COMP 8620
Advanced Topics in AI

Lecturers:
Philip Kilby &
Jinbo Huang
Part 1: Search  
Lecturer: Dr Philip Kilby  
Philip.Kilby@nicta.com.au  
Weeks 1-7  
(Week 7 is assignment seminars)

Part 2: Probabilistic Reasoning with Bayesian networks  
Lecturer: Dr Jinbo Huang  
Jinbo.Huang@nicta.com.au  
Weeks 8-13
### Course Outline – Part 1

<table>
<thead>
<tr>
<th>1-2</th>
<th>Introduction</th>
<th>7-8</th>
<th>Constraint Programming (Guest lecturer: Jason Li)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Systematic Search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-4</td>
<td>Linear Programming /</td>
<td>9-10</td>
<td>Case Studies:</td>
</tr>
<tr>
<td></td>
<td>Integer Programming /</td>
<td></td>
<td>• TSP</td>
</tr>
<tr>
<td></td>
<td>Branch and Bound</td>
<td></td>
<td>• Planning (Jussi Rintanen)</td>
</tr>
<tr>
<td>5-6</td>
<td>Neighbourhood-based methods:</td>
<td>11</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td></td>
<td>• Simulated Annealing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Tabu Search</td>
<td>12</td>
<td>Qualitative Reasoning (Guest lecturer: Jason Li)</td>
</tr>
<tr>
<td></td>
<td>• Genetic Algorithms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-8</td>
<td>Constraint Programming (Guest lecturer: Jason Li)</td>
<td>13-14</td>
<td>Seminars (Guest Lecturers: You lot!)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assessment

- 40% Exam
- 60% Assignment

Search
- 1 x Assignment (next week) 15%
- 1 x Seminar 15%

Reasoning
- 3 x Assignment (10% each) (tentative)
What is Search?

- Search finds a solution to a problem
- Landscape of solutions
- Some are “good”
- Lots (and lots and lots) are “bad”

How do we find the good ones?
What is Search?

- Landscape can continuous or discrete
- (We will only deal with discrete)
- Landscape can be (approximated by?) a tree
- Landscape can be (approximated by?) a graph
What is Search?

\[ X + 2 \times Y < 3, \quad X \in [0,5] \quad Y \in [1,5] \]
What is Search?

\[ X + 2 \times Y < 3, \quad X \in [0,5], \quad Y \in [1,5] \]
What is Search?

\[ X + 2 \times Y < 3, \quad X \in [0,5] \quad Y \in [1,5] \]
What is Search?

\[ X + 2 \times Y < 3, \quad X \in [0,5], \quad Y \in [1,5] \]
What is Search?

\[ X + 2 * Y < 3, \quad X \in [0,5] \quad Y \in [1,5] \]

\[ X > 0 \quad X = 0 \quad Y > 1 \quad Y = 1 \]
What is Search?

Different flavours
- Find a solution (Satisfaction/Decision)
- Find the *best* solution (Combinatorial Optimisation)

Decision problems:
- Search variant: Find a solution (or show none exists)
- Decision variant: Does a solution exist?
What is Search

- **Constructive Search** – Find a solution by construction
- **Local Search** – Given a solution, explore the “neighbourhood” of that solution to find other (possibly better) solutions
What is Search?

Anytime algorithm (for optimisation prob):
- After a (short) startup time, an answer is available
- The longer the alg is run, the better the solution
- Quality guarantee may be available (e.g. solution is within 5% of optimal)

One-shot algorithm
- Answer only available at completion
What is Search?

A problem is defined by

- Initial state
- Successor function
  (an action leading from one state to another)
- Goal test
- Path cost

A solution is a sequence of actions leading from
the start state to a goal state
Example: Romania
Problem formulation

A problem is defined by four items:

- initial state
  - e.g., “at Arad”

- successor function $S(x)$
  - set of action–state pairs
  - e.g., $S(\text{Arad}) = \{<\text{Arad} \rightarrow \text{Zerind}, \text{at Zerind}>, \ldots\}$

- goal test e.g., $x = \text{“at Bucharest”}$
  - can be implicit, e.g., $\text{HasAirport}(x)$

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

function Tree-Search \((\text{problem, fringe})\) returns a solution, or failure

\[
\text{fringe} \leftarrow \text{Insert} (\text{Make-Node (Initial-State (\text{problem})), fringe})
\]

loop do
  if Is-Empty (fringe) then return failure

  \(\text{node} \leftarrow \text{Remove-Front (fringe)}\)
  if Goal-Test (\text{problem, State[node]}) then return \text{node}

