Department of Computer Science
Undergraduate Events

More details @
https://www.cs.ubc.ca/students/undergrad/life/upcoming-events

SAP Code Slam
Sat. Oct 13 noon to
Sun. Oct 14 noon
DMP 110

IBM Info Session
Tues. Oct 16
5:30 pm
Wesbrook 100

Global Relay Open House
Thurs. Oct 18
4:30 – 6:30 pm
220 Cambie St. 2nd Floor

Stochastic Local Search

Computer Science cpsc322, Lecture 15

(Textbook Chpt 4.8)

Oct, 10, 2012

Announcements

- Thanks for the feedback, we'll discuss it on Mon

- Assignment-2 on CSP will be out on Fri (programming!)

Lecture Overview

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

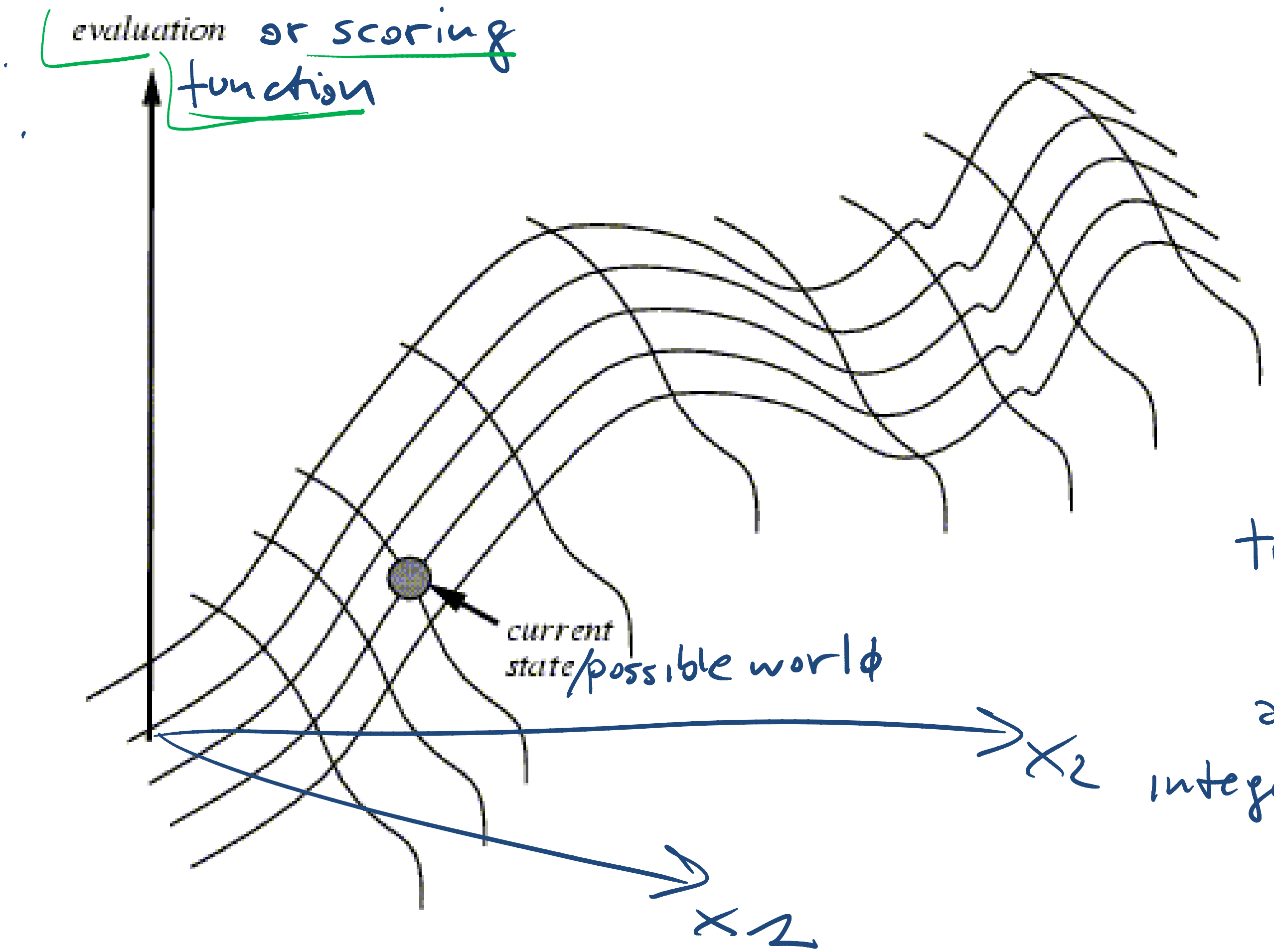
Local Search: Summary

- 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
    - ✓ Information about the cost/quality of the solution (you want the best solution, not just a solution)

