Game search.

Administration

- PS–2 due today
  - Report before the class begins
  - Programs through ftp

- PS-3 is out
  - on the course web page
  - due next week on Tuesday, September 24, 2002
    - Report
    - Programs
Topics

Search for optimal configurations (cont.)
- Review: Hill climbing, Simulated annealing
- Genetic algorithms
- Configuration search with continuous variables

Games
- Adversarial vs. Cooperative games
- Search tree for adversarial games
- Minimax algorithm
- Speedups:
  - Alpha-Beta pruning
  - Search tree cutoff with heuristics
Search for the optimal configuration

Configuration-search problems:
• Are often enhanced with some quality measure

Quality measure
• reflects our preference towards each configuration (or state)

Goal
• find the configuration with the optimal quality

Example: Traveling salesman problem

Problem:
• A graph with distances

• Goal: find the shortest tour which visits every city once and returns to the start

An example of a valid tour: ABCDEF
Iterative improvement algorithms

• Give solutions to the configuration-search with the optimality measure

Properties of iterative improvement algorithms:
• Search the space of “complete” configurations
• Operators make “local” changes to “complete” configurations
• Keep track of just one state (the current state), not a memory of past states
  – !!! No search tree is necessary !!!

Example: Traveling salesman problem

“Local” operator for generating the next state:
• divide the existing tour into two parts,
• reconnect the two parts in the opposite order

