Lecture 3b: Local Search

CSCI 360

Introduction to Artificial Intelligence

USC

Here is where we are…

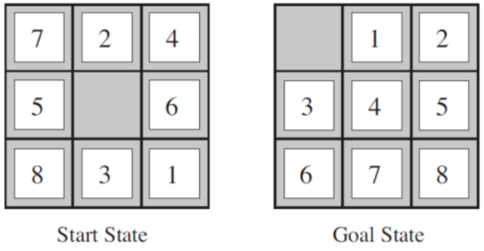
Week	30000D	30282R	Topics	Chapters
1	1/7	1/8	Intelligent Agents	[Ch 1.1-1.4 and 2.1-2.4]
	1/9	1/10	Problem Solving and Search	[Ch 3.1-3.3]
2	1/14	1/15	Uninformed Search	[Ch 3.3-3.4]
	1/16	1/17	Heuristic Search (A*)	[Ch 3.5]
3	1/21	1/22	Heuristic Functions	[Ch 3.6]
	1/23	1/24	Local Search	[Ch 4.1-4.2]
	1/25		Project 1 Out	
4	1/28	1/29	Adversarial Search	[Ch 5.1-5.3]
	1/30	1/31	Knowledge Based Agents	[Ch 7.1-7.3]
5	2/4	2/5	Propositional Logic Inference	[Ch 7.4-7.5]
	2/6	2/7	First-Order Logic	[Ch 8.1-8.4]
	2/8		Project 1 Due	
	2/8		Homework 1 Out	
6	2/11	2/12	Rule-Based Systems	[Ch 9.3-9.4]
	2/13	2/14	Search-Based Planning	[Ch 10.1-10.3]
	2/15		Homework 1 Due	
7	2/18	2/19	SAT-Based Planning	[Ch 10.4]
	2/20	2/21	Knowledge Representation	[Ch 12.1-12.5]
8	2/25	2/26	Midterm Review	
	2/27	2/28	**Midterm Exam**	

Outline

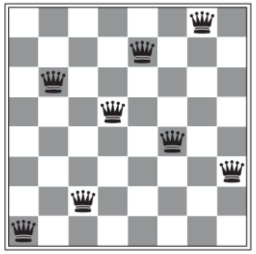
- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- **Local search**
  - Hill-climbing search
  - Simulated annealing
  - Local beam search
  - Genetic algorithm

Goal state vs. Path to goal

- Previously, solution to the search problem is a "sequence of actions" leading to a goal state
  - Example: 8-puzzle



- What if you just want the "goal state", not the "path to goal state"?
  - Example: 8-queens