Hill Climbing

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

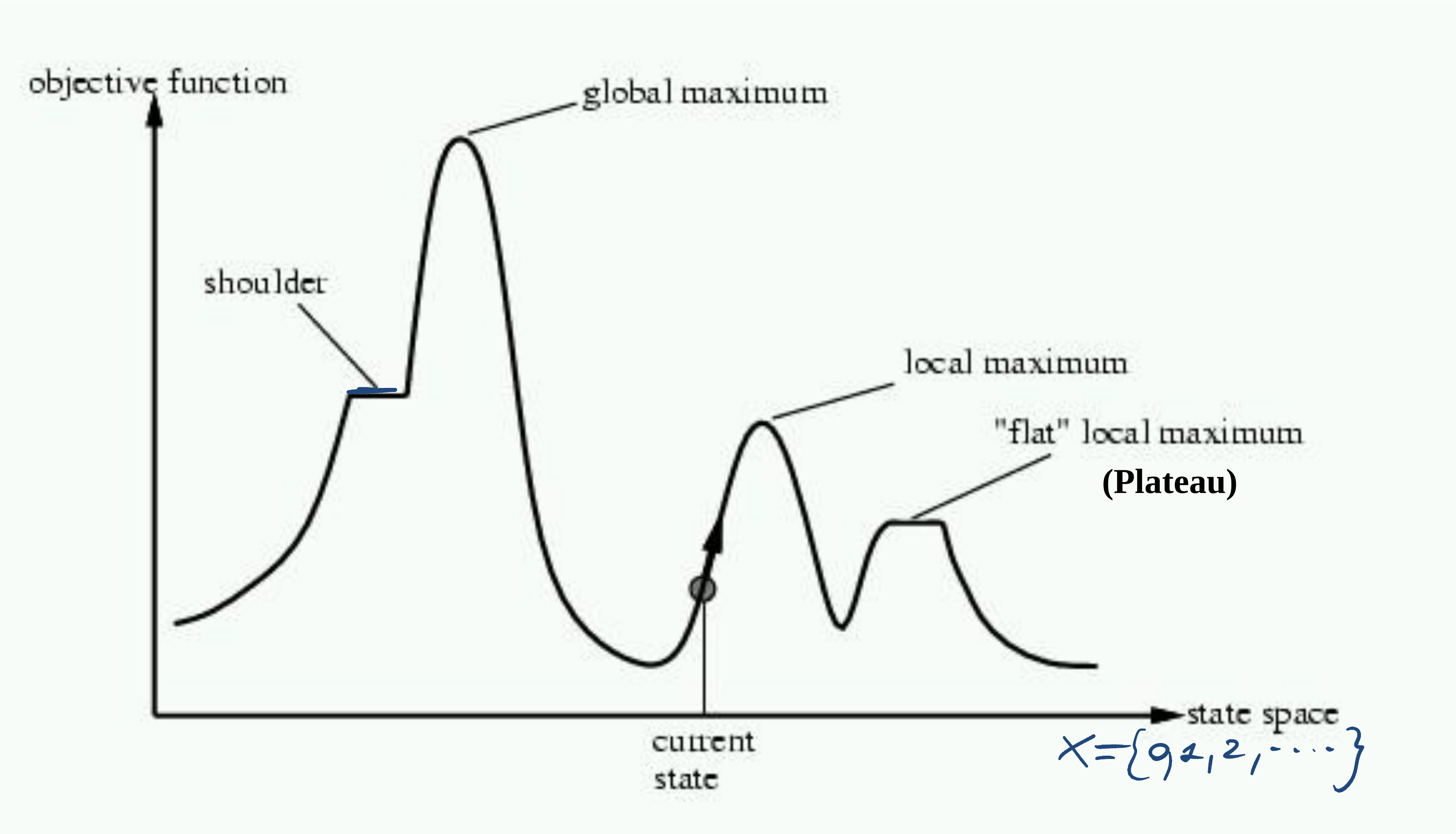


two vars
x_1 x_2
assume domain integer [0,1,2....]

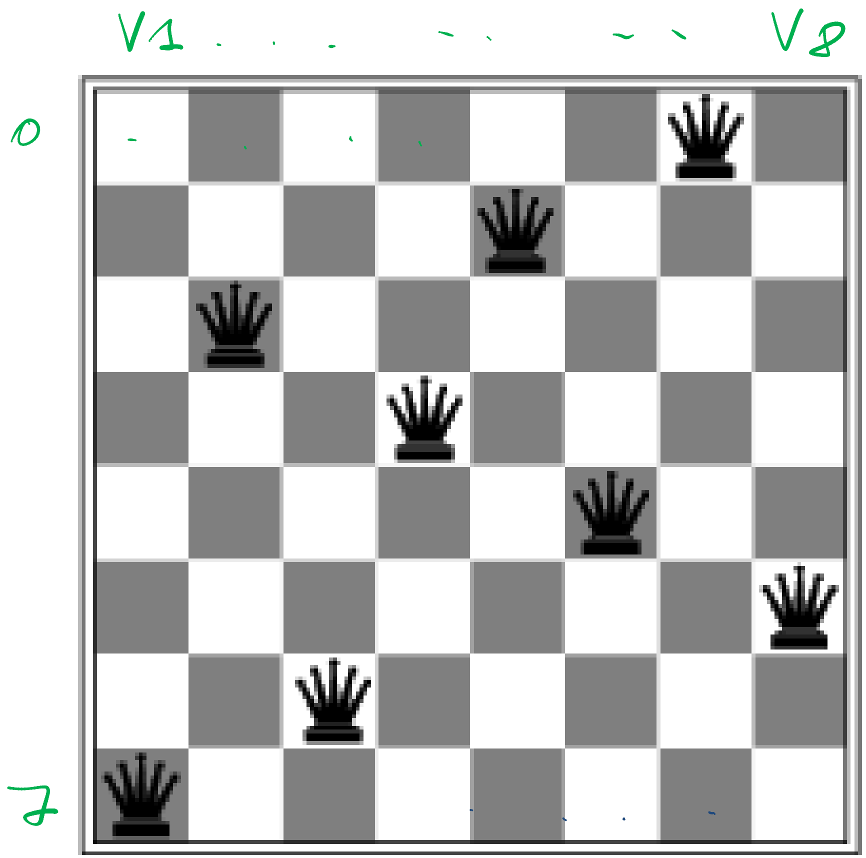
Problems with Hill Climbing

Local Maxima.

