Dijkstra Variants: A* and Potentials

Eric Price

UT Austin

CS 331, Spring 2020 Coronavirus Edition

Class Outline

1. Bottleneck Shortest Paths

2. A* search

3. Problems

Logistics

- “Raise hand” will hopefully not crash my connection now.

Logistics

- "Raise hand" will hopefully not crash my connection now.
  - So you can try that, as well as chat, for questions.

Logistics

- “Raise hand” will hopefully not crash my connection now.
  - So you can try that, as well as chat, for questions.
- We’re going to try using Zoom breakout rooms for problems, later today.

Logistics

- "Raise hand" will hopefully not crash my connection now.
  - So you can try that, as well as chat, for questions.
- We're going to try using Zoom breakout rooms for problems, later today.
  - Inside, you can "Ask for help" and it pops up a notification for me.

Logistics

- "Raise hand" will hopefully not crash my connection now.
  - So you can try that, as well as chat, for questions.
- We're going to try using Zoom breakout rooms for problems, later today.
  - Inside, you can "Ask for help" and it pops up a notification for me.
  - You stop being able to see my screen, so be sure to record the exercises before joining the breakout room.

Talk Outline

1. Bottleneck Shortest Paths

2. A* search

3. Problems

Bottleneck Shortest Paths

- Mentioned before spring break: network of roads, each has bridges of various heights on it. How high of a truck can go from s to t, or anywhere else?

Bottleneck Shortest Paths

- Mentioned before spring break: network of roads, each has bridges of various heights on it. How high of a truck can go from s to t, or anywhere else?
  - Max bandwidth path from s to t

Bottleneck Shortest Paths

- Mentioned before spring break: network of roads, each has bridges of various heights on it. How high of a truck can go from s to t, or anywhere else?
  - Max bandwidth path from s to t
  - We'll use it for network flows

Bottleneck Shortest Paths

- Mentioned before spring break: network of roads, each has bridges of various heights on it. How high of a truck can go from s to t, or anywhere else?
  - Max bandwidth path from s to t
  - We'll use it for network flows
- On undirected graph:

Bottleneck Shortest Paths

- Mentioned before spring break: network of roads, each has bridges of various heights on it. How high of a truck can go from s to t, or anywhere else?
  - Max bandwidth path from s to t
  - We'll use it for network flows
- On undirected graph: is maximum spanning tree

Bottleneck Shortest Paths

- Mentioned before spring break: network of roads, each has bridges of various heights on it. How high of a truck can go from s to t, or anywhere else?
  - Max bandwidth path from s to t
  - We'll use it for network flows
- On undirected graph: is maximum spanning tree
- On directed graph: Dijkstra/Prim variant solves in $O(E + V \log V)$.

Dijkstra's Algorithm

```
function DIJKSTRA(s)
    pred, dist ← {}, {}
    q ← PRIORITYQUEUE([(0, s, None)])          ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_min()
        if u ∈ pred then
            continue
        pred[u] ← parent
        dist[u] ← d
        for u → v ∈ E do
            q.push( (dist[u] + w(u → v), v, u) )
    return dist, pred
```

~~Dijkstra's~~ Prim's Algorithm

```
function PRIM(s)
    pred, dist ← {}, {}
    q ← PRIORITYQUEUE([(−∞, s, None)])          ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_min()
        if u ∈ pred then
            continue
        pred[u] ← parent
        dist[u] ← d
        for u → v ∈ E do
            q.push( (dist[u] + w(u → v), v, u) )
    return dist, pred
```

Dijkstra's Algorithm

```
function DIJKSTRA(s)
    pred, dist ← {}, {}
    q ← PRIORITYQUEUE([(0, s, None)])                ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_min()
        if u ∈ pred then
            continue
        pred[u] ← parent
        dist[u] ← d
        for u → v ∈ E do
            q.push( (dist[u] + w(u → v),    v, u) )
    return dist, pred
```

~~Dijkstra's~~ Bottleneck shortest path Algorithm

```
function BOTTLENECK(s)
    pred, dist ← {}, {}
    q ← PRIORITYQUEUE([(0, s, None)])                ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_max()
        if u ∈ pred then
            continue
        pred[u] ← parent
        dist[u] ← d
        for u → v ∈ E do
            q.push( (min(dist[u], w(u → v)), v, u) )
    return dist, pred
```

