

# Stochastic Local Search Variants

Computer Science cpsc322, Lecture 16  
*(Textbook Chpt 4.8)*

February, 8, 2010

~

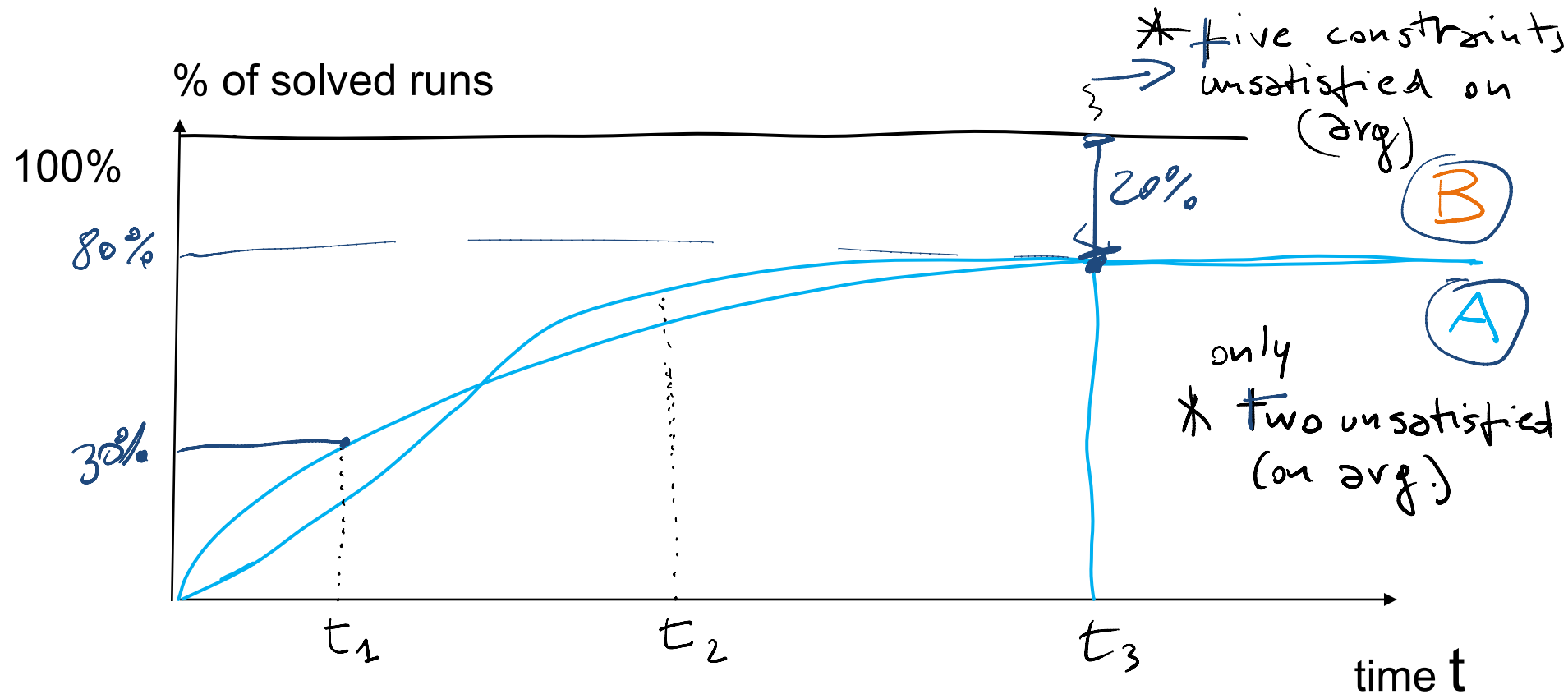
# Lecture Overview

- **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. *1% e.g.*
  - 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)

# Runtime Distributions



Which one would you use if you could wait

- $t = t_1$  ? **A**
- $t = t_2$  ? **B**
- $t = t_3$  ? **A** (because of the quality of the answers on unsolved problems see \*)