Plateau - Shoulders



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

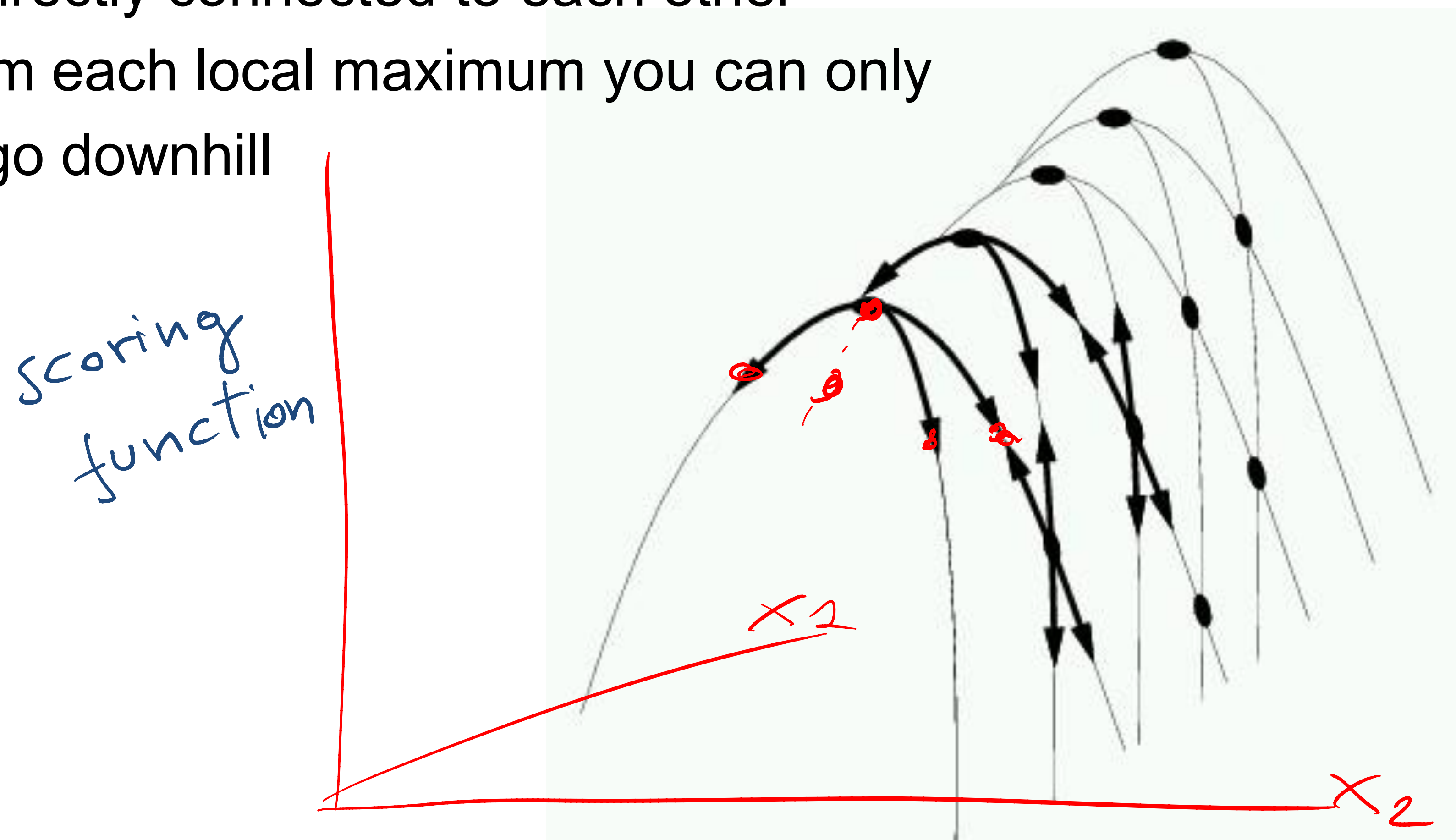


A local minimum with $h = 1$

Even more Problems 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



Lecture Overview

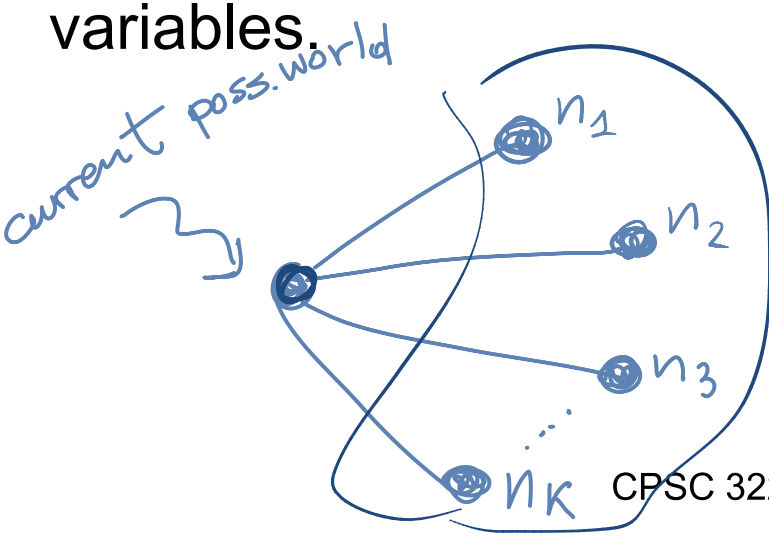
- Recap Local Search in CSPs
- Stochastic Local Search (SLS)
- Comparing SLS algorithms

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.

