Methods for finding optimal configurations

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

Search for the optimal configuration

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

If the space of configurations we search among is
• Discrete or finite
  – then it is a combinatorial optimization problem
• Continuous
  – then it is 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

![Graph of cities A, B, C, D, E, F connected with edges]

**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**

![Chessboard with 3 queens and 0 queens]

# 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 !!!
Example: N-queens

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

![Diagram of N-queens problem]

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

![Diagram of TSP example]
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:

\[
\begin{align*}
\text{ABCDEF} \\
\downarrow \\
\text{ABCD} | \text{EF} | \\
\downarrow \\
\text{ABCDFE}
\end{align*}
\]

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

**Later:** Extensions to multiple current states:

- Genetic algorithms

**Note:** Maximization is inverse of the minimization

\[ \min f(X) \Leftrightarrow \max [-f(X)] \]

Hill climbing

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

![Diagram showing the hill climbing process](image-url)
Hill climbing

- Look at the local neighborhood and choose the one with the best value

- What can go wrong?

• 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
end

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).

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

where \( \Delta E = E_{\text{NEXT}} - E_{\text{CURRENT}} \)
\( T > 0 \)
• The probability is:
  – Proportional to the energy difference
Simulated annealing algorithm

Possible moves

Pick randomly from local neighbors

Current configuration

Energy $E = 167$

Energy $E = 145$

Energy $E = 180$

Energy $E = 191$

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

$= 145 - 167 = -22$

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

Sometimes accept!
Simulated annealing algorithm

- Random pick

**Current configuration**

- Energy $E = 167$
- Energy $E = 145$
- Energy $E = 180$
- Energy $E = 191$

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

$= 180 - 167 > 0$

$p(\text{Accept}) = 1$

Always 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 \( T \to \infty \)  \( T \to 0 \) 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 t ← 1 to ∞ do
    T ← schedule[t]
    if T=0 then return current
    next ← a randomly selected successor of current
    ∆E ← VALUE[next] - VALUE[current]
    if ∆E > 0 then current ← next
    else current ← next only with probability $e^{∆E/T}$

Simulated annealing algorithm

- **Simulated annealing algorithm**
  - developed originally for modeling physical processes (Metropolis et al, 53)
  - Metal cooling and crystallization.
    - Fast cooling $\rightarrow$ many faults $\rightarrow$ 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
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 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 candidate solutions generated from combination of previous configuration candidates

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 selected pairs
  - Mutation of resulting individuals
- Evolve the population over multiple generation cycles
Genetic algorithms

- Selection of configurations to be combined to generate the new population:
  - Fitness function = value of the objective 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