Stochastic Local Search

Computer Science cpsc322, Lecture 15

(Textbook Chpt 4.8)

May, 30, 2017

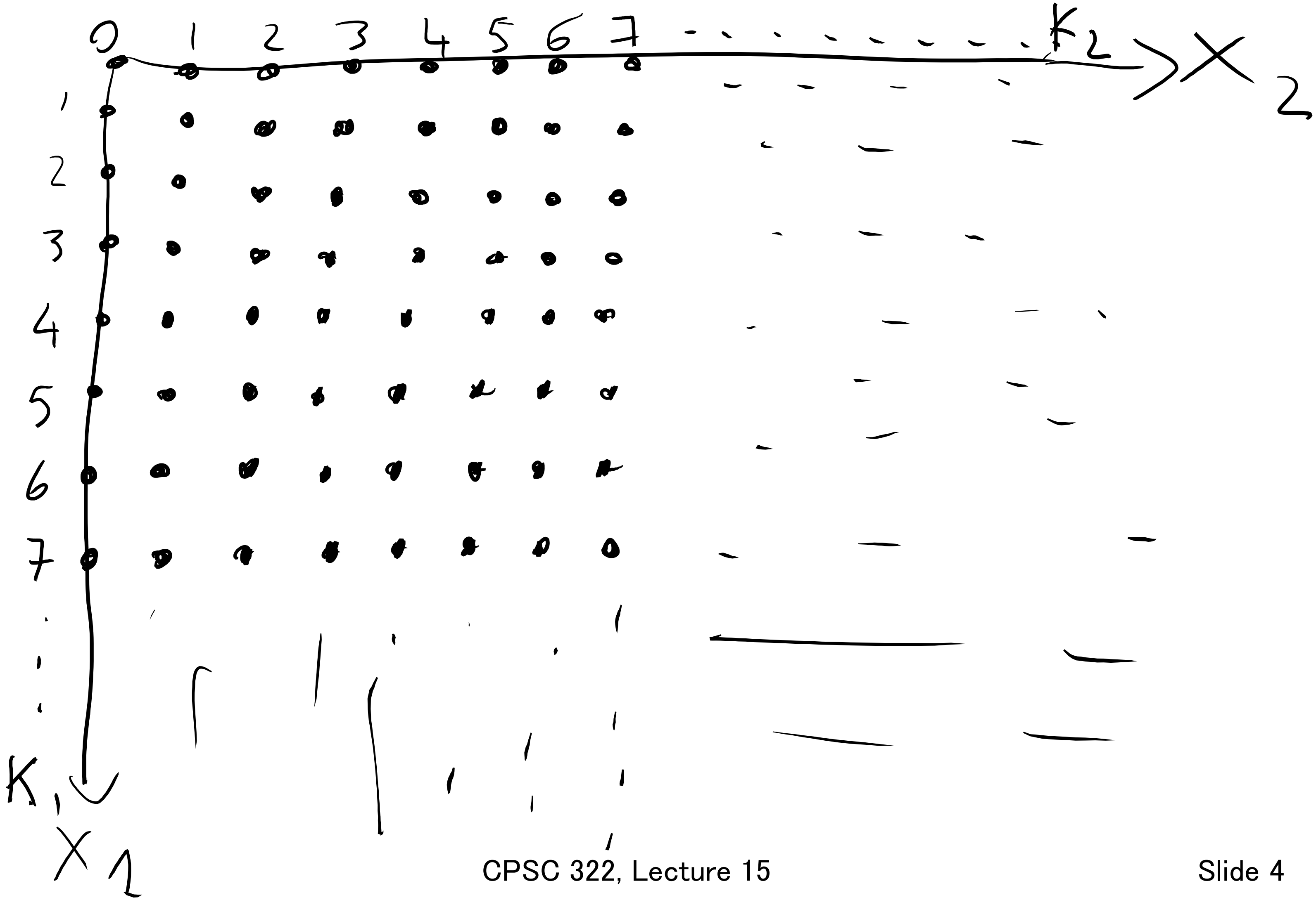
Lecture Overview

- **Recap Local Search in CSPs**
- Stochastic Local Search (SLS)
- Comparing SLS algorithms

Local Search: Summary

- A useful method in practice for large CSPs
  - Start from a possible world (randomly chosen)
  - Generate some neighbors ("similar" possible worlds)
    e.g. differ from current poss. world only by one variable's value
  - Move from current node to a neighbor, selected to minimize/maximize a scoring function which combines:
    - ✓ Info about how many constraints are violated/satisfied
    - ✓ Information about the cost/quality of the solution (you want the best solution, not just a solution)

$X_1 = \{0, \ldots, k_1\}$ $\quad$ $X_2 = \{0, \ldots, k_2\}$



Hill Climbing

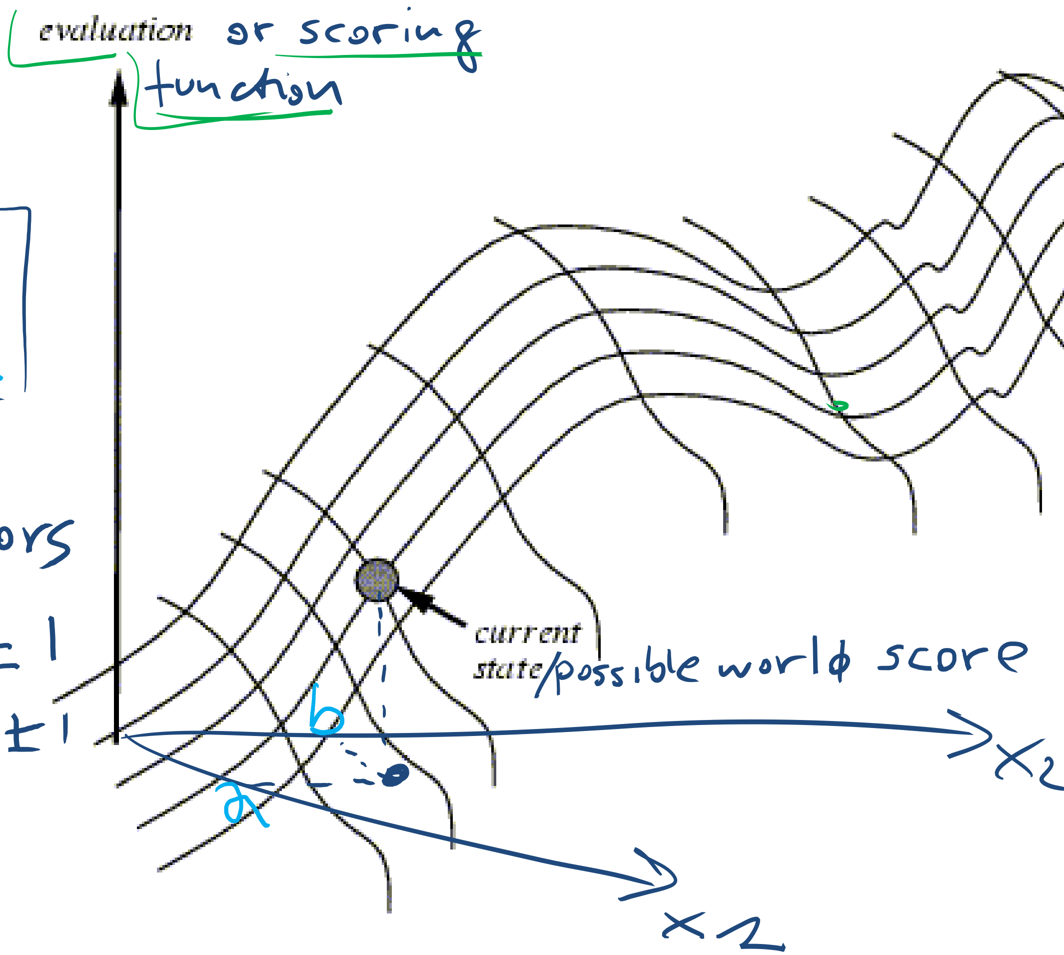
NOTE: Everything that will be said for Hill Climbing is also true for Greedy Descent