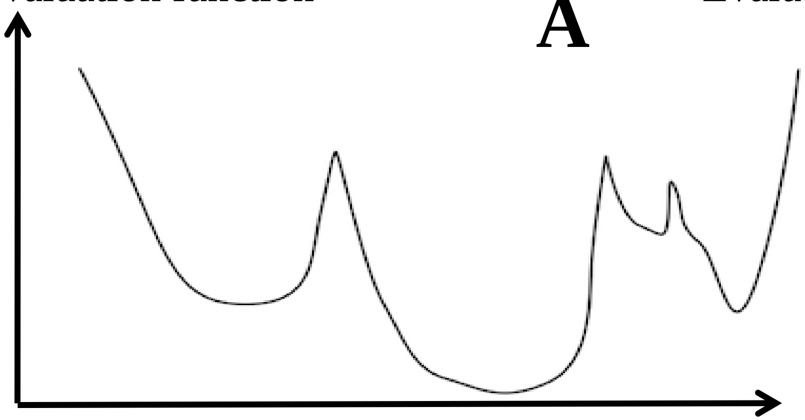


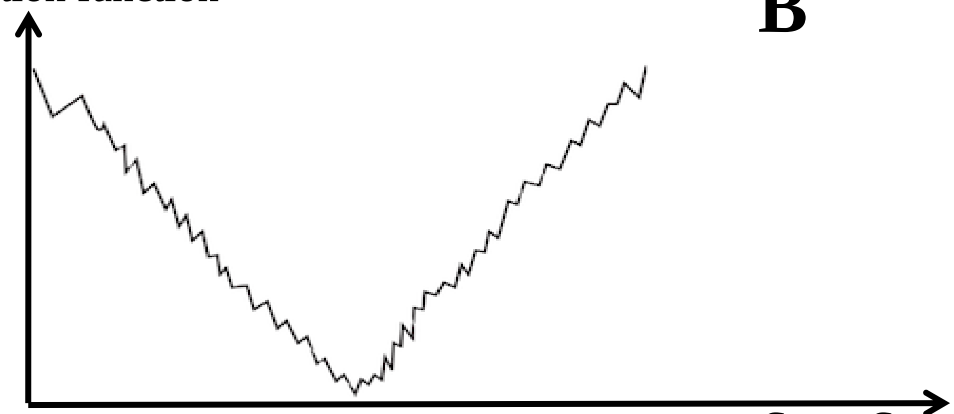
→ a) move to n_i which improves scoring function

→ 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?





Greedy descent with random steps best on A

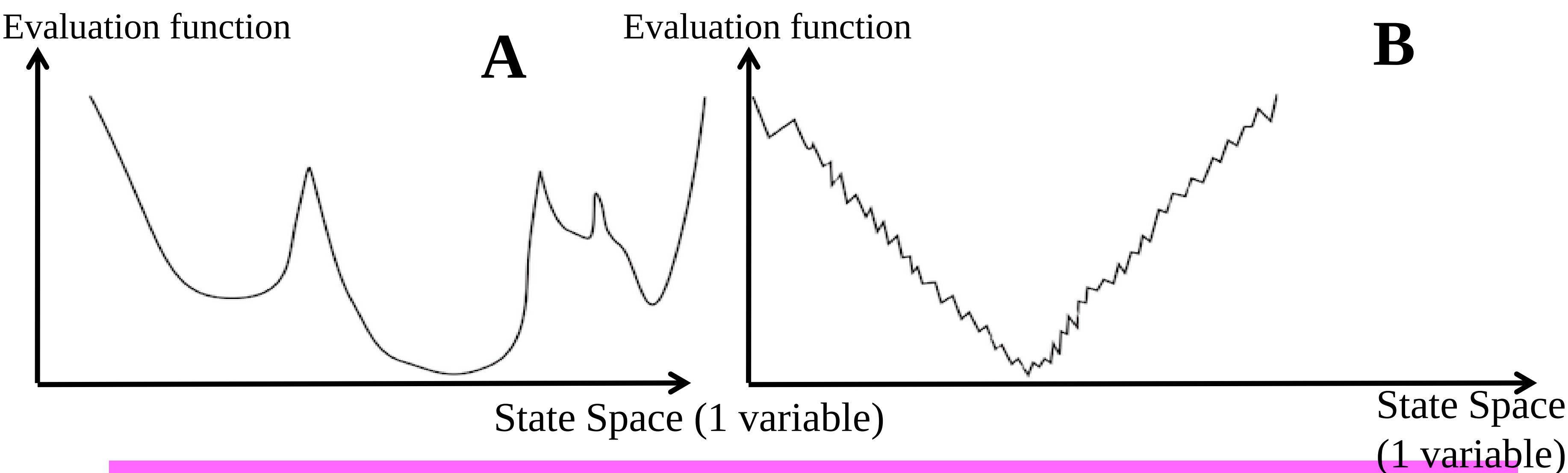
Greedy descent with random restart best on B

Greedy descent with random steps best on B

Greedy descent with random restart best on A

equivalent

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? $n(d-1)$

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

1. According to the scoring function
2. A random one

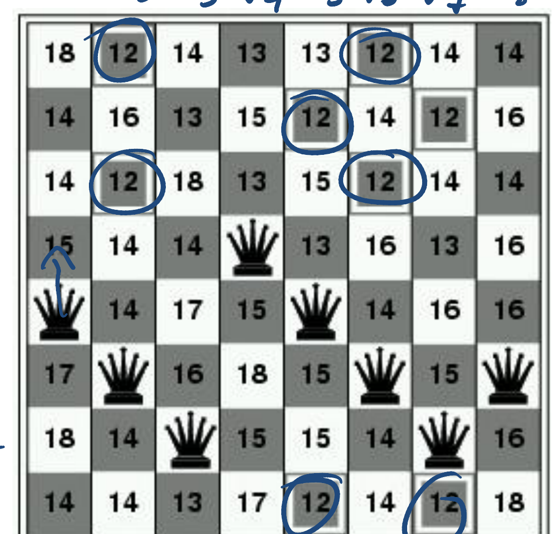
E.G in 8-queen

- How many neighbors? $8 \cdot 7 = 56$
- 1. choose one of the circled ones
  2. choose randomly one of the 56

8 variables

8 values

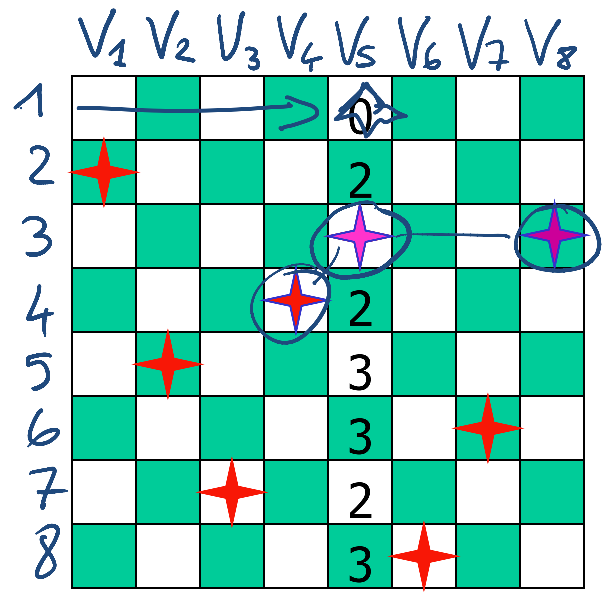
# of conflicts



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.
  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
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

