Contents

Beginning of 08/26/2024

1 Introduction

Before we dive into approximation algorithms, it is worth noting that optimization problems (the ones that output numerical values) are closely related to decision problems (the ones that output yes or no), which we have previously studied. To draw a few connections:

- 3SAT. Given a 3-conjunctive normal form[1]...
  - (Decision) Does there exist an assignment of variables to satisfy all clauses?
  - (Optimization) Find an assignment of variables that maximizes the number of satisfied clauses.
- GRAPH COLORING. Given a graph $G = (V, E)$...
  - (Decision) Is it possible to properly color[2] G using k colors?
  - (Optimization) Find the minimum k to properly color G.
- VERTEX COVER. Given a graph $G = (V, E)$...
  - (Decision) Is there a set $|S| = k$ of vertices such that for every edge $e = (u, v)$, $|\{u, v\} \cap S| \geqslant 1$?
  - (Optimization) Find a vertex cover S of minimal cardinality.
  - Note this problem is equivalent to INDEPENDENT SET: find a maximal subset $S \subset V$ such that for any $u, v \in S$, edge (u, v) does not exist. This is based on the following observation:

$$\begin{aligned}(S \text{ is an independent set}) &\iff (\text{no edge has both endpoints in } S)\\ &\iff (\text{each edge has } \geqslant 1 \text{ endpoint(s) in } S^c) \iff (S^c \text{ is a vertex cover})\end{aligned}$$

    and so the equivalence is given by

$$(G \text{ has an independnet set of size } \geqslant k) \iff (G \text{ has a vertex cover of size } \leqslant |V| - k).$$

1.1 Formulations of Optimization Problems

An optimization problem has four components:

(1) Input to the problem.

(2) Set of feasible solutions (e.g. in VERTEX COVER, the set of all valid vertex covers). Also known as constraints.

(3) Objective function, also known as constraints. Maps feasible solutions to numbers (e.g. cardinality of vertex covers).

(4) Maximization or minimzation (e.g. minimize in VERTEX COVER, maximize in INDEPENDENT SET).

In general, for problems that we cannot provide efficiently provide an optimal solution, we resort to designing an **approximation algorithm** as a workaround. Instead of demanding optimality, we require a worst-case quality assurance:

[1] A formula consisting of ANDs of any number of clauses, where each clause is the ORs of exactly three literals.
[2] No vertices that share an edge should be colored the same.

Definition

For a minimization [resp. maximization] problem, an algorithm ALG is called an **α-approximation**, $\alpha \geqslant 1$ [resp. $\alpha \leqslant 1$], if for all input instances I, $\text{ALG}(I) \leqslant \alpha \cdot \text{OPT}(I)$ [resp. $\geqslant$].

With these definitions, we briefly categorize algorithms to the following classes:

- Exact algorithms. These are algorithms that solve optimization problems to optimality (obtains exact maximizer / minimizer).
- **PTAS** (*polynomial time approximation schemes*). An algorithm that provides a solution within a factor $1 \pm \epsilon$ of being optimal given any $\epsilon > 0$, but as $\epsilon \to 0$ the running time blows up.
- Constant approximation $\mathcal{O}(1)$. An algorithm that does not yield optimal solution, but the quality of approximation does not degrade as input size increases. Value of α bounded from above.
- Superconstant, e.g. $\mathcal{O}(\log N)$, $\mathcal{O}(N^c)$, etc. The quality of approximation decays as input size increases.

A (probably) counterintuitive fact is that questions that can be Karp reduced to each other need not necessarily have the same approximation difficulty. For example, VERTEX COVER have 2-approximations (hence constant approximation), but INDEPENDENT SET is of $\mathcal{O}(N^c)$, one of the most resistant problems to approximation.

2 Approximations of VERTEX COVER

Let us return to the VERTEX COVER problem.

> Given $G = (V, E)$, find a $S \subset V$ of minimal cardinality such that for all $e = (u, v) \in E$, $|\{u, v\} \cap S| \geqslant 1$.

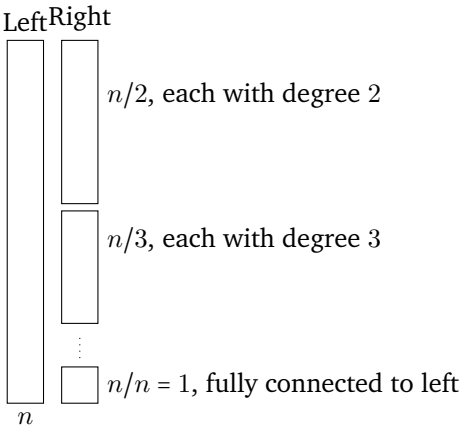
2.1 Greedy Approaches

The most intuitive approach would be to greedily select the vertex with highest degree, and iteratively repeat this process — after all, we want to more edges with fewer vertices, so high-degree vertices seem to be a natural choice. Unfortunately, curated examples show that this algorithm is of approximation factor $\Omega(\log n)$ (in fact it is $\Theta(\log n)$).

A "bad" graph for the naïve greedy algorithm.

Consider a bipartite graph. The left component consists of n nodes/vertices. The right component consists of $n - 1$ blocks, each with n/j nodes, $j \in [n]$[3]. This way, the right component has a total of approximately $n \log n$ nodes.

For each right side block of size n/j, we assume that each node has degree j, and that they all are connected to different nodes on the left side. (For example, the first node in the $n/2$ component can connect to the first two in the left component, and the second node in the $n/2$ component can connect to the 3^{rd} and 4^{th} on the left side, and so on.)



Now, what will our naïve greedy algorithm do? First, it will pick the one single node from the bottom right component since it has degree n and no other node does (the ones on left side each are connected to $n - 1$ nodes, one from each block on the right). After picking this node and removing all the connected edges, in the remaining graph, the nodes in the $n/(n - 1)$ block have degrees $n - 1$, whereas the left nodes now have degrees $n - 2$. So we pick the nodes in the $n/(n - 1)$ block. Iteratively, we see that we never get to pick the nodes from the left, and in the end we must have picked every single node on the right. That is $\mathcal{O}(n \log n)$ nodes, whereas we clearly see that an more optimal solution is to just pick every node on the left, a total of $\mathcal{O}(n)$ nodes.

From this instance alone we see that our algorithm must have an approximation factor $\geqslant \log n$. (And we'll see later that this algorithm is isomorphic to a $\log n$-approximation algorithm of SET COVER, so here the optimality factor $\Omega(\log n)$ becomes $\Theta(\log n)$.)

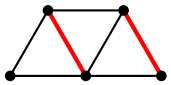
The fix, though seemingly less reasonable than the naïve greedy algorithm, is to add both vertices of the same edge in each iteration, and discard all edges that are not completely disjoint from our set:

[3] $i \in [n]$ is an abbreviation of $1 \leqslant i \leqslant n$.

Algorithm 1: Improved Greedy Algorithm

Inputs: graph $G = (V, E)$
start with $S = \varnothing$
while S is not a vertex cover **do**
    choose any edge $e = (u, v)$ with $|\{u, v\} \cap S| = 0$
    $S \leftarrow S \cup \{u, v\}$
return S

Indeed, even for simple cases, we see that sometimes this algorithm fails to choose optimally. Consider the following graph: it is clear that there exists a set of 3 vertices that covers the graph, but our algorithm terminates after 2 iterations, resulting in a total of 4 vertices being picked. Nevertheless, the following proof shows that this algorithm is a 2-approximation.



Beginning of 08/28/2024

Proof. Another way to view this algorithm is by paying attention to the edges it has selected among the iterations (i.e., the e instead of the $\{u, v\}$). The key observation of this proof lies in the following fact: for the edges chosen by this algorithm (call it ALG), they all are incident on two distinct sets of vertices. In other words, these chosen edges cannot share *any* common endpoint. A matching!

Suppose our graph G has a matching of size k. Consider any vertex cover and any edge $e = (u, v)$. It follows that either u and/or v is in the cover. That we have a matching of size k means we have k mutually disjoint edges, so the vertex cover must contain one endpoint from each edge, resulting in a cardinality of $\geqslant k$. In particular, if G admits a matching of size n, then the *optimal* vertex cover must also have at least k vertices.

Assuming the previous notations, our outputted vertex cover by ALG contains both endpoints of each edge of the matching, so its cardinality is $2k$. The previous paragraph showed that OPT has at least k vertices. Hence ALG achieves an approximation factor of 2, completing the proof. □

Remark. We will later show that optimal solution exist on bipartite graphs.

Remark. When the optimal solution isn't accessible, a common trick is that we compare it against a surrogate that serves as a stand-in for the optimal value, providing a lower bound (e.g., in the previous problem, the optimal vertex cover has at least size k). This approach allows us to establish a worst-case approximation factor.

2.2 Threshold Rounding

Another way to construct a 2-approximation algorithm for VERTEX COVER is by relaxing the problem and drawing some connection to **linear programming** (LP). The main idea is simple: we want to assign binary values x_v to each vertex v while preserving the algebraic structures of the original problem:

$$\min_{v \in V} x_v \qquad \text{subject to} \qquad \begin{cases} x_u + x_v \geqslant 1 \text{ for each edge } (u,v) \in E \\ x_v \in \{0,1\} \text{ for each } v \in V. \end{cases}$$

However, a problem immediately arises: the feasible set is not convex! Consider the simplest graph of one edge and two vertices u, v. We can set either one to be 1 and the other to be 0, but taking the average, $x_u = x_v = 0.5$, no longer a feasible solution. The fix? We deviate from the original problem and allow *fractional* values for x_v:

$$\min_{v \in V} x_v \qquad \text{subject to} \qquad \begin{cases} x_u + x_v \geqslant 1 \text{ for each edge } (u,v) \in E \\ x_v \in [0,1] \text{ for each } v \in V. \end{cases} \tag{1}$$

Notice that we can still assume $x_v \in [0,1]$ since if $x_v > 1$, we can simply set x_v to be 1 to further decrease the objective function. We will assume the fact that there are polynomial-time algorithms that solve these type of LP.

A few problems nevertheless persist. *Firstly, how do we define an approximation factor?* Now we have three quantities: the output of our algorithm, the output of the optimal solution to VERTEX COVER, and the output of the optimal solution to LP. Let us call these $\mathrm{ALG}, \mathrm{OPT(VC)}$, and $\mathrm{OPT(LP)}$, respectively. Immediately we have the following inequality:

$$\mathrm{ALG} \geqslant \mathrm{OPT(VC)} \geqslant \mathrm{OPT(LP)} \tag{2}$$

where the first $\geqslant$ follows from the optimality of $\mathrm{OPT(VC)}$ on VERTEX COVER, and the second $\geqslant$ follows from the fact that the LP has less constraints than VERTEX COVER. Therefore we can consider the ratio between ALG and $\mathrm{OPT(LP)}$ and define an approximation factor out of it.

Second problem: how do we recover an ALG *from* $\mathrm{OPT(LP)}$ *that makes sense?* The answer here is via **rounding**: given our optimal fractional solution, we want to recover an integer solution from which we can extract a valid vertex cover. And most naturally we round our fractional values to 0 and 1 with a threshold of 0.5: if $x_v \geqslant 0.5$ we round it to 1, and 0 otherwise.

> ***Proof that rounding is a 2-approximation.*** We first observe that rounding yields a valid vertex cover: for any edge $e = (u,v)$, if $x_u + x_v = 0$ post-rounding, then it means x_u, x_v are both < 0.5 pre-rounding, contradicting the constraints in (1). Therefore each edge has at least one value of 1 after rounding, i.e., this is a valid vertex cover. To see this algorithm has an approximation factor of 2, we note that the values assigned to each edge is at most doubled (0.5 to 1). Therefore the objective is at most doubled. □

Third problem: $\mathrm{OPT}(LP)$ *is just as elusive as* $\mathrm{OPT(VC)}$*, not to mention we obtained it using a LP-solving black box. What is going on here?* To answer this question, we will introduce a common technique in LP:

2.3 The Primal-Dual Method

The **primal-dual method** is used in LP as a tool to provide an alternative perspective that sometimes reveals additional relationships between constraints and objective function values. With the help of *strong* or *weak duality*, this method helps derive bounds or even optimal values for our objective functions, as we will see below.

Overall, the transformation of primal LP into dual LP assumes the following form:

$$\text{Primal} : \min\{c^T x \mid Ax \geqslant b, x \geqslant 0\} \iff \text{Dual} : \max\{b^T y \mid A^T y \leqslant c, y \geqslant 0\} \tag{3}$$

In slightly more details:

- Dual variables correspond to primal constraints: for each constraint in the primal, define a dual variable y. *In the* VERTEX COVER *example, each constraint is of form* $x_u + x_v \geqslant 1$*, one for each edge* (u, v)*, so a dual variable would be defined w.r.t. edges:* $y_e, e \in E$.

- Dual constraints correspond to primal variables: for each variable in the primal, there will be a constraint in the dual. *The primal coefficient for each vertex is* 1*, so the dual constraint requires*

$$\sum_{\text{edge } e \text{ incident to } v} y_e \leqslant 1 \qquad \text{for all } v \in V.$$

- If the primal is a minimization problem, the dual will be a maximization. Vice versa. *In our example, the dual would be a maximization.*

- The signs of dual variables also correspond to the primal constraints. If the primal constraint is $\leqslant$, the corresponding dual variable is non-negative. *This means* $y_e \geqslant 0$.

Therefore, we have successfully identified the dual LP of (1):

$$\underbrace{\min \sum_{v \in V} x_v \text{ subject to } \begin{cases} x_u + x_v \geqslant 1, (u,v) \in E \\ x_v \in [0,1] \end{cases}}_{\text{Primal LP}} \quad \Longrightarrow \quad \underbrace{\max \sum_{e \in E} y_e \text{ subject to } \begin{cases} \sum_{e \in S_v} y_e \leqslant 1 \text{ for all } v \in V \\ y_e \geqslant 0 \end{cases}}_{\text{Dual LP}} \tag{4}$$

where S_v denotes the "star graph" at v, or the collection of edges incident on v. Combining (2) and duality, we see that

$$\text{ALG} \geqslant \text{OPT(VC)} \geqslant \text{OPT(LP)} \qquad \text{and} \qquad \text{OPT(dual)} \geqslant \text{SOL(dual)}$$

since the dual LP is a maximization problem and OPT(dual) is assumed to attain the maximum value, and it must be no less than any valid dual solution, SOL(dual).

To relate the primal constraints to dual, we note that for each edge $e = (u, v)$,

$$y_e(x_u + x_v) = y_{(u,v)}(x_u + x_v) \geqslant y_{u,v}$$

since $x_u + x_v \geqslant 1$. Therefore,

$$\begin{aligned} \sum_{e \in E} y_e \leqslant \sum_{(u,v)} y_{(u,v)}(x_u + x_v) &= \sum_{(u,v)} y_{(u,v)} x_u + \sum_{(u,v)} y_{(u,v)} x_v \\ &= \sum_v \left(\sum_{(u,v) \in S_v} y_{(u,v)} x_v \right) && \text{(each } y_e \text{ is multiplied by both of its endpoint values)} \\ &\leqslant \sum_v x_v && \text{(dual constraint } \sum_{(u,v) \in S_v} y_{(u,v)} \leqslant 1) \end{aligned}$$

which shows that the objective value of the dual is always less than the objective value of the matching primal. By weak duality we therefore have

$$\text{ALG} \geqslant \text{OPT(VC)} \geqslant \text{OPT(LP)} \geqslant \text{OPT(dual)} \geqslant \text{SOL(dual)} \tag{5}$$

and so anything valid for the dual LP is a lower bound for $\mathrm{OPT}(\mathrm{VC})$. This is more useful because we have *complete* control over what $\mathrm{SOL}(\mathrm{dual})$ looks like.

> **Remark.** The algorithm was designed without any explicit mention of LP, but as we have seen, the analysis heavily involved it. This technique is called **dual fitting**.

Designing & Analyzing the Primal-Dual Algorithm

The high level idea is that we want to build up a primal solution, build up a dual solution simultaneously, and draw a connection between the primal and the dual via e.g. weak duality.
More generally, below is the general roadmap:

(1) Start with $\mathrm{P} = \mathrm{D} = 0$.

(2) (The following applies to VERTEX COVER; adjustment may be needed for general problems.) Recall that the primal is a minimization problem and dual maximization.

(3) Eventual goals:

- (Primal feasibility) P is feasible for primal LP.
- (Dual feasibility) D is feasible for dual LP.

(4) Observation:

- D is already feasible (set each $y_e = 0$). Therefore, *we want to keep dual feasibility invariant* under each iteration. Consequently,
- *The algorithm terminates when primal feasibility is satisfied.*

(5) To maintain an approximation factor, always ensure $\Delta\mathrm{P} \leqslant \alpha \cdot \Delta\mathrm{D}$ so that the primal will remain within a factor of α of D, resulting in an α-approximation. (In this case, $\alpha = 2$.)

Let us now apply the primal-dual algorithm to VERTEX COVER.

Algorithm 2: Dual LP for VERTEX COVER

Inputs: graph $G = (V, E)$, primal $x : V \to \mathbb{R}$, dual $y : E \to \mathbb{R}$
Initialization: $x(\cdot) = 0$ and $y(\cdot) = 0$
while Primal LP is not satisfied **do**
    find a dual variable y_e that can be increased without violating dual feasibility (guaranteed to exist when primal not satisfied)
    increase y_e until *some* dual constraint is tight
    for each primal variable v corresponding to the tight dual constraint **do**
        set $x_v \leftarrow 1$
return $\{v \in V : x_v = 1\}$

The correctness proof is omitted, but to see that this provides a 2-approximation algorithm, simply notice that ΔD comes from line 4 and ΔP comes from line 7. Each time a dual constraint becomes tight, its increment is precisely 1 (y_e must be going directly from 0 to 1, no in-betweens, since the y_e's we perturb are always disjoint), and line 7 repeats at most 2 times, one on each endpoint of y_e. Therefore, in each iteration, $\Delta P \leqslant 2\Delta D$, proving an approximation factor of 2.

> **Remark.** The primal-dual method is a *very* roundabout way of giving the exact result we pursued earlier. Why bother? The answer is because this alternative perspective is important: once we start look at even slightly more complicated problems (e.g. weighted vertices instead of uniform weight), our naïve approach quickly fails, but the primal dual algorithm, providing a more rigorous and systematic approach, continues to shine.

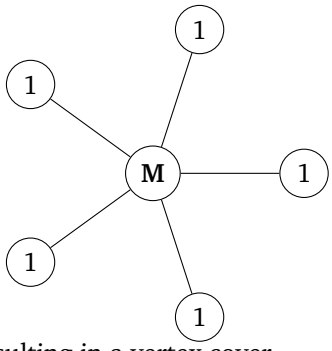
Beginning of 09/04/2024

2.4 Weighted VERTEX COVER

The weighted VERTEX COVER problem is identical to the standard VERTEX COVER, except now each vertex $v \in V$ has a weight $w_v \geqslant 0$, and we want to find a cover with the least weight. Put in fancy notations,

> Given $G = (V, E)$, where each vertex $v \in V$ is assigned a weight $w_v \geqslant 0$, find a $S \subset V$ with the smallest $\sum_{v \in S} w_v$ subject to the constraint that, for all $e = (u, v) \in E$, $|\{u, v\} \cap S| \geqslant 1$.

As promised in the previous remark, we will first show how even our improved greedy algorithm fails miserably. The simplest example to consider here is a graph $G = (\{u, v\}, (u, v))$, with $w_u = 1$ and $w_v = M$ for some large M. The optimal solution yields $\{u\}$ with total weight u, but the greedy algorithm would pick both endpoints, giving us a total weight of $(M + 1)$, which makes the approximation factor arbitrarily large!



Perhaps a less trivial example (which shares the same idea) is the star graph shown here, where each of the n leaf nodes has weight 1 and the center node has weight M. An optimal solution, assuming $M \gg n$ is large, would pick all leaf nodes, resulting in a total weight of n, whereas our greedy algorithm would have chosen the center node plus any leaf node, resulting in a vertex cover of weight $M + 1$. Again, this resulting approximation factor can be arbitrarily bad depending on how large M is, compared to n.

The fix for this problem directly is not to complicated: our intuition suggests that we should somehow prefer nodes with less weight when adding to our vertex cover. It's not hard to rigorously formulate this algorithm directly, but what's charming is that in fact, *the dual algorithm works with almost no modification*. The only change lies in the dual constraint: now the primal objective is $\min \sum_v w_v x_v$, so each dual constraint is bonded by w_v, namely, $\sum_{e \text{ incident to v}} y_e \leqslant w_v$ for each v:

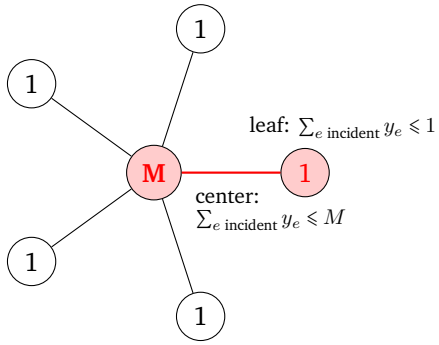
$$\min \sum_{v \in V} w_v x_v \text{ subject to } \begin{cases} x_u + x_v \geqslant 1, (u, v) \in E \\ x_v \in [0, 1] \end{cases} \implies \max \sum_{e \in E} y_e \text{ subject to } \begin{cases} \sum_{e \in S_v} y_e \leqslant w_v \text{ for all } v \in V \\ y_e \geqslant 0. \end{cases} \tag{6}$$

The algorithm itself works verbatim.

Algorithm 3: Dual LP for Weighted VERTEX COVER

Inputs: graph $G = (V, E)$ with weights $w : V \to \mathbb{R}^+$, primal x, dual y
Initialization: $x(\cdot) = 0$ and $y(\cdot) = 0$
while Primal LP is not satisfied **do**
    find y_e that can be increased without violating dual feasibility
    increase y_e until *some* dual constraint is tight
    for each primal variable v corresponding to the tight dual constraint **do**
        set $x_v \leftarrow 1$
return $\{v \in V : x_v = 1\}$

Before analyzing it, let's first see the algorithm in action. Same star graph example as above. Suppose our algorithm chose the red edge (call it e) initially. Clearly, y_e can be increased because none of the dual constraint related to y_e is currently tight. We note that in this graph, the dual constraint associated with the center is $\sum_{e \text{ incident}} y_e \leqslant M$, whereas for each leaf it is $\sum_{e \text{ incident}} y_e \leqslant 1$. Therefore we are free to increase y_e by 1, after which the constraint corresponding to the red leaf grows tight. Therefore we select the red leaf node but not the center node. If M is sufficiently large such that it always carries more weight in primal (and leaves more room in dual), then the center node will never be chosen. This solves our problem encountered before.



Of course, whenever a dual constraint becomes tight, nothing will make it loose again, so the algorithm is guaranteed to terminate in polynomial runtime. Let us now prove its correctness, and that it is 2-approximate like before.

Proof that the dual weighted* VERTEX COVER *algorithm is 2-approximate.

- Claim 1. The dual is feasible. This follows since we explicitly demanded so when tweaking y_e.
- Claim 2. The primal is feasible when the algorithm finishes. Suppose not, that two adjacent vertices $(u, v) = e$ are both unselected. This means $y_e = 0$ for any e incident on u and v. Hence

$$\sum_{e \text{ incident on } u} y_e = \sum_{e \text{ incident on } v} y_e = 0.$$

  But then we can increase $y_{(u,v)}$ by 1, contradiction.
- Claim 3. The algorithm is 2-approximate. Different from before, instead of showing $\Delta P \leqslant 2\Delta D$, here we directly relate the final output P to D. We recall out objectives:

$$P = \sum_{v \in V} w_v \mathbf{1}[x_v = 1]^4 \qquad \text{and} \qquad D = \sum_{e \in E} y_e.$$

  The rest is just pushing notations. When the algorithm terminates, the nodes labeled $x_v = 1$ are precisely

those whose dual constraints are saturated. Therefore, for each such v, $\sum_{e \text{ incident on } v} y_e = w_v$. Hence

$$\begin{aligned} P = \sum_{v \in V} w_v \mathbf{1}[x_v = 1] &= \sum_{x_v = 1} \left[\sum_{e \text{ incident on } v} y_e \right] \\ &= \sum_{(u,v) \in E} y_e \cdot |\{u, v\} \cap \{v : x_v = 1\}| \qquad \text{(count of } y_e = \text{num. of tight endpoints)} \\ &\leqslant \sum_{(u,v) \in E} y_e \cdot 2 = 2D. \end{aligned}$$

□

Remark. Previously we showed a 2-approximate algorithm for VERTEX COVER based on rounding. The very same algorithm extends trivially to the weighted case, giving us another 2-approximation (round x_v to 1 if the fractional optimal solution $x_v^* \geqslant 1/2$ and 0 otherwise.)