current

$X_1 = a$
$X_2 = b$

4 neighbors

$X_1 = a \pm 1$
$X_2 = b \pm 1$



two vars

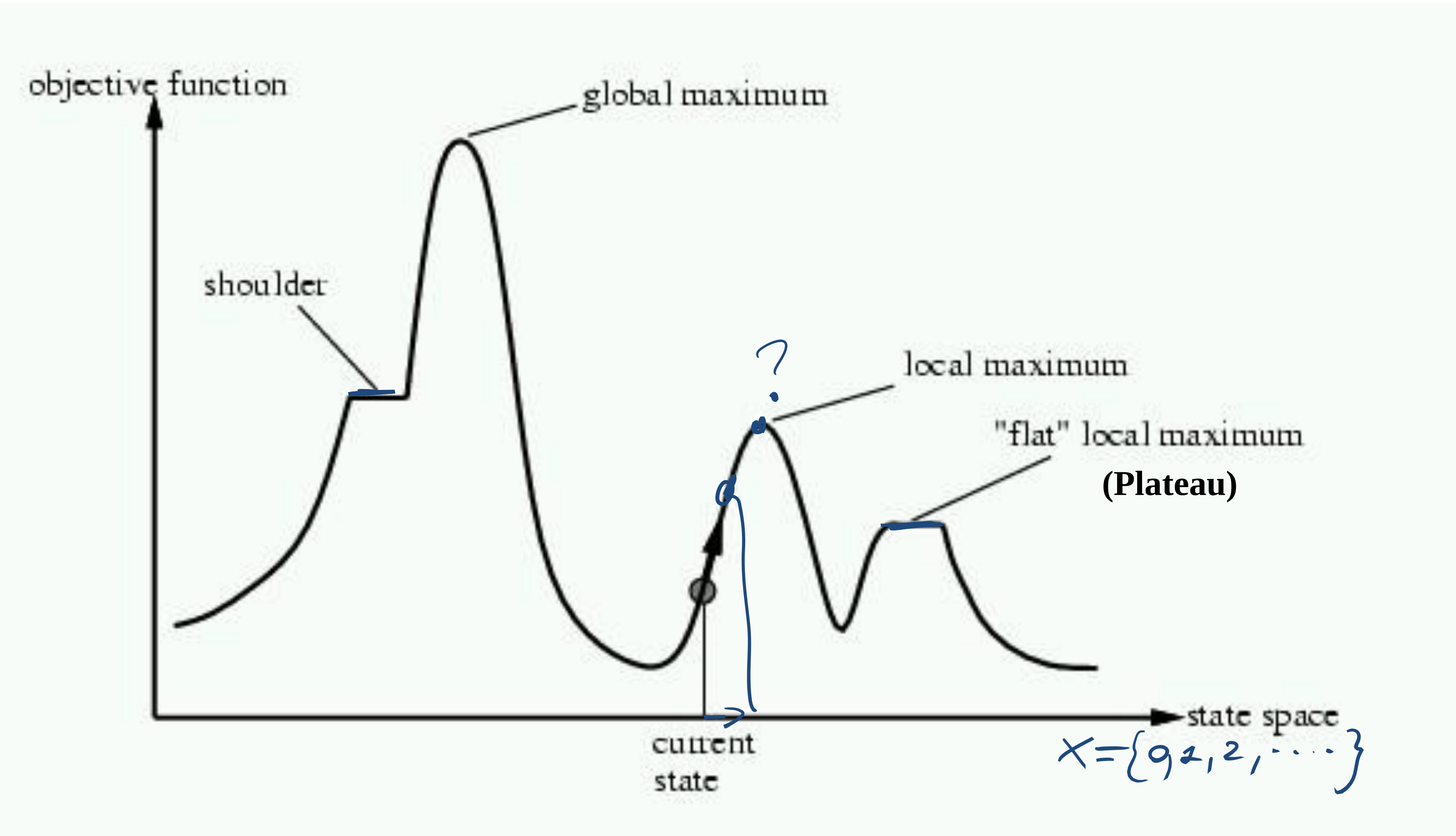
X_1 X_2

assume domain integer [0, 1, 2 ....]

Problems with Hill Climbing

Local Maxima.

