not change (Lemma 2.3)

$$f(u) = \widehat{\delta}(s, u) + h(s) \tag{10}$$

Cinvestav

Proof

We have that for a shortest path $\langle s = p_0, p_1, ..., u = p_n \rangle$ under $\widehat{w}$ with

$$f\left(p_1\right) = \widehat{w}(p_0, p_1) + h\left(s\right) \tag{8}$$

Thus

$$f\left(p_n\right) = \underbrace{\widehat{w}\left(p_n, p_{n-1}\right) + ... + \widehat{w}\left(p_2, p_1\right) + \widehat{w}\left(p_1, p_0\right)}_{\widehat{\delta}(s,u)} + h\left(s\right) \tag{9}$$

Given that once the shortest path is achieved, it does not change (Lemma 2.3)

$$f\left(u\right) = \widehat{\delta}\left(s, u\right) + h\left(s\right) \tag{10}$$

Proof

We have that for a shortest path $\langle s = p_0, p_1, ..., u = p_n \rangle$ under $\widehat{w}$ with

$$f(p_1) = \widehat{w}(p_0, p_1) + h(s) \quad (8)$$

Thus

$$f(p_n) = \underbrace{\widehat{w}(p_n, p_{n-1}) + ... + \widehat{w}(p_2, p_1) + \widehat{w}(p_1, p_0)}_{\widehat{\delta}(s,u)} + h(s) \quad (9)$$

Given that once the shortest path is achieved, it does not change (Lemma 2.3)

$$f(u) = \widehat{\delta}(s, u) + h(s) \quad (10)$$

Cinvestav

Proof

Hence, if $t \in T$ is selected from $Open$ and $\langle s = p_0, p_1, ..., t = p_n \rangle$

$$
\begin{aligned}
f(t) &= \widehat{\delta}(s,t) + h(s) \\
&= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) + h(s) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right] + h(s) \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) + h(t) - h(s) + h(s) \quad \text{(Telescopic Sum)} \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) \quad (h(t) = 0) \\
&= \delta(s,t)
\end{aligned}
$$

Proof

Hence, if $t \in T$ is selected from $Open$ and $\langle s = p_0, p_1, ..., t = p_n \rangle$

$$
\begin{aligned}
f(t) &= \widehat{\delta}(s,t) + h(s) \\
&= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) + h(s) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1}) \right] + h(s) \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) + h(t) - h(s) + h(s) \quad \text{(Telescopic Sum)} \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) \quad (h(t) = 0) \\
&= \delta(s,t)
\end{aligned}
$$

Proof

Hence, if $t \in T$ is selected from $Open$ and $\langle s = p_0, p_1, ..., t = p_n \rangle$

$$
\begin{aligned}
f(t) &= \widehat{\delta}(s,t) + h(s) \\
&= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) + h(s) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right] + h(s) \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) + h(t) - h(s) + h(s) \quad \text{(Telescopic Sum)} \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) \quad (h(t) = 0) \\
&= \delta(s,t)
\end{aligned}
$$

Proof

Hence, if $t \in T$ is selected from $Open$ and $\langle s = p_0, p_1, ..., t = p_n \rangle$

$$
\begin{aligned}
f(t) &= \widehat{\delta}(s,t) + h(s) \\
&= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) + h(s) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right] + h(s) \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) + h(t) - h(s) + h(s) \quad \text{(Telescopic Sum)} \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) \quad (h(t) = 0) \\
&= \delta(s,t)
\end{aligned}
$$

Proof

Hence, if $t \in T$ is selected from $Open$ and $\langle s = p_0, p_1, ..., t = p_n \rangle$

$$
\begin{aligned}
f(t) &= \widehat{\delta}(s,t) + h(s) \\
&= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) + h(s) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1}) \right] + h(s) \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) + h(t) - h(s) + h(s) \quad \text{(Telescopic Sum)} \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) \quad (h(t) = 0) \\
&= \delta(s,t)
\end{aligned}
$$

Proof

Hence, if $t \in T$ is selected from $Open$ and $\langle s = p_0, p_1, ..., t = p_n \rangle$

$$
\begin{aligned}
f(t) &= \widehat{\delta}(s,t) + h(s) \\
&= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) + h(s) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right] + h(s) \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) + h(t) - h(s) + h(s) \;\; \text{(Telescopic Sum)} \\
&= \sum_{i=1}^{n} w(p_{i-1}, p_n) \;\; (h(t) = 0) \\
&= \delta(s,t)
\end{aligned}
$$

Finally

Since

- $\widehat{w} \geq 0$, we have $f(v) \geq f(u)$ for all successors v of u.

Given that we take a less restrictive condition for a graph with negative weights

$$\delta(u, T) = \min\{\delta(u, t) \mid t \in T\} \geq 0 \ \forall u \quad (11)$$

Then

- The $f-$values increases monotonically so that at the first extraction of $t \in T$:

$$\delta(s, t) = \delta(s, T). \quad (12)$$

Finally

Since

- $\widehat{w} \geq 0$, we have $f(v) \geq f(u)$ for all successors v of u.

Given that we take a less restrictive condition for a graph with negative weights

$$\delta(u, T) = \min\{\delta(u, t) \,|\, t \in T\} \geq 0 \;\forall u \tag{11}$$

Then

- The $f-$values increases monotonically so that at the first extraction of $t \in T$:

$$\delta(s, t) = \delta(s, T). \tag{12}$$

Cinvestav

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - **Admissibility in A***
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Remember!!!

Lemma 2.3

- Let G be a weighted problem graph and $h : V \to \mathbb{R}$. Define the modified weight $\widehat{w}(u,v)$ as

$$\widehat{w}(u,v) = w(u,v) - h(u) + h(v) \tag{13}$$

Let $\delta(s,t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s,t)$ be the corresponding in the reweighed graph.

1. For a path p, we have $w(p) = \delta(s,t)$ if and only if $\widehat{w}(u,v) = \widehat{\delta}(s,t)$.
2. In addition, G has no negatively weighted cycles with respect to w if and only if it has none with respect $\widehat{w}$.

Remember!!!

Lemma 2.3

- Let G be a weighted problem graph and $h : V \rightarrow \mathbb{R}$. Define the modified weight $\widehat{w}(u, v)$ as

$$\widehat{w}(u, v) = w(u, v) - h(u) + h(v) \tag{13}$$

Let $\delta(s, t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s, t)$ be the corresponding in the reweighed graph.

1. For a path p, we have $w(p) = \delta(s, t)$ if and only if $\widehat{w}(u, v) = \widehat{\delta}(s, t)$.
2. In addition, G has no negatively weighted cycles with respect to w if and only if it has none with respect $\widehat{w}$.

Proof

It can be found at the Johnson's Algorithm part in "All-Pairs Shortest Path."

Remember!!!

Lemma 2.3

- Let G be a weighted problem graph and $h : V \to \mathbb{R}$. Define the modified weight $\widehat{w}(u, v)$ as

$$\widehat{w}(u, v) = w(u, v) - h(u) + h(v) \quad (13)$$

Let $\delta(s, t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s, t)$ be the corresponding in the reweighed graph.

1. For a path p, we have $w(p) = \delta(s, t)$ if and only if $\widehat{w}(u, v) = \widehat{\delta}(s, t)$.
2. In addition, G has no negatively weighted cycles with respect to w if and only if it has none with respect $\widehat{w}$.