Example:

```
ABCDEF
   ↓
ABCD | EF |
   ↓
ABCDFE
```

```text
Example:
```
```

```
```

```
A
```

```
B
```

```
C
```

```
D
```

```
E
```

```
F
```

Part 1

Part 2

CS 1571 Intro to AI

CS 1571 Intro to AI
Searching configuration space

Iterative improvement algorithms
• keep only one configuration (the current configuration) active

Problem:
• How to decide about which operator to apply?

Iterative improvement algorithms

Two strategies to choose the configuration (state) to be visited next:
– Hill climbing
– Simulated annealing

• Later: Extensions to multiple current states:
  – Genetic algorithms

• Note: Maximization is inverse of the minimization
\[ \min f(X) \leftrightarrow \max \left[ -f(X) \right] \]
Hill climbing

- **Local improvement algorithm**
- Look around at states in the local neighborhood and choose the one with the best value
- Assume: we want to maximize the

![Hill Climbing Diagram](image)

Hill climbing

- Hill climbing can get trapped in the local optimum

![Hill Climbing Diagram](image)
Hill climbing

• Hill climbing can get clueless on plateaus

Simulated annealing

• Permits “bad” moves to states with lower values, thus escape the local optima
• Gradually decreases the frequency of such moves and their size (parameter controlling it – temperature)
Simulated annealing algorithm

- The probability of moving into a state with a higher energy is 1
- The probability of moving into a state with a lower value is

$$e^{\Delta E / T}$$

The probability is:
- Proportional to the energy difference $\Delta E$
- Modulated through a temperature parameter $T$:
  - for $T \to \infty$ the probability of any move approaches 1
  - for $T \to 0$ the probability that a state with smaller value is selected goes down and approaches 0

- **Cooling schedule:**
  - Schedule of changes of a parameter $T$ over iteration steps

Simulated annealing algorithm

- **Simulated annealing algorithm**
  - developed originally for modeling physical processes
    (Metropolis et al, 53)

- **Properties:**
  - If $T$ is decreased slowly enough the best configuration (state) is always reached

- **Applications:**
  - VLSI design
  - airline scheduling
Simulated evolution and genetic algorithms

• Limitations of simulated annealing:
  – Pursues one state configuration;
  – Changes to configurations are typically local

Can we do better? May be …

• Assume we have two configurations with good values that are quite different
• We expect that the combination of the two individual configurations may lead to a configuration with higher value (Not guaranteed !!!)

This is the idea behind genetic algorithms in which we modify a population of configurations

Genetic algorithms

Algorithm idea:

• Create a population of random configurations
• Create a new population through:
  – Biased selection of pairs of configurations from the previous population
  – Crossover (combination) of pairs
  – Mutation of resulting individuals
• Evolve the population over multiple generation cycles

• Selection of configurations to be combined:
  – Fitness function = value function
    measures the quality of an individual (a state) in the population
Reproduction process in GA

- Assume that a state configuration is defined by a set variables with two values, represented as 0 or 1

Parametric optimization

- **Configuration search:**
  - Optimizes the measure of the configuration quality
  - Additional constraints are possible
- When state space we search is finite, the search problem is called a **combinatorial optimization problem**
- When parameters we want to find are real-valued
  - **parametric optimization problem**

**Parametric optimization:**

- Configurations are described by a vector of free parameters (variables) \( w \) with real-valued values
- **Goal:** find the set of parameters \( w \) that optimize the quality measure \( f(w) \)
Parametric optimization techniques

• Special cases (with efficient solutions):
  – Linear programming
  – Quadratic programming

• First-order methods:
  – Gradient-ascent (descent)
  – Conjugate gradient

• Second-order methods:
  – Newton-Rhapson methods
  – Levenberg-Marquardt

• Constrained optimization:
  – Lagrange multipliers

Gradient ascent method

• **Gradient ascent**: the same as hill-climbing, but in the continuous parametric space \( w \)

\[
f(w) \quad \frac{\partial}{\partial w} f(w) \bigg|_{w^*}
\]

• Change the parameter value of \( w \) according to the gradient

\[
w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) \bigg|_{w^*}
\]
Gradient ascent method

- New value of the parameter
  \[ w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) \big|_{w^*} \]
  \[ \alpha > 0 \quad \text{a learning rate (scales the gradient changes)} \]

- Problems: local optima, saddle points, slow convergence
- More complex optimization techniques use additional information (e.g. second derivatives)
Game search

• Game-playing programs developed by AI researchers since the beginning of the modern AI era
  – Programs playing chess, checkers, etc (1950s)

• Specifics of the game search:
  – Sequences of player’s decisions we can control
  – Opponent’s decisions (responses) we do not control

• Contingency problem: many possible opponent’s moves must be “covered” by the solution
  Opponent’s behavior introduces an uncertainty in to the game
  – We do not know exactly what the response is going to be

• Rational opponent – maximizes it own utility (payoff) function
Types of game problems

• Types of game problems:
  – **Adversarial games:**
    • win of one player is a loss of the other
  – **Cooperative games:**
