

Minimum-weight spanning tree problem

Used in design of communication networks:

Want all vertices connected, but links are expensive.

Given a connected graph $G = (V, E)$, weight function $w_e \geq 0$ on edges of E .

Build a ~~tree~~ ^{subgraph} $T = (V, E')$ such that T is connected and has minimum weight $\equiv \sum_{e \in E'} w_e = w(T)$

Greedy algorithm

Order edges by nondecreasing weight, so

$$w(e_1) \leq w(e_2) \leq \dots \leq w(e_n)$$

Start with $E' = \emptyset$

Construct E' as follows:

Proceed through edges in order

If $E' \cup \{e_i\}$ is acyclic, add $\{e_i\}$ to E'

Theorem The Greedy Algorithm terminates with a spanning tree T^* of minimum weight.

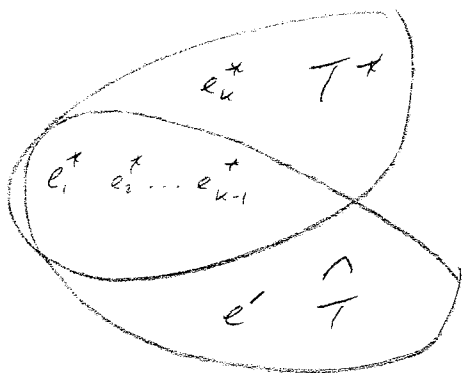
Proof Let $T^* = \{e_1^*, e_2^*, \dots, e_{n-1}^*\}$, where e_i^* is the i th edge selected by the algorithm.

Note T^* is a spanning tree.

Let \hat{T} be an optimal spanning tree.

Say \hat{T} and T^* agree to p places, $1 \leq p \leq n-2$, if $e_1^*, \dots, e_p^* \in \hat{T}$ and $e_{p+1}^* \notin \hat{T}$, agree to 0 places if $e_1^* \notin \hat{T}$, agree to $n-1$ places if $e_1^*, \dots, e_{n-1}^* \in \hat{T}$.

Suppose T^* and \hat{T} agree to $k-1$ places, $1 \leq k \leq n$. We will show there is another optimal spanning tree \tilde{T} such that T^* and \tilde{T} agree to $\geq k$ places.



Consider the circuit $C(\hat{T}, e_k^*)$ formed when e_k^* is added to \hat{T} .

$\exists e' \in C \setminus T^*$, and let $\tilde{T} = (\hat{T} + e_k^*) \setminus e'$

We know \tilde{T} is a spanning tree of G .

$w(\tilde{T}) = w(\hat{T}) + w(e_k^*) - w(e')$, and $w(e_k^*) \leq w(e')$ because

$\{e_1^*, \dots, e_{k-1}^*, e'\} \subseteq \hat{T}$, so \exists a cycle, and e_k^* was chosen instead of

Thus, $w(\tilde{T}) \leq w(\hat{T})$, so \tilde{T} is optimal.

Furthermore, T^* and \tilde{T} agree on $\geq k$ places //

How to check whether $T + e_j$ is acyclic?

As algorithm proceeds, keep a label $l(v)$ at each vertex $v \in V$ indicating the component of T to which v belongs.

Initially, $l(i) = i \quad \forall i = 1, \dots, n$

Let e_j have ends x, y . If $l(x) \neq l(y)$, then $T + e_j$ is acyclic.

$PTR(v)$ stores root of component containing v .

Merge smaller into larger.

The greedy algorithm requires $\sim m \log n$ operations (see P85, p. 272 -
 sorting $\sim m \log n$ operations \leftarrow dominant cost

Prim's algorithm

Divide V into 2 sets: V_c in tree
 V_o out of tree.

Add closest vertex in V_o to V_c , add the corresponding edge to the tree.

Algorithm: (Outputs min spanning tree (V, E'))

Step 1 (Initialize) \swarrow arbitrary

$V_c \leftarrow \{v_1\}$, $V_o \leftarrow V \setminus \{v_1\}$, $E' \leftarrow \emptyset$

for all $v \in V_o$, set $\text{closest}(v) \leftarrow v_{o1}$.

Step 2 (Loop)

While $V_c \neq V$,

Set $\text{min} = \infty$

for all $v \in V_o$,

if $d(v, \text{closest}(v)) < \text{min}$ then $\text{min} \leftarrow d(v, \text{closest}(v))$, $\text{next} \leftarrow v$

(i.e., find the node in V_o that is closest to V_c)

$V_c \leftarrow V_c \cup \{\text{next}\}$, $T \leftarrow T \cup \{\text{next}, \text{closest}(\text{next})\}$ (Update V_c)
 $V_o \leftarrow V_o \setminus \{\text{next}\}$

~~for all~~

for all $v \in V_o$,

if $d(v, \text{closest}(v)) > d(v, \text{next})$ then $\text{closest}(v) \leftarrow \text{next}$ (Update closest)

Algorithm has n stages, each vertex in V_o is examined at each stage

so need $\sim n^2$ operations.

