Local search

Announcements

- Assignment 0 due on Friday
- A1 coming next week
  - Please watch out for a team info

Local Search

- No search tree – just remember current state
  - No need for fringe = much less memory
  - Applicable to pathless problems (e.g. 8-queens)
    - Practical examples: integrated-circuit design, factory floor layout, telecommunications network optimization, and portfolio management
- Key idea: Try to find s that minimizes h(s)
  - Since h(goal) = 0

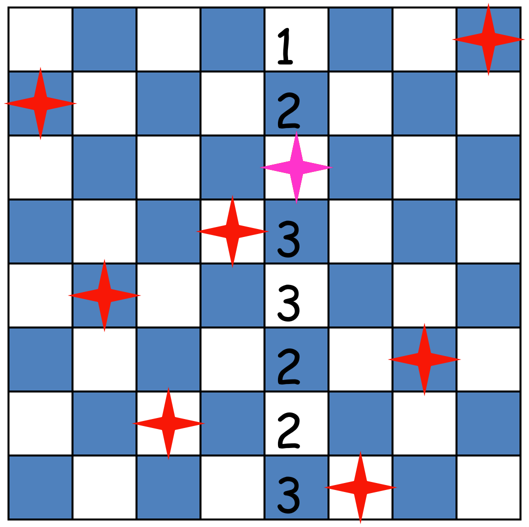
Steepest Descent

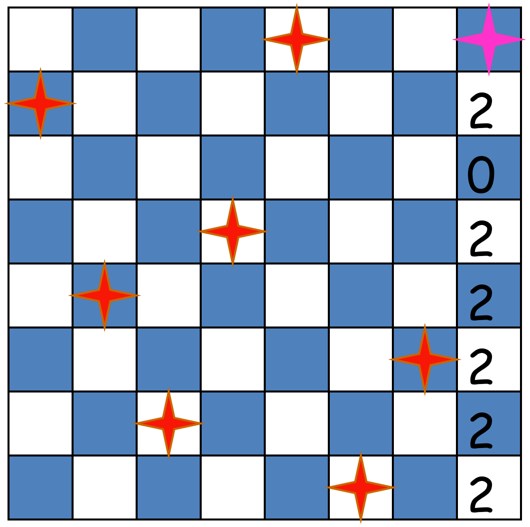
```
1. $S \leftarrow$ initial state
2. Repeat:
3.     $S' \leftarrow \arg\min_{S' \in SUCC(S)}\{h(S')\}$
4.     if GOAL?(S') return S'
5.     if $h(S') < h(S)$ then $S \leftarrow S'$ else failure
```

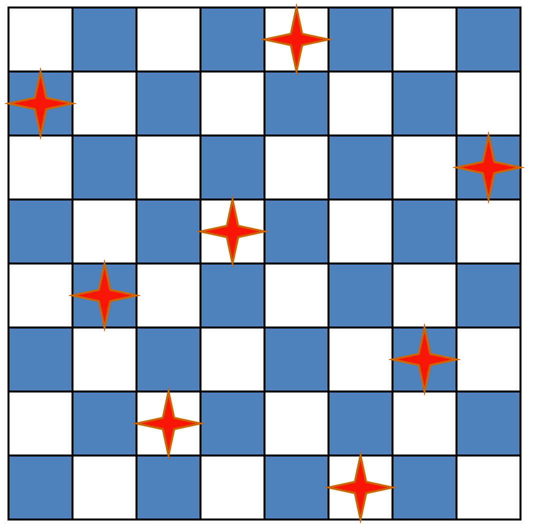
Application: 8-Queen

Repeat n times:

- Pick an initial state S at random with one queen in each column
- Repeat k times:
  - If GOAL?(S) then return S
  - Pick an attacked queen Q at random
  - Move Q in its column to minimize the number of attacking queens → new S [min-conflicts heuristic]
- Return failure



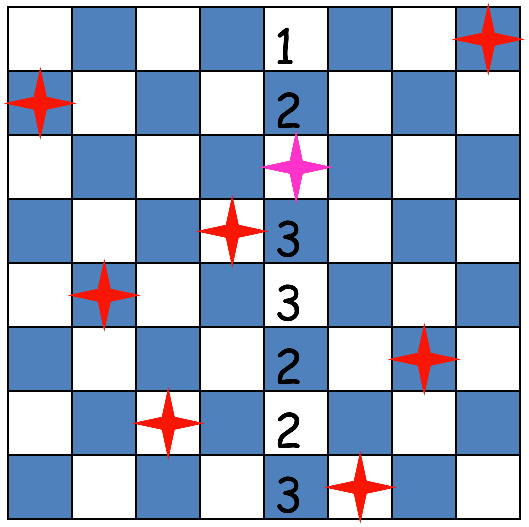


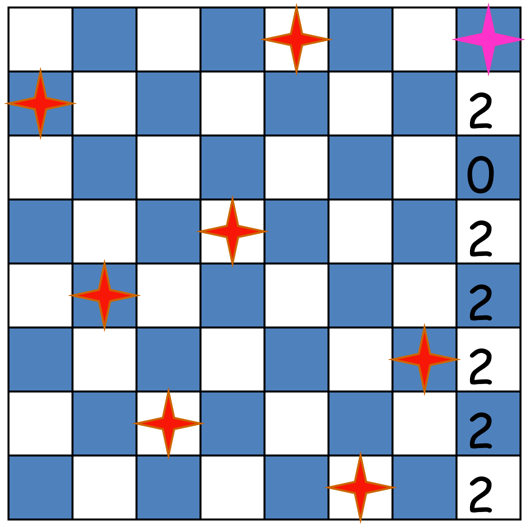


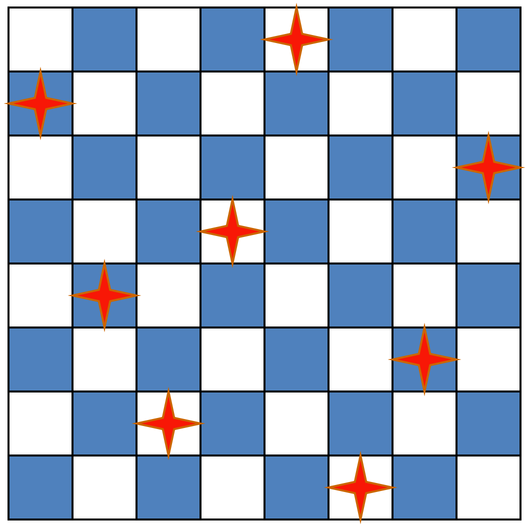
Application: 8-Queen

Repeat n times:

- Pick an initial state S at random with one queen in each column
- Repeat k times:
  - If GOAL?(S) then return S
  - Pick an attacked queen Q at random
  - Move Q in its column to minimize the number of attacking queens → new S [min-conflicts heuristic]
- Return failure