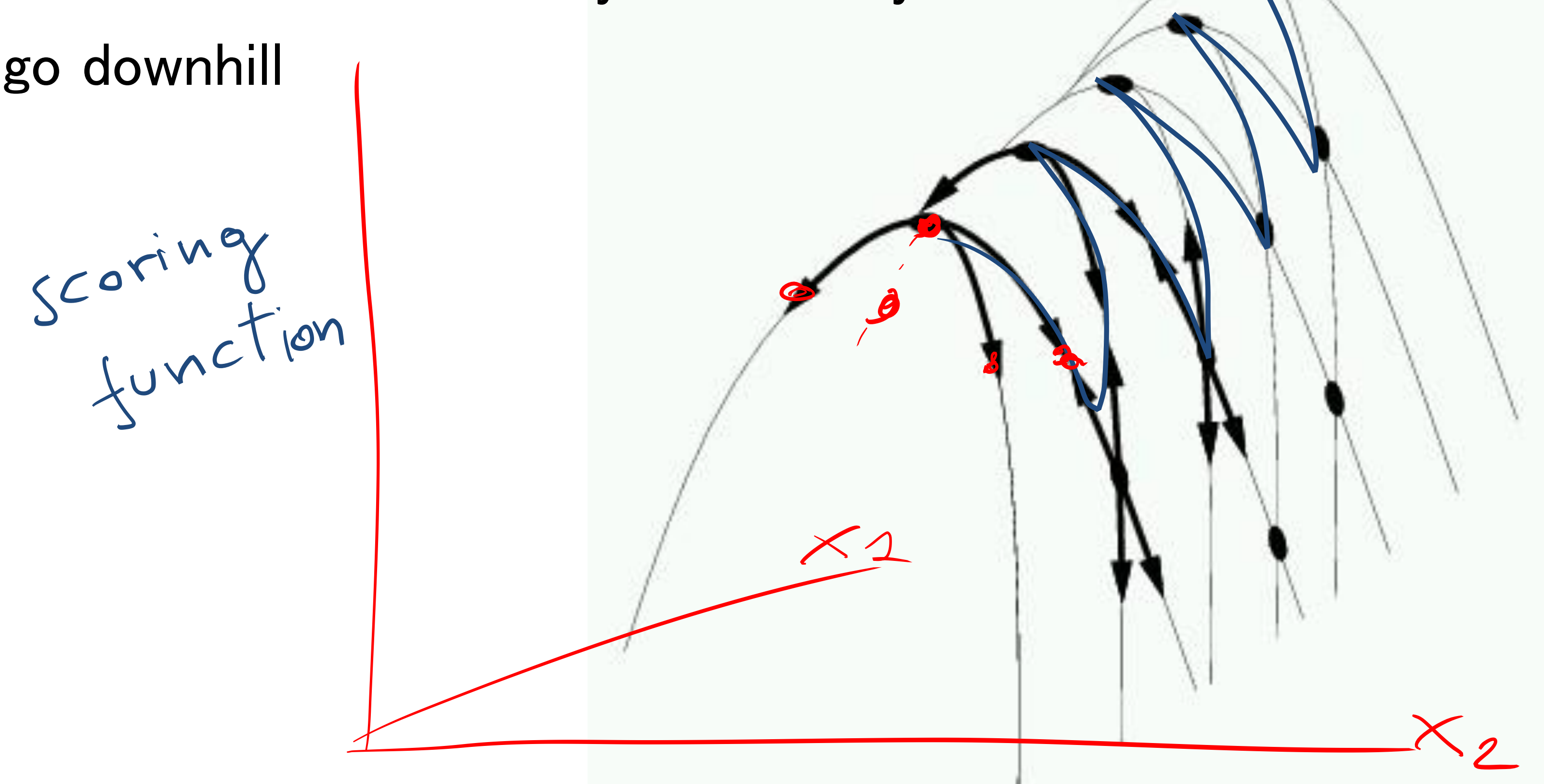
Plateau – Shoulders



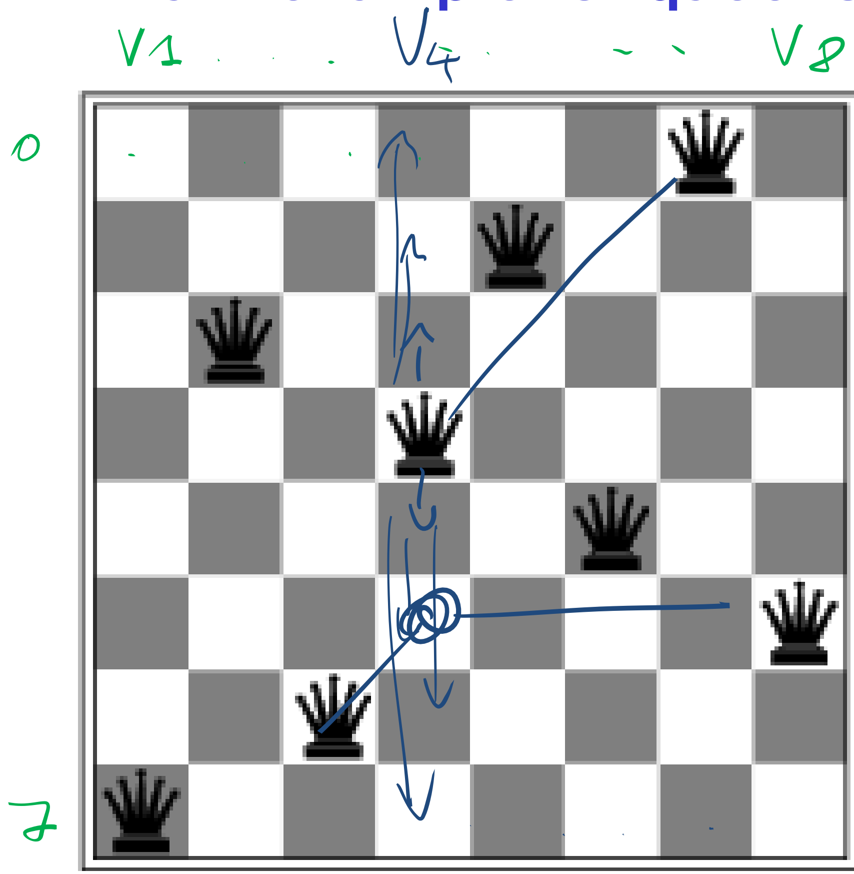
In higher dimensions…….

E.g., Ridges – sequence of local maxima not directly connected to each other

From each local maximum you can only go downhill



Corresponding problem for GreedyDescent
Local minimum example: 8-queens problem



for all the moves (neighbors) $h > 1$

$h = 0$ for solution

A local minimum with $h = 1$

Lecture Overview

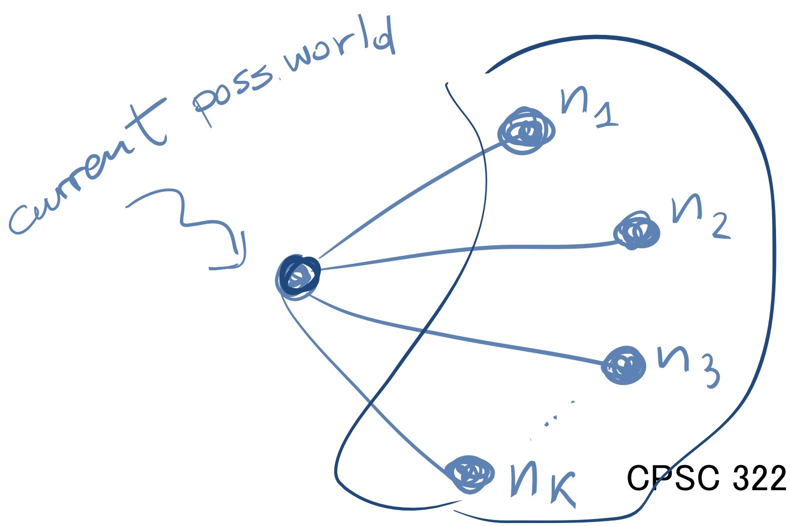
- Recap Local Search in CSPs
- Stochastic Local Search (SLS)
- Comparing SLS algorithms

Stochastic Local Search

GOAL: We want our local search

- to be guided by the scoring function
- Not to get stuck in local maxima/minima, plateaus etc.

