Stochastic Local Search Variants

CPSC 322 Lecture 15

Lecture Overview

- **Recap SLS**
- SLS variants

Stochastic Local Search

- **Key Idea:** combine greedily improving moves with randomization
  - As well as improving steps, we can allow a "small probability" of:
    - Random steps: move to a random neighbor.
    - Random restart: reassign random values to all variables.
  - Always keep best solution found so far
  - Stop when
    - Solution is found (in vanilla CSP, satisfies all constraints)
    - Run out of time (return best solution so far)

Learning Goals for today's class

You can:

- Implement a tabu list
- Implement the simulated annealing algorithm
- Implement population-based SLS algorithms:
  - Beam Search
  - Genetic Algorithms
- Explain pros and cons of different SLS algorithms

Lecture Overview

- Recap SLS
- **SLS variants**
  - **Tabu lists**
  - **Simulated Annealing**
  - **Beam search**
  - **Genetic Algorithms**

Tabu lists

- Potentially problematic behaviors:
  - Returning to recently visited nodes
  - Cycling

- Maintain a tabu list of the ***k*** last nodes visited
  - Don't visit a possible world that is on the **tabu list**

- Cost of this method depends on_______

Simulated Annealing

- **Key idea:** Change the degree of randomness

- Annealing: a metallurgical process where metals are heated and then slowly cooled.
  - Analogy: start with a high “temperature”: a high tendency to take random steps
  - Over time, cool down: more likely to follow the scoring function
- Temperature reduces over time, according to an annealing schedule

Simulated Annealing: algorithm

Here's how it works (for maximizing h(n)):

- You are in node ***n***. Pick a variable at random and a new value at random. You generate ***n'***
- If it **is** an "improvement" i.e., h(n') >= h(n), adopt it.
- If it **isn't** an improvement, adopt it probabilistically depending on the difference and a temperature parameter, *T*.
  - ✓ we move to *n'* with probability

$$e^{\left(\frac{h(n')-h(n)}{T}\right)}$$

- If it isn't an improvement, adopt it probabilistically depending on the difference and a temperature parameter, *T*.
  - we move to n' with probability $e^{\left(\frac{h(n')-h(n)}{T}\right)}$

