CS 540 Introduction to Artificial Intelligence

Search III: Advanced Search

Yudong Chen
University of Wisconsin-Madison

Nov 18, 2021
Announcements

• **Homeworks:**
  – HW8 due Tuesday.

• **Class roadmap:**
  – Today: Search III
  – Tuesday before and after Thanksgiving: Game (uses Search)
Outline

• Advanced Search & Hill-climbing
  – More difficult problems, basics, local optima, variations

• Simulated Annealing
  – Basic algorithm, temperature, tradeoffs

• Genetic Algorithms
  – Basics of evolution, fitness, natural selection
Search vs. Optimization

Before: wanted a **path** from start state to goal state

- Uninformed search, informed search

**New setting**: optimization

- States $s$ have values $f(s)$
- Want: $s$ with optimal value $f(s)$ (i.e., optimize over states)
- Challenging setting: **too many states** for previous search approaches, but maybe not a continuous function for SGD.
Examples: $n$ Queens

A classic puzzle:

- Place 8 queens on a 8 x 8 chessboard so that no two have the same row, column, or diagonal.
- Can generalize to $n \times n$ chessboard.

- What are states $s$? Values $f(s)$?
  - State: configuration of the board
  - $f(s)$: # of conflicting queens

\[
\begin{cases} 
0 & \text{if no conflict} \\
1 & \text{if there’s conflict.}
\end{cases}
\]
Examples: TSP

Famous graph theory problem.

• Get a graph \( G = (V,E) \). **Goal**: a path that visits each node exactly once and returns to the initial node (a tour).
  
  – State: a particular tour (i.e., ordered list of nodes)
  
  – \( f(s) \): total weight of the tour
    (e.g., total miles traveled)
Examples: Satisfiability

Boolean satisfiability (e.g., 3-SAT)

• Recall our logic lecture. Conjunctive normal form

\[(A \lor \neg B \lor C) \land (\neg A \lor C \lor D) \land (B \lor D \lor \neg E) \land (\neg C \lor \neg D \lor \neg E) \land (\neg A \lor \neg C \lor E)\]

– Goal: find if satisfactory assignment exists.
– State: assignment to variables
– \( f(s) \): # satisfied clauses

\[
\begin{align*}
R(x, a, d) & \land R(y, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(z, c, 0) \\
R(0, a, d) & \land R(0, b, d) & \land R(a, b, c) & \land R(c, d, f) & \land R(0, c, 0) \\
R(0, a, d) & \land R(0, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(1, c, 0) \\
R(0, a, d) & \land R(1, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(0, c, 0) \\
R(0, a, d) & \land R(1, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(1, c, 0) \\
R(1, a, d) & \land R(0, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(0, c, 0) \\
R(1, a, d) & \land R(0, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(1, c, 0) \\
R(1, a, d) & \land R(1, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(0, c, 0) \\
R(1, a, d) & \land R(1, b, d) & \land R(a, b, e) & \land R(c, d, f) & \land R(1, c, 0) \\
R(0, a, b) & \land R(b, y, c) & \land R(c, d, 0)
\end{align*}
\]
Hill Climbing

One approach to such optimization problems.

• Basic idea: move to a neighbor with a better $f(s)$

• Q: how do we define neighbor?
  – Not as obvious as our successors in search
  – Problem-specific
  – As we’ll see, needs a careful choice
Defining Neighbors: n Queens

In n Queens, a simple possibility:

- Look at the **most-conflicting column** (ties? right-most one)
- Move queen in that column vertically to a different location
Defining Neighbors: TSP

For TSP, can do something similar:

• Define neighbors by small changes
• Example: 2-change: A-E and B-F

A-B-C-D-E-F-G-H-A

flip

A-E-D-C-B-F-G-H-A
Defining Neighbors: SAT

For Boolean satisfiability,

- Define neighbors by flipping one assignment of one variable

Starting state: TFTTT

\[
\begin{align*}
(A=F, B=F, C=T, D=T, E=T) & \quad A \lor \neg B \lor C \\
(A=T, B=T, C=T, D=T, E=T) & \quad \neg A \lor C \lor D \\
(A=T, B=F, C=F, D=T, E=T) & \quad B \lor D \lor \neg E \\
(A=T, B=F, C=T, D=F, E=T) & \quad \neg C \lor \neg D \lor \neg E \\
(A=T, B=F, C=T, D=T, E=F) & \quad \neg A \lor \neg C \lor E
\end{align*}
\]
Hill Climbing Neighbors

Q: What’s a neighbor?

