Week 2: Informed Search, A*, Heuristics

27.09.2016

Tuesday 14:00 – 17:00, BM A3

Emre Ugur, BM 33
emre.ugur@boun.edu.tr
http://www.cmpe.boun.edu.tr/~emre/courses/CMPE540/
cmpe540@listeci.cmpe.boun.edu.tr
Sources & credits

- Course notes:
  - Russell and Norvig:
    - http://aima.cs.berkeley.edu/
  - Levent Akin, Pinar Yolum and Albert Ali Salah
    - http://www.cmpe.boun.edu.tr/~akin/

- Projects and content:
  - UC Berkeley CS188 Intro to AI
    - https://inst.eecs.berkeley.edu/~cs188/fa11/assignments.html
## Syllabus (subject to change)

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Chapters</th>
<th>Date</th>
<th>Topic</th>
<th>Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep 20</td>
<td>Agents &amp; Uninformed search</td>
<td>Ch 2, 3.1-4</td>
<td>Nov 1</td>
<td>Probability</td>
<td>Ch. 13.1-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bayes nets: Syntax and semantics</td>
<td>Ch. 14.1-3</td>
</tr>
<tr>
<td>Sep 27</td>
<td>A* search, heuristics</td>
<td>Ch. 3.5-6</td>
<td>Nov 8</td>
<td>Bayes nets: Exact inference</td>
<td>Ch. 14.3</td>
</tr>
<tr>
<td></td>
<td>Local search; search-based agents</td>
<td>Ch. 4</td>
<td></td>
<td>Bayes nets: Approximate inference</td>
<td>Ch. 14.4</td>
</tr>
<tr>
<td>Oct 1</td>
<td>Game playing</td>
<td>Ch. 5.1-5</td>
<td>Nov 15</td>
<td>Midterm 2</td>
<td></td>
</tr>
<tr>
<td>10:00 –</td>
<td>Constraint satisfaction problems</td>
<td>Ch. 6.1, 6.3-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oct 4</td>
<td>No class</td>
<td></td>
<td>Nov 22</td>
<td>Markov Models, Hidden Markov Models</td>
<td>Ch. 15.1-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Applications of HMMs</td>
<td>Ch. 15.5</td>
</tr>
<tr>
<td>Oct 11</td>
<td></td>
<td></td>
<td>Nov 29</td>
<td>Decision theory</td>
<td>Ch. 16.1-3</td>
</tr>
<tr>
<td></td>
<td>Midterm 1</td>
<td></td>
<td></td>
<td>Markov decision processes</td>
<td>Ch. 16.5-6</td>
</tr>
<tr>
<td>Oct 18</td>
<td>Propositional logic: semantics and inference</td>
<td>Ch. 7.1-4, 7.6.1</td>
<td>Dec 6</td>
<td>Machine learning: Classification and regression</td>
<td>Ch. 18.1-4</td>
</tr>
<tr>
<td></td>
<td>Propositional planning and logical agents</td>
<td>Ch. 7.7</td>
<td></td>
<td></td>
<td>18.6</td>
</tr>
<tr>
<td>Oct 25</td>
<td>First-order logic</td>
<td>Ch. 8.1-3, 9.1</td>
<td>Dec 13</td>
<td>Advanced Topics: Vision and robotics</td>
<td>Ch. 24,25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is AI?

Views of AI fall into four categories (Humanly vs. Rationally)

- Acting rationally, Thinking humanly, Acting humanly, Thinking rationally,

```
"The exciting new effort to make computers think ... machines with minds, in the full and literal sense" (Haugeland, 1985)

"The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)

"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)

"The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)
```

```
"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985)

"The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)

"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990)

"AI ... is concerned with intelligent behavior in artifacts." (Nilsson, 1998)
```
Overview

- PEAS (Performance, Environment, Actuators, Sensors)
- Environment types
- Agent functions and properties
- Agent types
Agents include humans, robots, softbots, thermostats, etc.
The **agent function** maps from $P^*$ to $A$

$f: P^* \rightarrow A$

The **agent program** runs on the physical **architecture** to produce $f$
PEAS for Part-picking robot

