Local Search:
Hill Climbing, Simulated Annealing, and Tabu Search

Vincent A. Cicirello, Ph.D.
Professor of Computer Science
cicirelv@stockton.edu https://www.cicirello.org/

1

Lesson 1

SEARCH FOR OPTIMIZATION PROBLEMS

2

Optimization Problems

- Optimization Problem:
  - A finite, discrete set, S.
    - Some examples:
      - set of all permutations of N elements,
      - set of all graphs with N nodes,
      - set of all set partitions of a set of N elements,
      - set of all possible variable assignments to a set of N variables, etc
  - A value function, $V : S \rightarrow \mathbb{R}$
  - Search Problem (2 variations):
    1. Find the $s \in S$ such that $s = \operatorname{argmax}_{s'} V(s')$
    2. Find the $s \in S$ such that $s = \operatorname{argmin}_{s'} V(s')$

3

An example: Boolean Satisfiability

- You have a set of boolean variables (i.e., variables with domain { true, false }.
- You have a boolean expression in Conjunctive Normal Form, which may or may not be satisfiable.
- Problem: Find an assignment of truth values to variables to maximize number of satisfied clauses.

$$(A \vee B \vee C) \wedge (B \vee C \vee \neg D) \wedge (D \vee E \vee F) \wedge (\neg D \vee \neg E \vee \neg F) \wedge$$
$$(\neg A \vee \neg C \vee \neg F) \wedge (E \vee F \vee \neg G) \wedge (\neg A \vee G \vee \neg H) \wedge$$
$$(D \vee \neg E \vee H) \wedge (B \vee I \vee \neg J) \wedge (\neg H \vee I \vee J) \wedge (G \vee \neg I \vee \neg J)$$

4

Constraint Satisfaction vs Constraint Optimization

- Wait! Isn't Boolean Satisfiability a constraint satisfaction problem?
- Shouldn't we be using the search techniques we saw for constraint satisfaction?
- **Constraint Satisfaction:**
  - Find an assignment of values to variables that satisfy all of the constraints.
- **Constraint Optimization:**
  - Find the assignment of values to variables that satisfy as many constraints as you can.
  - The DFS search techniques we saw previously will be too expensive.

5

Constraint Satisfaction vs Constraint Optimization

- Over-constrained Problems:
  - An over-constrained problem is a constraint satisfaction problem in which no solution exists.
  - In other words, it is impossible to satisfy all constraints simultaneously
- Constraint Satisfaction search algorithms in this case will:
  1. Exhaustively search (via DFS) all possible value assignments (even with all of our tricks, constraint propagation, variable/value ordering heuristics), and then
  2. Simply indicate "no solution."

6

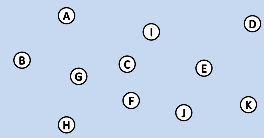
Constraint Satisfaction vs Constraint Optimization

- Often need better than "no solution" as result for over-constrained problem.
- Example:
  - Scheduling courses to classrooms.
  - Each class requires certain room characteristics (e.g., computer lab, chalk board vs whiteboards, etc).
  - Each class has other constraints (number of students, time of day, days of the week).
  - Very near room capacity on Stockton's Galloway campus.
  - Room scheduler used by registrar always fails to find rooms for some classes (over-constrained problem).
  - "No solution" would be an unacceptable outcome.
  - It allocates rooms to as many courses as possible, leaving people to figure out the rest (e.g., changing times, changing constraints, etc)

7

Another example: Traveling salesperson

- Traveling salesperson problem (TSP):
  - Set of cities.
  - Problem: Find the "tour" of the cities with minimum overall length.
  - A "tour" is a simple cycle consisting of **all** of the cities.
  - A simple cycle begins and ends at the same city, but otherwise contains no duplicates.