    • players have common interests and utility function
  – A spectrum of game problems in between the two:

Adversarial games

Fully cooperative games

Here we focus on adversarial games!!

Example of an adversarial 2 person game:
Tic-tac-toe

• We have the first move (x)
Game search problem

- **Game problem formulation:**
  - **Initial state:** initial board position + info whose move it is
  - **Operators:** legal moves a player can make
  - **Goal (terminal test):** determines when the game is over
  - **Utility (payoff) function:** measures the outcome of the game and its desirability

- **Search objective:**
  - find the sequence of player’s decisions (moves) maximizing its utility (payoff)
  - Consider the opponent’s moves and their utility

---

Game problem formulation (Tic-tac-toe)

**Objectives:**
- **Player 1:** maximize outcome
- **Player 2:** minimize outcome

**Initial state**

**Operators**

**Utility:** -1 0 1

**Terminal (goal) states**
Minimax algorithm

How to deal with the contingency problem?

- Assuming that the opponent is rational and always optimizes its behavior (opposite to us) we consider the best opponent’s response
- Then the **minimax algorithm** determines the best move

Minimax algorithm. Example
Minimax algorithm. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5
Minimax algorithm. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 3 6 2 2 1 9 5 3 1 5 4 7 5

MINIMAX

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 3 6 2 2 1 9 5 3 1 5 4 7 5

MINIMAX
Minimax algorithm

function MINIMAX-DECISION(game) returns an operator
  for each op in OPERATORS[game] do
    VALUE[op] = MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]

function MINIMAX-VALUE(state, game) returns a utility value
  if TERMINAL-TEST(game)(state) then
    return UTILITY(game)(state)
  else if MAX is to move in state then
    return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(state)

Complexity of the minimax algorithm

- We need to explore the complete game tree before making the decision

\[
\begin{align*}
\text{Complexity:} & \quad ? \\
\text{Depth:} & \quad m \\
\text{Width:} & \quad b
\end{align*}
\]
Complexity of the minimax algorithm

- We need to explore the complete game tree before making the decision.
  ![Game Tree Diagram]

- Impossible for large games
  - Chess: 35 operators, game can have 50 or more moves

Complexity:
\[ O(b^m) \]

Solution to the complexity problem

Two solutions:

1. **Dynamic pruning of redundant branches** of the search tree
   - identify provably suboptimal branch of the search tree even before it is fully explored
   - Cutoff the suboptimal branch
   Procedure: **Alpha-Beta pruning**

2. **Early cutoff of the search tree**
   - uses imperfect minimax value estimate of non-terminal states.
Alpha beta pruning

- Some branches will never be played by rational players since they include sub-optimal decisions (for either player)

\[ 2 \leq 3 \]

Alpha beta pruning. Example
Alpha beta pruning. Example

MAX

MIN

MAX

\[ \geq 4 \]

\[ \leq 4 \]

CS 1571 Intro to AI
Alpha beta pruning. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 = 4

6 ≥ 4

6 ≥ 4

Alpha beta pruning. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 = 4

6 ≥ 4

6 ≥ 4

CS 1571 Intro to AI

CS 1571 Intro to AI
Alpha beta pruning. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 \geq 4

\geq 6

\geq 2

\leq 2

\geq 4

CS 1571 Intro to AI
Alpha beta pruning. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 = 6 ≥ 4 = 4 ≥ 2 = 2 ≤ 2 ←!!
Alpha beta pruning. Example
Alpha beta pruning. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

= 4

≥ 6

= 2

≤ 2

= 5

≥ 7

nodes that were never explored !!!

CS 1571 Intro to AI
### Alpha-Beta pruning

Function `Max-Value(state, game, α, β)` returns the minimax value of state
- **Inputs:** state, current state in game; game, game description; α, the best score for MAX along the path to state; β, the best score for MIN along the path to state.

1. If `Goal-Test(state)` then return `Eval(state)`
2. For each `s` in `Successors(state)` do:
   - `α ← Max(α, Min-Value(s, game, α, β))`
   - If `α ≥ β` then return `β`
3. Return `α`

Function `Min-Value(state, game, α, β)` returns the minimax value of state
- **Inputs:** state, current state in game; game, game description; α, the best score for MAX along the path to state; β, the best score for MIN along the path to state.

1. If `Goal-Test(state)` then return `Eval(state)`
2. For each `s` in `Successors(state)` do:
   - `β ← Min(β, Max-Value(s, game, α, β))`
   - If `β ≤ α` then return `α`
3. Return `β`

---

### Using minimax value estimates

- **Idea:**
  - Cutoff the search tree before the terminal state is reached
  - Use imperfect estimate of the minimax value at the leaves

**Evaluation function**

```
MAX

MIN

Heuristic evaluation function

Cutoff level
```
Design of evaluation functions

- **Heuristic estimate** of the value for a sub-tree
- **Example of a heuristic functions:**
  - Material advantage in chess, checkers
    - Gives a value to every piece on the board, its position and combines them
  - More general **feature-based evaluation function**
    - Typically a linear evaluation function:
      \[
      f(s) = f_1(s)w_1 + f_2(s)w_2 + \ldots + f_k(s)w_k
      \]
      \[
      f_i(s) \quad - \text{a feature of a state } s
      \]
      \[
      w_i \quad - \text{feature weight}
      \]

Further extensions to real games

- Restricted set of moves to be considered under **the cutoff level** to reduce branching and improve the evaluation function
  - E.g., consider only the capture moves in chess