# Lecture Overview

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

# Tabu lists

- To avoid search to
  - Immediately going back to previously visited candidate
  - To prevent cycling
- Maintain a **tabu list** of the  $k$  last nodes visited.
  - Don't visit a poss. world that is already on the **tabu list**.
- Cost of this method depends on..... $k$

# Simulated Annealing

- **Key idea:** Change the degree of randomness....
- **Annealing:** a metallurgical process where metals are hardened by being 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):

$h$

- 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.,  $h(n') \geq h(n)$ , adopt it.
- If it isn't an improvement, adopt it probabilistically depending on the difference and a temperature parameter,  $T$ .



$$h(n') < h(n); h(n') - h(n) < 0$$

- we move to  $n'$  with probability  $e^{(h(n') - h(n))/T}$

see next slide

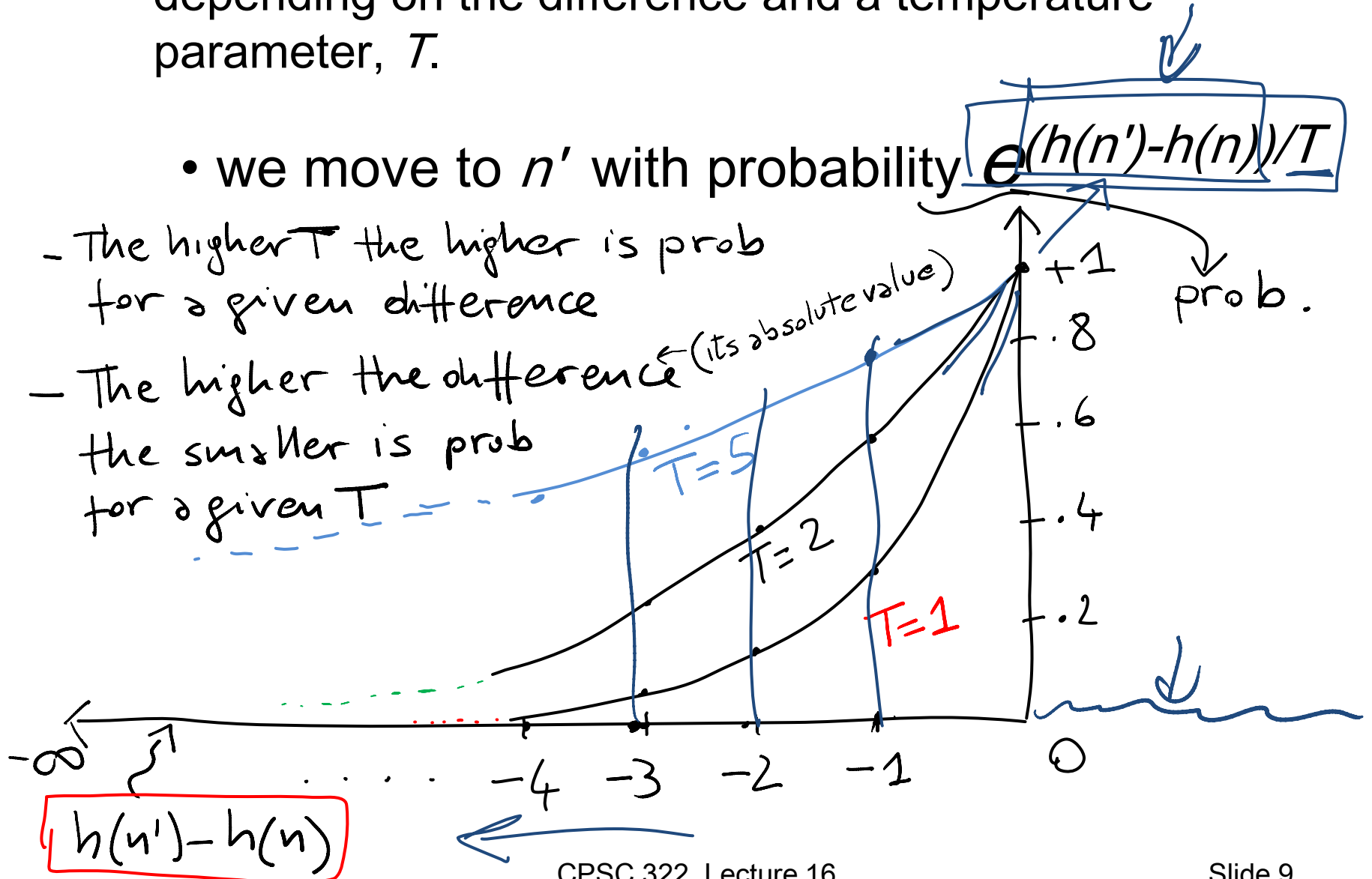




- If it isn't an improvement, adopt it probabilistically depending on the difference and a temperature parameter,  $T$ .

- we move to  $n'$  with probability  $e^{(h(n')-h(n))/T}$

- The higher  $T$  the higher is prob for a given difference
- The higher the difference the smaller is prob for a given  $T$



# Properties of simulated annealing search

One can prove: If  $T$  decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

Widely used in VLSI layout, airline scheduling, etc.

# 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

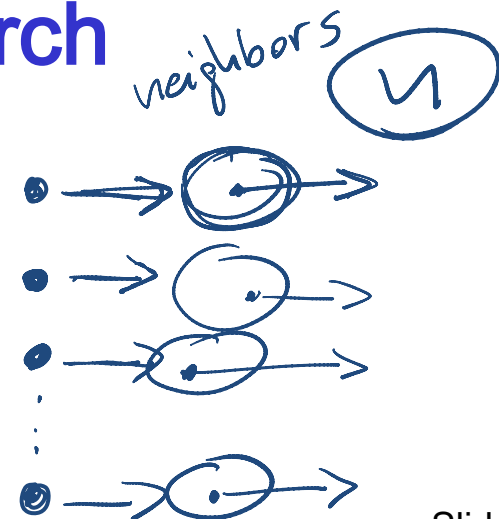
- At every stage, update your population.
- Whenever one individual is a solution, report it.

## Simplest strategy: Parallel Search

- All searches are independent
- Like  $k$  restarts

but more memory  
no reasons to use it!