- **SOLUTION:** We can alternate
  a) Hill-climbing steps
  b) Random steps: move to a random neighbor.
  c) Random restart: reassign random values to all variables.

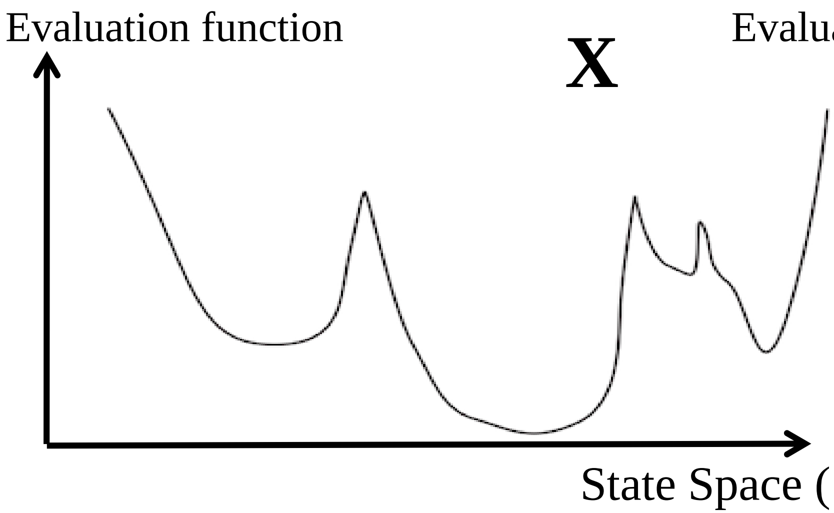


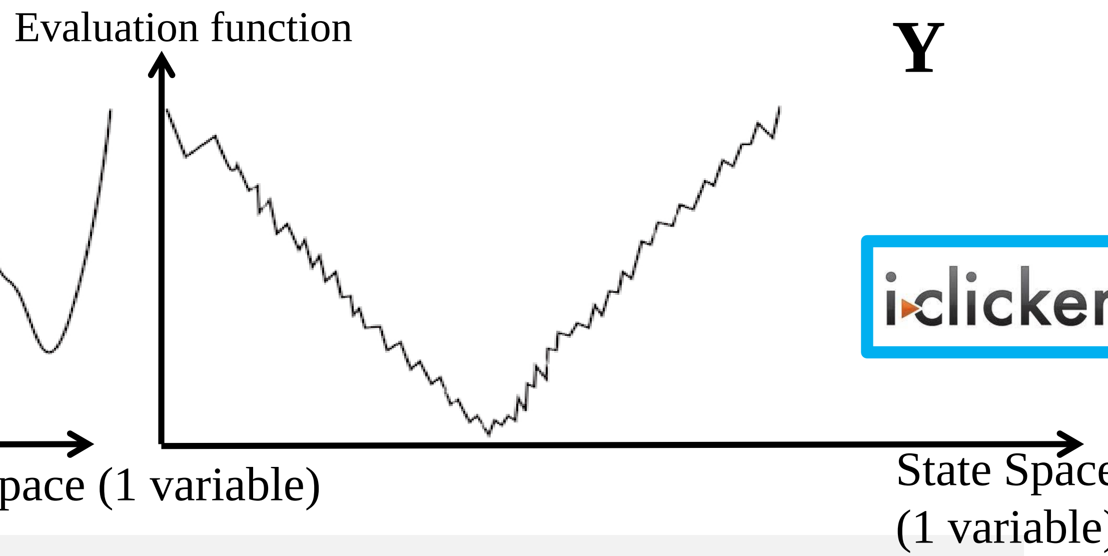
→ a) move to n_i which improves scoring function

→ b) select n_i randomly

→ c) Jump to a random poss. world

Which randomized method would work best in each of these two search spaces?