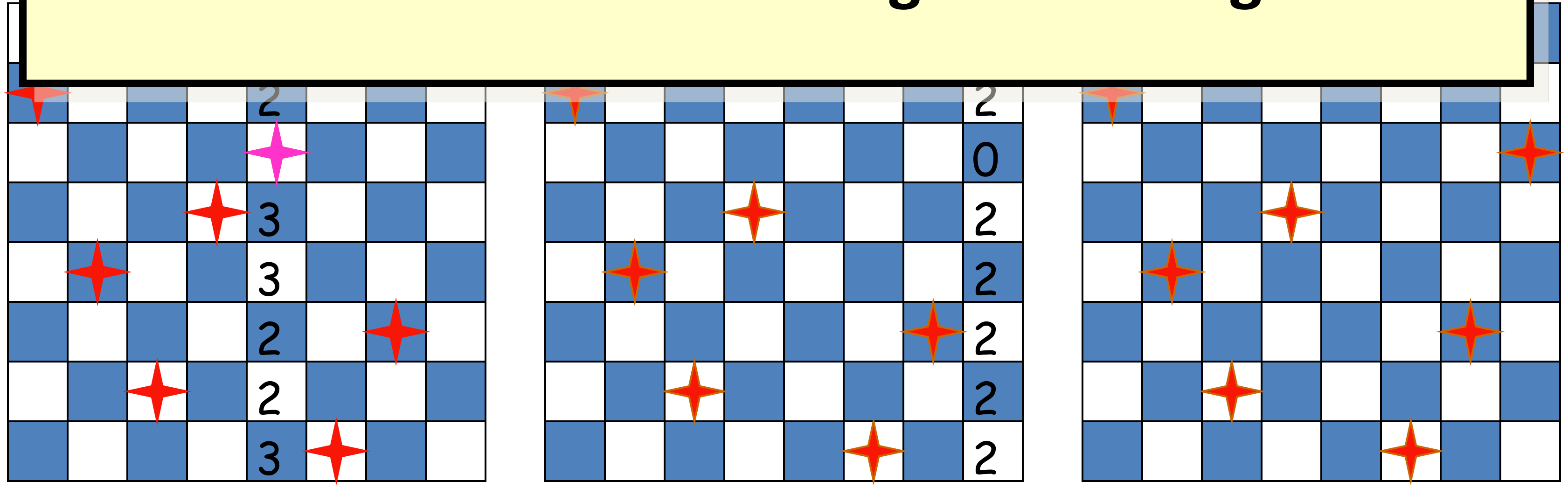




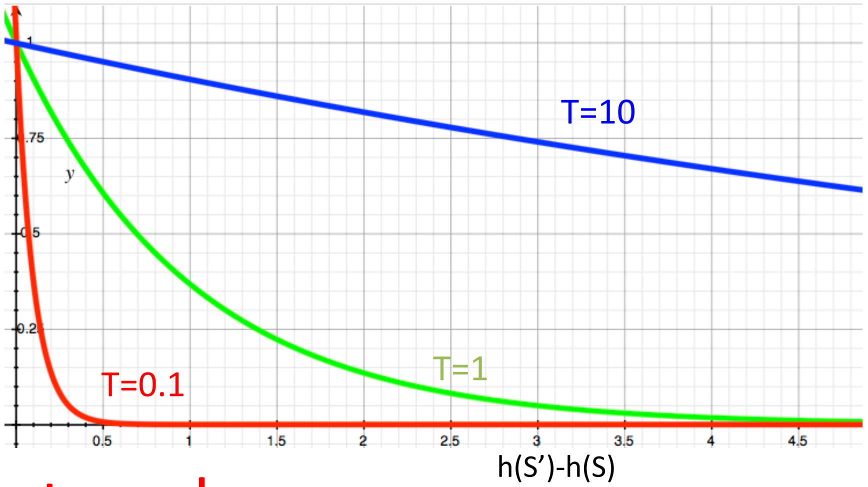
Application: 8-Queen

Why/when does it work well?

1) There are **many goal states** that are well-distributed over the state space
2) If no solution has been found after a few steps, it's **better to start it all over again**.
3) Building a search tree would be much less efficient because of the **high branching factor**



Monte Carlo Descent



```
1. S ← initial state
2. Repeat k times:
   – If GOAL?(S) then return S
   – S' ← successor of S picked at random
   – if h(S') ≤ h(S)  then S ← S'
     else with prob. exp(-(h(S')-h(S))/T) (for some T), S ← S'
3. Return failure
```

- **Simulated annealing** lowers T as k increases

Other local search techniques: Beam, genetic, branch-and-bound

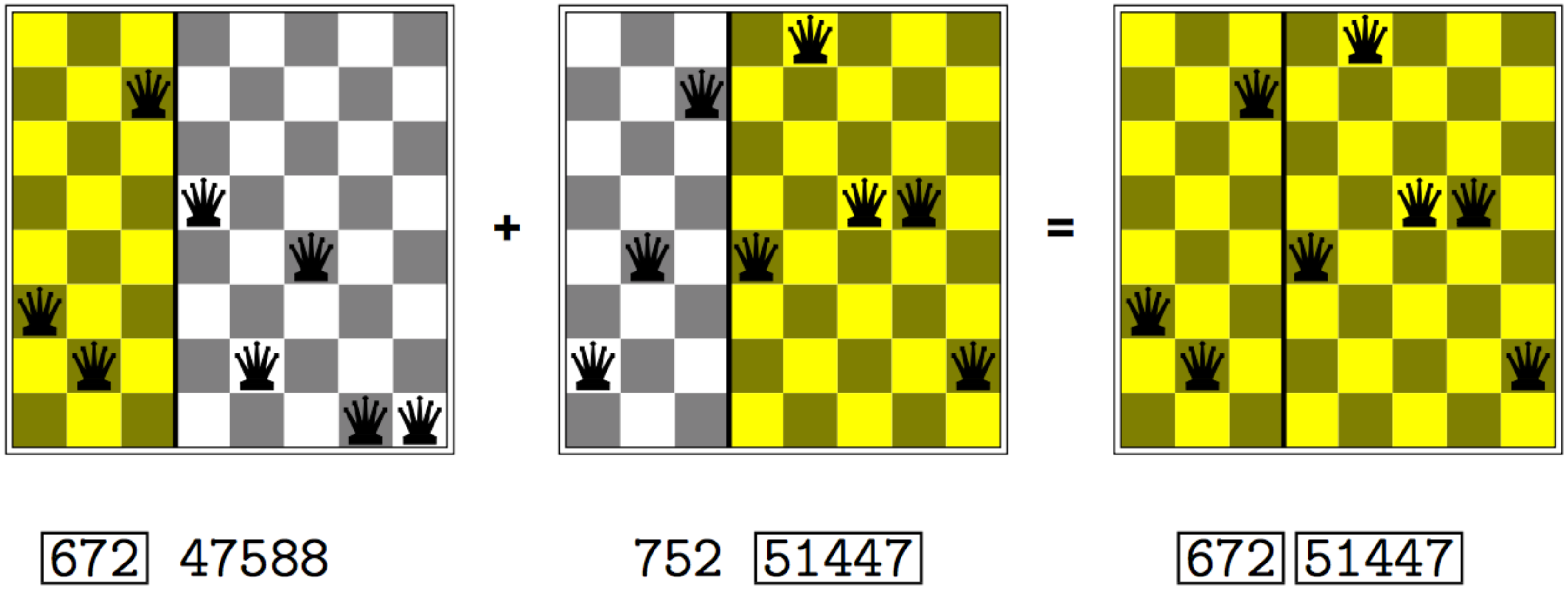
Beam search: Explore promising states in “parallel”

```
1. A ← Set of n randomly-generated states
2. Repeat k times:
    – A’ ← Successors of states in A
    – If there exists S in A’ such that GOAL?(S), then return S
    – A ← Subset of n best states in A’
3. Return failure
```

