Local Search, Part 3

Lecture 10

Chapter 4, Sections 4.2-4.4

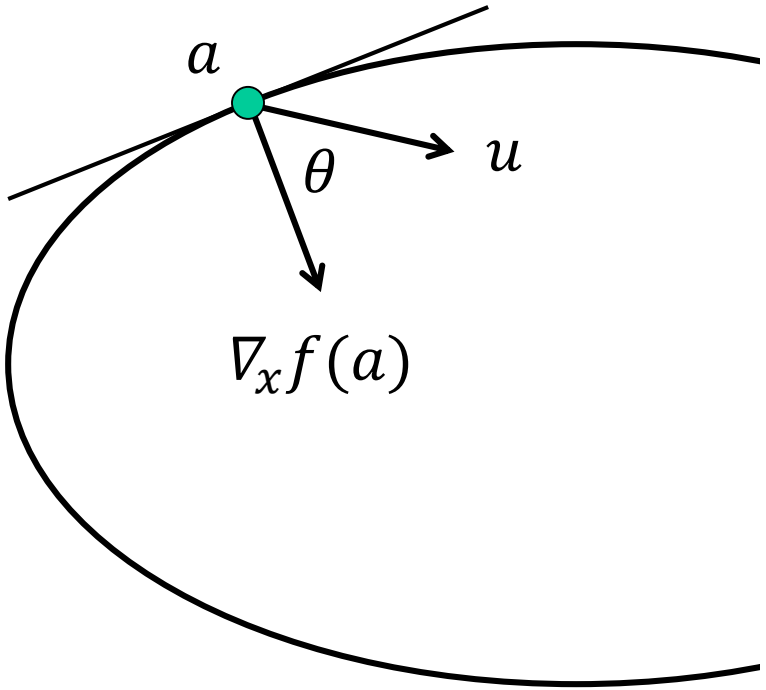
Jim Rehg

College of Computing

Georgia Tech

February 5, 2016

Why is the Gradient the Direction of Steepest Descent?



Local model for $f(x)$ at $x=a$:

$$\tilde{f}(x) = f(a) + \frac{\partial f}{\partial x_1}(a)(x_1 - a_1) + \cdots + \frac{\partial f}{\partial x_n}(a)(x_n - a_n)$$

Directional Derivative along u at $x=a$:

$$D_u f(x)|_{x=a} = \lim_{h \to 0} \frac{f(a+hu) - f(a)}{h}$$

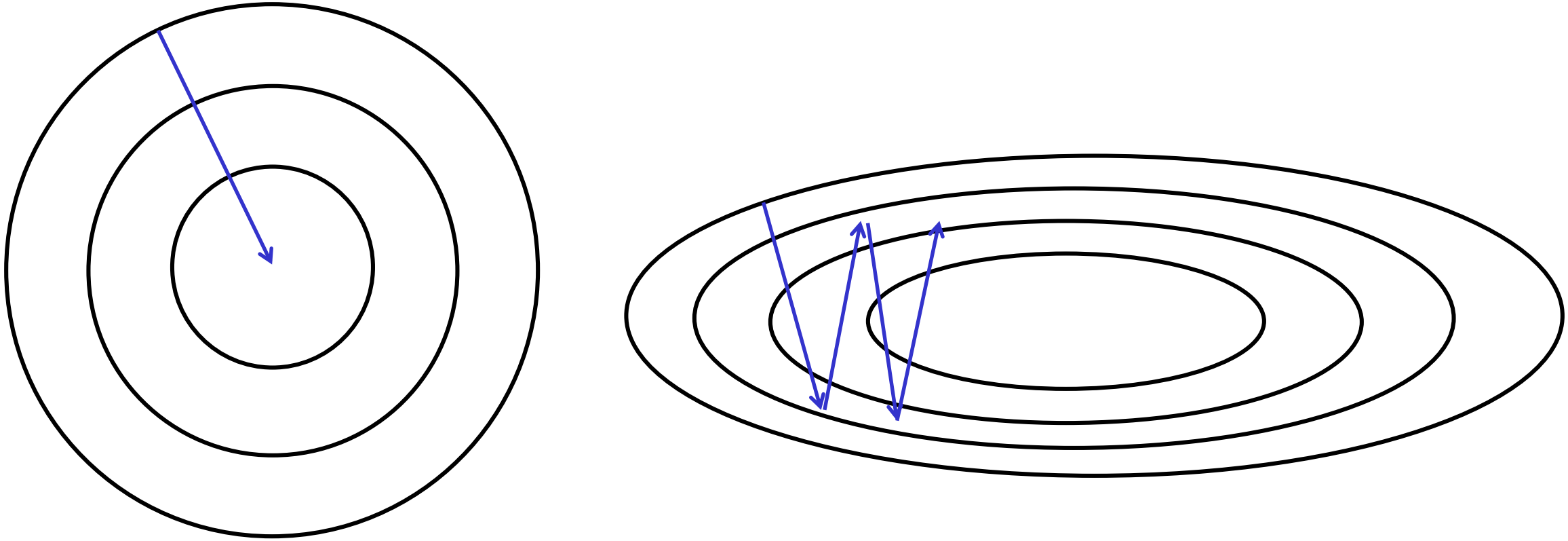
$$= \lim_{h \to 0} \frac{\tilde{f}(a+hu) - f(a)}{h} = \lim_{h \to 0} \frac{\frac{\partial f}{\partial x_1}(a)hu_1 + \cdots + \frac{\partial f}{\partial x_n}(a)hu_n}{h}$$

$$= \frac{\partial f}{\partial x_1}(a)u_1 + \cdots + \frac{\partial f}{\partial x_n}(a)u_n = \nabla_x f(a) \cdot u$$

$$\nabla_x f(a) \cdot u = \nabla_x f(a) \cos\theta$$

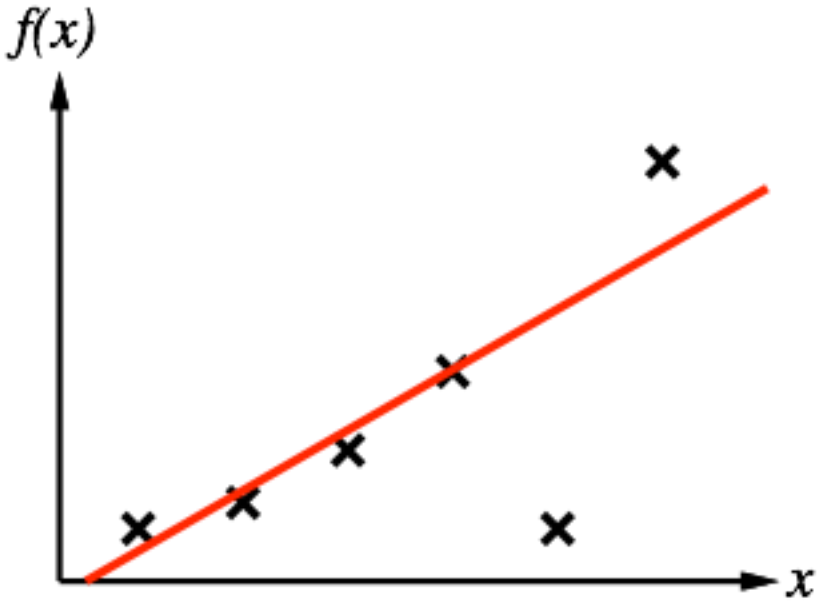
Maximized when $u = \nabla_x f(a)$

Gradient Descent – Shape of Energy Landscape



Gradient descent can get bogged down in narrow valleys, bouncing between walls while making slow forward progress

Normal Equations for Linear Regression



$$\theta = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

N data points

Rows of x values

Target y values

Since closed-form solution exists, why use gradient descent?