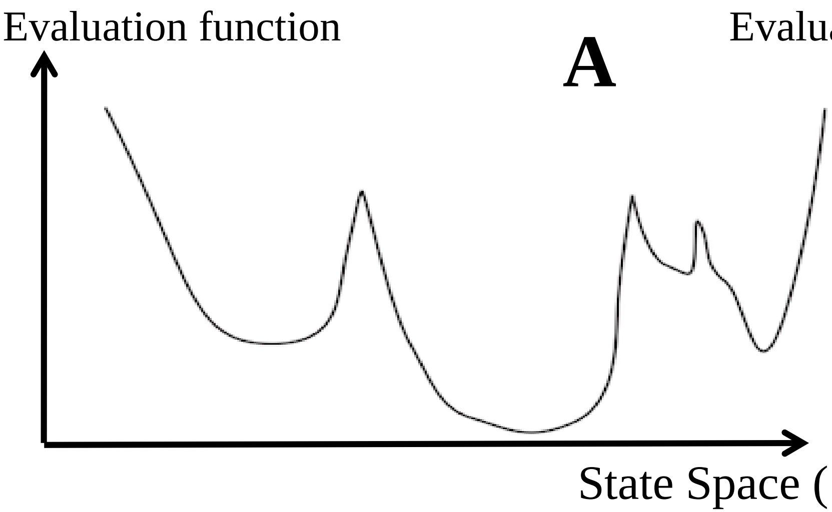


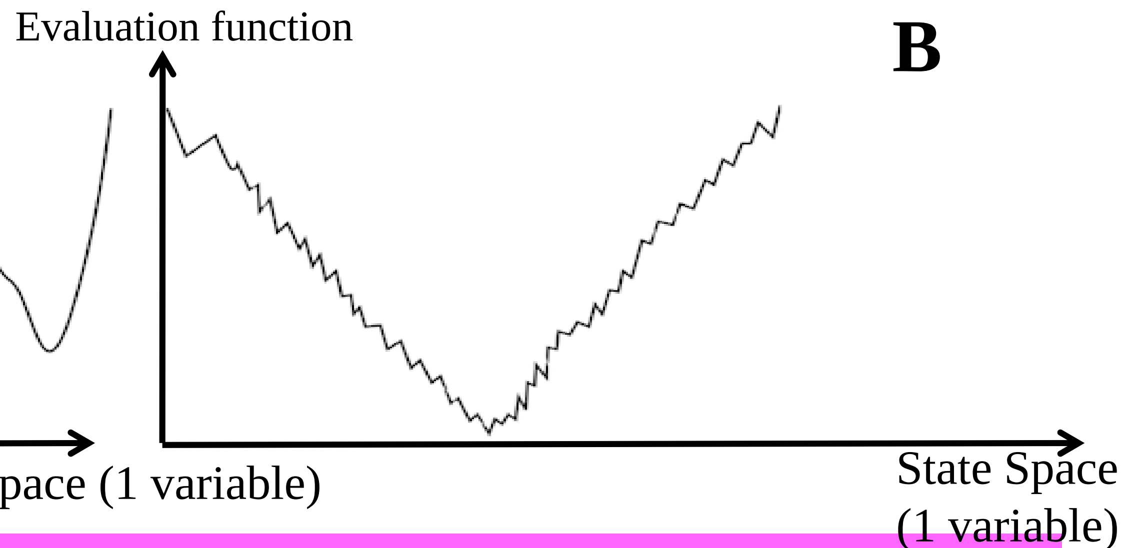
A. Greedy descent with random steps best on X
   Greedy descent with random restart best on Y

B. Greedy descent with random steps best on Y
   Greedy descent with random restart best on X

C. The two methods are equivalent on X and Y

Which randomized method would work best in each of the these two search spaces?





Greedy descent with random steps best on B

Greedy descent with random restart best on A

- But these examples are simplified extreme cases for illustration
  - in practice, you don't know what your search space looks like
- Usually integrating both kinds of randomization works best

Random Steps (Walk)

Let's assume that neighbors are generated as

- assignments that differ in one variable's value

How many neighbors there are given n variables with domains with d values?

$n(d-1)$

the neighbor!

8 variables

One strategy to add randomness to the selection of the variable–value pair. Sometimes choose the pair

1 • According to the scoring function

2 • A random one

E.G in 8-queen

- How many neighbors? $8 \cdot 7 = 56$

8 values

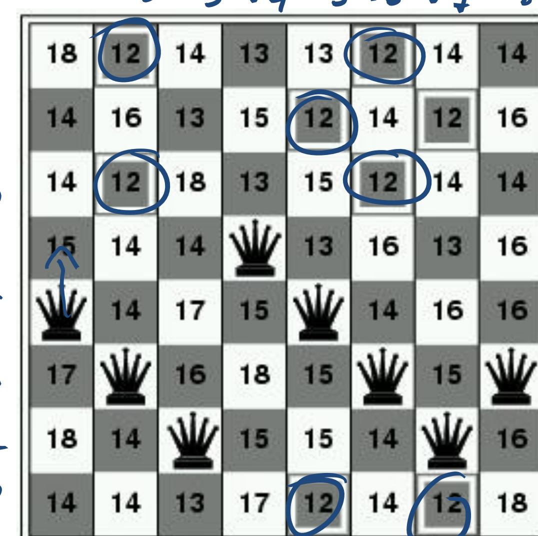
- ……..

1 choose one of the circled ones

2 choose randomly one of the 56

# of conflicts

V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8



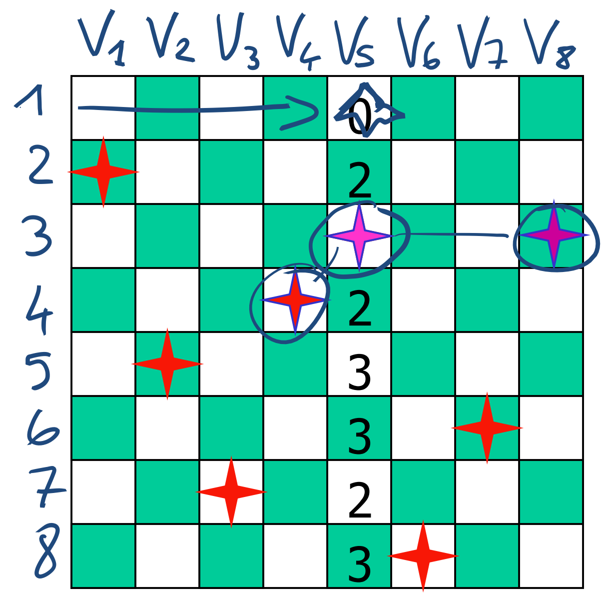
Random Steps (Walk): two-step

Another strategy: select a variable first, then a value:

- Sometimes select variable:
  1. that participates in the largest number of conflicts. V_5
  2. at random, any variable that participates in some conflict. $(V_4\ V_5\ V_8)$
  3. at random V_i
- Sometimes choose value
  a) That minimizes # of conflicts
  b) at random

Meth 1 selects V_5

Complete strategy 1.a) would select neighbor with $V_5 = 1$



conflicts

Aispace
2 a: Greedy Descent with
Min-Conflict Heuristic

Successful application of SLS

- **Scheduling of Hubble Space Telescope**: **reducing time** to schedule 3 weeks of observations:

from one week to around 10 sec.

Example: SLS for RNA secondary structure design

RNA strand made up of four bases: cytosine (C), guanine (G), adenine (A), and uracil (U)

2D/3D structure RNA strand folds into is important for its function

Predicting structure for a strand is "easy": $O(n^3)$

But what if we want a strand that folds into a certain structure?

- Local search over strands
  - ✓ Search for one that folds into the right structure
- Evaluation function for a strand
  - ✓ Run $O(n^3)$ prediction algorithm
  - ✓ Evaluate how different the result is from our target structure
  - ✓ Only defined implicitly, but can be evaluated by running the prediction algorithm

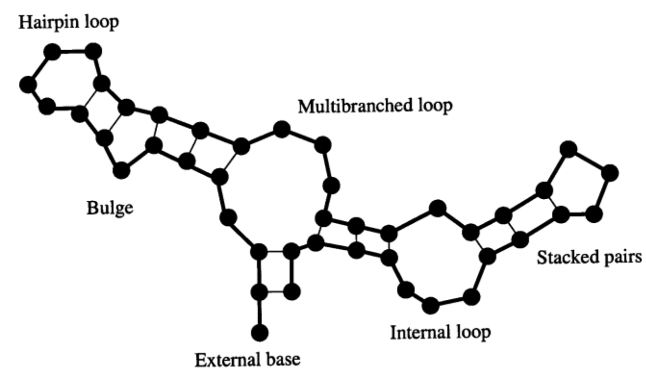
RNA strand

GUCCCAUAGGAUGUCCCAUAGGA

Easy

Hard

Secondary structure



Best algorithm to date: Local search algorithm RNA-SSD developed at UBC [Andronescu, Fejes, Hutter, Condon, and Hoos, Journal of Molecular Biology, 2004]