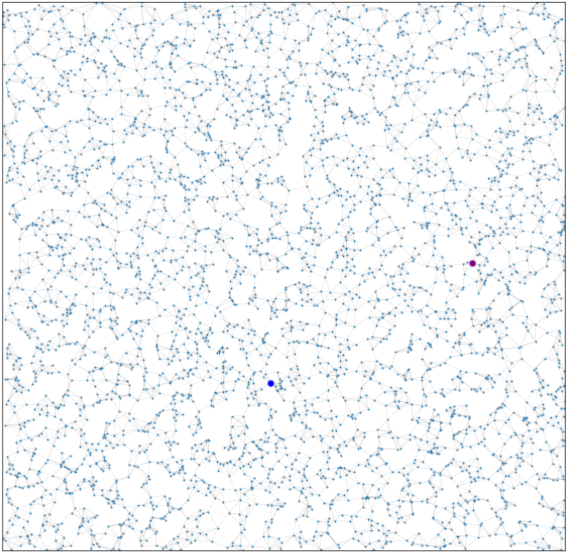
~~Dijkstra's~~ Bottleneck shortest path Algorithm

```
function BOTTLENECK(s)
    pred, dist ← {}, {}
    q ← PRIORITYQUEUE([(0, s, None)])                ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_max()
        if u ∈ pred then
            continue
        pred[u] ← parent
        dist[u] ← d
        for u → v ∈ E do
            q.push( (min(dist[u], w(u → v)), v, u) )
    return dist, pred
```

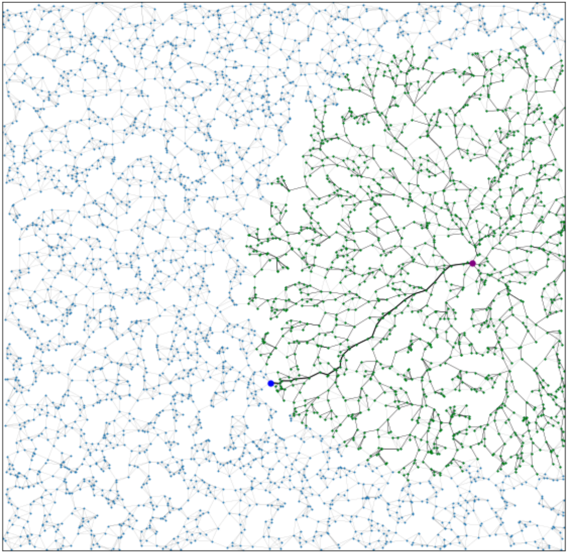
$$(\min, +) \to (\max, \min)$$

Talk Outline

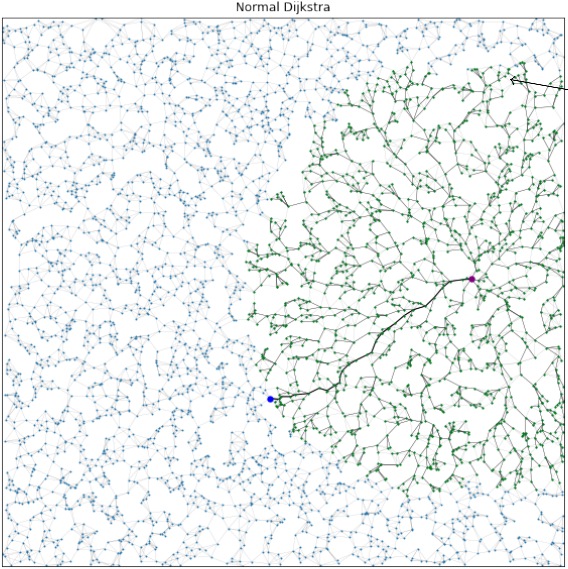
Shortest $s - t$ path with Dijkstra



Shortest $s-t$ path with Dijkstra



Shortest $s - t$ path with Dijkstra



This is the wrong direction. Why waste our timing exploring it?

