CS4234 Optimisation Algorithms

Polytime reduction:

CIRCUIT-SAT → SAT → 3-CNF-SAT → SUBSET-SUM
3-CNF-SAT → CLIQUE → VERTEX-COVER → HAM-CYCLE → TSP

Min Vertex Cover

Since VERTEX-COVER $\leq_p$ MIN-VERTEX-COVER, MVC is NP-hard (but not in NP).

MVC on tree: Polytime DP solution, $O(n)$ — MVC + MIS = n.

MVC with small answer (k): $O(2^k m)$, pick any edge and try colouring each endpoint separately and recurse.

Approx MVC:

- → Pick any edge, and colour both endpoints. It is a 2-approx.
- → Pick the vertex that will cover the most uncovered edges. It is a logn-approx.
- → Pick any edge, and randomly pick an endpoint. It is a randomised 2-approx.

Linear Programming / Simplex Method

° Objective function & constraints are all linear in all variables.

MWVC $\leq_p$ ILP : minimize $\left(\sum_{i=1}^{n} w_i x_i\right)$ where
$x_i + x_j \geq 1$ when (i,j) is an edge
$x_i \geq 0$ $\forall i$
$x_i \leq 1$ $\forall i$
$x_i \in \mathbb{Z}$ $\forall i$ ← "integer" constraint.

MWVC: weighted (each vertex has a weight)
ILP: integer linear program

Hence ILP is NP-hard. (and NP-complete)

LP is in P.

LP vector form: maximise $c^T x$ where $Ax \leq b$ and $x \geq 0$.

MWVC

Approx MWVC:

- → Reduce to ILP, solve LP, round up answer. It is a 2-approx.
- → Bar-Yehuda & Even's algorithm: Pick any edge, subtract weights of both endpoints by min weight in either endpoint. Answer is all vertices whose weight is zero.

Min Set Cover

$X = \{x_1, \cdots, x_n\}$

$S = \{S_1, S_2, S_3, \ldots, S_m\}$

where S_i = set of elements covered by this set.

MinVertexCover $\leq_p$ MinSetCover.

Approx MSC:

- → Pick the set that contains the most number of uncovered vertices. It is a logn-approx. (break ties by smaller set index)

Euclidean Steiner Tree

- can create Steiner points anywhere (no restriction)
- It is NP-hard
- Each Steiner point has degree 3, 120° angles.
- At most $n-2$ Steiner points in total.

Metric Steiner Tree

- Steiner points can only be chosen from a given set of candidates
- Distance function is metric:

$\forall u,v \in V : d(u,v) \geq 0$ [Non-negativity]

$\forall u \in V : d(u,u) = 0$ [Identity]

$\forall u,v \in V : d(u,v) = d(v,u)$ [Symmetry]

$\forall u,v,w \in V : d(u,v) + d(v,w) \geq d(u,w)$ [Triangle Ineq.]

Approx:

→ • Min Spanning Tree (of real vertices only) is 2-approx of Metric Steiner Tree. (see notes)

General Steiner Tree

- Points are also chosen from given set of candidates
- But distance function might not be metric

⇒ Do metric completion by relaxing edges using APSP algo (e.g. Floyd Warshall), run Metric Steiner Tree, then for each 'virtual' edge reconstruct the original edges.

Theorem: An α-approx for Metric Steiner Tree, when reconstructed will be an α-approx for General Steiner Tree.

Travelling Salesman Problem

Variants: → Metric (M) → General (G) ; → Repeat visit (R) → No repeat visits (NR)

All variants are NP-hard.

M-R ⇔ M-NR (due to metric, repeats are never useful)

M-R ⇔ G-R.

If we have a M-R algorithm that is c-approx, then metric complete the General ST, then run the algorithm, then reconstruct (might introduce repeats).

Approx:

→ * Min. Spanning Tree is 2-approx of TSP (those 3 variants). (see notes)

→ * Christofides' Algorithm is 1.5-approx of TSP (see add^n notes.

Maximum Flow

- Max flow = Min cut
- Aug. path thm: Flow is a max flow $\iff$ there are no aug. paths in the residual graph.
- To find min cut: DFS from source through edges with nonzero residual capacity, then every edge from S to $V \setminus S$ is part of the min cut.

Basic Ford-Fulkerson : $O(m^2 U)$ or $O(mnU)$ better

DFS to find any aug. path, and update... until no more aug. path.

max. capacity edge that leaves source

FF with Fattest Path : Choose path with largest min capacity : $O(m^2 \log n \log F)$ the max flow value

FF with capacity scaling : Solve scaled-down (with round down capacities) version $\log U$ times. $O(m^2 \log U)$.

Edmond-Karp : BFS to find shortest aug. path (i.e. min num. of edges) : $O(m^2 n)$

Dinic's : BFS to build level graph, and foreach flow on level graph send flow through : $O(mn^2)$

only has the edge (u,v) if (u,v) is in the original graph and $dist(s,u) + 1 = dist(s,v)$.