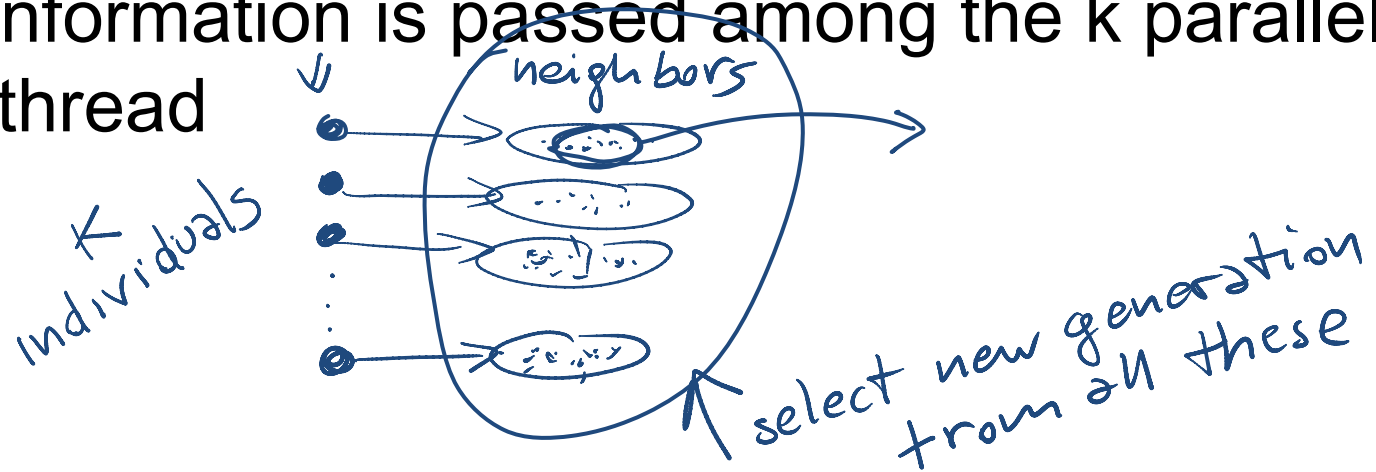
$k$  poss worlds



# 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  $k-1$  all generate bad successors.... the next generation will comprise very similar individuals  $\wedge$

# 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
  - But probability of selection proportional to their value (according to scoring function)

