Informed search methods.
Constraint satisfaction search.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Announcements

- Homework assignment 2 is out
  - Due on Wednesday, September 27, 2006
  - Two parts:
    • Pen and pencil part
    • Programming part – heuristics for (Puzzle 8)

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs1571/
Evaluation-function driven search

- A search strategy can be defined in terms of a node evaluation function
- Evaluation function
  - Denoted $f(n)$
  - Defines the desirability of a node to be expanded next

- Evaluation-function driven search: expand the node (state) with the best evaluation-function value
- Implementation: priority queue with nodes in the decreasing order of their evaluation function value

Best-first search

Best-first search
- incorporates a heuristic function, $h(n)$, into the evaluation function $f(n)$ to guide the search.
- heuristic function: measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):
  - Greedy search
    $$f(n) = h(n)$$
  - A* algorithm
    $$f(n) = g(n) + h(n)$$
  + iterative deepening version of A*: IDA*
Properties of greedy search

- **Completeness:** No. We can loop forever. Nodes that seem to be the best choices can lead to cycles. Elimination of state repeats can solve the problem.

- **Optimality:** No. Even if we reach the goal, we may be biased by a bad heuristic estimate. Evaluation function disregards the cost of the path built so far.

- **Time complexity:** \(O(b^m)\)
  Worst case !!! But often better!

- **Memory (space) complexity:** \(O(b^m)\)
  Often better!
A* search

- The problem with the greedy search is that it can keep expanding paths that are already very expensive.
- The problem with the uniform-cost search is that it uses only past exploration information (path cost), no additional information is utilized.
- A* search
  \[ f(n) = g(n) + h(n) \]
  - \( g(n) \) - cost of reaching the state
  - \( h(n) \) - estimate of the cost from the current state to a goal
  - \( f(n) \) - estimate of the path length
- **Additional A* condition:** admissible heuristic
  \[ h(n) \leq h^*(n) \text{ for all } n \]
Properties of A* search

- **Completeness**: Yes.
- **Optimality**: Yes (with the admissible heuristic)
- **Time complexity**:
  - Order roughly the number of nodes with $f(n)$ smaller than the cost of the optimal path $g^*$
- **Memory (space) complexity**:
  - Same as time complexity (all nodes in the memory)

Admissible heuristics

- Heuristics are designed based on relaxed version of problems
- **Example**: the 8-puzzle problem

<table>
<thead>
<tr>
<th>Initial position</th>
<th>Goal position</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td>1 2 3</td>
</tr>
<tr>
<td>6 1 8</td>
<td>4 5 6</td>
</tr>
<tr>
<td>7 3 2</td>
<td>7 8</td>
</tr>
</tbody>
</table>

- **Admissible heuristics**:
  1. number of misplaced tiles
  2. Sum of distances of all tiles from their goal positions (Manhattan distance)
Admissible heuristics

• We can have multiple admissible heuristics for the same problem
• **Dominance:** Heuristic function \( h_1 \) dominates \( h_2 \) if
  \[ \forall n \quad h_1(n) \geq h_2(n) \]
• **Combination:** two or more admissible heuristics can be combined to give a new admissible heuristic
  – Assume two admissible heuristics \( h_1, h_2 \)
  Then: \[ h_3(n) = \max(h_1(n), h_2(n)) \] is admissible

IDA*

**Iterative deepening version of A***

• Progressively increases the **evaluation function limit** (instead of the depth limit)

• Performs **limited-cost depth-first search** for the current evaluation function limit
  – Keeps expanding nodes in the depth first manner up to the evaluation function limit

• **Problem:** the amount by which the evaluation limit should be progressively increased
**IDA***

**Problem:** the amount by which the evaluation limit should be progressively increased

**Solutions:**

1. peak over the previous step boundary to guarantee that in the next cycle some number of nodes are expanded
2. Increase the limit by a fixed cost increment – say $\varepsilon$

![Diagram showing IDA* solutions]

**Solution 1:** peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

**Properties:**
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- Assume I choose a limit such that at least 5 new nodes are examined in the next DFS run
- What is the problem here?
**IDA***

**Solution 1:** peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

**Properties:**
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- Assume I choose a limit such that at least 5 new nodes are examined in the next DFS run
- What is the problem here?
  - We may find a sub-optimal solution
    - **Fix:**? Complete the search up to the limit to find the best

---

**IDA***

**Solution 1:** peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

**Properties:**
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- Assume I choose a limit such that at least 5 new nodes are examined in the next DFS run
- What is the problem here?
  - We may find a sub-optimal solution
    - **Fix:** complete the search up to the limit to find the best
**IDA**

**Solution 2: Increase the limit by a fixed cost increment (ε)**

![Diagram of search tree with cost limit = k \( \varepsilon \)](_k-th step_)

**Properties:**

- What is bad?

