Secretary Hiring Problems
&
A* Algorithm

Buy one share at the lowest price

- Which one to buy?

The Secretary Problem

- Suppose there are n secretaries that are interviewed one at a time. The hiring decision has to be made right after an interview is done. After we interview a candidate, we know how this candidate ranks in comparison to the earlier interviewed candidates. However, we have no information about the upcoming candidates.

- *Design a strategy to hire the best candidate?*

The Secretary Problem

- Each secretary i has an inherent rating v_i
- But we do not know what is the maximum possible rating
- Secretaries interview in an unknown order

raise your hand if you will hire the secretary

- 1. $vi = 10$
- 2. $vi = 21$
- 3. $vi = 1$
- 4. $vi = 5$
- 5. $vi = 15$
- 6. $vi = 45$
- 7. $vi = 11$
- 8. $vi = 3$
- 9. $vi = 2$
- 10. $vi = 9$

raise your hand if you will hire the secretary

- 1. $v_i = 21$
- 2. $v_i = 34$
- 3. $v_i = 45$
- 4. $v_i = 5$
- 5. $v_i = 19$
- 6. $v_i = 3$
- 7. $v_i = 32$
- 8. $v_i = 4$
- 9. $v_i = 55$
- 10. $v_i = 7$

raise your hand if you will hire the secretary

- 1. $vi = 45$
- 2. $vi = 4$
- 3. $vi = 5$
- 4. $vi = 6$
- 5. $vi = 7$
- 6. $vi = 8$
- 7. $vi = 9$
- 8. $vi = 10$
- 9. $vi = 1$
- 10. $vi = 46$

The Secretary Algorithm v. 1

- Observe the first $n/2$ secretaries but don't hire anyone
- Let i be the best of the first $n/2$ secretaries
- After observing the first $n/2$ secretaries, hire the first secretary that is better than i
- If none of the remaining are better than i, then hire the last one

The Secretary Algorithm v. 1

- Worst-Case Analysis
- Hopeless. The worst case is when the secretaries appear in a descending order of their rank. The algorithm is arbitrarily bad. In fact, any deterministic algorithm is arbitrarily bad.

The Secretary Problem

- Suppose there are n secretaries that are interviewed one at a time. The hiring decision has to be made right after an interview is done. After we interview a candidate, we know how this candidate ranks in comparison to the earlier interviewed candidates. However, we have no information about the upcoming candidates.

- Design a strategy to maximize the probability of hiring the best candidate assuming they appear in a uniformly at random order?

Results

- If the number of secretaries tends to infinity, the optimal rule is to observe the first n/e secretaries, and then pick the first secretary better than the best in the first n/e secretaries

The Secretary Algorithm v. 1

- Observe the first $n/2$ secretaries but don't hire anyone
- Let i be the best of the first $n/2$ secretaries
- After observing the first $n/2$ secretaries, hire the first secretary that is better than i
- If none of the remaining are better than i, then hire the last one

- **Expected case analysis?** Can we compute the probability of picking the best candidate?

The Secretary Algorithm v. 2

- Observe the first r-1 secretaries but don't hire anyone
- Let i be the best of the first r -1 secretaries
- After observing the first r-1 secretaries, hire the first secretary that is better than i
- If none of the remaining are better than i, then hire the last one

The Secretary Algorithm v. 2

- Observe the first r-1 secretaries but don't hire anyone
- Let i be the best of the first r-1 secretaries
- After observing the first r-1 secretaries, hire the first secretary that is better than i
- If none of the remaining are better than i, then hire the last one

- **Optimal strategy:**
- Best strategy is to observe the first ~37% candidates and then hire the first one better than the 37%
- Probability of picking the best candidate ~= 37%

Resources

- Primary: Ferguson, Thomas S. "Who solved the secretary problem?." Statistical science (1989): 282-289.

Many, many versions

- minimize the expected rank
- maximize expected rating
- ratings decay over time
- multi-choice (k choice) hiring
- submodular ratings
- matroidal constraints
- knapsack constraints
- unknown n
- sliding windows
- …

Many, many versions

- minimize the expected rank
- maximize expected rating
- ratings decay over time
- multi-choice (k choice) hiring
- submodular ratings
- matroidal constraints
- knapsack constraints
- unknown n
- sliding windows
- …

Maximize the expected rating

- Suppose there are n secretaries that are interviewed one at a time. The hiring decision has to be made right after an interview is done. After we interview a candidate, we know how this candidate ranks in comparison to the earlier interviewed candidates. However, we have no information about the upcoming candidates.