- Idea: Searches that find states will “recruit” other searches to join them

Genetic search: “Evolve” promising states

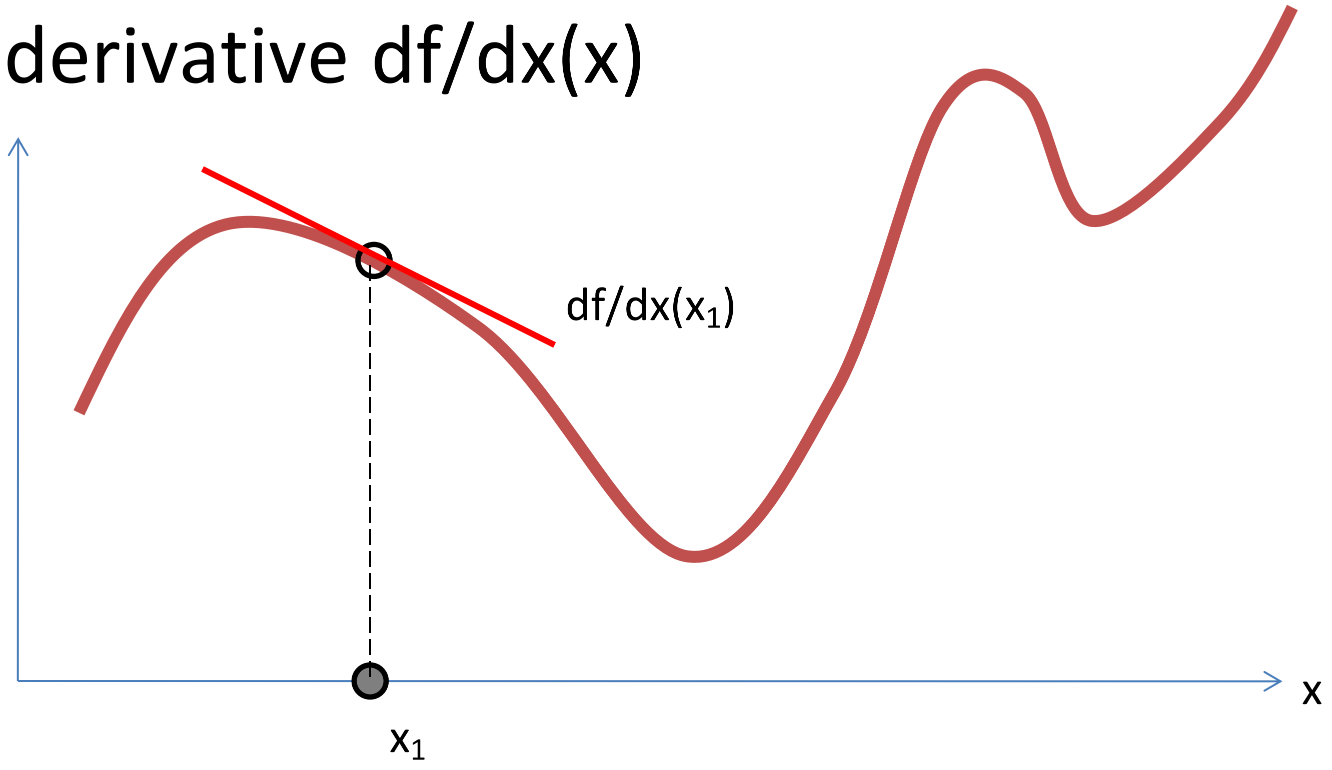
- Idea: Successors are generated by combining *pairs* of promising states. Most promising “offspring” states are kept.



Local search in continuous spaces: Gradient descent

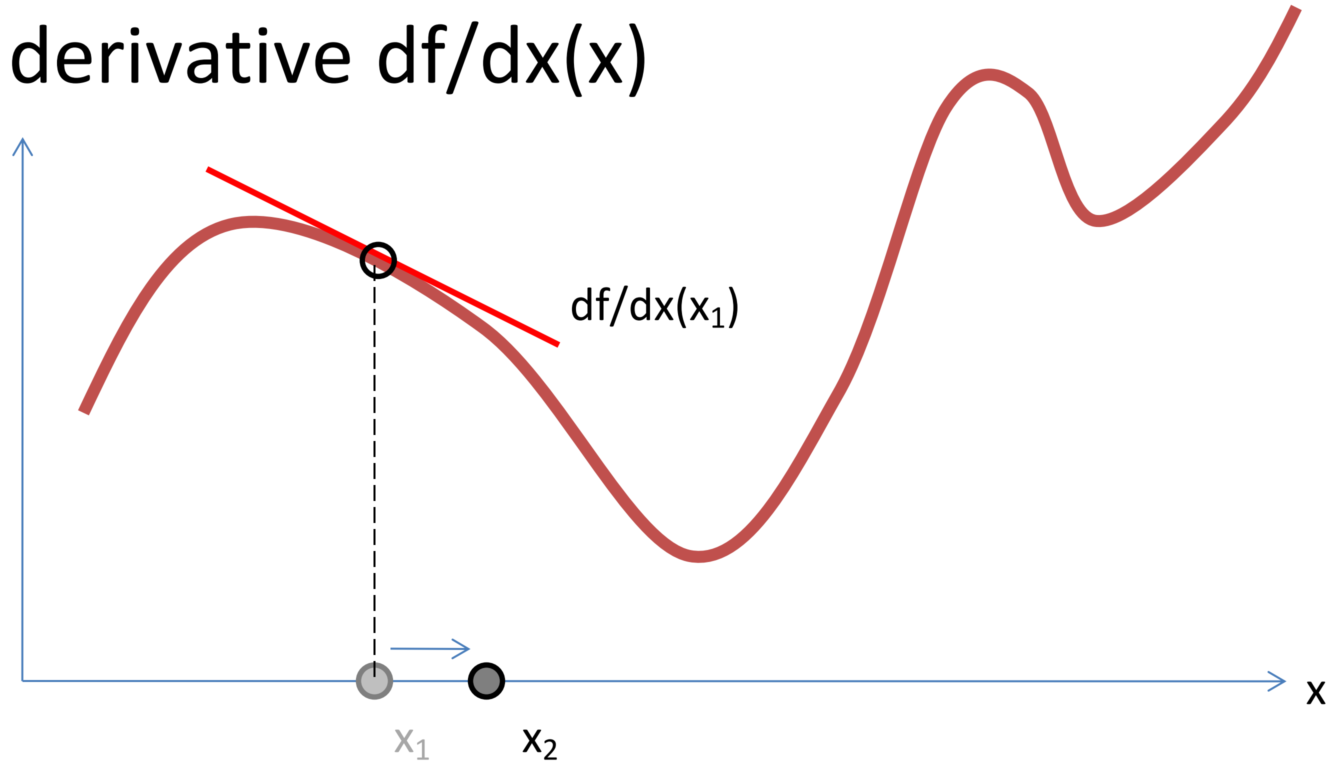
Gradient Descent in Continuous Space

- Minimize y=f(x)
- Move in opposite direction of derivative df/dx(x)