  \(\text{fringe} \leftarrow \text{InsertAll (Expand (node, problem), fringe)}\)

function Expand \((\text{node, problem})\) returns a set of nodes

\[\text{successors} \leftarrow \text{the empty set}\]

for each \((\text{action, result})\) in Successor-Fn \((\text{problem, State[node]})\) do
  \(\text{s} \leftarrow \text{a new Node}\)
  \(\text{Parent-Node}[s] \leftarrow \text{node}; \text{Action}[s] \leftarrow \text{action}; \text{State}[s] \leftarrow \text{result}\)
  \(\text{Path-Cost}[s] \leftarrow \text{Path-Cost[node]} + \text{Step-Cost(State[node], action, result)}\)
  \(\text{Depth}[s] \leftarrow \text{Depth[node]} + 1\)
  \(\text{successors} \leftarrow \text{Insert (s)}\)

return \text{successors}
Breadth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Breadth-first Search

A

B

D

H

E

I

F

L

G

M

O

C

M

L

I

H

D

B

A

Search – Lecture 1-2

25/07/2008
Breadth-first Search
Breadth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Breadth-first Search

A

B

D E

H I

C F

L

G M

O
Breadth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Breadth-first Search

A

B

C

D

E

F

G

H

I

J

K

L

M

O
Breadth-first Search

A
---
B
---
D
---
H
---
I
---
E
---
L
---
F
---
G
---
M
---
O
---
C
---

25/07/2008

Search – Lecture 1-2
Breadth-first Search

A
  B
  D
  H
  I
  E
  L
  F
  G
  M
  O
  C
Breadth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Breadth-first Search
Breadth-first Search
Breadth-first Search

A

B

D

H

I

E

F

G

L

M

O

C
Breadth-first Search

A

B

C

D

E

F

G

H

I

L

M

!!

O
Depth-first Search

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Depth-first Search

Node A is the root node. The tree is traversed in a depth-first manner, exploring as far as possible along each branch before backtracking. The path from the root to the goal node is highlighted in red.
Depth-first Search

![Depth-first Search diagram]

- **A**
- **B**
- **C**
- **D**
- **E**
- **F**
- **G**
- **H**
- **I**
- **L**
- **M**
- **O**
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Depth-first Search
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Depth-first Search
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Depth-first Search

A

B

D

H

I

E

F

L

C

G

M

O
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Depth-first Search

A

B

C

D

E

F

G

H

I

L

M

O
Breadth-first Search

A

B

C

D

E

F

G

H

I

L

M

O

!!
Iterative Deepening DFS Search

A

B

D

H

C

E

I

F

L

G

M

O

Iterative Deepening DFS Search

A

B

D

H

C

E

I

F

L

G

M

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

J

K

L

M

O
Iterative Deepening DFS Search

Diagram of a search tree with nodes labeled A, B, C, D, E, F, G, H, I, L, M, O. The tree starts with node A, and branches out with nodes B, C, D, E, F, G, H, I, L, M, O.
Iterative Deepening DFS Search

A

B

D

H

I

C

E

F

G

L

M

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search

A
B
C
D
E
F
G
H
I
L
M
O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search
Iterative Deepening DFS Search
Iterative Deepening DFS Search
Iterative Deepening DFS Search

Diagram of iterative deepening search algorithm.
Iterative Deepening DFS Search
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O
Iterative Deepening DFS Search
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search

- A
  - B
    - D
      - H
    - E
  - C
    - F
      - L
    - G
      - M
      - O
Iterative Deepening DFS Search

A

B
D
H
I

E

F
L

C
G
M
O
Iterative Deepening DFS Search
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

O
Iterative Deepening DFS Search

A

B

C

D

E

F

G

H

I

L

M

!!

O
Uniform cost search

- Expand in order of cost
- Identical to Breadth-First Search if all step costs are equal

Number of nodes expanded depends on $\varepsilon > 0$ – the smallest cost action cost.