Motivations for Using Gradient-based Local Search

Motivations for Using Gradient-based Local Search

Data set may not fit in memory

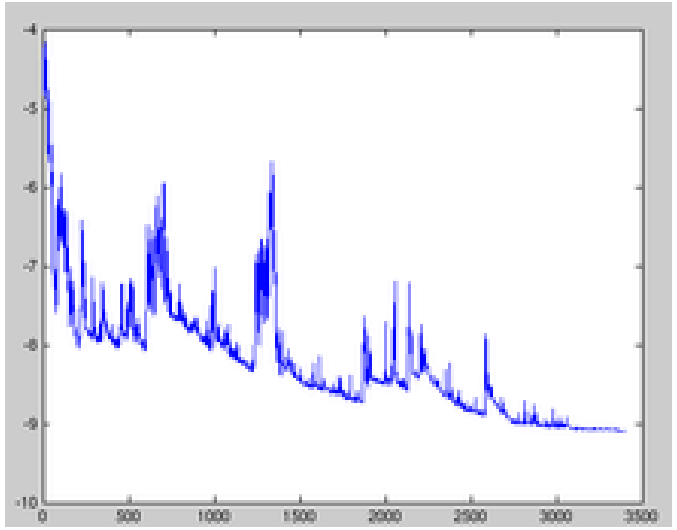
Matrix inversion can be expensive for large N

Stochastic Gradient Descent

- Randomly sample a subset of data points (mini-batch)
- Perform gradient descent using the sample only
- Repeat

Extreme version:

- Take a step from a single data point



Simulated Annealing

Compute the local search step

Accept that step with probability 1-T

Otherwise, pick a neighbor at random

Deliberate strategy to escape local minima

Slowly reduce T over time (annealing)

Can be efficient for discrete problems

Use stochastic gradient methods for continuous problems

Gradient Descent with Random Restarts

Pick a starting point at random

Find local minima

Repeat

Choose the minimum among the minima

Surprisingly effective method for complex energy landscapes

Can often beat more complex methods

Heuristics can be used to bias towards good starting points

Non-Deterministic Actions

Actions are no longer guaranteed to produce the desired output

Different cases:

- Actuation error – Output is offset from target goal by some delta
- Actuation failure – No action taken or wrong action taken

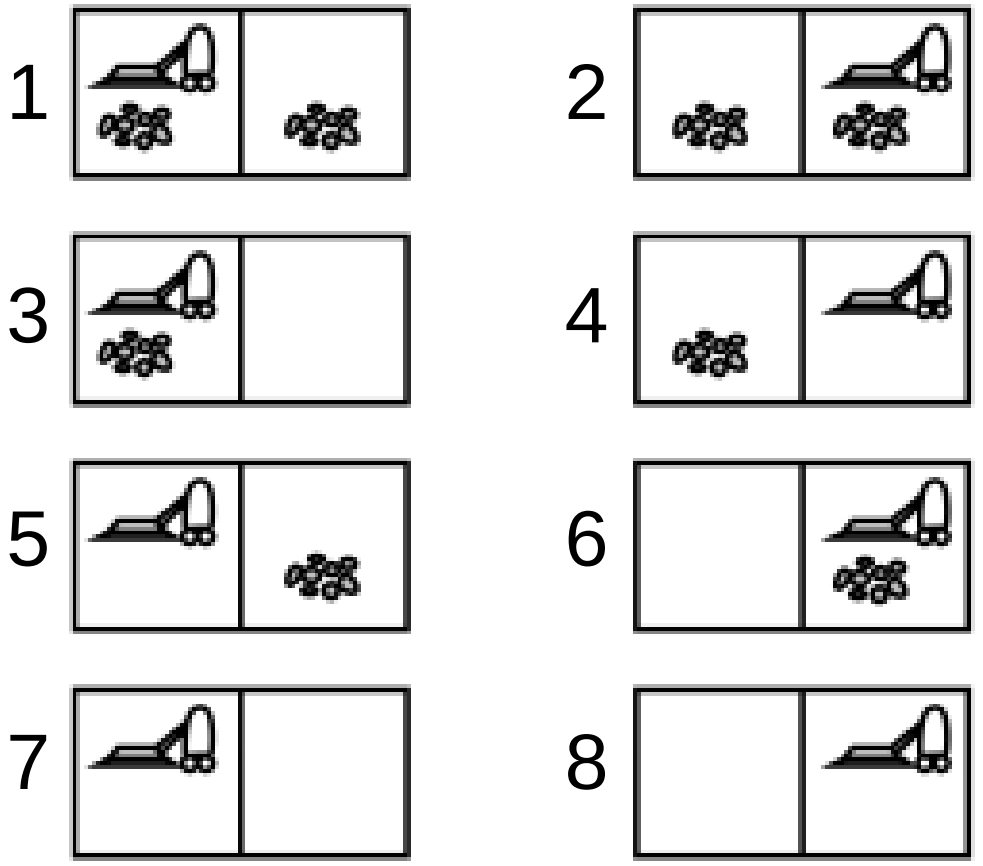
The solution produced by search is a contingency plan, specifying what to do under different conditions.

