Search for the optimal configuration

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

Homework assignment

• PS-2 is due today

• PS-3 is out, due on Wednesday, September 28
  – 2 Programming competitions for extra credit
    • Simulated annealing
    • Genetic algorithms
CSP revisited

• Constraint propagation
  Value propagation. Infers:
  – equations from the set of equations defining the partial assignment, and a constraint

  ![No equations/disequations are inferred](image1)

  ![No equations/disequations are inferred](image2)

CSP revisited

• Constraint propagation
  Forward checking. Infers:
  – disequations from a set of equations defining the partial assignment, and a constraint
  – Equations through the exhaustion of alternatives

  ![Invalid assignment](image3)
CSP revisited

• **Constraint propagation**

  **Arc consistency.** Infers:
  
  – disequations from the set of *equations and disequations* defining the partial assignment, and a constraint
  
  – equations through the exhaustion of alternatives

Search for the optimal configuration

**Configuration search problems:**

• 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

Example: N queens

- A CSP problem can be converted to the ‘optimal’ configuration problem
- The quality of a configuration in a CSP = the number of constraints violated
- Solving: minimize the number of constraint violations

# of violations = 3  # of violations = 1  # of violations = 0
Local search methods

- Often used to find solutions to large ‘optimal’ configuration problems

- Properties of local search 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: N-queens

- “Local” operators for generating the next state:
  - Select a variable (a queen)
  - Reallocate its position
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 |
ABCDDEF

Example: Traveling salesman problem

“Local” operator:
– generates the next configuration (state)
Searching the configuration space

Local search algorithms
- keep only one configuration (the current configuration) active

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

Local search 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
  \[
  \min f(X) \Leftrightarrow \max [-f(X)]
  \]
Hill climbing

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

![Diagram of hill climbing]

Hill climbing

- Always choose the next best successor state
- Stop when no improvement possible

```
function Hill-Climbing(problem) returns a solution state
    inputs: problem, a problem
    state: current, a node
            next, a node
    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        next ← a highest-valued successor of current
        if VALUE[next] < VALUE[current] then return current
        current ← next
    end
```

CS 2710 Foundations of AI
Hill climbing

• Look around at states in the local neighborhood and choose the one with the best value

• What can go wrong?

Hill climbing

• Hill climbing can get trapped in the local optimum

• What can go wrong?
Hill climbing

- Hill climbing can get clueless on plateaus

\[ \text{value} \]

\[ \text{states} \]

Hill climbing and n-queens

- The quality of a configuration is given by the number of constraints violated
- **Then:** Hill climbing reduces the number of constraints
- **Min-conflict strategy (heuristic):**
  - Choose randomly a variable with conflicts
  - Choose its value such that it violates the fewest constraints

Success !! But not always!!! The local optima problem!!!
Simulated annealing

- Permits “bad” moves to states with lower value, thus escape the local optima
- **Gradually decreases** the frequency of such moves and their size (parameter controlling it – **temperature**)

![Graph showing value vs. states, with states sometimes down and always up]

Simulated annealing algorithm

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

\[
p(\text{Accept } \text{NEXT} ) = e^{\Delta E / T} \quad \text{where} \quad \Delta E = E_{\text{NEXT}} - E_{\text{CURRENT}}
\]

The probability is:
- **Proportional to the energy difference**
- **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 at the time;
  - Changes to configurations are typically local

**Can we do better?**

- 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 grow a population of individual combinations

---

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 of variables with two values, represented as 0 or 1

![Diagram showing the reproduction process in GA]

Parametric optimization

**Optimal configuration search:**

- Configurations are described in terms of variables and their values
- Each configuration has a quality measure
- Goal: find the configuration with the best value

When the 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

**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 \( w \)

![](https://example.com/gradient_ascent_diagram.png)

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

\[
\begin{align*}
w & \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) \mid_{w^*}
\end{align*}
\]

- New value of the parameter

\[
\begin{align*}
w & \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) \mid_{w^*}
\end{align*}
\]

\( \alpha > 0 \) - a learning rate (scales the gradient changes)
Gradient ascent method

- To get to the function minimum repeat (iterate) the gradient based update few times

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