2.5 A Taste of Randomized Algorithms

Once again, we look back at "bad" examples we derived earlier, this time the carefully constructed graph which yielded an approximation factor of $\log n$ for our original greedy algorithm. Recall that our algorithm failed miserably, because we were forced to always choose nodes from the right side. A natural attempt is to allow some randomness, so that we may find ourselves an opportunity to break the curse of only choosing nodes from the right side:

Algorithm 4: Randomized VERTEX COVER Algorithm

Inputs: graph $G = (V, E)$
start with $S = \varnothing$
while S is not a vertex cover **do**
    choose any edge $e = (u, v)$ with $|\{u, v\} \cap S| = 0$ (i.e. uncovered)
    pick one of $\{u, v\}$ u.a.r. (uniformly at random) and add to S
return S

Not surprisingly, this gives us yet another 2-approximation algorithm for VERTEX COVER. The proof is quite different from the ones we've seen in primal-dual. Here, we will exploit the known / provable properties of the optimal solution, OPT, despite not having the slightest clue what it may look like. We will directly compare OPT with our randomized algorithm, ALG and attain useful bounds.

Proof. Let V_o be the set of vertices chosen by (an) OPT. For each $v \in V_o$, let S_v be the star graph with respect to the original $G = (V, E)$, i.e., $S_v = \{(v, u) \in E\}$, the collection of all edges incident on v. Since OPT is feasible, every edge is covered and in particular belongs to one of the star graphs, i.e., $\bigcup_{v \in V_o} S_v = E$. Therefore, our algorithm RLG is essentially *choosing a bunch of edges (or equivalently a bunch of vertices) from a bunch of star graphs*. (This is criminally oversimplifying things, of course, but the italicized claim is true nonetheless.)
So a natural question to ask is, how many vertices (on expectation, since our algorithm is probabilistic) will we choose from a *specific* graph? Let's just consider a certain star graph S_u with center u, and edges e_i leading to leaf nodes v_i. Let the random variable X_u denote the number of vertices chosen from S_u.

- The first time ALG considers an edge $e_i = (u, v_i)$ in S_u, there is a probability of $1/2$ such that it chooses u. If this is the case, ALG will never consider *any* edge in S_u again because u has been chosen. So $\mathbb{P}(X_u = 1) = 1/2$.

[4] I use $\mathbf{1}[S]$ to denote the indicator function of S, i.e., $\mathbf{1}[S](\omega) = 1$ if $\omega \in S$ and 0 otherwise.

- The event $X_u = 2$ means (i) when ALG consider an edge from S_u for the first time, it chose the leaf node over u, but (ii) the second time ALG considers an edge in S_u, it did choose u. So $\mathbb{P}(X_u = 2) = 1/2 \cdot 1/2 = 1/4$.
- We see that X_u is essentially a (truncated, since S_u is finite) geometric random variable with parameter $1/2$, i.e., $\mathbb{P}(X_u = k) = 2^{-k}$. Therefore $\mathbb{E}X_u <= 2$.

Since on expectation ALG chooses $\leqslant 2$ vertices from each star graph, and there are $|V_o|$ star graphs in total, we see that the expected number of vertices chosen by ALG is $\leqslant 2|V_o|$, i.e., it is 2-approximate. □

Now what? Update adding this randomized algorithm to approximate weighted VERTEX COVER, of course. Revisiting the simplest weighted example of $G = (\{u, v\}, \{(u, v)\})$, if the vertex weights are 1 and M respectively, then the most natural way to balance our randomized algorithm is to choose the vertex with small weight (1) with high probability, and choose the big weight (M) with low probability. We make the proportion linear, and prove that this indeed gives a 2-approximation.

Algorithm 5: Randomized Weighted VERTEX COVER Algorithm

Inputs: graph $G = (V, E)$ with weights $w : V \to \mathbb{R}^+$

start with $S = \varnothing$

while S is not a vertex cover **do**

  choose any edge $e = (u, v)$ with $|\{u, v\} \cap S| = 0$

  $$S \leftarrow \begin{cases} S \cup \{u\} & \text{with probability } w_v/(w_u + w_v) \\ S \cup \{v\} & \text{with probability } w_u/(w_u + w_v) \end{cases}$$

return S

Proof that randomized weighted* VERTEX COVER *is a 2-approximation. The idea is similar to that of the unweighted case. If V_o is the set of vertices chosen by OPT, then the total weight is $\sum_{v \in V_o} w_v$. Therefore, similar to the unweighted case, our goal is to show that for each star graph S_v, the expected total weight of the vertices chosen by our algorithm ALG is bounded by $2w_v$, so that linearity of expectation says

$$\mathbb{E}[\text{ALG}] = \mathbb{E}\left[\sum_{v \in \text{ALG}} w_v\right] = \mathbb{E}\left[\sum_{v \in V_o} \text{weights on } S_v\right] \leqslant \mathbb{E}\left[\sum_{v \in V_o} 2w_v\right] = 2\mathbb{E}\left[\sum_{v \in V_o} w_v\right] = 2[\text{OPT}].$$

So let us consider a star graph S. Let the center node be v with weight w, and let the leaves v_i each have weights w_i, $i \in [k]$ (so a total of k leaves). To compute the expected total weight, for each node, we can multiply the weight by the probability that this vertex is chosen, then sum over all vertices in S. We WLOG[5] assume the edges are considered in the order of $(v, v_1), (v, v_2), \ldots$. [We'll later show this is valid.] Like the unweighted counterpart, we observe that as soon as w is chosen, ALG will never consider any remaining edge from S again. Therefore,

$$\mathbb{P}(v_i \text{ is chosen}) = \mathbb{P}(v_i \text{ is chosen in } (v, v_i)) \cdot \mathbb{P}(v_j \text{ is chosen in } (v, v_j) \text{ for all } j < i). \qquad (*)$$

And now, time for algebra.

$$\begin{aligned}
\mathbb{E}[\text{total weight}] &= w \cdot \mathbb{P}(v \text{ is chosen}) + \sum_{i=1}^{k} w_i \cdot \mathbb{P}(v_i \text{ is chosen}) \\
&\leqslant w \cdot 1 + \sum_{i=1}^{k} w_i \cdot \mathbb{P}(v_i \text{ is chosen in } (v, v_i)) \cdot \prod_{j<i} (v_j \text{ is chosen in } (v, v_j)) \\
&= w + \sum_{i=1}^{k} w_i \cdot \frac{w}{w + w_i} \cdot \prod_{j<i} \frac{w}{w + w_j} = w + w \cdot \sum_{i=1}^{k} \frac{w_i}{w + w_i} \cdot \prod_{j<i} \frac{w}{w + w_j} \\
&= w + w \cdot \sum_{i=1}^{k} \frac{w_i}{w + w_i} \cdot \prod_{j<i} \left(1 - \frac{w_j}{w + w_j}\right) \\
&= w + w \cdot \sum_{i=1}^{k} p_i \cdot \prod_{j<i} (1 - p_j) \qquad \text{(define } p_i := w_i/(w + w_i)\text{).}
\end{aligned}$$

An easy probabilistic interpretation of the highlighted term is as follows. Suppose we have n coins, where the j^{th} coin has a probability p_i of landing on heads. Then $p_i \prod_{j<i}(1 - p_i)$ is the probability that the first $i - 1$ coins land on tails but the i^{th} coin lands on heads. Summing over i we see that the highlighted term is the probability of the union of some disjoint events which is obviously bounded by 1. So $\mathbb{E}[\text{total weight}] \leqslant 2w$. Now apply (*) and the proof is complete! □

Enough VERTEX COVER. Let's now look at something more general: the SET COVER problem.

[5] Without loss of generality.

3 SET COVER

> Given a universal set U of n elements and m subsets $S_1, \cdots, S_m \subset U$ with $\bigcup_i S_i = U$, find a minimum number of subsets that cover U.

As usual, we approach this problem using a naïve greedy algorithm: always pick the set that covers as many remaining elements as possible. Note that the "bad" example in VERTEX COVER can be transformed into an instance of SET COVER via the following:

- The universal set $U = E$, the set of all edges in the bipartite graph $G = (V, E)$.
- For each vertex v, the star graph $S_v = \{(v, u) \in E\}$ can be viewed as a collection of edges, hence a subset of U. Notice that $\deg(v) = |S_v|$, so naturally the notion of choosing the highest degree vertex in the naïve greedy VERTEX COVER algorithm corresponds to that of picking the set that covers most elements in SET COVER.

Since the naïve greedy VERTEX COVER problem has an approximation factor of (at least) $\log n$ we know our greedy SET COVER algorithm can do no better. Fortunately, the following proof shows it is doing no worse either.

Greedy* SET COVER *algorithm is* $\log n$*-approximate. Let $S_1, S_2, \cdots$ be the sequence of sets picked by our greedy algorithm, ALG. Say S_1 covers x_1 elements, and for each $i > 1$, S_i covers i *additional* (uncovered) elements. Consider the optimal solution OPT which consists of, with the abuse of notations, OPT subsets of U. The key observation is that:

- $\underline{x_1 \geqslant n/\mathrm{OPT}.}$ To see this, note that $|U| = n$, so by pigeonhole the largest set has $\geqslant n/\mathrm{OPT}$ elements.
- By the same token, $x_2 \geqslant (n - x_1)/\mathrm{OPT}$ since all sets in OPT must still collectively cover the remaining $n - x_1$ elements. Thus, the best remaining set performs at least as good as the average.
- Iteratively, $x_i \geqslant (n - \sum_{j<i} x_j)/\mathrm{OPT}$.

Rearranging the last inequality, we obtain (abusing the notation that ALG also means the cardinality of the outputted set)

$$1 \leqslant \mathrm{OPT} \cdot \frac{x_i}{\sum_{j \geqslant i} x_j} \qquad \text{for each } i \in [\mathrm{ALG}]$$

Summing over i we see

$$\begin{aligned}
\mathrm{ALG} &\leqslant \mathrm{OPT} \cdot \sum_{i=1}^{\mathrm{ALG}} \frac{x_i}{\sum_{j \geqslant i} x_j} \\
&= \mathrm{OPT} \cdot \left[\frac{x_1}{n} + \frac{x_2}{n - x_1} + \frac{x_3}{n - (x_1 + x_2)} + \cdots \right] \\
&\leqslant \mathrm{OPT} \cdot \Bigg[\underbrace{\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n - x_1 + 1}}_{n \text{ total}} \\
&\qquad + \underbrace{\frac{1}{n - x_1} + \frac{1}{n - x_1 - 1} + \cdots + \frac{1}{n - (x_1 + x_2) + 1}}_{x_2 \text{ total}} \\
&\qquad + (x_3) \text{ total decomposed terms for } \frac{x_3}{n - (x_1 + x_2)} + \cdots \Bigg] \\
&\leqslant \mathrm{OPT} \cdot (1 + 1/2 + \cdots + 1/n) = \mathrm{OPT} \cdot \Theta(\log n).
\end{aligned}$$

□

Beginning of 09/09/2024

3.1 Another Type of Rounding: Randomized Rounding

Randomized Rounding on Weighted SET COVER

Previously we talked about viewing VERTEX COVER as a LP problem, from which we can obtain a fractional optimal solution and round it to an integer solution while maintaining an approximation factor of 2. Here, we apply a similar approach, viewing SET COVER as an LP problem, but use a different rounding technique, **randomized rounding**, to recover an integer solution. Clearly, in the LP, we assign a variable x_i to each set indicating whether it's selected:

$$\min \sum x_i \qquad \text{subject to} \qquad \begin{cases} \sum_{i:j\in S_i} x_i \geqslant 1 & \text{for each element } j \in U \\ x_i \geqslant 0 & \text{for each set index } i. \end{cases} \tag{SET COVER LP}$$

By solving the LP we obtain an optimal solution x_i^*. As usual, note that the optimal solution must have $x_i^* \leqslant 1$ since otherwise we can just reduce the value to 1 while further decreasing the objective. That $0 \leqslant x_i^* \leqslant 1$ suggests a probabilistic perspective: maybe we can round x_i^* to 1 with probability x_i^*, i.e., for our integer solution $\tilde{x}_i$,

$$\mathbb{P}(\tilde{x}_i = 1) = x_i^*.$$

This is natural, since intuitively larger values of x_i^* should be more likely to be rounded up to 1.

The good? Expectation-wise, this rounding is just as good as the fractional LP:

$$\mathbb{E} \sum_i \tilde{x}_i = \sum_i \mathbb{E}\tilde{x}_i = \sum_i x_i^*. \tag{7}$$

But of course, this rounding may or may not work — what if (albeit unlikely) our randomized rounding decides to round all variables down, setting $\tilde{x}_i = 0$ for all i? In that case clearly we don't have a set cover. So, an alternate question we can ask is, *what is the probability that something goes bad*? Let us focus on a certain element $j \in U$ and calculate the probability that j is uncovered after rounding:

$$\begin{aligned} \mathbb{P}(j \text{ uncovered after ronuding}) &= \mathbb{P}\Big(\sum_{i:j\in S_i} \tilde{x}_i < 1\Big) = \mathbb{P}(\tilde{x}_i = 0 \text{ for all } i \text{ such that } j \in S_i) \\ &= \prod_{i:j\in S_i} (1 - x_i^*) \leqslant \prod_{i:j\in S_i} \exp(-x_i^*) = \exp\left(-\sum_{i:j\in S_i} x_i^*\right) \leqslant 1/e, \end{aligned} \tag{8}$$

where the first $\leqslant$ used the Taylor expansion[6] $e^{-x} = 1-x+o(x^2)$ and the last one because $\sum_{i:j\in S_i} x_i^* \geqslant 1$ by the feasibility of x_i^*. This is not really impressive: that each element has a probability of $1/e$ being missed after rounding is quite bad. So how do we improve the result, reducing $\mathbb{P}(j \text{ uncovered after rounding})$? Well, one way[7] is to **boost** it by making it more likely to round up. That is, we introduce a boost variable $\alpha > 0$ and replace the rounding scheme

[6] Just a caveat, but for those sets with $\alpha x_i^* \geqslant 1$, if an element falls within such set, it is already deterministically covered, so we can just ignore its analysis. Instead, we only perform analysis on sets that are not deterministically covered. (This is to avoid multiplying by a term $(1 - x_i^*) = 0$ in the product.)

[7] Another way, not covered in lecture (at least for this year), is to repeat the randomized rounding for t times and combine the results $x_i := \max_t \tilde{x}_{i,t}$. This new scheme is t-approximate (since each rounding is optimal in expectation), and $\mathbb{P}(j \text{ uncovered after multi-rounding}) \leqslant 1/e^t$. So the union bound [observe the covergae of each element is not independent, so the answer is not $1-(1-e^{-t})^n$], $\mathbb{P}(\text{some element uncovered}) \leqslant ne^{-t}$, i.e., $\mathbb{P}(\text{output is a valid set cover}) \geqslant 1 - ne^{-t}$, which can be made arbitrarily close to 1.

with

$$\mathbb{P}(\tilde{x}_i = 1) = \min(\alpha x_i^*, 1). \tag{9}$$

Consequently, (8) becomes

$$\mathbb{P}(j \text{ uncovered after rounding}) \leqslant \exp\left(-\alpha \sum_{i:j\in S_i} x_i^*\right) \leqslant 1/e^{\alpha}. \tag{10}$$

Here is the general idea of what follows: we pick a relatively large α so that the rounding is expected to cover most of the elements in U. Then we manually cleanup the missed ones, and since there are only a few elements missed, this would worsen the overall objective significantly. The magic number here is $\alpha = 2\log n$, which makes

$$\mathbb{P}(j \text{ uncovered after rounding}) \leqslant \exp(-2\log n) = 1/n^2.$$

By union bound (note that coverage of elements are *not* independent),

$$\mathbb{P}(\text{some element uncovered}) \leqslant 1/n.$$

So, with probably $1 - 1/n$, randomized rounding directly yields a valid set cover, where the total cost was $\alpha \cdot \mathrm{OPT} = 2\log n \cdot \mathrm{OPT}$. With probability $1/n$ the rounding fails to yield a valid set cover, so we manually fix it by picking one set for each of the missed vertices. The extra cost is high (because we are literally picking sets arbitrarily), but it is certainly bounded by $n \cdot \mathrm{OPT}$ (anything we can possibly pick is bounded by this). So our total expected number of sets picked is

$$2\log n \cdot \mathrm{OPT} \cdot (1 - 1/n) + (2\log n \cdot \mathrm{OPT} + n \cdot \mathrm{OPT}) \cdot (1/n) = (2\log n + 1) \cdot \mathrm{OPT}$$

and the algorithm can be described as follows:

Algorithm 6: SET COVER Randomized Rounding

Input: sets $S_1, \cdots, S_m \subset U$
Initialization: declare variables $\tilde{x}_1, \cdots \tilde{x}_n$
solve SET COVER LP to obtain fractional solutions x_i^*
for each x_i^* **do**
  set $\tilde{x}_i \leftarrow 1$ with probability $\min(2\log n x_i^*, x_i)$
for each element j not yet covered, i.e., $\sum_{i:j\in S_i} \tilde{x}_i = 0$ **do**
  pick any set S_i containing j and set $\tilde{x}_i \leftarrow 1$
return $\tilde{x}_1, \cdots, \tilde{x}_n$ as indicators of whether each S_i is selected

Randomized rounding with some post processing is $\Theta(\log n)$-approximate, not bad.

Randomized Rounding on Weighted SET COVER

What if we introduce weights to sets? That is, the LP now becomes

$$\min \sum w_i x_i \qquad \text{subject to} \qquad \begin{cases} \sum_{i:j\in S_i} x_i \geqslant 1 & \text{for each element } j \in U \\ x_i \geqslant 0 & \text{for each set index } i. \end{cases} \tag{Weighted SET COVER LP}$$

What would change and what not? First, notice that (7) becomes

$$\mathbb{E}\sum_i w_i\tilde{x}_i = \sum_i w_i\mathbb{E}\tilde{x}_i \leqslant \sum_i w_i\alpha x_i^* = \alpha\sum_i w_i x_i^* \tag{11}$$

so the same relation holds. And notice that the computations in (8) do not involves the weights since the rounding is based on α and x_i^* only, so (10) still holds. The only change we need is during post-processing, where this time we choose the *minimum* weight set covering each remaining element. Observe that for each uncovered j, the set we choose to cover j is a lower bound of the *entire* OPT, so the cost of covering n vertices is again bounded by $n\cdot\mathrm{OPT}$. By the same calculating, setting $\alpha = 2\log n$ we see that the following algorithm is $(2\log n+1)$-approximate.

Algorithm 7: Weighted SET COVER Randomized Rounding

Input: sets $S_1,\cdots,S_m \subset U$
Initialization: declare variables $\tilde{x}_1,\cdots\tilde{x}_n$
solve weighted SET COVER LP to obtain fractional solutions x_i^*
for each x_i^* **do**
  set $\tilde{x}_i \leftarrow 1$ with probability $\min(2\log n x_i^*, x_i)$
for each element j not yet covered, i.e., $\sum_{i:j\in S_i}\tilde{x}_i = 0$ **do**
  pick the *minimum* weight S_i containing j and set $\tilde{x}_i \leftarrow 1$
return $\tilde{x}_1,\cdots,\tilde{x}_n$ as indicators of whether each S_i is selected

Remark. The approximation of the probabilistic bounds can also be done using the **Chernoff-Hoeffding bounds**, which states the following.
Let $p_1,\cdots,p_n$ be n independent random coins. Let X_i be the indicator that the i^{th} coin is heads. Then

$$\mathbb{P}\left(|\sum_{i=1}^n x_i - \sum_{i=1}^n p_i| > \epsilon\cdot|\sum_{i=1}^n p_i|\right) < \exp\left(-c\cdot\epsilon^2\cdot\sum_{i=1}^n p_i\right).$$

3.2 A Better Approximation: Improving from $\log n$ to $\log\max_i|S_i|$

In this section, we will replace α with $\log\max_i|S_i|$, i.e., the log of the max size of all sets. By (10) and (10), the rounding step will product an output bounded by $\log\max_i|S_i|\cdot\mathrm{OPT}$, and each element has a probability $1/\max_i|S_i|$ of not being covered by this output. All that remains is to analyze the amount of post-processing one needs to perform.

We first take a look at the unweighted case. By linearity, the expected number of uncovered vertices after randomized rounding is $\leqslant n/\max_i|S_i|$. But notice that $n/\max_i|S_i| \leqslant \mathrm{OPT}$, because *any* solution that covers all n vertices would need this many sets, since the largest set is of size $\max_i|S_i|$ and there are n total elements to cover. Therefore, the $\log\max_i|S_i|$ approximation factor is attained as the additional $o(1)\cdot\mathrm{OPT}$ is ignorable.

How about the expected cost in the weighted case? As usual, for each j, we follow the strategy to choose the minimum cost set covering j. It follows that for each uncovered element j, the expected additional cost is

$\min_{i:j\in S_i} w_i/\max_i|S_i|$. Summing over all j's, an upper bound for the expected post-processing cost is

$$\frac{1}{\max_i|S_i|}\cdot\sum_j \min_{i:j\in S_i} w_i. \tag{*}$$

We claim that magically, (*) is still bounded from above by OPT. To see this, we need to look at the universal set U and think of what OPT is doing. This can be summarized by considering the individual contribution of elements to OPT: if S is a set in OPT and contributes w to the total value, then alternatively each element in S contributes a total of $w/|S|$. Summing over all sets in OPT,

$$\text{OPT} = \sum_{S_i\in\text{OPT}} w_i = \sum_{S_i\in\text{OPT}} |S_i|\cdot\frac{w_i}{|S_i|} = \sum_{\substack{S_i\in\text{OPT}\\ j\in S_i}}\frac{w_i}{|S_i|}.$$

On the other hand, if an element j is covered by S_k in OPT, we also know

$$\frac{w_k}{|S_k|}\geqslant\frac{w_k}{\max_i|S_i|}\geqslant\frac{\min_{i:j\in S_i} w_i}{\max_i|S_i|} = (*).$$

Therefore, the total cost of post processing is $\leqslant$ OPT, which once again shows the overall algorithm is $(1+\log\max_i|S_i|)\sim(\log\max_i|S_i|)$-approximate.

> **Remark.** In the above analysis, instead of looking at the sets S_i, we shifted our attention to the elements that make up these sets. This is a hint for dualization! It is fairly straightfoward to construct the dual of the weighted SET COVER LP: where we assign weights on the elements, subject to the constraint that the total weight assigned to each set shouldn't exceed the weight of that set.
>
> $$\max\sum_j y_j \qquad \text{subject to} \qquad \begin{cases}\sum_{j\in S_i} y_j\leqslant w_i & \text{for each set index } i\\ y_j\geqslant 0 & \text{for each element } j.\end{cases} \qquad \text{(Weighted SET COVER Dual LP)}$$
>
> The cool realization here is that (*) is a feasible solution to the dual (setting $y_j=\min_{i:j\in S_i} w_i/\max_i|S_i|$) and clearly OPT is primal feasible. Therefore, $(*)\leqslant$ OPT.

Beginning of 09/11/2024

3.3 Limitations of LP: Integrality Gaps

Previously, we have seen plenty of examples where we relax an integer-valued optimization problem to fractional LP, solve for the fractional optimal solution, then use some kind of rounding to retrieve an integer solution and make sense of it. In the process of doing so, we derived a chain of inequalities (assuming primal is maximization and dual minimization):

$$D^{\text{opt}}_{\text{integer}}\leqslant D^{\text{opt}}_{\text{fractional}}\leqslant P^{\text{opt}}_{\text{fractional}}\leqslant P^{\text{opt}}_{\text{integer}}\leqslant \text{ALG (our algorithm to be analyzed)}.$$

When analyzing the approximation factor, we try to compare our algorithm to the integral optimum, where we focused exclusively on the last $\leqslant$. However, since this quantity is hard to directly bound, we have resorted to LP rounding, comparing against $D^{\text{opt}}_{\text{fractional}}$, or dual fitting, comparing against $D^{\text{opt}}_{\text{integer}}$. However, both of these induce the extra gap between $P^{\text{opt}}_{\text{fractional}}$ and $P^{\text{opt}}_{\text{integer}}$. We call the ratio $D^{\text{opt}}_{\text{integer}}/D^{\text{opt}}_{\text{fractional}}$ the **integrality gap**. When such gap

exists (i.e., > 1), the rounding factor ($P^{\text{opt}}_{\text{fractional}}$ against ALG) is strictly better than the approximation factor.
As usual, we will revisit the two examples we've seen over and over again, VERTEX COVER and SET COVER, and compute their integrality gaps.

Integrality Gap of VERTEX COVER

Recall that our objective is to minimize $\sum_v x_v$ subject to $x_u + x_v \geqslant 1$ for each edge (u, v). An easy example to show the discrepancy between fractional and rounded solution is by considering a triangle. Clearly, an integer solution would need to pick 2 out of the 3 vertices, hence achieving an objective value of 2. On the other hand, the optimal fractional solution assigns $1/2$ to each vertex, resulting in an objective value of $3/2$. Therefore, for this specific graph, the gap is $2/(3/2) = 4/3$.
More generally, consider a complete graph K_n (i.e. vertices are pairwise directly connected). The integral solution needs to choose $n-1$ vertices, but the fractional solution can again assign $1/2$ to each vertex. Here, the gap becomes $(n-1)/(n/2) = 2(1-1/n)$. Since n is arbitrary, we see that the integrality gap for the problem must be at least 2. In other words,

> It is impossible for any rounding algorithm on VERTEX COVER to achieve an approximation factor better than 2.

