Local Search

Evaluate and modify one or more *current states* rather than systematically exploring paths from an initial state.

* Figures are from the textbook site (or drawn by the instructor).
Local Search

Evaluate and modify one or more *current states* rather than systematically exploring paths from an initial state.

Outline

I. Hill climbing

II. Simulated annealing

III. Genetic algorithms

* Figures are from the textbook site (or drawn by the instructor).
Advantages of Local Search

- Use of very little memory.
- Finding good solutions in state spaces *intractable* for a systematic search.
- Useful in pure optimization (e.g., gradient-based descent methods)
Advantages of Local Search

- Use of very little memory.
- Finding good solutions in state spaces *intractable* for a systematic search.
- Useful in pure optimization (e.g., gradient-based descent methods)

Maximize $f(x, y)$

![Diagram showing level curves and steepest ascent]

$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$

$c_1 > c_2 > c_3 > c_4$
State Space Landscape

- Objective function
- Global maximum
- Shoulder
- Local maximum
- "Flat" local maximum
- Current state
- State space
I. Hill Climbing

function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
I. Hill Climbing

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

    current ← problem.INITIAL

while true do

    neighbor ← a highest-valued successor state of current // to break a tile

    if VALUE(neighbor) ≤ VALUE(current) then return current

    current ← neighbor

// random choice
I. Hill Climbing

```plaintext
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current // random choice
        if VALUE(neighbor) ≤ VALUE(current) then return current // to break a tile
        current ← neighbor
```

8-queens problem
I. Hill Climbing

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

\begin{verbatim}
current ← problem.INITIAL
while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
\end{verbatim}

// random choice
// to break a tile

8-queens problem

- \( h(s) = \# \) pairs of queens attacking each other, directly or indirectly, in the state \( s \).
I. Hill Climbing

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

\begin{align*}
\text{current} & \leftarrow \text{problem.INITIAL} \\
\text{while true } \text{ do} & \\
\quad \text{neighbor} & \leftarrow \text{a highest-valued successor state of } \text{current} \quad // \text{ random choice} \\
\quad \text{if VALUE(neighbor) \leq VALUE(current) then return current} & \quad // \text{ to break a tile} \\
\quad \text{current} & \leftarrow \text{neighbor}
\end{align*}

8-queens problem

- \( h(s) = \# \text{ pairs of queens attacking each other, directly or indirectly, in the state } s. \)
I. Hill Climbing

\[
\text{function HILL-CLIMBING(problem) returns a state that is a local maximum}
\]
\[
current \leftarrow \text{problem.INITIAL}
\]
\[
\text{while true do}
\]
\[
\quad \text{neighbor} \leftarrow \text{a highest-valued successor state of current} \quad \text{// random choice}
\]
\[
\quad \text{if VALUE(neighbor) \leq VALUE(current) then return current} \quad \text{// to break a tile}
\]
\[
\quad current \leftarrow \text{neighbor}
\]

8-queens problem

- \( h(s) = \# \text{ pairs of queens attacking each other, directly or indirectly, in the state } s. \)
- Successor is a state generated from relocating a queen in the same column.

\[
h = 17
\]
I. Hill Climbing

function HILL-CLIMBING\((\text{problem})\) \textbf{returns} a state that is a local maximum

\[\text{current} \leftarrow \text{problem.\text{INITIAL}}\]

\textbf{while} true \textbf{do}

\[\text{neighbor} \leftarrow \text{a highest-valued successor state of current} \quad // \text{random choice}\]

\textbf{if} \ \text{VALUE}(\text{neighbor}) \leq \text{VALUE}(\text{current}) \ \textbf{then} \ \textbf{return} \ \text{current}\]

\[\text{current} \leftarrow \text{neighbor} \quad // \text{to break a tile}\]

8-queens problem

\begin{itemize}
  \item \(h(s) = \#\) pairs of queens attacking each other, directly or indirectly, in the state \(s\).
  \item Successor is a state generated from relocating a queen in the same column.
\end{itemize}

\[h = 17\]
1. Hill Climbing

function HILL-CLIMBING(\textit{problem}) returns a state that is a local maximum
\texttt{current} \leftarrow \texttt{problem}.INITIAL
\textbf{while} true \textbf{do}
\hspace{1em} \texttt{neighbor} \leftarrow \text{a highest-valued successor state of current} \hspace{1em} \text{// random choice}
\hspace{1em} \textbf{if} \ \texttt{VALUE(neighbor)} \leq \texttt{VALUE(current)} \textbf{then return current} \hspace{1em} \text{// to break a tile}
\hspace{1em} \texttt{current} \leftarrow \texttt{neighbor}