CSP/logic: formal verification

Hardware verification
(e.g., IBM)

Software verification
(small to medium programs)

Most progress in the last 10 years based on:
Encodings into propositional satisfiability (SAT)

(Stochastic) Local search advantage: Online setting

- **When the problem can change** (particularly important in scheduling)
- **E.g., schedule for airline:** thousands of flights and thousands of personnel assignment
  - Storm can render the schedule infeasible
- **Goal:** Repair with minimum number of changes
- This can be easily done with a local search starting form the current schedule
- Other techniques usually:
  - require more time
  - might find solution requiring many more changes

SLS limitations

- **Typically no guarantee to find a solution even if one exists**
  - SLS algorithms can sometimes stagnate
    - ✓ Get caught in one region of the search space and never terminate
  - Very hard to analyze theoretically

- **Not able to show that no solution exists**
  - SLS simply won't terminate
  - You don't know whether the problem is infeasible or the algorithm has stagnated

SLS Advantage: anytime algorithms

- When should the algorithm be stopped ?
  - When a solution is found
    (e.g. no constraint violations)
  - Or when we are out of time: you have to act NOW
  - Anytime algorithm:
    - ✓ maintain the node with best h found so far (the "incumbent")
    - ✓ given more time, can improve its incumbent

Lecture Overview

- Recap Local Search in CSPs
- Stochastic Local Search (SLS)
- Comparing SLS algorithms

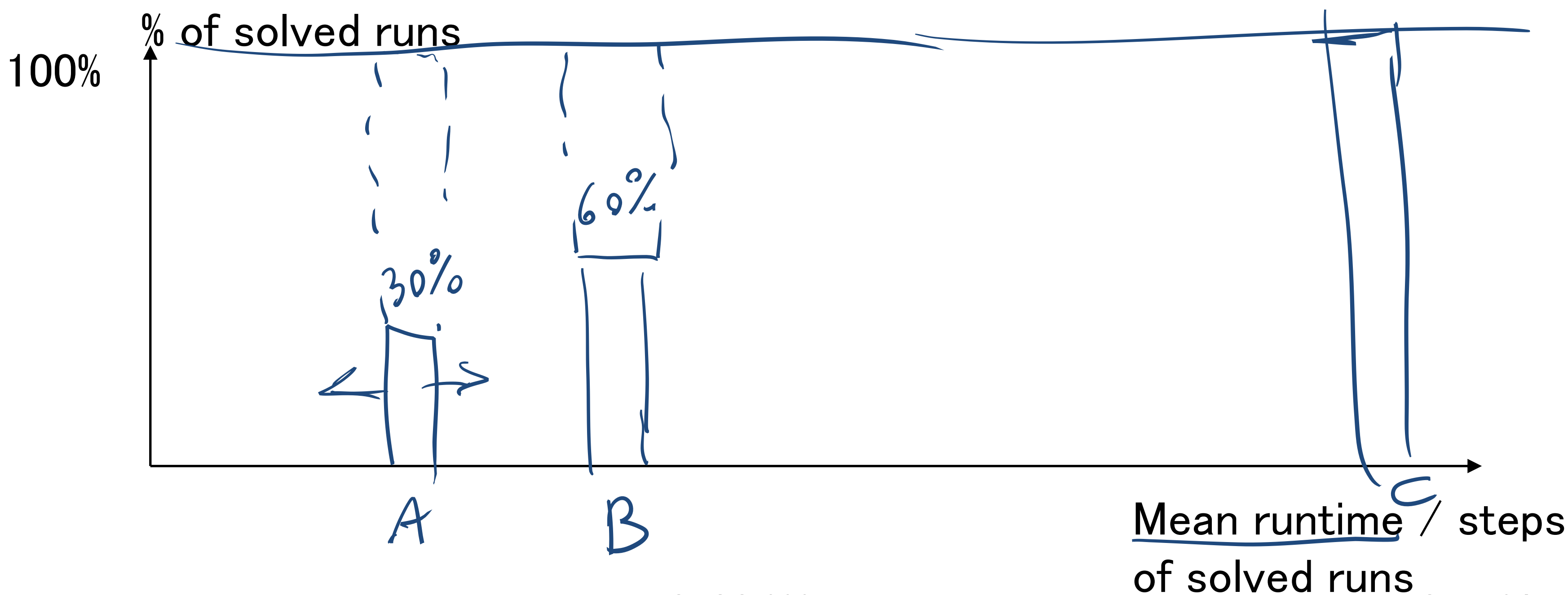
Evaluating SLS algorithms

- SLS algorithms are randomized
  - The time taken until they solve a problem is a random variable
  - It is entirely normal to have runtime variations of 2 orders of magnitude in repeated runs!
    - ✓ E.g. 0.1 seconds in one run, 10 seconds in the next one
    - ✓ On the same problem instance (only difference: random seed)
    - ✓ Sometimes SLS algorithm doesn't even terminate at all: stagnation

- If an SLS algorithm sometimes stagnates, what is its mean runtime (across many runs)?
  - Infinity!
  - In practice, one often counts timeouts as some fixed large value X
  - Still, summary statistics, such as **mean** run time or **median** run time, don't tell the whole story
    - ✓ E.g. would penalize an algorithm that often finds a solution quickly but sometime stagnates

First attempt….

- How can you compare three algorithms when
  A. one solves the problem 30% of the time very quickly but doesn't halt for the other 70% of the cases
  B. one solves 60% of the cases reasonably quickly but doesn't solve the rest
  C. one solves the problem in 100% of the cases, but slowly?