(If $\varepsilon = 0$, uniform-Cost search may infinite-loop)
Bi-directional Search

- Extend BFS from Start and Goal states simultaneously

\[ b^{d/2} + b^{d/2} \ll b^d \]
Bi-directional Search

Restrictions:

- Can only use if 1 (or a few) known goal nodes
- Can only use if actions are reversible
  i.e. if we can efficiently calculate Pred(s) (for state s)

where

\[
Pred(s) = \{ r : \exists a \ r \xrightarrow{a} s \}
\]
## Summary

<table>
<thead>
<tr>
<th>Criterion</th>
<th>BFS</th>
<th>DFS</th>
<th>IDDFS</th>
<th>Uniform Cost</th>
<th>Bi-directional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^m)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{C*/\varepsilon})$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^{d+1})$</td>
<td>$O(bm)$</td>
<td>$O(bd)$</td>
<td>$O(b^{C*/\varepsilon})$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal</td>
<td>Yes, for unit cost</td>
<td>No</td>
<td>Yes, for unit cost</td>
<td>Yes</td>
<td>Yes, for unit cost</td>
</tr>
</tbody>
</table>

$b = \text{branching factor} ;  m = \text{max depth of tree} ;  d = \text{depth of goal node}$

$C^* = \text{cost of goal} ; \varepsilon = \text{min action cost}$
Repeated State Detection

- Test for previously-seen states
- Essentially, search graph mimics state graph
Repeated State Detection

✓ Can make exponential improvement in time
✓ Can make DFS complete
✖ DFS now has potentially exponential space requirement
✖ Have to be careful to guarantee optimality

In practice, almost always use repeated state detection
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 \( n \)
- \( h(n) \) = estimated cost from \( n \) to the closest goal
- \( f(n) \) = 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
When \( h(n) \) is admissible, \( f(n) \) never overestimates the total cost of the shortest path through \( n \) to the goal
Theorem: A* search is optimal
A* Example

Arad
366=0+366
A* Example

Arad
366 = 0 + 366
A* Example

Arad

Sibiu
393 = 140 + 253

Timisoara
447 = 118 + 329

Zerind
449 = 75 + 374
A* Example

- Arad
  - Sibiu: 393 = 140 + 253
  - Timisoara: 447 = 118 + 329
  - Zerind: 449 = 75 + 374
A* Example

- Arad
  - Sibiu
    - Arad: 646 = 280 + 366
    - Fagaras: 415 = 239 + 176
    - Oradea: 671 = 291 + 380
    - Rimnicu Vilcea: 413 = 220 + 193
  - Timisoara: 447 = 118 + 329
  - Zerind: 449 = 75 + 374
A* Example

Arad

Sibiu

Fagaras 415=239+176

Oradea 671=291+380

Timisoara 447=118+329

Rimnicu Vilcea 413=220+193

Zerind 449=75+374

Arad 646=280+366
A* Example

- **Arad**
  - **Sibiu**
    - **Arad**
      - Distance: 646 = 280 + 366
    - **Fagaras**
      - Distance: 415 = 239 + 176
    - **Oradea**
      - Distance: 671 = 291 + 380
  - **Timisoara**
    - Distance: 447 = 118 + 329
  - **Rimnicu Vilcea**
    - **Craiova**
      - Distance: 526 = 366 + 160
    - **Pitesti**
      - Distance: 417 = 317 + 100
  - **Sibiu**
    - Distance: 553 = 300 + 253
- **Zerind**
  - Distance: 449 = 75 + 374
A* Example

Arad

Sibiu

Timisoara
447=118+329

Zerind
449=75+374

Arad
646=280+366

Fagaras
415=239+176

Oradea
671=291+380

Rimnicu Vilcea

Craiova
526=366+160

Pitesti
417=317+100

Sibiu
553=300+253
A* Example

A* Search

Nodes:
- Arad
- Sibiu
- Timisoara
- Zerind
- Fagaras
- Oradea
- Rimnicu Vilcea
- Bucharest
- Craiova
- Pitesti
- Sibiu

Costs:
- Arad to Sibiu: 646 = 280 + 366
- Sibiu to Fagaras: 450 = 450 + 0
- Arad to Bucharest: 591 = 338 + 253
- Sibiu to Craiova: 526 = 366 + 160
- Arad to Timisoara: 447 = 118 + 329
- Sibiu to Pitesti: 417 = 317 + 100
- Arad to Zerind: 449 = 75 + 374
- Sibiu to Sibiu: 553 = 300 + 253.
A* Example

- Arad
  - Sibiu
    - Arad: 646 = 280 + 366
    - Fagaras: 450 = 450 + 0
    - Bucharest: 450 = 450 + 0
  - Timisoara: 447 = 118 + 329
    - Oradea: 671 = 291 + 380
    - Craiova: 526 = 366 + 160
    - Pitesti: 417 = 317 + 100
  - Zerind: 449 = 75 + 374
    - Rimnicu Vilcea: 553 = 300 + 253
    - Sibiu: 591 = 338 + 253
A* Example