Proof

It can be found at the Johnson's Algorithm part in "All-Pairs Shortest Path."

Remember!!!

Lemma 2.3

- Let G be a weighted problem graph and $h : V \to \mathbb{R}$. Define the modified weight $\widehat{w}(u, v)$ as

$$\widehat{w}(u, v) = w(u, v) - h(u) + h(v) \quad (13)$$

Let $\delta(s, t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s, t)$ be the corresponding in the reweighed graph.

1. For a path p, we have $w(p) = \delta(s, t)$ if and only if $\widehat{w}(u, v) = \widehat{\delta}(s, t)$.
2. In addition, G has no negatively weighted cycles with respect to w if and only if it has none with respect $\widehat{w}$.

Proof

It can be found at the Johnson's Algorithm part in "All-Pairs Shortest Path."

Remember!!!

Lemma 2.3

- Let G be a weighted problem graph and $h : V \to \mathbb{R}$. Define the modified weight $\widehat{w}(u,v)$ as

$$\widehat{w}(u,v) = w(u,v) - h(u) + h(v) \tag{13}$$

Let $\delta(s,t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s,t)$ be the corresponding in the reweighed graph.

1. For a path p, we have $w(p) = \delta(s,t)$ if and only if $\widehat{w}(u,v) = \widehat{\delta}(s,t)$.
2. In addition, G has no negatively weighted cycles with respect to w if and only if it has none with respect $\widehat{w}$.

Proof

It can be found at the Johnson's Algorithm part in "All-Pairs Shortest Path."

Lemma

Lemma 2.4 - Toward Admissibility

- Let G be a weighted problem graph, h be a heuristic, and $\widehat{w}(u,v) = h(v) - h(u) + w(u,v)$. If h is admissible, then $\widehat{\delta}(u,T) \geq 0$.

Proof

- Since $h(t) = 0$ and the shortest path costs remains invariant under re-weighting of G by Lemma 2.3, we have...

Lemma

Lemma 2.4 - Toward Admissibility

- Let G be a weighted problem graph, h be a heuristic, and $\widehat{w}(u,v) = h(v) - h(u) + w(u,v)$. If h is admissible, then $\widehat{\delta}(u,T) \geq 0$.

Proof

- Since $h(t) = 0$ and the shortest path costs remains invariant under re-weighting of G by Lemma 2.3, we have...

Cinvestav

Proof

By the definition of $\widehat{\delta}(u,T)$

$$
\begin{aligned}
\widehat{\delta}(u,T) &= \min\left\{\widehat{\delta}(u,t)\,|t\in T\right\}\\
&= \min\left\{\delta(u,t) - h(u) + h(t)\,|t\in T\right\}
\end{aligned}
$$

Proof

By the definition of $\widehat{\delta}(u, T)$

$$\begin{aligned}\widehat{\delta}(u, T) &= \min\left\{\widehat{\delta}(u, t) \,|t \in T\right\}\\ &= \min\left\{\delta(u, t) - h(u) + h(t) \,|t \in T\right\}\end{aligned}$$

Because the telescopic sum

$$\begin{aligned}\widehat{\delta}(u, t) &= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1})\\ &= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right]\\ &= \delta(u, t) + h(t) - h(u)\end{aligned}$$

Proof

By the definition of $\widehat{\delta}(u,T)$

$$\widehat{\delta}(u,T) = \min\left\{\widehat{\delta}(u,t) \mid t \in T\right\} = \min\left\{\delta(u,t) - h(u) + h(t) \mid t \in T\right\}$$

Because the telescopic sum

$$\widehat{\delta}(u,t) = \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1})$$

$$= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right]$$

$$= \delta(u,t) + h(t) - h(u)$$

Proof

By the definition of $\widehat{\delta}(u,T)$

$$
\begin{aligned}
\widehat{\delta}(u,T) &= \min\left\{\widehat{\delta}(u,t)\,|t \in T\right\} \\
&= \min\left\{\delta(u,t) - h(u) + h(t)\,|t \in T\right\}
\end{aligned}
$$

Because the telescopic sum

$$
\begin{aligned}
\widehat{\delta}(u,t) &= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1}) \\
&= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n} \left[h(p_i) - h(p_{i-1})\right] \\
&= \delta(u,t) + h(t) - h(u)
\end{aligned}
$$

Proof

By the definition of $\widehat{\delta}(u, T)$

$$\begin{aligned}\widehat{\delta}(u, T) &= \min\left\{\widehat{\delta}(u, t) \,|t \in T\right\}\\ &= \min\left\{\delta(u, t) - h(u) + h(t) \,|t \in T\right\}\end{aligned}$$

Because the telescopic sum

$$\begin{aligned}\widehat{\delta}(u, t) &= \sum_{i=1}^{n} \widehat{w}(p_i, p_{i-1})\\ &= \sum_{i=1}^{n} w(p_i, p_{i-1}) + \sum_{i=1}^{n}\left[h(p_i) - h(p_{i-1})\right]\\ &= \delta(u, t) + h(t) - h(u)\end{aligned}$$

Proof

Therefore

$$\widehat{\delta}(u,T) = \min\{\delta(u,t) - h(u) \,|\, t \in T\}$$
$$= \min\{\delta(u,t) \,|\, t \in T\} - h(u)$$
$$= \delta(u,T) - h(u) \geq 0 \text{ Q.E.D.}$$

Cinvestav

Proof

Therefore

$$\widehat{\delta}(u,T) = \min\{\delta(u,t) - h(u) \,|\, t \in T\}$$
$$= \min\{\delta(u,t) \,|\, t \in T\} - h(u)$$
$$= \delta(u,T) - h(u) \geq 0 \text{ Q.E.D.}$$

Proof

Therefore

$$\widehat{\delta}(u,T) = \min\{\delta(u,t) - h(u)\,|t \in T\}$$
$$= \min\{\delta(u,t)\,|t \in T\} - h(u)$$
$$= \delta(u,T) - h(u) \geq 0 \text{ Q.E.D.}$$

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Now

Something Notable

- Given a graph with non-negative weights we have that Dijkstra's algorithms is optimal (Theorem 2.1).

Now

Something Notable

- Given a graph with non-negative weights we have that Dijkstra's algorithms is optimal (Theorem 2.1).

But in negative weighted graphs

- Negatively weighted graphs may contain negatively weighted cycles!!!
  - Thus, we can handle this situation by using the Bellman-Ford Algorithm.

- Look at page 57 Edelkamp

- And a Lemma about the Invariance of the Extended Dijkstra

Now

Something Notable

- Given a graph with non-negative weights we have that Dijkstra's algorithms is optimal (Theorem 2.1).

But in negative weighted graphs

- Negatively weighted graphs may contain negatively weighted cycles!!!
  - Thus, we can handle this situation by using the Bellman-Ford Algorithm.

But we can use a less restrictive condition

1. To define a new Improve for the new Extended Dijkstra.
   - Look at page 57 Edelkamp
2. And a Lemma about the Invariance of the Extended Dijkstra.

Now

Something Notable

- Given a graph with non-negative weights we have that Dijkstra's algorithms is optimal (Theorem 2.1).

But in negative weighted graphs

- Negatively weighted graphs may contain negatively weighted cycles!!!
  - Thus, we can handle this situation by using the Bellman-Ford Algorithm.