Thm Let G be a connected graph and let U be a subtree of G such that there exists a minimum weight spanning tree T^* of G with $E(U) \subseteq E(T^*)$, $E(U) \neq E(T^*)$. For $k \in V(U)$ and $l \in V(G) \setminus V(U)$ such that $\{k, l\} \in E(G)$ and w_{kl} is as small as possible, then $U + \{k, l\}$ is contained in an optimal spanning tree of G .

Proof Suppose $e = \{k, l\} \notin E(T^*)$.

Then $C = C(T^*, e)$ contains an edge $e' \in [V(U), V(G) \setminus V(U)]$

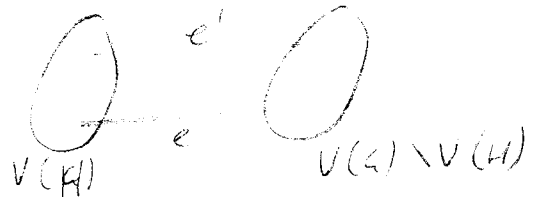
Then $\tilde{T} = (T^* - e) \cup e'$ is

a spanning tree of G .

$$w(\tilde{T}) = w(T^*) + w(e) - w(e') \leq w(T^*)$$

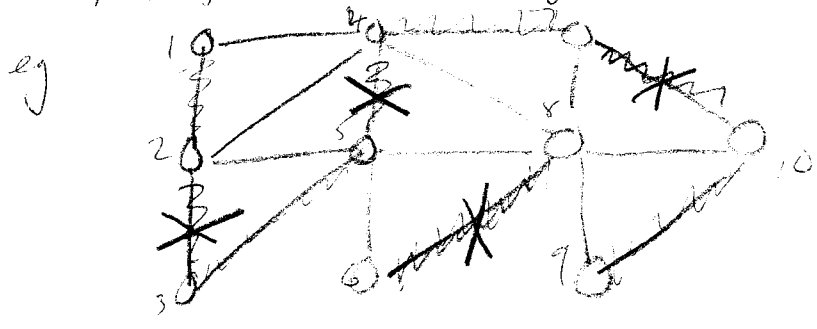
So \tilde{T} is optimal and $E(U) \cup e \subseteq E(\tilde{T})$ //

Hence Prim's algorithm works.



Matching.

A matching M of a graph $G = (V, E)$ is a subset of the edges with the property that no two edges of M share the same node.



$(2,3), (4,5), (6,8), (7,9)$

maximal, not maximum

$(1,2), (3,5), (4,7), (6,9), (8,10)$
maximum.

Matching problem: Given a graph $G = (V, E)$ find a maximum matching M of G .

Perfect matching: one which uses $|V|/2$ edges.

Consider a graph G with a fixed matching M .

Edges in M are matched edges; other edges are free.

Nodes not incident on any matched edge are exposed; others are matched.

A path $p = [u_1, u_2, u_3, \dots, u_k]$ is called alternating if its edges

$[u_1, u_2], [u_3, u_4], \dots, [u_{2i-1}, u_{2i}], \dots$ are free, whereas

$[u_2, u_3], [u_4, u_5], \dots, [u_{2i}, u_{2i+1}], \dots$ are matched.

An alternating path is called augmenting if both u_1 and u_k are exposed vertices.

Lemma Let P be the set of edges on an augmenting path $p = \{u_1, u_2, \dots, u_{2k}\}$ in a graph G with a matching M . Then $M' = M \oplus P$ is a matching of cardinality $|M| + 1$.

Symmetric difference
 $(M - P) \cup (P - M)$

Proof $M \oplus P$ is a matching:

Proof by contradiction: Assume two edges $e, e' \in M \oplus P$ share the same node of G .

Now $M \oplus P = (M - P) \cup (P - M)$, so we have three cases:

- i) $e, e' \in M - P \Rightarrow e, e' \in M$ ~~✗~~
- ii) $e, e' \in P - M \Rightarrow e, e'$ both of form $\{u_{2j-1}, u_{2j}\}$ ~~✗~~
- iii) $e \in P - M, e' \in M - P \Rightarrow e = \{u_{2j}, u_{2j+1}\}$ for some j , and e' has (say) u_{2j} as one endpoint, with other end point off path. But $\{u_{2j}, u_{2j+1}\} \in M$ ~~✗~~.

$|M'| = |M| + 1$:

P contains $2k-1$ edges, k free and $k-1$ in M .

$|M'| = |M - P| + |P - M| = |M| - (k-1) + k = |M| + 1.$

Theorem A matching M in a graph G is maximum if and only if there is no augmenting path in G with respect to M .

Proof \Rightarrow : follows from Lemma.

\Leftarrow : Assume there is no augmenting path in G w.r.t M , and also M is not maximum.

So \exists matching M' with $|M'| > |M|$

Consider the edges in $M \oplus M'$. Let $G' = (V, M \oplus M')$.