STOCKTON
STOCKTON UNIVERSITY www.stockton.edu

8

Traveling Salesperson Problem

- May sound like a purely theoretical problem with a very old-fashioned application (e.g. salespeople don't usually travel around selling anymore).
- TSP has many important real-world applications.
- Package delivery is an example:
  - Instead of cities, you have delivery addresses within a portion of a city, and the address of the truck depot.
  - You want the delivery driver to visit each address exactly once, returning to the depot, while minimizing distance or fuel, etc.

STOCKTON
STOCKTON UNIVERSITY www.stockton.edu

9

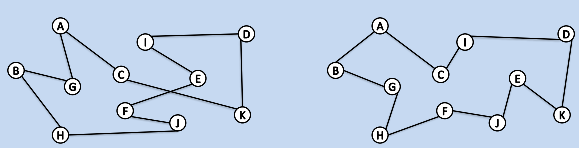
Traveling Salesperson Problem

- Wait! Isn't this just a shortest path in a graph problem, like single-source shortest path, or maybe all pairs shortest paths?
- No! The TSP is much harder.
  - We're not simply looking for shortest paths.
  - We're looking for the shortest cycle that visits **all** of the cities (all of the vertices of the graph).
  - This is very different than those other shortest path problems.
  - Algorithms like Dijkstra's for single-source shortest path, or Floyd-Warshall for all-pairs shortest paths, won't help us here.
  - A* also won't help us here.

STOCKTON
STOCKTON UNIVERSITY www.stockton.edu

10

TSP Tour Examples



STOCKTON
STOCKTON UNIVERSITY www.stockton.edu

11

Informal Problem Characteristics

- Some structure to optimize.
- Value function to either minimize or maximize.
- Searching all possible configurations of the structure is intractable (e.g., TSP with n cities has n! possible configurations)
- DFS is too expensive (e.g., runtime for DFS for TSP is O(n!)).
- No known algorithm for efficiently finding optimal.
  - Best known exact algorithms have exponential runtime or worse.
  - For example, the problem is NP Hard.
- Similar solutions have similar costs.
  - For example, if 2 tours for a TSP have n-2 edges in common and differ only by 2 edges, the cost of the tours will be similar.

STOCKTON
STOCKTON UNIVERSITY www.stockton.edu

12

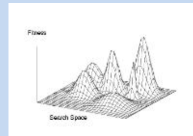
Lesson 2

LOCAL SEARCH AND HILL CLIMBING

13

Fitness Landscape

- Fitness Landscape:
  - Imagine all possible configurations located on the surface of a landscape. The altitude of a point on the landscape corresponds to the corresponding configuration's "fitness" (i.e., value of the optimization function).
  - Similar configurations near each other on landscape.
    - Two TSP tours that differ by only 2 edges would be next to each other on landscape.
  - Term originated in biology within context of genetics.



14

Local Search

- Local Search is sometimes called Iterative Improvement.
- Basic structure of a Local Search:
  1. Start at a random initial configuration.
  2. While termination criteria not met:
     a) Select a "neighbor".
     b) Decide whether or not to accept that neighbor.
  3. (Some local search algorithms) Restart at step 1.
- Requires a "neighborhood function" (sometimes called "moveset operator").
- We'll look at several algorithms that fit this pattern.

15

Steepest Ascent/Descent Hill Climbing

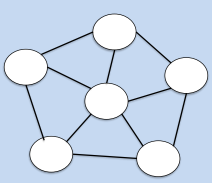
```
X := random initial configuration
Terminate := false
While Not Terminate Do
    Terminate := true
    BestNeighbor := X
    For each neighbor X' of X do
        If (V(X') < V(BestNeighbor)) then
            Terminate = false
            BestNeighbor := X'
    If (Not Terminate) then
        X := BestNeighbor
Return X
```

Pseudocode written assuming minimization.

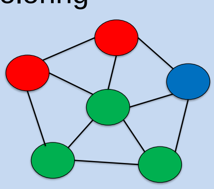
For maximization, change This < to >.