But we can use a less restrictive condition

1. To define a new Improve for the new Extended Dijkstra.
   1. Look at page 57 Edelkamp
2. And a Lemma about the Invariance of the Extended Dijkstra.

Now

Something Notable

- Given a graph with non-negative weights we have that Dijkstra's algorithms is optimal (Theorem 2.1).

But in negative weighted graphs

- Negatively weighted graphs may contain negatively weighted cycles!!!
    - Thus, we can handle this situation by using the Bellman-Ford Algorithm.

But we can use a less restrictive condition

1. To define a new Improve for the new Extended Dijkstra.
    1. Look at page 57 Edelkamp
2. And a Lemma about the Invariance of the Extended Dijkstra.

Now

Something Notable

- Given a graph with non-negative weights we have that Dijkstra's algorithms is optimal (Theorem 2.1).

But in negative weighted graphs

- Negatively weighted graphs may contain negatively weighted cycles!!!
  - Thus, we can handle this situation by using the Bellman-Ford Algorithm.

But we can use a less restrictive condition

1. To define a new Improve for the new Extended Dijkstra.
   1. Look at page 57 Edelkamp
2. And a Lemma about the Invariance of the Extended Dijkstra.

New Improved Algorithm

Improved Algorithm

Input: Nodes u and v, v successor of u

Side effects: Update parent of v, $f(v)$, $Open$, and $Closed$.

```
if (v ∈ Open)
        if (f(u) + w(u,v) < f(v)) ◁ Shorter Path
                parent(v) ← u and f(v) ← f(u) + w(u,v)
elseif (v ∈ Closed)
        if (f(u) + w(u,v) < f(v))
                parent(v) ← u and f(v) ← f(u) + w(u,v)
                Remove v from Closed and Insert it into Open with f(v)
else
        parent(v) ← u and Init f(v) ← f(u) + w(u,v)
        Insert v into Open with f(v)
```

New Improved Algorithm

Improved Algorithm

Input: Nodes u and v, v successor of u

Side effects: Update parent of v, $f(v)$, $Open$, and $Closed$.

1. **if** $(v \in Open)$
2. **if** $(f(u) + w(u,v) < f(v))$ $\triangleleft$ **Shorter Path**
3. $parent(v) \leftarrow u$ **and** $f(v) \leftarrow f(u) + w(u,v)$
4. **elseif** $(v \in Closed)$
5. **if** $(f(u) + w(u,v) < f(v))$
6. $parent(v) \leftarrow u$ **and** $f(v) \leftarrow f(u) + w(u,v)$
7. **Remove** v **from Closed and Insert it into** $Open$ **with** $f(v)$
8. **else**
9. $parent(v) \leftarrow u$ **and Init** $f(v) \leftarrow f(u) + w(u,v)$
10. **Insert** v **into** $Open$ **with** $f(v)$

New Improved Algorithm

Improved Algorithm

Input: Nodes u and v, v successor of u

Side effects: Update parent of v, $f(v)$, $Open$, and $Closed$.

1. **if** $(v \in Open)$
2. **if** $(f(u) + w(u, v) < f(v))$ ◁ **Shorter Path**
3. $parent(v) \leftarrow u$ **and** $f(v) \leftarrow f(u) + w(u, v)$
4. **elseif** $(v \in Closed)$
5. **if** $(f(u) + w(u, v) < f(v))$
6. $parent(v) \leftarrow u$ **and** $f(v) \leftarrow f(u) + w(u, v)$
7. **Remove** v **from Closed and Insert it into** $Open$ **with** $f(v)$
8. **else**
9. $parent(v) \leftarrow u$ **and Init** $f(v) \leftarrow f(u) + w(u, v)$
10. **Insert** v **into** $Open$ **with** $f(v)$

Given

The less restrictive condition

$$\delta(u,T) = \min\{\delta(u,t) \,|\, t \in T\} \geq 0 \;\forall u \tag{14}$$

Note:

1. That is, the distance from each node to the goal is non-negative.
2. Figuratively speaking, we can have negative edges when far from the goal, but they get "eaten up" when coming closer.
3. The condition implies that no negatively weighted cycles exist.

Cinvestav

Given

The less restrictive condition

$$\delta(u,T) = \min\{\delta(u,t) \,|\, t \in T\} \geq 0 \;\forall u \tag{14}$$

Note:

1. That is, the distance from each node to the goal is non-negative.
2. Figuratively speaking, we can have negative edges when far from the goal, but they get "eaten up" when coming closer.
3. The condition implies that no negatively weighted cycles exist.

Thus, we get a more general version of the Dijkstra's Algorithm

That contains an invariance that we need to prove...

Given

The less restrictive condition

$$\delta(u, T) = \min\{\delta(u, t) \mid t \in T\} \geq 0 \ \forall u \tag{14}$$

Note:

1. That is, the distance from each node to the goal is non-negative.
2. Figuratively speaking, we can have negative edges when far from the goal, but they get “eaten up” when coming closer.
3. The condition implies that no negatively weighted cycles exist.

Thus, we get a more general version of the Dijkstra's Algorithm

That contains an invariance that we need to prove...

Given

The less restrictive condition

$$\delta\left(u,T\right)=\min\left\{\delta\left(u,t\right)|t\in T\right\}\geq 0\ \forall u \tag{14}$$

Note:

1. That is, the distance from each node to the goal is non-negative.
2. Figuratively speaking, we can have negative edges when far from the goal, but they get "eaten up" when coming closer.
3. The condition implies that no negatively weighted cycles exist.

Thus, we get a more general version of the Dijkstra's Algorithm

That contains an invariance that we need to prove...

Cinvestav

Given

The less restrictive condition

$$\delta\left(u,T\right)=\min\left\{\delta\left(u,t\right)|t\in T\right\}\geq 0\ \forall u \tag{14}$$

Note:

1. That is, the distance from each node to the goal is non-negative.
2. Figuratively speaking, we can have negative edges when far from the goal, but they get "eaten up" when coming closer.
3. The condition implies that no negatively weighted cycles exist.

Thus, we get a more general version of the Dijkstra's Algorithm

That contains an invariance that we need to prove...

Invariance for Extended Dijkstra's Algorithm

Lemma 2.2

Let $G = (V, E, w)$ be a weighted graph. $p = (s = v_0, ..., v_n = t)$ be a least cost path from the start node s to a goal node $t \in T$, and f be the approximation in the extended Dijkstra's Algorithm. At each selection of a node u from $Open$, we have the following invariance:

(I). Unless v_n is in Closed with $f(v_n) = \delta(s, v_n)$, there is a node $v_i \in Open$ such that $f(v_i) = \delta(s, v_i)$, and no $j > i$ exists such that v_j is in $Closed$ with $f(v_j) = \delta(s, v_j)$.

Cinvestav

Proof

Given that

- Without loss of generality let i be maximal among the nodes satisfying the invariance (I).
- We have two cases...

Proof

Given that

- Without loss of generality let i be maximal among the nodes satisfying the invariance (I).
- We have two cases...

Case I

- Node u is not on p or $f(u) > \delta(s, u)$
- Then, $v_i \neq u$ remains in Open.
- Since no v in $Open \cap p \cap Succ(u)$ with $f(v) = \delta(s, v) \leq f(u) + w(u, v)$ is changed and no other node is added to $Closed$
- (I) is preserved

Proof

Given that

- Without loss of generality let i be maximal among the nodes satisfying the invariance (I).
- We have two cases...