Having a **higher** temperature *T* means that the algorithm is

i>clicker

A. **more** likely to move to *n'* if it is **"better"** than *n*

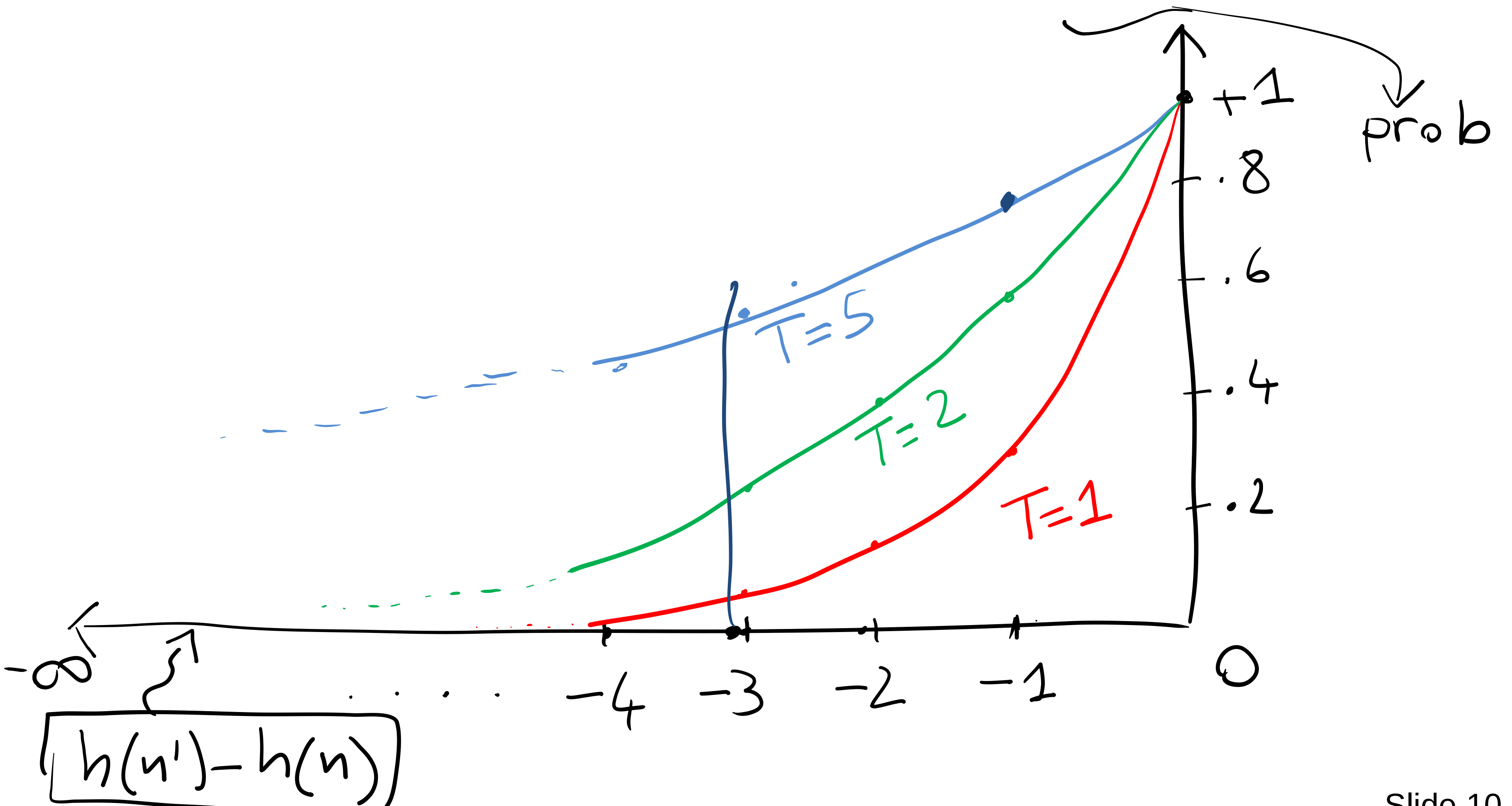
B. **less** likely to move to *n'* if it is **"better"** than *n*

C. **more** likely to move to *n'* if it is **"worse"** than *n*

D. **less** likely to move to *n'* if it is **"worse"** than *n*

E. one of the causes of climate change

- If it isn't an improvement, adopt it probabilistically depending on the difference and a temperature parameter, *T*.
  - we move to *n'* with probability $e^{\left(\frac{h(n')-h(n)}{T}\right)}$



Properties of simulated annealing search

One can prove *(but we won't in this course)***:**
If ***T*** **decreases slowly enough**, then simulated annealing search will find a **global optimum with probability approaching 1**

Widely used in **Very Large Scale Integration (VLSI) layout**, **airline scheduling**, etc.

Finding the ideal cooling schedule is unique to each class of problems

- Want to move off of local extrema but not global extrema
- Want to minimize computation time while taking enough time to ensure we reach a good minimum

Lecture Overview

- Recap SLS
- **SLS variants**
  - **Simulated Annealing**
  - **Population Based**
    - **Beam search**
    - **Genetic Algorithms**

Population Based SLS

Often we have more memory than the one required for current node (+ best so far + tabu list)

Key Idea: maintain a population of *k* individuals

- At every stage, update your population.
- Whenever one individual is a solution, report it.

