Methods for finding optimal configurations

Milos Hauskrecht
milos@pitt.edu
5329 Sennott Square

Search for the optimal configuration

Constrain satisfaction problem:
Objective: find a configuration that satisfies all constraints

Optimal configuration (state) problem:
Objective: find the best configuration (state)
The quality of a state: is defined by some quality measure that reflects our preference towards each configuration (or state)

Our goal: optimize the configuration according to the quality measure also referred to as objective function
Search for the optimal configuration

Optimal configuration search problem:
- Configurations (states)
- Each state has a quality measure $q(s)$ that reflects our preference towards each state
- **Goal:** find the configuration with the best value
  - Expressed in terms of the objective function $s^* = \text{argmax}_s q(s)$

If the space of configurations we search among is
- **Discrete or finite**
  - then it is a combinatorial optimization problem
- **Continuous**
  - then it solved as a parametric optimization problem

Example: Traveling salesman problem

Problem:
- A graph with distances
- A tour – a path that visits every city once and returns to the start e.g. ABCDEF

- **Goal:** find the shortest tour
Example: N queens

• A CSP problem
• Is it possible to formulate the problem as an optimal configuration search problem? Yes.

• The quality of a configuration in a CSP can be measured by the number of violated constraints

• Solving: minimize the number of constraint violations

# of violations = 3
# of violations = 1
# of violations = 0
Example: N queens

- Originally a CSP problem
- But it is also possible to formulate the problem as an optimal configuration search problem:
- Constraints are mapped to the objective cost function that counts the number of violated constraints

![N queens example](image)

- # of violations = 3
- # of violations = 0

Iterative optimization methods

- Searching systematically for the best configuration with the DFS may not be the best solution
- Worst case running time:
  - Exponential in the number of variables
- Solutions to large ‘optimal’ configuration problems are often found more effectively in practice using iterative optimization methods

- Examples of Methods:
  - Hill climbing
  - Simulated Annealing
  - Genetic algorithms
Iterative optimization methods

**Basic Properties:**

- Search the space of “complete” configurations
- Take advantage of local moves
  - Operators make “local” changes to “complete” configurations
- **Keep track of just one state** (the current state)
  - no memory of past states
  - !!! No search tree is necessary !!!

Iterative optimization methods

Current configuration

Quality \( q(s) = 167 \)

Next configurations

- \( q(s) = 145 \)
- \( q(s) = 150 \)
- \( q(s) = 180 \)
- \( q(s) = 180 \)
- \( q(s) = 167 \)
- \( q(s) = 191 \)
Iterative optimization methods

Quality $q(s) = 167$

Current configuration

Next configurations

Local operators

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

Part 1

Part 2

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

\[ \text{Part 1} \]

\[ \text{Part 2} \]
Example: Traveling salesman problem

“Local” operator:
- generates the next configuration (state)
  by rearranging the existing tour

\[ \begin{array}{c}
\text{ABCDEF} \\
\downarrow \\
\text{ABCD | EF} \\
\downarrow \\
\text{ABCDFE}
\end{array} \]

Searching the configuration space

Search algorithms
- keep only one configuration (the current configuration)

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

Strategies to choose the configuration (state) to be visited next:
- Hill climbing
- Simulated annealing

• Extensions to multiple current states:
  - Genetic algorithms

• Note: Maximization is inverse of the minimization
  \[ \min q(s) \leftrightarrow \max \left[ -q(s) \right] \]

Hill climbing

• Only configurations one can reach using local moves are consired as candidates
• What move the hill climbing makes?

value

A B C D E F

states

I am currently here

M. Hauskrecht
Hill climbing

- Look at 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

- How to solve the problem? Ideas?
Hill climbing

- **Solution: Multiple restarts** of the hill climbing algorithms from different initial states

A new starting state may lead to the globally optimal solution

Hill climbing

- Hill climbing can get clueless on plateaus
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 algorithm

- Based on a random walk in the configuration space

**Basic iteration step:**
- Choose uniformly at random one of the local neighbors of the current state as a candidate state
- if the candidate state is better than the current state
  then
  accept the candidate and make it the current state;
  else
  calculate the probability \( p(\text{ACCEPT}) \) of accepting it;
  using \( p(\text{ACCEPT}) \) choose randomly whether to accept or reject the candidate

---

Simulated annealing algorithm

**The probability \( p(\text{ACCEPT}) \) of the candidate state:**
- The probability of accepting a state with a better objective function value is always 1
- The probability of accepting a candidate with a lower objective function value is < 1 and equal:
- Let \( E \) denotes the objective function value (also called energy).

\[
p(\text{Accept } \text{NEXT}) = e^{\Delta E/T}
\]

where \( \Delta E = E_{\text{NEXT}} - E_{\text{CURRENT}} \)

\( T > 0 \)

- The probability is:
  - Proportional to the energy difference
Simulated annealing algorithm

Possible moves

Current configuration

Energy $E = 167$

Energy $E = 145$

Energy $E = 180$

Energy $E = 191$

Pick randomly from local neighbors

Simulated annealing algorithm

Possible moves

Current configuration

Energy $E = 167$

Energy $E = 145$

Energy $E = 180$

Energy $E = 191$

Random pick

$\Delta E = E_{NEXT} - E_{CURRENT}$

$= 145 - 167 = -22$

$p(Accept) = e^{\Delta E / T} = e^{-22 / T}$

Sometimes accept!
Simulated annealing algorithm

The probability of accepting a state with a lower value is

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

The probability \( p(\text{accept}) \) is:

- **Modulated through a temperature parameter \( T \):**
  - for \( T \to \infty \) ?
  - for \( T \to 0 \) ?

- **Cooling schedule:**
  - Schedule of changes of a parameter \( T \) over iteration steps
Simulated annealing algorithm

The probability of accepting a state with a lower value is

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

The probability is:

– **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

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
        schedule, a mapping from time to “temperature”
static: current, a node
        next, a node
        T, a “temperature” controlling the probability of downward steps

current ← MAKE-NODE(INITIAL-STATE(problem))
for i ← 1 to ∞ do
    T ← schedule[i]
    if T=0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] − VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability exp(ΔE/T)

Simulated annealing algorithm

• **Simulated annealing algorithm**
  - developed originally for modeling physical processes
    (Metropolis et al, 53)
  - Metal cooling and crystallization.
    Fast cooling → many faults → higher energy
  - Energy minimization (as opposed of maximization in the previous slides)

• **Properties of the simulated annealing methods**
  - If temperature T is decreased slowly enough the best configuration (state) is always reached

• **Applications:** (very large optimization problems)
  - VLSI design
  - airline scheduling