8-queens problem

- $h(s) =$ # pairs of queens attacking each other, directly or indirectly, in the state $s$.
- Successor is a state generated from relocating a queen in the same column.
I. Hill Climbing

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

current ← problem.INITIAL

while true do

    neighbor ← a highest-valued successor state of current

    if VALUE(neighbor) ≤ VALUE(current) then return current

    current ← neighbor

end

// random choice
// to break a tile

- **8-queens problem**

  - \( h(s) = \# \text{ pairs of queens attacking each other, directly or indirectly, in the state } s. \)
  
  - Successor is a state generated from relocating a queen in the same column.

  - \( h(s) = 12 \) for the best successor.

- \( h = 17 \)
I. Hill Climbing

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

\[ \text{current} \leftarrow \text{problem.INITIAL} \]

\[ \text{while true do} \]

\[ \text{neighbor} \leftarrow \text{a highest-valued successor state of current} \]

\[ \text{if VALUE(neighbor) \leq VALUE(current) then return current} \]

\[ \text{current} \leftarrow \text{neighbor} \]

// random choice
// to break a tile

8-queens problem

- \( h(s) = \) # pairs of queens attacking each other, directly or indirectly, in the state \( s \).

- Successor is a state generated from relocating a queen in the same column.

- \( h(s) = 12 \) for the best successor.
I. Hill Climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
    // random choice to break a tile
```

8-queens problem

- $h(s) = \#$ pairs of queens attacking each other, directly or indirectly, in the state $s$.
- Successor is a state generated from relocating a queen in the same column.
- $h(s) = 12$ for the best successor.

8-way tie!.
I. Hill Climbing

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

current ← problem.INITIAL

while true do

neighbor ← a highest-valued successor state of current

if VALUE(neighbor) ≤ VALUE(current) then return current

current ← neighbor

// random choice
// to break a tile

8-queens problem

- $h(s) = \#$ pairs of queens attacking each other, directly or indirectly, in the state $s$.

- Successor is a state generated from relocating a queen in the same column.

- $h(s) = 12$ for the best successor.

8-way tie!

Hill climbing randomly picks one.

$h = 17$
Efficiency?
Efficiency?

5 moves
Efficiency?

- 5 moves
- $h = 1$
Hill climbing can make rapid progress toward a solution.
Drawback of Hill Climbing (1)

Hill climbing terminates when a peak is reached with no neighbor having a higher value.
Hill climbing terminates when a peak is reached with no neighbor having a higher value.

- **Local maximum**

  Not the global maximum.
Drawback of Hill Climbing (1)

Hill climbing terminates when a peak is reached with no neighbor having a higher value.

♠ Local maximum

Not the global maximum.
Drawback of Hill Climbing (1)

Hill climbing terminates when a peak is reached with no neighbor having a higher value.

♠ Local maximum

Not the global maximum.

Every move of one queen introduces more conflicts.
Drawback of Hill Climbing (1)

Hill climbing terminates when a peak is reached with no neighbor having a higher value.

- **Local maximum**

  Not the global maximum.

  Hill climbing in the vicinity of a local maximum will be drawn toward it and then get stuck there.

  Every move of one queen introduces more conflicts.
Drawback of Hill Climbing (2)

- **Ridge:** A sequence of local maxima difficult to navigate.

At each local maximum, all available actions are downhill.
Drawback of Hill Climbing (2)

- **Ridge:** A sequence of local maxima difficult to navigate.
  - At each local maximum, all available actions are downhill.

- **Plateaus:** No uphill actions exist.
Drawback of Hill Climbing (2)

- **Ridge**: A sequence of local maxima difficult to navigate.

At each local maximum, all available actions are downhill.

- **Plateaus**: No uphill actions exist.
Variations of Hill Climbing

- Stochastic hill climbing
  - Random selection among the uphill moves.
  - Probability of selection varying with steepness..
Variations of Hill Climbing

♦ Stochastic hill climbing
  • Random selection among the uphill moves.
  • Probability of selection varying with steepness..

♦ First-choice hill climbing
  • Random generation of successors until a better (than the current) one is found.
  • Useful when many successors exist and/or the objective function is costly to evaluate.
Variations of Hill Climbing

♦ **Stochastic hill climbing**
  - Random selection among the uphill moves.
  - Probability of selection varying with steepness.

♦ **First-choice hill climbing**
  - Random generation of successors until a better (than the current) one is found.
  - Useful when many successors exist and/or the objective function is costly to evaluate.

♦ **Random restart hill climbing**
  - Restart search from random initial state.
II. Simulated Annealing

Annealing: Heat a metal to a high temperature and then gradually cool it, allowing the material to reach a low-energy crystalline state so it is hardened.
Annealing: Heat a metal to a high temperature and then gradually cool it, allowing the material to reach a low-energy crystalline state so it is hardened.

* from http://www.cs.us.es/~fsancho/?e=206
II. Simulated Annealing

Annealing: Heat a metal to a high temperature and then gradually cool it, allowing the material to reach a low-energy crystalline state so it is hardened.

- Start by shaking hard (i.e., at high temperature).
- Gradually reduce the intensity of shaking (i.e., lower the temperature).

* from [http://www.cs.us.es/~fsancho/?e=206](http://www.cs.us.es/~fsancho/?e=206)
Simulated Annealing Algorithm