Push-Relabel : Each node starts with height $h(u)$ initially 0, except $h(s) = n$: $O(mn^2)$

IF can push (i.e. $\exists u \in V \setminus \{s,t\}$ and $v \in V$ s.t. there is excess at u, and (u,v) has residual capacity, and $h(u) > h(v)$), then push in (u,v).

Otherwise find some vertex with excess capacity to relabel (increment $h(u)$).

See slide 15

max height $= 2n$
number of relabel $\leq 2n^2$
number of sat. push $\leq 2mn$
number of non-sat push $\leq$
$2n^2 + (2mn)(2n) \leq O(4n^2 m)$

Proof using potential argument in slides.

Graph Matching

- Bipartite?
  - Yes → Can exploit problem structure
    - Yes → Run fast algorithm (e.g. greedy)
    - No → (Edge) Weighted?
      - No → Unweighted MCBM : use Augmenting Path / Augmenting Path ++.
      - Yes → Weighted MCBM : use Kuhn-Munkres (Hungarian)
  - No → Small?
    - Yes → DP with bitmask
    - No → Weighted?
      - No → Unweighted MCM : Edmond's Matching ++
      - Yes → Weighted MCM

Augmenting Path Algorithm (for unweighted MCBM)

Berge's Lemma : Matching is maximum $\Leftrightarrow$ There is no augmenting path (applies to general graph, not just bipartite ones)

Algorithm : Find and flip augmenting paths until there are none left : $O(\underset{\text{max number of flips}}{V}(\underbrace{V+E}_{\text{DFS}})) = O(VE)$

Maximum Flow (for unweighted MCBM with vertex capacities)

e.g. each vertex on the left must be matched with k vertices on the right, or with some vertices with total flow of k.

$O(m^2)$ for FF / $O(m\sqrt{n})$ for Dinic

Hopcroft-Karp Algorithm (for unweighted MCBM, faster) (sort of similar to Dinic)

Like augmenting path algorithm, but instead of finding one aug. path per iteration, it finds a maximal set of vertex-disjoint augmenting paths

$O(E)$ - Use BFS to find augmenting paths from all free vertices on left, stop at the level when one or more free vertices on the right is reached.

$O(E)$ - Find a maximal set of vertex-disjoint augmenting paths by doing DFS backward on the BFS level graph from the previous step.

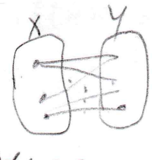
Worst case $O(E\sqrt{V})$.

Augmenting Path ++

- For each left vertex, pick an unmatched right vertex at random (if exists), and match them.
- Run normal augmenting path algorithm from each unmatched left vertex.
- $O(kE)$ — k unknown, maybe depends on V.

Hall's Marriage Theorem

Given a bipartite graph :



There is a matching that covers X $\Longleftrightarrow$ $\forall W \subseteq X,\ |W| \leq |N(W)|$

$N(W)$: set of all neighbours of vertices in W.

Kuhn-Munkres' Algorithm (for weighted MCBM)

- $O(n^3)$
- Default version is for max weighted perfect bipartite matching, but can modify
  - for min weighted (negate edge weights)
  - for imperfect matching (add dummy vertices/edges with irrelevant weights)
- Theorem : If $\forall x,y \in V,\ \ell(x)+\ell(y) \geq w(x,y)$, then any perfect matching in the equality graph (i.e. only containg those edges with $\ell(x)+\ell(y) = w(x,y)$) is maximum in the original graph.
- Pseudocode :
  1. - Initialise labels : $\ell(u) := \max_{v\in R}(w(u,v))$, $\forall u \in L$ (left vertices) ; $\ell(v) = 0, \forall v \in R$ (right vertices)
  2. - While matching is not complete,
     - Build equality graph
     - Pick any unmatched left vertex, and DFS (with fresh visited array) (alternating paths only) to find augmenting path
       - If augmenting path is found, flip and go to step 2.
       - Otherwise, let $S :=$ set of visited vertices on left; $T :=$ set of visited vertices on right
       - $\Delta := \min_{u\in S,\ v\in Y\setminus T}\{\ell(u)+\ell(v)-w(u,v)\}$; subtract Δ from $\ell(u), \forall u \in S$; add Δ to $\ell(v), \forall v \in T$
       - Go to 2

Edmond's Matching (Blossom) Algorithm

- $O(V^3)$
- Shrink & re-expand blossoms recursively
- Randomised greedy pre-processing also applicable, like AugmentingPath++.

Stochastic Local Search

- Start at a (random) position in search space
- Iteratively move to neighbouring position

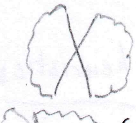
- Perturbative Search vs Constructive Search : slides — SLS p.6
- Systematic Search vs Local Search : slides — Completeness, Any-time Property, Complementarity (combining local and systematic search)
  - → when optimality or proof of insolubility is required
  - → time constraint not critical.
  - → reasonably good solutions in short time
  - → when parallel processing is used.