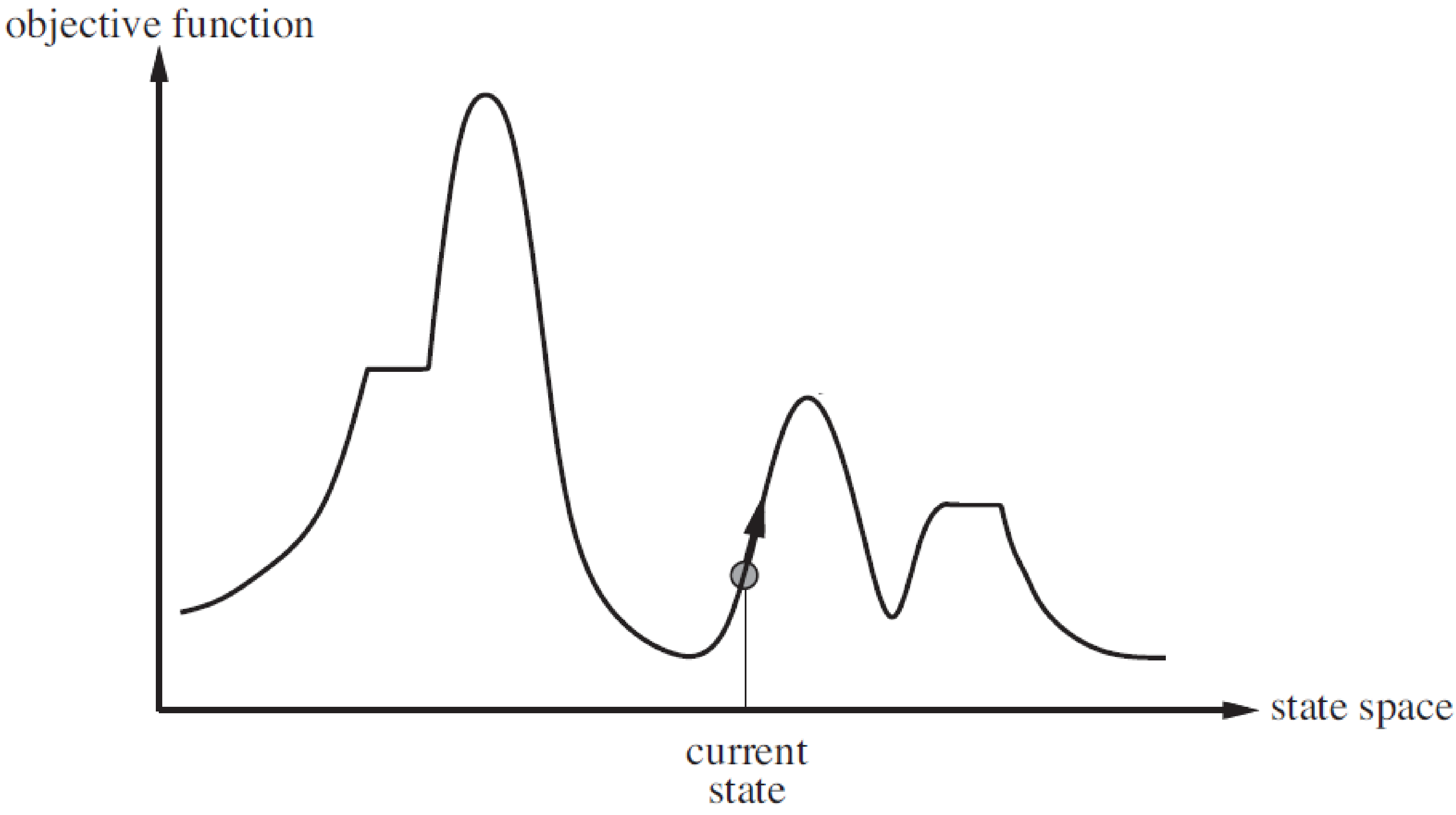
  - In such cases, you may use "**Local Search**"

Local Search: *the idea*

- Operate using a **single, current node** (rather than paths) and move only to **neighbors** of that node