$m$  neighbors  $\{n_1 \dots n_m\}$

$h$ : scoring function

$$\text{Probability of selecting } (n_j) = \frac{h(n_j)}{\sum_i h(n_i)}$$

# 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**

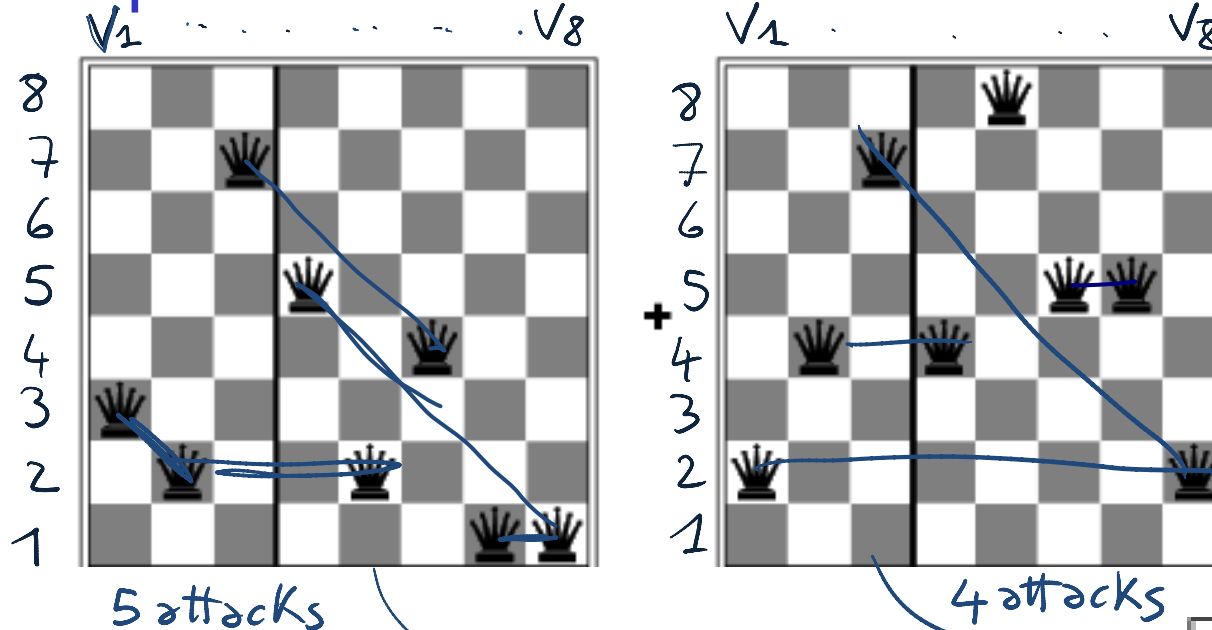


# Population Based SLS: Genetic Algorithms

- 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 8-queen

## Representation and fitness function



# of queen pairs possibly attacking each other

$$\frac{8 \cdot 7}{2} = 28$$

**State:** string over finite alphabet

24748552 **24**

**Fitness function:** higher value

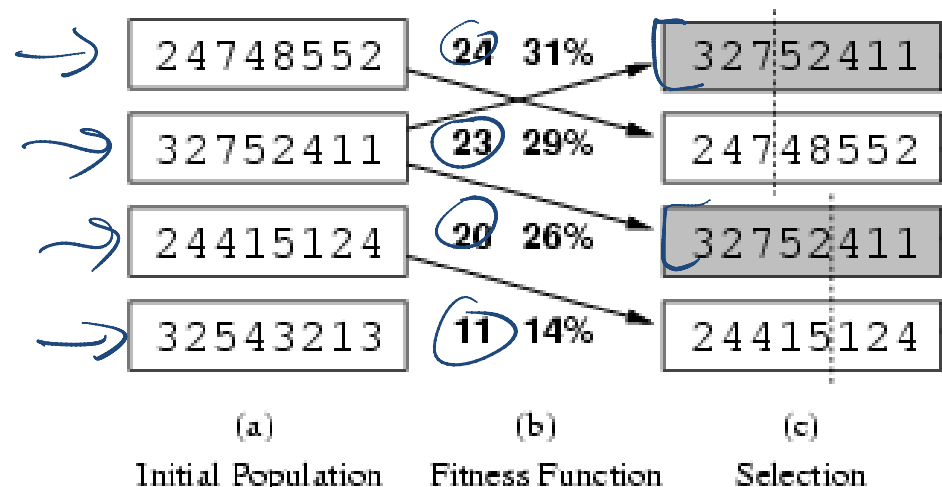
better states. # queen pairs not attacking each other