• Vague definition. For a given problem structure, neighbors are states that can be produced by a small change

• Tradeoff!
  – Too small? Will get struck.
  – Too big? Not very efficient

• Q: how to pick a neighbor? Greedy

• Q: terminate? When no neighbor has bigger value
Hill Climbing Algorithm

Pseudocode:

1. Pick initial state $s$
2. Pick $t$ in $\text{neighbors}(s)$ with the largest $f(t)$
3. if $f(t) \leq f(s)$ THEN stop, return $s$
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**
Hill Climbing: Local Optima

Q: Why is it called hill climbing?

L: What’s actually going on.

R: What we get to see.

Global optimum, where we want to be
Hill Climbing: Local Optima

Note the **local optima**. How do we handle them?
Escaping Local Optima

Simple idea 1: random restarts
• Stuck: pick a random new starting point, re-run.
• Do $k$ times, return best of the $k$.

Simple idea 2: reduce greed
• “Stochastic” hill climbing: randomly select between neighbors
• Probability proportional to the value of neighbors
Hill Climbing: Variations

Q: neighborhood too large?
• Generate a few random neighbors, one at a time. Take the better one.

Q: relax requirement to always go up?
• Often useful for harder problems
• 3SAT algorithm: Walk-SAT
Q 1.1: Hill climbing and SGD are related by
(i) Both head towards local optima
(ii) Both require computing a gradient
(iii) Both will find the global optimum for a convex problem (when minimizing)

• A. (i)
• B. (i), (ii)
• C. (i), (iii)
• D. All of the above
Q 1.1: Hill climbing and SGD are related by

(i) Both head towards local optima
(ii) Both require computing a gradient
(iii) Both will find the global optimum for a convex problem (when minimizing)

- A. (i)
- B. (i), (ii)
- C. (i), (iii)
- D. All of the above
Q 1.1: Hill climbing and SGD are related by

(i) Both head towards local optima
(ii) Both require computing a gradient
(iii) Both will find the global optimum for a convex problem (when minimizing)

- A. (i) (No: (iii) also true since convexity->local optima are global)
- B. (i), (ii) (No: (ii) is false. Hill-climbing looks at neighbors only.)
- C. (i), (iii)
- D. All of the above (No: (ii) false, as above.)
Simulated Annealing

A more sophisticated optimization approach.

- **Idea**: move quickly at first, then slow down
Simulated Annealing

A more sophisticated optimization approach.

- **Idea:** move quickly at first, then slow down
- **Pseudocode:**

  Pick initial state \( x \)
  
  For \( k = 0 \) through \( k_{\text{max}} \):
  
  Reduce temperature \( T \)
  
  Pick a random neighbour, \( y \leftarrow \text{neighbor}(x) \)
  
  If \( f(y) \geq f(x) \), then \( x \leftarrow y \)
  
  Else, with prob. \( P(f(x), f(y), T) \) then \( x \leftarrow y \)

**Output:** the final state \( x \)
Simulated Annealing: Picking Probability

How do we pick probability $P$?

- Decrease with gap $|f(x) - f(y)|$
- Decrease with time $k$

Pick initial state $x$
For $k = 0$ through $k_{\text{max}}$:
- Reduce temperature $T$
- Pick a random neighbour, $y \leftarrow \text{neighbor}(x)$
- If $f(y) \geq f(x)$, then $x \leftarrow y$
- Else, with prob. $P(f(x), f(y), T)$ then $x \leftarrow y$

**Output**: the final state $x$
Simulated Annealing: Picking Probability

How do we pick probability $P$?

- Decrease with gap $|f(x) - f(y)|$
- Decrease with time $k$
- Temperature $T$ cools over time
  - High temperature, accept any $y$
  - Low temperature, behaves like hill-climbing
  - Still, $|f(x) - f(y)|$ plays a role: if big, replacement probability low.

\[ P(x, y, T) = \exp \left( - \frac{|f(x) - f(y)|}{T} \right) \]
Simulated Annealing: Visualization
Simulated Annealing: Picking Parameters

• Have to balance the various parts., e.g., cooling schedule.
  – Too fast: becomes hill climbing, stuck in local optima
  – Too slow: takes too long.

