Repetition: Primal Dual for Set Cover

Primal Relaxation:

$$\begin{array}{llrcl} \min & & \sum_{i=1}^{k} w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u \in S_i} x_i & \geq & 1 \\ & \forall i \in \{1,\dots,k\} & x_i & \geq & 0 \end{array}$$

Dual Formulation:

$$\begin{array}{llrcl} \max & & \sum_{u \in U} y_u & & \\ \text{s.t.} & \forall i \in \{1,\dots,k\} & \sum_{u:u \in S_i} y_u & \leq & w_i \\ & & y_u & \geq & 0 \end{array}$$

Repetition: Primal Dual for Set Cover

Algorithm:

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While x not feasible
  - Identify an element e that is not covered in current primal integral solution.
  - Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e \le f \cdot \sum_e y_e \le f \cdot \mathrm{OPT}$$

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

If we would also fulfill dual slackness conditions

$$y_e > 0 \Rightarrow \sum_{j : e \in S_j} x_j = 1$$

then the solution would be optimal!!!

We don't fulfill these constraint but we fulfill an approximate version:

$$y_e > 0 \Rightarrow 1 \leq \sum_{j:e\in S_j} x_j \leq f$$

This is sufficient to show that the solution is an f-approximation.

Suppose we have a primal/dual pair

min		$\sum_j c_j x_j$		
s.t.	$\forall i$	$\sum_{j:} a_{ij} x_j$	$\geq$	b_i
	$\forall j$	x_j	$\geq$	0

max		$\sum_i b_i y_i$		
s.t.	$\forall j$	$\sum_i a_{ij} y_i$	$\leq$	c_j
	$\forall i$	y_i	$\geq$	0

and solutions that fulfill approximate slackness conditions:

$$x_j > 0 \Rightarrow \sum_i a_{ij} y_i \geq \frac{1}{\alpha} c_j$$

$$y_i > 0 \Rightarrow \sum_j a_{ij} x_j \leq \beta b_i$$

Then

right hand side of j-th dual constraint

$$\underbrace{\sum_j \boxed{c_j} x_j}_{\text{primal cost}} \le \alpha \sum_j \left(\sum_i a_{ij} y_i\right) x_j$$

$$= \alpha \sum_i \left(\sum_j a_{ij} x_j\right) y_i$$

$$\le \alpha\beta \cdot \underbrace{\sum_i b_i y_i}_{\text{dual objective}}$$

Feedback Vertex Set for Undirected Graphs

- Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.
- Choose a minimum cost subset of vertices s.t. every cycle contains at least one vertex.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.
- However, this encoding gives a Set Cover instance of non-polynomial size.
- The $O(\log n)$-approximation for Set Cover does not help us to get a good solution.

Let $\mathcal{C}$ denote the set of all cycles (where a cycle is identified by its set of vertices)

Primal Relaxation:

$$\begin{array}{llrcl}
\min & & \sum_v w_v x_v & & \\
\text{s.t.} & \forall C \in \mathcal{C} & \sum_{v \in C} x_v & \geq & 1 \\
& \forall v & x_v & \geq & 0
\end{array}$$

Dual Formulation:

$$\begin{array}{llrcl}
\max & & \sum_{C \in \mathcal{C}} y_C & & \\
\text{s.t.} & \forall v \in V & \sum_{C : v \in C} y_C & \leq & w_v \\
& \forall C & y_C & \geq & 0
\end{array}$$

If we perform the previous dual technique for Set Cover we get the following:

- Start with $x = 0$ and $y = 0$
- While there is a cycle C that is not covered (does not contain a chosen vertex).
  - Increase y_C until dual constraint for some vertex v becomes tight.
  - set $x_v = 1$.

Then

$$\begin{aligned}\sum_v w_v x_v &= \sum_v \sum_{C:v\in C} y_C x_v \\ &= \sum_{v\in S} \sum_{C:v\in C} y_C \\ &= \sum_C |S\cap C|\cdot y_C\end{aligned}$$

where S is the set of vertices we choose.

If every cycle is short we get a good approximation ratio, but this is unrealistic.

Algorithm 1 FeedbackVertexSet

1: $y \leftarrow 0$
2: $x \leftarrow 0$
3: **while** exists cycle C in G **do**
4: increase y_C until there is $v \in C$ s.t. $\sum_{C: v \in C} y_C = w_v$
5: $x_v = 1$
6: remove v from G
7: repeatedly remove vertices of degree 1 from G

Idea:

Always choose a short cycle that is not covered. If we always find a cycle of length at most α we get an α-approximation.

Observation:

For any path P of vertices of degree 2 in G the algorithm chooses at most one vertex from P.

Observation:

If we always choose a cycle for which the number of vertices of degree at least 3 is at most α we get a 2α-approximation.

Theorem 2

In any graph with no vertices of degree 1, there always exists a cycle that has at most $\mathcal{O}(\log n)$ vertices of degree 3 or more. We can find such a cycle in linear time.

This means we have

$$y_C > 0 \Rightarrow |S \cap C| \le \mathcal{O}(\log n) \ .$$

Primal Dual for Shortest Path

Given a graph $G = (V, E)$ with two nodes $s, t \in V$ and edge-weights $c : E \to \mathbb{R}^+$ find a shortest path between s and t w.r.t. edge-weights c.

$$\begin{array}{llrcl} \min & & \sum_e c(e) x_e & & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0,1\} \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S: e \in \delta(S)} y_S & \le & c(e) \\ & \forall S \in \mathcal{S} & y_S & \ge & 0 \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surounding the set S.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Algorithm 1 PrimalDualShortestPath

$y \leftarrow 0$
$F \leftarrow \emptyset$
while there is no s-t path in (V, F) **do**
    Let C be the connected component of (V, F) containing s
    Increase y_C until there is an edge $e' \in \delta(C)$ such that $\sum_{S: e' \in \delta(S)} y_S = c(e')$.
    $F \leftarrow F \cup \{e'\}$
Let P be an s-t path in (V, F)
return P

Lemma 3

At each point in time the set F forms a tree.