32752411 **23**

28 - 4  
28 - 5

# Genetic algorithms: Example

**Selection:** common strategy, probability of being chosen for reproduction is directly proportional to fitness score



→  $24/(24+23+20+11) = 31\%$

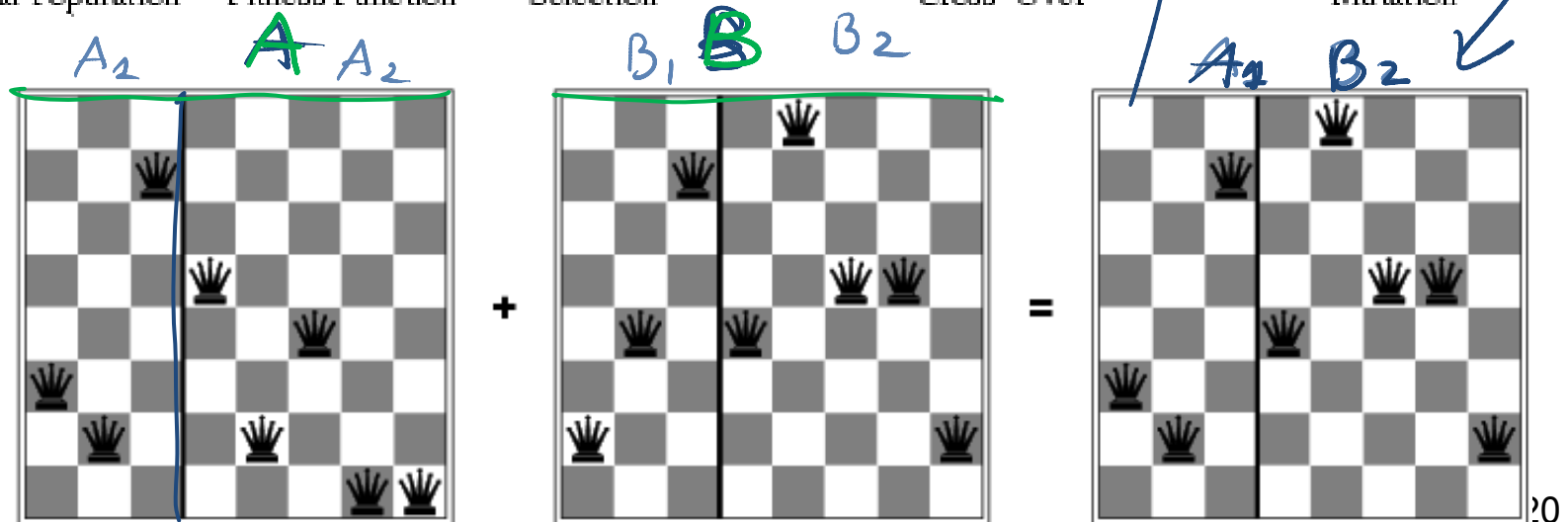
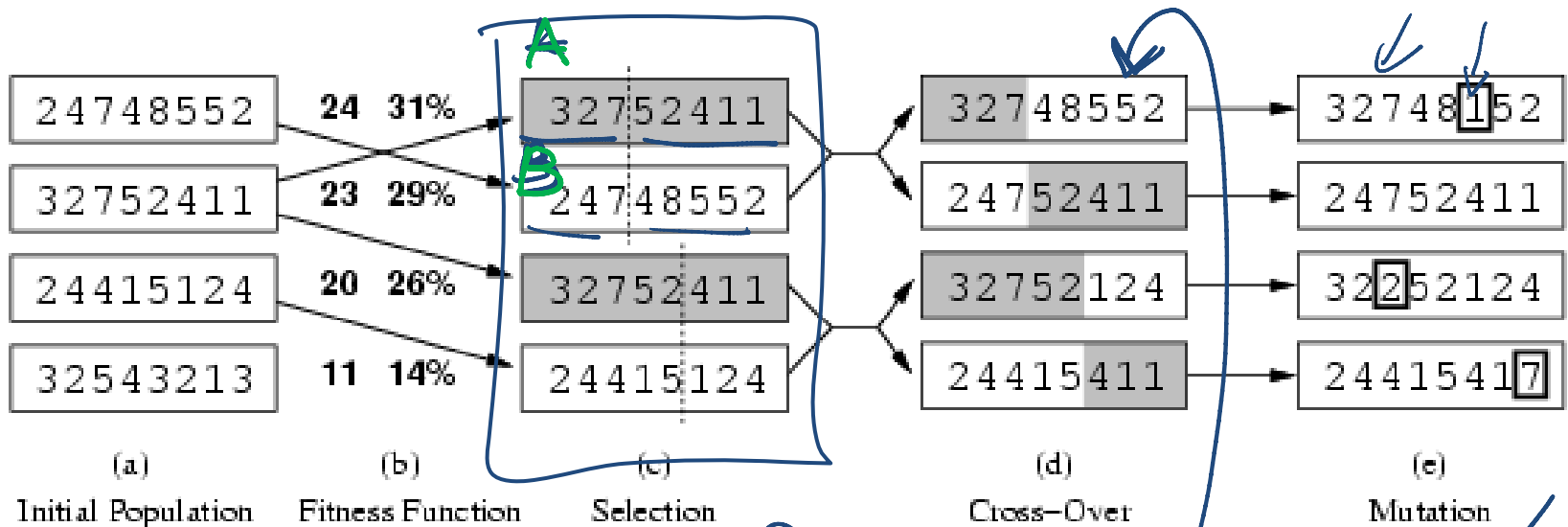
→  $23/(24+23+20+11) = 29\%$  etc

.....