16

Example: Graph Coloring



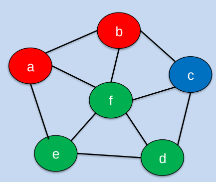
- **Minimize # constraint conflicts.**
- First step: generate a random initial configuration, i.e., a random color assignment for the variables.
- Compute # constraint conflicts.



V = 4 constraint conflicts

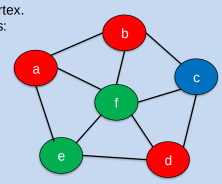
17

Example: Graph Coloring



V = 4 constraint conflicts

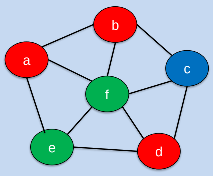
- Neighbor function: Change color of 1 vertex.
- Iterate over neighbors:
  - a to green, V=5
  - a to blue, V=3
  - b to green, V=4
  - b to blue, V=4
  - c to red, V=5
  - c to green, V=6
  - **d to red, V=2**
  - d to blue, V=3
  - e to red, V=3
  - e to blue, V=2
  - f to red, V=4
  - f to blue, V=3



V = 2 constraint conflicts

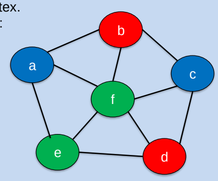
18

Example: Graph Coloring



V = 2 constraint conflicts

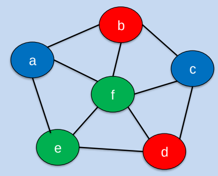
- Neighbor function:
  Change color of 1 vertex.
- Iterate over neighbors:
  - a to green, V=3
  - **a to blue, V=1**
  - b to green, V=2
  - b to blue, V=2
  - c to red, V=4
  - c to green, V=3
  - d to green, V=4
  - d to blue, V=3
  - e to red, V=3
  - e to blue, V=1
  - f to red, V=4
  - f to blue, V=2



V = 1 constraint conflicts

19

Example: Graph Coloring



V = 1 constraint conflicts

- Neighbor function:
  Change color of 1 vertex.
- Iterate over neighbors:
  - a to red, V=2
  - a to green, V=3
  - b to green, V=2
  - b to blue, V=3
  - c to red, V=3
  - c to green, V=2
  - d to green, V=3
  - d to blue, V=2
  - e to red, V=1
  - e to blue, V=1
  - f to red, V=2
  - f to blue, V=2

We are at a local optima so search is finished.

None of the neighbors are better than the current solution.

20

First Ascent/Descent Hill Climbing

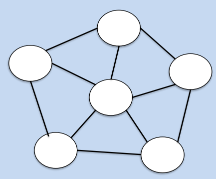
```
X := random initial configuration
Terminate := false
While Not Terminate Do
    Terminate := true
    For each neighbor X' of X do
        If (V(X') < V(X)) then
            Terminate = false
            X := X'
            Break out of inner loop
Return X
```

Pseudocode written assuming minimization.

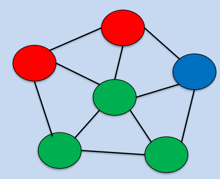
For maximization, change This < to >.

21

Example: Graph Coloring



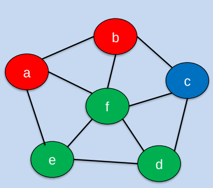
- **Minimize # constraint conflicts.**
- First step: generate a random initial configuration, i.e., a random color assignment for the variables.
- Compute # constraint conflicts.