Case I

- Node u is not on p or $f(u) > \delta(s,u)$
- Then, $v_i \neq u$ remains in Open.
- Since no v in $Open \cap p \cap Succ(u)$ with $f(v) = \delta(s,v) \leq f(u) + w(u,v)$ is changed and no other node is added to $Closed$
- (I) is preserved

Proof

Given that

- Without loss of generality let i be maximal among the nodes satisfying the invariance (I).
- We have two cases...

Case I

- Node u is not on p or $f(u) > \delta(s, u)$
- Then, $v_i \neq u$ remains in Open.
- Since no v in $Open \cap p \cap Succ(u)$ with $f(v) = \delta(s, v) \leq f(u) + w(u, v)$ is changed and no other node is added to $Closed$
- (I) is preserved

Proof

Given that

- Without loss of generality let i be maximal among the nodes satisfying the invariance (I).
- We have two cases...

Case I

- Node u is not on p or $f(u) > \delta(s, u)$
- Then, $v_i \neq u$ remains in Open.
- Since no v in $Open \cap p \cap Succ(u)$ with $f(v) = \delta(s, v) \leq f(u) + w(u, v)$ is changed and no other node is added to $Closed$
- (I) is preserved

Cinvestav

Proof

Given that

- Without loss of generality let i be maximal among the nodes satisfying the invariance (I).
- We have two cases...

Case I

- Node u is not on p or $f(u) > \delta(s, u)$
- Then, $v_i \neq u$ remains in Open.
- Since no v in $Open \cap p \cap Succ(u)$ with $f(v) = \delta(s, v) \leq f(u) + w(u, v)$ is changed and no other node is added to $Closed$
- (I) is preserved

Proof

Case II

- Node u is on p and $f(u) = \delta(s, u)$. If $u = v_n$, there is nothing to show.

Now the proof, first assume $u = v_i$

- Then, Improve will be called for $v = v_{i+1} \in Succ(u)$

Then

- For all other nodes in $Succ(u) - \{v_{i+1}\}$, the argument of case 1 holds.

Proof

Case II

- Node u is on p and $f(u) = \delta(s, u)$. If $u = v_n$, there is nothing to show.

Now the proof, first assume $u = v_i$

- Then, Improve will be called for $v = v_{i+1} \in Succ(u)$

Then

- For all other nodes in $Succ(u) - \{v_{i+1}\}$, the argument of case 1 holds.

Proof

Case II

- Node u is on p and $f(u) = \delta(s, u)$. If $u = v_n$, there is nothing to show.

Now the proof, first assume $u = v_i$

- Then, Improve will be called for $v = v_{i+1} \in Succ(u)$

Then

- For all other nodes in $Succ(u) - \{v_{i+1}\}$, the argument of case 1 holds.

Proof

According to (I)

- If v is in $Closed$, then $f(v) > \delta(s,v)$ and it will be reinserted in $Open$ with $f(v) = \delta(s,u) + w(u,v) = \delta(s,v)$.

If v is not in $Open$ nor $Closed$

- It is inserted into Open with $f(v) = \delta(s,u) + w(u,v)$
- Otherwise the operation will set it to $\delta(s,u)$.

It does not matter

- The invariance holds in both cases!!!

Cinvestav

Proof

According to (I)

- If v is in $Closed$, then $f(v) > \delta(s,v)$ and it will be reinserted in $Open$ with $f(v) = \delta(s,u) + w(u,v) = \delta(s,v)$.

If v is not in $Open$ nor $Closed$

- It is inserted into Open with $f(v) = \delta(s,u) + w(u,v)$
- Otherwise the operation will set it to $\delta(s,u)$.

It does not matter

- The invariance holds in both cases!!!

Proof

According to (I)

- If v is in $Closed$, then $f(v) > \delta(s,v)$ and it will be reinserted in $Open$ with $f(v) = \delta(s,u) + w(u,v) = \delta(s,v)$.

If v is not in $Open$ nor $Closed$

- It is inserted into Open with $f(v) = \delta(s,u) + w(u,v)$
- Otherwise the operation will set it to $\delta(s,u)$.

It does not matter

- The invariance holds in both cases!!!

Proof

Now suppose $u \neq v_i$

- By the maximality of i, we have that for $k < i$ $u = v_k$

If $v = v_i$

- Any improve operation will not change the optimal value of $f(v) = \delta(s, u) + w(u, v) = \delta(s, v)$

In the other case

- v_i remains in $Open$ with an unchanged f value and no other node besides u is inserted into Closed, thus $v_i$preserves (I).

Proof

Now suppose $u \neq v_i$

- By the maximality of i, we have that for $k < i$ $u = v_k$

If $v = v_i$

- Any improve operation will not change the optimal value of $f(v) = \delta(s, u) + w(u, v) = \delta(s, v)$

In the other case

- v_i remains in $Open$ with an unchanged f value and no other node besides u is inserted into Closed, thus $v_i$preserves (I).

Proof

Now suppose $u \neq v_i$

- By the maximality of i, we have that for $k < i$ $u = v_k$

If $v = v_i$

- Any improve operation will not change the optimal value of $f(v) = \delta(s, u) + w(u, v) = \delta(s, v)$

In the other case

- v_i remains in $Open$ with an unchanged f value and no other node besides u is inserted into Closed, thus $v_i$preserves (I).

Cinvestav

From this Lemma, we get

Theorem 2.3 - Correctness of the Extended Dijkstra

- Let $G = (V, E, w)$ be a weighted graph so that for all $u \in V$ we have $\delta(u, T) \geq 0$. The Extended Dijkstra is optimal; that is, at the first extraction of a node $t \in T$ we have $f(t) = \delta(s, T)$

Cinvestav

From Algorithms

Lemma 2.4

- Let G be a weighted problem graph, h be a heuristic, and

$$\widehat{w}(u,v) = w(u,v) - h(u) + h(v)$$

If h is admissible, then $\widehat{\delta}(u,T) \geq 0$

Finally, Admissibility in A*

Theorem (A* for Admissible Heuristics)

- For weighted graphs $G = (V, E, w)$ and admissible heuristics h, algorithm A* is complete and optimal.
  - This comes from the previous Lemma and Theorem

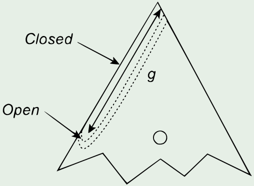
Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

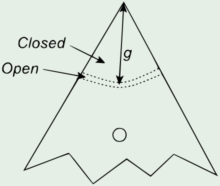
Cinvestav

Expansion of Different Strategies

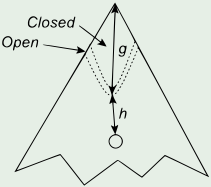
The expansion trees



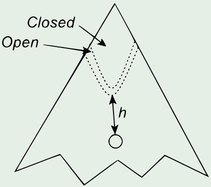
Expansion Criterion:
$u \in Open$ with max. $g(u)$
(a)



Expansion Criterion:
$u \in Open$ with min. $g(u)$
(b)



Expansion Criterion:
$u \in Open$ with min. $g(u) + h(u)$
(c)



Expansion Criterion:
$u \in Open$ with min. $h(u)$
(d)