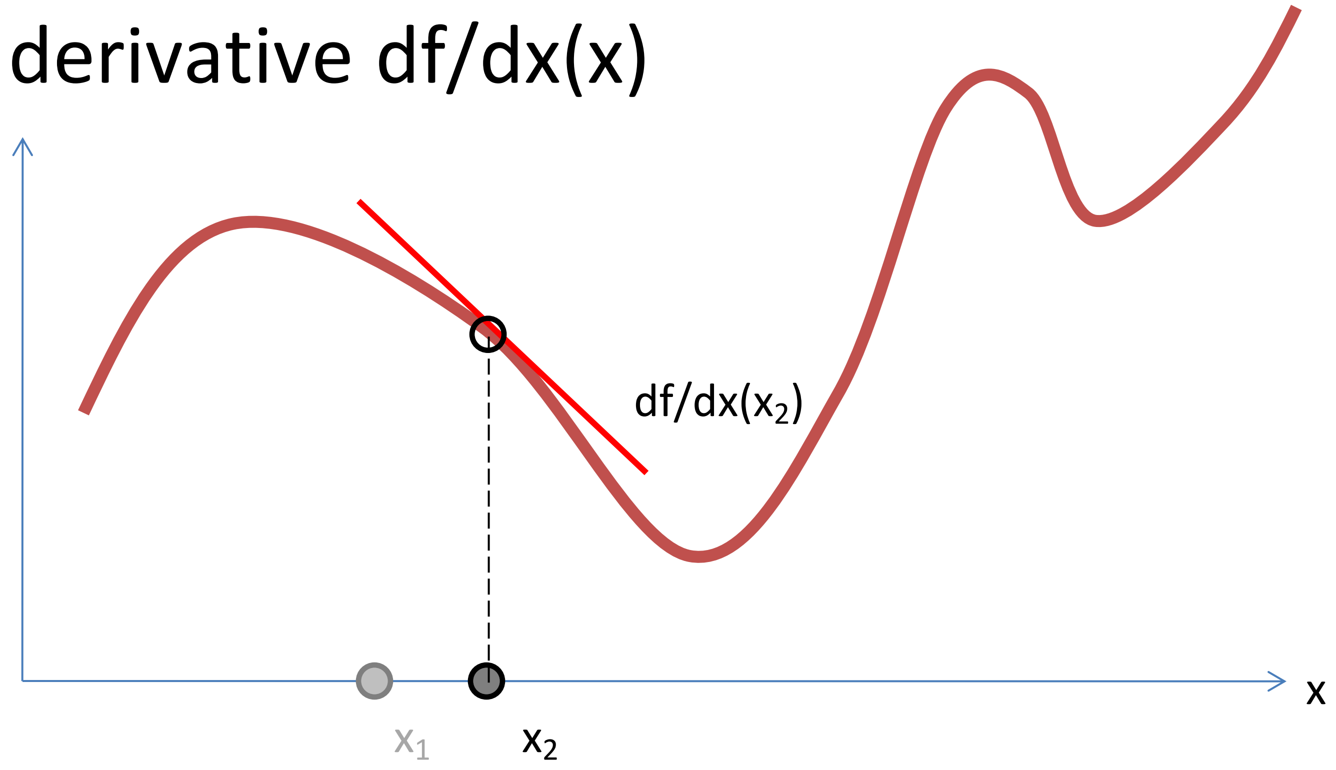
Gradient Descent in Continuous Space

- Minimize y=f(x)
- Move in opposite direction of derivative df/dx(x)



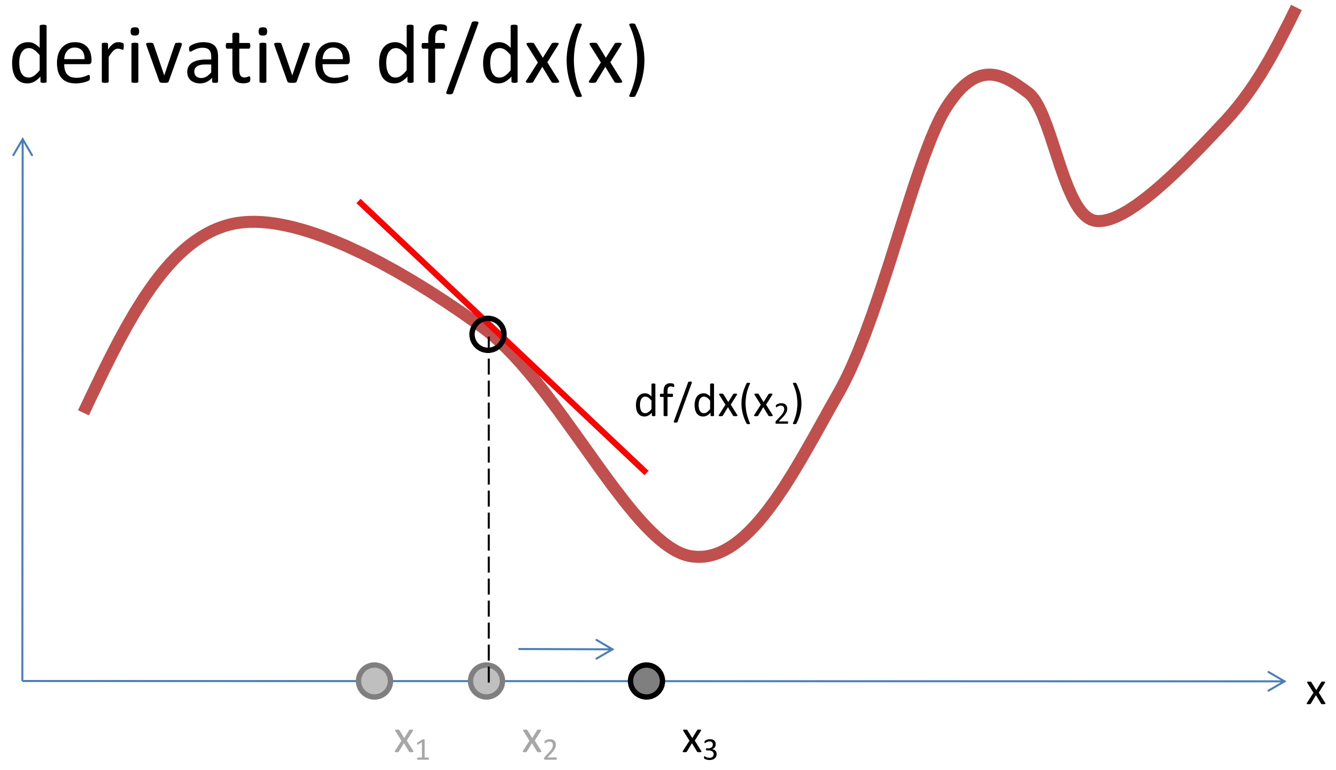
Gradient Descent in Continuous Space

- Minimize y=f(x)
- Move in opposite direction of derivative df/dx(x)