And since we showed that threshold rounding indeed achieves an approximation of 2, we conclude that **the integrality gap of VERTEX COVER is 2**.

Integrality Gap of SET COVER

Recall that our objective is to minimize $\sum_i x_i$ such that $\sum_{i:j\in S_i} x_i \geqslant 1$ for each j in the universal set U.
Consider the following instance of set cover. Say we have m sets, $S_1, \cdots, S_m$. For every collection X of $m/2$ sets, define an element x that only belongs to these $m/2$ sets. By Sterling's formula, there are roughly

$$\binom{m}{m/2} \approx \frac{2^m}{\sqrt{\pi m}}$$

elements. By the pigeonhole principle, the optimal integral solution must contain $m/2 + 1$ elements, but it suffices to assign $2/m$ to each element in a fractional solution, resulting in a total weight of $2/m \cdot m = 2$. Therefore, the gap is $m/4$ which, by taking the log of $\binom{m}{m/2}$, becomes $\Theta(\log n)$. Therefore, **rounding in SET COVER with n elements cannot get better than a factor of $\Theta(\log n)$.**

4 Growing Balls (Literally)

4.1 Shortest $s-t$ Path

In this section we consider the shortest $s-t$ path problem. Let $G=(V,E)$ be an undirected graph with edge weights/lengths ℓ_e. Our goal is to simply compute the shortest $s-t$ path. Clearly we've known a fair amount of algorithms, such as Dijkstra or Bellman-Ford.
As usual, let us formulate the shortest $s-t$ path problem as LP:

$$\min \sum_e \ell_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} x_e \geqslant 1 & \text{for all connected sets } S \subset V \text{ with } |S \cap \{s,t\}| = 1 \\ x_e \geqslant 0. \end{cases}$$

Essentially (S, S^c) can be viewed as an $s-t$ cut on G: one contains s and the other t. Of course, we assume S to be connected, so the shortest path "leaves" any such S once and never returns. And $e \in \partial S$ means e crosses the boundary of S, or in more formal expression, $\partial S = \{e=(u,v) : |S \cap \{u,v\}| = 1\}$. The dual can be written as

$$\max \sum_S y_S, \text{ indexed over all connected subset } \textit{cuts } S, \text{ subject to} \qquad \begin{cases} \sum_{S: e \in \partial S} y_S \leqslant \ell_e & \text{for all } e \\ y_S \geqslant 0. \end{cases}$$

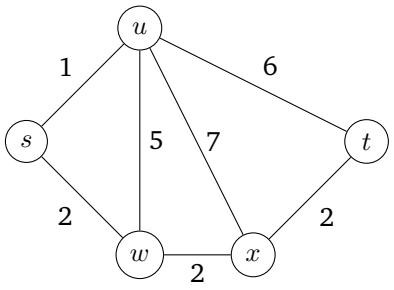
We will use a primal-dual algorithm to solve for the shortest path. Like before, this approach seems unnecessarily convoluted on simple questions, but its power lies in the ease of generalization. Specifically, we will use "**ball growing**" to find a shortest path.

Imagine if we have a ball at s and another at t, and they grow over time. Consider the s-ball and denote it by $B_{s,r}$ where r is the radius. Initially the ball only contains s, i.e., $B_{s,0} \cap V = \{s\}$, so the dual variable this ball corresponds to is $y_{\{s\}}$. Say the shortest edge leaving s is edge (s,w) of length 1. Then, $B_{s,r} \cap V = \{s\}$ as long as $r < 1$, but it jumps to $\{s,w\}$ immediately as r hits 1. This corresponds to the fact that in Dijkstra, after traveling for a distance of 1, we have for the first time reached a node starting from s, i.e., we found the nearest vertex.

At this point, in Dijkstra, for any path continuing on t, we no longer need to consider the first hop (s,t) since we know the shortest path from s to t is directly via this edge. The "settling" of a node via Dijkstra corresponds to saturating the dual constraint along the edge that lead us to this node. In this (s,w) example, at $r=1$, the dual constraint for $e=(s,w)$ becomes tight, so we no longer increase $y_{\{s\}}$ but instead consider a new variable $y_{\{s,w\}}$, starting from $y_{\{s,w\}}=0$. Rinse and repeat.

What about terminal condition? Remember, that as we are growing balls from the s-side we are simultaneously growing another one from the t-side. If we view ball growing as finding the shortest path to reach nodes within a certain radius, then the shortest path should be found when the two balls "collide." Say they collide on an edge (u,v), where the current balls are B_1 and B_2. Then, moments before the collision, $y_{B_1 \cap V}$ and $y_{B_2 \cap V}$ are both increasing, so the dual constraint (u,v) is being pressed by both sides. Therefore, the terminal condition is when (i) an edge is feeling pressure from both ends, and (ii) this edge becomes tight.

To put the above into context, consider this graph, where the shortest path clearly is $s-w-x-t$. When the ball growing reaches radius > 2, the LHS becomes $\{s,u,w\}$ and the RHS $\{t,x\}$. At this point, the dual constraint for (w,x) is being pressured by both $y_{\{s,u,w\}}$ and $y_{\{t,x\}}$. Note that we start increasing $y_{\{s,u,w\}}$ from 0 when radius $r = 2$, so $y_{\{s,u,w\}} = r - 2$. Likewise for $y_{\{t,x\}}$. Since the dual variables are increasing at a uniform rate, the constraint for (w,x) saturates when $y_{\{s,u,w\}} = y_{\{t,x\}} = 1$, corresponding to $r = 3$. So at this point we conclude that the shortest $s-t$ path length is $2r = 6$.



But how do we recover the desired $s-t$ path? Observe that throughout the ball growing process, we saturated $(s,w),(w,x)$, and (x,t), the three edges we want, but in the process of doing so, our balls also saturated excess edges like (s,u). Therefore, by the time the ball growing terminates,

$$\mathcal{F} = \{e \in E : \text{ dual constraint for } e \text{ is tight}\}$$

probably contains more edges than needed, and we need to conduct **reverse deletion**. Observe that if we repeatedly delete edges e such that $\mathcal{F}\backslash\{e\}$ remains feasible, then in the end we will have no excess edges. Also observe that in doing so we recover a *path*, since the edge removal process guarantees that we would have broken any cycle if it exists in $\mathcal{F}$. It remains to argue that the modified $\mathcal{F}$ is the *shortest* $s-t$ path.

> ***Proof that $\mathcal{F}$ is the shortest path.*** We will prove this by showing that the primal and dual objectives equal, and the rest follows by duality theorem.
> Recall that each time we add an edge (u,v), the dual constraint is tight, i.e., $\sum_{S:e\in\partial S} y_S = \ell_e$. Therefore, we can rewrite the primal objective as
>
> $$\sum_{e\in E} \ell_e x_e = \sum_{e\in\mathcal{F}} \ell_e = \sum_{e\in\mathcal{F}} \sum_{S:e\in\partial S} y_S = \sum_S y_S \cdot |\mathcal{F}\cap\partial S|$$
>
> by rearranging based on the number of appearances of y_S.
> Clearly, since $\mathcal{F}$ is a path and S an $s-t$ cut, $|\mathcal{F}\cap\partial S| \geqslant 1$. However, during the contraction process, if two edges e_1, e_2 leave the connected cut S, then one of them will be eventually pruned, because if S, S^c are both connected, then it suffices to keep only one between e_1 and e_2 while also maintaining a valid $s-t$ path. Therefore, $|\mathcal{F}\cap\partial S| = 1$ for all connected cut S, so the primal and dual objectives equal, and the proof of optimality is complete. □

4.2 MSTs & Steiner Trees — From Balls to Moats

Minimum Spanning Trees

We now consider trees, and start off by considering the Minimum Spanning Tree, MST:

> Given $G = (V,E)$ with weighted, undirected edges, fund a subset $E' \subset E$ such that (V,E') is also connected and $\sum_{e\in E'} \ell_e$ is minimized.

Like shortest $s-t$ path, we first formulate the problem as an LP. The objective remains unchanged. As for the constraint, we note that to ensure the tree is connected, for *any* subset $S \subset V$, there needs to be an edge connecting

S with outside. Therefore, the MST can be represented by LP via

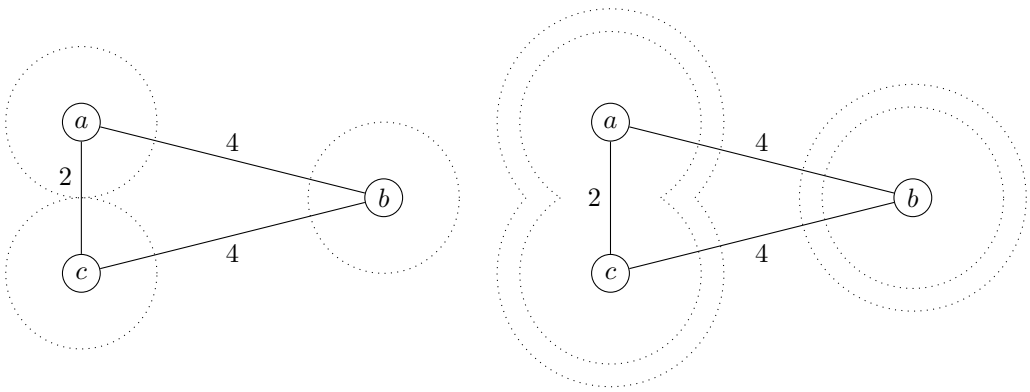
$$\min \sum_e \ell_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} x_e \geqslant 1 & \text{for all nonempty } S \subset V \\ x_e \geqslant 0. \end{cases}$$

Before discussing how to "grow balls" to construct MST, let us first consider its integrality gap. Using the unilateral triangle graph as an example again, we see that an integer solution needs to pick 2 edges resulting in an objective value of 2, while a fractional solution can assign $1/2$ to all three edges. This gives an integrality gap of $4/3$.
More generally, consider a complete graph of n vertices, which under this context is called a ***k*-clique**. The integer solution needs to pick at least $n-1$ edges, setting $x_e = 1$ each. The fractional solution can uniformly assign $1/(n-1)$ to each of the $\binom{n}{2}$ edges, resulting in a total objective value $n/2$. Thus the integrality gap approaches 2 as $n \to \infty$. Hence *any* LP-based algorithm cannot do better than being 2-approximate on MST.

Let us now write down the dual of the MST problem, which again highly resembles that of shortest $s-t$ path:

$$\max \sum_S y_S, \text{ indexed over all subsets } S \subset V, \text{ subject to} \qquad \begin{cases} \sum_{S : e \in \partial S} y_S \leqslant \ell_e & \text{for all } e \\ y_S \geqslant 0. \end{cases}$$

Time to grow balls. In the shortest path problem, we grew two balls simultaneously, one from s and the other t, and we stop growing balls once the balls collide, essentially giving us a viable path connecting s and t. In MST, we want to make *every* vertex connected, not just s and t, so naturally we grow a ball at *every* vertex.



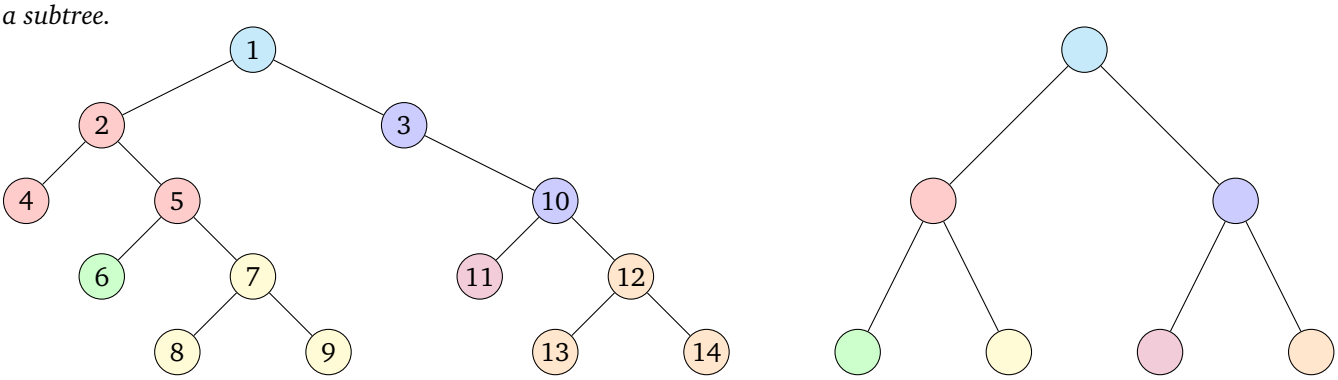
For example, in this simple triangle graph where edge costs are given by labels, we initially grow balls at each vertex, increasing values of $y_{\{a\}}$, $y_{\{b\}}$, and $y_{\{c\}}$ simultaneously. At time $\tau = 1$, the balls on the left collide, at which point we realize the dual constraint for edge (a, c) becomes tight, so we add (a, c) to our primal solution, and in the dual, replace the growing of $y_{\{a\}}$ and $\{y_c\}$ by $y_{\{a,c\}}$. At this point, $y_{\{b\}} = 1$ and $y_{\{a,c\}} = 0$, and they increase at the same rate. Unfortunately after the merging of a and c, the left component no longer resembles a ball. Hence, we sometimes call this process **moat growing** instead. Note that this is essentially Kruskal's algorithm, but stated in a much more convoluted way. Again, the motivation here is that LP allows us to generalize this approach to more complicated problems, such as the computation of Steiner Tree below.

Initially, we grow a ball for each vertex, so we have $|V| = n$ balls. As they iterative marge, we have fewer moats. Finally, our ball/moat growing process terminates once everything is merged into one big component.

Following our previous path, the next question to ask is, after getting $\mathcal{F}$, the set of primal edges whose corresponding dual constraints are saturated, do we need to perform reverse deletion? The answer is no, because $\mathcal{F}$ is a tree, and removing an edge will break the tree-like structure. To see $\mathcal{F}$ is a tree, note that we are always engulfing previously disconnected nodes into the "main" moat, and doing so will never create a cycle.

Clearly, $\mathcal{F}$ is a valid spanning tree. Remember that we said this MST problem has an integrality gap of 2, so no algorithm can be better than being 2-approximate. To show $\mathcal{F}$ is an MST, it suffices to show that it achieves the approximation factor 2.

Proof that $\mathcal{F}$ is an 2-approximate MST. Let us first draw $\mathcal{F}$. Initially, all vertices in $\mathcal{F}$ belong to different balls/moats, but as time passes, they merge. An important observation here is that *any moat at any time is also a subtree.*



For example, consider the tree above, where at some stage, $\mathcal{F}$ is broken into several color-coded maots, each forming its own subtree. The condensed tree $\mathcal{T}$ is shown on the right, where each colored subtree is condensed into one node.

We will now analyzie ΔD versus ΔP but "in hindsight." When the algorithm terminates, we know every edge in $\mathcal{F}$ is tight, so $\sum_{S:e\in\partial S} y_S = \ell_e$. Therefore, after the MSQ terminates,

$$\text{Primal} = \sum_{e\in E} \ell_e x_e = \sum_{e\in\mathcal{F}} \ell_e = \sum_{e\in\mathcal{F}} \sum_{S:e\in\partial S} y_S = \sum_S [y_S \cdot \deg_{\mathcal{F}}(S)],$$

where $\deg_{\mathcal{F}}(s)$ is the number of edge crossing S with respect to $\mathcal{F}$. And of course the dual is always $\sum_S y_S$. This proof is backwards in the sense that we will always keep track of how the double sum $\sum_{e\in\mathcal{F}} \sum_{S:e\in\partial S} y_S$ changes, even though it doesn't mean anything useful when $\mathcal{F}$ isn't finalized. What we *do* know is that this quantity *eventually* will become the primal value.

At time τ, if we increase each dual variable y_S by δ, then the total change is $\Delta D = N\delta$, where N is the current number of condensed nodes (or the number of moats). On the other hand, observe that if S is a subtree in $\mathcal{F}$ and s is the condensed node in $\mathcal{T}$, then $\deg_{\mathcal{F}}(S) = \deg_{\mathcal{T}}(s)$! Therefore,

$$\Delta P = \sum_S \delta \cdot \deg_{\mathcal{F}}(S) = \delta \sum_S \deg_{\mathcal{F}}(S) = \delta \sum_{v\in\mathcal{T}} \deg_{\mathcal{T}}(v).$$

But $\mathcal{T}$ is a condensed tree with N nodes, so $\sum_v \deg_{\mathcal{T}}(v) = 2(N-1)$, and so $\Delta P = 2\delta(N-1)$. Thus $\Delta P/\Delta D \leqslant 2$, and our proof is complete. □

Steiner Trees

Now we consider a more general version of MST, the (minimum) **Steiner Tree**. The formulation is simple — instead of picking a subset of edges to cover all of V, we now want to pick a min-cost subset of edges covering some given

subset $R \subset V$. Clearly, the shortest $s-t$ path is an instance of Steiner Tree where the nodes to connect are $\{s,t\}$. And MST is the instance of Steiner Tree where $R = V$.

Here is where the hard work we devoted into LP algorithm for $s-t$ path and MST pays off: to solve Steiner Tree via LP, we only need minimal adjustment to the problem formulation and the proof.

First, how do we modify the primal constraints? Well, in MST we needed to ensure *all* vertices are connected, so given any subset of vertices, there must be some edge connecting these vertices to the outside. Here in Steiner Tree, we need to enforce something similar. But note that if $S \subset V$ is disjoint from R, or if $S \supset R$, then we don't need to impose any constraints, as such constraints bear no effect on R. Therefore, we only care when $1 \leqslant |S \cap R| < |R|$, namely, when S encloses *some*, but not *all*, nodes of R. And in this case, we require that an edge leaves S.

$$\min \sum_e \ell_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{S: e \in \partial S} y_S \geqslant 1 & \text{for all } S \text{ with } 1 \leqslant |S \cap R| \leqslant |R| - 1 \\ x_e \geqslant 0. \end{cases}$$

And how do we grow balls or moats? Initially, we only grow on terminals (i.e. for $v \in R$), and the algorithm terminates when all terminals are unified into one universal moat. Do observe that reverse deletion is needed in Steiner Tree: recall that we needed to reverse delete excess edges in the shortest $s-t$ path problem, which is a special case of Steiner Tree. The proof that Steiner Tree LP is 2-approximate, however, can be copied verbatim from the previous section, with $\mathcal{F}$ replaced by the pruned $\mathcal{F}'$.

Remark. There exist much simpler 2-approximations for Steiner Tree. One such algorithm is:

- Compute pairwise distances between the terminals (nodes in R).
- Construct an MST on the terminals. From this, recover a Steiner Tree.

To see why this is 2-approximate, let G' be the subgraph of G formed by R. Consider the OPT. One immediate observation is that every leaf node is a terminal node in R, for otherwise we can discard it. Now we use post-order or pre-order traversal to travel along the tree, and we can easily create a cycle, starting from the root, traversing through nodes OPT, and return to the root. In particular, for each pair of terminal nodes u and v, we traveled from u to v once, and another time backwards. Therefore, the cost of this cycle is $2\mathrm{OPT}$. On the other hand, the cost from u to v must be at least the length of the shortest $u-v$ path in G. Therefore, if we simply create a new graph consisting of nodes in R while preserving the pairwise distance they have in G, the MST on the new graph must satisfy

$$\text{MST on new graph} \leqslant \text{ cost of cycle } = 2\mathrm{OPT}.$$

Remark. Better approximations for Steiner Trees also exist. Our *cut covering* LP has an integrality of gap so any analysis based on this LP cannot do better. However, later we will discuss **hypergraphic LP** which, if used on Steiner Tree, gives an approximation factor of $\log 2$.

Beginning of 09/18/2024

4.3 Steiner Forest — When Do We Grow Moats?

In this section, we generalize our problem one step further and consider **Steiner Forests**. Instead of considering a subset R of terminal nodes, the Steiner Forest problem considers a set of terminal pairs $\{(s_i, t_i) : s_i, t_i \in V\}$ and aims to compute a cost minimizing subgraph of G such that each (s_i, t_i) is connected.
Note that this is a direct generalization of the Steiner Tree problem, where a subgraph covers $R = \{v_1, \cdots, v_k\}$ if and only if every pair of form (v_1, v_i), $i \geqslant 2$, is connected.

It is still clear that a cost minimizing subgraph does not admit a cycle, but now it's possible for a feasible solution to be the disjoint union of more than one trees. The intuition, following the Steiner Tree, is that we need to somehow construct Steiner Trees for each "component." The challenge, of course, is we have no idea what defines these components a priori.

The LP formulation for Steiner Forest is similar to that of Steiner Tree, except now we would require that there exists an edge going across *any* cut separating *any* (s_i, t_i) pair. Let us again default to using c_e to represent edge costs. In other words:

$$\min \sum_{e \in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} x_e \geqslant 1 & \text{if } |S \cap \{s_i, t_i\}| = 1 \text{ for some } i \\ x_e \geqslant 0. \end{cases}$$

For convenience, let us rewrite the constraint of S using a **demand** function

$$f(S) = \begin{cases} 1 & \text{if } |S \cap \{s_i, t_i\}| = 1 \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

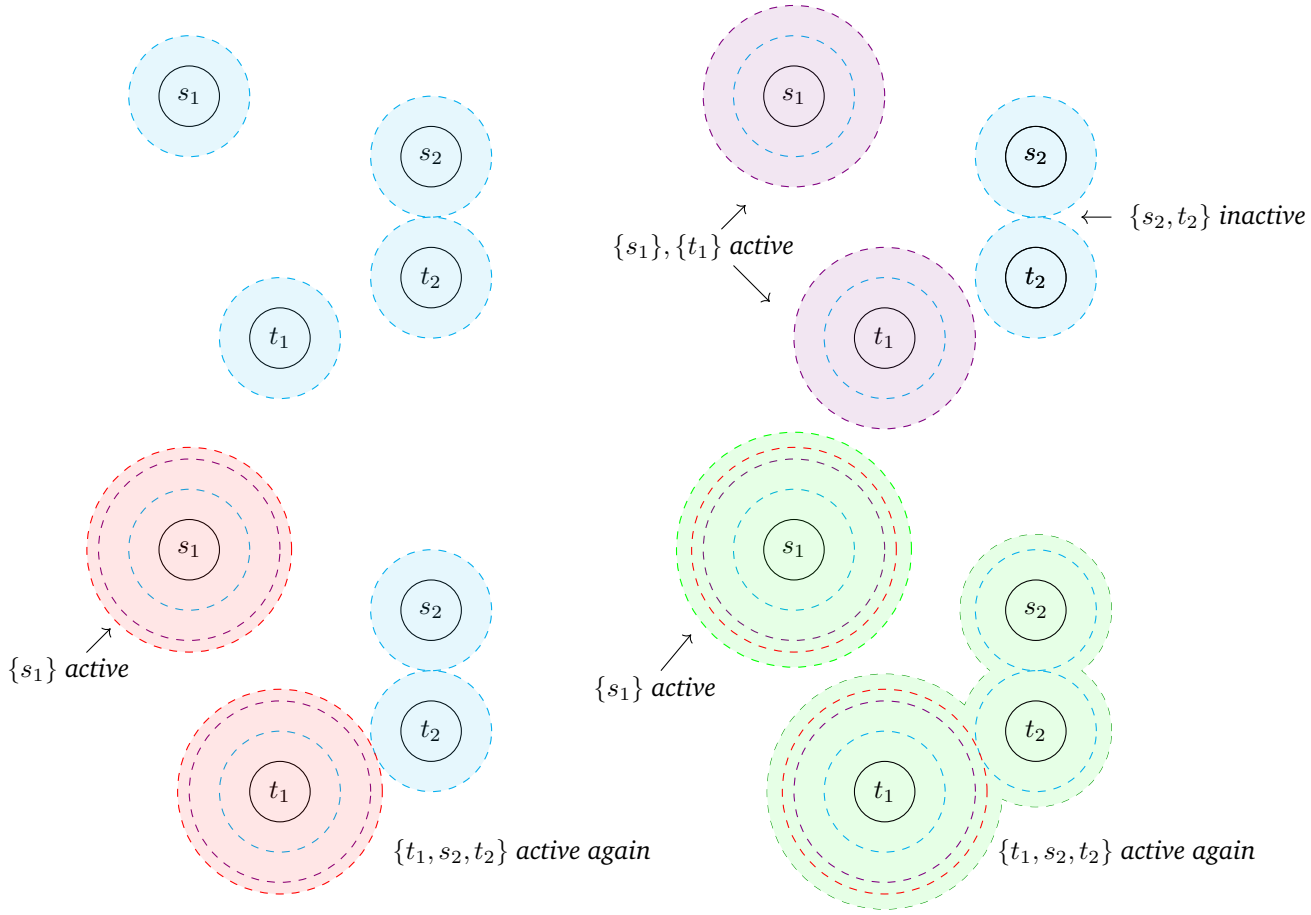
so the primal constraints become $\sum_{e \in \partial S} x_e \geqslant 1$ for all S with $f(S) = 1$. And of course, the dual is

$$\max \sum_{S : f(S) = 1} y_S \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} y_S \leqslant c_e & \text{for all } e \\ y_S \geqslant 0. \end{cases}$$

How do we grow moats? Of course, we start by growing tiny balls around each terminal node s_i, t_i. Consider the following example, where we have two pairs, (s_1, t_1) and (s_2, t_2). All four balls start growing, until at a certain time, $\{s_2\}$ and $\{t_2\}$ touch. In the Steiner Tree problem we would continue growing them, but in Steiner Forest, observe that $f(\{s_2, t_2\}) = 0$, i.e., the newly formed moat has no demand, since it is not crossing any (s_i, t_i) pair (for the same pair). Therefore, in the next step, we only grow $\{s_1\}$ and $\{t_1\}$, and $\{t_1\}$ later meets with $\{s_2, t_2\}$. Now observe that $f(\{t_1, s_2, t_2\}) = 1$ again, so after another merging, this moat becomes active again. Therefore, in the bottom-right figure, both $\{s_1\}$ and $\{t_1, s_2, t_2\}$ are being grown.
In other words, the moat growing process can be summarized as follows:

- Start by growing a ball around each s_i, t_i.
- Whenever two moats (initially balls) S_1, S_2 touch, replace S_1, S_2 with $S_1 \cup S_2$, and grow this new moat $S_1 \cup S_2$ if and only if $f(S_1 \cup S_2) = 1$.