---

**IDA**

**Solution 2: Increase the limit by a fixed cost increment (ε)**

![Diagram of search tree with cost limit = k \( \varepsilon \)](_k-th step_)

**Properties:**

- What is bad? Too many or too few nodes expanded – no control of the number of nodes
- What is the of the solution?
**Solution 2:** Increase the limit by a fixed cost increment ($\varepsilon$)

Properties:
- What is bad? Too many or too few nodes expanded – no control of the number of nodes
- What is the cost of the solution? The solution differs by $< \varepsilon$

Cost limit = $k \varepsilon$

---

**Constraint satisfaction search**
Search problem

A search problem:
- **Search space (or state space):** a set of objects among which we conduct the search;
- **Initial state:** an object we start to search from;
- **Operators (actions):** transform one state in the search space to the other;
- **Goal condition:** describes the object we search for

- **Possible metric on a search space:**
  - measures the quality of the object with respect to the goal

Search problems occur in planning, optimizations, learning

---

Constraint satisfaction problem (CSP)

Two types of search:
- **path search** (a path from the initial state to a state satisfying the goal condition)
- **configuration search** (a configuration satisfying goal conditions)

**Constraint satisfaction problem (CSP)** is a **configuration search problem** where:
- A **state** is defined by a set of variables
- **Goal condition** is represented by a set constraints on possible variable values

Special properties of the CSP allow more specific procedures to be designed and applied for solving them
Example of a CSP: N-queens

Goal: n queens placed in non-attacking positions on the board

Variables:
- Represent queens, one for each column:
  - $Q_1, Q_2, Q_3, Q_4$
- Values:
  - Row placement of each queen on the board
    \{1, 2, 3, 4\}

Constraints:
- $Q_i \neq Q_j$  
  Two queens not in the same row
- $|Q_i - Q_j| \neq |i - j|$  
  Two queens not on the same diagonal

Satisfiability (SAT) problem

Determine whether a sentence in the conjunctive normal form (CNF) is satisfiable (can evaluate to true)
- Used in the propositional logic (covered later)

\[(P \lor Q \lor \neg R) \land (\neg P \lor \neg R \lor S) \land (\neg P \lor Q \lor \neg T)\ldots\]

Variables:
- Propositional symbols (P, R, T, S)
- Values: True, False

Constraints:
- Every conjunct must evaluate to true, at least one of the literals must evaluate to true
  \[(P \lor Q \lor \neg R) \equiv True , (\neg P \lor \neg R \lor S) \equiv True , \ldots\]
Other real world CSP problems

Scheduling problems:
- E.g. telescope scheduling
- High-school class schedule

Design problems:
- Hardware configurations
- VLSI design

More complex problems may involve:
- real-valued variables
- additional preferences on variable assignments – the optimal configuration is sought

Map coloring

Color a map using k different colors such that no adjacent countries have the same color

Variables: ?

• Variable values: ?

Constraints: ?
Map coloring

Color a map using $k$ different colors such that no adjacent countries have the same color.

**Variables:**
- Represent countries: $A, B, C, D, E$
- Values: $k$ different colors
  - {Red, Blue, Green,..}

**Constraints:**

An example of a problem with **binary constraints**
Constraint satisfaction as a search problem

Formulation of a CSP as a search problem:
• **States.** Assignment (partial, complete) of values to variables.
• **Initial state.** No variable is assigned a value.
• **Operators.** Assign a value to one of the unassigned variables.
• **Goal condition.** All variables are assigned, no constraints are violated.

• **Constraints** can be represented:
  – **Explicitly** by a set of allowable values
  – **Implicitly** by a function that tests for the satisfaction of constraints

Solving CSP as a standard search
Solving a CSP through standard search

• Maximum depth of the tree (m): ?
• Depth of the solution (d) : ?
• Branching factor (b) : ?

Unassigned: Q₁, Q₂, Q₃, Q₄
Assigned: 

Unassigned: Q₂, Q₃, Q₄
Assigned: Q₁ = 1

Unassigned: Q₂, Q₃, Q₄
Assigned: Q₁ = 2

Unassigned: Q₃, Q₄
Assigned: Q₁ = 2, Q₂ = 4

CS 1571 Intro to AI  M. Hauskrecht

Solving a CSP through standard search

• Maximum depth of the tree: Number of variables in the CSP
• Depth of the solution: Number of variables in the CSP
• Branching factor: if we fix the order of variable assignments
  the branch factor depends on the number of their values

Unassigned: Q₁, Q₂, Q₃, Q₄
Assigned: 

Unassigned: Q₂, Q₃, Q₄
Assigned: Q₁ = 1

Unassigned: Q₂, Q₃, Q₄
Assigned: Q₁ = 2

Unassigned: Q₃, Q₄
Assigned: Q₁ = 2, Q₂ = 4

CS 1571 Intro to AI  M. Hauskrecht
Solving a CSP through standard search

• What search algorithm to use: ?

Depth of the tree = Depth of the solution = number of vars

Unassigned: $Q_1, Q_2, Q_3, Q_4$
Assigned:

Unassigned: $Q_2, Q_3, Q_4$
Assigned: $Q_1 = 1$

Unassigned: $Q_2, Q_3, Q_4$
Assigned: $Q_1 = 2$

Unassigned: $Q_1, Q_4$
Assigned: $Q_1 = 2, Q_4 = 4$

...
Solving a CSP through standard search

- **What search algorithm to use:** Depth first search !!!
  - Since we know the depth of the solution
  - We do not have to keep large number of nodes in queues

---

Backtracking

**Depth-first search for CSP** is also referred to as backtracking

The violation of constraints needs to be checked for each node, either during its generation or before its expansion

**Consistency of constraints:**
- Current variable assignments together with constraints restrict remaining legal values of unassigned variables;
- The remaining legal and illegal values of variables may be inferred (effect of constraints propagates)
- To prevent “blind” exploration it is necessary to keep track of the remaining legal values, so we know when the constraints are violated and when to terminate the search