Hill-climbing

Optimization strategy:
- Start with any solution that meets the constraints.
- Repeat:
  - Try to improve the solution via a “local” change, still satisfying the constraints.
  - When there’s no simple way to improve the solution: output it.

Like greedy algorithms:
- More often than not hill-climbing does NOT find an optimal solution, just a “local optimum”
- Often used as an approximation algorithm or heuristic.

Also called “gradient ascent”, “interior point method”
Local optima

- One can view the set of all possible solutions as a high-dimensional region. The objective function then gives a height for each point.

- Want to find the highest point.

- But we usually find a local optima, a point higher than others near it.

- So while global optima are local optima, the reverse is not always true.

Network flow problem

How much can you send from s to t, subject to edge capacities?
For each node on the way: flow in = flow out
Network flow problem

Network flow

- Instance: Directed graph $G = (V,E)$ with non-negative edge capacities $c(e)$; vertices $s,t$
- Solution format: An assignment of non-negative values $f(e)$ to each edge. This is the “flow”.
- Constraints: $f(e) \leq c(e)$ for all edges $e$. At any vertex except $s$ and $t$, total flow in = total flow out.
- Objective: maximize total flow out of $s$ = total flow into $t$
Why is network flow so important?

- Comes up directly: traffic, network routing

- Duality: max flow = min cut. Often what we are interested in is min cut

- Expressive power: As we’ll see, we can reduce many problems such as bipartite matching to network flow. Many of these problems don’t seem to have anything to do with networks, graphs, or flows.

Ford Fulkerson method

- Hill-climbing: start with trivial solution

- Keep on trying to make it better.

- In this case, we’ll be able to represent “can we make it better?” as “Is there any flow in a residual network?”
Network flow problem

• What is a flow that we can start with?

Hill climbing for network flow

• Trivial flow: zero flow along every edge
• Better than zero: Any path of non-zero edges from s to t
• How much better?: We can send the minimum capacity of an edge in the path along that path.
Ford-Fulkerson insight: We can represent the problem of improving a flow as another flow problem, for the residual graph.
Residual flow graph

- Ford-Fulkerson insight: We can represent the problem of improving a flow as another flow problem, for the residual graph
- If \( f(e) \) is flow on edge \( e \) and \( c(e) \) is capacity of edge \( e \) then change the capacity to \( c(e) - f(e) \) and add \( f(e) \) to the capacity of the reverse edge.

Network flow problem
Example: residual network

Key observation (Additivity)

• Let $f$ be any flow in the original.

• Let $f'$ be any flow in the residual network with respect to $f$

• Then $f+f'$ is a flow in the original network, and

  \[ \text{Flow}(f+f') = \text{Flow}(f) + \text{Flow}(f'). \]

• So we can keep on finding flow in the residual network, update the residual network, until we cannot increase it any more.
Flow so far

Network flow problem
Network flow problem

Flow so far
Network flow problem

Network flow problem
Flow so far

Network flow problem
Network flow problem

Flow so far
Network flow problem

Ford Fulkerson method

- Start with zero flow $F$.
- While there is a path from $s$ to $t$ in residual network $G_F$
  - Find such a path (many possibilities)
  - $w = \text{min residual capacity of edge in path}$
  - Increase flow by $w$ along path
  - Update $G_F$ (decrease residual capacity forwards along path, increase backwards)
- Return $F$. 
FF terminates (and time bound)

- Say capacities are all integers, at most $W$.
- Flow can be at most the total capacity of edges leaving $s$, so at most $nW$.
- Each iteration, we increase flow by 1.
- So total number of iterations is at most $nW$.
- Total time is: $O(nW \times \text{time to find path})$

Correctness: why is FF flow optimal?

- ``Achieve the bound” argument
- Bound = Total capacity of edges in some cut $(S, T)$
Why did we stop?

• End when there is no path from s to t in the residual graph.
• Let S be the set of all vertices reachable from s in the residual graph. Then: t is not in S.
• Let T = V-S, the unreachable vertices. Then all edges e=(u,v) with u in S and v in T are not in the residual graph. Therefore, they are all being used to full capacity.
• Let Cut(S,T)= total capacity of all such edges. Then: Flow(f)=Cut(S,T).

Duality: Max Flow= Min Cut

• Lemma 1: For any flow, and any cut, the flow across the cut = total flow

Corollary: For any flow and any cut, the total flow is at most the sum of the capacities of edges in the cut.
Optimality

- Let $(S,T)$ be cut found by FF and F flow found by FF.
- Let $F'$ be some other flow, $(S',T')$ some other cut.
- We have $\text{TotalFlow}(F) = \text{total capacity of edges in } (S,T)$
- By the Corollary, $\text{TotalFlow}(F') \leq \text{TotalCapacity}(S,T) = \text{TotalFlow}(F) \leq \text{TotalCapacity}(S',T')$
- Therefore: $F$ is max flow.
- AND $(S,T)$ is the min total capacity cut
- AND Max-Flow=Min-Cut

Is time bound tight?

- At most $Wn$ iterations
- There exist networks, ways of picking path, with $W$ iterations.
- If $W$ is large, this can be very slow.
- Which path should we pick to speed it up the most?
Pick path using max Bandwidth path

- $O(|E| \log |V|)$ time per iteration.
- But can show each iteration gets at least a $1/|E|$ fraction of remaining possible flow.

- Total number of iterations at most $O(|E| \log (W|V|))$.
- So total time $O(|E|^2 \log |V| (\log|V|+\log W))$.

Perfect Bipartite matching
Perfect Bipartite matching

- Instance: A bipartite graph $G=(L,R,E)$, $|L|=|R|=n$
- Solution: A set of $n$ edges $M$
- Constraints: Every vertex is in exactly one edge from $M$, called perfect matching in $G$

- Problem: Does $G$ have a perfect matching?
Reducing perfect matching to max flow

- Create a network flow with two additional vertices, s, and t, as well as all vertices in L and all vertices in R
- Think of edges between L and R as being directed from L to R
- Add edges from s to every vertex in L, and from every vertex in R to t
- Give all edges capacity 1 (in the forward direction, s to L to R to t)

Why equivalent

- If there’s a perfect matching, send 1 flow from s to each u in L, to its matched vertex v in R, and then to t
- Total flow = n

- If there’s a flow of value n, must send 1 flow from s to each u in L.
- That flow must cross one edge to some v in R.
- Since only one flow possible out of v, can’t use the same v for different u’s. So the used edges between L and R are a perfect matching.
Many other applications

- Network flow has ``high expressive power’’,
- Many other problems, including e.g., scheduling problems with no explicitly given graph, reduce to network flow
- Wide variety of algorithms, substantially improve over FF

Linear programming

- Linear programming is a very expressive optimization problem.
- The constraints and objective function are all linear equations or inequalities in $n$ variables; the problem is to find a setting of the variables that meets the constrains and maximizes the objective
- Essentially, the constraints limit the solution space to a multi-dimensional polyhedron.
- Then since the objective function is linear as well, there are no local optima so the global optimum occurs at a corner of the polyhedron.
- Linear programming is solvable in polynomial time using the ellipsoid algorithm, Karmarkar’s algorithm, or other approaches