Outline

1 Informed Optimal Search
- What is an Heuristic?
- Formal Definition of a Heuristic
- Desirable Properties of a Heuristic
  - Consistency and Monotonicity
  - Dominance

2 A* Algorithm
- The Heuristic A*
- Pseudo-Code
- Consistency of A*
- Admissibility in A*
- Lemma Toward Admissibility of A*
- Expansion of Different Strategies
- **Optimality in A***
- Iterative-Deepening for A*
- A*: Re-weighting Edges
- Dealing with the problem
- Best-First Searches
  - Algorithm
- Greedy Best First Search
- Greedy Best-First Search Vs. A* Algorithm

3 Limits in Heuristics
- Where do heuristics come from?
- Abstraction Transformations and Valtortas's Theorem

Optimality in A* - Once we have dealt with the negative edges

Theorem 2.11. (Efficiency Lower Bound)

Let G be a problem graph with nonnegative weight function, with initial node s and final node set T, and let $f^* = \delta(s, T)$ be the optimal solution cost. Any optimal algorithm has to visit all nodes $u \in V$ with $\delta(s, u) < f^*$.

Explanation

- We can view a search with a **consistent heuristic** as a search in a re-weighted problem graph with nonnegative costs!!!

Optimality in A* - Once we have dealt with the negative edges

Theorem 2.11. (Efficiency Lower Bound)

Let G be a problem graph with nonnegative weight function, with initial node s and final node set T, and let $f^* = \delta(s, T)$ be the optimal solution cost. Any optimal algorithm has to visit all nodes $u \in V$ with $\delta(s, u) < f^*$.

Explanation

- We can view a search with a **consistent heuristic** as a search in a re-weighted problem graph with nonnegative costs!!!

Cinvestav

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

PROBLEM!!!

We have a BFS style Algorithm

A* is a BFS style algorithm!!!

Improvement

We can use the iterative-deepening to improve it!!!

PROBLEM!!!

We have a BFS style Algorithm

A* is a BFS style algorithm!!!

Improvement

We can use the iterative-deepening to improve it!!!

ITERATIVE-DEEPENING A*

Procedure IDA*-Driver

Input: Start node s, function w, heuristics h, function $Expand$ and function $Goal$

Output: Path from s to $t \in T$ or $\emptyset$ if no such path exists

1. $U' \leftarrow h(s)$
2. $bestPath \leftarrow \emptyset$
3. **while** $(bestPath == \emptyset$ **and** $U' \neq \infty) \triangleleft$ **Goal not found, unexplored nodes left**
4. $\quad U \leftarrow U' \triangleleft$ **Reset Global Threshold**
5. $\quad U' \leftarrow \infty$
6. $\quad bestPath \leftarrow IDA*(s, 0, U)$
7. **return** $bestPath$

ITERATIVE-DEEPENING A*

Procedure IDA*

Input: Node u, path length g, upper bound U

Output: Shortest path to a goal node $t \in T$ or $\emptyset$ if no such path exists

SideEffects: Update of threshold U'

```
if (Goal(u)) return Path(u)
Succ(u) ← Expand(u)
for each v in Succ(u)
    if (g + w(u,v) + h(v) > U)
        if (g + w(u,v) + h(v) < U')
            U' ← g + w(u,v) + h(v)
    else
        p ← IDA*(v, g + w(u,v), U)
        if (p ≠ ∅) return (u, p)
return ∅
```

ITERATIVE-DEEPENING A*

Procedure IDA*

Input: Node u, path length g, upper bound U

Output: Shortest path to a goal node $t \in T$ or $\emptyset$ if no such path exists

SideEffects: Update of threshold U'

if $(Goal\,(u))$ **return** $Path\,(u)$
$Succ\,(u) \leftarrow Expand\,(u)$
for each v **in** $Succ\,(u)$
  if $(g + w\,(u,v) + h\,(v) > U)$
    if $(g + w\,(u,v) + h\,(v) < U')$
      $U' \leftarrow g + w\,(u,v) + h\,(v)$
  else
    $p \leftarrow IDA*\,(v, g + w\,(u,v)\,, U)$
    if $(p \neq \emptyset)$ **return** (u, p)
return $\emptyset$

ITERATIVE-DEEPENING A*

Procedure IDA*

Input: Node u, path length g, upper bound U

Output: Shortest path to a goal node $t \in T$ or $\emptyset$ if no such path exists

SideEffects: Update of threshold U'

if $(Goal(u))$ **return** $Path(u)$
$Succ(u) \leftarrow Expand(u)$
for each v **in** $Succ(u)$
    if $(g + w(u,v) + h(v) > U)$
        if $(g + w(u,v) + h(v) < U')$
            $U' \leftarrow g + w(u,v) + h(v)$
    else
        $p \leftarrow IDA*(v, g + w(u,v), U)$
        if $(p \neq \emptyset)$ **return** (u, p)
return $\emptyset$

ITERATIVE-DEEPENING A*

Procedure IDA*

Input: Node u, path length g, upper bound U

Output: Shortest path to a goal node $t \in T$ or $\emptyset$ if no such path exists

SideEffects: Update of threshold U'

if $(Goal(u))$ **return** $Path(u)$

$Succ(u) \leftarrow Expand(u)$

for each v **in** $Succ(u)$

  if $(g + w(u, v) + h(v) > U)$

    if $(g + w(u, v) + h(v) < U')$

      $U' \leftarrow g + w(u, v) + h(v)$

  else

    $p \leftarrow IDA*(v, g + w(u, v), U)$

    if $(p \neq \emptyset)$ **return** (u, p)

return $\emptyset$

Optimality of ITERATIVE-DEEPENING A*

Theorem 5.4 (Optimality Iterative-Deepening A*)

Algorithm IDA* for graphs with admissible weight function is optimal.

Proof

Something Notable

Something Notable

Properties

Proof

Something Notable

Something Notable

Properties

Proof

Something Notable

Something Notable

Properties

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Casting A* as a Dijkstra's Algorithm

Something Notable

We can use the following re-weighting to incorporate the heuristic the weight function and sometimes to avoid negative weights!!!

$$\widehat{w}(u,v) = w(u,v) - h(u) + h(v)$$

Note: as Dijkstra's Algorithm on a re-wighted graph!!!

Why?

One motivation for this transformation is to inherit correctness proofs!!!

Casting A* as a Dijkstra's Algorithm

Something Notable

We can use the following re-weighting to incorporate the heuristic the weight function and sometimes to avoid negative weights!!!

$$\widehat{w}(u,v) = w(u,v) - h(u) + h(v)$$

Note: as Dijkstra's Algorithm on a re-wighted graph!!!

Why?

One motivation for this transformation is to inherit correctness proofs!!!

A*: Re-Weighting Edges

Lemma 2.3

Let G be a weighted problem graph and $h : V \to \mathbb{R}$ a consistent heuristic. Define the modified weight $\widehat{w}(u,v) = w(u,v) - h(u) + h(v) \geq 0$. Let $\delta(s,t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s,t)$ be the corresponding value in the re-weighted graph.

1. For a path p, we have $w(p) = \delta(s,t)$, if and only if $\widehat{w}(p) = \widehat{\delta}(s,t)$.
2. Moreover, G has no negatively weighted cycles with respect to w if and only if it has none with respect to $\widehat{w}$.

