Constraint Satisfaction Problems: Local Search

Alice Gao

Lecture 8

Based on work by K. Leyton-Brown, K. Larson, and P. van Beek

Outline

Learning Goals

By the end of the lecture, you should be able to

- Describe/trace/implement the local search algorithms (hill climbing, hill climbing with random restarts, simulated annealing, and genetic algorithms).
- Describe strategies for escaping local optima.
- Compare and contrast the properties of local search algorithms.

Questions

The problem formulation:

- What is the neighbour relation?
- What is the cost function?

Executing the algorithm:

- Where do we start?
- Which neighbour do we move to?
- When do we stop?

Properties and performance of the algorithm:

- Given enough time, will the algorithm find the global optimum solution?
- How much memory does it require?
- How does the algorithm perform in practice?

Hill climbing

- Where do we start?
  Start with a random or good solution.
- Which neighbour do we move to?
  Move to a neighbour with the lowest cost. Break ties randomly. Greedy: does not look ahead beyond one step.
- When do we stop?
  Stop when no neighbour has a lower cost.
- How much memory does it require?
  Only need to remember the current node.
  No memory of where we've been.

Hill climbing in one sentence

Climbing Mount Everest in a thick fog with amnesia

CQ: Will hill climbing find the global optimum?

CQ: Will hill climbing find the global optimal solution given enough time?

(A) Yes. Given enough time, hill climbing will find the global optimal solution for every problem.

(B) No. There are problems where hill climbing will NOT find the global optimal solution.

Hill Climbing

Algorithm 1 Hill Climbing

```
1: current ← a random state
2: while true do
3:     next ← get-best-neighbour(current)
4:     if cost(current) ≤ cost(next) then
5:         break
6:     end if
7:     current ← next
8: end while
9: return current
```

Hill Climbing with Sideway Moves

Algorithm 2 Hill Climbing with Sideway Moves

```
 1: current ← a random state
 2: while true do
 3:     next ← get-best-neighbour(current)
 4:     if cost(current) < cost(next) then
 5:
 6:
 7:     end if
 8:     if cost(current) == cost(next) then
 9:
10:
11:     end if
12:
13:
14: end while
15: return current
```

Hill Climbing with Tabu List

- How do you keep track of the most recent nodes visited?
- How would you update the list?

Performance of hill climbing

- Perform quite well in practice.
- Makes rapid progress towards a solution.
  Easy to improve a bad state.

8-queens problem: $\approx$ 17 million states.

- Basic hill climbing

  % of instances solved: 14%
  # of steps until success/failure: 3-4 steps on average until success or failure.

- Basic hill climbing + $\leq$ 100 consecutive sideway moves:

  % of instances solved: 94%
  # of steps until success/failure: 21 steps until success and 64 steps until failure.

Dealing with local optima

Hill climbing can get stuck at a local optimum.
What can we do?

- Restart search in a different part of the state space.
  Hill climbing with random restarts
- Move to a state with a higher cost occasionally.
  Simulated annealing

Hill climbing with random restarts

If at first you don't succeed, try, try again.

Restart the search with a randomly generated initial state when

- we found a local optimum, and
- we've found a plateau and
  made too many consecutive sideway moves.
- we've made too many moves.

Choose the best solution out of all the local optima found.

CQ: Will hill climbing + random restarts find the global optimum?

CQ: Will hill climbing with random restarts find the global optimal solution given enough time?

(A) Yes.

(B) No. There are problems where hill climbing with random restarts will NOT find the global optimal solution.

Simulated Annealing

- Where do we start?
  Start with a random solution and a large T.
- Which neighbour do we move to?
  Choose a random neighbour.
  If the neighbour is better than current, move to the neighbour.
  If the neighbour is not better than the current state,
  move to the neighbour with probability $p = e^{\Delta E/T}$.
- When do we stop?
  Stop when $T = 0$.

Simulated Annealing

Algorithm 3 Simulated Annealing

```
1: current ← initial-state
2: T ← a large positive value
3: while T > 0 do
4:     next ← a random neighbour of current
5:     ΔE ← current.cost - next.cost
6:     if ΔE > 0 then
7:         current ← next
8:     else
9:         current ← next with probablity p = e^(ΔE/T)
10:    end if
11:    decrease T
12: end while
13: return current
```

CQ: How does T affect $p = e^{\Delta E/T}$?

CQ: Consider a neighbour with a higher cost than the current node ($\Delta E < 0$).

As T decreases, how does $p = e^{\Delta E/T}$ change?
($p = e^{\Delta E/T}$ is the probability of moving to this neighbour.)

(A) As T decreases, $p = e^{\Delta E/T}$ increases.

(B) As T decreases, $p = e^{\Delta E/T}$ decreases.

CQ: How does ΔE affect $p = e^{\Delta E/T}$?

CQ: Assume that T is fixed. Consider a neighbour where $\Delta E < 0$

As ΔE decreases (becomes more negative),
how does $p = e^{\Delta E/T}$ change?
($p = e^{\Delta E/T}$ is the probability of moving to this neighbour.)

(A) As ΔE decreases, $p = e^{\Delta E/T}$ increases.

(B) As ΔE decreases, $p = e^{\Delta E/T}$ decreases.

Annealing Schedule

How should we decrease T?

- Linear
- Logarithmic
- Exponential

If the temperature decreases slowly enough,
simulated annealing is guaranteed to find the global optimum with probability approaching 1.

Examples of Simulated Annealing

- Example: getting a tennis ball into the deepest hole.
- Exploration versus exploitation

Genetic algorithm

1. Keep track of a set of states. Each state has a fitness.
2. Randomly select two states to reproduce.
   The fitter a state, the most likely it's chosen to reproduce.
3. Two parent states crossover to produce a child state.
4. The child state mutates with a small independent probability.
5. Add the child state to the new population.
6. Repeat steps 2 to 5 until we produce a new population.
   Replace the old population with the new one.
7. Repeat until one state in the population has high enough fitness.

Backtracking with Inferences and Heuristics

Algorithm 4 Generic Algorithm

1: $i = 0$
2: create initial population $pop(i) = \{X_1, ..., X_n\}$
3: **while** true **do**
4: **if** $\exists x \in pop(i)$ with high enough $f(x)$ **then**
5: break
6: **end if**
7: for each $X_i \in pop(i)$ calculate $pr(X_i) = f(X_i) / \sum_i f(X_i)$
8: **for** j from 1 to n **do**
9: choose a randomly based on $pr(X_i)$
10: choose b randomly based on $pr(X_i)$
11: child $\leftarrow$ crossover(a, b)
12: child mutates with small probability
13: add child to $pop(i+1)$
14: **end for**
15: $i = i + 1$
16: **end while**
17: **return** $x \in pop(i)$ with highest fitness

Revisiting the Learning Goals

By the end of the lecture, you should be able to

- Describe/trace/implement the local search algorithms (hill climbing, hill climbing with random restarts, simulated annealing, and genetic algorithms).
- Describe strategies for escaping local optima.
- Compare and contrast the properties of local search algorithms.