V = 4 constraint conflicts

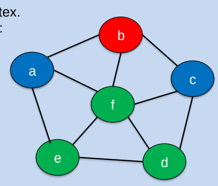
22

Example: Graph Coloring



V = 4 constraint conflicts

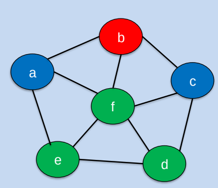
- Neighbor function:
  Change color of 1 vertex.
- Iterate over neighbors:
  - a to green, V=5
  - **a to blue, V=3**



V = 3 constraint conflicts

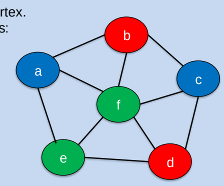
23

Example: Graph Coloring



V = 3 constraint conflicts

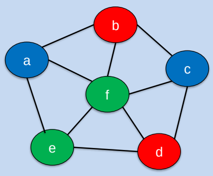
- Neighbor function:
  Change color of 1 vertex.
- Iterate over neighbors:
  - a to red, V=4
  - a to green, V=5
  - b to green, V=4
  - b to blue, V=5
  - c to red, V=4
  - c to green, V=5
  - **d to red, V=1**



V = 1 constraint conflicts

24

Example: Graph Coloring



V = 1 constraint conflicts

- Neighbor function:
  Change color of 1 vertex.
- Iterate over neighbors:
  - a to red, V=2
  - a to green, V=3
  - b to green, V=2
  - b to blue, V=3
  - c to red, V=3
  - c to green, V=2
  - d to green, V=3
  - d to blue, V=2
  - e to red, V=1
  - e to blue, V=1
  - f to red, V=2
  - f to blue, V=2

We are at a local optima so search is finished.

None of the neighbors are better than the current solution.

25

Randomized Hill Climbing

```
X := random initial configuration
While Termination Criteria Not Met do
    X' := randomly selected neighbor of X
    If (V(X') < V(X)) then
        X := X'
Return X
```

Pseudocode written assuming minimization.

For maximization, change This < to >.

- When do you terminate? Here are a couple options:
  - Some maximum number of iterations of the loop.
  - Or perhaps some maximum number of iterations without accepting a neighbor.

26

Restarting a Hill Climber

- **Local optima:** A solution that is at least as good as all of its neighbors.
  - Locally, you cannot do any better.
- **Global optima:** A solution that is at least as good as all other solutions to the problem.
- Hill climbers find local optima, and you have no way of knowing whether or not it is the global optima.
- Common practice: Restart the hill climber multiple times.
  - Each restart finds a local optima.
  - Return the best of those local optima.

```
RestartHillClimber(P: the problem)
    X := HillClimber(P)
    for i = 1 to MAX_RESTARTS do
        X' := HillClimber(P)
        if (V(X') < V(X)) then
            X := X'
    return X
```

Where `HillClimber` refers to steepest descent, or first descent, or randomized, or any other form of hill climbing.

27

Hill Climbing Issues

- Memory: O(1)
  - Only constant memory since you only ever have two configurations in memory at any time (the current solution and one neighbor).
- Neighborhood Function
  - Critical part of application of algorithm to a problem
  - Need to determine what a neighbor is.
  - Too small a neighborhood → easy to get stuck in local optima
  - Too large a neighborhood → inefficient (loop over all neighbors)
- Tends to get stuck in local optima
  - Common Strategy: Restart when local optima reached

28

Lesson 3

LOCAL SEARCH FOR BOOLEAN SATISFIABILITY

29

Boolean Satisfiability as an Optimization Problem

- Given a Boolean expression in 3-CNF format, find an assignment of Boolean values (true/false) for the variables to maximize the number of satisfied clauses.
- Conjunctive Normal Form (CNF): Boolean expression is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of Boolean literals (a variable or its negation).

$$(A \vee B \vee C) \wedge (B \vee C \vee \neg D) \wedge (D \vee E \vee F) \wedge (\neg D \vee \neg E \vee \neg F) \wedge$$
$$(\neg A \vee \neg C \vee \neg F) \wedge (E \vee F \vee \neg G) \wedge (\neg A \vee G \vee \neg H) \wedge$$
$$(D \vee \neg E \vee H) \wedge (B \vee I \vee \neg J) \wedge (\neg H \vee I \vee J) \wedge (G \vee \neg I \vee \neg J)$$

30

GSAT

- GSAT is a classic hill climbing example for Boolean Satisfiability.
  - Steepest Ascent Hill Climber
  - Neighborhood function: flip the value of any one variable
    - Flip simply means change from true to false (or false to true).
  - The G in GSAT means Greedy.