(We say a moat S is **active** if $f(S) = 1$ and **inactive** otherwise.) And of course, every time when a dual constraint e is saturated, we add this edge to the set $\mathcal{F}$ in the primal solution. By the very same token as before, $\mathcal{F}$ does not contain a cycle. Because of the nature of this problem, it may become a forest. We define, as usual, $\mathcal{F}'$ as the set of edges chosen after reverse delete (recall Steiner Tree requires reverse delete; therefore, so does Steiner Forest).



Analyzing the Steiner Forest LP Moat-Growing Algorithm

The proof won't be much different from that of the Steiner Tree problem. However, we note a few complications:

(1) $\mathcal{F}'$ is now a forest, not necessarily a tree.

(2) In each iteration of the dual problem (moat growing), we are only growing along *active* moats — for example, in the above figure, we have never grown $y_{\{s_2,t_2\}}$.

Recall that in the proof of Steiner Tree, after moat growing terminates,

$$\text{Primal} = \sum_{e \in E} c_e x_e = \sum_{e \in \mathcal{F}'} c_e = \sum_{e \in \mathcal{F}'} \sum_{S : e \in \partial S} y_S = \sum_S [y_S \cdot \deg_{\mathcal{F}'}(S)].$$

The Steiner Tree analysis works, based on the observation that the average degree of a node in a (condensed) tree is < 2, so $\Delta P < 2\Delta D$. Here, however, if we look only at *active* moats, it is not clear whether their average degree in $\mathcal{F}'$ is still < 2. So this requires some work.

Fortunately, we can prove that if S is inactive, then $\deg_{\mathcal{F}'}(S) > 1$. To see this, suppose there is only one edge $e_{\mathcal{F}'}$ leaving S in $\mathcal{F}'$. The fact that this edge remains in $\mathcal{F}'$ means it is crucial to connecting some (s_i, t_i) pair (without it, s_i, t_i won't be connected). But then, by definition, we know $|S \cap \{s_i, t_i\}| = 1$, which means S has demand and is active. Contradiction. Therefore, all inactive moats have high degree, ensuring that the average degree of active moats is bounded by 2. The rest of the proof follows, and once again we obtain a 2-approximation.

Remark. The primal LP can in fact be re-written as

$$\min \sum_{e\in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e\in\partial S} x_e \geqslant f(S) \\ x_e \geqslant 0 \end{cases}$$

where $f(S)$ is the demand function.
We say a set function $f : 2^V \to \mathbb{R}$ is **proper** if $f(V) = 0$, $f(S) = f(V\backslash S)$, and for disjoint A, B, $f(A \sqcup B) \leqslant \max(f(A), f(B))$. It can be shown that our Primal-Dual analysis generalizes to any proper function $f(S)$.

Beginning of 09/23/2024

4.4 Survivable Network Design Problems (SND/SNDP)

Note: this section is very terse and may contain more errors than others.
In the Steiner Forest problem we required connectivity between certain pairs of nodes. Namely, for every cut separating some pair (s_i, t_i), there needs to be at least one edge leaving the cut. This can be stated with respect to a binary function $r(u, v)$ such that $r(s_i, t_i) = 1$ and $r(u, v) = 0$ otherwise, and we impose the condition that

$$\sum_{e\in\partial S} x_e \geqslant r(u, v) \qquad \text{for all } S \text{ and all } (u, v) \text{ with } |S \cap \{u, v\}| = 1.$$

To further generalize the problem, what if we allow the requirement function $r(u, v)$ to take arbitrary integer values $\geqslant 0$? Think of this as requiring k-connectivity. For example, $r(u, v) = 3$ means for any cut separating u and v, there needs to be at least 3 edges crossing it. So let us formulate the **Survivable Network Design** (SND) problem as below:

$$\min \sum_{e\in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum\limits_{e\in\partial S} x_e \geqslant r(u, v) \text{ for all } S \text{ and } (u, v) \text{ with } |S \cap \{u, v\}| = 1 \\ x_e \geqslant 0. \end{cases}$$

Further notice that for every cut S, the LHS $\sum_{e\in\partial S} x_e$ just needs to be $\geqslant$ the maximal $r(u, v)$ where (u, v) crosses S. Define

$$r(S) := \max\{r(u, v) : |S \cap \{u, v\}| = 1\},$$

so the first constraint reduces to $\sum_{e\in\partial S} x_e \geqslant r(S)$. Also note that in the current problem, we can "cheat" by allowing duplicates: if previously there is one edge e crossing S, then by increasing the value of x_e we can easily achieve a higher connectivity between S and S^c. This defeats our goal, so to ban such shortcuts, we also require $0 \leqslant x_e \leqslant 1$: when transformed into an integer solution, we either pick each edge e (once) or not.

$$\min \sum_{e\in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e\in\partial S} x_e \geqslant r(S) \text{ for all } S \\ 0 \leqslant x_e \leqslant 1. \end{cases} \qquad \text{(Survivable Network Design)}$$

To solve the SND problem, we use an augmentation approach to iteratively increase the connectivity of cuts until all of them meet their respective requirements. We first find the set of edges that allow us to give connectivity 1 to all cuts with $r(S) = 1$. In the next iteration we pick some additional edges so now cuts with $r(S) = 2$ are also satisfied. Repeat until all requirements are satisfied. This approach is based on a paper by Williamson et al. here. A

high-level algorithm is shown below. To avoid confusing the overall requirement function $r(u,v)$ and $r(S)$ with the intermediate ones (in each iteration), we use r' to denote the latter.

Algorithm 7: SND Augmentation

Given: $G = (V, E)$ that is $[r'(u,v) - 1]$-connected for each (u,v)
Given: a set of edges F with costs c_e
Goal: find a min cost set of edge $F' \subset F$ such that $G = (V, E \cup F')$ is $r'(u,v)$-connected.
repeat

So let us write out the augmentation LP first. The objective, clearly, is to pick a set of min cost edges from F, $\min \sum_{e \in F} c_e x_e$. The constraint is that after augmentation, the number of edges crossing S should be $\geqslant r'(S)$. Hence,

$$\underbrace{\sum_{e \in F \cap \partial S} x_e}_{\text{new edges}} + \underbrace{|E \cap \partial S|}_{\text{existing edges}} \geqslant r'(S) = \max\{r'(u,v) : |S \cap \{u,v\}| = 1\}.$$

Rearranging gives $\sum_{e \in F \cap \partial S} x_e \geqslant r'(S) - |E \cap \partial S|$. Note the RHS $\leqslant 1$ because we are increasing connectivity by 1 in each step. Define the **residual demand** to be

$$f(S) := \max(\text{RHS}, 0) = \max(r'(S) - |E \cap \partial S|, 0).$$

Note this measures how much change is needed in order to satisfy connectivity requirements of S. We see we have another binary function: we either need 1 additioanl edge or not, since our augmentation objective is just to increase connectivity by 1. The augmentation LP can finally be simplified as

$$\min \sum_{e \in F} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in F \cap \partial S} x_e \geqslant f(S) & \text{for each } S \subset V \\ x_e \geqslant 0 & \text{for each } e \in E. \end{cases}$$

4.4.1 Set Functions & The Original $2k$-Approximation

Let $f : 2^V \to \mathbb{R}$ (in our case, binary $f : 2^V \to \{0,1\}$ suffices) be a set function defined on V. We introduce the following notions:

(1) f is a **submodular** function if it satisfies the law of diminishing returns:

$$\text{if } A \subset B \qquad \text{then} \qquad f(A + x) - f(A) \geqslant f(B + x) - f(B).$$

(2) f is a **supermodular** function if the above holds with $\geqslant$ replaced by $\leqslant$.

- f is called **skew supermodular** if one of the following holds:
  - $f(A) + f(B) \leqslant f(A \backslash B) + f(B \backslash A)$, or
  - $f(A) + f(B) \leqslant f(A \cap B) + f(A \cup B)$. [*Note this is equivalent to being supermodular, so with another condition above, skew supermodular is a weaker condition.*]

(3) f is said to be **uncrossable** if:

- $f(V) = 0$,
- $f(S) = f(V \backslash S)$ for all $S \subset V$, and
- if $f(A) = f(B) = 1$ then $f(A \backslash B) = f(B \backslash A) = 1$ **or** $f(A \cap B) = f(A \cup B) = 1$.

We will state, without proof, that $r'(S)$ is skew supermodular, $S \mapsto |E \cap \partial S|$ is submodular, and that the difference between a skew supermodular function and a submodular function is also skew supermodular.
Consequently, the residual demand $f(S)$ is also skew supermodular. It is easy to check that this condition, along with the fact that f is binary, implies that ***f* is uncrossable**.

Enough definitions. Let us see what does an uncrossable f give us. Consider sets of vertices A, B with residual demands 1. This implies either $f(A \backslash B) = f(\backslash B) = 1$ or $f(A \cap B) = f(A \cup B) = 1$. In either case, we see that there exists a smaller set that also has residual demand 1. This means that the **minimal sets[8]with active residual demand must be disjoint**.
But when would a set of vertices have an active residual demand? It happens precisely when the $r'(S)$ demand is not yet satisfied, meaning the connectivity of (S, S^c) needs to be increased. **The minimal sets with active residual remand are precisely, in the language of Steiner Forests, active moats**. We finally established a concrete connection between the primal and the dual.

> **Remark**. The augmentation step itself gives a 2-approximation based on the proof that our Steiner Forest algorithm is 2-approximate. Summing over all augmentation steps, we obtain a $2k$-approximation factor, where $k = \max_e r(e)$.
>
> Later, Goemans et al. observer that we can in fact reverse the augmentation order. First take the nodes with connectivity requirements k and make them 1-connected. In the new graph, the highest demand has now decreased by 1, to $k - 1$. Repeat until done. Note that if we simply take our solution obtained from the first augmentation iteration and scale it by k, then we obtain a feasible solution. This means at the first step, we obtain a 2-approximation w.r.t. the optimum scaled by k, giving us an actual approximation factor of $2/k$. The next iteration will give a factor of $2/(k-1)$, and so on, so overall, with reversed augmentation order, we obtain a $2H_k$-approximation, where H_n is the n^{th} harmonic number. (So we improved from $\Theta(k)$ to $\Theta(\log k)$.)

Beginning of 09/25/2024

4.4.2 Reverse Augmentation: Improving to $\Theta(\log k)$-Approximation

For convenience, let us recall some helpful definitions. Related to demand, we define

$$f(S) = \max r(u, v) \qquad \text{taken over all} \qquad (u, v) \text{ with } |S \cap \{u, v\}| = 1$$

and correspondingly

$$f_{\max} = \max_{S \subset V} f(S).$$

Recall that $f_{\max}$ denotes the maximum demand of any possible cut, and consequently our augmentation algorithm will need to run for a total of $f_{\max}$ times, where in each iteration we will add a set of edges to our intermediate

[8] In the sense that no strict subset also has active residual demand.

solution. Define $\mathcal{F}_{p-1}$ to be the set of edges added in iteration $1, \cdots, p-1$. In the current iteration, our objective is

$$\min \sum_{e \in E \backslash \mathcal{F}_{p-1}} c_e z_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} x_e \geqslant r_p(S) & \text{for all } S \\ z_e \geqslant 0 & \text{for all } e \in E \backslash \mathcal{F}_{p-1}. \end{cases} \tag{SND Reverse LP(p))}$$

(Of course, we have not yet defined $r_p(S)$.) The design choice for r_p is subject to the goal that it decreases any "active" demand by 1 in each iteration. The ultimate goal is that **in each iteration, the maximum demand decreases by 1**. For example, consider three sets in a graph whose demands initially are $(3, 5, 7)$. Our first goal would be to reduce the maximum demand from 7 to 6. But in doing so, maybe the edge we added also affected the demand of, say, the second set, resulting in $(3, 4, 6)$. The third set still has the highest demand, so we further reduce it to $(3, 4, 5)$, and to $(3, 4, 4)$. Repeat until all demands are gone.

Therefore, we want $r_p(S)$ to be an indicator of whether the demand of a set S is currently on the high threshold. The current remand is $f(S)$ minus the number of edges added that now cross S. Call this latter quantity $\delta_{p-1}(S)$. On the other hand, at iteration p (1-indexed), the highest demand threshold should be $f_{\max} - (p-1)$. Therefore,

$$r_p(S) = 1 \qquad \text{if} \qquad \max\left(f(S) - \delta_{p-1}(S), 0\right) = f_{\max} - (p-1) \tag{12}$$

and 0 otherwise. This way, we know that if $r_p(S) = 1$ then we *must* reduce the demand by S. (And of course, we may get bonus demand reductions for any set, like outlined above.)

As usual, it is now time to write the dual for (SND Reverse LP(p)):

$$\max \sum_{S \subset V} r_p(S) \cdot y'_S \qquad \text{subject to} \qquad \begin{cases} \sum_{S : e \in \partial S} y'_S \leqslant c_e & \text{for all } e \in E \backslash \mathcal{F}_{p-1} \\ y'_S \geqslant 0 & \text{for all } S \subset V. \end{cases} \tag{SND Reverse Dual(p))}$$

... and recall that we also have the overall primal and dual LPs. We attach them below for a refresher.

$$\min \sum_{e \in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} x_e \geqslant f(S) & \text{for all } S \subset V \\ 0 \leqslant x_e \leqslant 1 & \text{for all } e \in E. \end{cases} \tag{SND LP}$$

For the dual, we assign variables y_S for each $S \subset V$ and γ_e for each edge e. Because we want $x_e \leqslant 1$, which is equivalent to $-x_e \geqslant -1$, in the dual objective, the coefficients for γ_e are negative.

$$\max \sum_{S \subset V} f(S) - \sum_{e \in E} \gamma_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} y_S \leqslant c_e + \gamma_e & \text{for all } e \in E \\ y_S, \gamma_e \geqslant 0 & \text{for all } S \subset V, e \in E. \end{cases} \tag{SND Dual LP}$$

Yes, now we have two primals and two duals. To analyze the algorithm, let us use ALG, Dual, $\mathrm{ALG}(p)$, and $\mathrm{Dual}(p)$ to denote the objective values of SND primal, SND dual, SND reverse augmentation primal at iteration p, and SND reverse augmentation dual at iteration p, respectively.

Using a similar argument to the forward augmentation, the reverse augmentation is 2-approximate, so $\mathrm{ALG}(p) \leqslant 2 \cdot \mathrm{Dual}(p)$. And by duality, $\mathrm{Dual} \leqslant \mathrm{ALG}$. Therefore, we just need to relate the overall problem to the reverse augmentation.

Claim. For each p, $\mathrm{Dual}(p) \leqslant \mathrm{Dual}/(f_{\max} - (p-1))$. If this claim is true, then

$$\mathrm{ALG} = \sum_{p=1}^{f_{\max}} \mathrm{ALG}(p) \leqslant 2 \sum_{p} \mathrm{Dual}(p) \leqslant 2 \sum_{p} \frac{\mathrm{Dual}}{f_{\max} - (p-1)} = \mathrm{Dual} \cdot \Theta(\log f_{\max}),$$

which completes the approximation ratio analysis for SND when using reverse augmentation.

Proof of claim. Let us first consider a feasible way to assign values to γ_e in (SND Dual LP). To make sure the constraints are satisfied, one way is to define

$$\gamma_e = \begin{cases} \sum_{S: e \in \partial S} y'_S & \text{if } e \in \mathcal{F}_{p-1} \\ 0 & \text{otherwise.} \end{cases}$$

This ensures $y_S = y'_S$ for all S. With such assignment,

$$\begin{aligned}
\mathrm{Dual} &= \sum_S f(S) - \sum_e \gamma_e = \sum_S f(S) - \sum_{e \in \mathcal{F}_{p-1}} \sum_{S: e \in \partial S} y'_S \\
&= \sum_S [f(S) - \delta_{p-1}(S)] \cdot y'_S && (\delta_{p-1}(S) = \text{ size of boundary of } S \text{ w.r.t. } \mathcal{F}_{p-1}) \\
&\qquad [\text{observe that } y'_S \text{ appears} \\
&\qquad \text{once for each edge in } \delta_{p-1}(S)] \\
&= \sum_{S: r_p(S)=1} [f(S) - \delta_{p-1}(S)] \cdot y'(S) + \sum_{S: r_p(S)=0} [f(S) - \delta_{p-1}(S)] \cdot y'(S) \\
&\qquad [\text{observe in (SND Reverse LP}(p)) \\
&\qquad y'_S \text{ only matters if } r_p(S) = 1 \\
&\qquad \text{if } r_p(S) = 0, \text{ can set } y'_S = 0] \\
&\leqslant \sum_{S: r_p(S)=1} [f_{\max} - (p-1)]\, y'_S = \sum_S r_p(S) y'_S = \mathrm{Dual}(p).
\end{aligned}$$

Thus we have completed the final piece of the puzzle, and the claim that reverse augmentation achieves a $\Theta(\log f_{\max})$ overall approximation factor has been proven. □

5 Assignment Problems and Iterative Rounding

5.1 Min-Cost Matching Problem

Consider a problem where we have to assign n tasks to n people, one for each. Let $w_{i,j}$ be the weight or cost of assigning task i to person j. Our goal is to minimize the sum of weights. Using $x_{i,j}$ to represent the action of assigning item i to person j, the LP relaxation is

$$\min \sum_{i,j} w_{i,j} x_{i,j} \qquad \text{subject to} \qquad \begin{cases} \sum_i x_{i,j} = 1 & \text{for all } j \\ \sum_j x_{i,j} = 1 & \text{for all } i \\ x_{i,j} \geqslant 0. \end{cases}$$

We state, without proof, some important facts about **extreme point solutions**. Given a linear LP, the feasible solutions form a polytope P. A feasible solution x is said to be an extreme point solution if for any $y \neq 0$, either $x + y \notin P$ or $x - y \notin P$. The important property is that *if the original LP has m variables and x is an extreme point solution, then x has m tight linearly independent constraints.*

Back to our problem: we have $2n$ constraints, one for each i, and one for each j. But notice that we only have $2n-1$ nontrivial linearly independent constraints, for if we satisfy any $2n-1$ constraints, the last one is automatically satisfied by noting that $\sum_i \left[\sum_j x_{i,j}\right] = \sum_j \left[\sum_i x_{i,j}\right]$. On the other hand, we have $m = n^2$ variables, one for each edge.

Now, consider the concept of extreme points in the context of this problem. For an extreme point solution, there must be $n^2 - (2n-1)$ variables whose corresponding $x_{i,j}$ equal 0, as they are not involved in any active constraints. The remaining $2n-1$ tight active variables will form a sparse assignment configuration. There are n people and $2n-1$ edges, and because the assignment is valid, it must be the case that one person, say person j, has degree 1 to, say task i, and everyone else degree 2. If we remove this pair $(\text{task } i, \text{person } j)$ from consideration, then recurse, considering the subproblem with $n-1$ people and $2(n-1)-1$ tight constraints. This process is called **iterative rounding**.

5.2 Generalized Assignment Problems & Budget Allocation

Beginning of 09/30/2024

Generalized Assignment Problems (GAP)

In this part, we consider a direct generalization of the assignment problem. Suppose that we are to assign n jobs to m agents, where we also have the following quantities defined:

- $c_{i,j}$ measures the cost of assigning job j to agent i,
- $P_{i,j}$ measures the processing time of agent i working on job j, and
- T_i is the total available time agent i has.

The goal, naturally, is to produce a feasible solution of minimum cost such that all jobs are processed:

$$\min \sum_{i,j} c_{i,j} x_{i,j} \qquad \text{subject to} \qquad \begin{cases} \sum_i x_{i,j} = 1 & \text{for each job } j \in [n] \\ \sum_j P_{i,j} x_{i,j} \leqslant T_i & \text{for each agent } i \in [m] \\ x_{i,j} \geqslant 0. \end{cases} \qquad \text{(GAP LP)}$$

However, notice that this LP can have arbitrarily large integrality gaps: consider one job and m agents, such that for agent 1, processing time is 1 but cost is some large M, and for *all* other agents, cost is 0 but processing time is $m-1$. Further assume that all agents have available time $T_i = 1$. This would force the integral solution to assign the job to agent 1, whereas in the integral solution, a much better alternative would be to assign $1/m$ fraction of the job to each agent.

So what's the problem here? We should force the fractional solution to also discard assignments that we know are impossible: *if $P_{i,j} > T_i$, we don't assign job j to agent i*. So we need some **preprocessing**: remove all $x_{i,j}$ (or set it to 0) where $P_{i,j} > T_i$.

A known lower bound to the approximation ratio of the **makespan** of this problem is $3/2$, but it is not known whether it is possible to achieve any approximation ratio < 2. In what follows, we provide a 2-approximation algorithm, i.e., find an assignment satisfying $\sum_j P_{i,j}x_{i,j} \leqslant 2T_i$. Note the difference here: we are relaxing *constraints* by a factor of 2, not the *objective* value.

Designing a 2-approximate makespan GAP Solution. Let $\mathcal{F}$ be the solution set (i.e. the set of variables $x_{i,j}$ set to 1), initially empty. Like previously, we will start with x^*, an extremely point solution. Recall that this means there are $m+n$ variables and $m+n$ tight constraints, one for each $j \in [n]$ and one for each $i \in [m]$.

CASE 1. If $x^*_{i,j} = 0$ for some i, j, we can simply remove these variables from the LP without causing any change.

CASE 2. If $x^*_{i,j} = 1$, then this means an assignment of job j to agent i has been made. So we are done with job j and also need to update agent i's remaining availability. That is, (i) add the assignment (job j, agent i) to $\mathcal{F}$, (ii) remove job j from the set of jobs, and (iii) update $T_i \leftarrow T_i - P_{i,j}$.

CASE 3. Otherwise, $x^*_{i,j} \in (0,1)$ for all i, j. Recall that there are $m+n$ such tight constraints. We now check the degrees of each job or agent vertex. Clearly, there must *not* be a degree 1 job, for otherwise the job constraint $\sum_i x_{i,j} = 1$ implies that $x^*_{i,j} = 1$ for some i, j. But what about the agents? The previous two conditions took care of edge cases, so we may assume each remaining agent has degree $\geqslant 1$. We use d_i to denote the degree of agent i.

(3.1) Suppose $d_i = 1$ for some i. Suppose this agent is assigned job j. Consider the following sequence of actions: (i) remove j from the set of jobs, (ii) add (job j, agent i) to $\mathcal{F}$, and (iii) remove everything related to agent i. From now on, agent i will never be updated again. What is the total time we assigned to agent i, compared against T_i? Previously, we may have updated T_i according to CASE 2, but the total deduction does not exceed T_i because x^* is a feasible solution, and CASE 2 is invoked only if $x^*_{i,j} = 1$. Thus, for the previous deductions, we have $\sum_j P_{i,j} = \sum_j P_{i,j}x^*_{i,j} \leqslant T_i$. For our current assignment in (3.1), we also know that our final assignment has price $P_{i,j} \leqslant T_i$ (directly by original LP's post-processing condition), so the total runtime on agent i is bounded by $2T_i$.

(3.2) The harder case is if every agent i has degree $d_i \geqslant 2$. In this case we can view E, the set of edges in the graph, as a set of cycles. And because of matchings, these cycles are all even. Further, the number of agent and jobs are equal in this case, say k.

We claim that **there exists an agent i with $d_i = 2$ and $\sum_j x^*_{i,j} = 1$.** This is obvious, since among all existing $x^*_{i,j}$'s,

$$\sum_j \overbrace{\sum_i x^*_{i,j}}^{=1} = k = \sum_i \sum_j x^*_{i,j}.$$

For convenience, consider agent i and the two adjacent jobs, j_1 and j_2. We perform the following actions:

- remove j_1, j_2 from the set of remaining jobs,
- add both assignments (job j_1, agent i) and (job j_2, agent i) to $\mathcal{F}$, and
- remove agent i from the set of remaining agents.