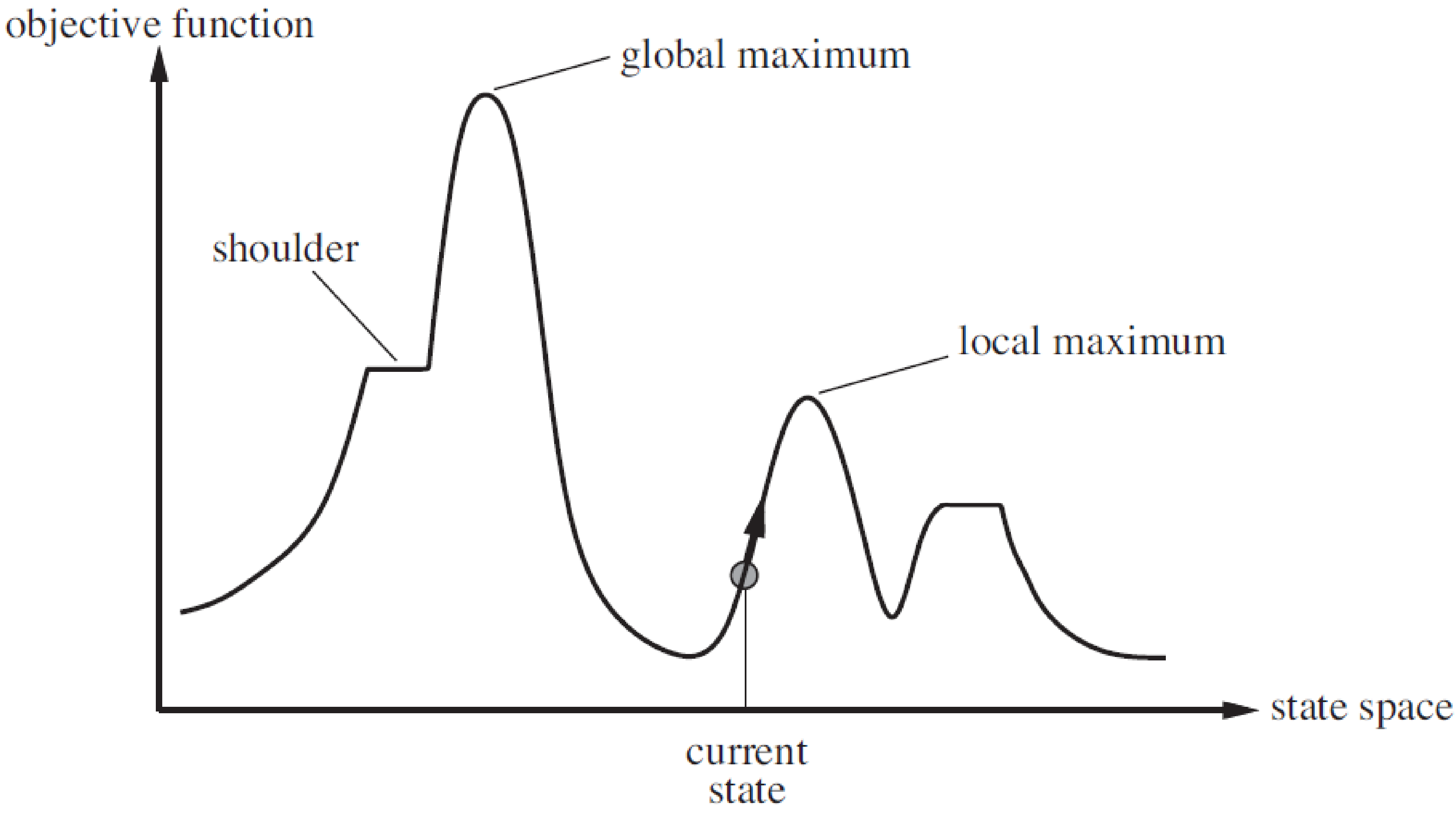
- Advantages
  - Use very little memory
  - Often find reasonable solutions in large and infinite spaces

Local Search: *the state space landscape*



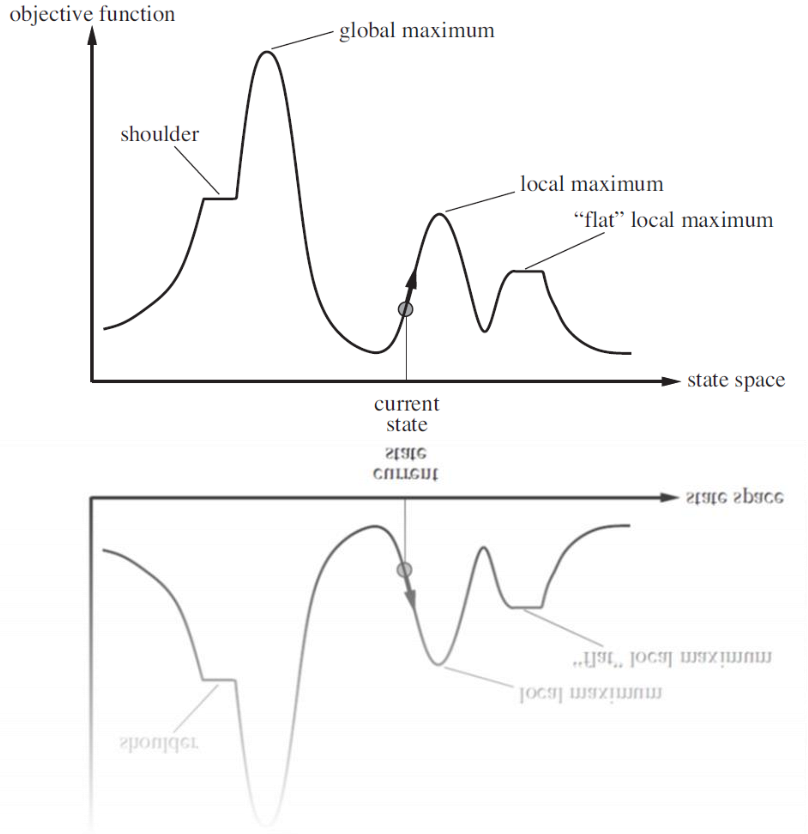
Local Search: *the state space landscape*