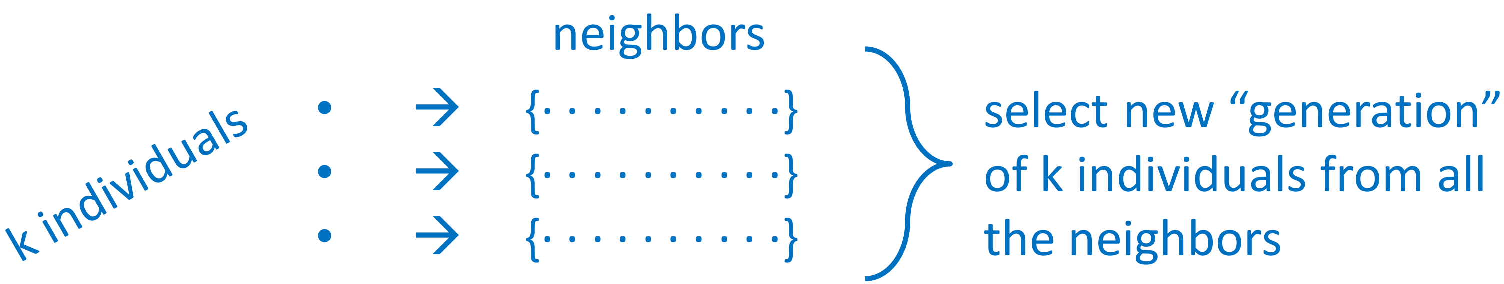
Simplest strategy: Parallel Search

- All searches are independent
- No information shared
- Essentially running k random restarts in parallel rather than in sequence
- but we can take this idea…

Population Based SLS: Beam Search

Non Stochastic

- Like parallel search, with *k* individuals, but you choose the *k* best out of all of the neighbors.
- Useful information is passed among the k parallel search thread



- **Troublesome case:** If one individual generates several good neighbors, and the other k-1 individuals all generate bad successors, then the next generation will consist of very similar individuals ☹

Population Based SLS: Stochastic Beam Search

- **Non Stochastic** Beam Search may suffer from lack of diversity among the k individual (just a more expensive hill climbing)
- **Stochastic** version alleviates this problem:
  - Selects the k individuals at random (from neighbors n_1 to n_m)
  - But probability of selection proportional to their value (according to scoring function h(n))

$$Prob\ of\ selecting\ n_j\ ?=$$

A $\dfrac{n_j}{\sum_i n_i}$

B $\dfrac{\sum_i h(n_i)}{h(n_j)}$

C $\dfrac{h(n_j)}{\sum_i h(n_i)}$

iclicker.

Stochastic Beam Search: Advantages

- It **maintains diversity** in the population.
- **Biological metaphor** (asexual reproduction):
  - each individual generates "mutated" copies of itself (its neighbors)
  - The scoring function value reflects the fitness of the individual
  - the higher the fitness the more likely the individual will survive (i.e., the neighbor will be in the next generation)