same as Beam Search  
slide 14

# Genetic algorithms: Example

## Reproduction: cross-over and mutation





# Genetic Algorithms: Conclusions

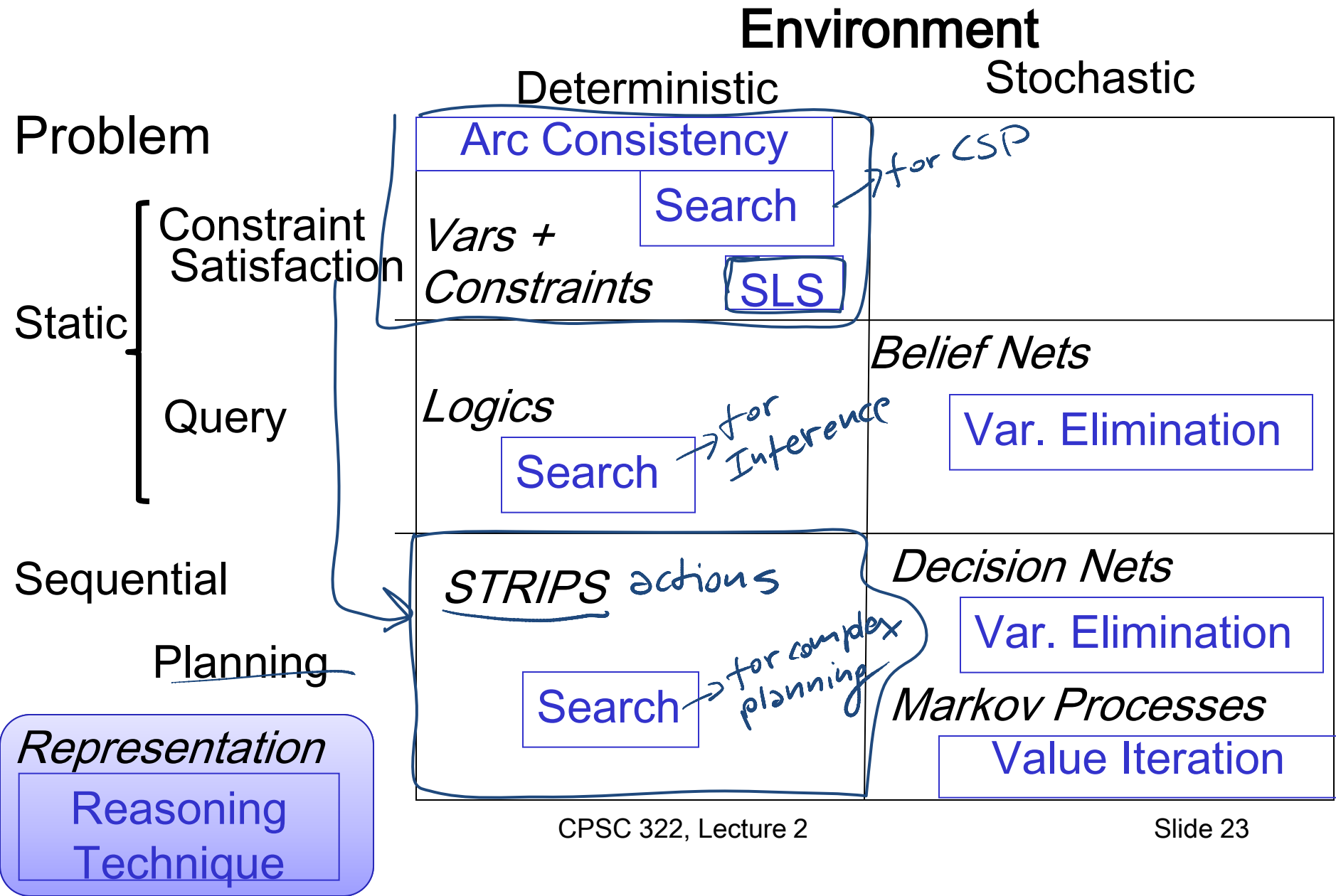
- Their performance is very sensitive to the choice of state representation and fitness function
- **Extremely slow** (not surprising as they are inspired by evolution!)

# 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 .

# Modules we'll cover in this course: R&Rsys



# Next class

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

.....