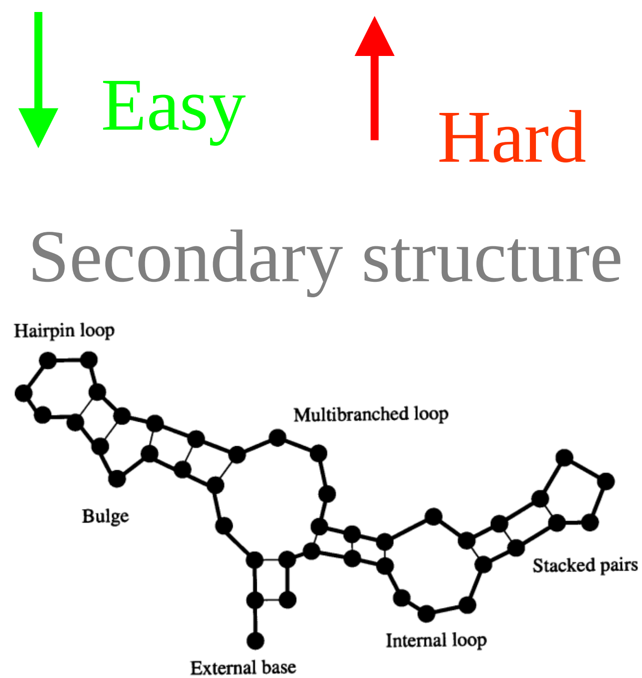
Predicting structure for a strand is “easy”: $O(n^3)$