Cinvestav

A*: Re-Weighting Edges

Lemma 2.3

Let G be a weighted problem graph and $h : V \rightarrow \mathbb{R}$ a consistent heuristic. Define the modified weight $\widehat{w}(u,v) = w(u,v) - h(u) + h(v) \geq 0$. Let $\delta(s,t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}(s,t)$ be the corresponding value in the re-weighted graph.

1. For a path p, we have $w(p) = \delta(s,t)$, if and only if $\widehat{w}(p) = \widehat{\delta}(s,t)$.
2. Moreover, G has no negatively weighted cycles with respect to w if and only if it has none with respect to $\widehat{w}$.

A*: Re-Weighting Edges

Lemma 2.3

Let G be a weighted problem graph and $h : V \to \mathbb{R}$ a consistent heuristic. Define the modified weight $\widehat{w}(u, v) = w(u, v) - h(u) + h(v) \geq 0$. Let $\delta\,(s, t)$ be the length of the shortest path from s to t in the original graph and $\widehat{\delta}\,(s, t)$ be the corresponding value in the re-weighted graph.

1. For a path p, we have $w(p) = \delta(s, t)$, if and only if $\widehat{w}(p) = \widehat{\delta}(s, t)$.
2. Moreover, G has no negatively weighted cycles with respect to w if and only if it has none with respect to $\widehat{w}$.

Cinvestav

However

Given the implicit graphs

We have the following question

Given a Incosistent Heuristic Re-Weighting helps at all?

Sometimes it does not work...

However

Given the implicit graphs

We have the following question

Given a Incosistent Heuristic Re-Weighting helps at all?

Sometimes it does not work...

Example of Re-weighting Edges on an Inconsistent Heuristic

Example: A problem graph before (left) and after (right) re-weighting.

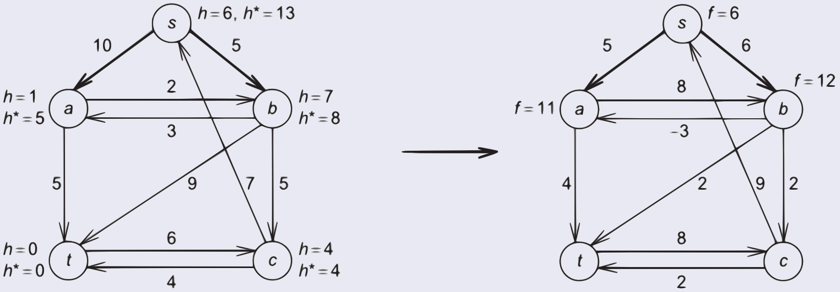


Figure: $h*(u) = \delta(u, t)$ and f for the first expansions in the new graph

Problem!!!

We have a INCONSISTENT heuristic

$$h(b) \geq h(a) + w(b, a)!!!$$

That creates a negative weight

How do we deal with an inconsistent heuristic?

Problem!!!

We have a INCONSISTENT heuristic

$$h(b) \geq h(a) + w(b, a)!!!$$

That creates a negative weight

How do we deal with an inconsistent heuristic?

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - **Dealing with the problem**
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Dealing with inconsistent but admissible heuristics

We use the idea of Pathmax

- Taking the maximum of the accumulated weights on the path to a node to enforce a monotone growth in the cost function.

Pathmax

For a node u with child v

- $f(v) = \max\{f(v), f(u)\}$ or equivalent $h(v) = \max\{h(v), h(u) - w(u,v)\}$.

Dealing with inconsistent but admissible heuristics

We use the idea of Pathmax

- Taking the maximum of the accumulated weights on the path to a node to enforce a monotone growth in the cost function.

Pathmax

For a node u with child v

- $f(v) = \max\{f(v), f(u)\}$ or equivalent $h(v) = \max\{h(v), h(u) - w(u, v)\}$.

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
    - Remember the code
- We lose the information for $(a, 12)$

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
  - Remember the code
- We lose the information for $(a, 12)$

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
  - Remember the code
- We lose the information for $(a, 12)$

Cinvestav

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
  - Remember the code
- We lose the information for $(a, 12)$

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
  - Remember the code
- We lose the information for $(a, 12)$

Cinvestav

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
    - Remember the code
- We lose the information for $(a, 12)$

Cinvestav

However

Even with this!!!

In the previous figure:

- After expanding s and a, we have $Open = \{(b, 12), (t, 15)\}$ and $Closed = \{(s, 6), (a, 11)\}$.
- Now a is reached by $(b, 12)$, and it is moved to $Closed$
- Then, it is compared to the closed list
- 12 is now the pathmax on path (s, b, a), but we never added to $Closed$
  - Remember the code
- We lose the information for $(a, 12)$

Therefore

Even with the Pathmax

- We have a problem!!!

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - **Best-First Searches**
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Best-First Searches

Best-First Searches

- They are a family of search algorithms which explores a graph by expanding the most promising node chosen according to a specified rule.
  - First described by Judea Pearl in "*Heuristics: Intelligent Search Strategies for Computer Problem Solving*," Addison-Wesley, 1984, p. 48.

  - *"It may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." — Judea Pearl*

Cinvestav

Best-First Searches

Best-First Searches

- They are a family of search algorithms which explores a graph by expanding the most promising node chosen according to a specified rule.
  - First described by Judea Pearl in "*Heuristics: Intelligent Search Strategies for Computer Problem Solving*," Addison-Wesley, 1984. p. 48.

For this they use...

- A heuristic evaluation function $f(n)$ for each node.
  - *"It may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." - Judea Pearl*

Best-First Searches

Best-First Searches

- They are a family of search algorithms which explores a graph by expanding the most promising node chosen according to a specified rule.
  - First described by Judea Pearl in "*Heuristics: Intelligent Search Strategies for Computer Problem Solving*," Addison-Wesley, 1984. p. 48.

For this they use...

- A heuristic evaluation function $f(n)$ for each node.
  - *"It may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." - Judea Pearl*

Best-First Searches

Best-First Searches

- They are a family of search algorithms which explores a graph by expanding the most promising node chosen according to a specified rule.
  - First described by Judea Pearl in "*Heuristics: Intelligent Search Strategies for Computer Problem Solving*," Addison-Wesley, 1984. p. 48.

For this they use...

- A heuristic evaluation function $f(n)$ for each node.
  - *"It may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." - Judea Pearl*

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Best-First Generic Algorithm

Best-First Generic Algorithm

1. $Open=$ [initial state]
2. $Closed=$ []
3. while
4. Remove the best node from $Open$, call it n, add it to $Closed$.
5. If n is the goal state, back-trace path to n and return path.
6. Create n's successors.
7. For each successor do:
8. If it is not in $Closed$:
   Evaluate it, add it to $Open$, and record its parent.
9. else change recorded parent if this new path is better than previous one.

Cinvestav

Best-First Generic Algorithm

Best-First Generic Algorithm

1. $Open=$ [initial state]
2. $Closed=$ []
3. while
4. Remove the best node from $Open$, call it n, add it to $Closed$.
5. If n is the goal state, back-trace path to n and return path.
6. Create n's successors.
7. For each successor do:
8. If it is not in $Closed$:
   Evaluate it, add it to $Open$, and record its parent.
9. else change recorded parent if this new path is better than previous one.