Some SLS methods:
- → Evolutionary (Genetic) Algorithm
- → Simulated Annealing
- → Tabu Search

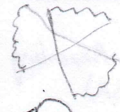
Definitions : SLS p11 - end

TSP

2-exchange (2-opt):

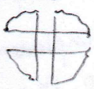


3-opt: (or any of 7 ways)



2.5-opt:

4-opt: (cannot be achieved by 2-opt)



Hill Climbing / Iterative Improvement - p 18

→ pick an improving state uniformly randomly

Delta Evaluations

→ Evaluation function of neighbouring states can be calculated quickly from current state.

Escaping local optima

- → restart (from random state) if local optima is encountered
- → allow picking non-improving and worse candidates when in a local optima.

} not always guaranteed to escape effectively from local optima.

Search strategy

- → Intensification: Aim to greedily increase solution quality or probability (e.g. hill climbing)
- → Diversification: Aims to prevent search stagnation (by getting trapped in confined regions) (e.g. un-informed random walk)
- → We want a balance of intensification & diversification

Larger neighbourhood → neighbourhood graph has smaller diameter
→ fewer local minima

Smaller neighbourhood → Faster iteration (need to look at all neighbours to determine where to walk).

"Exact neighbourhood": when the local optimum is guaranteed to be global optimum

Neighbourhood Pruning: reducing size of neighbourhood by excluding neighbours likely/guaranteed not to yield improvements (more important for large neighbourhoods, but might also help for small).

→ e.g. candidate list for TSP: for each vertex, store a limited number of vertices near enough, and only consider 2-exchange moves of those nearby neighbours.

Best improvement: Choose randomly from $I^*(s) := \underset{s' \in N(s)}{\operatorname{argmin}}\, g(s')$ ← set of best improving neighbours.

First improvement: Evaluate neighbours in fixed order, choosing the first improving step encountered (faster, but maybe weaker overall)

Variable neighbourhoods: Use k neighbourhood relations, usually ordered by increasing neighbourhood size and pick the smallest neighbourhood that facilitates improving steps.

Randomised Iterative Improvement (RII): In each search step, with a fixed probability, perform an uninformed random walk step instead of an iterative improvement step.

→ then we do not need to terminate when a local optima is encountered

↳ instead, bound by number of steps or CPU time from beginning of search or after last improvement.

Theoretical optimality → when run sufficiently long, RII is guaranteed to find optimal solution with arbitrarily high probability

Probabilistic Iterative Improvement (PII): Accept worsening steps with probability that depends on respective deterioration in evaluation function value.

Create function $p(g, s)$ that determines the probability distribution over neighbours of s.
(g: eval. func.; s: current state)

→ II and RII are special cases of PII.

For TSP → Meta-heuristics p 7

Metropolis condition: if selected step is worsening, accept it with probability $e^{\frac{f(s)-f(s')}{T}}$ (note: $f(s)-f(s')$ is negative)
if selected step is improving, always accept.
($f(s)$: original position; $f(s')$: new position; T: tunable parameter.)

Meta-heuristics: generic technique used to control an underlying problem-specific heuristic.

Simulated Annealing (SA): Vary a temperature parameter T → slides Meta-heuristics p. 10-11

→ Proposal mechanism to select candidate neighbour

→ Acceptance mechanism to decide whether to accept candidate neighbour (e.g. Metropolis condition)

→ Annealing schedule: how T varies with running time

↳ Some implementation details p. 14.
- → neighbourhood pruning
- → greedy initialisation
- → low temperature starts (to prevent good initial candidate solutions from being destroyed)
- → lookup table for acceptance probabilities: map $\frac{f(s)-f(s')}{T} \to e^{\frac{f(s)-f(s')}{T}}$ because the exp() function is expensive

Theoretical optimality → under sufficiently slow cooling and sufficiently long time (trajectory), SA is guaranteed to find optimal solution.

Tabu Search (TS): Use aspects of search history (i.e. memory M) to escape from local minima.
→ forbid steps to search positions recently visited.

→ Admissible neighbours: non-tabu positions in $N(s)$

→ Tabu tenure: how long a state is declared tabu (very important in TS) → p.19

→ Aspiration criterion: conditions in which tabu status may be overridden (e.g. improvement in global best solution so far)

→ can use additional intermediate-term or long-term memory to achieve additional intensification or diversification → p. 20

Hybrid SLS methods

Iterated local search p. 23
- → subsidary local search to reach local optimum (intensification)
- → perturbation steps for effectively escaping local optima (diversification)
  - ↳ chosen such that the effect cannot be undone by local search, e.g. searching in larger neighbourhood.

Evolutionary algorithm / Genetic algorithm: population → mutate → recombine → select

Memetic Algorithm: like genetic, but after each "mutate" and "recombine", use subsidiary local search to find local optimum, and keep local optimum instead (increases intensification)

SLS Design & Tuning Problem:
- Black box / White box parameter tuning
- Fitness Landscape Search Trajectory (FLST) Visualisation
- Integrated black-box/white-box approach

Quadrature assignment problem: Given $n \in \mathbb{N}$, $A, B \in M_{n \times n}$, find $\pi \in \Pi(n)$ minimising $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{\pi(i)\pi(j)} b_{ij}$.

→ assigning factories to locations, where some known quantity of goods need to travel between each pair of factories.