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



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

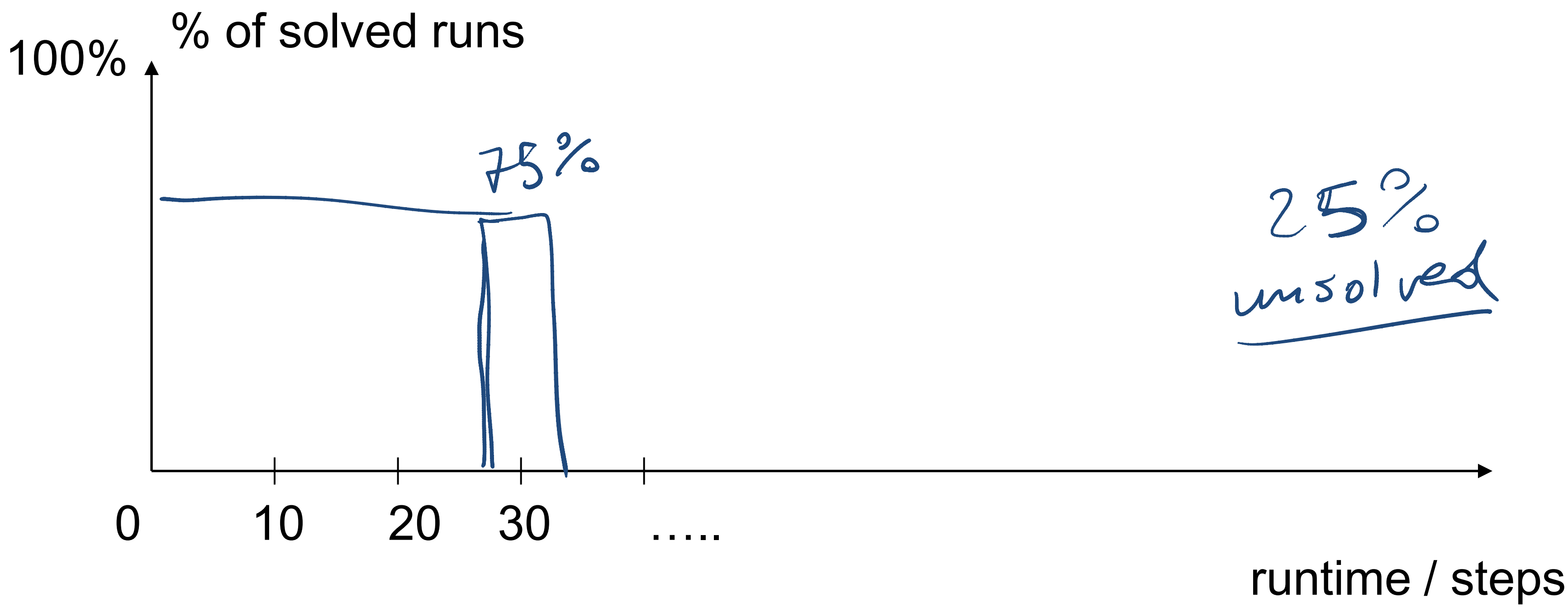
Evaluating SLS algorithms

- 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

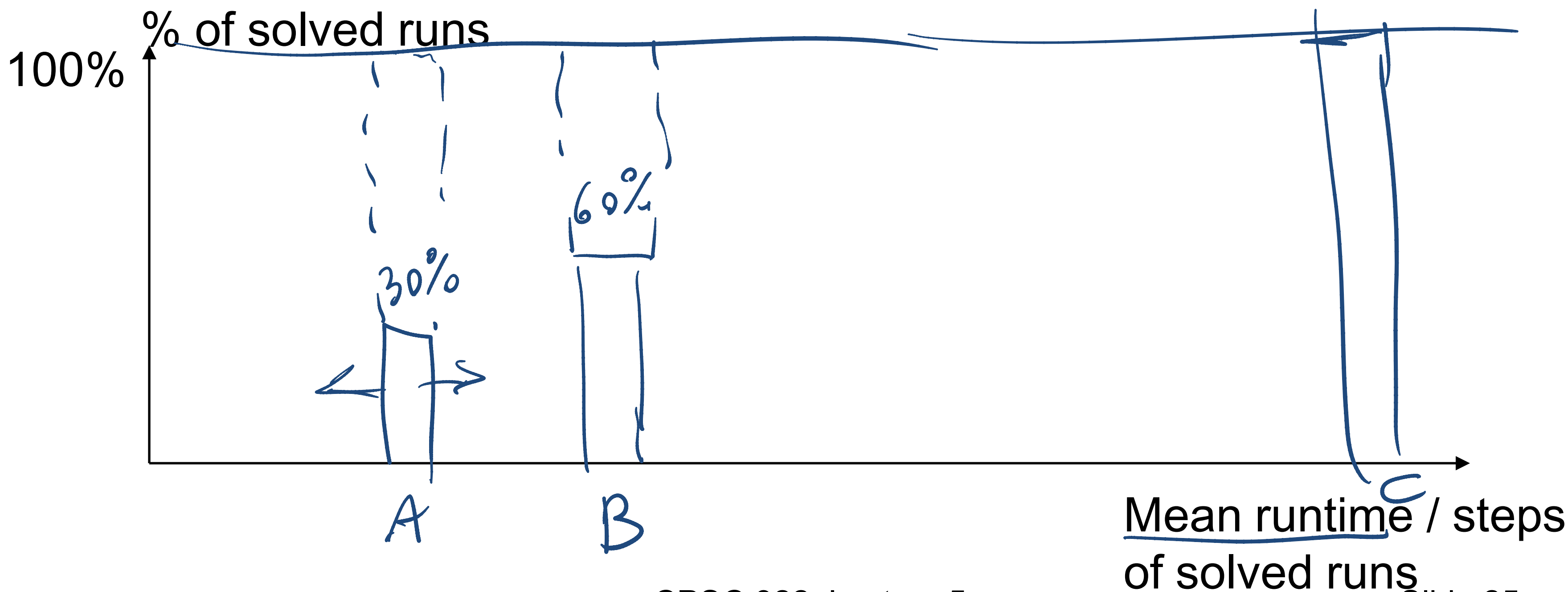
Comparing Stochastic Algorithms: Challenge

- Summary statistics, such as **mean** run time, **median** run time, and **mode** run time don't tell the whole story
  - What is the running time for the runs for which an algorithm ***never*** finishes (infinite? stopping time?)



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?



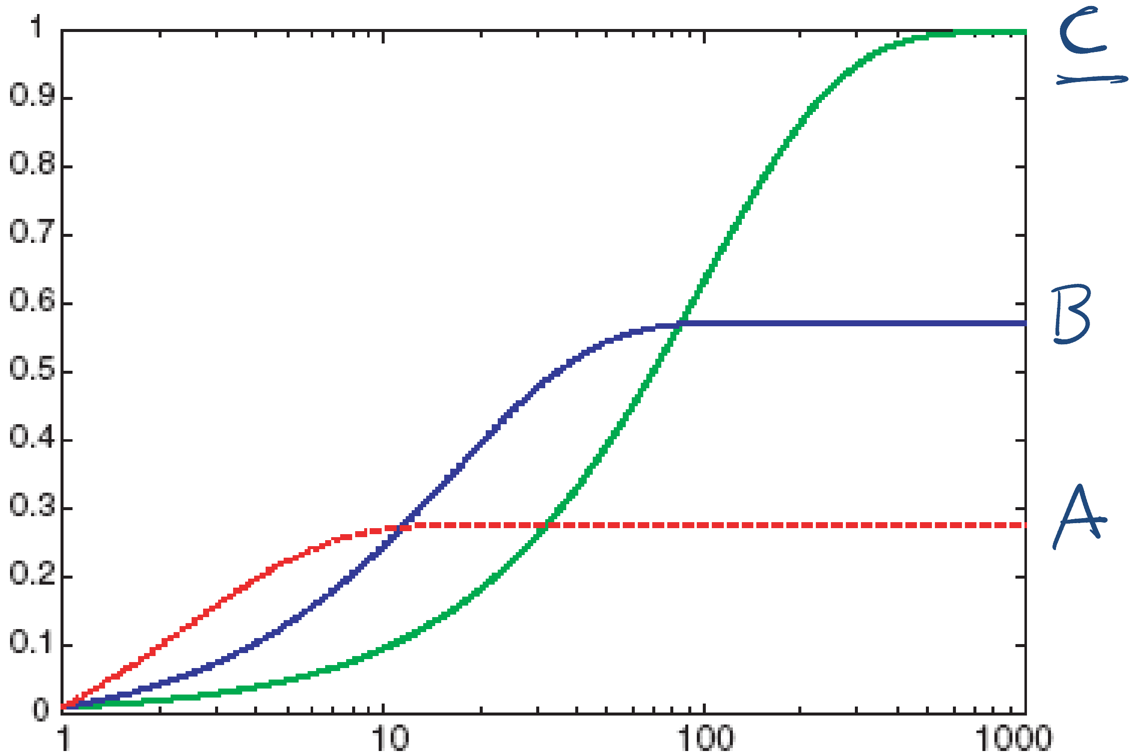
Runtime Distributions are even more effective