Suppose agent i has quota T_i' remaining right before the above actions are carried out. This means that all previous assignments to agent i, like in (3.1), must have been performed in CASE 2, and therefore have a total cost bounded by $T_i - T_i'$, i.e., with respect to the old $\mathcal{F}$,

$$\sum_{j:(i,j)\in\mathcal{F}} P_{i,j} \leqslant T_i - T_i'.$$

On the other hand, it is clear that $\max(P_{i,j_1}, P_{i,j_2}) \leqslant T_i$ by feasibility, and

$$\begin{cases} x_{i,j_1}P_{i,j_1} + x_{i,j_2}P_{i,j_2} \leqslant T_i' & \text{(by feasibility)} \\ x_{i,j_1} + x_{i,j_2} = 1 & \text{(by claim above)} \end{cases} \Rightarrow \min(P_{i,j_1}, P_{i,j_2}) \leqslant T_i'.$$

Agent i will be permanently removed after the actions listed in the bullet points above, and it will have worked for a total of

$$\sum_{j:(i,j)\in\mathcal{F}} P_{i,j} + P_{i,j_1} + P_{i,j_2} \leqslant (T_i - T_i') + T_i' + T_i = 2T_i.$$

This completes the proof that our new assignment has is at worst 2-approximate with respect to makespan.

But what about the objective values $\sum_{i,j} c_{i,j}x_{i,j}$ which we never talked about? The cool idea here is that all of our previous cases can be done simply by removing variables and constraints, without modifying any actual value. In CASE 1, an unused assignment $x_{i,j}^* = 0$ is as if it didn't exist at all, so we can simply remove the variables. In CASE 2, once an assignment $x_{i,j}^* = 1$ has been made, we also no longer care about the corresponding variables. Finally, in each iteration of CASE 3, we are essentially removing a constraint imposed by agent i. To sum up, another way to implement this algorithm is by dynamically updating the set of jobs and agents until none is left, while keeping the LP inequalities verbatim. Under this perspective, the algorithm is just relaxing constraints, so certainly the overall cost won't deteriorate. Therefore, we have shown that we have found an assignment whose cost is optimal while makepsan is 2-approximate. □

Budgeted Allocation

Now we consider the "opposite" of GAP. The **budget allocation** problem instead wants to maximize the objective value. Say we have an auction with again n auction items, indexed by j, and m buyers, indexed by i. The value $b_{i,j}$ measures the bit buyer i is willing to place on item j, and each buyer i has a total budget B_i. Clearly, no one is allowed to overspend, and the goal for the auction organizer is to maximize the money raised in the event:

$$\max \sum_{i,j} b_{i,j}x_{i,j} \qquad \text{subject to} \qquad \begin{cases} \sum_i x_{i,j} \leqslant 1 & \text{for each item } j \\ \sum_j b_{i,j}x_{i,j} \leqslant B_i & \text{for each buyer } i \\ x_{i,j} \geqslant 0. \end{cases} \qquad \text{(Budgeted Allocation)}$$

Just like the naïve GAP, we have a problem: imagine if everyone is broke with budget 1 but there is one expensive

item. The fractional solution would give everyone "fractions" of the item, but the integer solution would simply deny everyone from buying. Therefore, just like before, we assume $b_{i,j} > B_i$ for each i, j.

Now let x^* be an extreme point solution to the budgeted allocation problem, and we will use a similar rounding scheme to cover an assignment. Let $\mathcal{F}$ be the set of assignments, initially empty.

CASE 1. If $x^*_{i,j} = 0$, like before, simply remove the variable $x_{i,j}$.
CASE 2. If $x^*_{i,j} = 1$, then buyer i buys item j, so assign (item j, buyer i) to $\mathcal{F}$, and update buyer i's remaining budget by $B_i \leftarrow B_i - b_{i,j}$.
CASE 3. Otherwise, $x^*_{i,j} \in (0, 1)$ for all i, j. Because x^* is an extreme point solution, there are $|E| = m + n$ variables and $m + n$ constraints. Based on the degrees of vertices, we once again have two cases: either every vertex has degree 2, or maybe there is a degree 1 vertex.

(3.1) If every vertex has degree 2, by the nature of matching, the induced graph consists of even cycles where notes alternate between items and buyers. We use a technique known as **cycle pushing**. For each cycle $C = u_1 \to v_1 \to \cdots \to u_k \to v_k \to u_1$ [where we use u's and v's to represent the bipartiteness], consider two new cycles, C' defined by pushing $+\delta$ on $u_1 \to v_1$, $-\delta$ on $v_1 \to u_2$, $+\delta$ on $u_2 \to v_2$, and likewise, C'' defined by pushing $-\delta$ on $u_1 \to v_1$, $+\delta$ on $v_1 \to u_2$, $-\delta$ on $u_2 \to v_2$, and so on.

It is immediately clear that $(C' + C'')/2 = C$. This means at least one of the directions along which we push alternating δ weights will result in a new cycle that is no worse than the original one. Fix this direction and make δ as large as possible so that some edge now becomes 0 or 1. At this point, we have converted C into an acyclic path that is at least as good as original. This leads to the following case, which we will WLOG assume is what happens in CASE 3:

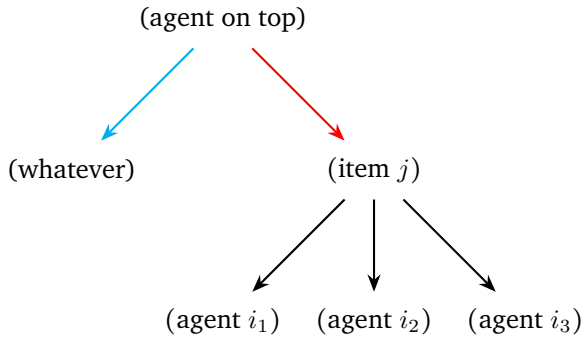
(3.2) **The graph induced by any extreme point solution x^* is acyclic.**

Beginning of 10/02/2024

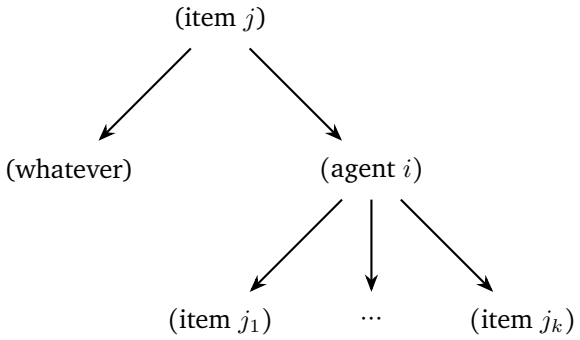
In other words, we assume that $\{(i, j) : x^*_{i,j} > 0\}$ is a forest. For the following analysis, we restrict our attention to one single tree, say T, and think of how to make progress on it. The idea is to look at leaf agents.

Observe that if we have a leaf agent i, we can always just assign the parent item j to this agent and be done with item j. This assignment is valid because our preprocessing ensured $b_{i,j} = \min(b_{i,j}, B_i)$. What if item j has three leaf children, i_1, i_2, i_3? Intuitively, we should make the assignment to the agent with the highest bid.

But there is one more complication. If item j has a parent node, that node is also an agent. What if the highest bid comes from this parent agent (red edge)? Note that this agent may also have some other bidding (represented by the blue edge) affecting their available budget.



We put aside the above observation temporarily. Let us also consider the reverse case, where items become leaves, and we want to analyze the buying actions of an agent i. Of course, to maximize the revenue generated by agent i, we want to let agent i buy as many items incident to them as possible, but clearly the budget B_i imposes a constraint.



Quoting CASE 2, let us first consider an extreme special case: that $x^*_{i,j_\ell} = 1$ for all edges between agent i and children j_ℓ. Certainly, in this case, that $x^*_{i,j_\ell} = 1$ implies that an assignment (i, j_ℓ) can be made for all ℓ without violating the budgeting constraint B_i. Now let us consider the change to the total revenue (objective) after assigning all leaf items to agent i. Originally, revenue generated by agent i according to x^* is

$$R_i = \sum_{\ell=1}^{k} b_{i,j_\ell} x^*_{i,j_\ell} + b_{i,j} x^*_{i,j} = \sum_{\ell=1}^{k} b_{i,j_\ell} + b_{i,j} x^*_{i,j},$$

whereas after the assignment, the revenue obtained from the parent item j is now the minimum between $b_{i,j}$ and $B_i - \sum_{\ell=1}^{k} b_{i,j_\ell}$ due to an updated budget constraint. So the new revenue is

$$R'_i = \sum_{\ell=1}^{k} b_{i,j_\ell} + \underbrace{\min\left(b_{i,j}, B_i - \sum_{\ell=1}^{k} b_{i,j_\ell}\right)}_{:=b'_{i,j}} x^*_{i,j}.$$

Claim. $R'_i \geqslant R_i/2$.

Proof of claim. Observe that $B_i \geqslant R_i$ because clearly the $B_i \geqslant \sum_{j \text{ adjacent}} b_{i,j} \geqslant R_i$.

If $\sum_{\ell=1}^{k} b_{i,j_\ell} \geqslant B_i/2 \geqslant R_i/2$ then there is nothing to show. Otherwise, we observe that if $\sum_{\ell=1}^{k} b_{i,j_\ell} < B_i/2$, then

$$b'_{i,j} \geqslant \min(b_{i,j}, B_i/2) \Rightarrow b'_{i,j} x^*_{i,j} \geqslant \frac{b_{i,j} x^*_{i,j}}{2}.$$

END OF PROOF OF CLAIM

Notice that *we never relied on the value of* x^*_{i,j_ℓ} in the proof above. Therefore, we may relax this assumption and still get the result that $R'_i \geqslant R_i/2$. That is, by applying this assignment argument to every leaf item in every iteration, **we get a 2-approximation from this analysis** as our revenue gained for each item is at least $1/2$ of the optimal value gained from that item.

5.3 SND & Iterative Rounding: a 2-Approximation

In this section we will revisit the SND problem, for which the method of iterative rounding is originally designed. A result due to Jain [1998], this iterative rounding algorithm for the first time reduces SND approximation ratio

down to a constant factor (much better than $\Theta(\log f_{\max})$ as we mentioned before).

First, let us recall the problem formulation: given a pair (u, v) of vertices, the function value $r(u, v)$ denotes the requirement/demand of the number of paths between the two endpoints. Each edge is weighted with cost c_e, and the LP is

$$\min \sum_{e \in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \partial S} x_e \geqslant r(S) = \max_{\substack{u,v \\ |S \cap \{u,v\}|=1}} r(u,v) & \text{for all } S \subset V \\ 0 \leqslant x_e \leqslant 1 & \text{for all } e \in E. \end{cases}$$

Furthermore, we will assume that the demand r is integer valued. Recall that in the primal-dual analysis, we used the fact that r is skew supermodular; that if we have a partial solution S, then the weighted sum of S, then the cardinality function $S \mapsto |E \cap \partial S|$ is submodular; and that the residual demand

$$f(S) := \max(r(S) - |E \cap \partial S|, 0)$$

is also skew supermodular. We slightly generalize the notion of $|E \cap \partial S|$. Instead, let y be a set of solutions, i.e., a set of edges whose values are $y_e \in \{0, 1\}$. Then define $\varphi(y, S) = \sum_{e \in \partial S} y_e$, the size of $E \cap \partial S$ with respect to the edges chosen by y. This are the edges with $y_e = 0$. This still gives a submodular function, and consequently, the generalized residual demand (w.r.t. any partial solution y)

$$f_y(S) = \max(f(S) - \varphi(y, S), 0)$$

is skew supermodular for any $y \in \{0, 1\}^{|E|}$. This is the one and only known fact about $f_y(S)$ throughout the rounding scheme (the solutions themselves may become barely interpretable).

Now, looking back at the primal-dual approach, we see that the primal-dual algorithm uses the LP in the analysis but not the LP itself in the solution. Here, iterative rounding starts with an LP, so we need to solve it. That is, we have one more obstacle: we need to know we *can* solve the LP a priori. This is not immediately clear, since the number of constraints is exponential w.r.t. set size. The fix to this is via a **separation oracle**. The high-level idea is, although we cannot efficiently and explicitly check if a solution is feasible (because there are $\sim 2^{|E|}$ constraints), we can derive a polynomial-time **verifier**/certificate because we only have polynomial number of variables. In this LP, specifically, the verifier can be implemented using Max-Flow: for *each* pair (u, v), run a Max-Flow algorithm from u to v and see if it admits a flow with value $\geqslant r(u, v)$. If this holds for all pairs (u, v) then we know all constraints are satisfied. Else, the Min-Cut would return a set S whose SND LP constraint is violated.

As usual, let x^* be an extreme point solution. If $x^*_e = 0$ then we can simply ignore the variable edge e. If $x^*_e = 1$, then essentially edge e has been chosen, so we set $y_e = 1$ (i.e., add edge e to the partial solution set y) and update the residual demands $f_y(S)$ for each S [note even though there are expnentially many sets S, the total number of updates is polynomial w.r.t. graph size because we only modified one edge]. Otherwise, let us assume that $x^*_e \in (0, 1)$ for each e. That is, none of the constraints of form $0 \leqslant x_e \leqslant 1$ are currently tight.

Recall that the number of linearly independent tight constraints equals that of variables for an extreme point solution. This is known as the **rank lemma**. We have $|E| = m$ variables, so we have m linearly independent tight constraints of form $x^*(S) = f_y(S)$.

Next up, we will inspect sets whose demand constraints are currently tight and analyze their structures. In particular, we show the following:

Claim. There exists a **laminar** family of sets satisfying the rank lemma with tight constraints $x^*(S) = f_y(S)$.

(A family of sets is laminar if any two sets are either disjoint or one is a subset of the other. Note that applying topological hierarchy on a laminar family of sets gives a tree.)

To establish this lemma, we will take a slightly different perspective of sets. In particular, we want to use linear independence by relating sets with vectors. To this end, we associate each set S with a **characteristic vector** $\chi_S \in \{0,1\}^m$ such that $\chi_S(e) = 1$ iff $e \in \partial S$. In other words, if we were to enumerate all edges $e \in S$, then the corresponding component in χ_S is 1 iff e is in the cut ($e \in \partial S$).
Let $\mathcal{T}$ be the collection of the m independent tight constraints according to the rank lemma. Intuitively, this means $\chi_{\mathcal{T}} = \{\chi_S : S \in \mathcal{T}\}$ is also a set of linearly independent vectors. Therefore, $\chi_{\mathcal{T}}$ spans $\mathbb{R}^m$.

Now we prove the claim. Let $\mathcal{L}$ be a maximal laminar subset of $\mathcal{T}$. If $|\mathcal{L}| = m$ then we are done. Otherwise, if $|\mathcal{L}| < m$, then $\operatorname{span}(\mathcal{L})$ is a strict subspace of $\mathbb{R}^m$, and in particular, there exists $S \in \mathcal{T}$ with $S \notin \operatorname{span}(\mathcal{L})$.

Beginning of 10/07/2024

Before we continue, now is a good time to take a step back and review the various set functions we have previously discussed. We state, without proof, the following (most of which have already been stated before). To simplify notations slightly, we will replace $x^*(S)$ by $x(S)$, the sum of the set of edges crossing S, as well as $f_y(S)$ by $f(S)$, the residual demand.

(1) $x(A) + x(B) = f(A) + f(B)$.

(2) $x(A) + x(B) \geqslant x(A \backslash B) + x(B \backslash A)$.

(3) $x(A) + x(B) \geqslant x(A \cap B) + x(A \cup B)$, and

(4) One of the following holds due to skew supermodularity:

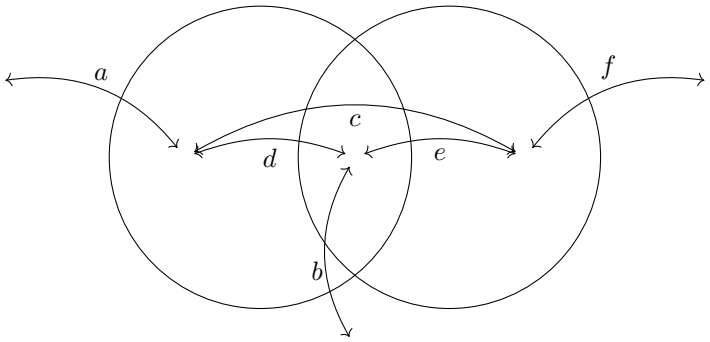
(4a) $f(A) + f(B) \leqslant f(A \backslash B) + f(B \backslash A)$, or

(4b) $f(A) + f(B) \leqslant f(A \cap B) + f(A \cup B)$.

These are a *lot* of inequalities to digest, so let us consider arbitrary sets A, B, and categorize all types of edges by checking the location of their endpoints. Because two A, B partition the space into $A \cap B, A \backslash B, B \backslash A$, and $A^c \cap B^c$, we have $\binom{4}{2} = 6$ types of edges, labelled below. We first assume that

Case 1: (4a) holds. Then (1), (2), and (4a) imply that

$$x(A) + x(B) = x(A \backslash B) + x(B \backslash A).$$



Now note that the LHS contains $\{a, b, c, e\}$ and $\{b, c, d, f\}$, whereas the RHS contains $\{a, c, d\}$ and c, e, f. Equating both sides, we see that the total contribution of (twice times) edges of type b is 0. Since $x^*_e \neq 0$ by assumption, we conclude that no edge of type b exists. So really, the graph looks like the following:
The cool realization is that if no edges connecting $A \cap B$ to $A^c \cap B^c$ exist, then the characteristic vectors satisfy

$$\chi_A + \chi_B = \chi_{A \backslash B} + \chi_{B \backslash A}.$$

Likewise, in CASE 2 where (4b) holds, using (1), (3), and (4b), we would obtain

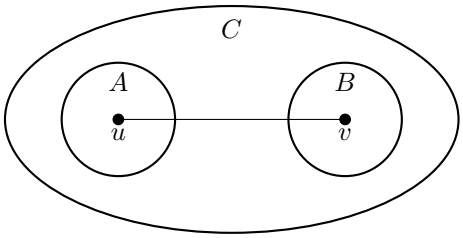
$$\chi_A + \chi_B = \chi_{A\cap B} + \chi_{A\cup B}.$$

Let us now return to the discussion of laminar sets, and assume $|\mathcal{L}| < m$ with $S \in \mathcal{T}\backslash\text{span}(\mathcal{L})$. WLOG, let us assume that S crosses the *fewest* sets in $\mathcal{L}$ (note this quantity is positive, and S exists: for otherwise S would be disjoint from $\mathcal{L}$, and $\mathcal{L}\cup\{S\}$ is also laminar).
Let $S' \in \mathcal{L}$ be a set that intersects S. We assume (4a) holds, so $\chi_S + \chi_{S'} = \chi_{S\backslash S'} + \chi_{S'\backslash S}$. The other case is analogous. Then, by replacing S, S' with $S\backslash S'$ and $S'\backslash S$, respectively, the number sets that now cross S must have been reduced: sets become smaller, so certainly the number of crossings should decrease. We then repeat this process until no more sets cross S. But now, based on our previous observation, this means $\mathcal{L}$ must span the space of all tight sets, so $|\mathcal{L}| = m$, and we are done.

To wrap up the proof, we will use what is known as the **total counting argument**. Our goal is to prove the following claim:

> Anytime during the algorithm, there exists an edge e with $x_e \geqslant 1/2$ to allow us to round up to 1. This ensures an overall 2-approximation ratio.

To show this, we suppose not, that $x_e < 1/2$ for all edges e. We will use a "token distribution scheme," where in each iteration, we assign one token to each edge e, and ask it to distribute tokens to three specific sets:



- distribute x_e tokens to the smallest set $A \in \mathcal{L}$ containing u;
- distribute x_e tokens to the smallest set $B \in \mathcal{L}$ containing v; and
- distribute $1 - 2x_e$ to the smallest set $C \in \mathcal{L}$ containing both u and v. Note because $\mathcal{L}$ is laminar, such set contains both A and B.

We will prove several important claims:

Claim 1. Some tokens are left undisributed.

Proof. Now let S be any maximal laminar set (this just means no other set contains it). The set exists because the set characteristic vectors χ_X are linearly independent and hence nonzero (or because the constraint w.r.t. S is tight). What about an edge $e = (u, v) \in \partial S$, with $u \in S, v \notin S$? Well, the token redistribution scheme asks e to give x_e tokens to S, and also x_e tokens to the smallest set containing v. But no set contains both u and v by the maximality of S, so e is left with $1 - 2x_e$ undistributed tokens. □

Claim 2. Each set gets $\geqslant 1$ tokens via this distribution scheme.

Proof. Let $S \in \mathcal{L}$ be given, and let $T_1, \cdots, T_k$ be the collection of laminar sets inside S (maybe there is none, but the proof works as-is). Call the latter *child sets*. There are four types of edges that we are interested in:

(a) $e = (u, v)$ where $u \in T_i, v \in S\backslash\bigcup T_i$, i.e., from a vertex in a child set to one in $S\backslash\bigcup T_i$;

(b) $e = (u, v)$ where $u \in T_i, v \in T_j$, i.e., from one child set to another;

(c) $e = (u, v)$ where $u \in S\backslash\bigcup_{i\in} T_i$ and $v \notin S$, i.e., from a vertex in $S\backslash\bigcup T_i$ to outside S; and

(d) $e = (u, v)$ where $u \in T_i$ and $v \notin S$, i.e., from a child set to outside.

By appealing to the definition of the token redistribution scheme, we see that the amount of token S receives can be broken down as follows:

(a) For an edge of this form, S receives $x_e + (1 - 2x_e) = 1 - x_e$ (the other x_e token are given to some T_i).

(b) S receives $1 - 2x_e$ (since each endpoint gives x_e to some T_i).

(c) S receives only x_e, from the one endpoint in S.

(d) S receives nothing, since one endpoint is outside of S, and the other gives x_e to some T_i instead of S.

Summing over everything, the total token S gained is

$$(\text{type } a) + (\text{type } b) + (\text{type } c) + (\text{type } d) = \sum_{\text{type } a} (1 - x_e) + \sum_{\text{type } b} (1 - 2x_e) + \sum_{\text{type } c} x_e + 0. \tag{*}$$

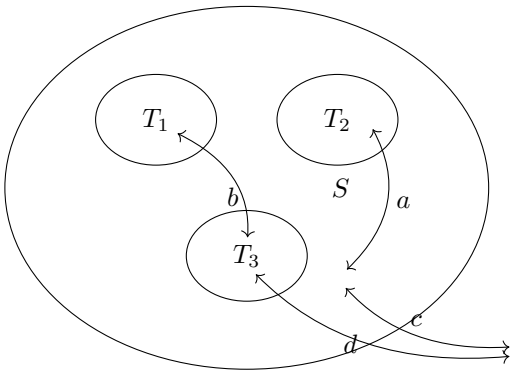
Observe that for any given set S, we must have at least one non-type d edge, for otherwise the set characteristic vector χ_S can be expressed as a linear combination of the χ_{T_i}'s, violating linear independence. Therefore, (*) is strictly positive!

To show (*) $\geqslant 1$, it suffices now to show that it is integer-valued. Clearly, rearranging gives

$$\begin{aligned}(*) &= |\{\text{type } a \text{ edges}\}| + |\{\text{type } b \text{ edges}\}| + \left(\sum_{\text{type } c} x_e - 2 \sum_{\text{type } b} x_e - \sum_{\text{type } a} x_e \right) \\ &= \text{integer} + \sum_c x_e + \sum_d x_e - \left(\sum_a x_e + 2 \sum_b x_e + \sum_d x_e \right) \\ &= \text{integer} + f(S) - \sum_{T_i \subset S} f(T_i) = \text{integer}.\end{aligned}$$

This completes the proof that every set gets $\geqslant 1$ tokens in each iteration via redistribution. □

So if $x_e < 1/2$ for all edges e, with the help of the two claims, we inevitably arrive at a contradiction: we have $m = |E|$ tokens to distribute, and by the rank lemma we also have m sets to receive these tokens. Some tokens were lost, but somehow all sets end up receiving $\geqslant 1$ tokens, resulting in a total of $\geqslant m$ tokens, which is absurd. Therefore, during any stage, there exists some $x_e \geqslant 1/2$. We simply round it up to 1 and repeat. Clearly, this **iterative rounding** approach achieves the claimed 2-approximation factor.



Beginning of 10/09/2024

6 Data Rounding on "Really Good" Approximations

In this section we consider a technique called **data rounding**. The high level idea is that we can round the input values in a way to drastically reduce the number of solutions which, combined with other algorithms such as DP, leads to efficient solutions.

6.1 Revisiting the $0-1$ Knapsack Problem

The first example we examine is the knapsack problem, which is sometimes treated as the first example of dynamic programming. Given n items, each with weight w_i and values v_i, our goal is to maximize the values of items we take whose total weight is bounded by some capacity W. WLOG assume each $w_i \leqslant W$ for otherwise the item cannot be picked regardless. The classic DP-based approach solves to solve the following recursion:

$$K(i,w) = \begin{cases} K(i-1,w) & w < w_i \\ \max(K(i-1,w), K(i-1,w-w_i)+v_i) & w \geqslant w_i \end{cases}$$

where $K(i,w)$ measures the optimal total value that one can take from items $1,\cdots,i$, with total budget w. And it is well-known that knapsack solved via this approach is pseudopolynomial, where its runtime also depends on the maximum weight budget.