- **Arad**
  - Sibiu
    - Arad: 646 = 280 + 366
    - Sibiu: 591 = 338 + 253
    - Bucharest: 450 = 450 + 0
  - Fagaras
    - Bucharest: 418 = 418 + 0
  - Oradea: 671 = 291 + 380
    - Craiova: 526 = 366 + 160
    - Pitesti
      - Bucharest: 418 = 418 + 0
      - Craiova: 615 = 455 + 160
      - Rimnicu Vilcea: 607 = 414 + 193
- Timisoara: 447 = 118 + 329
- Zerind: 449 = 75 + 374

Pitesti

- Sibiu: 553 = 300 + 253

Rimnicu Vilcea

- Bucharest
- Craiova: 615 = 455 + 160
- Rimnicu Vilcea: 607 = 414 + 193

25/07/2008
A* Example

- **Arad**
  - **Sibiu**
    - **Arad**
      - **Bucharest**
        - **Sibiu**: 591 = 338 + 253
      - **Fagaras**
        - **Bucharest**: 450 = 450 + 0
    - **Oradea**
      - **Craiova**: 526 = 366 + 160
      - **Rimnicu Vilcea**: 607 = 414 + 193
    - **Timisoara**: 447 = 118 + 329
    - **Zerind**: 449 = 75 + 374
  - **Fagaras**
  - **Oradea**: 671 = 291 + 380
  - **Rimnicu Vilcea**
  - **Craiova**: 591 = 338 + 253
  - **Pitesti**: 553 = 300 + 253
  - **Bucharest**: 418 = 418 + 0
  - **Craiova**: 615 = 455 + 160
  - **Rimnicu Vilcea**: 607 = 414 + 193
A* Example

- Arad
  - Sibiu
    - Arad: 646 = 280 + 366
    - Fagaras: Bucharest: 450 = 450 + 0
  - Oradea: 671 = 291 + 380
  - Timisoara: 447 = 118 + 329
  - Zerind: 449 = 75 + 374
  - Sibiu: 591 = 338 + 253
    - Bucharest: 450 = 450 + 0
  - Craiova: Rimnicu Vilcea: 615 = 455 + 160
    - Craiova: 526 = 366 + 160
  - Pitesti: Sibiu: 553 = 300 + 253
    - Pitesti: Bucharest: 418 = 418 + 0
  - Rimnicu Vilcea: Bucharest: 607 = 414 + 193
    - Bucharest: 615 = 455 + 160
A* Search

✓ Optimal
✓ No other optimal algorithm expands fewer nodes
  (modulo order of expansion of nodes with same cost)
✗ Exponential space requirement
Greedy Search

- Use (always admissible) $h(n) = 0$

- in $f(n) = g(n) + h(n)$

- *i.e.* always use the cheapest found so far
Variants of A* Search

Many variants of A* developed

- **Weighted A***: \( f(n) = g(n) + W h(n) \)
  - Not exact
  - Faster
  - Answer is within W of optimal
Variants of A* Search

Many variants of A* developed

- Weighted A*: $f(n) = g(n) + W h(n)$
  - Not exact
  - Faster
  - Answer is within $W$ of optimal
Variants of A* Search

Many variants of A* developed

- **Weighted A***: \( f(n) = g(n) + W h(n) \)
  - Not exact
  - Faster
  - Answer is within \( W \) of optimal
Variants of A* Search

- **Iterative Deepening A* (IDA*)**
  - Incrementally increase max depth

- **Memory-bounded A***
  - Impose bound on memory
  - Discard nodes with largest $f(x)$ if necessary

- Others to come in seminars…
Limited Discrepancy Search

- Requires a heuristic to order choices
  (e.g. admissible heuristic, but don’t require underestimate)
- If heuristic was right, we could always take the choice given by the heuristic
- LDS allows the heuristic to make mistakes:
  $k$-LDS allows $k$ mistakes.
Limited Discrepancy Search \((k = 1)\)
Limited Discrepancy Search \((k = 1)\)
Limited Discrepancy Search ($k = 1$)
Limited Discrepancy Search (\(k = 1\))
Limited Discrepancy Search (\(k = 1\))
Limited Discrepancy Search ($k = 1$)
Limited Discrepancy Search ($k = 1$)
Limited Discrepancy Search ($k = 1$)
Limited Discrepancy Search \((k = 1)\)
Limited Discrepancy Search \((k = 1)\)
Limited Discrepancy Search \((k = 1)\)
Next week....

(Integer) Linear Programming and Branch & Bound