- Performance measure:
  - Percentage of parts in correct bins
- Environment:
  - Conveyor belt with parts, bins
- Actuators:
  - Jointed arm and hand
- Sensors:
  - Camera, joint angle sensors
# Environment types

<table>
<thead>
<tr>
<th></th>
<th>Solitaire</th>
<th>Backgammon</th>
<th>Internet Shopping</th>
<th>Taxi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observable?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Deterministic?</td>
<td>Yes</td>
<td>No</td>
<td>Partly</td>
<td>No</td>
</tr>
<tr>
<td>Episodic?</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Static?</td>
<td>Yes</td>
<td>Semi</td>
<td>Semi</td>
<td>No</td>
</tr>
<tr>
<td>Discrete?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Single-agent?</td>
<td>Yes</td>
<td>No</td>
<td>Yes (except auctions)</td>
<td>No</td>
</tr>
</tbody>
</table>

The environment type largely determines the agent design. The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent.
Agent types

- Four basic agent types in order of increasing generality:
  - simple reflex agents
  - reflex agents with state
  - goal-based agents
  - utility-based agents
- All these can be turned into learning agents
Outline

◊ Problem-solving agents
◊ Problem types
◊ Problem formulation
◊ Example problems
◊ Basic search algorithms
A **problem** is defined by four items:

---

**initial state** e.g., “at Arad”

**successor function** $S(x) =$ set of action–state pairs
  e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind \rangle, \ldots \}$

**goal test**, can be
  - **explicit**, e.g., $x =$ “at Bucharest”
  - **implicit**, e.g., $NoDirt(x)$

**path cost** (additive)
  e.g., sum of distances, number of actions executed, etc.
  $c(x, a, y)$ is the **step cost**, assumed to be $\geq 0$

---

A **solution** is a sequence of actions leading from the initial state to a goal state
Example: vacuum world state space graph

**states??**: integer dirt and robot locations (ignore dirt amounts etc.)

**actions??**: *Left, Right, Suck, NoOp*

**goal test??**: no dirt

**path cost??**: 1 per action (0 for *NoOp*)
Tree search example

[Diagram of a tree search example with cities such as Arad, Sibiu, Timisoara, Zerind, etc.]
Implementation: states vs. nodes

A state is a (representation of) a physical configuration.
A node is a data structure constituting part of a search tree includes parent, children, depth, path cost $g(x)$
States do not have parents, children, depth, or path cost!

The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.
function Tree-Search(problem, fringe) returns a solution, or failure

fringe ← INSERT(Make-Node(Initial-State[problem]), fringe)

loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-Test(problem, State(node)) then return node
    fringe ← INSERTALL(Expand(node, problem), fringe)

function Expand(node, problem) returns a set of nodes

successors ← the empty set

for each action, result in SUCCESSOR-Fn(problem, State[node]) do
    s ← a new Node
    Parent-Node[s] ← node; Action[s] ← action; State[s] ← result
    Path-Cost[s] ← Path-Cost[node] + Step-Cost(node, action, s)
    Depth[s] ← Depth[node] + 1
    add s to successors

return successors
Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

Depth-limited search

Iterative deepening search
## Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes*</td>
<td>Yes*</td>
<td>No</td>
<td>Yes, if $l \geq d$</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$b^{d+1}$</td>
<td>$b^{C*/\epsilon}$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^{d+1}$</td>
<td>$b^{C*/\epsilon}$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes*</td>
</tr>
</tbody>
</table>
Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!

![Diagram showing repeated states]

A diagram on the right illustrates the concept with a branching structure, highlighting how repeated states can lead to exponential growth in the problem's complexity.
Graph search

function Graph-Search(problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← Insert(Make-Node(Initial-State[problem]), fringe)

loop do
    if fringe is empty then return failure
    node ← Remove-Front(fringe)
    if Goal-Test(problem, State[node]) then return node
    if State[node] is not in closed then
        add State[node] to closed
        fringe ← InsertAll(Expand(node, problem), fringe)
    end
end
Quiz