31

Walksat

- Walksat is another classic local search for Boolean Satisfiability.
- It isn't really a hill-climber, in that it sometimes chooses a neighbor that is worse than the current solution (on purpose).
- Neighborhood function:
  1. Pick a random unsatisfied clause. This clause has 3 variables since we are assuming 3-SAT.
  2. Consider 3 neighbors, flipping the value of exactly one of those 3 variables.
- Logic for determining which neighbor to choose:
  1. If any of the 3 neighbors is better than current solution, greedily pick the best of the 3 (just like a steepest ascent hill-climber).
  2. Generate a random number $r \epsilon [0.0, 1.0)$.
     If $r < 0.5$, choose the least bad neighbor, otherwise choose a random neighbor.

32

Stepping through GSAT and Walksat

- We'll now step through examples of GSAT and Walksat.

33

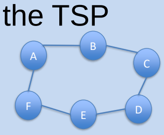
Lesson 4

LOCAL SEARCH FOR TRAVELING SALESPERSON PROBLEM (TSP)

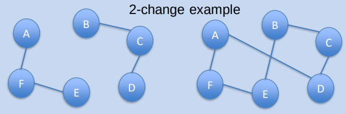
34

Hill Climbing for the TSP

- The Traveling Salesperson Problem
  - Find the tour (cycle which visits each exactly once) of a set of N cities that minimizes total distance.
- Configuration:
  - Permutation of the N cities
- Neighborhood function:
  - 2-change, ..., k-change

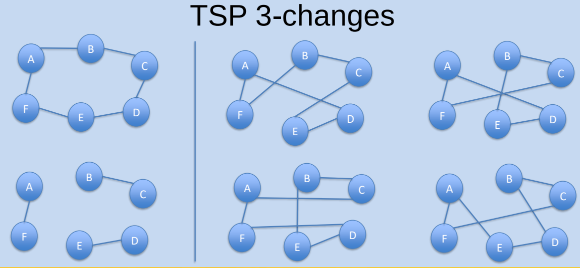


2-change example



35

TSP 3-changes



36

Classic Example for TSP: k-Opt

- kOpt is a Steepest Descent Hill Climber
- Neighborhood function (for some k):
  - The set of all 2-changes, 3-changes, …, k-changes.
  - 2-Opt neighborhood is all 2-changes, for 3-Opt, it is all 2-changes and 3-changes.
- Some (classic) Findings (Lin):
  - 3-Opt Solutions are generally much better than 2-Opt solutions.
  - 4-Opt solutions, though better than 3-Opt, are not sufficiently better to justify extra computational cost.
  - For a particular class of TSP, Lin showed that the probability that 3-Opt finds the optimal solution is: $0.5^{n/10}$ where n is number of cities.
  - Can restart R times and estimate probability of having found optimal solution:

    $1 - (1 - 0.5^{\frac{n}{10}})^R$

37

Lesson 5

ESCAPING FROM LOCAL OPTIMA

38

Local Optima

- Local Optima (definition):
  - A configuration that is at least as good as all immediate neighbors.
- Global Optima (definition):
  - A configuration that is at least as good as all possible configurations.
- Hill climbing continues until stuck:
  - We may or may not be at a global optima
  - We have no way of knowing
- Approaches to dealing with local optima:
  - Restarting (see earlier)
  - Accepting worsening neighbors sometimes (e.g., Walksat)

39

Simple Attempt at Dealing with Local Optima

```
X := random initial configuration
BestFound := X
While Termination Criteria Not Met do
    X' := randomly selected neighbor of X
    If (V(X') < V(X)) then
        X := X'
        if (V(X') < V(BestFound)) then
            BestFound := X'
    Else
        r := random real in [0.0, 1.0)
        If r < P then
            X := X'
Return BestFound
```

Pseudocode written assuming minimization.