- *Design a strategy to maximize the expected rating of the hired candidate assuming that the ratings are drawn from an unknown distribution.*

Thoughts?

Maximize the expected rating

- Same strategy (Secretary Algorithm v.2)
- In the limit, the probability of picking the best candidate is 1/e
- Therefore, the expected rating of the hired candidate is 1/e times the highest rating
- This is also the optimal algorithm!

Babaioff, Moshe, et al. "Online auctions and generalized secretary problems." ACM SIGecom Exchanges 7.2 (2008): 7.

Multiple Choice Secretary Problem

- Suppose there are n secretaries that are interviewed one at a time. We wish to *hire at most k* secretaries. The hiring decision has to be made right after an interview is done. Each candidate has a *rating vi that is revealed* to us after we interview the candidate i. We have no information about the upcoming candidates.

- Design a strategy to *maximize the expected sum of ratings* of the hired candidates.

Ideas?

Build on the k=1 algorithm

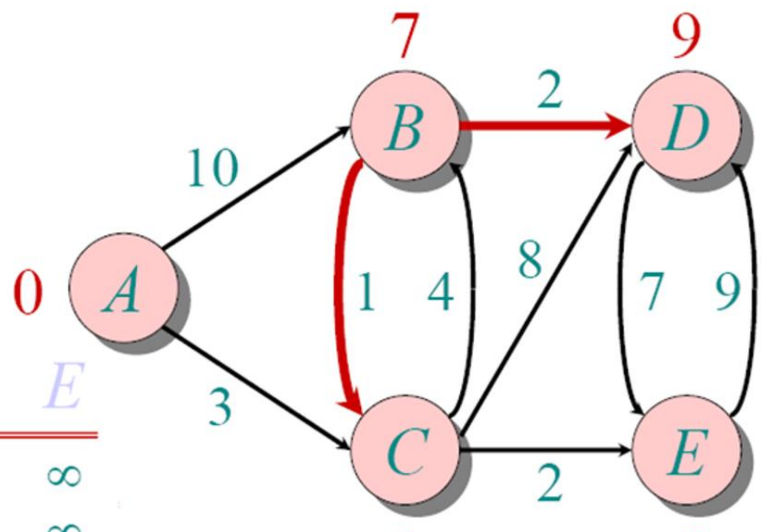
- Observe the first n/e - 1 secretaries but don’t hire anyone
- Let R = $\{v_1, v_2, \dots, v_k\}$ be the k best ratings observed
- After observing the first n/e - 1 secretaries, if a secretary has rating better than the worst rating in R, then
  - hire this secretary; and
  - remove the lowest rating in R

Build on the k=1 algorithm

- The previous algorithm achieves an expected rating of 1/e times the sum of the k largest ratings, in the limit that n tends to infinity.

Babaioff, Moshe, et al. "A knapsack secretary problem with applications." Approximation, randomization, and combinatorial optimization. Algorithms and techniques (2007): 16-28.