- *Local optimal vs. global optimal*



Local Search: state space landscape

- *Maximal vs. Minimal*



Hill-climbing search

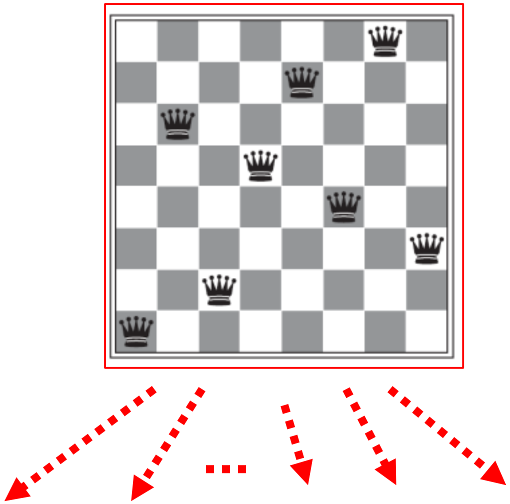
- Continually moves in the direction of increasing value (***steepest-ascent*** version)

function HILL-CLIMBING($problem$) **returns** a state that is a local maximum

    $current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
    loop do
        $neighbor \leftarrow$ a highest-valued successor of $current$
        if neighbor.VALUE $\leq$ current.VALUE **then return** $current$.STATE
        $current \leftarrow neighbor$

Hill-climbing search: 8-queens example

- Successors of a state are generated by moving a single queen to another square
  - How many?
  - 8*7 = 56 successors

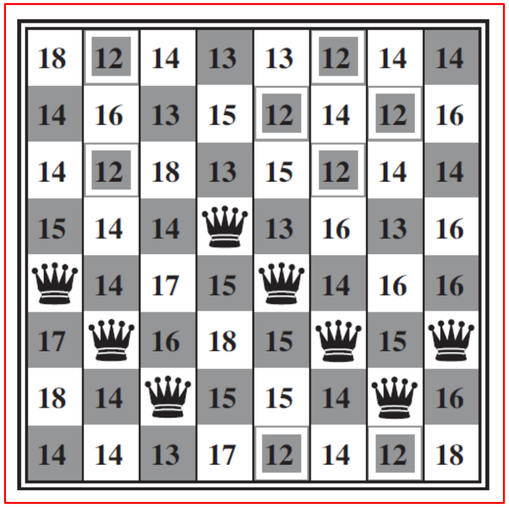


Hill-climbing search: 8-queens example

- Successors of a state are generated by moving a single queen to another square
  - How many?
  - 8*7 = 56 successors

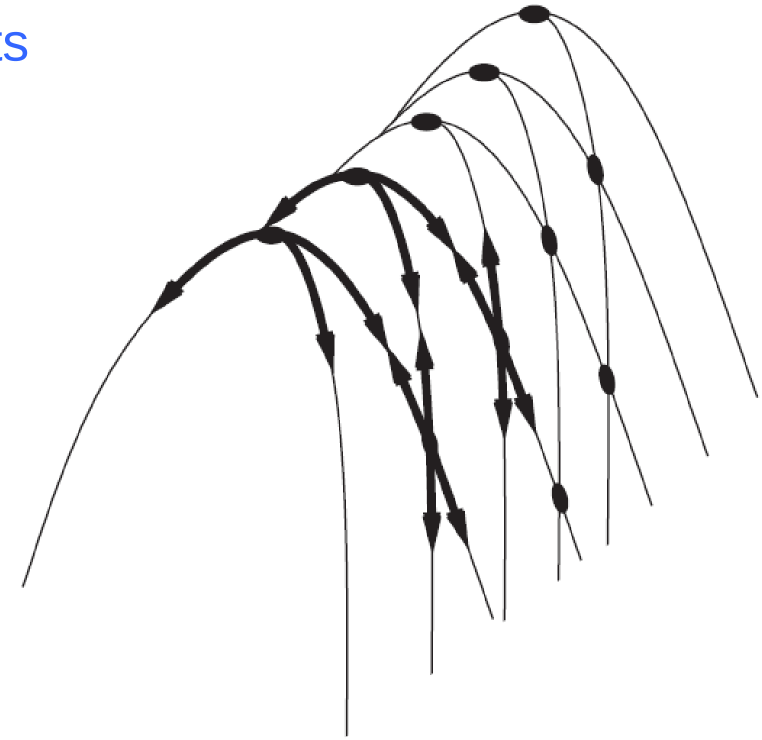
- Heuristic **cost** function
  - the number of “non-attacking” pairs