```python
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    current ← problem.INITIAL
    for t = 1 to ∞ do
        T ← schedule(t)
        if T = 0 then return current // solution
        next ← a randomly selected successor of current
        ΔE ← VALUE(current) − VALUE(next)
        if ΔE > 0 then current ← next
        else current ← next only with probability e^{−ΔE/T}
```

Temperature \(\rightarrow\) Minimization

Badness \(\rightarrow\) \(-\Delta E\)
Simulated Annealing Algorithm

```python
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    current ← problem.INITIAL
    for t = 1 to ∞ do
        T ← schedule(t)
        if T = 0 then return current // solution
        next ← a randomly selected successor of current
        ΔE ← VALUE(current) − VALUE(next)
        if ΔE > 0 then current ← next
        else current ← next only with probability \[ e^{\frac{\Delta E}{T}} \] // \[ \Delta E \leq 0 \]
```

Minimization
Simulated Annealing Algorithm

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current // solution
    next ← a randomly selected successor of current
    ΔE ← VALUE(current) − VALUE(next)
    if ΔE > 0 then current ← next
    else current ← next only with probability $e^{\Delta E / T}$
  end for

• Accept the next state if it is an improvement ($\Delta E > 0$).

Minimization

Temperature → Badness $-\Delta E$ → $e^{\Delta E / T}$ // $\Delta E \leq 0$
Simulated Annealing Algorithm

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current // solution
    next ← a randomly selected successor of current
    ΔE ← VALUE(current) - VALUE(next)
    if ΔE > 0 then current ← next
    else current ← next only with probability e^{ΔE/T} // ΔE ≤ 0
```

- Accept the next state if it is an improvement (ΔE > 0).
- Otherwise, accept it with a probability that decreases exponentially
  - as the badness −ΔE of the move increases, and
  - as the “temperature” goes down.
Simulated Annealing Algorithm

```
f\text{unction SIMULATED-ANNEALING}(\text{problem, schedule}) \text{return}s a solution state
  \text{current} \leftarrow \text{problem.\textsc{INITIAL}}
  \text{for } t = 1 \text{ to } \infty \text{ do}
    T \leftarrow \text{schedule}(t)
    \text{if } T = 0 \text{ then return current // solution}
    \text{next} \leftarrow \text{a randomly selected successor of current}
    \Delta E \leftarrow \text{VALUE(current)} - \text{VALUE(next)}
    \text{if } \Delta E > 0 \text{ then } \text{current} \leftarrow \text{next}
    \text{else current} \leftarrow \text{next only with probability } e^{\Delta E / T}
\text{Minimization}

\Delta E \leq 0
```

- Accept the next state if it is an improvement ($\Delta E > 0$).
- Otherwise, accept it with a probability that decreases exponentially
  - as the badness $-\Delta E$ of the move increases, and
  - as the “temperature” goes down.