Proof:

- In each iteration we take the current connected component from (V, F) that contains s (call this component C) and add some edge from $\delta(C)$ to F.
- Since, at most one end-point of the new edge is in C the edge cannot close a cycle.

$$\sum_{e\in P} c(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$
$$= \sum_{S:s\in S, t\notin S} |P \cap \delta(S)| \cdot y_S \ .$$

If we can show that $y_S > 0$ implies $|P \cap \delta(S)| = 1$ gives

$$\sum_{e\in P} c(e) = \sum_S y_S \leq \text{OPT}$$

by weak duality.

Hence, we find a shortest path.

If S contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S, S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

Steiner Forest Problem:

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \dots, k$, and a cost function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F.

$$\begin{array}{llrcl} \min & & \sum_e c(e) x_e & & \\ \text{s.t.} & \forall S \subseteq V : S \in S_i \text{ for some } i & \sum_{e \in \delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0, 1\} \end{array}$$

Here S_i contains all sets S such that $s_i \in S$ and $t_i \notin S$.

$$\begin{array}{llrcl} \max & & \sum_{S\,:\,\exists i \text{ s.t. } S\in\mathcal{S}_i} y_S & & \\ \text{s.t.} & \forall e\in E & \sum_{S:e\in\delta(S)} y_S & \le & c(e) \\ & & y_S & \ge & 0 \end{array}$$

The difference to the dual of the shortest path problem is that we have many more variables (sets for which we can generate a moat of non-zero width).

Algorithm 1 FirstTry

$y \leftarrow 0$
$F \leftarrow \emptyset$
while not all s_i-t_i pairs connected in F **do**
    Let C be some connected component of (V, F) such that $|C \cap \{s_i, t_i\}| = 1$ for some i.
    Increase y_C until there is an edge $e' \in \delta(C)$ s.t. $\sum_{S \in S_i : e' \in \delta(S)} y_S = c_{e'}$
    $F \leftarrow F \cup \{e'\}$
return $\bigcup_i P_i$

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F| \cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.
- The i-th pair is v_0-v_i.
- The first component C could be $\{v_0\}$.
- We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.
- The final set F contains all edges $\{v_0, v_i\}$, $i = 1, \dots, k$.
- $y_{\{v_0\}} > 0$ but $|\delta(\{v_0\})\cap F| = k$.

Algorithm 1 SecondTry

$y \leftarrow 0$; $F \leftarrow \emptyset$; $\ell \leftarrow 0$

while not all s_i-t_i pairs connected in F **do**

  $\ell \leftarrow \ell + 1$

  Let $\mathcal{C}$ be set of all connected components C of (V, F) such that $|C \cap \{s_i, t_i\}| = 1$ for some i.

  Increase y_C for all $C \in \mathcal{C}$ uniformly until for some edge $e_\ell \in \delta(C')$, $C' \in \mathcal{C}$ s.t. $\sum_{S: e_\ell \in \delta(S)} y_S = c_{e_\ell}$

  $F \leftarrow F \cup \{e_\ell\}$

$F' \leftarrow F$

for $k \leftarrow \ell$ downto 1 **do** // reverse deletion

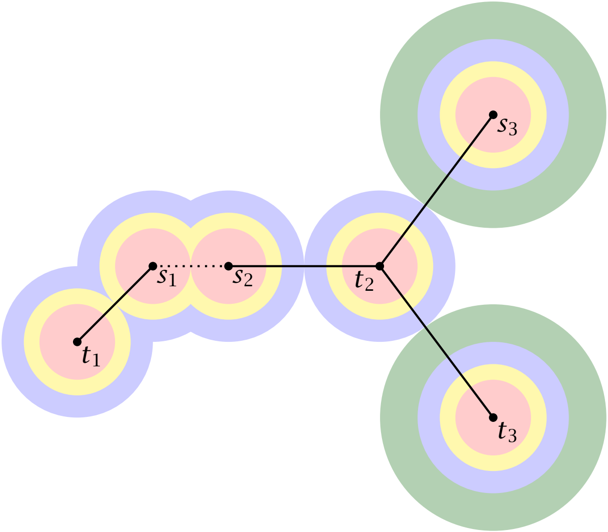
  if $F' - e_k$ is feasible solution **then**

    remove e_k from F'

return F'

The reverse deletion step is not strictly necessary this way. It would also be sufficient to simply delete all unnecessary edges in any order.

Example



Lemma 4
For any $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

This means that the number of times a moat from $\mathcal{C}$ is crossed in the final solution is at most twice the number of moats.

Proof: later...

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the i-th iteration the increase of the left-hand side is

$$\epsilon \sum_{C\in \mathcal{C}} |F' \cap \delta(C)|$$

  and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.
- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

Lemma 5

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration i. e_i is the set we add to F. Let F_i be the set of edges in F at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in H are necessary for the solution.

- Contract all edges in F_i into single vertices V'.
- We can consider the forest H on the set of vertices V'.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from C (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in C} |\delta(C) \cap F'| \overset{?}{\leq} 2|C| = 2|R|$$

- Suppose that no node in B has degree one.
- Then

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \leq 2(|R| + |B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from H, and, hence, is necessary.
  - But this means that the cluster corresponding to b must separate a source-target pair.
  - But then it must be a red node.