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

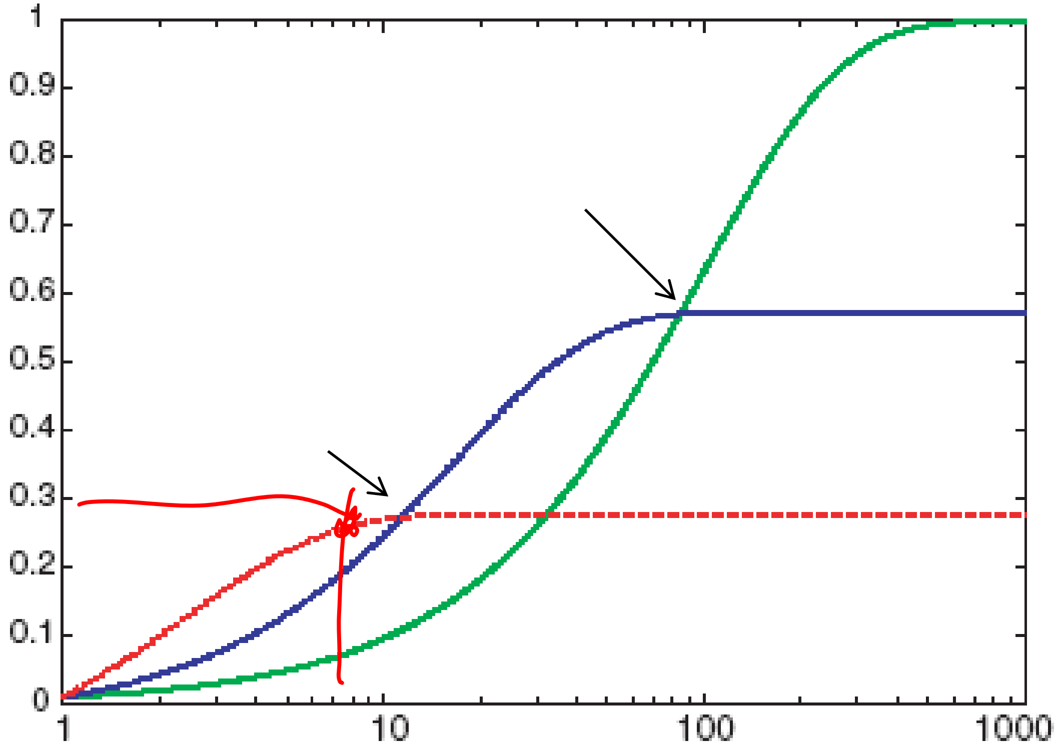
Comparing runtime distributions

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?

blue

red

green

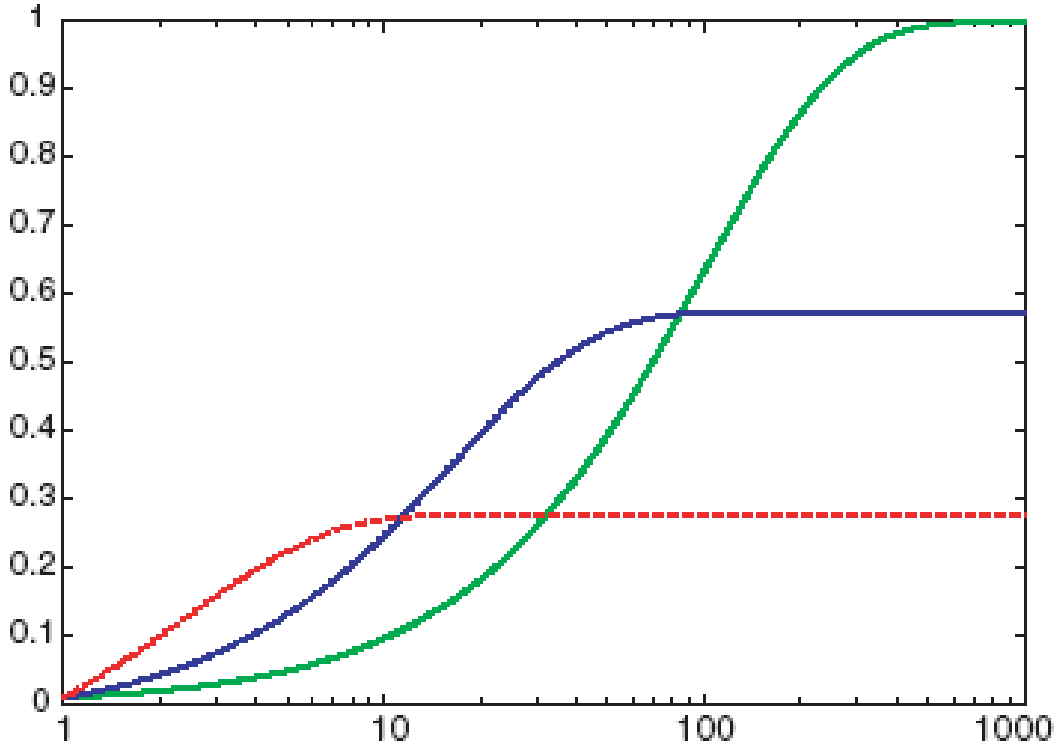
Comparing runtime distributions

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?

red

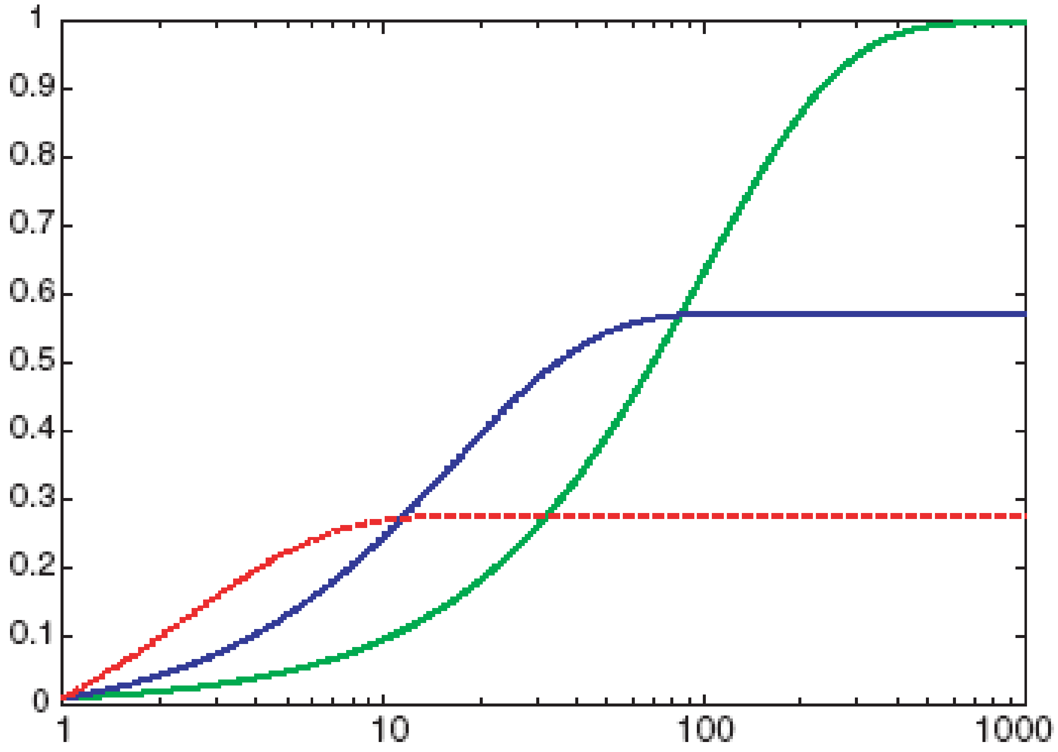
Comparing runtime distributions

- 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?

blue red green

Fraction of solved runs, i.e.