Bad moves are more tolerated at the start when $T$ is high, and become less likely as $T$ decreases.
Simulated Annealing Algorithm

function SIMULATED-ANNEALING(problem, schedule) returns a solution state

\begin{itemize}
  \item current \leftarrow \text{problem}.\text{INITIAL}
  \item for \( t = 1 \) to \( \infty \) do
    \begin{align*}
      & T \leftarrow \text{schedule}(t) \\
      & \text{if } T = 0 \text{ then return current} \quad \text{// solution} \\
      & \text{next} \leftarrow \text{a randomly selected successor of } \text{current} \\
      & \Delta E \leftarrow \text{VALUE}(\text{current}) - \text{VALUE}(\text{next}) \\
      & \text{if } \Delta E > 0 \text{ then } \text{current} \leftarrow \text{next} \\
      & \text{else } \text{current} \leftarrow \text{next} \text{ only with probability } e^{\frac{-\Delta E}{T}} \\
    \end{align*}
  \end{itemize}

- Accept the next state if it is an improvement (\( \Delta E > 0 \)).
- Otherwise, accept it with a probability that decreases exponentially
  - as the badness \(-\Delta E\) of the move increases, and
  - as the “temperature” goes down.

- Escape local minima by allowing bad moves.

Minimization

\[ \mathcal{E} \leftarrow e^{\Delta E/T} \quad \text{// } \Delta E \leq 0 \]
More About SA

- $T \to 0$ slowly enough
More About SA

- $T \to 0$ slowly enough

\[ e^{\Delta E / T} \]

A property of Boltzmann distribution $e^{\Delta E / T}$ guarantees the global minimum with probability $\to 1$. 
More About SA

- $T \to 0$ slowly enough

\[ e^{\Delta E/T} \]

A property of Boltzmann distribution $e^{\Delta E/T}$ guarantees the global minimum with probability $\to 1$.

- Commonly used $T \leftarrow cT$ with constant $c < 1$ and close to 1 at each step.
More About SA

- $T \to 0$ slowly enough

  $e^{\Delta E/T}$ guarantees the global minimum with probability $\to 1$.

- Commonly used $T \leftarrow cT$ with constant $c < 1$ and close to 1 at each step.

- Applied to many problems:
  
  - VLSL layout
  - factory scheduling
  - aircraft trajectory planning
  - NP-hard optimization (i.e., the traveling salesman problem)
  - large-scale stochastic optimization tasks
Local Beam Search

Keep track of $k$ states rather than one.

1. Start with $k$ randomly generated states.
2. Generate all their successors.
3. Stop if any successor is a goal.
4. Otherwise, keep the $k$ best successors and go back to step 2.
Local Beam Search

Keep track of $k$ states rather than one.

1. Start with $k$ randomly generated states.
2. Generate all their successors.
3. Stop if any successor is a goal.
4. Otherwise, keep the $k$ best successors and go back to step 2.

May suffer from a lack of diversity among the $k$ states.
Local Beam Search

Keep track of $k$ states rather than one.

1. Start with $k$ randomly generated states.
2. Generate all their successors.
3. Stop if any successor is a goal.
4. Otherwise, keep the $k$ best successors and go back to step 2.

- May suffer from a lack of diversity among the $k$ states.

Solution: stochastic beam search which chooses successors with probabilities proportional to their values.
III. Evolutionary Algorithms

- Also called *genetic algorithms*.
- Inspired by natural selection in biology.
III. Evolutionary Algorithms

- Also called genetic algorithms.
- Inspired by natural selection in biology.

1. Start with a population of $k$ randomly generated states (individuals).
III. Evolutionary Algorithms

✩ Also called genetic algorithms.

✩ Inspired by natural selection in biology.

1. Start with a population of $k$ randomly generated states (individuals).

2. Select the most fit individuals to become parents of the next generation
III. Evolutionary Algorithms

- Also called *genetic algorithms*.
- Inspired by natural selection in biology.

1. Start with a population of $k$ randomly generated states (individuals).
2. Select the *most fit* individuals to become parents of the next generation.
3. Combine every $\rho$ parents to form an offspring (typically $\rho = 2$).
III. Evolutionary Algorithms

- Also called *genetic algorithms*.
- Inspired by natural selection in biology.

1. Start with a population of $k$ randomly generated states (individuals).

2. Select the *most fit* individuals to become parents of the next generation.

3. Combine every $\rho$ parents to form an offspring (typically $\rho = 2$).

   **Crossover:** Split each of the parent strings and recombine the parts to form two children.
III. Evolutionary Algorithms

- Also called *genetic algorithms*.
- Inspired by natural selection in biology.