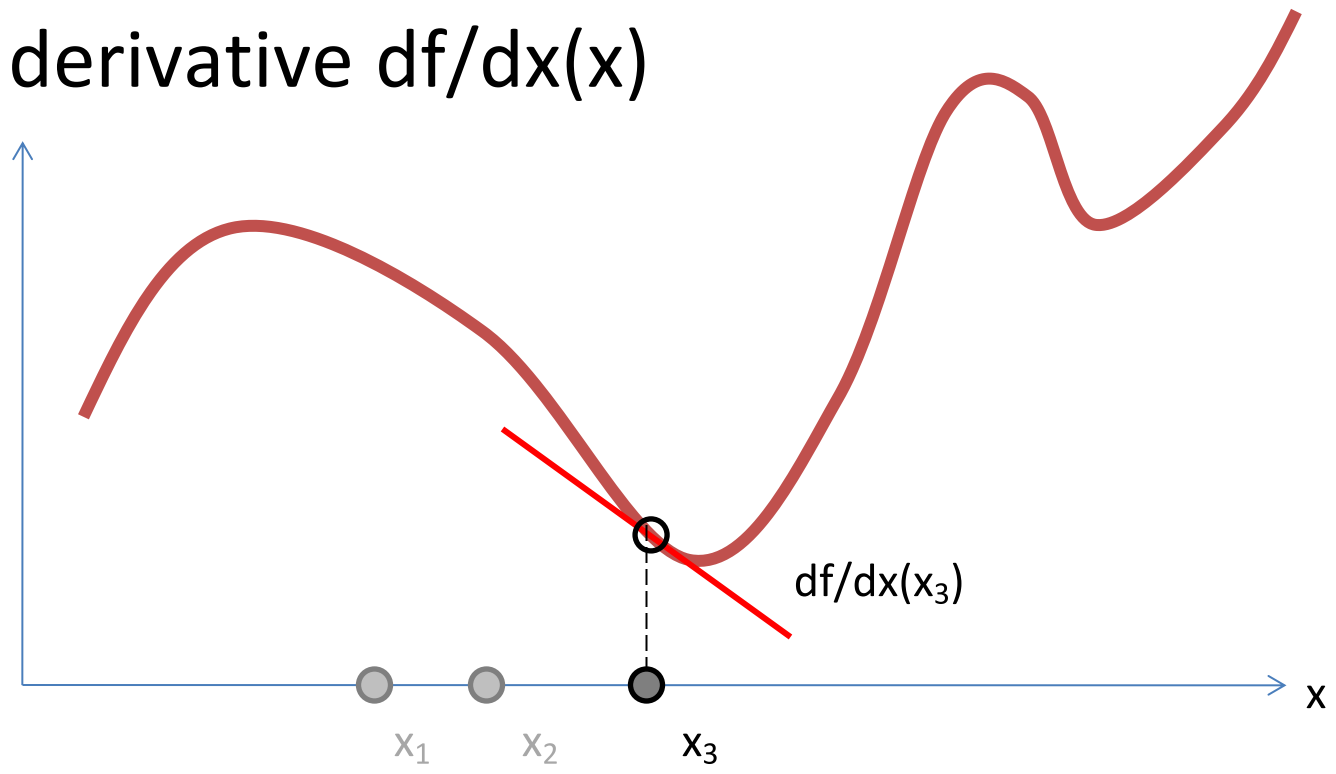
Gradient Descent in Continuous Space

- Minimize y=f(x)
- Move in opposite direction of derivative df/dx(x)



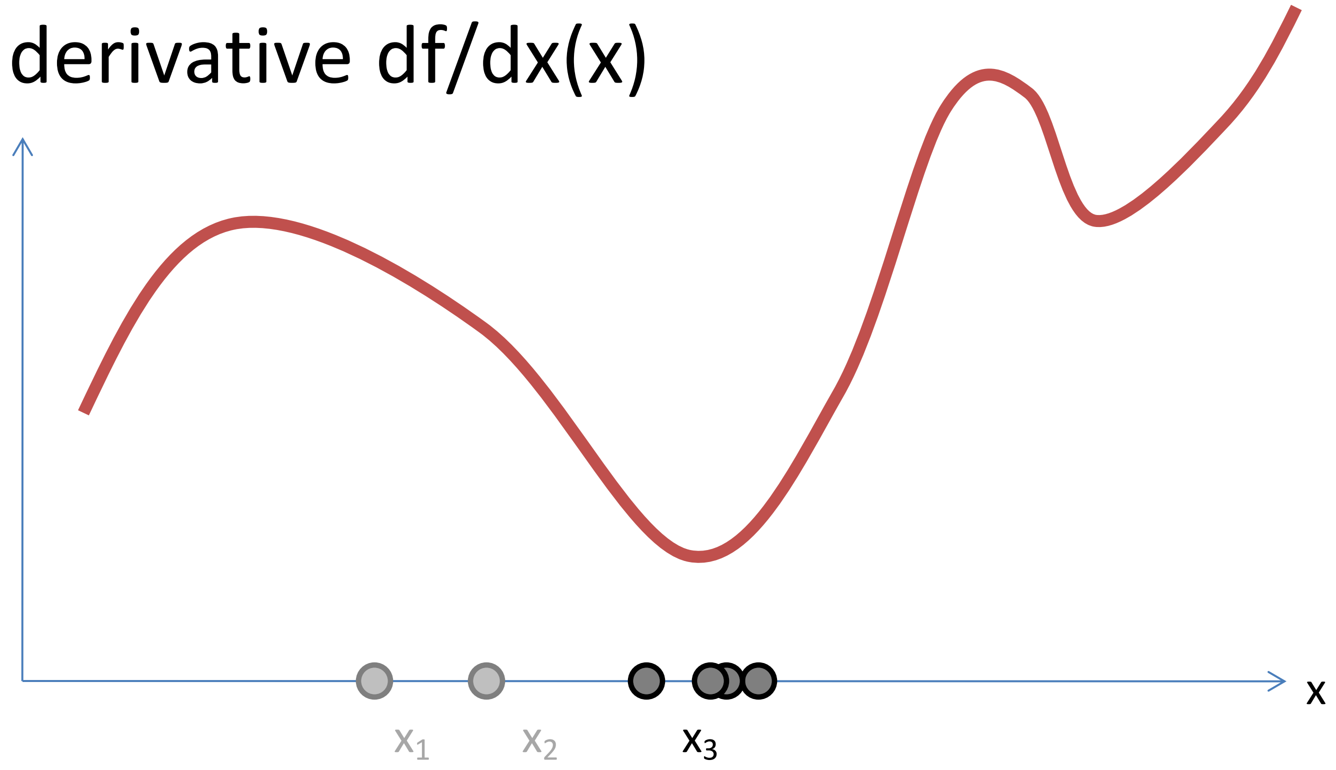
Gradient Descent in Continuous Space

- Minimize y=f(x)
- Move in opposite direction of derivative df/dx(x)