Revisit Dijkstra



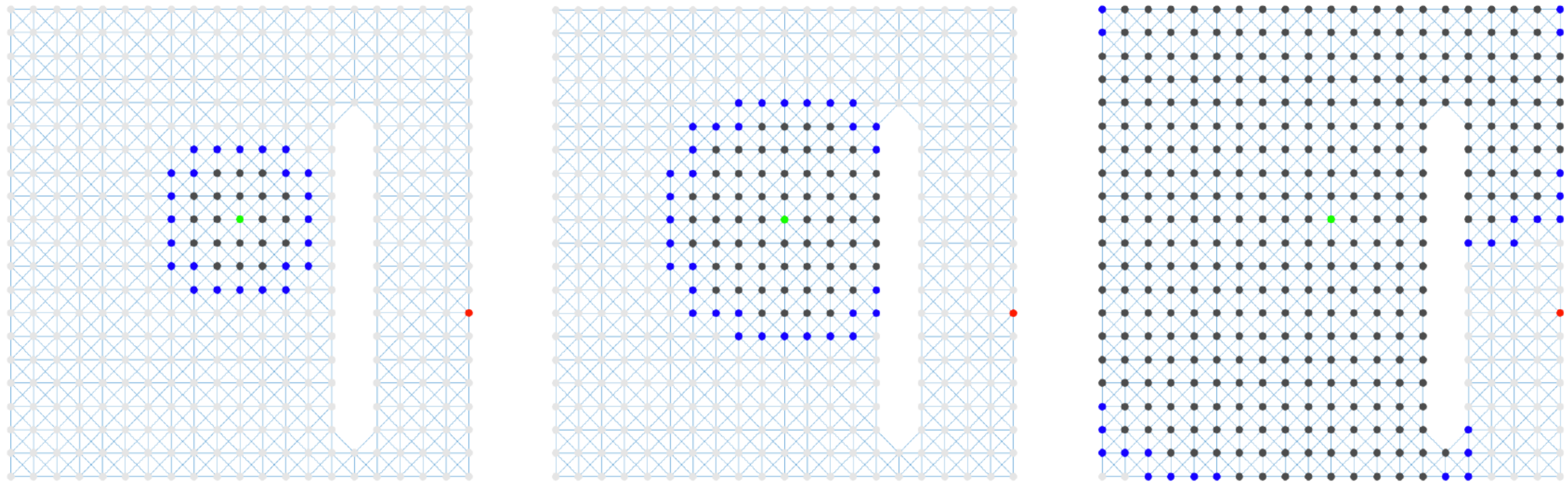
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

A* Algorithm

Open and Closed Lists

- Think of open lists as frontiers of expansion
- We start with x_o and expand neighbors until we reach x_G
  - Dijkstra: expand frontier that is closest to x_o

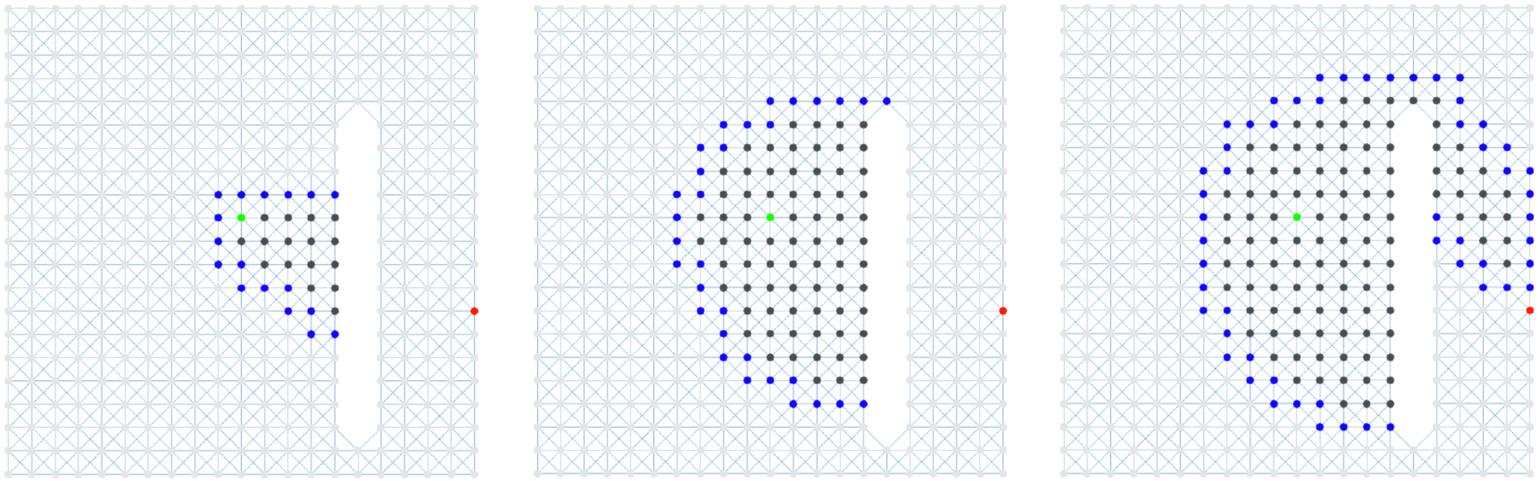


A*

- Open list O = {x_o}, closed list C = {}
- V(x_o) = 0, V(x_i) = ∞
- repeat until x_G is in C
  - x_j : vertex in O with lowest cost-to-come + heuristic
    - remove it from O and store it in C, closed list
  - for each neighbor x_i of x_j not already in C
    - compute V_{new} = C(x_j, x_i) + V(x_j)
    - update V(x_i) if it is more than V_{new}
    - if not in O, then add to O