A^* search

- Dijkstra explores outward from s.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $= 1740$ miles.
  - Austin$\rightarrow$ San Francisco $= 1760$ miles.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $= 1740$ miles.
  - Austin$\rightarrow$ San Francisco $= 1760$ miles.
- Dijkstra will visit NYC before SF.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $= 1740$ miles.
  - Austin$\rightarrow$ San Francisco $= 1760$ miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.
  - Austin$\rightarrow$ San Francisco $=$ 1760 miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $= 1740$ miles.
  - Austin$\rightarrow$ San Francisco $= 1760$ miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.
  - Austin$\rightarrow$ San Francisco $=$ 1760 miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?
  - Fact: you cannot get from NYC to SF in 20 miles.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $= 1740$ miles.
  - Austin$\rightarrow$ San Francisco $= 1760$ miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?
  - Fact: you cannot get from NYC to SF in 20 miles.
- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.
  - Austin$\rightarrow$ San Francisco $=$ 1760 miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?
  - Fact: you cannot get from NYC to SF in 20 miles.
- A^*: uses a *heuristic* $h(u)$, estimating $d(u,t)$.
  - Dijkstra: visit node of smallest dist[u]

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.
  - Austin$\rightarrow$ San Francisco $=$ 1760 miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?
  - Fact: you cannot get from NYC to SF in 20 miles.
- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - Dijkstra: visit node of smallest dist$[u]$
  - A^*: visit node of smallest dist$[u] + h(u)$

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.
  - Austin$\rightarrow$ San Francisco $=$ 1760 miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?
  - Fact: you cannot get from NYC to SF in 20 miles.
- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - Dijkstra: visit node of smallest dist$[u]$
  - A^*: visit node of smallest dist$[u] + h(u)$
- Example: $h(\text{NYC})$ is Euclidean distance from NYC to SF.

A^* search

- Dijkstra explores outward from s.
  - Can stop if it reaches t, but doesn't bias search toward t.
- Consider Dijkstra from Austin to San Francisco:
  - Austin$\rightarrow$ New York $=$ 1740 miles.
  - Austin$\rightarrow$ San Francisco $=$ 1760 miles.
- Dijkstra will visit NYC before SF.
  - Once it visits SF, it stops searching
  - So it needs to visit NYC, in case it can get from NYC to SF in 20 miles.
  - ...with a portal or something?
  - Fact: you cannot get from NYC to SF in 20 miles.
- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - Dijkstra: visit node of smallest dist$[u]$
  - A^*: visit node of smallest dist$[u] + h(u)$
- Example: $h(\text{NYC})$ is Euclidean distance from NYC to SF.
  - Any path through NYC will take *at least* 3700 miles.

Dijkstra's Algorithm

```
function DIJKSTRA(s)
    pred, dist ← {}, {}
    q ← PRIORITYQUEUE([(0 , s, None)])          ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_min()
        if d ≥ dist[u] then
            continue
        pred[u] ← parent
        dist[u] ← d
        for u → v ∈ E do
            q.push( (dist[u] + w(u → v), v, u) )
    return dist, pred
```

~~Dijkstra's~~ A* Search Algorithm

```
function A*(s)
    pred, dist ← {}, {}
    q ← PriorityQueue([(0 + h(s), s, None)])    ▷ dist, vertex, pred
    while q do
        d, u, parent ← q.pop_min()
        if d−h(u) ≥ dist[u] then
            continue
        pred[u] ← parent
        dist[u] ← d - h(u)
        for u → v ∈ E do
            q.push( (dist[u] + w(u → v)+ h(v), v, u) )
    return dist, pred
```

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

[$h(u)$ is height of a hill: easy to go down, hard to go up.]

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u,t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

    $$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

    $$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:
  - Heuristic "admissible:" $h(u) \leq d(u, t)$

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u,t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:
  - Heuristic "admissible:" $h(u) \leq d(u,t)$
    - ⋆ Admissible $\implies$ first visit to t gives optimal path, so correct.

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u,t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:
  - Heuristic "admissible:" $h(u) \leq d(u,t)$
    - ⋆ Admissible $\implies$ first visit to t gives optimal path, so correct.
  - Heuristic "consistent:" $h(t) = 0$ and $h(u) \leq w(u,v) + h(v)$.