- It’s a “Greedy Local Search”



Difficulties for hill-climbing search

- Local maxima
  - A peak that is higher than each of its neighboring states, but lower than the global maximum
- Ridges
  - A sequence of local mixima
- Plateaux
  - A flat area of the state-space landscape



Variants of hill-climbing search

- Stochastic hill climbing
  - Choose **at random** among the "uphill" moves, with probability varying with the steepness of the uphill move
- First-choice hill climbing
  - Same as "stochastic hill climbing" but choose the **first successor** that is better than the current state
- Random-restart hill climbing
  - Conduct a series of hill-climbing search from randomly generated initial states

Look like a desperate move…

Success rate of “random-restart”

- If each “restart” has a **probability (*p*)** of success, the expected number of “restarts” would be (***1/p***)

  - **Example**: for 8-queens, normal hill-climbing has 14% success rate

$p=0.14$

$(1/p = 7.14285)$ is roughly 7 restarts

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- **Local search**
  - **Hill-climbing search**
  - **Simulated annealing**
  - **Local beam search**
  - **Genetic algorithm**

Simulated annealing

- Combines “**hill climbing**” with “**random walk**” in a way that yields both efficiency and completeness

Simulated annealing

- Combines “**hill climbing**” with “**random walk**” in a way that yields both efficiency and completeness

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
        schedule, a mapping from time to “temperature”

  current $\leftarrow$ MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**

    next $\leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE
    if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*

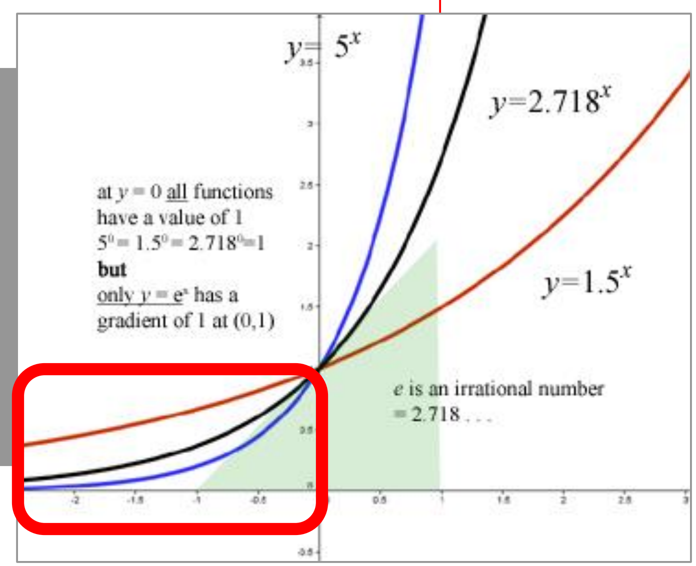
Simulated annealing

- Combines "**hill climbing**" with "**random walk**" in a way that yields both efficiency and completeness

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for t = 1 **to** ∞ **do**

  next ← a randomly selected successor of *current*
  ΔE ← *next*.VALUE – *current*.VALUE
  if $\Delta E > 0$ **then** *current* ← *next*
  else *current* ← *next* only with probability $\boxed{e^{\Delta E/T}}$



Simulated annealing

- Combines "**hill climbing**" with "**random walk**" in a way that yields both efficiency and completeness

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

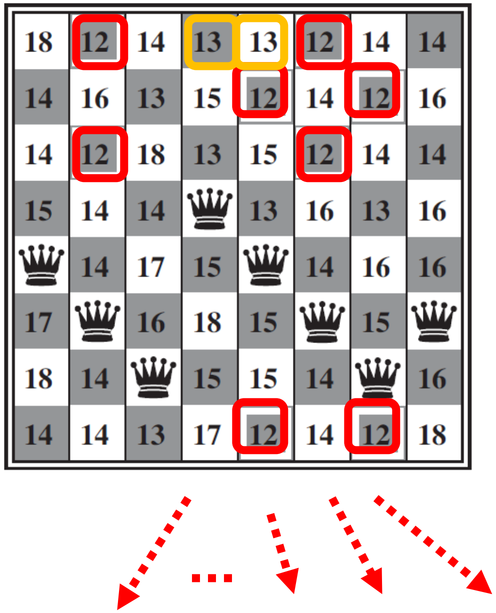
T decreases over time

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- **Local search**
  - **Hill-climbing search**
  - **Simulated annealing**
  - **Local beam search**
  - **Genetic algorithm**