We use sensing (percepts) to determine which action (among multiple contingencies) to take during execution

Erratic Vacuum Example

- Sometimes the action “suck” in a dirty square cleans an adjacent square as well (Transition from 1 to 7)
- Sometimes “suck” deposits dirt when applied to a clean square (Transition from 5 to 1)

- Use *AND-OR search tree* to solve these problems



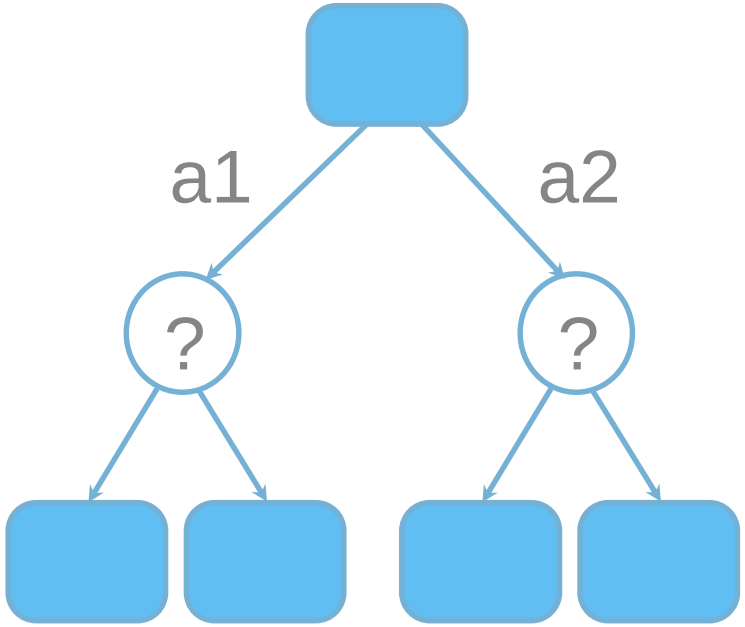
AND-OR Search

Alternating rows of action (OR) and chance (AND) nodes

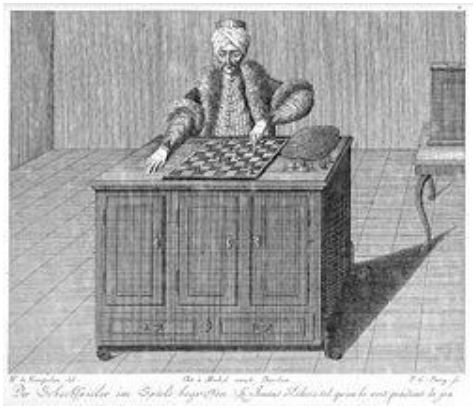
OR nodes – Agent chooses best action (as before)