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:
  - Heuristic "admissible:" $h(u) \leq d(u, t)$
    - ★ Admissible $\implies$ first visit to t gives optimal path, so correct.
  - Heuristic "consistent:" $h(t) = 0$ and $h(u) \leq w(u, v) + h(v)$.
    - ★ Equivalent: $h(t) = 0$ and $w'(u, v) \geq 0$ for all u, v.

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u,t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

$$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:
  - Heuristic "admissible:" $h(u) \leq d(u,t)$
    - Admissible $\implies$ first visit to t gives optimal path, so correct.
  - Heuristic "consistent:" $h(t) = 0$ and $h(u) \leq w(u,v) + h(v)$.
    - Equivalent: $h(t) = 0$ and $w'(u,v) \geq 0$ for all u, v.
    - $w' \geq 0 \implies$ Dijkstra is fast/correct (depending on implementation).

Heuristics/potential functions

- A^*: uses a *heuristic* $h(u)$, estimating $d(u, t)$.
  - A^*: visit node of smallest $\text{dist}[u] + h(u)$
  - Paths equivalent to Dijkstra on a *reweighted* graph:

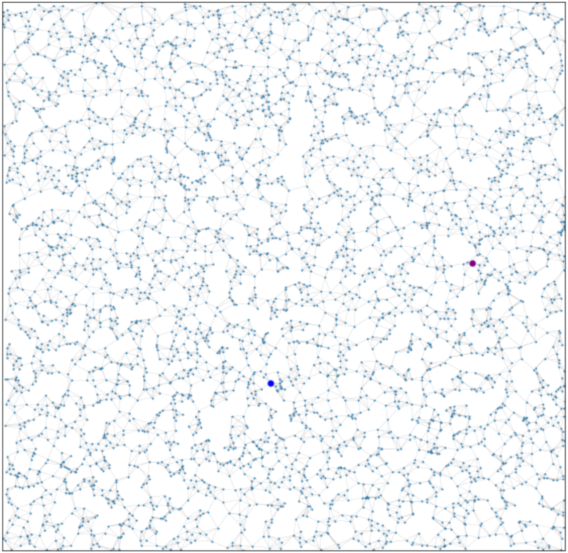
    $$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

    [$h(u)$ is height of a hill: easy to go down, hard to go up.]
  - Every $s \rightsquigarrow t$ path P on w' has length

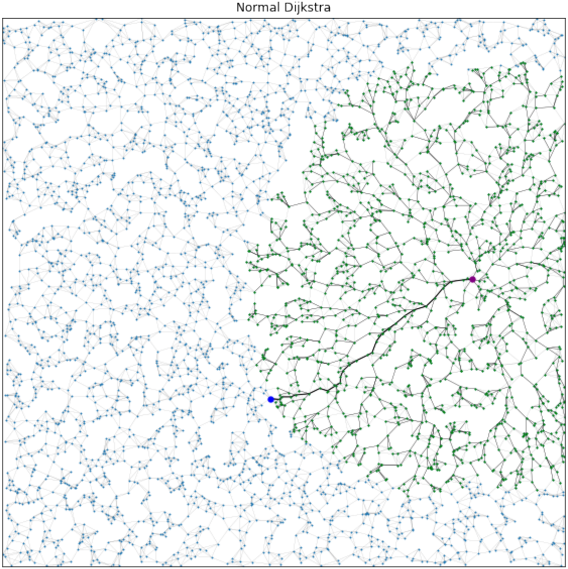
    $$\sum_{e \in P} w'(e) = h(t) - h(s) + \sum_{e \in P} w(e)$$

  - So which path is shortest is same under w or w'.
- Heuristics:
  - Heuristic "admissible:" $h(u) \leq d(u, t)$
    - Admissible $\implies$ first visit to t gives optimal path, so correct.
  - Heuristic "consistent:" $h(t) = 0$ and $h(u) \leq w(u, v) + h(v)$.
    - Equivalent: $h(t) = 0$ and $w'(u, v) \geq 0$ for all u, v.
    - $w' \geq 0 \implies$ Dijkstra is fast/correct (depending on implementation).
    - And consistent $\implies$ admissible.