Local beam search

- Keep track of (k) states rather than just one
  - It begins with (k) randomly generated states
  - At each step, all successors of (k) states are generated
  - Select (k) best successors from complete list
  - Repeat



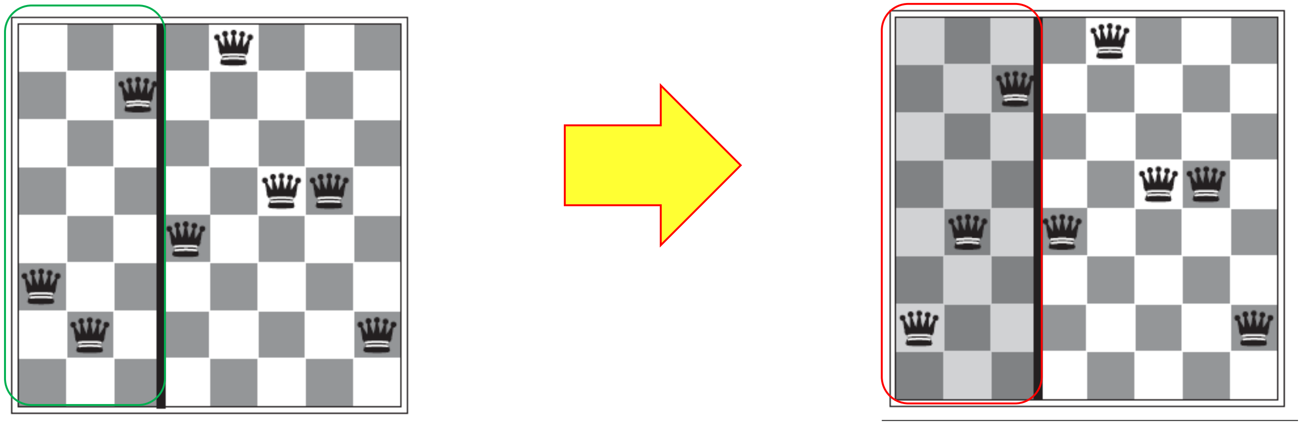
- It differs from (k) random-restart searches
  - Random restarts are independent from each other
  - Local beam search passes information among (k) threads
  - Potential disadvantage: Lack of diversity

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- **Local search**
  - **Hill-climbing search**
  - **Simulated annealing**
  - **Local beam search**
  - **Genetic algorithm**

Raising the granularity of searh

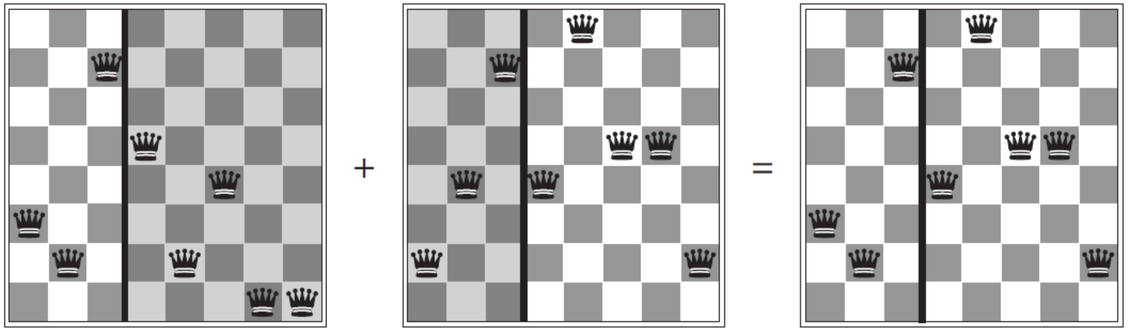
- Move “**3 queens per step**” instead of “**1 queen per step**”
  - **Why?** The block can be a meaningful component of a solution



Question: How to achieve this?