**Breath-first search**: First in first out (FIFO) queue.

**Uniform-cost search**: queue ordered by path cost, lowest first

- Assign costs and goals so that solution of breath-first search is not optimal
- Assign costs and goals so that solution of uniform search is not optimal
Today

- Informed Search
- A*, Heuristics
Overview

- Informed search strategies
- Heuristic functions
- Local search and optimization
- Local search in continuous spaces
- Searching with nondeterministic actions
- Searching with partial observations
- Online search agents and unknown environments
A strategy is defined by picking the order of node expansion
Best-first search

Idea: use an evaluation function for each node
   – estimate of “desirability”

⇒ Expand most desirable unexpanded node

Implementation:
fringe is a queue sorted in decreasing order of desirability

Special cases: greedy search A* search

f(n) : evaluation function
lowest evaluation is selected for expansion
maintain fringe in the ascending order of f-values.
Romania with step costs in km

Straight-line distance to Bucharest:
- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobroța: 242
- Eforie: 161
- Făgăraș: 178
- Giurgiu: 77
- Hîrsoa: 151
- Iași: 226
- Lugoj: 244
- Mehedia: 241
- Neața: 234
- Oradea: 380
- Pitești: 98
- Rimnicu Vâlcea: 193
- Sibiu: 253
- Timisoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
Greedy search

Evaluation function $h(n)$ (heuristic)

= estimate of cost from $n$ to the closest goal

^ of the cheapest path

E.g., $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest

Greedy search expands the node that 

appears

to be closest to goal

if $f(n) = h(n)$ --> greedy search
Greedy search example

<table>
<thead>
<tr>
<th>City</th>
<th>h_{SLD}</th>
<th>City</th>
<th>h_{SLD}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

Values of $h_{SLD}$—straight-line distances to Bucharest.
Greedy search example

Values of $h_{SLD}$—straight-line distances to Bucharest.
Greedy search example

Values of $h_{SLD}$—straight-line distances to Bucharest.
Greedy search example

Optimal?
Greedy...
false starts? from Iasi to Fagaras
Properties of greedy search

Complete??
Properties of greedy search

**Complete??** No–can get stuck in loops, e.g., with Oradea as goal,
   Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

**Time??**
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

**Time??** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space??**
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,
\[ l_{asi} \rightarrow N_{eamt} \rightarrow l_{asi} \rightarrow N_{eamt} \rightarrow \]
Complete in finite space with repeated-state checking

**Time??** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space??** $O(b^m)$—keeps all nodes in memory

**Optimal??**
Properties of greedy search

**Complete??** No—can get stuck in loops, e.g.,

{lasi} → {Neamt} → {lasi} → {Neamt} →

Complete in finite space with repeated-state checking

**Time??** $O(b^m)$, but a good heuristic can give dramatic improvement

**Space??** $O(b^m)$—keeps all nodes in memory

**Optimal??** No
A* search

Idea: avoid expanding paths that are already expensive

Evaluation function  \( f(n) = g(n) + h(n) \)

\( g(n) \) = cost so far to reach \( n \)  -- estimated cost from start to \( n \)
\( h(n) \) = estimated cost to goal from \( n \)
\( f(n) \) = estimated total cost of path through \( n \) to goal

A* search uses an admissible heuristic
i.e., \( h(n) \leq h^*(n) \) where \( h^*(n) \) is the true cost from \( n \).
(Also require \( h(n) \geq 0 \), so \( h(G) = 0 \) for any goal \( G \).)

E.g., \( h_{SLD}(n) \) never overestimates the actual road distance

Theorem: A* search is optimal  \( g(n) \) is the actual cost:
\( \rightarrow f(n) \) never overestimates the cost
A* search example

Values of $h_{SLD}$—straight-line distances to Bucharest.
A* search example

Values of $h_{SLD}$—straight-line distances to Bucharest.