The problem lies in the fact that the DP memoization table for the above approach has W columns. One way to improve, naturally, is to ask whether it is possible to reduce the number of columns to a quantity that is polynomial w.r.t. the number of bits representing it.

Before proceeding, we first notice that we must *not* modify the weights w_i of items, for otherwise we are left with a *completely* different problem, and there is no way to draw connection to the optimal solutions of both problems. Instead, (and to make this sound like an approximation algorithm) we modify the *values* v_i via **discretization**: we impose thresholds, and round each original value v_i to an associated discretized value.

Let $\epsilon > 0$ be given. We prove the following result:

Theorem

Let V be the maximal value $\max_i v_i$ among all items and define $k = V\epsilon/n$ to be the **discretization scale**. For each item i, we round its value down to the nearest multiple of k, i.e., $\overline{v}_i = \lfloor v_i/k \rfloor \cdot k$. Then, this rounding algorithm is an $1-\epsilon$ approximation with runtime $\mathcal{O}(n^3/\epsilon)$, which is strongly polynomial.

Proof. Observe that via rounding, each item losses value of at most k. Since at most n items can be picked, the total loss is bounded by $V\epsilon$. On the other hand, the original OPT is certainly at least V for this can be attained by simply picking the most valuable item. Combining both claims, we see that the total loss is $\leqslant \epsilon \cdot \mathrm{OPT}$, completing the proof.

As for the runtime, it is certainly bounded by $\mathcal{O}(n^2V/k) = \mathcal{O}(n^3/\epsilon)$, which amazingly is now free of W. □

6.2 Bin Packing

Now we consider the bin packing problem: there are n items, each with weights $s_1,\cdots,s_n \leqslant 1$. There are also bins with capacity $= 1$. Our goal is to pack the items into as few bins as possible.

There are many applications to this problem, for example caching and memory allocation. Some of the commonly heard strategies include "first fit" where one finds the first available bin with enough space to fit s_i, and if none exists, get a new bin. Another common strategy is "best fit" where one finds the fullest bin that can contain s_i, and get a new bin if no such bin exists.

It is easy to prove that both strategies satisfy an *asymptotic* 2-approximation: $\mathrm{ALG} \leqslant 2\mathrm{OPT}+1$. This is because at any time in the algorithm, *all but at most one bin must be $\geqslant$ half full.* Otherwise, if we have two bins that are less than half full, we could have merged them (slightly more justification needed, but this is the idea).

In what follows, we show that **it is possible to derive a polynomial-time solution such that $\mathbf{ALG \leqslant (1+\epsilon)OPT+1}$,**

for any given $\epsilon > 1$. The "+1" at the end implies that bin packing is *asymptotically* PTAS (but not). Without it, the best approximation ratio cannot achieve $\leqslant 3/2$, as otherwise this would contradict the NP-hardness of the partition problem. But that is okay, since when the input size is large, the trailing 1 is negligible, and in the end we "almost" still have an $1+\epsilon$ approximation.

First, we will assume the two following properties of the s_i's and show that this yields a valid polynomial-time solution. Then, we will show that it is possible to transform *any* instance of bin packing into one satisfying the following two while losing no more than a factor of ϵ. This will get the proof completed.

Assumption 1. All s_i's satisfy $s_i \geqslant \epsilon$.

Assumption 2. The number of distinct values among the s_i's is constant. Call it k.

Deriving an Approximate Bin Packing Algorithm. We will approach this problem via a multi-step approach, each time with less additional constraints and eventually solve the approximate bin packing.

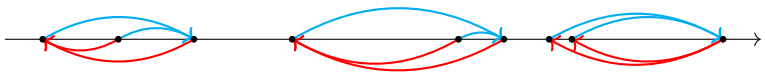
STEP 1. Assuming both assumptions hold, there exists a polynomial-time exact algorithm (i.e. finds the best answer to the bin packing instance).

Proof of STEP 1. First, by assumption 2, we can "categorize" items by type 1 up to type k. Therefore, in each bin, there are at most k distinct types. Further, each bin contains $\leqslant 1/\epsilon$ items by assumption 1.
Think of representing each bin by a dictionary with k keys. One (not so elegant) way to represent this specific bin/dicitonary is via a binary string using stars-and-bars, where 1's serve as category separators, and the count of 0's between bars represent the count of items of that type. For example, if $k = 4$, and a bin's item count for type 1 to 4 are $2, 4, 0,$ and 3, respectively, then the bin could be represented by

$$00\ 1\ 0000\ 1\ 1\ 000.$$

It's apparent that each configuration uniquely characterizes a bin. Such a string has at most $k + 1/\epsilon$ bits (can be shorter), so the total space of configurations has size bounded by $M := \binom{k+1/\epsilon}{k}$.
Now we repeat the same argument, but this time for the overall arrangement of bins. There are M types of bins, and the total number of bins is bounded by n, since there are only n items in total. Another stars-and-bars argument implies that the total number of solutions to this instance of bin packing is bounded by $\binom{M+n}{n} \leqslant (M+n)^n$ which is also polynomial w.r.t. n and ϵ.
Therefore, a brute force based approach would indeed find the exact optimal solution in polynomial time. (Do observe that the exponent would be astronomical! So while this algorithm is polynomial, it is almost never practical. For example, if $\epsilon = 1/20, n = 20$, and $k = 3$, the exponent would be 23 choose $3 = 1171$ and the overall runtime $\mathcal{O}(n^{1171})$. Well...) END OF PROOF OF STEP 1

STEP 2. We now relax assumption 2 and deal with any bin packing instance, only subject to the constraint that $s_i \geqslant \epsilon$ for each i. We show that there still exists a $1+\epsilon$ approximation algorithm.
Proof of STEP 2. The rounding scheme goes as follows: for some prescribed r that we determine later, (i) sort the bins in increasing sizes, (ii) partition items into groups of size r, and (iii) within each group, round everything to the largest item. By doing so, we reduce the problem to STEP 1, with $k = \lceil n/r \rceil$. Therefore, as our brute force algorithm outputs the exact optimal solution, to calculate the approximation factor, it suffices to compare OPT (un-rounded optimal solution) directly against OPT' (rounded up optimal solution).
It turns out a very elegant of bounding this difference is by introducing a third instance in which we round everything *down* to the smallest item in each group of size r. Call the corresponding optimal solution OPT''.



The key observation here is that we can compare the rounded up values of group j against the rounded down values of group $j+1$. This works for all but one pair, namely, the rounded up value of the largest group against the rounded down value of the smallest group. Put slightly more formally, let $\mathrm{OPT}'(i)$ denote the sum of values from group i, and likewise for $\mathrm{OPT}''(j)$. Then,

$$\begin{aligned}\mathrm{OPT}' - \mathrm{OPT}'' &= \sum_{i=1}^{n}[\mathrm{OPT}'(i) - \mathrm{OPT}''(i)] \\ &= \sum_{i=1}^{n-1}[\mathrm{OPT}'(i) - \mathrm{OPT}''(i+1)] + [\mathrm{OPT}'(n) - \mathrm{OPT}''(1)] \\ &\leqslant \mathrm{OPT}'(n) - \mathrm{OPT}''(1) \leqslant 1 \cdot r,\end{aligned}$$

since each group contains r elements, and all elements are within $[0,1]$. (This reminds me of ancient Chinese game theory strategy known as Tian Ji's horse racing strategy.)

Beginning of 10/16/2024

Therefore, if $r = n\epsilon^2$ then we have obtained a $1+\epsilon$ appromate algorithm, since the total weight is at least $n\epsilon$, and OPT therefore contains at least $n\epsilon$ bins, each of capacity 1. END OF PROOF OF STEP 2

STEP 3. Finally, we also relax assumption 1. But that's easy. We first pack everything $\geqslant \epsilon$ into bins as described above. Then we can use any reasonable strategy, for example first fit, to simply put all remaining small elements of size $< \epsilon$ into the bins, and include a new bin whenever necessary. Two possibilities:

(1) This procedure does not introduce any new bins. Then there's nothing to show, for certainly the result is still $1+\epsilon$ approximate without new bins.

(2) This procedure opens (one or more) bins. In order to open a bin, we know for sure that all current bins are at least $1-\epsilon$ full, since the new item to be packed is $< \epsilon$. This shows that eventually, all but one bin is at least $1-\epsilon$ full. Therefore,

$$\mathrm{ALG} \leqslant 1 + (1-\epsilon)^{-1}\sum_{i=1}^{n} s_i \leqslant 1 + (1+2\epsilon)\sum_{i=1}^{n} s_i \leqslant 1 + (1+2\epsilon)\mathrm{OPT}.$$

Since ϵ is arbitrary, we are done.

To sum up, below is the full procedure that we have described above.

Algorithm 8: Approximate PTAS Bin Packing

Input: $\epsilon > 0$, and n items with capcities/sizes s_i
temporarily discard all "small" items with $s_i < \epsilon$
for remaining "large" items **do**
    $k \leftarrow n\epsilon^2$
    sort these items in increasing sizes and group into blocks of size k
    round all items to the max item size in the block it lies in
    with rounded "large" items: solve exactly using brute force
for "small" items **do**
    pack each item using first fit (or any other), and open new bin if necessary

7 Cuts and Metrics

7.1 Multi-Cuts

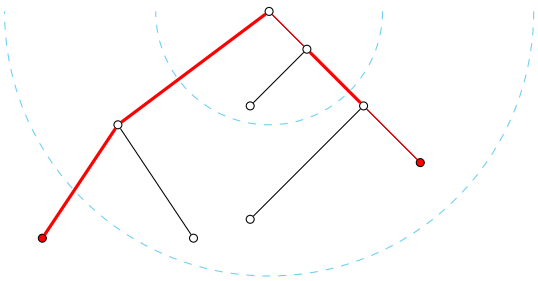
Recall that in the minimum $s-t$ cut problem, given a graph $G = (V, E)$ with nonnegative edge weights c_e, we want to find the minimum cost cut/separator of s and t. Here, we consider a direct generalization: we seek a partition of vertices, and instead of one pair (s, t), we have n pairs, where each s_i is to be separated from t_i. This problem is known to be NP-hard for $n \geqslant 3$.

Let us first write out the LP formulation of the problem:

$$\min \sum_{e \in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in P_i} x_e \geqslant 1 & \text{for all unique } s_i - t_i \text{ path } P_i \\ x_e \geqslant 0 & \text{for all } e. \end{cases}$$

We begin the simplest nontrivial graph among all, a graph with three nodes, s, r, t (where r is "root"). Clearly, if given a fractional solution x_e^* to this three-node graph, then $x_{(s,r)} + x_{(r,t)} \geqslant 1$, meaning that at least one of the edges $(s, r), (r, t)$ has a value $\geqslant 1/2$. It follows that we can simply round this value up to 1 and the other down to 0 to recover an integer solution that is no worse than a 2-approximation, must alike the threshold rounding examples we've seen many times before.

Now, to add slightly more generality, let us consider a larger tree.



Let s_i, t_i be any non-root node (highlighed as red in the figure). The path from s_i to t_i must walk past the root. Now assume we have already solved the fractional multi-cut LP and obtained the solution x_e^*. It follows, from feasibility, that the sum of all x_e^* along this path is $\geqslant 1$, so we want to design a rounding scheme that picks a certain subset of sets along this edge, round them up to 1, all while ensuring that this scheme can be simultaneously applied to various s_i, t_i pairs.

A natural attempt would be to claim that either the left path or the right path (w.r.t root t) has a total value of $\geqslant 1/2$, so can round that half up. But it is not clear which *edge* in that half-path should we round up, let alone the feasibility of parallel rounding across multiple $s_i - t_i$ paths. Instead, we will again grow "balls." The statement is

extremely simple:

- Pick a starting radius R uniform at random from $(0, 1/2)$.
- Grow a ball centered at root r with radius R, increasing its radius each time by $1/2$ (so $R, R+1/2, R+1$, and so on), where the distance metric of the tree is given by values of x_e^*.
- Stop when the ball engulfs all $s_i - t_i$ pairs, and round all edges cut by the growing ball up to 1.

For example, in the figure above, let the red vertices be s_i and s_i. We end up rounding 3 out of the 5 edges up. One of the other two (thin red) edges has a value x_e^* to small and was missed by the initial $R \in (0, 1/2)$, and the other also too small and missed by the $1/2$ discrete increment of R.

Firstly, this rounding scheme admists a feasible solution: if we view $s_i - t_i$ as a combination of the paths $s_i - r$ and $r - t_i$, where r is the *lowest common ancestor* (LCA), then one of the paths must have length $\geqslant 1/2$ because the total length $\geqslant 1$ by feasiblity. Then since initially $R < 1/2$, it must cut one side, and the edge being cut will be rounded up, making the result feasible. Therefore, the reasl question is, are we maintaining a 2-approximation factor? The answer is (of course), yes.
We consider individual edges along the path, and we have two cases based on the value of x_e^*.

CASE 1. $x_e^* \geqslant 1/2$. Certainly this edge will be cut because R increases by $1/2$ every time. On the other hand it's also clear that rounding up still induces an approximation factor of $\leqslant 2$.
CASE 2. $x_e^* \leqslant 1/2$. In this case, e is either cut once or not at all. A simple argument in proability shows that $\mathbb{P}(e \text{ is cut}) = x_e^*/0.5 = 2x_e^*$, so in expectation, x_e^* is rounded to $2x_e^*$ which also retains the 2-approximation factor.

Therefore, this rounding scheme is 2-approximation, completing our claim.

Beginning of 10/23/2024

Our goal is to consider a direct generalization of the multi-cut problem. First obseve that the original rounding scheme results in a solution set with the following properties:

(*) Among the graph of remaining edges (excluding the ones being chosen via rounding), **the diameter of each component is < 1**. Otherwise, consider two nodes u, v with $d(u, v) > 1$. This means at least one of the paths from s_i or t_i to their LCA r has distance $\geqslant 1/2$. But then some edge will be cut because we increment R by $1/2$.

(**) For each edge e, $\mathbb{P}(e \text{ is cut}) \leqslant 2x_e^*$. Obvious from the previous analysis.

Both of the claims would change according to our criterion for selecting the initial R. For rationales that will be explained later, let us generalize $R \in (0, 1/2)$ by replacing it with $R \in (0, d/2)$ for some d and increment R by $d/2$ instead of $1/2$ each time. Correspondingly, the two claims naturally generalize to:

(*) The diameter of any remaining component is $< d$, and

(**) Each edge has probability $\leqslant 2x_e^*/d$ of being cut.

This leads us to a more generlaized problem:

7.2 Low Diameter Decomposition (LDD)

Given $G = (V, E)$ with x_e as edge length of e, we want to partition the graph such that

(1) The diameter of each component is $\leqslant D$ for some D, and

(2) For each edge e, $\mathbb{P}(e \text{ is cut}) \leqslant \alpha \cdot x_e/D$ for some α depending on the nature of the graph.

Observe that as analyzed before, when G is a tree, α can be made constant. In particular, LDD outputs an α-approximation of multi-cut with $D = 1$, for feasibility is guaranteed by (*), and when $D = 1$,

$$\mathbb{E} \sum_{e \in \mathcal{F}} c_e \leqslant \frac{\alpha}{D} \sum_{e \in E} c_e x_e = \alpha \sum_{e \in E} c_e x_e.$$

Further, using a separation oracle, we can also efficiently solve the multi-cut LP, where we just need to run shortest path algorithm for each (s_i, t_i) pair and see if all pairs are now with distance $\geqslant 1$, and return feasible iff this condition holds for all pairs. Therefore, to solve the multi-cut problem, it suffices to simply find a satisfying LDD, which we will focus on from now on.

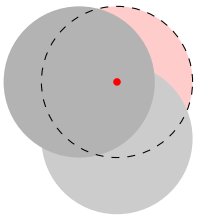
For general graphs, however, the factor α need not be constant — it could be dependent on the graph.

What would be our strategy now? Certainly, growing one ball and cutting edges is no longer sufficient — consider the annulus/ring of width $d/2$ between two circles. If there exists a long $s - t$ path contained entirely within this annulus, then our rippling circles will not be able to capture and cut it.
The fix is, of course, easy: grow rings from every vertex, but not simultaneously. Let π be a permutation, i.e., let $\pi(1), \cdots, \pi(n)$ be an enumeration of all vertices. Let $B(x, r)$ be the ball centered at x and with radius r (w.r.t. path lengths). The process is simple: in each step i, assign all vertices in $B(\pi(i), R)$ that is not yet assigned (by a previous iteration) to a component represented $\pi(i)$. When we are done, remove all edges crossed by any of the boundaries.
Some immediate observations:

(1) $\pi(i)$ need not be assigned to the component represented by itself, and

(2) components defined this way may even end up being disconnected.

(This illustration shows an example where $\pi(i)$ (red dot) is not assigned to the component (red) it represents.) To see that we have a feasible solution, let s_i, t_i belonging to the same component S_j be given. If there is to exist a path P after components have been formed and edges removed, certainly



$$P \in S_j = B(\pi(j), R) \backslash \{ \bigcup_{k<j} B(\pi(k), R \} \subset B(\pi(j), R).$$

Even $B(\pi(j), R)$ has radius $R < d/2$, so the path length of P must be no more than d, satisfying the feasibility constraint (*).

Now let us analyze the second condition (**), the probability of an edge $e = (u, v)$ being cut by the boundary of *some* ball. To do so, we first simplify the graph, focusing only on $e = (u, v)$, as well as all other vertices but not edges. Specifically, we transform the graph into a long chain with $e = (u, v)$ in the center. For each other vertex w, we define

$$d(w) = \min(d(v, w), d(u, w))$$

and attach w to the chain leading to u if $d(w) = d(u, w)$, and to v otherwise. Further, for every non-$\{u, v\}$ node, we sort and relabel their indices to be $\sigma_1, \sigma_2, \cdots, \sigma_{n-2}$ based on $d(w)$. For example, σ_1 is closer to u than to v, whereas σ_2 is closer to v than to u. Below is one example:

Note that these paths between σ_i's do not represent the actual structure of the graph — they are visualized in this way only to highlight distance to u and v.
Clearly, if $e = (u, v)$ were to be cut, it has to be cut by one of the σ's. Consider, for example, σ_2, where we ask, "what does it take for $B(\sigma_2)$ to cut through e? The condition is $d_i \leqslant R \leqslant d_i + x_e$, which has probability at most $x_e/(d/2)$. There are potentially $n - 2 = \mathcal{O}(n)$ possible nodes, from which we can draw balls of radius R and possibly cut e. Therefore, union bound shows that the total probability of e is bounded by $\mathbb{P}(e \text{ cut}) \leqslant \mathcal{O}(n) \cdot (x_e/d)$, so this gives the factor $\alpha = \mathcal{O}(n)$.

Observe, however, that there is a way to improve: for example, if e is cut by σ_2, we should no longer care if it is also cut by some further nodes σ_j, $j > 2$. That is, we care about the *first* occasion where e is cut. Because the σ_j's are a random permutation of the vertices (recall how we defined π), σ_1 has a probability of $\sim 1/n \cdot (x_e/(d/2))$ of being the first to cut e. The larger the index j for σ_j, the smaller the denominator and hence the larger the probability is:

$$\mathbb{P}(e \text{ is cut}) = \mathcal{O}(x_e/D) \sum_{i=1}^{n-2} \log n = \mathcal{O}(\log n)(x_e/D),$$

from which we have improved $\alpha = \mathcal{O}(n)$ to $\mathcal{O}(\log n)$. This is still worse than the constant factor that we can easily achieve on trees, but it's good enough.

7.3 Multi-Way Cut

Beginning of 10/28/2024

In this section we consider a generalized problem, where instead of looking for a decomposition/vertex decomposition that separates all given (s_i, t_i) pairs, we are given a set $\{s_i\}_{i=1}^k$ of k terminals and aim to pairwise separate all of them. This is known as the **Multi-Way Cut** problem.

Let us first consider a baseline solution:

- Fix some s_i. Our goal is to separate s_i from all $s_j, j \neq i$.
- This can be done by augmenting a super sink t_i, and add an edge (s_j, t_i) with infinite capacity for each $j \neq i$.
- Any (s_i, t_i) min cut must not contain an edge (s_j, t_i) because of its infinite capacity, so it will separate s_i from $\{s_j : j \neq i\} \cup \{t_i\}$. This gets the job done, since we can efficiently compute min cut.
- Repeat for each different i for a total of k times.

Trivially, this gives us a k-approximate solution. We will, however, take a closer look at the structure of the solution and show that it is in fact much better than this naïve estimate.
Let OPT be given. Observe that OPT must have k connected components where there is exactly one terminal s_i in each component. Let us call these components $S_1, \cdots, S_k$ where $s_i \in S_i$.
For each $i \in [k]$, let X_i be the total cost of edges crossing S_i in OPT. Similarly, let C_i be the cost of edges crossing S_i in our naïve solution ALG. Because C_i is obtained via computing the min cut splitting s_i and all $s_j, j \neq i$, certainly the cost $c(C_i) \leqslant c(X_i)$. Summing over all $i \in [k]$ and using the fact that each edge is precisely double counted,

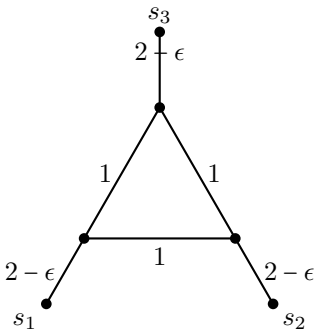
$$\text{ALG} \leqslant \sum_{i=1}^{k} \sum_{e \in C_i} c_e \leqslant \sum_{i=1}^{k} \sum_{e \in X_i} c_e = 2\text{e} \in \text{OPT} c_e = 2\text{OPT},$$

i.e., ALG is in fact 2-approximate. In fact, we can further slightly optimize this algorithm by terminating one iteration early: if $s_1, \cdots, s_{k-1}$ are all already separated from s_k, there is no need to perform iteration k. By reordering the cuts, we may safely discard the heaviest cut and obtain a total factor of $2(1-1/k)\mathrm{OPT}$

This bound is indeed tight: consider an n-gon with edge weights 1, and for each vertex, we further connect it to a terminal s_i with edge weight $2-\epsilon$. The following graph illustrates the case $n=3$. The optimal solution is to remove all edges along the n-gon, giving a total cost of n. Our algorithm, however, would be forced to remove $(n-1)$ edges of weights $2-\epsilon$. The comparison between $(n-1)(2-\epsilon)$ against n approaches $2(1-1/k)$ as $n \to \infty$.

Unfortunately, the standard LP itself

$$\min \sum_{e \in E} c_e x_e \qquad \text{subject to} \qquad \begin{cases} \sum_{e \in \mathcal{P}} x_e \geqslant 1 & \text{for all } (s_i, t_i) \text{ path } \mathcal{P} \\ x_e \geqslant 0 \end{cases}$$



admits an integrality gap of $2(1-1/k)$ as well: consider a star graph where all leaves are terminals and all edge weights are 1. The optimal integer solution is to remove $k-1$ edges, but a fractional solution assigning each edge the value $1/2$ gets the job done. Therefore, we have no hope of improving our algorithm beyond even the baseline, unless we consider a different LP.

So what went wrong here? The answer is that our LP is directly considering the edge costs w_e. A graph with edge weights naturally induces a **metric** $d(u,v)$ defined by the cheapest cost from u to v. More generally, a metric d can be any function defined on a space (in our case, the set of vertices) that is (i) nondegenerate, i.e., $d(\cdot,\cdot) \geqslant 0$ with equality if and only if both arguments equal, (ii) symmetric, i.e., $d(u,v) = d(v,u)$, and (iii) satisfies triangle inequality: $d(u,v) \leqslant d(u,t) + d(t,v)$.

The problem with our approach is that we are using the canonical metric. Instead, now we consider a more curated one which gives us additional properties that we can use to get better bounds.

7.3.1 Improving Multi-Way Cut using Metric Embedding

Now we start over by designing a different metric. Recall that OPT must contain exactly k connected components, each containing one terminal s_i. The distinction between individual vertices in V are much less interesting than the distinction between the components. So, how about we categorize the vertices $v \in V$ into clusters, based on the terminal it contains? In particular, we can define a binary function $d(u,v)$ such that $d(u,v) = 1$ if and only if u, v belong to the same component S_i. It is easy to verify that this function indeed induces a discrete metric.