• Combines with variations (e.g., with random restarts)
  – Probably should try hill-climbing first though.

• Inspired by cooling of metals
  – We’ll see one more alg. inspired by nature
Break & Quiz

Q 2.1: Which of the following is likely to give the best cooling schedule for simulated annealing?

k = iteration count

A. $T_{k+1} = T_k \times 1.25$
B. $T_{k+1} = T_k$
C. $T_{k+1} = T_k \times 0.8$
D. $T_{k+1} = T_k \times 0.0001$
Break & Quiz

Q 2.1: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $\text{Temp}_{k+1} = \text{Temp}_k \times 1.25$
B. $\text{Temp}_{k+1} = \text{Temp}_k$
C. $\text{Temp}_{k+1} = \text{Temp}_k \times 0.8$
D. $\text{Temp}_{k+1} = \text{Temp}_k \times 0.0001$
Q 2.1: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $\text{Temp}_{k+1} = \text{Temp}_k \times 1.25$ (No, temperate is increasing)

B. $\text{Temp}_{k+1} = \text{Temp}_k$ (No, temperature is constant)

C. $\text{Temp}_{k+1} = \text{Temp}_k \times 0.8$

D. $\text{Temp}_{k+1} = \text{Temp}_k \times 0.0001$ (Cools too fast---basically hill climbing)
Q 2.2: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges

ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power

iii. Finding the fastest way through a maze

• A. (i)
• B. (ii)
• C. (i) and (ii)
• D. (ii) and (iii)
Q 2.2: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

• A. (i)
• B. (ii)
• C. (i) and (ii)
• D. (ii) and (iii)
Q 2.2: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges

ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power

iii. Finding the fastest way through a maze

• A. (i) (Too many states for A*, don’t care about path)
• B. (ii) (Similar to above)
• C. (i) and (ii)
• D. (ii) and (iii) ( (iii) is good for A*: few successors, want path)
Genetic Algorithms

Another optimization approach based on nature

- Survival of the fittest!
Evolution Review

Encode genetic information in DNA (four bases)

• A/C/T/G: nucleobases acting as symbols

• Two types of changes
  – Crossover: exchange between parents’ codes
  – Mutation: rarer random process
    • Happens at individual level
Natural Selection

Competition for resources
• Organisms better fit $\rightarrow$ better probability of reproducing
• Repeated process: fit become larger proportion of population

Goal: use these principles for optimization
– New terminology: state s ‘individual’
– Value $f(s)$ is now the ‘fitness’
Genetic Algorithms Setup I

Keep around a fixed number of states/individuals
• A bit like beam search
• Call this the **population**

For our n Queens game example, an individual:

```
(3 2 7 5 2 4 1 1)
```
Genetic Algorithms Setup II

Goal of genetic algorithms: optimize using principles inspired by mechanism for evolution

• E.g., analogous to **natural selection, cross-over, and mutation**

\[
\frac{8 \times 7}{2} = 28
\]

\[f(s) = \text{# of non-attacking pairs}\]

\[\text{prob. reproduction } \propto \text{ fitness}\]

→ Next generation
Genetic Algorithms Pseudocode

Just one variant:

1. Let $s_1, ..., s_N$ be the current population.
2. Let $p_i = f(s_i) / \sum_j f(s_j)$ be the reproduction probability.
3. for $k = 1; k<N; k+=2$
   • parent1 = randomly pick according to $p$
   • parent2 = randomly pick another
   • randomly select a crossover point, swap strings of parents 1, 2 to generate children $t[k], t[k+1]$
4. for $k = 1; k<=N; k++$
   • Randomly mutate each position in $t[k]$ with a small probability (mutation rate).
5. The new generation replaces the old: $\{ s \} \leftarrow \{ t \}$. Repeat.
Reproduction probability: \( p_i = \frac{f(s_i)}{\sum_j f(s_j)} \)

- **Example**: \( \sum_j f(s_j) = 5 + 20 + 11 + 8 + 6 = 50 \)
- \( p_1 = \frac{5}{50} = 10\% \)

<table>
<thead>
<tr>
<th>Individual</th>
<th>Fitness</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>10%</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>40%</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>22%</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>16%</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>12%</td>
</tr>
</tbody>
</table>
Example: Scheduling Courses