Open and Closed Lists

- Think of open lists as frontiers of expansion
- We start with x_o and expand neighbors until we reach x_G
  - A*: expand frontier that our heuristic says will lead to minimum cost path



What's a good heuristic?

- heuristic gives an overestimate of actual V(x_G)
  - h(x_j) > minimum cost to reach x_G from x_j
- heuristic gives an underestimate of actual V(x_G)
  - h(x_j) >= minimum cost to reach x_G from x_j

Admissible Heuristic

- A heuristic, h(x_j), is called as admissible heuristic if and only if h(x_j) <= minimum cost to reach x_G from x_j for all x_j

- A* will find the optimal solution as long as you have an admissible heuristic.
  - Actually, it also needs to be consistent (satisfy triangle inequality)
    - h(x_i) <= C(x_i, x_j) + h(x_j); h(x_G) = 0

Good Heuristics

- Needs to be an underestimate & satisfy triangle inequality to guarantee we find optimal path

- The closer your heuristic is to actual estimate, the faster you will find the optimal path
  - *need fewer expansions*

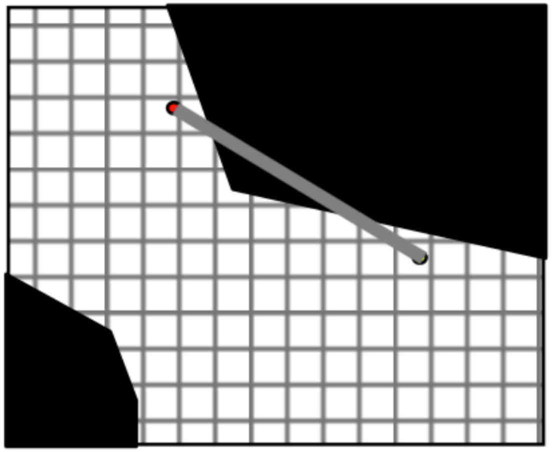
- The closer your heuristic is to 0, the slower your algorithm
  - *approaches Dijkstra; more expansions*

Admissible Heuristic

$h(x_i) = 0$ (Dijkstra)

$h(x_i)$ = Euclidean distance from x_i to x_G

$\ldots ?$



A*: Backtracking optimal path

$O = \{x_O\}$, $C = \{\}$

$V(x_O) = 0$, $V(x_i) = \infty$, $\mathbf{B(x_i)}$

repeat until x_G is in C

- x_j: vertex in O with lowest cost-to-come + heuristic
  - remove it from O and store it in C, closed list
- for each neighbor x_i of x_j not already in C
  - compute $V_{new} = C(x_j, x_i) + V(x_j)$
  - update $V(x_i)$ if it is more than V_{new} **and set $B(x_i) = x_j$**
  - if not in O, then add to O

Analyzing Algorithms

1. *Completeness*

2. *Optimality*

3. *Efficiency*

Analyzing Algorithms

1. *Completeness:* Algorithm is complete if it finds the optimal solution in finite time, if a solution exists. Else, it declares failure in finite time.

2. *Optimality:* Algorithm that finds a solution whose cost is the minimum (or maximum) possible cost.

3. *Efficiency:* An algorithm is efficient if it finds the solution in the least possible time (for all inputs).

Analyzing Algorithms

1. *Completeness:* DP, Dijkstra, A* are complete

2. *Optimality:* DP, Dijkstra, A* are optimal.

3. *Efficiency:*
   - DP is NOT efficient.
   - Dijkstra (and A*) is efficient if no heuristic.
   - A* is efficient (and Dijkstra is not) with any admissible heuristic.

Optimality vs. Efficiency

- Sometimes you want a “good-enough” solution as fast as possible.
- Can we trade-off optimality and efficiency?