Cross-over

- Create a state by combining components of two states

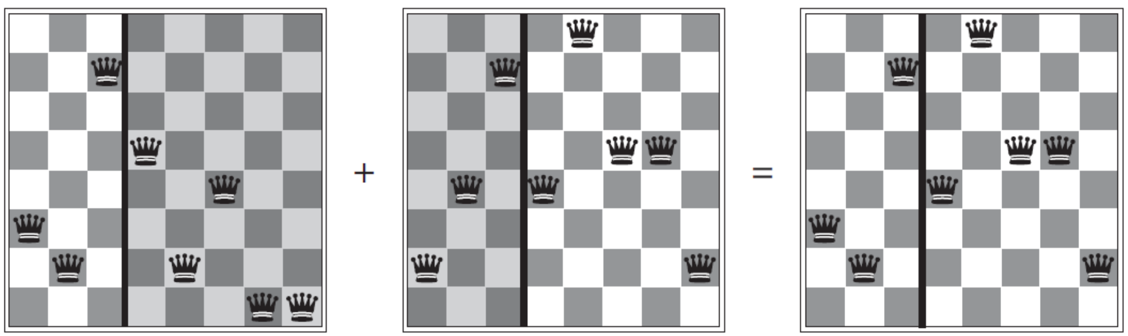


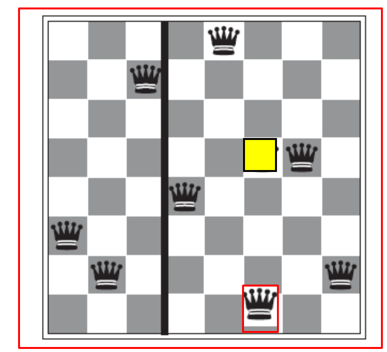
32752411 + 24748552 = 32748552

Problem: Lack of diversity (new components)?