Start Planning (Chp 8.1-8.2)



# Feedback summary



• Assignments ( <i>prog. , unclear</i> )	10	1	12	(-2)
• Practice Exercises ( <i>too easy</i> )	6	1	4	(+2)
• TAs	6	0	0	(+6)
• Lectures ( <i>more interactive</i> )	12	7	6	(+8)
• Course Topics ( <i>Bayesian Nets?</i> )	9	1	0	(+8)
• Learning Goals	10	0	0	(+10)
• Textbook	14	0	6	(+12)
• Slides ( <i>hard to read</i> )	17	2	2	(+15)
• Alspace	17	0	2	(+15)

# Feedback specific suggestions (>2 people)

- Post precise due textbook readings
- Sync slides right before lecture
- Reading list of recent research papers
- Use clickers

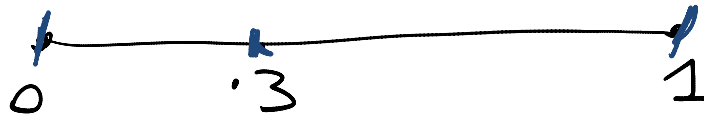
# Sampling a discrete probability distribution

e.g. Sim. Annealing. Select  $n'$  with probability  $P$

$$P = .3$$

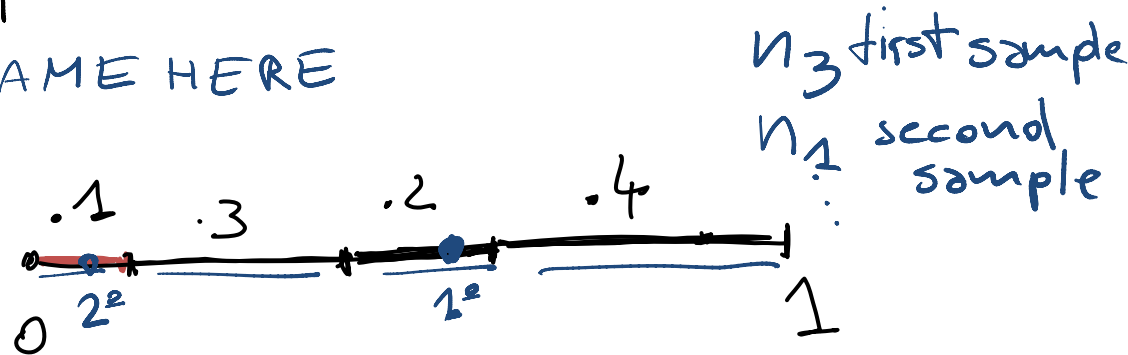
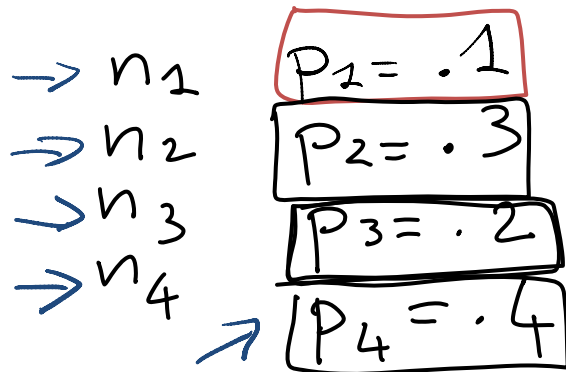
generate random number in  $[0, 1]$

if  $< .3$  accept  $n'$



e.g. Beam Search: Select  $K$  individuals. Probability of selection proportional to their value

SAME HERE



# Systematically solving CSPs: Summary

- Build Constraint Network
- Apply Arc Consistency
  - One domain is empty → *no solution* ↙
  - • Each domain has a single value → *unique solution* ↙
  - Some domains have more than one value →  
*may or may not be a solution* ↙
- Apply Depth-First Search with Pruning ↙
- Split the problem in a number of disjoint cases ↙
  - • Apply Arc Consistency to each case ↙

# CSPs summary

Find a single variable assignment that satisfies all of our constraints (atemporal)

- Systematic Search approach (search space .....?)
  - Constraint network support  $n^2d^3$ 
    - ✓ inference e.g., Arc Consistency (can tell you if solution does not exist)
    - ✓ Decomposition
  - Heuristic Search (degree, min-remaining)
- (Stochastic) Local Search (search space .....?)
  - Huge search spaces and highly connected constraint network but solutions densely distributed
  - No guarantee to find a solution (if one exists).
  - Unable to show that no solution exists