Cinvestav

Best-First Generic Algorithm

Best-First Generic Algorithm

1. $Open$= [initial state]
2. $Closed$= []
3. while
4. Remove the best node from $Open$, call it n, add it to $Closed$.
5. If n is the goal state, back-trace path to n and return path.
6. Create n's successors.
7. For each successor do:
8. If it is not in $Closed$:
   Evaluate it, add it to $Open$, and record its parent.
9. else change recorded parent if this new path is better than previous one.

Cinvestav

Best-First Generic Algorithm

Best-First Generic Algorithm

1. $Open=$ [initial state]
2. $Closed=$ []
3. while
4. Remove the best node from $Open$, call it n, add it to $Closed$.
5. If n is the goal state, back-trace path to n and return path.
6. Create n's successors.
7. For each successor do:
8. If it is not in $Closed$:
   Evaluate it, add it to $Open$, and record its parent.
9. else change recorded parent if this new path is better than previous one.

Cinvestav

Best-First Generic Algorithm

Best-First Generic Algorithm

1. $Open =$ [initial state]
2. $Closed =$ []
3. while
4. Remove the best node from $Open$, call it n, add it to $Closed$.
5. If n is the goal state, back-trace path to n and return path.
6. Create n's successors.
7. For each successor do:
8. If it is not in $Closed$:
   Evaluate it, add it to $Open$, and record its parent.
9. else change recorded parent if this new path is better than previous one.

Cinvestav

Best-First Generic Algorithm

Best-First Generic Algorithm

1. $Open=$ [initial state]
2. $Closed=$ []
3. while
4. Remove the best node from $Open$, call it n, add it to $Closed$.
5. If n is the goal state, back-trace path to n and return path.
6. Create n's successors.
7. For each successor do:
8. If it is not in $Closed$:
   Evaluate it, add it to $Open$, and record its parent.
9. else change recorded parent if this new path is better than previous one.

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - **Greedy Best First Search**
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

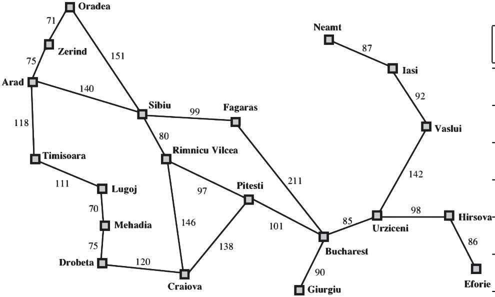
Cinvestav

Greedy Best First Search

Definition

- Evaluation function $f(n) = h(n)$
- $h(n)$= estimate of cost from n to goal.
- Greedy best-first search **expands** the node that appears to be closest to goal

Example



Straight Line Distance	to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgu	77

Straight Line Distance	to Bucharest
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Meamt	234
Oradea	380
Pitasti	10
Rimnicu Vilcea	193
Sibiu	253
Timisora	329
Urziceni	80
Vaslui	199
Zerind	374

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Cinvestav

Greedy BFS Vs. A*

Properties of greedy Best-First Search

- **Complete?** No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->
- **Time?** $O(bm)$, but a good heuristic can give dramatic improvement
- **Space?** $O(bm)$ – keeps all nodes in memory
- **Optimal?** No

- **Time?** Exponential $O(b^m)$
- **Space?** Keeps all nodes in memory Worst case $O(b^m)$
- **Optimal?** Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

- **Complete?** No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->
- **Time?** $O(bm)$, but a good heuristic can give dramatic improvement
- **Space?** $O(bm)$ – keeps all nodes in memory
- **Optimal?** No

Vs A* Properties

- **Complete?** Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)
- **Time?** Exponential $O(b^m)$
- **Space?** Keeps all nodes in memory Worst case $O(b^m)$
- **Optimal?** Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

Complete? No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->

Time? $O(bm)$, but a good heuristic can give dramatic improvement

Space? $O(bm)$ – keeps all nodes in memory

Optimal? No

Vs A* Properties

Complete? Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)

Time? Exponential $O(b^m)$

Space? Keeps all nodes in memory Worst case $O(b^m)$

Optimal? Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

Complete? No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->

Time? $O(bm)$, but a good heuristic can give dramatic improvement

Space? $O(bm)$ – keeps all nodes in memory

Optimal? No

Vs A* Properties

Complete? Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)

Time? Exponential $O(b^m)$

Space? Keeps all nodes in memory Worst case $O(b^m)$

Optimal? Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

Complete? No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->

Time? $O(bm)$, but a good heuristic can give dramatic improvement

Space? $O(bm)$ – keeps all nodes in memory

Optimal? No

Vs A* Properties

Complete? Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)

Time? Exponential $O(b^m)$

Space? Keeps all nodes in memory Worst case $O(b^m)$

Optimal? Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

Complete? No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->

Time? $O(bm)$, but a good heuristic can give dramatic improvement

Space? $O(bm)$ – keeps all nodes in memory

Optimal? No

Vs A* Properties

Complete? Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)

Time? Exponential $O(b^m)$

Space? Keeps all nodes in memory Worst case $O(b^m)$

Optimal? Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

- **Complete?** No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->
- **Time?** $O(bm)$, but a good heuristic can give dramatic improvement
- **Space?** $O(bm)$ – keeps all nodes in memory
- **Optimal?** No

Vs A* Properties

- **Complete?** Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)
- **Time?** Exponential $O(b^m)$
- **Space?** Keeps all nodes in memory Worst case $O(b^m)$
- **Optimal?** Yes

Greedy BFS Vs. A*

Properties of greedy Best-First Search

Complete? No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt ->

Time? $O(bm)$, but a good heuristic can give dramatic improvement

Space? $O(bm)$ – keeps all nodes in memory

Optimal? No

Vs A* Properties

Complete? Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$)

Time? Exponential $O(b^m)$

Space? Keeps all nodes in memory Worst case $O(b^m)$

Optimal? Yes

Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Origin of Heuristics

Common View

- Heuristic could come from relaxing the constraints of a problem and trying to solve it exactly!!!

- It can be interpreted as adding straight routes to the map

- It is used to automate the generation of heuristics

Origin of Heuristics

Common View

- Heuristic could come from relaxing the constraints of a problem and trying to solve it exactly!!!

Example

- A prominent example for this is the straight-line distance estimate for routing problems.
- It can be interpreted as adding straight routes to the map.

Cinvestav

Origin of Heuristics

Common View

- Heuristic could come from relaxing the constraints of a problem and trying to solve it exactly!!!

Example

- A prominent example for this is the straight-line distance estimate for routing problems.
- It can be interpreted as adding straight routes to the map.

Example

- This is captured by the **abstraction transformation**.
- It is used to automate the generation of heuristics.

Origin of Heuristics

Common View

- Heuristic could come from relaxing the constraints of a problem and trying to solve it exactly!!!

Example

- A prominent example for this is the straight-line distance estimate for routing problems.
- It can be interpreted as adding straight routes to the map.

Example

- This is captured by the **abstraction transformation**.
- It is used to automate the generation of heuristics.

Cinvestav

Origin of Heuristics

Common View

- Heuristic could come from relaxing the constraints of a problem and trying to solve it exactly!!!

Example

- A prominent example for this is the straight-line distance estimate for routing problems.
- It can be interpreted as adding straight routes to the map.

Example

- This is captured by the **abstraction transformation**.
- It is used to automate the generation of heuristics.