Mutation

- Randomly change the position of a queen

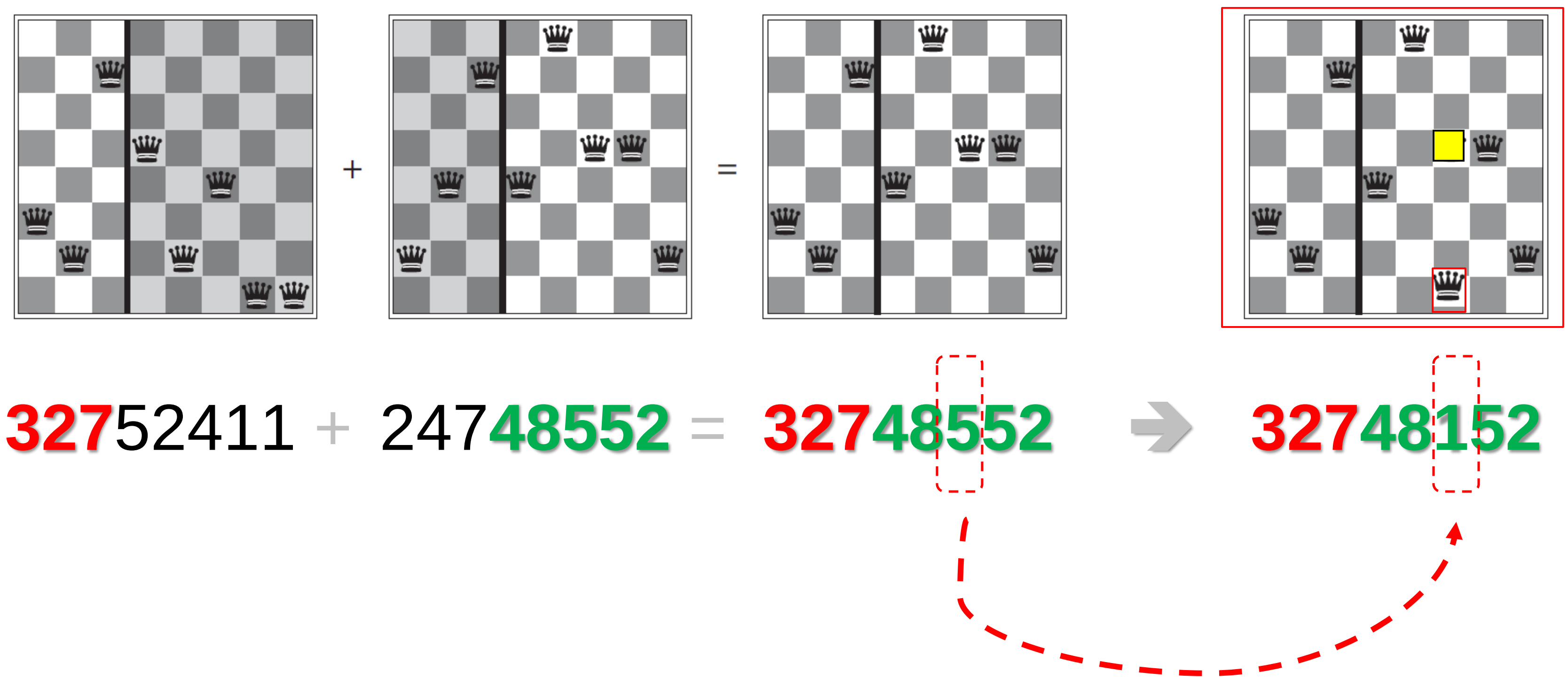




32752411 + 247**48552** = **32748552**

Mutation

- Randomly change the position of a queen



Putting them together

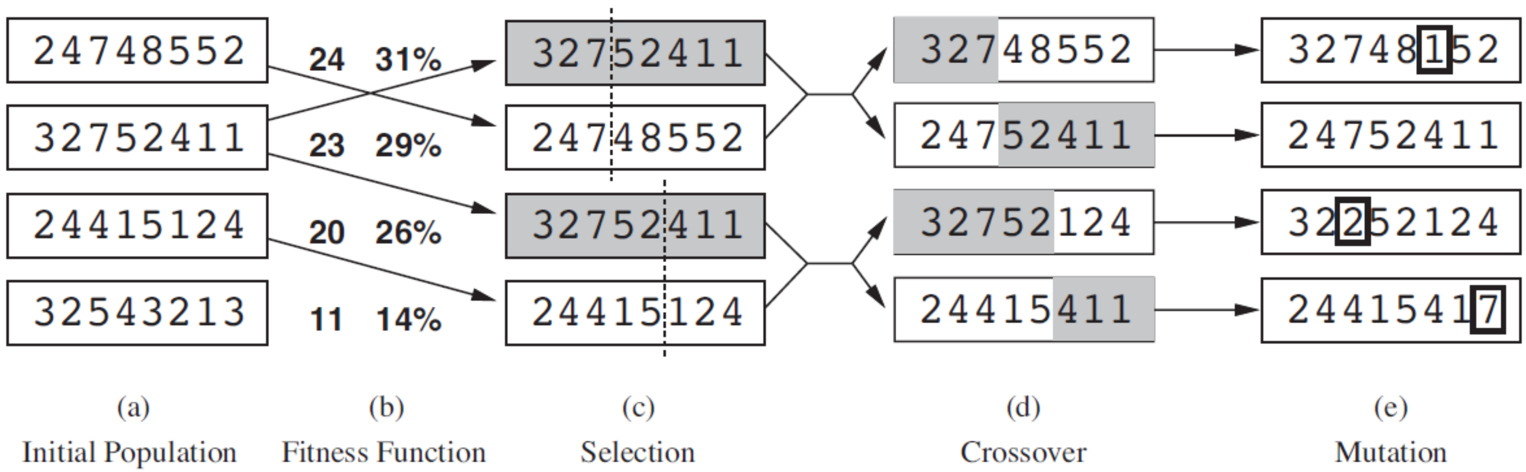
Cross-over + Mutation + Selection

=

Genetic Algorithm

Fitness of states → Selection

- Use most promising states for "cross-over" and "mutation" -- they form a population



For 8-queens: 24, 23, 20, and 11 are numbers of non-attacking pairs

SUM(24,23,20,11) = 78

(24/78) = 31% (23/78) = 29% (20/78) = 26% (11/78) = 14%

Genetic algorithm

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

Selection

Cross-over

Mutation

Outline

- What is AI?
- Problem-solving agent
- Uninformed search
- Informed search (A*)
- **Local search**
  - **Hill-climbing search**
  - **Simulated annealing**
  - **Local beam search**
  - **Genetic algorithm**