Lecture Overview

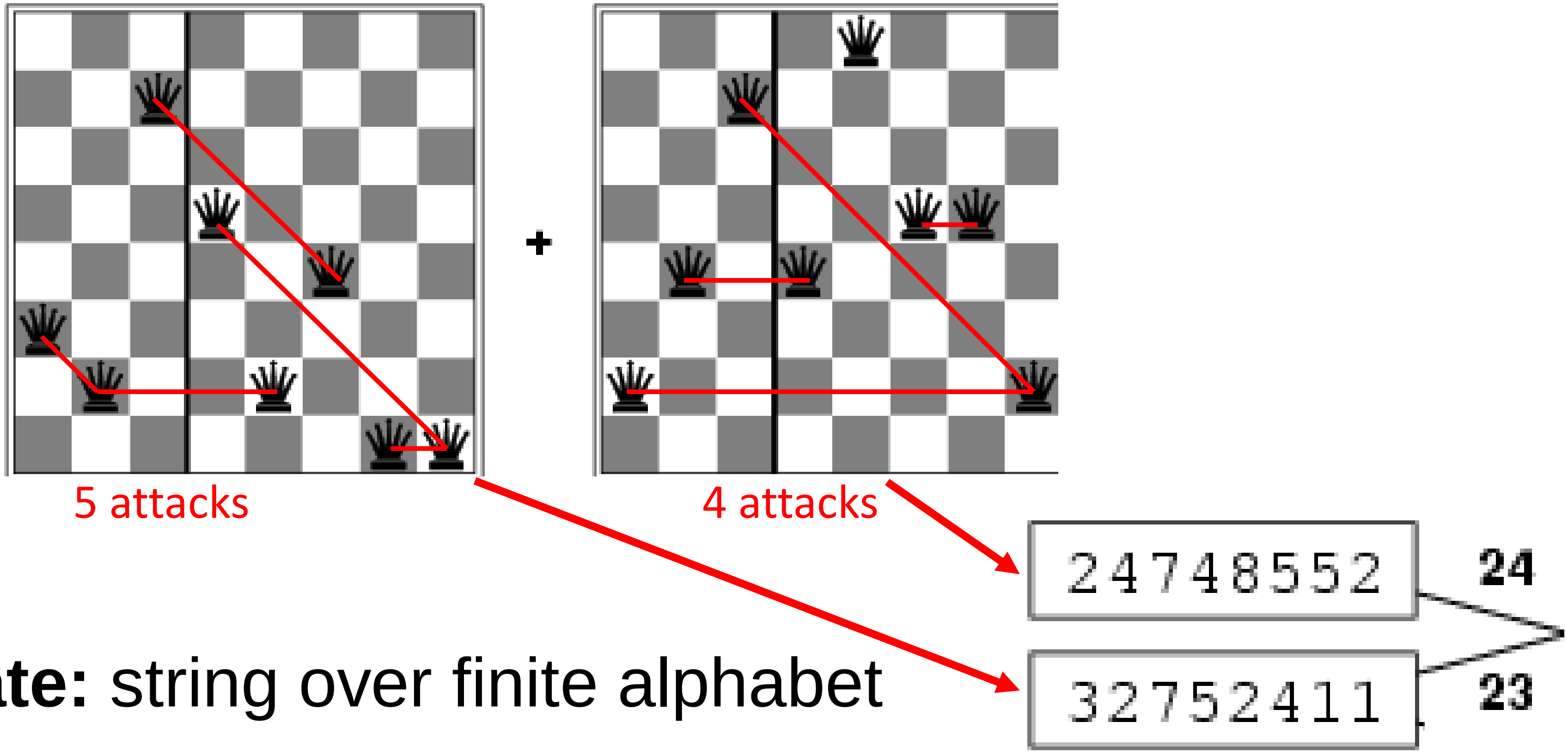
- Recap SLS
- **SLS variants**
  - **Simulated Annealing**
  - **Population Based**
    - ✓ **Beam search**
    - ✓ **Genetic Algorithms**

Population Based SLS: Genetic Algorithms

- Start with *k* randomly generated individuals (population)
- An individual is represented as a string over a finite alphabet *(often a string of 0s and 1s)*
- A successor is generated by combining two parent individuals *(loosely analogous to how DNA is spliced in sexual reproduction)*
- Evaluation/Scoring function (fitness function). Higher values for better individuals.
- Produce the next generation of individuals by selection, crossover, and mutation

Genetic algorithms: Example

Representation and fitness function

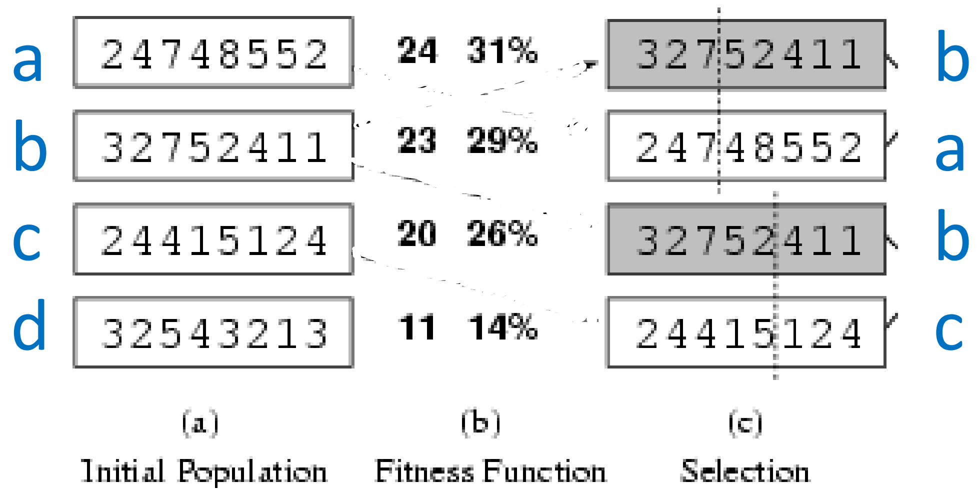


State: string over finite alphabet

Fitness function: # queen pairs that are ***not*** attacking each other (higher is better; max is 8x7/2 = 28)