1. Start with a population of $k$ randomly generated states (individuals).
2. Select the *most fit* individuals to become parents of the next generation.
3. Combine every $\rho$ parents to form an offspring (typically $\rho = 2$).

**Crossover:** Split each of the parent strings and recombine the parts to form two children.

**Mutation:** Randomly change the bits of an offspring.
III. Evolutionary Algorithms

Also called *genetic algorithms*.

Inspired by natural selection in biology.

1. Start with a population of $k$ randomly generated states (individuals).

2. Select the *most fit* individuals to become parents of the next generation.

3. Combine every $\rho$ parents to form an offspring (typically $\rho = 2$).

   - **Crossover**: Split each of the parent strings and recombine the parts to form two children.
   - **Mutation**: Randomly change the bits of an offspring.
   - **Culling**: All individuals below a threshold are discarded from the population.
III. Evolutionary Algorithms

- Also called *genetic algorithms*.
- Inspired by natural selection in biology.

1. Start with a population of \( k \) randomly generated states (individuals).

2. Select the *most fit* individuals to become parents of the next generation.

3. Combine every \( \rho \) parents to form an offspring (typically \( \rho = 2 \)).

   **Crossover:** Split each of the parent strings and recombine the parts to form two children.

   **Mutation:** Randomly change the bits of an offspring.

   **Culling:** All individuals below a threshold are discarded from the population.

4. Go back to step 2 and repeat until *sufficiently fit* states are discovered (in which case the best one is chosen as a solution).
Genetic Algorithm on 8-Queen

(a) Initial Population
(b) Fitness Function
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

(a) Initial Population

(b) Fitness Function

- 24748552
- 32752411
- 24415124
- 32543213
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

Score = # non-attacking pairs of queens

(a) Initial Population
(b) Fitness Function
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

Score = 
# non-attacking pairs of queens

(a) Initial Population

(b) Fitness Function

14% = \frac{11}{24 + 23 + 20 + 11}
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

Score = # non-attacking pairs of queens

(a) Initial Population
(b) Fitness Function
(c) Selection

14% = \frac{11}{24 + 23 + 20 + 11}
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

Score = \# non-attacking pairs of queens

(a) Initial Population
(b) Fitness Function
(c) Selection
(d) Crossover

\[ 14\% = \frac{11}{24 + 23 + 20 + 11} \]
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

Score = # non-attacking pairs of queens

(a) Initial Population
(b) Fitness Function
(c) Selection
14% = \( \frac{11}{24 + 23 + 20 + 11} \)

(d) Crossover
(e) Mutation
Genetic Algorithm on 8-Queen

Row number of the queen in column 1

Score = # non-attacking pairs of queens

(a) Initial Population
(b) Fitness Function
(c) Selection
(d) Crossover
(e) Mutation

14% = \frac{11}{24 + 23 + 20 + 11}

Move the queen in column 8 to a random square (7).
Crossover

(d)
Crossover
Crossover
Crossover

(d) Crossover
Genetic Algorithm (Pseudocode)

function GENETIC-ALGORITHM(population, fitness) returns an individual
 repeat
   weights $\leftarrow$ WEIGHTED-BY(population, fitness)
   population2 $\leftarrow$ empty list
   for $i = 1$ to SIZE(population) do
     parent1, parent2 $\leftarrow$ WEIGHTED-RANDOM-CHOICES(population, weights, 2)
     child $\leftarrow$ REPRODUCE(parent1, parent2)
     if (small random probability) then child $\leftarrow$ MUTATE(child)
     add child to population2
   population $\leftarrow$ population2
 until some individual is fit enough, or enough time has elapsed
 return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
 $n \leftarrow$ LENGTH(parent1)
 $c \leftarrow$ random number from 1 to $n$
 return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, $c + 1$, n))
Applications of GA

- Complex structured problems
  - Circuit layout, job-shop scheduling

- Evolving the architecture of deep neural networks

- Finding bugs of hardware

- Molecular structure optimization

- Image processing.

- Learning robots, etc.
Applications of GA

- Complex structured problems
  Circuit layout, job-shop scheduling
- Evolving the architecture of deep neural networks
- Finding bugs of hardware
- Molecular structure optimization
- Image processing.
- Learning robots, etc.