Let’s run through an example:

• 5 courses: A, B, C, D, E
• 3 time slots: Mon/Wed, Tue/Thu, Fri/Sat
• Students wish to enroll in three courses
• Goal: maximize student enrollment

<table>
<thead>
<tr>
<th>Courses</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>2</td>
</tr>
<tr>
<td>A B D</td>
<td>7</td>
</tr>
<tr>
<td>A D E</td>
<td>3</td>
</tr>
<tr>
<td>B C D</td>
<td>4</td>
</tr>
<tr>
<td>B D E</td>
<td>10</td>
</tr>
<tr>
<td>C D E</td>
<td>5</td>
</tr>
</tbody>
</table>
Example: Scheduling Courses

Let’s run through an example:

- State: course assignment to time slot

<table>
<thead>
<tr>
<th>M</th>
<th>M</th>
<th>F</th>
<th>T</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>

• Here:
  - Courses A, B, E scheduled Mon/Wed
  - Course D scheduled Tue/Thu
  - Course C scheduled Fri/Sat

<table>
<thead>
<tr>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
</tr>
<tr>
<td>A B D</td>
</tr>
<tr>
<td>A D E</td>
</tr>
<tr>
<td>B C D</td>
</tr>
<tr>
<td>B D E</td>
</tr>
<tr>
<td>C D E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

= MMFTM
Example: Scheduling Courses

Value of a state? Say MMFTM

<table>
<thead>
<tr>
<th>Courses</th>
<th>Students</th>
<th>Can enroll?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>A B D</td>
<td>7</td>
<td>No</td>
</tr>
<tr>
<td>A D E</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>B C D</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>B D E</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>C D E</td>
<td>5</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Here 4+5=9 students can enroll in desired courses
Example: Scheduling Courses

First step:

• Randomly initialize and evaluate states

- $S_1$: $MMFTM = 9$, $S_1'$: 8, $MMFTM = 26\% = \frac{9+4+19+3}{4}$
- $S_2$: $TTFMM = 4$, $S_2'$: 3, $TTFMM = 11\%$
- $S_3$: $FMTTF = 19$, $S_3'$: 11, $FMTTF = 54\%$
- $S_4$: $MTTF = 3$, $S_4'$: 2, $MTTF = 9\%$

• Calculate reproduction probabilities

<table>
<thead>
<tr>
<th>Courses</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>2</td>
</tr>
<tr>
<td>A B D</td>
<td>7</td>
</tr>
<tr>
<td>A D E</td>
<td>3</td>
</tr>
<tr>
<td>B C D</td>
<td>4</td>
</tr>
<tr>
<td>B D E</td>
<td>10</td>
</tr>
<tr>
<td>C D E</td>
<td>5</td>
</tr>
</tbody>
</table>
Example: Scheduling Courses

Next steps:
• Select parents using reproduction probabilities
• Perform crossover
• Randomly mutate new children
Example: Scheduling Courses

Continue:

• Now, get our function values for updated population
• Calculate reproduction probabilities

<table>
<thead>
<tr>
<th>Courses</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>2</td>
</tr>
<tr>
<td>A B D</td>
<td>7</td>
</tr>
<tr>
<td>A D E</td>
<td>3</td>
</tr>
<tr>
<td>B C D</td>
<td>4</td>
</tr>
<tr>
<td>B D E</td>
<td>10</td>
</tr>
<tr>
<td>C D E</td>
<td>5</td>
</tr>
</tbody>
</table>

FMFTTT = 11  FMFTTT = 39%
MMTTTF = 13  MMTTTF = 46%
MMTFFF = 4   MMTFFF = 14%
FTTTTF = 0   FTTTTF = 0%
Variations & Concerns

Many *possibilities*:

- Parents survive to next generation
- Ranking instead of exact value of $f(s)$ for reproduction probabilities

Some *challenges*

- State encoding
- Lack of diversity: converge too soon
- Must pick a lot of parameters
Summary

• Challenging optimization problems
  – First, try hill climbing. Simplest solution

• Simulated annealing
  – More sophisticated approach; helps with local optima

• Genetic algorithms
  – Biology-inspired optimization routine
Acknowledgements: Adapted from materials by Fred Sala, Jerry Zhu + Tony Gitter (University of Wisconsin), Andrew Moore