General

Prim → O(E + VLog(V))

Kruskal → O(Vlog(E))

Dijkstra → E + VlogV → cannot do negative weight cycles

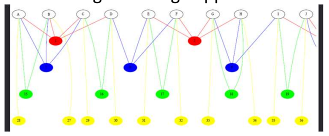
Bellman Ford → EV → can detect negative weight cycles

Strongly Connected Component → Tarjan's SCC algo → O(V+E)

- If every DFS can reach back itself, the cycle is a SCC
- Used to solve 2-CNF-SAT → if a & !a in same SCC, means contradiction, not solvable
- Because each clause implies !b -> a or !a -> b

Connectivity → BFS → V+E

MVC

- Dual of MIS
- Brute force → 2^n * m
  - 2^n combinations * checking m edges for each
- On Bipartite → matching → konig use MCBM
- On tree → DP → O(n)
  - Start from leaves, min(
    sum of in(c)),
    1+ sum of min(in (c) , out(c))
)
- Small k → size of the cover
- For edge → pick 1 vertex to keep → dp on the rest of graph → recursion depth constrained by k
- General → NP-hard
- Deterministic 2OPT
  - Randomly pick edge → take both vertices → 2 OPT
  - Greedy pick edge → take the vertex with most degree → logn approx.
    - Worst case when tie breaking by id → 
    - Best is top row letters m but end up with lower numbers when top row always lose tiebreak
- Randomized Approximation 2OPT
  - For each edge, randomly pick one vertex
  - Then remove all edges connected to the selected vertex
- MWVC 2OPT → Bar-Yehuda and Even's algo
  - For each edge, decrease both sides by smaller value →anything that is 0 is in the MWVC
- MVC > Matching → triangle → MVC = 2, matching = 1

LP

Rounded answer <= 2x OPT(LP)

Min/Max some objective func

Constraint int, other constraints

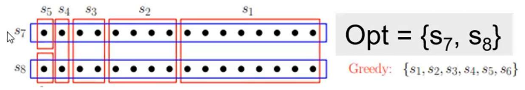
$$
\begin{array}{rcl}
\max \sum_{j:(s,j)\in E} f_{sj} & \text{where} & \text{// maximize flow that goes out from source vertex } s \\
\forall i,j \in E, f_{ij} \geq & 0 & \text{// } f_{ij} \text{ is the (integer) flow in edge } (i,j) \\
\forall i,j \in E, f_{ij} \leq & c_{ij} & \text{// capacity constraints} \\
\forall j \in V\backslash\{s,t\}, \sum_{i:(i,j)\in E} f_{ij} = & \sum_{k:(j,k)\in E} f_{jk} & \text{// flow conservation constraints}
\end{array}
$$

Set Cover

DP → bitmask pick don't pick fork → return recursion when all covered

- O (2^k) where k is number of sets

Greedy Set Cover

- Pick the group which covers the largest number of uncovered elements
- Ln (n) + O(1) OPT when each chosen set covers ½ of the remaining pts
- 

Steiner Tree

Min-Spanning-Tree but we can add more points!

Euclidean : Euclidean plane (aka 2d geometry)

Metric : Distance metric → ie triangle inequality + symmetrical + non-neg + identity

non-negativity: d (u,v) >= 0

identity: d(u,u) = 0

symmetry: d(u,v) = d(v,u)

triangle inequality: d(u,v) + d(v,w) >= d(u,w)

General : Arbitrary graph where some pts are required, some optional steiner pts

Optimal Metric steiner tree has at most n-2 steiner points; steiner points have degree 3 @120 degrees

Euclidean

MST is 2/sqrt(3) -OPT → unit triangle

MST is 2-OPT for metric → unit polygon → OPT is all to central pt of distance 1, approx is MST of distance 2 to next pt along perimeter

Algos

Edge Cases

1. connect 2 pt ==> straight line
2. when steiner set is empty ==> normal MST

Brute force

`2^s`! runs of MST on each combination of using/not using steiner pts ==> 2^(|n-s| * n^2 logn)

Approximation proof for just using MST on metric

regular polygon bounding ==> MST approximation is no better than 2(n-1)/n

as n -> inf, approaches 2 for euclidean

DFS on optimal steiner tree

- visit each edge in the optimal steiner tree twice
- when bypassing the steiner points, new direct dist is <= 2x distance due to triangle inequality
- further cutting off of repeats gives even shorter
- then take one edge off the loop gives the MST of the non-steiner pts
- this MST is <= 2* optimal MST of steiner tree

For general Steiner Tree?