AND nodes – Execution of chosen action is non-deterministic (random), multiple successors are possible

Searching the tree produces a strategy that is resilient to non-determinism



Building AI Agents to Play Games – Long History



1770: Kempelen -- Mechanical Turk

1912: Zermelo -- paper that first mentions an algorithm like Minimax (for optimal game play)

1951: Turing -- first chess program

1952: Samuel -- uses ML to improve

1956: McCarthy -- pruning to allow deeper search with efficiency

1997: Deep Blue beats Kasparaov with great evaluation functions

Online Search

Offline Search:

Simulate the world and search for a complete sequence of actions to achieve the goal, then execute them

Online Search:

Interleave search (planning) and execution

Search to determine an action,

execute action,

search again, …

When to Use On-Line Search?

In a dynamic environment,

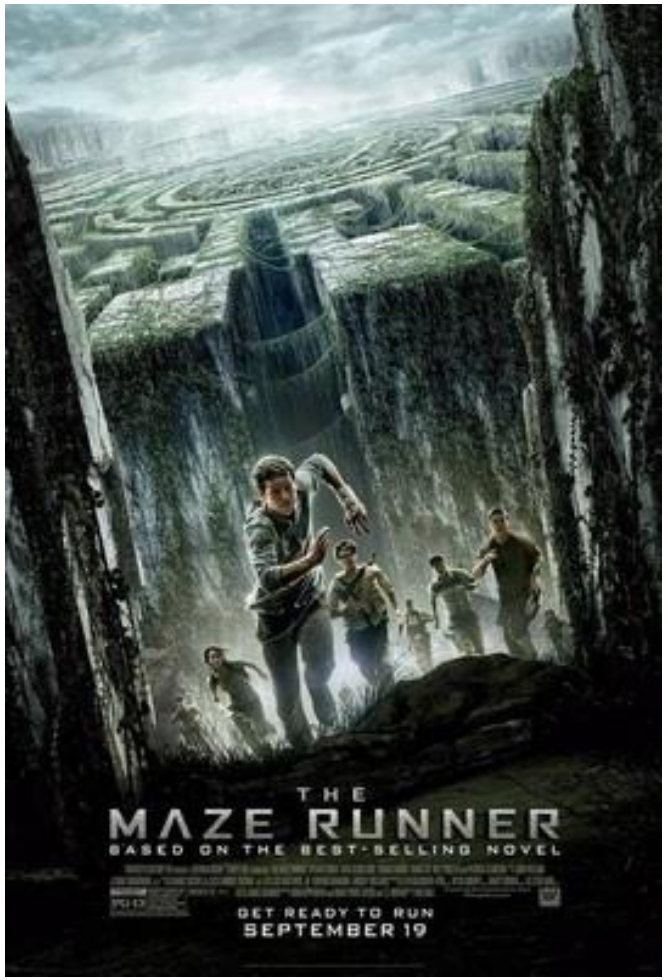
things change too quickly for a static plan

In a non-deterministic environment,

deal with what actually happens rather than planning for all contingencies

In an unknown environment,

affect of actions is unknown until they are executed



Elements of On-Line Search

Actions(s) – Returns the allowable action choices for state s

Goal-Test(s) – Determines whether s is a goal state

C(s,a,s') – Cost of executing (allowable) action a from state s, resulting in state s'

Issues in Online Search

Unpredictable consequences of actions

- Irreversible actions
- Dead ends

Environment has to be safely explorable

Learning Real-Time A* (LRTA*)

Guaranteed to find a goal in any finite safely-explorable environment

Learns Result(s,a) through experience

Updates heuristic function H(s) based on experience, try to avoid being stuck in local minima

Always chooses the lowest cost action

Connections to reinforcement learning

Questions?