For maximization, change This < to >.

Basic Idea: By allowing some bad moves sometimes, might wander away from the local optima into a more promising area.

40

How should the probability P be set?

- Perhaps some constant like 0.1, or 0.5, etc?
- Perhaps probability that decreases over time?
  - Start P = 1 (i.e., accept all moves even bad ones)
  - Decrease it during search (i.e., the longer the search the less worsening moves are accepted) until P = 0.
  - When P=0, becomes a randomized hill climber by end of search.
- Perhaps probability that decreases with the "badness" of the neighbor?
  - The worse the move, the lower P is (lower probability of accepting)
  - The less bad the move, the higher P is (higher probability of accepting)
- Perhaps a combination of the above?

41

Simulated Annealing

- If random neighbor is an improvement (or of same quality), then definitely accept it.
- If random neighbor is worse than current, then accept it with the following probability:

  $e^{(V(x)-V(x'))/T_i}$
- Known as the Boltzmann distribution
- Expressed above if minimizing, if maximizing, we use:

  $e^{(V(x')-V(x))/T_i}$
- T is a "temperature" parameter that is "cooled" over time
- Common "cooling schedule", for cooling rate, 0 <= a < 1, and T_0 some large initial value:

  $T_i = a * T_{i-1}$
- High temp (accept all moves), Low temp (randomized hill climbing)
- Terminate after pre-specified number of steps

42

Simulated Annealing

```
X := random initial configuration
BestFound := X
T := T0
for i = 1 to MAX_ITERATIONS do
    X' := randomly selected neighbor of X
    If (V(X') <= V(X)) then
        X := X'
        if (V(X') < V(BestFound)) then
            BestFound := X'
    Else
        r := random real in [0.0, 1.0)
        If r < e^((V(x)-V(x'))/T) then
            X := X'
    T := a * T
Return BestFound
```

Pseudocode written assuming minimization.

Parameters: a and T_0 are parameters, where 0.0 < a < 1.0.

a is usually 0.9 < a < 0.999

T_0 should be "high" but what this means depends on the range of the V function.

43

More on Simulated Annealing

- Simulated Annealing introduced in 1953 by Metropolis.
  - Yes, really that long ago.
- Based on analogy to how alloys manage to find globally minimal energy level if cooled slowly.
- Much better (empirically) than hill climbing at avoiding local optima.
- Weird, provable, but not entirely useful, fact:
  - With an infinitely slow cooling rate, you'll find the global optima.
- More elaborate cooling schedules exist, including one that is "optimal" given a pre-determined search length
  - E.g., Modified Lam Schedule (Justin Boyan)
  - E.g., Lam Schedule (Lam and Delosme)

44

Tabu Search

- Among the definitions of Tabu is "forbidden."
- Tabu Search uses a "tabu" list of configurations to try to guide the search into considering things it hasn't seen before.
- "Tabu List": Tabu search never accepts a neighbor if it is in the tabu list (unless all neighbors are Tabu).
- Tabu List has a finite, predetermined max length.
- When a neighbor is accepted, it's added to the Tabu List (removing the oldest thing from the Tabu List if it is full).
- The Tabu List is sort of like a Queue (first in first out).
- Note: There are more elaborate versions of Tabu Search.

45

Tabu Search

```
x := random initial configuration
BestFound := x
T := a Queue initially containing only X
for i = 1 to MAX_ITERATIONS do
    If (Neighbors(x) - T ≠ Ø) then
        x := argmin_{x' ϵ Neighbors(x)-T} V(x')
        If V(x) < V(BestFound)
            BestFound := x
    Else
        x := random neighbor of x
    Add x to tail of T
    if length(T) > limit then
        remove head of T
Return BestFound
```

Pseudocode written assuming minimization.

46