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Rimmnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Vaslui</td>
<td>80</td>
</tr>
<tr>
<td>Urziceni</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
A* search example

Values of $h_{SLD}$—straight-line distances to Bucharest.
A* search example

- Arad
  - Fagaras: 646=280+366
  - Oradea: 415=239+176
  - Rimnicu Vilcea: 671=291+380

- Sibiu

- Timisoara: 447=118+329

- Zerind: 449=75+374
A* search example

- Arad
  - Sibiu
    - Bucharest
      - Sibiu
      - Craiova
    - Oradea
      - Fagaras
      - Sibiu
  - Timisoara
    - 447 = 118 + 329
  - Zerind
    - 449 = 75 + 374

Costs:
- Arad to Sibiu: 646 = 280 + 366
- Sibiu to Bucharest: 591 = 338 + 253
- Sibiu to Craiova: 526 = 366 + 160
- Sibiu to Pitești: 553 = 300 + 253

A* search example

Optimal? Yes if heuristic is
1- admissible ... never overestimates: \( h(n) \leq h^*(n) \)
2- consistency... \( h(n) < h(n') + c(n,a,n') \)
Every consistent heuristic is admissible.
* Admissible but inconsistent heuristics require bookkeeping to ensure optimality.
Optimality of A* (standard proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[
f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0 \\
> g(G_1) \quad \text{since } G_2 \text{ is suboptimal} \\
\geq f(n) \quad \text{since } h \text{ is admissible}
\]

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.
Proof of lemma: Consistency

A heuristic is **consistent** if

\[ h(n) \leq c(n, a, n') + h(n') \]

If \( h \) is consistent, we have

\[
\begin{align*}
    f(n') &= g(n') + h(n') \\
    &= g(n) + c(n, a, n') + h(n') \\
    &\geq g(n) + h(n) \\
    &= f(n)
\end{align*}
\]

I.e., \( f(n) \) is nondecreasing along any path.

if \( h(n) \) is consistent, then the values of \( f(n) \) along any path are nondecreasing.
Consistency (Monotonicity): \( h(n) \leq c(n,a,n') + h(n') \)
--- Suppose \( n' \) is successor of \( n \)
\[ f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n) \]

**Optimality of A* (more useful)**

**Lemma:** A* expands nodes in order of increasing \( f \) value

Gradually adds “\( f \)-contours” of nodes (cf. breadth-first adds layers)
Contour \( i \) has all nodes with \( f = f_i \), where \( f_i < f_{i+1} \)

A* expands fringe nodes with lowest \( f \)-cost.
Heuristic = 1: shape, heuristic = direct-path .. ?

Optimal: all other goal nodes will have higher \( f \)-costs, thus \( g \)-costs (as \( h(n)=0 \)).
Properties of A*

Complete??
Properties of A*

**Complete??** Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time??**
Properties of A*

**Complete??** Yes, unless there are infinitely many nodes with $f \leq f(G')$

**Time??** Exponential in [relative error in $h \times$ length of soln.]

**Space??**
Properties of $A^*$

**Complete**
Yes, unless there are infinitely many nodes with $f \leq f(G)$

**Time**
Exponential in $[\text{relative error in } h \times \text{length of soln.}]$

**Space**
Keeps all nodes in memory

**Optimal**
Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G')$

Time?? Exponential in $[\text{relative error in } h \times \text{length of soln.}]$

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand $f_{i+1}$ until $f_i$ is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$
Proof of lemma: Consistency

A heuristic is **consistent** if

\[ h(n) \leq c(n, a, n') + h(n') \]

If \( h \) is consistent, we have

\[
\begin{align*}
    f(n') &= g(n') + h(n') \\
          &= g(n) + c(n, a, n') + h(n') \\
          &\geq g(n) + h(n) \\
          &= f(n)
\end{align*}
\]

I.e., \( f(n) \) is nondecreasing along any path.

if \( h(n) \) is consistent, then the values of \( f(n) \) along any path are nondecreasing.
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

\[
\begin{array}{ccc}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & \\
\end{array}
\]

Start State

Goal State

\[ h_1(S) = ?? \]
\[ h_2(S) = ?? \]
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

\[ h_1(S) = \text{?? 6} \]
\[ h_2(S) = \text{?? 4+0+3+3+1+0+2+1 = 14} \]
Dominance

If \( h_2(n) \geq h_1(n) \) for all \( n \) (both admissible)
then \( h_2 \) dominates \( h_1 \) and is better for search