Genetic algorithms: Example

Selection: common strategy, probability of being chosen for reproduction is directly proportional to fitness score (as with beam search)



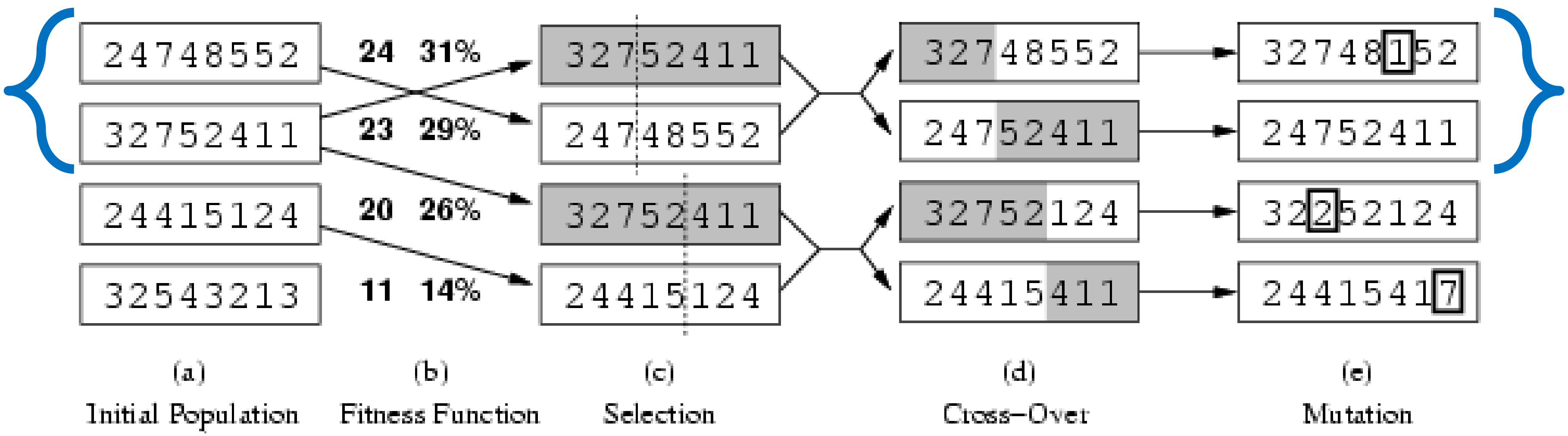
24/(24+23+20+11) = 31%

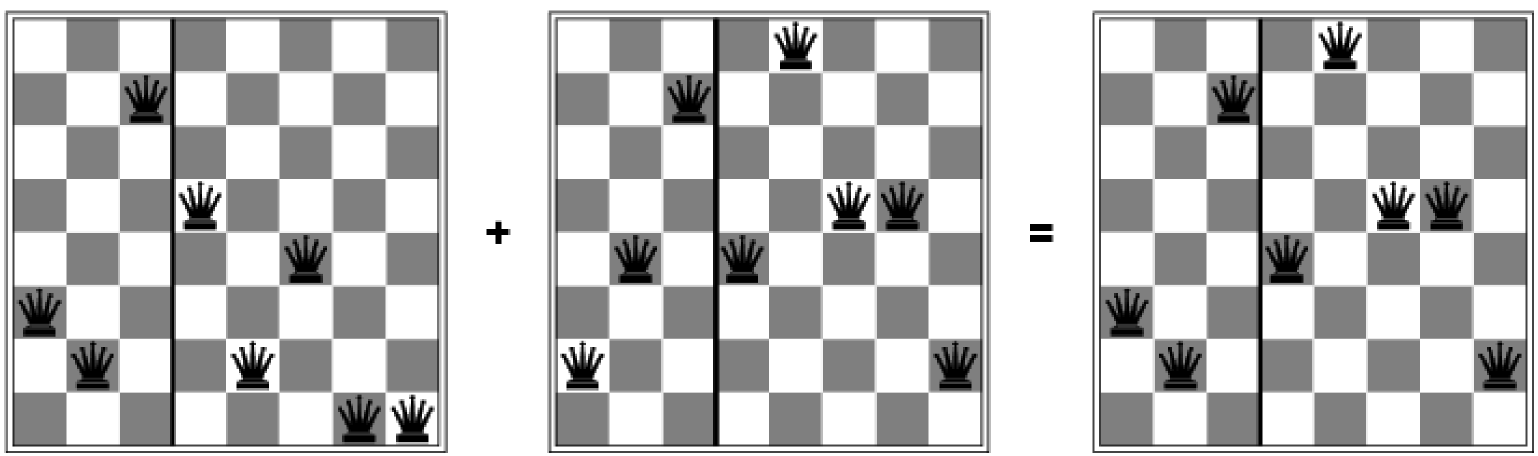
23/(24+23+20+11) = 29%

etc.

Genetic algorithms: Example

Reproduction: cross-over and mutation





Genetic Algorithms: Conclusions

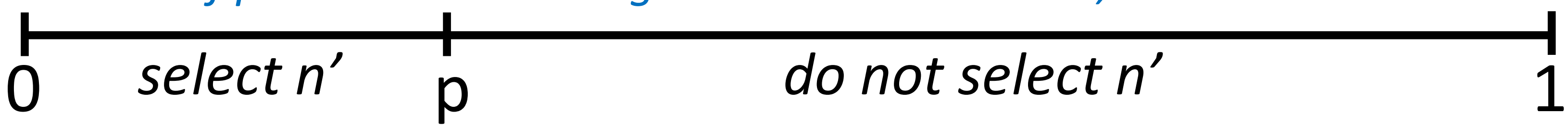
- Their performance is very sensitive to the choice of state representation and fitness function
- **Extremely slow** (not surprising as they are inspired by evolution!)
- Ongoing work to make them practical for real-world problems

Sampling a discrete probability distribution

Approach: select random numbers in [0,1]

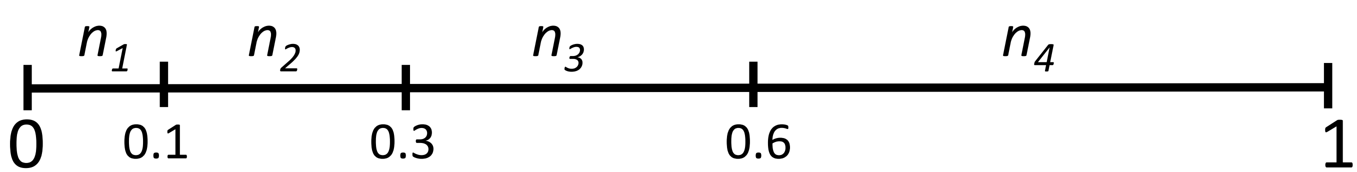
eg. Simulated Annealing – select n' with probability p

If p=0.2 and random generator returns 0.42, do not select n'



eg. Stochastic Beam Search – select k individuals with probability of selection proportional to their value

$p(n_1)=0.1$
$p(n_2)=0.2$
$p(n_3)=0.3$
$p(n_4)=0.4$



If random generator returns 0.42, select n_3

R&R systems we'll cover in this course

		Environment	
Problem		**Deterministic**	**Stochastic**
Static	Constraint Satisfaction	*Variables + Constraints* Search Arc Consistency Local Search	
	Query	*Logics* Search	*Bayesian (Belief) Networks* Variable Elimination
Sequential	Planning	*STRIPS* Search	*Decision Networks* Variable Elimination

Representation
Reasoning Technique

Next: Planning

How to select and organize a sequence of actions to achieve a given goal…

Start **Planning**

- Textbook 6.1 (skip 6.1.3), 6.2, 6.4