Now consider a new interpretation of variables: for each $u \in V$ and $i \in [k]$, let $x_{u,i}$ denote the **assignment variable** that evaluates to 1 if $u \in S_i$ and 0 otherwise. Now, if u, v are assigned to the same cluster, then $x_{u,i} = x_{v,i}$ for all i's. Otherwise, $|x_{u,i} - x_{v-i}| = 1$ for two such indices (one for each cluster). Essentially, the ℓ^1 distance between vectors $(x_{u,1}, \cdots, x_{u,k})$ and $(x_{v,1}, \cdots, x_{v,k})$ is either 0 or 2, and our metric is half of it. Therefore, the original LP can now be re-written as

$$\min \frac{1}{2} \sum_{(u,v) \in E} c(u,v) \cdot \underbrace{\sum_{i=1}^{k} |x_{u,i} - x_{v,i}|}_{\ell^1 \text{ distance}/2} \qquad \text{subject to} \qquad \begin{cases} \sum_{i=1}^{k} x_{u,i} = 1 & \text{for all } u \in V \text{ (assignment)} \\ x_{s_i,i} = 1 & \text{(cluster center)} \\ x_{u,i} \geqslant 0. \end{cases}$$

We have already implicitly stated this right above, but because of the form the objective takes, it is natural to

vectorize the variables into k-dimensional vectors $x_u = (x_{u,1}, \cdots, x_{u,k})$ for each $u \in V$. Under the view of this **vector relaxation**, the first and third constraints imply that each x_u is in the k-dimensional simplex Δ_k, whereas in addition, the second implies that all terminal nodes s_i require x_{s_i} being one-hot (i.e. one coordinate as 1 and all others as 0). Let us now simply the notation and state multi-way cut as a **simplex LP** problem. In the following, define $\|x\| = \sum_{i=1}^{k} |x_i|/2$, i.e., the half ℓ^1 norm.

$$\min \sum_{(u,v)\in E} c(u,v)\|x_u - x_v\| \qquad \text{subject to} \qquad \begin{cases} x_u \in \Delta_k \subset \mathbb{R}^k & \text{for each } u \in V \\ x_{s_i} = e_i \in \mathbb{R}^k & \text{for each } s_i. \end{cases} \qquad \text{(Multi-Way Cut Simplex LP)}$$

Once we have obtained a fractional solution, the rounding itself is rather straightforward, as we once again grow balls and remove edges cut by the surface sphere, the only difference being the balls are now drawn with respect to the norm $\|\cdot\|$.

Algorithm 9: Rounding Simplex LP Multi-Way Cut

Inputs: $x_v \in \mathbb{R}^k$, solution to the simplex LP
randomly select $R \in (0, 1)$
for $i = 1, \cdots, k$ **do**
    define $C_i = \text{Ball}(s_i, R) \setminus \bigcup_{j<i} C_j$
return all edges (u, v) where u, v belong to different C_i's

Beginning of 10/30/2024

Claim. The output is $3/2$-approximate.

To prove this claim, we first take a look at what it means if $x_v \in \text{Ball}(s_i, R)$.

$$\begin{aligned} x_v \in \text{Ball}(s_i, R) &\implies \|x_v - e_i\| \leqslant R \implies \sum_j |x_{v,i} - e_{i,j}|/2 \leqslant R \\ &\implies \frac{1 - x_{v,i}}{2} + \frac{1}{2}\underbrace{\sum_{j\neq i} x_{v,j}}_{=1-x_{v,i}} \leqslant R \implies 1 - x_{v,i} \leqslant R. \end{aligned}$$

We have therefore obtained the following characterization:

$$\text{Ball}(s_i, R) = \{v : 1 - x_{v,i} \leqslant R\}. \tag{*}$$

For convenience call the balls B_i, and $U = V \setminus \bigcup_{i=1}^{k} B_i$ be the set of vertices not enclosed by any ball. Observe that the partition $B_1 \cup U, B_2, \cdots, B_k$ is feasible: each component still contains exactly one terminal, and we just need to remove all edges in ∂B_i as well as ∂U. That is, it suffices to analyze (which is also an upper bound, not necessarily optimal even after rounding)

$$\sum_{i=1}^{k} c(\partial B_i) + c(\partial U).$$

First fix an i and consider an edge $(u, v) \in \partial B_i$. By assumption, one endpoint lies in B_i but the other does not. Therefore, this requires that (assuming WLOG $x_{u,i} \geqslant x_{v,i}$) $1 - x_{u,i} \leqslant R < 1 - x_{v,i}$. Since R is drawn uniformly at

random from $(0,1)$, such event has probability $|x_{u,i} - x_{v,i}|$ (and note that now we may drop the assumption that $x_{u,i} \geqslant x_{v,i}$). Summing over all balls i,

$$\mathbb{P}((u,v) \in \bigcup_i \partial B_i) = \sum_i |x_{u,i} - x_{v,i}| = \|x_u - x_v\|. \tag{**}$$

Now we consider an edge $(u,v) \in \partial U$. WLOG assume $u \in U$. Then $1 - x_{u,i} > R$ for all i by (*), so $\max_i x_{u,i} < 1 - R$. Because $v \notin U$, it lies in some ball B_j, so by (*) again $1 - x_{v,j} \geqslant R$ and in particular $\max_i 1 - x_{v,i} \geqslant R$. Thus, $(u,v) \in \partial U$ implies

$$\max_i x_{u,i} < 1 - R \leqslant \max_i x_{v,i},$$

and so

$$\mathbb{P}((u,v) \in \partial U) = |\max_i x_{v,i} - \max_i x_{u,i}| \leqslant \frac{1}{2} \sum_{i=1}^{k} |x_{u,i} - x_{v,i}| = \|x_u - x_v\|/2. \tag{***}$$

(The proof of the inequality is omitted here.) Combining (**) and (***), we see that by picking edges crossing $\{B_1 \cup U, B_2, \cdots, B_k\}$, we obtain an overall $3/2$-approximate algorithm.

One last issue remains: we need to resolve overlaps. If $B_i \cap B_j \neq \varnothing$, then one potential solution to resolve this is by replacing $\{B_i, B_j\}$ with either $\{B_i, B_j \backslash B_i\}$ or $\{B_i \backslash B_j, B_j\}$. Observe that U remains unaffected by such operations. We recall the ancient definitions of super- and sub-modularity, and state without proof that

$$c(\partial X) + c(\partial Y) \geqslant c(\partial(X \backslash Y)) + c(\partial(Y \backslash X)).$$

This immediately implies that either $c(\partial X) \geqslant c(\partial(X \backslash Y))$ or $c(\partial Y) \geqslant c(\partial(Y \backslash X))$ (for otherwise the $\geqslant$ cannot hold). Then, correspondingly, we add $c(\partial Y)$ or $c(\partial X)$ to both sides, and we see that we can replace $\{X, Y\}$ with either $\{X, X \backslash Y\}$ or $\{Y, Y \backslash X\}$ without inducing extra costs. This completes the proof.

7.4 Sparsest Cut

Beginning of 11/04/2024

In this section we consider the **sparsest cut** problem. The difference from before is that now when an (s,t) pair is separated by the cut, we gain some credit instead of being charged some cost. More formally, given undirected $G = (V,E)$ with capacity matrix C (where $C_{i,j}$ is the capacity of a path, not necessarily a direct edge, (i,j)) and demand matrix $D_{i,j}$ measuring credits, we want to find a cut that minimizes the cost relative to credit. That is, we want to find a cut S that minimizes

$$\sum_{(u,v) \in \partial S} C_{u,v} \Big/ \sum_{(u,v) \in \partial S} D_{u,v}.$$

This is a direct generalization of some of the problems we have previously encountered. For example, if the demand is just a single pair, then after some rescaling this problem reduces to minimum $s-t$ cut.

To effectively model whether $(u,v) \in \partial S$, we can use a indicator random variable $\mathbf{1}[(u,v) \in \partial S]$. Then, we can relax the sum to over all $u, v \in V$ and replace $C_{u,v}$ and $D_{u,v}$ with the product with this indicator variable:

$$\min_{S \subset V} \sum_{u,v \in V} C_{u,v} \mathbf{1}[(u,v) \in \partial S] \Big/ \sum_{u,v \in V} D_{u,v} \mathbf{1}[(u,v) \in \partial S].$$

Two additional improvements that we make. First, observe that the above can be viewed as a function of S. For each fixed S, the indicator random variable is deterministic. Second, $\mathbf{1}[(u,v) \in \partial S]$ defines a metric (this is called

a **cut metric** associated with S) on G, and so does any scalar multiple of it. These imply that given a fixed cut S, the ratio is equivalent to that on a graph equipped with a scaled metric such that the denominator equals 1.
Finally, there is a one-to-one correspondence The objective is to optimize this ratio over all S. Writing each cut metric as some $d_S(u,v) = \mathbf{1}[(u,v) \in \partial S]$, the (rescaled) objective becomes

$$\min_{d_S \text{ cut metric}} \sum_{u,v \in V} C_{u,v} d_S(u,v).$$

We will perform one further relaxation, by replacing cut metrics with general metrics on graphs:

$$\min \sum_{u,v\in V} C_{u,v} d(u,v) \qquad \text{subject to} \qquad \begin{cases} \sum_{u,v\in V} D_{u,v} d(u,v) = 1 & \text{(rescaling assumption)} \\ d(u,v) \leqslant d(u,w) + d(w,v) & \text{(metric)} \\ d(u,v) = d(v,u) \geqslant 0. & \text{(metric)} \end{cases} \qquad \text{(Sparsest Cut LP)}$$

Cut Metrics, Cones, and ℓ^1 Metrics

We introduce the notion of **cone** of cut metrics: if d_1, d_2 are two cut metrics, then $\{\alpha d_1 + \beta d_2, \alpha, \beta \geqslant 0\}$ forms the **cone** of d_1, d_2. Note that this definition extends to any finite cut metrics.
We will resort to our old friend, then ℓ^1 metric. Our claim is that cut metrics are "essentially" ℓ^1 metric, in the sense that **the cone of cut metrics can be converted into an ℓ^1 metric, and vice versa.** Since

$$\frac{\alpha_1 C_1 + \alpha_2 C_2 + \cdots + \alpha_k C_k}{\alpha_1 D_1 + \alpha D_2 + \cdots + \alpha_k D_k} \geqslant \min_k \frac{C_k}{D_k},$$

it suffices to optimize among the cones, for the true optimal is guaranteed to be no worse.

Let us consider the easier direction first: a cone of cut metric can be converted into an ℓ^1 metric. Let a cone metric be given. We represent it using $\{(\alpha_i, S_i)\}_{i=1}^k$, i.e., the linear combination of cut metrics associated with S_i and coefficient α_i. Scaled cut metric is just a scaled bipartition of vertices, where the cut metric distance between nodes are either 0 or α_i. This naturally allows us to embed each node v into a vector in $v'\mathbb{R}^k$, where the i^{th} component is 0 if $v \in S_i$, and α_i if otherwise. Then, looking at the i^{th} component, the difference between u' and v' is either 0 (when they belong to both S_i or both S_i^c) or α_i (when they are separated by S). Hence,

$$\|u' - v'\| = \sum_{i=1}^k |u_i' - v_i'| = \sum_{i=1}^k \alpha_i \mathbf{1}[(u,v) \in \partial S_i] = \sum_{i=1}^k \alpha_i d_{S_i}(u,v).$$

The converse requires slightly more work. Let us begin with a ℓ^1 metric in $\mathbb{R}^k$. For simplicity consider $k = 1$ first. Suppose we have n points, $0, x_1, \cdots, x_{n-1}$, and we want to find a cone metric that agrees with the pairwise ℓ^1 distance between these points. Assume $0 < x_1 < x_2 < \cdots < x_{n-1}$. We define a cut metric d_1 that separates 0 against $\{x_1, \cdots, x_{n-1}\}$, with a coefficient of x_1 (i.e. $d(0, x_i) = x_1$ and $d(x_i, x_j) = x_1$ for $1 \leqslant i < j \leqslant n-1$). Likewise, we define a second cut metric d_2 separating $\{0, x_1\}$ with $\{x_2, \cdots, x_{n-1}\}$ with coefficient $x_2 - x_1$. We repeat the process and define a metric d_s separating $\{0, x_1, \cdots, x_{s-1}\}$ against $\{x_s, \cdots, x_{n-1}\}$, with coefficient $x_k - x_{k-1}$. It follows that for $i > j$,

$$|x_i - x_j| = (x_i - x_{i-1}) + \cdots + (x_{j+1} - x_j) = \sum_s \alpha_s d_s(x_i, x_j).$$

For higher dimension k, we focus on one coordinate at a time, and represent the ℓ^1 metric in $\mathbb{R}^k$ as a linear

combination of nk cut metrics. This is valid since ℓ^1 is additive over coordinates.

Next up, we provide a theorem that ensures that the dimension required for such metric embedding is at most reasonably large.

Theorem: (Bourgain)

Any metric space (V, d_1) on n points can be embedded into a ℓ^1 metric space (V, d_2) with $\mathcal{O}(\log^2 n)$ dimension with a distortion factor of $\mathcal{O}(\log n)$. That is, for all $x, y \in V$,

$$d_1(x, y) \leqslant d_2(x, y) \leqslant \mathcal{O}(\log n) d_1(x, y).$$

One can also replace the claim with a non-deterministic one, where (V, d_1) is to be embedded into a distribution $\mathcal{D}$ of distributions, such that

$$d_1(x, y) \leqslant d_2(x, y) \qquad \text{for all } d_2 \in \mathcal{D},$$

and

$$\mathop{E}_{d_2 \sim \mathcal{D}} d_2(x, y) \leqslant \mathcal{O}(\log n) d_1(x, y).$$

The term $\mathcal{O}(\log n)$ here is the **distortion factor** of the embedding.

Beginning of 11/11/2024

8 Semidefinite Programming (SDP)

I missed a lecture on introduction to SDP; the following italicized section are notes borrowed from 532's SDP lecture.

In this section, we consider an alternative to linear programming LP, where we introduce a certain amount of geometric complexity by drawing connection to PSD (positive semidefinite) matrices in higher dimensions.

MAX-CUT

Firstly, we consider the MAX-CUT *problem, where given an undirected graph $G = (V, E)$ with edge weights $w_e \geqslant 0$, we want to find the cut (S, S^c) that maximizes the weights of edges crossing the cut, i.e., $\max \sum_{e \in \partial S} w(e)$. It is known that* MAX-CUT *is NP despite its counterpart, min-cut or max-flow, can be solved efficiently.*

A baseline for MAX-CUT *is that any algorithm we propose must do better than "random guess." Specifically, if for each edge we independently include it with probability $1/2$ in S, then*

$$\mathbb{E}(w(S)) := \mathbb{E} \sum_{e \in \partial S} w(e) = \frac{1}{2} \sum_{e \in E} w(e) \geqslant \frac{\mathrm{OPT}}{2}$$

so the baseline is we can certainly achieve a $1/2$-approximation factor. A slightly more sophisticated threshold rounding scheme on the LP relaxation gives a $3/4$-approximation. But how can we do better?

The idea behind SDP is that instead of assigning each variable $y_i \in \{0, 1\}$, we let $y_i \in \{-1, 1\}$. Then, $(1 - y_i y_j)/2 = 0$ is a

binary variable that equals 1 *if and only if* y_i, y_j *are assigned different values. Therefore, the SDP objective is*

$$\max \frac{1}{2} \sum_{(i,j)\in E} (1 - y_i y_j) w_{i,j} \qquad \textit{subject to} \qquad y_i \in \{-1, 1\}.$$

To fully generalize this problem into matrix algebraic notations and enjoy the benefits of geometry of PSD matrices, for vertex variable y_i*, we define its* ***vertex relaxation*** *to be* $v_i \in \mathbb{R}^n$*, such that all* v_i*'s are unit vectors. The problem can now be written w.r.t. vector products*

$$\max \frac{1}{2} \sum_{(i,j)\in E} (1 - v_i^T v_j) w_{i,j} \qquad \textit{subject to} \qquad \|v_i\| = 1 \textit{ for all } v_i \in \mathbb{R}^n. \qquad \text{(MAX-CUT SDP)}$$

*First, we answer the question "****why SDP?****" Suppose we know how to solve the problemm, obtaining a set of vector relaxation solution* $\{v_i^*\}$*. By definition, these points lie on the unit sphere in* $\mathbb{R}^n$*. Recall that our original objective is to partition* V *into two sets of vertices, one for* S *and another for* S^c*.*
The most natural interpretation of $\{v_i^*\}$*, therefore, is to partition these unit vectors into two sets as well. We consider an extremely simple approach: (i) pick a random unit vector* r*, and (ii) choose* $S = \{i : r^T v_i^* > 0\}$*, i.e., one of the hemispheres defined by the hyperplane to which* r *is normal. Let* $\theta_{i,j}$ *denote the angle between* v_i *and* v_j*, which can be calculated by* $\theta_{i,j} = \cos^{-1}(v_i^{*T} v_j^* / (\|v_i^T\| \|v_j^*\|)) = \cos^{-1}(v_i^{*T} v_j^*)$*. A simple geometric interpretation shows that* $\mathbb{P}(i, j \textit{ separated by cut}) = \mathbb{P}(v_i^*, v_j^* \textit{ in different hemispheres}) = \theta_{i,j}/\pi = \pi^{-1} \cos^{-1}(v_i^{*T} v_j^*)$*.*
It can be shown that θ*,* $\pi^{-1} \cos^{-1}(x) \geqslant 0.878(1 - x)/2$*, so this implies that the SDP randomized algorithm achieves an approximation factor of* 0.878*. Two side notes:*

- *Unless* $\mathrm{P} = \mathrm{NP}$*, it can be shown that* MAX-CUT *cannot have an approximation factor better than* 0.94*, and*
- *Under UGC (unique game conjecture), this factor of* 0.878 *is already the best possible.*

A side remark on how to solve the problem: the more general form of SDP (and also how it got this name) is

$$\max \sum_{i,j} c_{i,j} X_{i,j} \qquad \textit{subject to} \qquad \begin{cases} \sum_{i,j} a_{i,j_k} X_{i,j} = b_k & \textit{for each constraint } k \\ X \succeq 0 & \textit{(PSD)}. \end{cases}$$

With these assumptions, the Ellipsoid method works. So in the future we will assume that we have the black box that solves the vector relaxations of SDP.

Actual beginning of 11/11/2024

8.1 Graph Coloring via SDP: a $\mathcal{O}(n^{0.39})$ Solution

Let $G = (V, E)$ be a graph. A ***k*-coloring** for G is a way to color each vertex such that both ends of any edge are colored differently. Formally, a k-coloring is a function $f : V \to [k]$ such that $f(u) \neq f(v)$ for all $(u, v) \in E$. We say G is ***k*-colorable** if there exists a k-coloring of G. It is known that finding a k-coloring for G is solvable in polynomial time for $k = 2$ but NP-hard for $k \geqslant 3$, so instead we turn to approximation algorithms, where the approximation ratio is with respect to the number of colors we use.

A baseline we should consider is the following greedy algorithm: When a graph is 3-colorable, given any vertex x, its neighbors must be 2-colorable. We can then greedily start by coloring neighborhoods of vertices with high degree, and throw out parts that have already been colored. Some analysis shows that this happens at most $\sqrt{n}$ times, each with linear time. In each iteration we gain 3 colors, so the total number of colors is $3\sqrt{n}$, or $\mathcal{O}(\sqrt{n})$.

Now we will attempt to improve upon this $\mathcal{O}(\sqrt{n})$ solution by resorting to SDP. We will embed each vertex into a vector in some higher dimensional space, with the intuition that vertices that are connected should be far from each other, and that vertices with the same color should be clustered:

$$\min \lambda \qquad \text{subject to} \qquad \begin{cases} v_i^T v_j \leqslant \lambda & \text{for all edges } (i,j) \in \mathbb{E} \\ \|v_i\| = 1 & \text{for all vertices } i \\ v_i \in \mathbb{R}^n. \end{cases}$$

What would a feasible solution look like? Let us consider 3-coloring first. Certainly, if a graph is 3-colorable, we can find three single-point clusters in $\mathbb{R}^n$ that form an equilateral triangle on the unit sphere, so that their pairwise angle is $2\pi/3$. By assigning each vertex's vector relaxation to one of these clusters, we obtain $v_i^T v_j = -1/2$ whenever i, j are colored differently.

More generally, we show that if λ^* is the optimal/minimum value our SDP can attain, then

For a k-colorable graph G, $\lambda^* \leqslant -1/(k-1)$.

We show this via induction, and observe that the bases cases are already done: for $k = 2$, clearly we can force the two representations to be directly opposite, giving $\lambda^* = -1$. For $k = 3$, as argued above, $\lambda^* \leqslant -1/2$. We further assume that it is sufficient to represent such k vectors in $\mathbb{R}^{k-1}$, which clearly holds for base cases.

Now assume the claim holds for $k-1$, that a $(k-1)$-colorable graph admits a $\lambda^* \leqslant -1/(k-2)$. Let $\alpha_1, \cdots, \alpha_{k-1} \in \mathbb{R}^{k-2}$ be the vectors representing the $k-1$ colors, and now let the k^{th} color be given. Our goal is to embed α_i into $\mathbb{R}^{k-1}$, introduce a new $(k-1)$-dimensional vector for the last color, and show that their pairwise dot product is bounded by $-1/(k-1)$. This is easy: for each $\alpha_i \in \mathbb{R}^{k-2}$, we augment $\alpha_i' \in \mathbb{R}^{k-1} = (\beta\alpha_i, -1/(k-1))$ where $\beta^2 + (-1/(k-1))^2 = 1$ or $\beta = \sqrt{1 - 1/(k-1)^2}$; for the last additional color, associate it with $(0, \cdots, 0, 1) \in \mathbb{R}^{k-1}$. It follows that the dot product between color k and any other $j \leqslant k-1$ is just the product of the last coordinate, or $-1/(k-1)$. On the other hand, for $1 \leqslant i \leqslant j \leqslant k-1$,

$$\begin{aligned}(\alpha_i')^T(\alpha_j') = (\beta\alpha_i, -1/(k-1))^T(\beta\alpha_j, -1/(k-1)) &= \beta^2 \alpha_i^T \alpha_j + \frac{1}{(k-1)^2} \\ &\leqslant \left(1 - \frac{1}{(k-1)^2}\right) \underbrace{\left(-\frac{1}{k-2}\right)}_{\text{by IH}} + \frac{1}{(k-1)^2} \leqslant -\frac{1}{k-1}.\end{aligned}$$

In fact, this bound is also tight! Consider k-clique (k fully connected vertices) or any graph that contains a k-clique. The best possible way to separate them cannot get better than the bound above.

So now let's assume we have the optimal solution and instead talk about rounding. Using the same idea as before, we consider a random vector and the hyperplane defined by setting this vector as the normal. The random hyperplane cuts the unit sphere in $\mathbb{R}^n$ into two equal halves. If v_i, v_j are partitioned into different halves, then we assign different colors to the corresponding vertices. It follows that by doing so, for an edge $(i,j) \in E$,

$$\begin{aligned}\mathbb{P}((i,j) \text{ monochromatic}) &= \mathbb{P}((i,j) \text{ on same side of hyperplane}) \\ &= 1 - \frac{\theta_{i,j}}{\pi} = 1 - \frac{\cos^{-1}(v_i^T v_j)}{\pi} \leqslant 1 - \frac{\cos^{-1}(-1/2)}{3} = 1 - \frac{2\pi/3}{\pi} = \frac{1}{3}\end{aligned}$$

because $v_i^T v_j \leqslant -1/2$ in the optimal solution. That is, if we only have one hyperplane to determine if two vertices should be colored differently, then each edge has $1/3$ probability of failure. We can, of course, boost this by

considering t random hyperplanes that partition the space into 2^t parts. The probability that (i,j) falls into the same section is now $(1/3)^t$, which is much better.

How Much Should We Boost Rounding?

Regardless of how many random hyperplanes we choose, the nature of this algorithm guarantees that there will most likely be some edges that are monochromatic (so they need to be fixed), so we need to come up with a way to address them.

The cool observation is that if we can bound the number of monochromatic edges by $n/4$, then certainly the number of "bad" vertices is $\leqslant n/2$, and consequently we have $\geqslant n/2$ "good" vertices that are already properly colored. Therefore, keeping the good half frozen, we may recursively define a SDP subproblem, fixing half of the "bad" vertices, and so on. If in each round we use t random hyperplanes, then the total number of colors will be $2^t \log n$, since there will be $\log n$ iterations.

So we want to ensure that there are no more than $n/4$ monochromatic edges. Observe by a Markov-like argument that if $\mathbb{E}((i,j) \text{ monochromatic}) \leqslant n/8$, then with probability $\geqslant 1/2$, the total number of monochromatic edges is $\leqslant n/4$ (where we use the fact that $\mathbb{P}(X \geqslant a\mathbb{E}X) \leqslant 1/a$). We call results as such a **semi-coloring**. This is not completely deterministic, but if the rate of success is $\geqslant 1/2$, we may as well repeat it until we succeed, for we are expected to succeed after only 2 tries.

So now it amounts to requiring that $\mathbb{E}((i,j) \text{ monochromatic}) \leqslant n/8$. Let Δ be the maximum degree amount all vertices, which implies the total number of edges $m \leqslant n\Delta/2$. Therefore, it amounts to requiring that

$$\frac{n\Delta}{2} \cdot \frac{1}{3^t} \leqslant \frac{n}{8} \implies 3^t \geqslant 4\Delta \implies t \geqslant \log_3(4\Delta) \implies t \geqslant \log_3 \Delta + 1.$$

This in turn means the total color needed for our rounding is $2^t \log n = \mathcal{O}(n^{\log_3 2} \log n) \approx \mathcal{O}(n^{0.63} \log n)$, which is worse than our greedy baseline. What happened? We could have applied the idea behind the greedy algorithm to optimize our SDP rounding. Specifically, if a vertex has high degree, it might be better if we simply 3-color that vertex and its neighbors first, and only run SDP when the degrees of each vertex is below a certain threshold, say σ. This requires no more than $3n/\sigma$ colors for preprocessing, along with $\mathcal{O}(\sigma^{\log_3 2})$ colors to SDP round the pruned graph. Setting $\sigma = n^{\log_6 3}$ this gives an algorithm using $\mathcal{O}(n^{0.39})$ that outputs, with probability $\geqslant 1/2$, a semi-coloring. We can simply repeat this process until a desired result is obtained.