P(solved by this # of steps/time)



# of steps

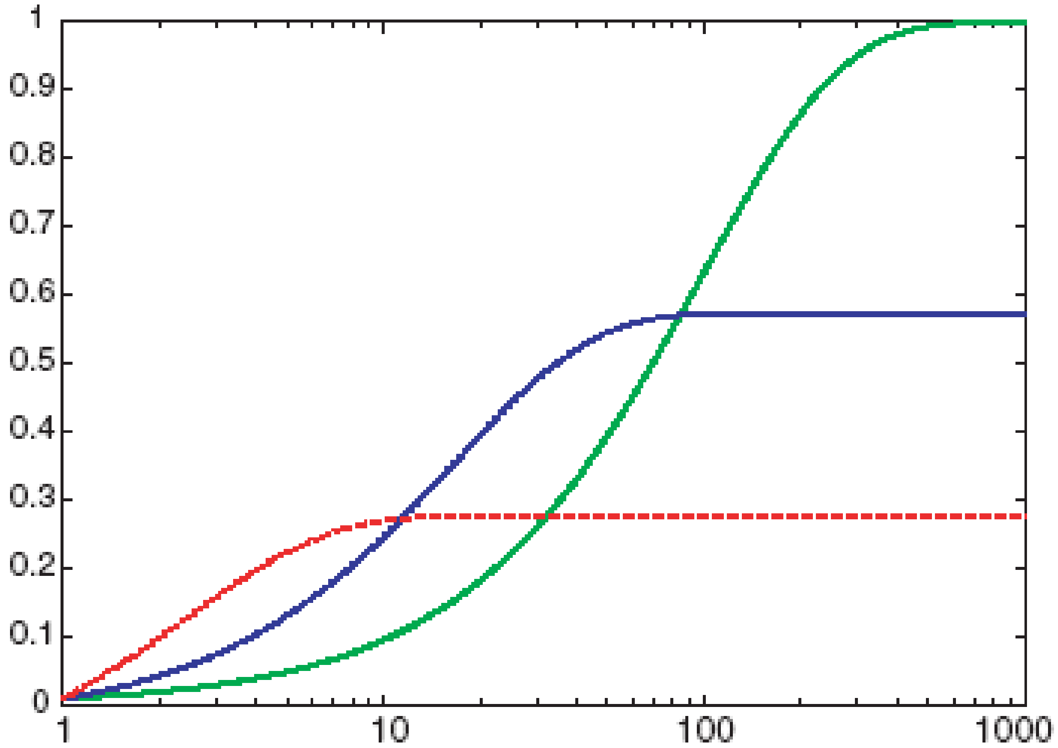
Comparing runtime distributions

- 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?

blue

Fraction of solved runs, i.e.

P(solved by this # of steps/time)



# of steps

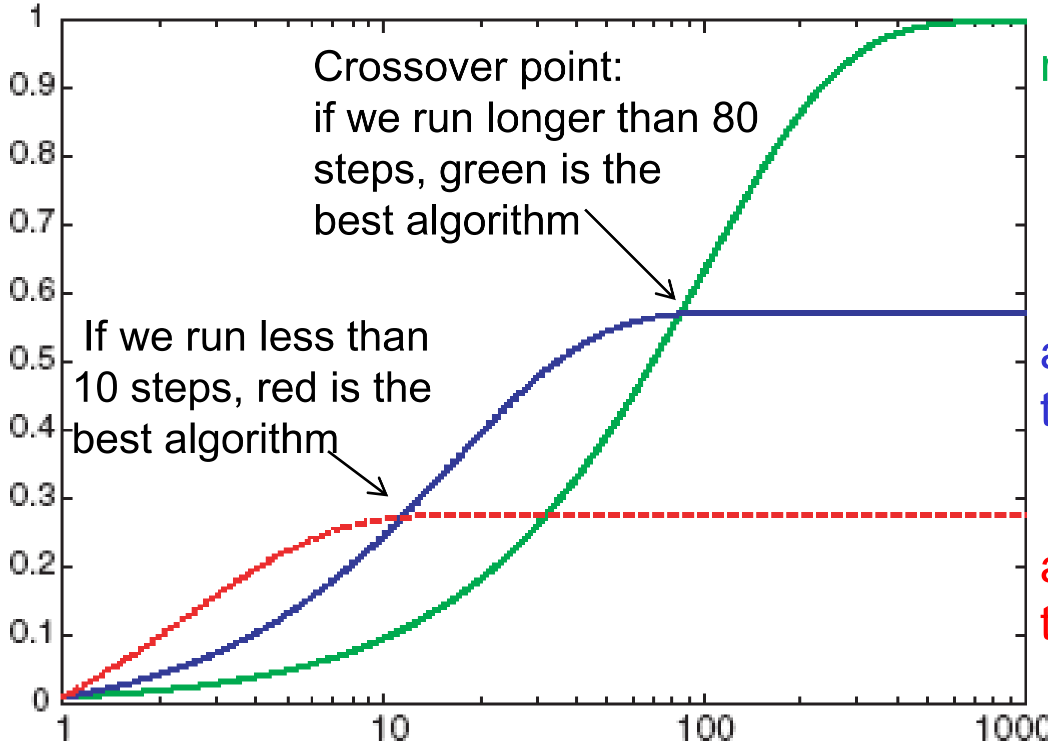
Comparing runtime distributions

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)



Slow, but does not stagnate

57% solved after 80 steps, then stagnate

28% solved after 10 steps, then stagnate

# of steps

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 2ab

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 on Tue
- Assignments will be weighted:

A0 (12%), A1…A4 (22%) each

Next Class

- More SLS variants
- Finish CSPs
- (if time) Start planning