Typical search costs:

\[
\begin{align*}
  d = 14 & \quad \text{IDS} = 3,473,941 \text{ nodes} \\
  & \quad A^*(h_1) = 539 \text{ nodes} \\
  & \quad A^*(h_2) = 113 \text{ nodes} \\
  d = 24 & \quad \text{IDS} \approx 54,000,000,000 \text{ nodes} \\
  & \quad A^*(h_1) = 39,135 \text{ nodes} \\
  & \quad A^*(h_2) = 1,641 \text{ nodes}
\end{align*}
\]

Given any admissible heuristics \( h_a, h_b, \)

\( h(n) = \max(h_a(n), h_b(n)) \)

is also admissible and dominates \( h_a, h_b \)

QUESTION: HOW TO INVENT ADMISSIBLE HEURISTICS?
Relaxed problems

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.

If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

eg. traveling salesman problem
Relaxed problems contd.

Well-known example: **travelling salesperson problem** (TSP)
Find the shortest tour visiting all cities exactly once

Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

8-puzzle: A tile can move from A to B if
A is adjacent to B
and B is blank
Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once

Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

eg. 8-puzzle: other admissible heuristics?
subproblem? how to combine?
Pattern Databases

- Store exact solution of every possible subproblem instance
- Lookup table
- Do this for different subproblems – take max()
- How about sum?
  - Share common moves
  - Disjoint pattern databases:
    - Random 15-puzzles in a few seconds
    - Reduced by a factor of 10,000 compared to Manhattan distance
Summary

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest $h$
  – incomplete and not always optimal

A* search expands lowest $g + h$
  – complete and optimal
  – also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems
Previous search algorithms explored state space systematically. Kept paths in the memory. Path to the goal was part of solution.

## Iterative improvement algorithms

In many optimization problems, path is irrelevant; the goal state itself is the solution.

Then state space = set of “complete” configurations;
find **optimal** configuration, e.g., TSP other examples?
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use **iterative improvement** algorithms;
keep a single “current” state, try to improve it -- local search in the neighbourhood

1 Constant space, suitable for online as well as offline search
2 Often find solutions in large or infinite (continuous) state spaces
Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

Variants of this approach get within 1% of optimal very quickly with thousands of cities
Example: $n$-queens

Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.

Move a queen to reduce number of conflicts.

Almost always solves $n$-queens problems almost instantaneously for very large $n$, e.g., $n = 1\text{ million}$.
Hill-climbing (or gradient ascent/descent)

“Like climbing Everest in thick fog with amnesia”

function Hill-Climbing(problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
              neighbor, a node

    current ← Make-Node(Initial-State[problem])
loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
end
Hill-climbing contd.

Useful to consider state space landscape

Random-restart hill climbing overcomes local maxima—trivially complete
Random sideways moves 😊 escape from shoulders 😔 loop on flat maxima
Hill-climbing contd.

- **h** = number of pairs attacking (direct or indirect)
- Steepest-ascent solves only 14%. If limited sideways moves: 94%
- Stochastic hill-climbing: select with probability steepness
- Random-restart hill-climbing: start-over. Expected number of restarts for 8-queens?
  --> good for few local minima
Hill-climbing: incomplete: might always stuck in local optima (even stochastic)  
Random-walk: complete: inefficient

Simulated annealing

Idea: escape local maxima by allowing some “bad” moves  
but gradually decrease their size and frequency

function Simulated-Annealing(problem, schedule) returns a solution state

inputs: problem, a problem  
schedule, a mapping from time to “temperature”

local variables: current, a node  
next, a node  
T, a “temperature” controlling prob. of downward steps

current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] − VALUE[current]
    if ΔE > 0 then current ← next .. accept if it improves
    else current ← next only with probability $e^{ΔE/T}$ .. accept with small prob.  
    higher chance in the beginning

shake hard enough to bounce the ball at the local minimum  
but not hard enough to dislodge it from global minimum.
Local beam search

Idea: keep $k$ states instead of 1; choose top $k$ of all their successors

Not the same as $k$ searches run in parallel!
Searches that find good states recruit other searches to join them

Problem: quite often, all $k$ states end up on same local hill

Idea: choose $k$ successors randomly, biased towards good ones

Observe the close analogy to natural selection!

stochastic beam search
how to set probability?
Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime.”

