Introduction to Metaheuristics - Local Search

DrC. Alejandro Piad Morffis

Review: Why optimization is hard

Review: Why optimization is hard

- Most interesting problems are NP-Hard
- Weak global structure.
- Disparate attraction basins.
- Most real-life functions are black-box.
- Function evaluation can be very costly.

Review: What can we do

Just search, but with common sense!

- **Assume there is some local structure**: Near good solutions we can find other good solutions.
- **Approximate gradients:** Follow steps that improve the function fitness.
- **Avoid local optima:** Take measures to prevent getting stuck.

Any search method must balance between:

- **exploration** steps that lead you to discover new (and potentially better) attraction basins, and
- **exploitation** steps that lead you to improve your estimate of the best attraction basin.

Back to the basics

What is the simplest search method beyond random search?

Back to the basics

What is the simplest search method beyond random search?

Hill-climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else $x_{i+1} = x_i$
- return x_n

Back to the basics

What is the simplest search method beyond random search?

Hill-climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else $x_{i+1} = x_i$
- return x_n

The devil is in the details

- When do we know we're done?
- What is *close enough*?

Analyzing hill climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else $x_{i+1} = x_i$
- return x_n

How likely is to get stuck?

Analyzing hill climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else $x_{i+1} = x_i$
- return x_n

How likely is to get stuck?

Theorem: For any *fixed* value of closeness, hill climbing will eventually get stuck. (Why?)

Analyzing hill climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else $x_{i+1} = x_i$
- return x_n

How to avoid geting stuck?

Fixing hill climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else if $U(0,1) < \beta$ then $x_{i+1} = x_j$ **<– here**
  - else $x_{i+1} = x_i$
- return x^* *(best ever seen)* **<– and here**

How to avoid geting stuck?

Sometimes, take non-exploitative steps.

What is the best value for β

Fixing hill climbing

- start with a random solution $x_i = x_0$
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else if $U(0, 1) < \beta$ then $x_{i+1} = x_j$ **<– here**
  - else $x_{i+1} = x_i$
- return x^* *(best ever seen)* **<– and here**

How to avoid geting stuck?

Sometimes, take non-exploitative steps.

What is the best value for β

A changing value...

Fixing hill climbing = Simulated annealing

- start with a random solution $x_i = x_0$
- set $\beta = \beta_0$ relatively high
- while not finished:
    - sample a random solution x_j *close to* x_i
    - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
    - else if $U(0,1) < \beta$ then $x_{i+1} = x_j$
    - else $x_{i+1} = x_i$
    - decrease β *a little bit* <– **here**
- return x^*

Ways to decrease β:

Fixing hill climbing = Simulated annealing

- start with a random solution $x_i = x_0$
- set $\beta = \beta_0$ relatively high
- while not finished:
  - sample a random solution x_j *close to* x_i
  - if $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else if $U(0, 1) < \beta$ then $x_{i+1} = x_j$
  - else $x_{i+1} = x_i$
  - decrease β *a little bit* <– **here**
- return x^*

Ways to decrease β:

- Linearly
- Exponentially
- Only if no improvement
- ...

Simulated annealing

How good is this?

- If the probability of sampling any solution x_j from the current x_i is never zero
- And the "cooling schedule" is *sufficiently slow*
- Then SA converges to a global optima *eventually*!

Simulated annealing

How good is this?

- If the probability of sampling any solution x_j from the current x_i is never zero
- And the "cooling schedule" is *sufficiently slow*
- Then SA converges to a global optima *eventually*!

Improvements

- Decrease the size of the vicinity iteratively
- Make the probability of choosing a solution x_j proportional to how good it is compared to x_i
- Restart β after some iterations stuck

Quick detour: beyond single-solution

One key problem with HC and SA is that you only at one solution at a time.

What if we look at several solutions and take the best one?

- start with a random solution $x_i = x_0$
- while not finished:
  - sample N random solutions X_j *close to* x_i
  - if any $x_j \in X_j$ is such that $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else break
- return x^*

Avoiding past mistakes

What are we trying to achieve with simulated annealing?

Avoiding past mistakes

What are we trying to achieve with simulated annealing?

Avoid local optima, by accepting bad solutions.

But who can say that we won't circle back to the same local optima?

Avoiding past mistakes

What are we trying to achieve with simulated annealing?

Avoid local optima, by accepting bad solutions.

But who can say that we won't circle back to the same local optima?

Enter **Tabu-search:** avoiding past mistakes.

Tabu search

Key idea: Store good solutions to avoid circling back

- start with a random solution $x_i = x_0$
- initialize memory M
- while not finished:
  - sample N random solutions X_j not in M.
  - if any $x_j \in X_j$ is such that $F(x_j) > F(x_i)$ then $x_{i+1} = x_j$
  - else accept a random x_j solution
  - add x_i to M
- return x^*

Tabu search extensions

What can we put in the memory?

Tabu search extensions

What can we put in the memory?

- Exact solutions
- Components of solutions
- Steps that lead to good solutions

How long should information stay in memory?

Tabu search extensions

What can we put in the memory?

- Exact solutions
- Components of solutions
- Steps that lead to good solutions

How long should information stay in memory?

- Short-term memory for exact solutions
- Long-term memory for more abstract patterns
- Any level of granularity is possible

Tabu search extensions

What can we put in the memory?

- Exact solutions
- Components of solutions
- Steps that lead to good solutions

How long should information stay in memory?

- Short-term memory for exact solutions
- Long-term memory for more abstract patterns
- Any level of granularity is possible

Flexibilize the meaning of tabu features

Instead of forbididng solutions, penalize them according to how many tabu features they contain.

Summarizing

Key ideas in local search:

- Estimate gradients sampling from a neighborhood function
- Select best solutions to exploit gradients
- Accept random bad solutions to explore

How to avoid local optima:

- Accept bad solutions
- Forbid or penalize seen solutions
- Restart frequently

How to balance exploration/exploitation:

- Via a tradeoff parameter (e.g., probability to select bad solutions)

Time to practice