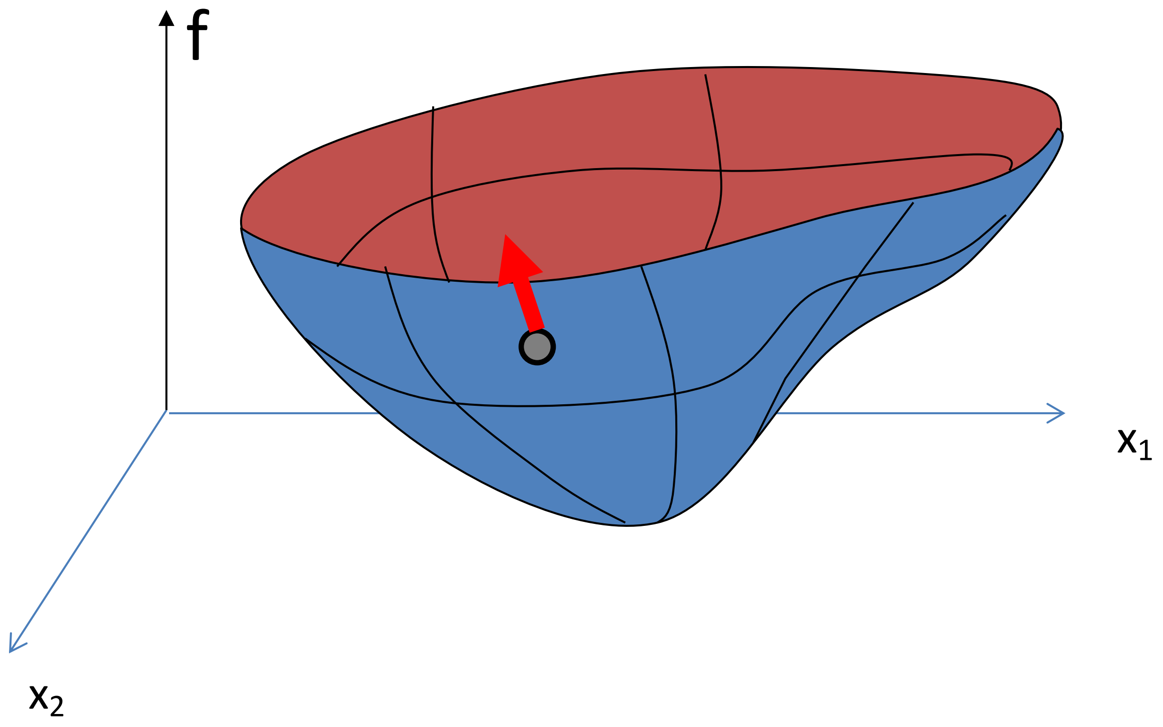
Gradient Descent in Continuous Space

- Minimize y=f(x)
- Move in opposite direction of derivative df/dx(x)

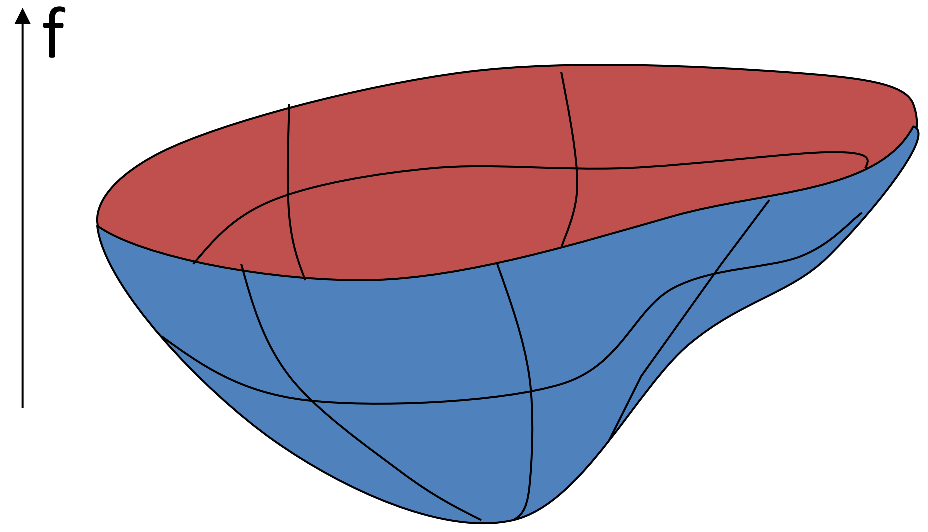


Gradient: analogue of derivative in multivariate functions $f(x_1,\ldots,x_n)$

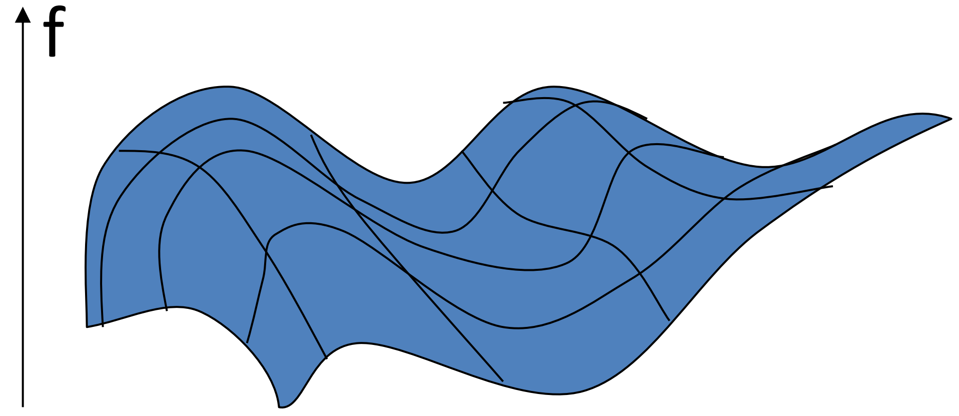
Direction that you would move $x_1,\ldots,x_n$ to make the steepest increase in f



GD works well



GD works poorly



Algorithm for Gradient Descent

- Input: continuous *objective function* f, initial point $\mathbf{x}^0=(x_1^0,\ldots,x_n^0)$