- Salvatore Mangano

*Computer Design*, May 1995
The Genetic Algorithm

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970’s)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems
The Genetic Algorithm (cont.)

- Provide efficient, effective techniques for optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles
Components of a GA

A problem to solve, and ...

- Encoding technique  (*gene, chromosome*)
- Initialization procedure  (*creation*)
- Evaluation function  (*environment*)
- Selection of parents  (*reproduction*)
- Genetic operators  (*mutation, recombination*)
- Parameter settings  (*practice and art*)
Simple Genetic Algorithm

{
  initialize population;
  evaluate population;
  while TerminationCriteriaNotSatisfied
  {
    select parents for reproduction;
    perform recombination and mutation;
    evaluate population;
  }
}

Wendy Williams
Metaheuristic Algorithms
The GA Cycle of Reproduction

- **Reproduction**
  - Parents
  - Population
  - Discard

- **Modification**
  - Children
  - Modified children

- **Evaluation**
  - Evaluated children
Population

Chromosomes could be:

- Bit strings  
  - (0101 ... 1100)
- Real numbers  
  - (43.2 -33.1 ... 0.0 89.2)
- Permutations of element  
  - (E11 E3 E7 ... E1 E15)
- Lists of rules  
  - (R1 R2 R3 ... R22 R23)
- Program elements  
  - (genetic programming)
- ... any data structure ...

Wendy Williams
Metaheuristic Algorithms

Geneic Algorithms: A Tutorial
Reproduction

Parents are selected at random with selection chances biased in relation to chromosome evaluations.
Chromosome Modification

- Modifications are stochastically triggered
- Operator types are:
  - Mutation
  - Crossover (recombination)
Mutation: Local Modification

Before: \( (1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0) \)
After: \( (0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0) \)

Before: \( (1.38 \ -69.4 \ 326.44 \ 0.1) \)
After: \( (1.38 \ -67.5 \ 326.44 \ 0.1) \)

- Causes movement in the search space (local or global)
- Restores lost information to the population
Crossover: Recombination

\[
\begin{array}{cccccccc}
\text{P1} & (0 & 1 & 1 & 0 & 1 & 0 & 0) & \rightarrow & (0 & 1 & 0 & 0 & 1 & 0 & 0 & 0) & \text{C1} \\
\text{P2} & (1 & 1 & 0 & 1 & 1 & 0 & 1 & 0) & \rightarrow & (1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0) & \text{C2}
\end{array}
\]

Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)
Evaluation

- The evaluator decodes a chromosome and assigns it a fitness measure
- The evaluator is the only link between a classical GA and the problem it is solving
Deletion

- Generational GA:
  entire populations replaced with each iteration

- Steady-state GA:
  a few members replaced each generation
An Abstract Example

Distribution of Individuals in Generation 0

Distribution of Individuals in Generation N
A Simple Example

“The Gene is by far the most sophisticated program around.”

- Bill Gates, Business Week, June 27, 1994
A Simple Example

The Traveling Salesman Problem:

Find a tour of a given set of cities so that

♦ each city is visited only once
♦ the total distance traveled is minimized
Representation

Representation is an ordered list of city numbers known as an order-based GA.

1) London  
2) Venice  
3) Dunedin  
4) Singapore  
5) Beijing  
6) Phoenix  
7) Tokyo  
8) Victoria

CityList1: (3 5 7 2 1 6 4 8)  
CityList2: (2 5 7 6 8 1 3 4)  

Wendy Williams  
Metaheuristic Algorithms  
18  
Genetic Algorithms: A Tutorial
Crossover

Crossover combines inversion and recombination:

\[ \text{Parent 1: } (3 \ 5 \ 7 \ 2 \ 1 \ 6 \ 4 \ 8) \]
\[ \text{Parent 2: } (2 \ 5 \ 7 \ 6 \ 8 \ 1 \ 3 \ 4) \]
\[ \text{Child: } (5 \ 8 \ 7 \ 2 \ 1 \ 6 \ 3 \ 4) \]