do `metric completion`

ie convert each edge |(u,v)| = shortest_path(u,v)

multiple dijkstra

v^3 floyd-warshall

but reconstruction of shortest(i,j) might be (i,k,l,j) so need to take into account if route needed

Travelling Salesman

Metric → follow triangle ineq

Metric Repeat

Non-Metric Repeat

Metric No-Repeat

> Above 3 are equivalent
> An approximation for one give an approximation for the others
> 1. metric repeat --> metric no repeat : skip repeated vertex, by triangle inequality
> 2. metric no repeat --> repeat : no-repeat sol is a valid sol for repeat and provides c-approximation
> 3. general repeat --> Metric repeat: trivial
> 4. metric repeat --> general repeat: metric completion (all pairs shortest path) on the general graph, then use metric repeat algo to solve

Non-Metric Non-Repeat

Brute force / Held Karp

Algos

Meet in the middle PS → split problem into two in different parts, find path distances in each half, check if sum = L

Is a loop so start can be 0

Pick a midpoint

Set half to be true → before midpoint, half to be false→ after midpoint

Brute force front half, throw into hashtable ; brute force back half and check if there is a front_half that matches the total distance req

2-Approx for M-NR-TSP

O(E^2)?

E logV MST + O(V) for DFS on tree

(since equivalent, only need for 1 of the 3)

```
1. MST on input graph
2. DFS on resulting MST
3. Output vertices in cycle induced by DFS traversal
4. (optional) remove duplicates for NR
```

1.5 approx for M-NR-TSP

in V^3 due to min weight matching step

```
Christofides's Algorithm:
Make MST→ convert MST into cycle by shortest path matching between vertices with odd degree
1. T = Min-Spanning-Tree(G) and let E be all the edges in T
2. Let O be set of vertices in T that has odd degree
```

  – O has even number of vertices (Handshaking lemma)
3. Find M = Min-Weight-Perfect-Matching on subgraph of G induced by O
4. Combine T+M to get a multigraph H whereby all vertices have even degree
5. Get the Eulerian circuit in H (cross each edge exactly once)
6. Output the vertices in Eulerian cycle (no repeat in VA)
```
### Held Karp DP → exact soln
Time complexity: O(2^V * V^2).
Space → n * 2^n

## Min-Cut-Max-Flow

Min cut when there is no path from Source s to Sink t in the residual/capacity graph
Because all paths are covered by the flow graph
Find Min-cut by doing BFS from source on the capacity graph
Or BFS from sink on the residual graph

Ford Fulkerson → E * U → U = max flow
- Each round of FF adds at least 1 to flow, each round of FF uses DFS, costs O(m)

Edmond Karp → E^2 V
Dinic → E*V^2

FF → find any possible augmenting path
EK → run BFS each time to take shortest augmenting path
Dinic → run BFS to find shortest paths, find augmenting paths on this level graph

Blocking question of kattis → use min-cut property
- 2 tier graph, 1 representing IN to vertex, 1 representing OUT of vertex
- Flow from int node's IN to OUT is the weight of itself, all other connections are inf
  - Source to A_in, B_out to sink, i_out ti j_in if adjacent

## Matching

Maximal (unweighted) matching is 2-approx of MCM on general graph → from midterm

### MCBM

#### Greedy O(nlogn)

If pairings can be sorted in some way that has meaning → eg slots and job times

#### UnWeighted

Augmenting Path → O(VE) → keep finding paths where start and end with unused vertex, alternating used and unused edges
Start with open on left, cross to right on an open edge
(augmenting path has 1 end in each side)
If free → aug path found
If taken → hop to the taken partner on left and recurse
Augment Path++ → O(V^2.5) → start with randomised/greedy selection before augmenting path ^

*A matching **M** in graph **G** is maximum iff there is no more augmenting path in G.*
Max Flow → left and right with weights 1 → run dinic ( e*v^2 )
Hopcroft-Karp → E*sqrt(V) = V^2.5 → run BFS to find all augmenting paths of (level graph) length

Maximum matching = smaller of the bipartite sides in complete bipartite
If no edge, will not be part of any matching
IF graph has |E| disjoint edges → |MCBM| = |E|

#### Weighted

Min-Cost-Max-Flow → O(V^2E^2)
Use Dijkstra as the way to find path instead of BFS

Hungarian → Weighted MCBM in O(V^3)
gets max-weight matching by default --> negate values + offset? to make max
1) complete the bipartite graph by add missing edges with weight -inf
2) label
   1) label LHS with largest outgoing edge
   2) label RHS with 0
3) create equality subgraph
   1) edges visible if sum of Left and right labels = weight of edge
4) search for augmenting path on the equality subgraph
5) for all unadded edges, delta = sum of node labels - edge weight
   1) (V^2) step
   2) pick the edge with smallest delta to add to equality subgraph
   3) decrease the LHS label to expose the new edge
   4) update the corresponding RHS to maintain the current edge sum
6) try to find augmenting path on the new bipartite graph
7) V iterations --> O(V^2) search at every step

### MCM

#### Small

DP with Bitmask O(v^2)
- Pick first unselected, test against all untaken and recurse
- Test option which does not pick that pairing too

#### Unweighted General

Edmonds- Blossom O(v^3)
Find augmenting path (standard is biased toward smaller vertex index)
If cycle back to self → collapse blossom and find more blossom
If aug path found → expand
```