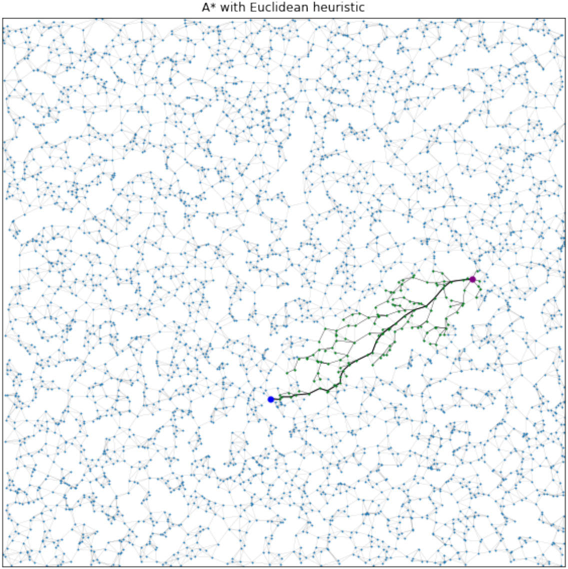
Shortest $s-t$ path with Dijkstra



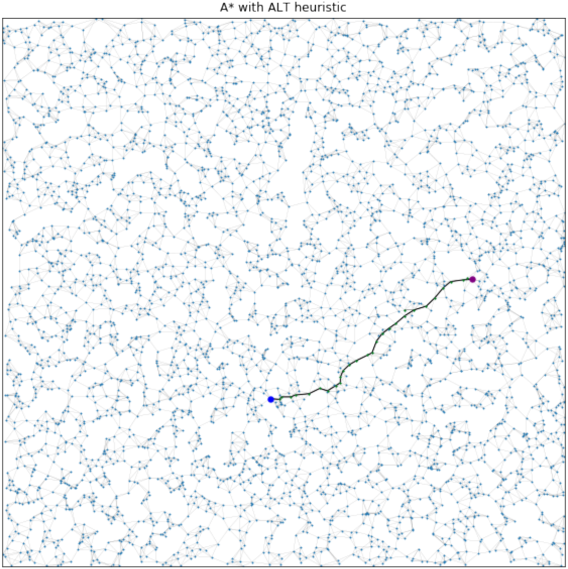
Shortest $s - t$ path with Dijkstra



Shortest $s - t$ path with Dijkstra



Shortest $s - t$ path with Dijkstra



Summary of Dijkstra variants

- Bottleneck shortest paths

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?
- Dijkstra with potentials:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?
- Dijkstra with potentials:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

  - Can adjust the graph to have nonnegative weights

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?
- Dijkstra with potentials:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

  - Can adjust the graph to have nonnegative weights
  - Can adjust the graph to bias toward goal t (A^* search).

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?
- Dijkstra with potentials:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

  - Can adjust the graph to have nonnegative weights
  - Can adjust the graph to bias toward goal t (A^* search).
  - Admissible $\implies$ correct

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?
- Dijkstra with potentials:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

  - Can adjust the graph to have nonnegative weights
  - Can adjust the graph to bias toward goal t (A^* search).
  - Admissible $\implies$ correct
  - Consistent $\implies$ correct and $O(E + V \log V)$

Summary of Dijkstra variants

- Bottleneck shortest paths
  - How do they relate to MSTs?
- Dijkstra with potentials:

$$w'(u \to v) = w(u \to v) - h(u) + h(v).$$

  - Can adjust the graph to have nonnegative weights
  - Can adjust the graph to bias toward goal t (A^* search).
  - Admissible $\implies$ correct
  - Consistent $\implies$ correct and $O(E + V \log V)$
  - Can be faster in many cases.

Talk Outline

1. Bottleneck Shortest Paths
2. A* search
3. Problems

Shortest Path Problems

http://jeffe.cs.illinois.edu/teaching/algorithms/book/08-sssp.pdf

- Problem 2: Dijkstra with k negative edges.
- Problem 3: *vertices*, not edges, have weight.
- Problem 5: edge reinsertion
- Problem 4: Replacement paths on directed graphs
- Problem 12: Smallest shortest path
- Problem 16, 17: Remember reductions?
- Problem 1 of https://www.cs.utexas.edu/~ecprice/courses/331h/psets/331h-ps6.pdf