Beginning of 11/13/2024

8.2 Improving to $\mathcal{O}(n^{1/4})$ Colors

We consider a different approach to the coloring problem, noting that **the set of vertices that are assigned the same color forms an independent set**. It is known that the INDEPENDENT SET problem itself is very hard and there are no good approximations of it, so instead we will attempt to identify *one* independent set.

We will still use the typical SDP rounding scheme where after obtaining a set of n-dimensional unit vectors v_i for each vertex i, we partition the unit sphere into two halves, based on the hempishere defined by a randomly chosen normal vector $r \in \mathbb{R}^n$. Then, we focus on one party of it. The idea is that **we want to discard certain vertices on this hemisphere so that the remaining ones form an independent set**.

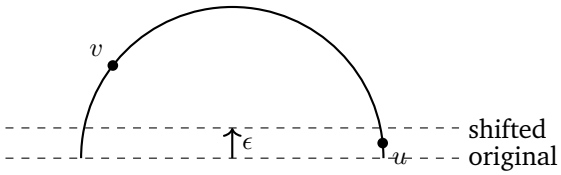
A naïve approach simply drops a vertex as long as it is incident to some other vertex in the same hemisphere, then repeat until no such vertex can be found. This however may end up with $\varnothing$ since the dropping criterion is too strict,

and no finer constraints are provided.

Instead, we observe the following:

- Any two vertices with an edge between them must be far apart, as $v_i^T v_j = -1/2$ implies their angle is $2\pi/3$.
- This means if v has a neighbor in the hempishere, then v itself must be relatively close to hyperplane.

These observation suggest that we should consider *shifted* hyperplanes to eliminate one of the endpoints to make the remaining vertices independent. For example, in the figure to the right, originally (u, v) are not separated by the original hyperplane, but by allowing a shift of ϵ, they are now separated.



To this end, let us define

$$S \coloneqq \{i \in V : v_i^T r \geqslant 0\} \qquad S(\epsilon) \coloneqq \{i \in V : v_i^T r \geqslant \epsilon\}$$

and

$$S'(\epsilon) \coloneqq \{i \in S(\epsilon) : j \notin S(i) \text{ for all vertex } j \text{ incident on } i\}$$

which gives an independent set. Our goal is to estimate $\mathbb{E}S'(\epsilon)$. Observe that since v_i is deterministic (we are only concerned with the rounding scheme) and each coordinate of $r = (r_1, \cdots, r_n)$ independently follows a Gaussian, so does $v_i^T r$, since Gaussian is rotationally invariant. Let $\Phi(x)$ be the CDF of the standard normal and $\Phi'(x) = 1 - \Phi(x)$. Then by linearity of expectation, summing over each vertex vector v_i,

$$\mathbb{E}\left[|S(\epsilon)|\right] = n \cdot \mathbb{P}(v_i^T r \geqslant \epsilon) = n\Phi'(s)$$

What about $S'(\epsilon)$? Let Δ be the max degree of any vertex in the graph. Then for any vertex i,

$$\begin{aligned}\mathbb{P}(i \notin S'(\epsilon) \mid i \in S(\epsilon)) &= \mathbb{P}(i \in S(\epsilon) \text{ but so does some neighbor}) \\ &\leqslant \Delta \cdot \mathbb{P}(j \in S(\epsilon) \mid i \in S(\epsilon), (i,j) \in E).\end{aligned}$$

We try to analyze the term $\mathbb{P}(j \in S(\epsilon) \mid i \in S(\epsilon), (i,j) \in E)$ via the following diagram. Let u be orthogonal to v_i. It follows that because the angle between v_i, v_j is $\pi/3$, we can write $v_j = -v_i/2 + \sqrt{3}u/2$. Therefore

$$v_j^T r = -\frac{1}{2} v_i^T r + \frac{\sqrt{3}}{2} u^T r.$$

By assuming $(i,j) \in E$ and $j \in S(\epsilon)$, we have $v_i^T r \geqslant \epsilon$ and so $(\sqrt{3}/2)u^T r \geqslant 3\epsilon/2$, implying $u^T r \geqslant \sqrt{3}\epsilon$. Like argued before, $u^T r$ follows a normal distribution, so

$$\mathbb{P}(i \notin S'(\epsilon) \mid i \in S(\epsilon)) \leqslant \Delta\Phi'(\sqrt{3}\epsilon).$$

Our goal is to choose an appropriate ϵ to ensure this quantity is bounded by $1/2$. To do so, we use the following fact:

$$\frac{xp(x)}{1+x^2} \leqslant \Phi'(x) \leqslant \frac{p(x)}{x}$$

where $p(x)$ is the PDF of a standard normal. With the help of this identity, setting $\epsilon = (2\log\Delta/3)^{1/2}$ kills Δ, as

$$\Delta\Phi'(\sqrt{3}\epsilon) \leqslant \frac{1}{\sqrt{3}\epsilon} \frac{1}{\sqrt{2\pi}} \leqslant \frac{1}{2}.$$

Combining everything we have discussed so far,

$$\mathbb{E}\left[|S'(\epsilon)|\right] \geqslant \frac{\mathbb{E}\left[|S(\epsilon)|\right]}{2} = \frac{n}{2}\Phi'(\epsilon) \geqslant \frac{n}{2}\frac{\epsilon}{1+\epsilon^2}e^{-\epsilon^2/2} = \frac{n}{2}\frac{\epsilon}{1+\epsilon^2}\frac{1}{\Delta^{1/2}} = \mathcal{O}\left(\frac{n}{\Delta^{1/3}\sqrt{\log\Delta}}\right). \tag{*}$$

So now we obtained an independent set. What next? **How to get a coloring**? The cleanup is just like before: we interactively compute independent sets on the shrinking graph, each time decreasing by a factor (*), until we have a graph of $\mathcal{O}(1)$ size left. Starting with a graph of size n, this requires $k = \mathcal{O}(\log n\Delta^{1/3}\sqrt{\log\Delta})$ steps. Then we use brute force.

This approach works well when Δ is relatively small. For large Δ, on the other hand, because the graph is 3-colorable, we prune each high-degree vertex v by assigning one color to v and then 2-color its neighbors. To sum up, we have the following two options: given a threshold σ,

- If $\Delta > \sigma$, we reduce Δ to σ via $\mathcal{O}(n/\sigma)$ colors by pruning high-degree vertices, and
- If $\Delta \leqslant \sigma$, we use $\mathcal{O}(\Delta^{1/3}) = \mathcal{O}(\sigma^{1/3})$ colors as in (*).

It follows that the best way to manage the trade-off is by setting $\sigma = n^{3/4}$ so that in either cases, the total number of colors required is $\mathcal{O}(n^{1/4})$.

9 Facility Locations & Local Search

Beginning of 11/18/2024

In this section, we consider the **facility location** problem, which is a special instance of clustering. Specifically, we consider the following problem setup:

> Let (V, c) be a metric space. Each point $v \in V$ can act as supplier facility or client, or both. Specifically, to produce any supply, each $v \in V$ needs to first induce an **opening cost** $f_v \geqslant 0$. For each i, j, the quantity $c_{i,j}$ denotes the **serving cost** of using (facility) i to serve (client) j. Each client will be served by the nearest open facility. The goal is to ensure all customers $v \in V$ are served, while ensuring the total opening costs plus serving costs are minimized.

Remark. Note that in the above definition, $v \in V$ can simultaneously be a supplier and a client. The problem also covers the special case where clients and suppliers are disjoint: simply set the opening cost of clients to ∞, so they must not become suppliers.
Also, we are analyzing a more restricted case where c forms a metric. A more general problem, the **non-metric facility location**, is essentially equivalent to SET COVER by assigning opening cost of every $v \in V$ to 1 and serving cost $c_{i,j}$ to 0.

Let us first define the LP formulation of the problem. We use $x_{i,j}$ to indicate whether i serves j, and use y_i to denote

if $i \in V$ is a supplier. It follows that the LP relaxation is given by

$$\min\left[\sum_{i,j\in V} c_{i,j}x_{i,j} + \sum_{i\in V} f_i y_i\right] \qquad \text{subject to} \qquad \begin{cases} \sum_i x_{i,j} \geqslant 1 & \text{for all } j \text{ (clients must be served)} \\ x_{i,j} \leqslant y_i & \text{for all } i,j \text{ (must open facility before serving)} \\ x_{i,j}, y_i \geqslant 0. \end{cases}$$

Instead of solving the fractional LP, we consider an alternate approach called **local search**. The core idea is deceptively simple: we make small changes a solution and see if there is any improvement. In our problem, we allow ourselves to choose from one of the following operations:

- OPEN (a new facility),
- CLOSE (an existing facility), and
- SWAP (i.e. closing an open facility and opening a new facility).

We further assume that every time an operation is performed on the set of open facilities, the serving assignment is updated correctly (so that clients are still served by their nearest open facility). By doing so, we will end up at a locally optimal solution (F, σ) where F is the set of open facilities, and $\sigma : V \to F$ maps clients to open facilities. Observe that because our operations are chosen such that each one results in a strict increase in performance, no two operations are identical. This ensures that our algorithm terminates, and with further optimizations (e.g. only consider a move if it improves the performance by a certain factor) this can be achieved in polynomial time.

So now let (F, σ) be a locally optimal solution and let (F^*, σ^*) be the global optimum. The remainder of this section will be devoted to analyzing the approximation ratio of (F, σ).

Warning: the proofs will be overloaded with all kinds of different notations. Each of them will be highlighted.
With an abuse of notation, let F, C denote the facility cost and serving cost of solution (F, σ) respectively [we'll use F interchangeably; hopefully the context is clear enough to avoid confusions], and likewise F^*, C^* for (F^*, σ^*).

Bounding (F, σ)'s Facility Cost F: Effects of OPEN

We first claim that there exists $i^* \in F^*$ but not F. Otherwise, if F is a superset of F^*, then either (F, σ) is identical to (F^*, σ^*) or F is a strict superset of F^*, which makes (F, σ) impossible to be locally optimal.
Now suppose we want to open $i^* \in F^*$. By local optimality this makes things worse. In other words,

$$\sum_{j:\sigma^*(j)=i^*} c_{i,j} \leqslant \sum_{j:\sigma^*(j)=i^*} c_{i^*,j} + f_i^*$$

for the LHS is the old cost of taking care of these clients and the LHS is the new cost. Summing over all $i \in F^*$,

$$\sum_{i\in F^*} \sum_{j:\sigma^*(j)=i^*} c_{i,j} = F \leqslant F^* + C^* = \sum_{i\in F^*} f_i^* + \sum_{i\in F^*} \sum_{j:\sigma^*(j)=i^*} c_{i^*,j}. \tag{13}$$

Bounding (F, σ)'s Serving Cost C: Effects of CLOSE

This is significantly more involved, but similar to above, let us first consider the effects of CLOSE-ing a facility. Let $i \in F$ be the facility that we aim to close. Let j be such that $\sigma(j) = i$. We need to find a new facility to assign j to! The complication arises in that we have two cases.

The first case is easy: if $\sigma^*(j) \in F$ then we simply assign j to i^*.

The more annoying case is when $\sigma^*(j)$ is not in F. In this case, it makes sense to assign i to the closest facility to i^* that is currently open, hoping that eventually this distance is not much different from $c_{j,\sigma^*(j)}$. To this end, **let $\Phi(i^*)$ be the closest facility $i \in F$ to i^***, i.e., $\Phi(i^*) = \operatorname{argmin}_{i \in F} d(i, i^*)$, and we assign i to $\Phi(i^*)$. Note this definition is consistent with the first case, so in either cases we are free to redirect j to $\Phi(\sigma^*(j)) = \Phi(i^*)$. To make subscripts more visible, we let $i' = \Phi(i^*)$, the new facility to serve j. Then

$$c_{i',j} \leqslant c_{i',i^*} + c_{i^*,j} \leqslant c_{i^*,j} + c_{i,i^*} \leqslant c_{i^*,j} + c_{i,j} + c_{j,i^*} = c_{i,j} + 2c_{i^*,j},$$

where the second $\leqslant$ is by definition of Φ, and the first and third are because of triangle inequality. In other words,

$$c_{i',j} - c_{i,j} \leqslant 2c_{i^*,j} \qquad \text{or} \qquad c_{\Phi(\sigma^*(j)),j} - c_{i,j} \leqslant 2c_{\sigma^*(j),j}. \tag{14}$$

This inequality is good, since we are associating change in costs to something entirely dependent on the optimal solution. Once again, by local optimality,

$$\overbrace{f_i}^{\text{gain of closing } i} \leqslant \underbrace{\sum_{j:\sigma(j)=i} \left[c_{\Phi(\sigma^*(j)),j} - c_{i,j}\right]}_{\text{total cost of closing } i} \leqslant 2 \sum_{j:\sigma(j)=i} c_{\sigma^*(j),j}.$$

But there is a problem: what if $\Phi(\sigma^*(j)) = i$ itself? Our analysis above fails because after i is closed, we do not know what to assign j to. To this end, we will define two types of facilities: the ones that are **safe** and those that are **unsafe**. We define the set of **safe facilities** to be

$$S = \{i \in F : \text{no } i^* \in F^* \text{ has } \Phi(i^*) = i\},$$

and unsafe as its complement. What we have shown above is that if $i \in S$ and $\sigma(j) = i$, then after closing i, we can redirect j to $i' = \Phi(\sigma^*(i))$ with

$$f_i \leqslant 2 \sum_{jL\sigma(j)=i} c_{\sigma^*(j),j}.$$

Now, what about unsafe facilities? If i is unsafe, we define the set of troubles to be

$$R_i = \{i^* \in F^* : \Phi(i^*) = i\}.$$

After closing i, we do need to relocate all j where $\sigma(j) = i$. A natural choice would be to consider $\tilde{i}$ as $\operatorname{argmin}_{i^* \in R_i} c_{i,i^*}$, for this facility is relatively close to i and would serve as a fairly well substitute.

To be used later: for any $i^* \in R_i \backslash \{\tilde{i}\}$, we know eventually they will be opened, but not yet. If we were to open such i^*, the clients affected would be those j satisfying $\sigma(j) = i$ but $\sigma^*(j) = i^*$. Consequently, by local optimality we have

$$f_{i^*} + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)=i^*}} \left[c_{i^*,j} - c_{i,j}\right] = f_{i^*} + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)=i^*}} \left[c_{\sigma^*(j),j} - c_{i,j}\right] \geqslant 0 \qquad \text{for all } i^* \in R_i \backslash \{\tilde{i}\}. \tag{15}$$

Analyzing SWAP

So now let us analyze the effects of swapping i with $\tilde{i}$. We need to relocate all clients j where $\sigma(j) = i$. Again, two cases:

- CASE 1. $i^* = \sigma^*(i) \notin R_i$. This means $\Phi(i^*) \neq i$ and is open by definition — we simply move j to $\Phi(i^*)$.
- CASE 2. $i^* \in R_i$ so we cannot move j to $\Phi(i^*) = i$. In this case, we make use of the newly opened $\tilde{i}$.

The old cost is $f_i + \sum_{\sigma(j)=i} c_{i,j}$ and the new cost is $f_{\tilde{i}} + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)\in R_i}} c_{\tilde{i},j} + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)\notin R_i}} c_{\Phi(\sigma^*(j)),j}$. By local optimality and splitting the sums, we see that

$$(f_{\tilde{i}} - f_i) + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)\notin R_i}} \left[c_{\Phi(\sigma^*(j)),j} - c_{i,j}\right] + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)\in R_i}} \left[c_{\tilde{i},j} - c_{i,j}\right] \geqslant 0. \tag{16}$$

Let us analyze the terms separately. The middle term, by (14), is bounded by $2\sum c_{\sigma^*(j),j}$. The third term can be bounded by

$$\sum \left[c_{\tilde{i},j} - c_{i,j}\right] \leqslant \sum c_{i,\tilde{i}} \leqslant \sum c_{i,i^*} = \sum c_{\sigma(j),\sigma^*(j)} \leqslant \sum \left[c_{\sigma(j),j} + c_{\sigma^*(j),j}\right]$$

where the first inequality is by triangle inequality, second by definition of $\tilde{i} \in R_i$, and third by triangle inequality again. Cleaning up, we obtain

$$(f_{\tilde{i}} - f_i) + 2\sum_{\substack{\sigma(j)=i \\ \sigma^*(j)\notin R_i}} c_{\sigma^*(j),j} + \sum_{\substack{\sigma(j)=i \\ \sigma^*(j)\in R_i}} \left[c_{i,j} + c_{\sigma^*(j),j}\right] \geqslant 0. \tag{17}$$

Now consider the sum of (17), along with (15), summed over each $i^* \in R_i \backslash \{\tilde{i}\}$. Three cases depending on the structure of j. Clearly the sum involves $\sum_{i\in R-i} f_i - f_{\tilde{i}}$, along with some costs, which has three cases depending on where $\sigma^*(j)$ is.

- If $\sigma^*(j) = i^* \in R_i \backslash \{\tilde{i}\}$, then the contribution is $(c_{\sigma^*(j),j} - c_{i,j}) + (c_{i,j} + c_{\sigma^*(j),j}) = 2c_{\sigma^*(j),j}$.
- If $\sigma^*(j) = i^* \notin R_i$, then the contribution is $2c_{\sigma^*(j),j}$.
- Finally, if $\sigma^*(j) = i^* \in R_i$, the contribution via the original expression is $c_{\tilde{i},j} - c_{i,j} \leqslant c_{\sigma^*(j),j} - c_{i,j} \leqslant c_{\sigma^*(j),j}$.

In all three cases, the contribution is no more than $2c_{\sigma^*(j),j}$, so we conclude that for a fixed i,

$$\sum_{i\in R_i} f_i - f_{\tilde{i}} + 2\sum_{\sigma(j)=i} c_{\sigma^*(j),j} \geqslant 0. \tag{18}$$

Finally, summing over all facilities i, we obtain the long-desired bound

$$F = \sum_i f_i \leqslant \underbrace{\sum_{i\in F}\sum_{i^*\in R_i} f_i}_{F^*} + 2\underbrace{\sum_{i\in F}\sum_{\sigma(j)=i} c_{\sigma^*(j),j}}_{C^*} = F^* + 2C^*.$$

This combined with (**??**) help us conclude that **local search yields a 3-aprpoximation**.

9.1 k-Median

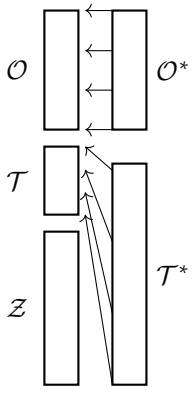
In this section, we consider the k-median problem, where given (V, c), we want to find k points, assign then as the centers of clusters, and partition V into k clusters, each corresponding to a center. The objective is to minimize the sum of distances for each v to the nearest assigned center.

To stay consistent with our notations in the previous section, let F of size k be a set of centers that we pick. The local search involves only one operation, SWAP, since in the end we also want to return a set of size k as the cluster centers. Once again, let F be an locally optimal solution and F^* be the global optimum. For $i^* \in F^*$, let $\Phi(i^*) = \operatorname{argmin}_{i \in F} c_{i,i^*}$, the closest point to i^* that is currently chosen. Finally, let σ, σ^* denote the mapping of points to their nearest center in F and F^*, respectively. We will show that F is a 5-approximation. We will also adopt the language from the previous section, treating the chosen centers as "facilities."

We first partition F, F^* into subsets based on the structure of Φ: each $i \in F$ may correspond to zero, one, or more $i^* \in F^*$ where $\Phi(i^*) = i$. Using initials, we let $\mathcal{Z}, \mathcal{O}$, and $\mathcal{T}$ to denote the sets

$$\mathcal{Z} = \{i \in F : \text{ no } i^* \in F^* \text{ satisfies } \Phi(i^*) = i\}$$
$$\mathcal{O} = \{i \in F : \text{ exactly one } i^* \in F \text{ satisfies } \Phi(i^*) = i\}$$
$$\mathcal{T} = \{i \in F : \text{ two or more } i^* \in F^* \text{ satisfy } \Phi(i^*) = i\}.$$

Correspondingly, we partition F^* into $\mathcal{O}^*$ and $\mathcal{T}^*$ where Φ maps $\mathcal{O}^*$ bijectively to $\mathcal{O}$, and $\mathcal{T}^*$ to $\mathcal{T}$, where the pre-image Φ^{-1} of each $i \in \mathcal{T}$ has at least two elements in $\mathcal{T}^*$ by definition. It follows that $|\mathcal{O}| = |\mathcal{O}^*|$, $|\mathcal{T}| \leqslant |\mathcal{T}^*|/2$, and $|\mathcal{Z}| \geqslant |\mathcal{T}^*|/2$.



We are primarily interested in SWAP operations that involve replacing some $i \in F$ with some $i^* \in F^*$. We call them the **critical swaps**. Based on the structure of $\mathcal{Z}, \mathcal{O}$, and $\mathcal{T}$, we are interested in the following two types:

(1) If $i^* \in \mathcal{O}^*$, then we swap $i = \Phi(i^*)$ for i^*, and

(2) If $i^* \in T^*$, then we swap any $i \in \mathcal{Z}$ for i^*, subject to the condition that i^* has been swapped via this type for at most once (so counting the new one, at most two). This is possible, since $|\mathcal{Z}| \geqslant |\mathcal{T}^*|/2$. The purpose of this seemingly arbitrary requirement will be made clear toward the end.

(3) Importantly, we do not allow $i \in \mathcal{T}$ to participate in critical swaps.

There is one caveat we need to consider when performing critical swaps as such. Suppose we swap $i \in F$ for $i^* \in F^*$. Suppose j is such that $\sigma(j) = i$ but $\sigma^*(j) = i' \in F^*$ where $i' \neq i^*$. If i' is not yet open, we should look for a workaround: assign j to $\Phi(i') = \Phi(\sigma^*(j))$, the closest currently open facility to the true optimum i'.

We need to ensure one thing to make our operation well-defined: when $\sigma^*(j) \neq i^*$, we must have $\Phi(i') = \Phi(\sigma^*(j)) \neq i$. To see this, first observe that $i \notin \mathcal{T}$ by the definition of a critical swap. If $i \in \mathcal{Z}$, there is nothing to show by the definition of $\mathcal{Z}$. On the other hand, if $i \in \mathcal{O}$, then by definition i^* is the only element in F^* that maps to i by Φ. This concludes the proof of this observation.

With this caveat taken care of, we are now ready to formally describe the effects a critical swap (i, i^*) imposes on all j's. We also have two cases after closing i and opening i^*:

- CASE 1. Regardless of anything else, if $\sigma^*(j) = i^*$, we certainly want to reassign j to i^*.
- CASE 2. If $\sigma^*(j) \neq i^*$ *and* $\sigma(j) = i$, we reassign j to $\Phi(\sigma^*(j))$.

The two cases successfully reassigns every affected j with $\sigma(j) = i$, as well as some additional j that may also benefit from an update (in CASE 1).

The rest of the proof is just triangle inequality and symbol pushing. Fix (i, i^*). For a single j as in CASE 1, we have

$$c_{i^*,j} - c_{i,j} = c_{\sigma^*(j),j} - c_{\sigma(j),j}.$$

For a single j as in CASE 2, we have

$$\begin{aligned} c_{\Phi(\sigma^*(j)),j} - c_{i,j} &\leqslant c_{\sigma^*(j),j} + c_{\Phi(\sigma^*(j)),\sigma^*(j)} - c_{i,j} && (\Delta\text{ – ineq}) \\ &\leqslant c_{\sigma^*(j),j} + c_{i,\sigma^*(j)} - c_{i,j} && (\text{by definition of } \Phi) \\ &\leqslant c_{\sigma^*(j),j} + c_{i,j} + c_{\sigma^*(j),j} - c_{i,j} \leqslant 2c_{\sigma^*(j),j}. && (\Delta\text{ – ineq again}) \end{aligned}$$

Therefore, for a single critical swap (i, i^*), summing over all affected j's gives

$$\sum_{\sigma^*(j)=i^*} \left[c_{\sigma^*(j),j} - c_{\sigma(j),j}\right] + 2 \sum_{\substack{\sigma^*(j)\neq i^* \\ \sigma(j)=i}} c_{\sigma^*(j),j}.$$

Now if we sum over all such critical swaps, the first term gives $\mathrm{OPT} - \mathrm{ALG}$; the second term, by the assumption that no $i \in \mathcal{Z}$ is used more than twice by critical swap pairs, yields $4\mathrm{OPT}$. Of course, ALG is locally optimal, so this implies $\mathrm{OPT} - \mathrm{ALG} + 4\mathrm{OPT} \geqslant 0$, and in other words we have obtained a 5-approximation, as claimed.