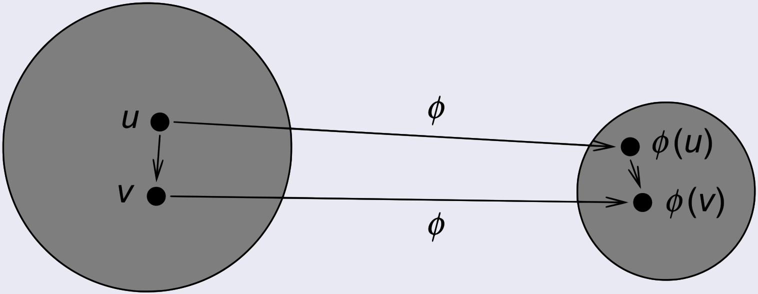
Outline

1. Informed Optimal Search
   - What is an Heuristic?
   - Formal Definition of a Heuristic
   - Desirable Properties of a Heuristic
     - Consistency and Monotonicity
     - Dominance
2. A* Algorithm
   - The Heuristic A*
   - Pseudo-Code
   - Consistency of A*
   - Admissibility in A*
   - Lemma Toward Admissibility of A*
   - Expansion of Different Strategies
   - Optimality in A*
   - Iterative-Deepening for A*
   - A*: Re-weighting Edges
   - Dealing with the problem
   - Best-First Searches
     - Algorithm
   - Greedy Best First Search
   - Greedy Best-First Search Vs. A* Algorithm
3. Limits in Heuristics
   - Where do heuristics come from?
   - Abstraction Transformations and Valtortas's Theorem

Abstraction Transformations

Definition 4.1

- An **abstraction transformation** $\phi : S \to S'$ maps states u in the concrete problem space to abstract states $\phi(u)$ and concrete actions a to abstract actions $\phi(a)$.



Thus

We have the following Intuition

- Intuitively, this agrees with a common explanation of the origin of heuristics.
- As the cost of exact solutions to a relaxed problem.
- A relaxed problem is one where we drop constraints (e.g., on move execution).

Cinvestav

Thus

We have the following Intuition

- Intuitively, this agrees with a common explanation of the origin of heuristics.
- As the cost of exact solutions to a relaxed problem.
- A relaxed problem is one where we drop constraints (e.g., on move execution).

Example

- For example, the Manhattan distance for sliding-tile puzzles can be regarded as acting in an abstract problem space that allows multiple tiles to occupy the same square.

Cinvestav

Thus

We have the following Intuition

- Intuitively, this agrees with a common explanation of the origin of heuristics.
- As the cost of exact solutions to a relaxed problem.
- A relaxed problem is one where we drop constraints (e.g., on move execution).

Example

- For example, the Manhattan distance for sliding-tile puzzles can be regarded as acting in an abstract problem space that allows multiple tiles to occupy the same square.

Thus

We have the following Intuition

- Intuitively, this agrees with a common explanation of the origin of heuristics.
- As the cost of exact solutions to a relaxed problem.
- A relaxed problem is one where we drop constraints (e.g., on move execution).

Example

- For example, the Manhattan distance for sliding-tile puzzles can be regarded as acting in an abstract problem space that allows multiple tiles to occupy the same square.

Cinvestav

Embedding and Homomorphism

Definition 4.2

- An Abstraction Transformation (Map) ϕ is an **embedding transformation** if it adds edges to S such that the concrete and abstract state sets are the same; that is, $\phi(u) = u$ for all $u \in S$.
- An **Abstract Homomorphism** requires that for all edges $(u, v) \in S$, there must also be an edge $(\phi(u), \phi(v)) \in S'$.

Cinvestav

Embedding and Homomorphism

Theorem 4.1 (Admissibility and Consistency of Abstraction Heuristics)

- Let S be a state space and $S' = \phi(S)$ be any homomorphic abstraction transformation of S. Let heuristic function $h_\phi(u)$ for state u and goal t be defined as the length of the shortest path from $\phi(u)$ to $\phi(t)$ in S .
    - Then h_ϕ is an admissible, consistent heuristic function.

Embedding and Homomorphism

Theorem 4.1 (Admissibility and Consistency of Abstraction Heuristics)

- Let S be a state space and $S' = \phi(S)$ be any homomorphic abstraction transformation of S. Let heuristic function $h_{\phi}(u)$ for state u and goal t be defined as the length of the shortest path from $\phi(u)$ to $\phi(t)$ in S .
  - Then h_{ϕ} is an admissible, consistent heuristic function.

VALTORTA'S THEOREM

VALTORTA'S THEOREM

- Let u be any state necessarily expanded, when the problem (s, t) is solved in S with Breadth-First Serch. In addition:

VALTORTA'S THEOREM

VALTORTA'S THEOREM

- Let u be any state necessarily expanded, when the problem (s, t) is solved in S with Breadth-First Serch. In addition:
  - $\phi : S \rightarrow S'$ be any abstraction mapping; the heuristic estimate $h(u)$ be computed by blindly searching from $\phi(u)$ to $\phi(t)$.

VALTORTA'S THEOREM

VALTORTA'S THEOREM

- Let u be any state necessarily expanded, when the problem (s, t) is solved in S with Breadth-First Serch. In addition:
  - $\phi : S \rightarrow S'$ be any abstraction mapping; the heuristic estimate $h(u)$ be computed by blindly searching from $\phi(u)$ to $\phi(t)$.
  - If the problem is solved by the A* algorithm using h, then either u itself will be expanded, or $\phi(u)$ will be expanded.

Consequences of Valtora's Theorem

Corollary 4.1

For an embedding ϕ, A*-using h computed by blind search in the abstract problem space-necessarily expands every state that is expanded by blind search in the original space.

Consequences of Valtora's Theorem

Observe!!

- Based on this theorem, we define "**Valtorta's Barrier**" to be the number of nodes expanded when blindly searching in a space.

- It is more, they can reduce the search effort, since the abstract space is often smaller than the original one.

Consequences of Valtora's Theorem

Observe!!

- Based on this theorem, we define "**Valtorta's Barrier**" to be the number of nodes expanded when blindly searching in a space.
- Valtorta's theorem states that this barrier cannot be "broken" using any embedding transformation.

Cinvestav

Consequences of Valtora's Theorem

Observe!!

- Based on this theorem, we define "**Valtorta's Barrier**" to be the number of nodes expanded when blindly searching in a space.
- Valtorta's theorem states that this barrier cannot be "broken" using any embedding transformation.

HOWEVER!!!

- Contrary to the case of embeddings, this negative result of Valtorta's theorem does not apply in this way to abstractions based on homomorphisms.
- It is more, they can reduce the search effort, since the abstract space is often smaller than the original one.

Cinvestav

Consequences of Valtora's Theorem

Observe!!

- Based on this theorem, we define "**Valtorta's Barrier**" to be the number of nodes expanded when blindly searching in a space.
- Valtorta's theorem states that this barrier cannot be "broken" using any embedding transformation.

HOWEVER!!!

- Contrary to the case of embeddings, this negative result of Valtorta's theorem does not apply in this way to abstractions based on homomorphisms.
- It is more, they can reduce the search effort, since the abstract space is often smaller than the original one.

Bibliography

S. Edelkamp and S. Schrodl, *Heuristic Search - Theory and Applications.*
Academic Press, 2012.