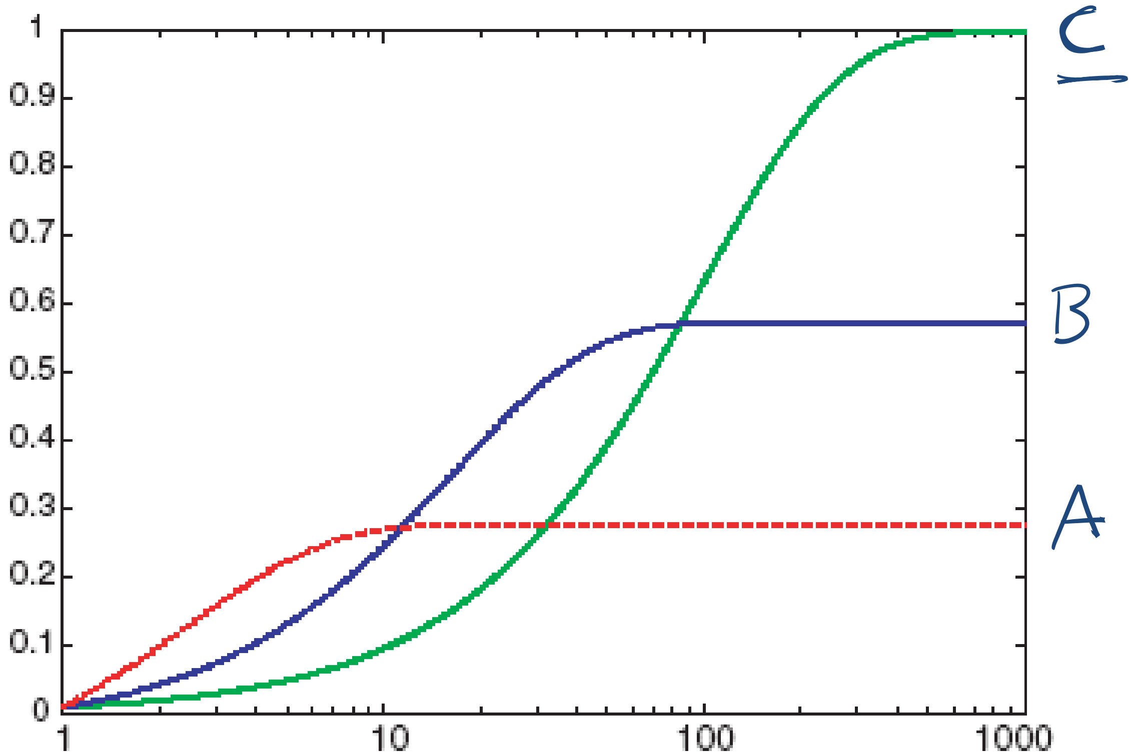
Runtime Distributions are even more effective

Plots runtime (or number of steps) and the proportion (or number) of the runs that are solved within that runtime.

- log scale on the *x* axis is commonly used

Fraction of solved runs, i.e.

P(solved by this # of steps/time)



# of steps

Comparing runtime distributions

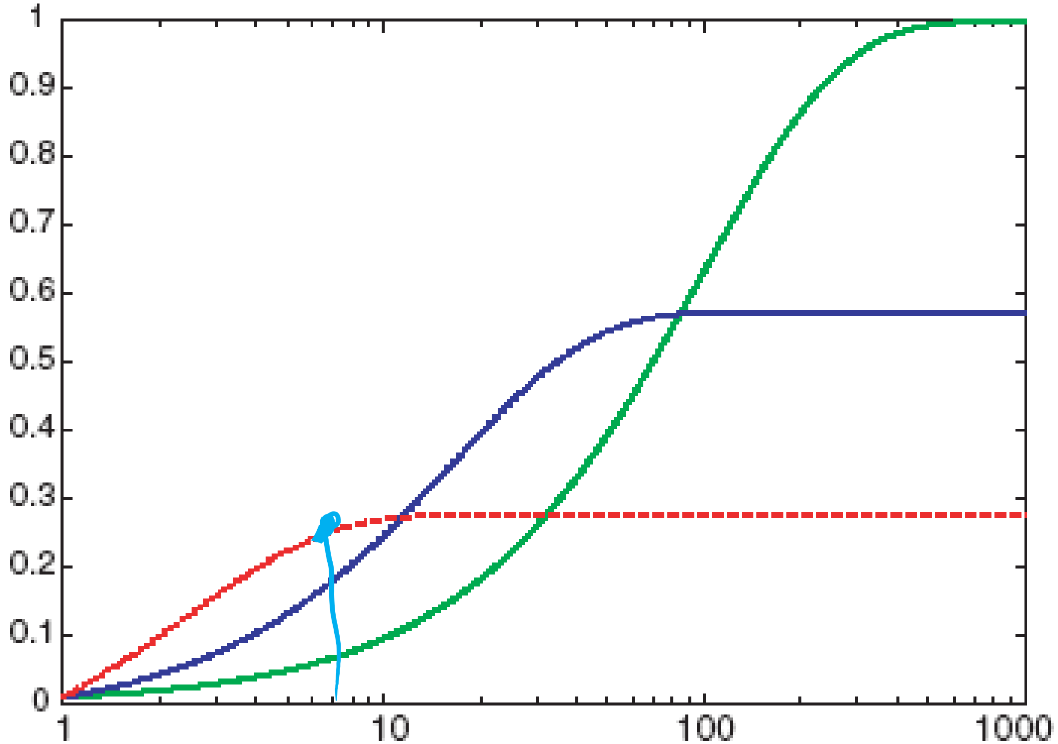
x axis: runtime (or number of steps)

y axis: proportion (or number) of runs solved in that runtime

- Typically use a log scale on the x axis

Fraction of solved runs, i.e.

P(solved by this # of steps/time)



# of steps

Which algorithm is most likely to solve the problem within 7 steps?

A. blue

B. red

C. green

Comparing runtime distributions

- Which algorithm has the best median performance?
  - I.e., which algorithm takes the fewest number of steps to be successful in 50% of the cases?