Weighted General

GG -> np hard

Local Search

• Perturbative Search
– search space = complete candidate solutions
– search step = modification of one/more sol. components
– e.g. swap two edges (2-exchange) in a TSP tour
• Constructive Search (aka construction heuristics)
– search space = partial candidate solutions
– search step = extension with one/more sol. components – search step = extension with one/more sol. components
– e.g. from one vertex, go to nearest neighbor vertex, the Greedy Nearest Neighbor heuristic

Systematic search → complete search

General format

```
Create initial state
While not terminate {
  Step; evaluate; take best
}
```

Hill climbing get stuck in local optima

Randomised Iterative Improvement → random step transform → take best score at the end of time

Probabilistic Iterative Improvement → accept worsening step with probability based on quality → bigger deterioration, smaller probability

Simulated Annealing

Extension of PII, where probability is controlled by annealing schedule →
Geometric cooling T := 0.95 * T with n * (n-1) steps at each temperature (n = number of with n * (n-1) st

Temperature that informs range of allowed "step size"

Tabu Search

Create a tabu list of solution variables which cannot be changed for K iterations

Aspiration Criteria → override a solution's tabu state, thereby including the otherwise-excluded solution in the allowed set

Tabu tenure

TT too low → stagnates

TT too large → overly restrictive search path

Backtrack / force certain variables etc

Iterated Local Search

(hybrid)

Subsidiary local search → reach local optima asap

Perturbation step → escape local optima's "catchment" zone

- Too weak → cannot escape catchment zone
- Too strong → closer to random restart's loss of search history

Evolutionary

Encode problem as gene → mutation + recombination to create population

Tuning Local Search

Identifying areas that can be tweaked

Type 1 → parameters → "inputs" to the SLS which can tweak

Type 2 → Components → Choice of subsidiary local search, neighbourhood, perturbation etc

Type 3 → Search Strategies → details about acceptance criteria / termination criteria

- (type 1) Tabu tenure
- (type 2) Creating /selecting local neighbourhood
- (type 2) Selecting tabu mechanism
- (type 3) Adding Search strategy

Black box

- Human start with designing, then let machine tune the (selected) parameters based on systematic search

White box

- Human analyse the SLS behaviour; machine does analysis, but tweaks done by human

Push-Relabel

There exists an implementation that can solve Max flow in V^3

Start by pushing edge capacity from source to all neighbours → first ring will be unbalanced {}

For each unbalanced vertex: try to push to a connected edge where

1) Other end has a lower height
2) Edge still has capacity

Push min(capacity, excess in unbalanced vertex)

- Saturating push → the edge capacity is now 0, vertex excess might be 0?
- Non-saturating push → the vertex excess is now 0

If cannot push → raise height

- Any unused excess will eventually have height > source, and return flow to source

Think of a line graph

Max height → 2V → last node before sink has to be high enough to return excess to root

Max relabel → 2V* V → everything is = 2V ish

Saturating pushes → 2EV →

Non-saturating pushes → 2V^2 + 2EV*2V = 4V^2E →

Mini Project

DragonBall

- Pick a random point and send
- Increase the priority of all candidates identified by the response
- Keep picking points based on highest priority and add priority too

Cheese If You please

- Construct the LP & solve using simplex

Argument

- If not perfect matching → pick free vertex → opp cannot respond
- If perfect matching → just match anything that the opponent raises
  - Use the perfect matching to choose, so that anything he choose will definitely have a response

Chinese Postman

If all vertices have an even cardinality → there exists a closed euler path (all edges covered)

- Goal is to add "repeated" edges such that all vertices have even cardinality

Flyod warshall to get shortest distance between any 2 vertices (metric completion)

1x each path + additional distance needed to get from odd edge vertex to odd vertex

- End at same station → all vertices must be matched at the end
- Use MinCostMatching on complete graph of vertices are those with odd cardinality,