This operator is called the \textit{Order1} crossover.
Mutation

Mutation involves reordering of the list:

Before: (5 8 7 2 1 6 3 4)

After: (5 8 6 2 1 7 3 4)
TSP Example: 30 Cities
Solution \( s_i \) (Distance = 941)
Solution $j$ (Distance = 800)

TSP30 (Performance = 800)
Solution \( k \)(Distance = 652)
Best Solution (Distance = 420)

TSP30 Solution (Performance = 420)
Overview of Performance

TSP30 - Overview of Performance

Distance

Generations (1000)

Best
Worst
Average
Issues for GA Practitioners

- Choosing basic implementation issues:
  - representation
  - population size, mutation rate, ...
  - selection, deletion policies
  - crossover, mutation operators

- Termination Criteria

- Performance, scalability

- Solution is only as good as the evaluation function (often hardest part)
Benefits of Genetic Algorithms

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed
Benefits of Genetic Algorithms (cont.)

- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use
When to Use a GA

- Alternate solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements
# Some GA Application Types

<table>
<thead>
<tr>
<th>Domain</th>
<th>Application Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>gas pipeline, pole balancing, missile evasion, pursuit</td>
</tr>
<tr>
<td>Design</td>
<td>semiconductor layout, aircraft design, keyboard configuration,</td>
</tr>
<tr>
<td></td>
<td>communication networks</td>
</tr>
<tr>
<td>Scheduling</td>
<td>manufacturing, facility scheduling, resource allocation</td>
</tr>
<tr>
<td>Robotics</td>
<td>trajectory planning</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>designing neural networks, improving classification algorithms,</td>
</tr>
<tr>
<td></td>
<td>classifier systems</td>
</tr>
<tr>
<td>Signal Processing</td>
<td>filter design</td>
</tr>
<tr>
<td>Game Playing</td>
<td>poker, checkers, prisoner’s dilemma</td>
</tr>
<tr>
<td>Combinatorial</td>
<td>set covering, travelling salesman, routing, bin packing,</td>
</tr>
<tr>
<td>Optimization</td>
<td>graph colouring and partitioning</td>
</tr>
</tbody>
</table>
Conclusions

Question: ‘If GAs are so smart, why ain’t they rich?’

Answer: ‘Genetic algorithms are rich - rich in application across a large and growing number of disciplines.’

- David E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning
Genetic algorithms

= stochastic local beam search + generate successors from pairs of states

Fitness  Selection  Pairs  Cross–Over  Mutation
Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps iff substrings are meaningful components

\[
\begin{array}{c|c|c}
\text{State 1} & \text{State 2} & \text{State 3} \\
\hline
\text{Row 1} & \text{Row 2} & \text{Row 3} \\
\end{array}
\]

GAs $\neq$ evolution: e.g., real genes encode replication machinery!
Representation in GA

- How to represent the n-queens problem?
- Encode just the position of the queen in each column.
- You need $\log_2(n)$ bits per column and n columns in total. (you can leave out the last column too).
Representation in GA

- How to represent a Careless Secretary?
- Two piles of bills, mixed into a single pile and we just know what the total should be in each pile -> An NP-complete knapsack problem!
- Chromosome: 011010010110...
- Each bit one bill, and 0 means left pile, 1 means right pile.
- Sort the bills, so that crossover operates on a meaningful sequence!
Continuous state spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by \((x_1, y_2), (x_2, y_2), (x_3, y_3)\)
- objective function \(f(x_1, y_2, x_2, y_2, x_3, y_3) = \text{sum of squared distances from each city to nearest airport}\)

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers \(\pm \delta\) change in each coordinate

Gradient methods compute

\[
\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)
\]

to increase/reduce \(f\), e.g., by \(\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})\)

Sometimes can solve for \(\nabla f(\mathbf{x}) = 0\) exactly (e.g., with one city). 
Newton–Raphson (1664, 1690) iterates \(\mathbf{x} \leftarrow \mathbf{x} - H_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})\)
to solve \(\nabla f(\mathbf{x}) = 0\), where \(H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}\)