A. blue

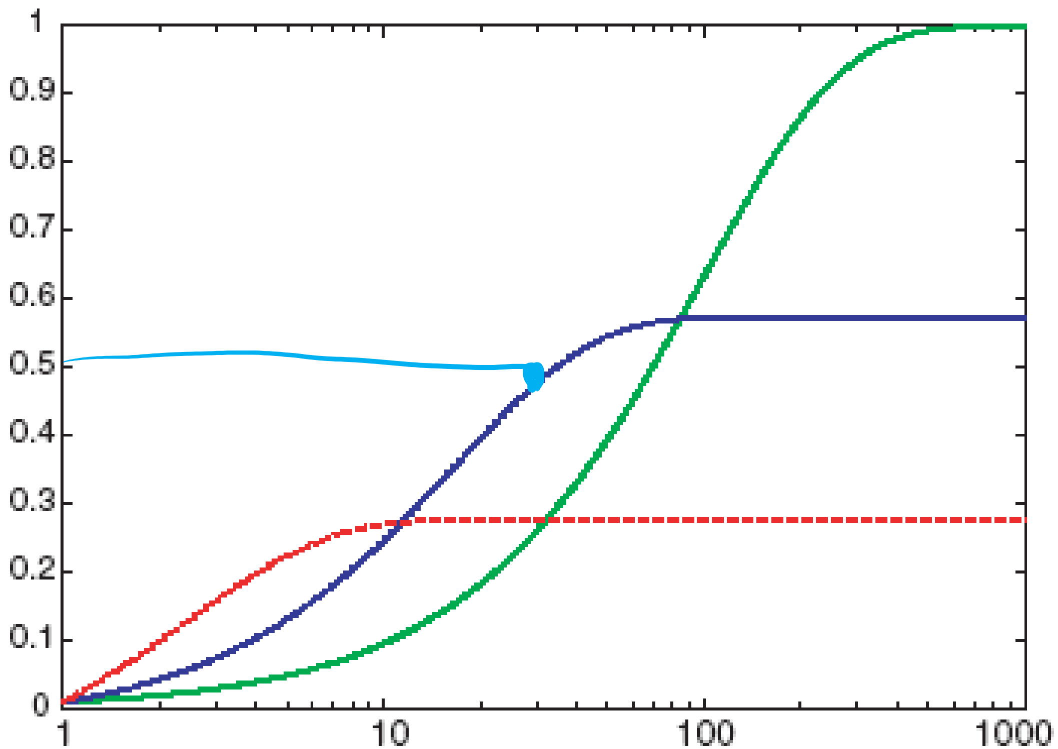
B. red

C. green

iclicker

Fraction of solved runs, i.e.

P(solved by this # of steps/time)



# of steps

Comparing runtime distributions

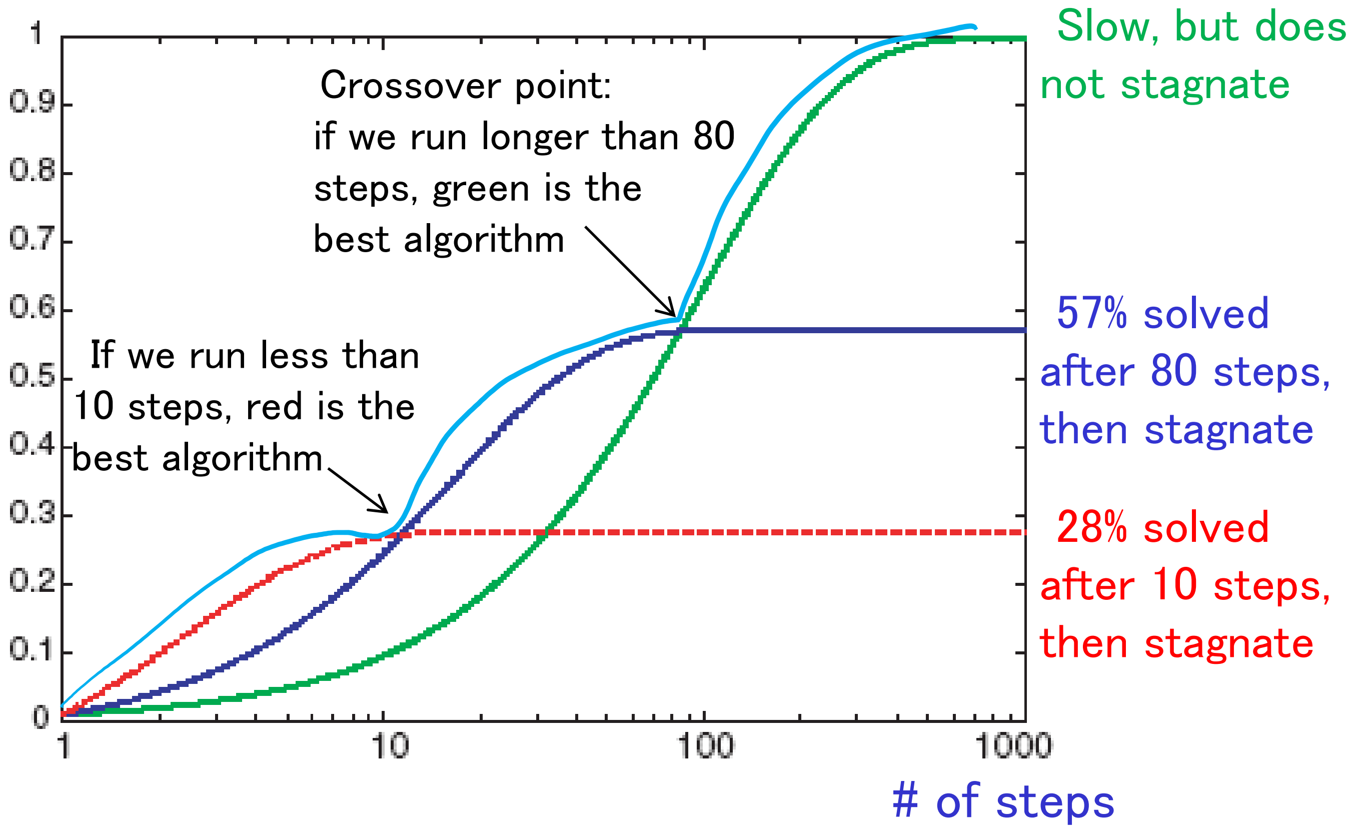
x axis: runtime (or number of steps)

y axis: proportion (or number) of runs solved in that runtime

- Typically use a log scale on the x axis

Fraction of solved runs, i.e.

P(solved by this # of steps/time)



Runtime distributions in AIspace

- Let’s look at some algorithms and their runtime distributions:
    1. Greedy Descent
    2. Random Sampling
    3. Random Walk
    4. Greedy Descent with random walk

- Simple scheduling problem 2 in AIspace:

What are we going to look at in AIspace

When selecting a variable first followed by a value:

- Sometimes select variable:
  1. that participates in the largest number of conflicts.
  2. at random, any variable that participates in some conflict.
  3. at random
- Sometimes choose value
  a) That minimizes # of conflicts
  b) at random

…..

AIspace terminology

keeps restarting

Random sampling
restart

Random walk 3b

Greedy Descent 1a

Greedy Descent Min conflict 2a

Greedy Descent with random walk 1-2a-b

Greedy Descent with random restart

Stochastic Local Search

- **Key Idea:** combine greedily improving moves with randomization

- As well as improving steps we can allow a “small probability” of: e.g.
  - Random steps: move to a random neighbor. 1%
  - Random restart: reassign random values to all variables. 5%

- Always keep best solution found so far

- Stop when
  - Solution is found (in vanilla CSP …………………… pw satisfying all C)
  - Run out of time (return best solution so far)

Learning Goals for today's class

You can:

- Implement SLS with
  - random steps (1-step, 2-step versions)
  - random restart
- Compare SLS algorithms with runtime distributions

Assign-2

- **Will be out today – due June**

Next Class

- Finish CSPs: More SLS variants Chp 4.9
- Planning: Representations and Forward Search Chp 8.1 - 8.2
- Planning: Heuristics and CSP Planning Chp 8.4