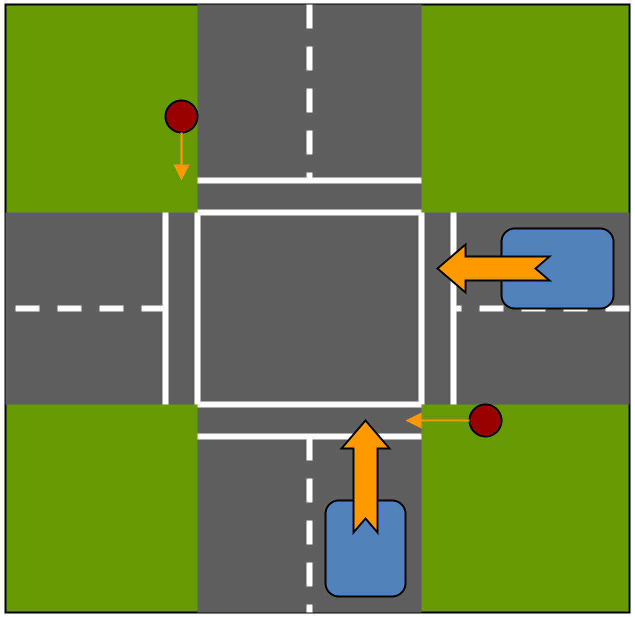
```
1. For t=0, …, N-1:
2.     Compute gradient $\mathbf{g}^t=(\partial f/\partial x_1(\mathbf{x}^t),\ldots,\partial f/\partial x_n(\mathbf{x}^t))$
3.     If length of $\mathbf{g}^t$ is small enough, return $\mathbf{x}$
4.     Pick a step size $\alpha^t$
5.     Let $\mathbf{x}^{t+1}= \mathbf{x}^t - \alpha^t \mathbf{g}^t$
6. Return failure
```

When will this work? What can go wrong?

Online search

How to handle imperfect observations?

- Classical search assumes that:
  - World states are perfectly observable,
    $\rightarrow$ the current state is exactly known
  - Action representations are perfect,
    $\rightarrow$ states are exactly predicted
- How an agent can cope with adversaries, uncertainty, and imperfect information?



Distance, speed, acceleration?

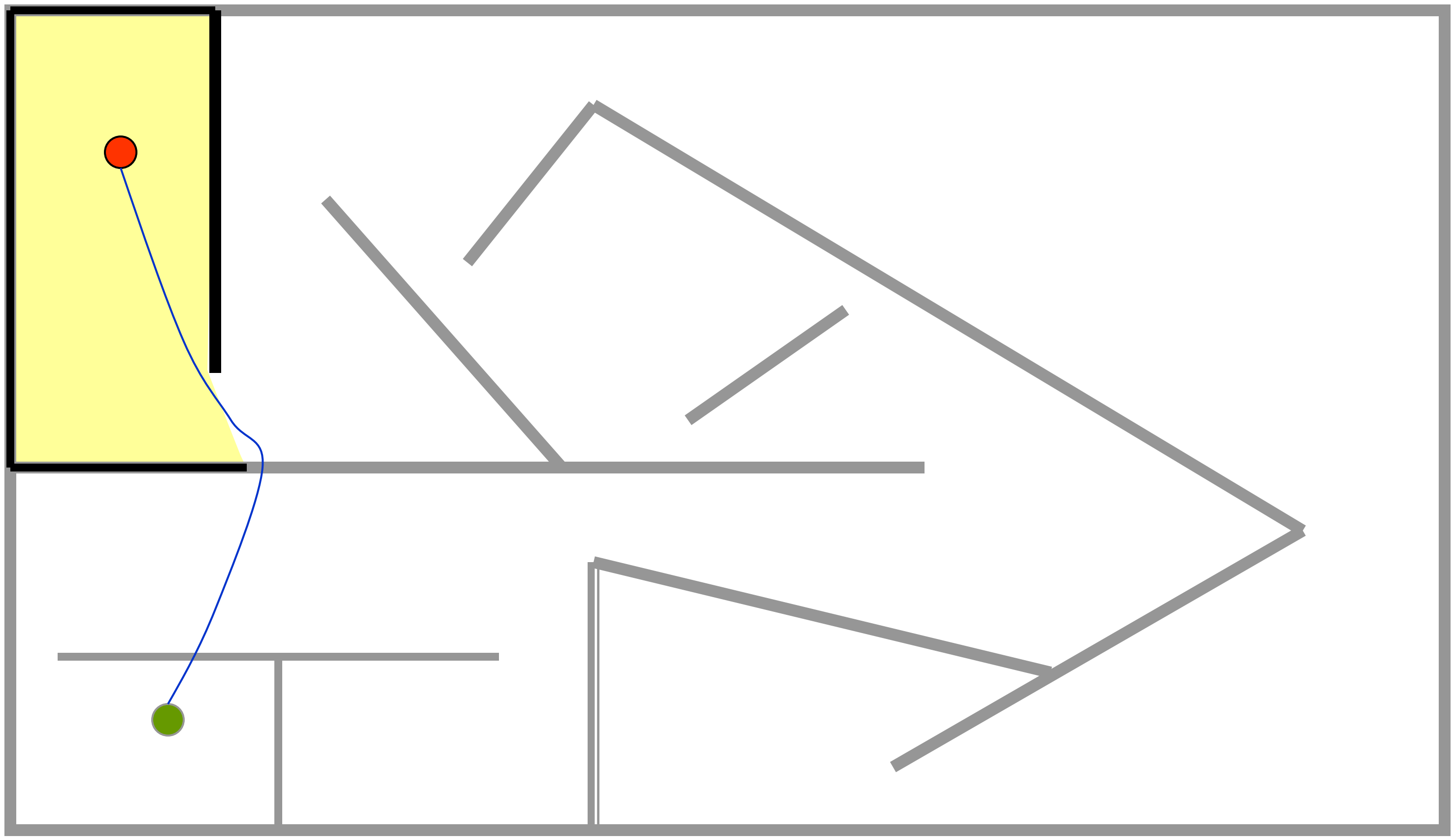
Intent?

Personality?