Weighted A*

- **Dijkstra**
  - expand based on lowest $V(x_i)$

- **A***
  - expand based on lowest $V(x_i) + h(x_i)$

- **ϵ Weighted A***
  - expand based on lowest *???*

- **Dijkstra**
  - expand based on lowest $V(x_i)$
- **A***
  - expand based on lowest $V(x_i) + h(x_i)$
- **ϵ Weighted A***
  - expand based on lowest $V(x_i) + \epsilon\, h(x_i)$
  - $\epsilon >= 1$

Weighted A*

- ϵ >= 1

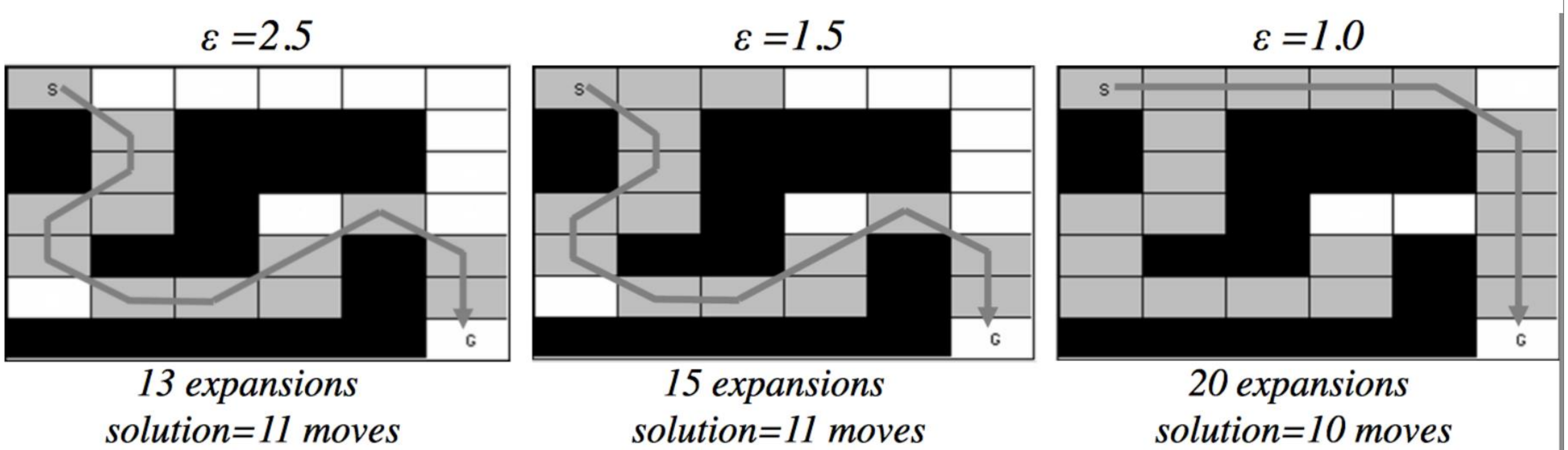
- Expand based on inflated heuristic
  - V(x_i) + ϵ h(x_i)

- *Can we guarantee optimality?*

- $\epsilon >= 1$
- Expand based on inflated heuristic
  - $V(x_i) + \epsilon\, h(x_i)$

- **Can we guarantee optimality?**
  - No!
  - However, we can guarantee that it will find a path whose cost is no more than ϵ times the minimum cost path.
  - $V'(x_G) <= \epsilon\, V^*(x_G)$

- $\epsilon >= 1$
- In practice, faster than A*. In fact, the higher we set ϵ to be, the faster we find the solution.



Anytime Algorithm

- If we stop the algorithm at any point in time, we should be able to return a good solution.
- More time we give an algorithm, the closer to optimal the returned solution should be.

- *Are DP, Dijkstra, A* anytime algorithms?*

Anytime Algorithm

If we stop the algorithm at any point in time, we should be able to return a good solution.

More time we give an algorithm, the closer to optimal the returned solution should be.

Are DP, Dijkstra, A anytime algorithms?*

- ***NO! We find the optimal path only at the last iteration.***

An anytime algorithm using weighted A*?

- set ϵ = very high number
- repeat until stopped
  - find path using weighted A*
  - $\epsilon = \epsilon/2$

- Works but we need to recompute path from scratch in every iteration. Speed up?