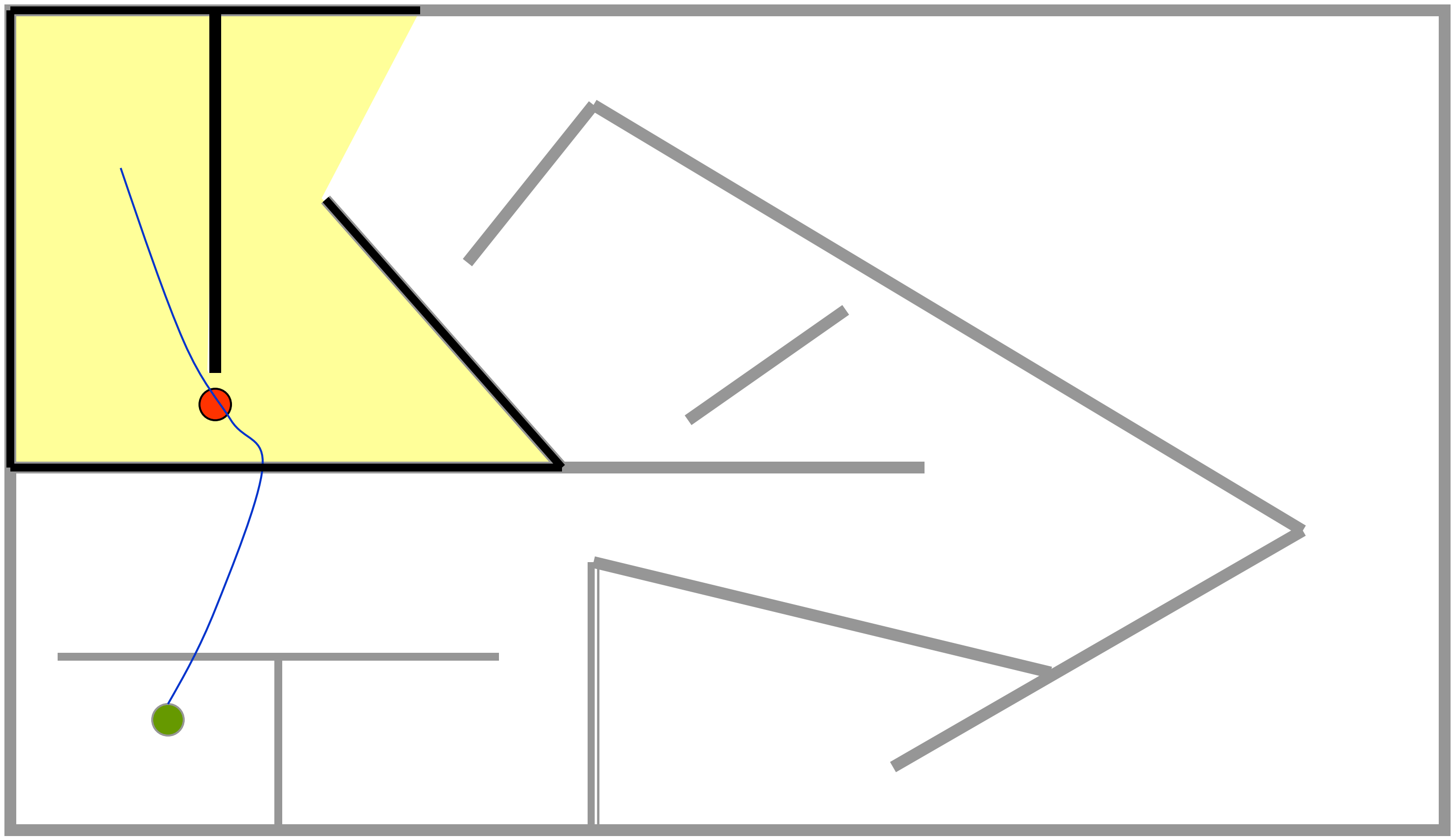
Online Search

- On-line search: repeatedly observe effects, and replan
  - A proactive approach for planning
  - A reactive approach to uncertainty
- Example: A robot must reach a goal position. It has no prior map of the obstacles, but its vision system can detect all the obstacles visible from the robot's current position

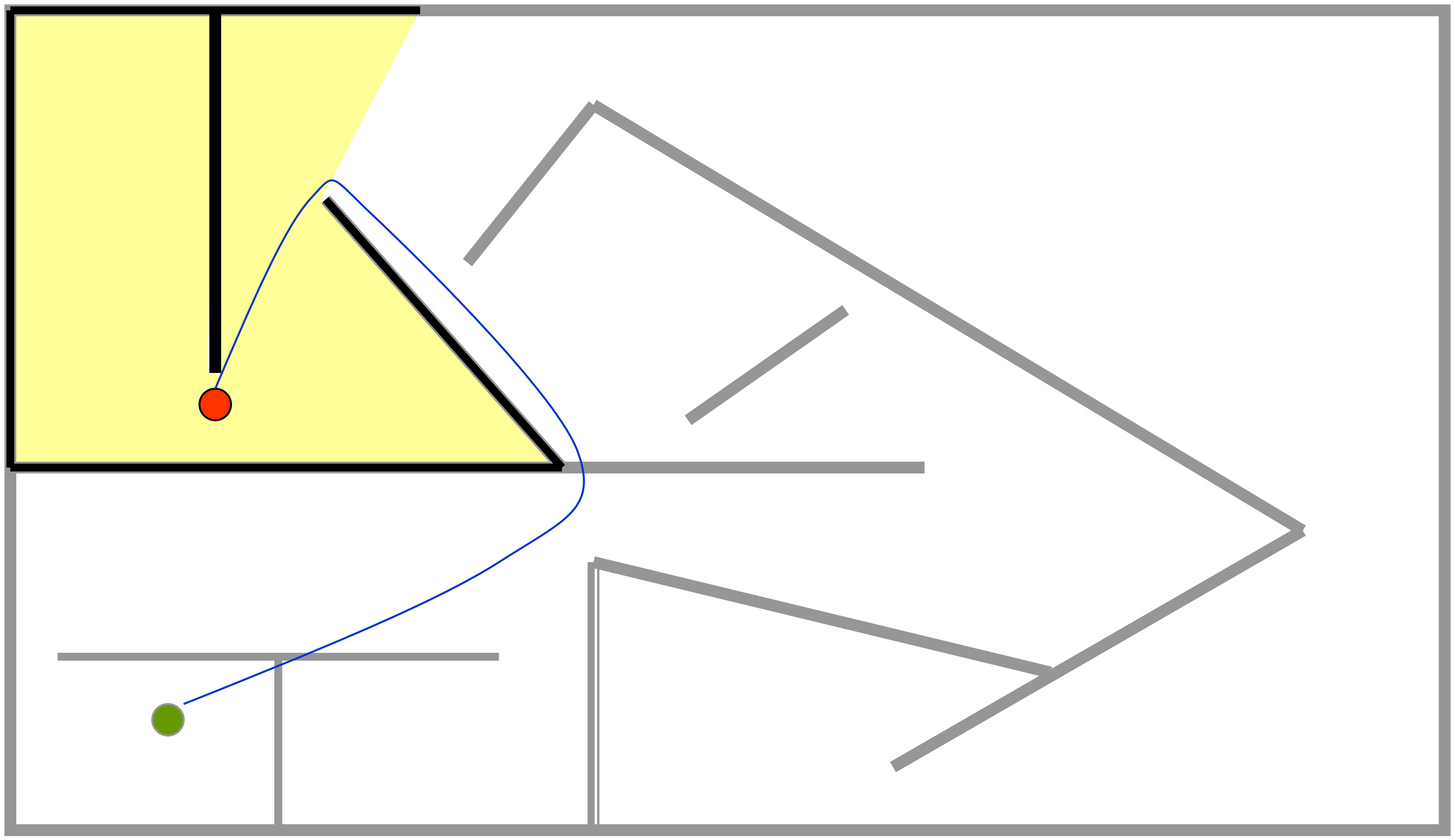
Assuming no obstacles in the unknown region and taking the shortest path to the goal is similar to searching with an admissible (optimistic) heuristic



Assuming no obstacles in the unknown region and taking the shortest path to the goal is similar to searching with an admissible (optimistic) heuristic



Assuming no obstacles in the unknown region and taking the shortest path to the goal is similar to searching with an admissible (optimistic) heuristic



Just as with classical search, on-line search may detect dead-ends and move to a more promising position

Next class

- Adversarial search and game playing