Each vertex is adjacent to at most two edges in $M \oplus M'$.

If degree of vertex = 2, one edge is in M , the other in M' .

So all components are either circuits of even length or alternating paths.

In all circuits, same number of edges from M as from M' .

Because $|M'| > |M|$, one of alternating paths must have more edges from M' .

Then for M : this is an augmenting path w.r.t M . ~~⊥~~

Suggests algo:

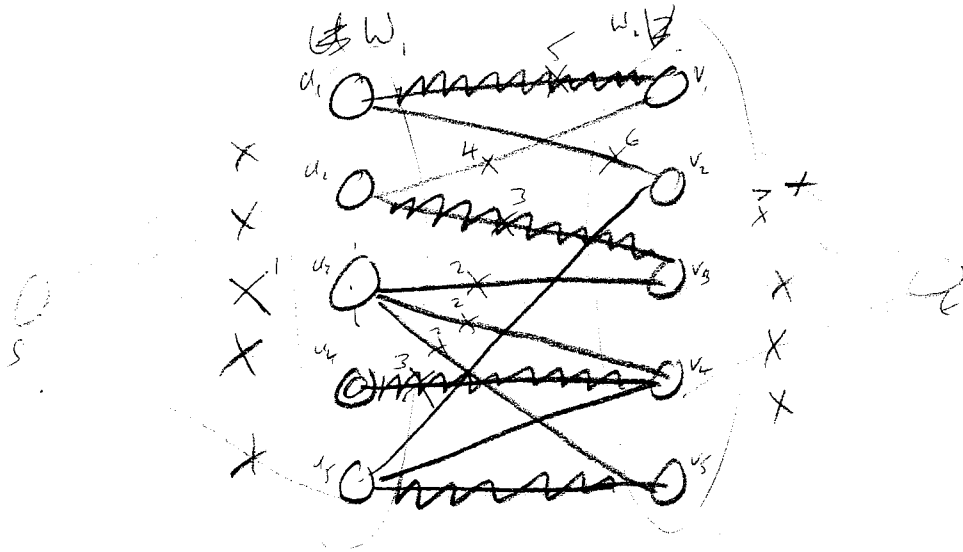
Find augmenting path P w.r.t current matching M .

Improve matching to $(M - P) \cup (P - M)$.

This algo works; problems arise from reliance of finding augmenting paths.

Graph is bipartite

Problem can be recast as a max-flow problem:



All edges have capacity one, directed from left to right.

Get max flow augmenting path $s, u_3, v_3, u_2, v_1, u_1, v_2, t$

Correspond to matching augmenting path $u_3, v_3, u_2, v_1, u_1, v_2$

Alternative method:

Grow tree of ~~augmenting~~ alternating paths from an exposed node u_i in W_1 . ~~if there is a path~~

if \exists augmenting path starting at this node, we will find it,

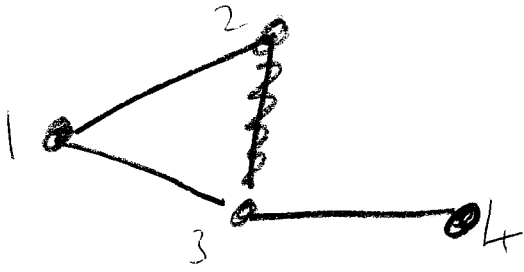
because every node ~~is~~ on the path in W_1 will be ~~at~~ found after traversing a matched edge, and every node ~~is~~ on the path in W_2 will be found after traversing a free edge.

This polarity is important.

General graphs

It can not easily be recast as max flow problem.

Also, tree growing algorithm does not work:



Grow tree from 1:

At first stage, label 2 and 3. — label them ODD

From each vertex, look for a matched edge. But opposite end of matched edge is already labelled, so terminate.

odd distance
from root vertex 1.

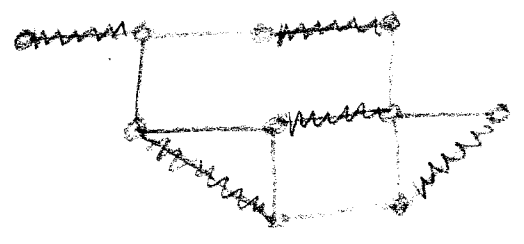
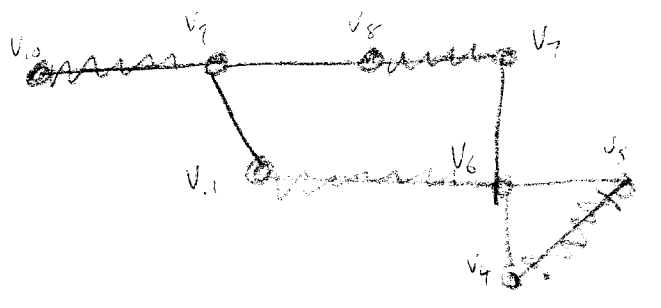
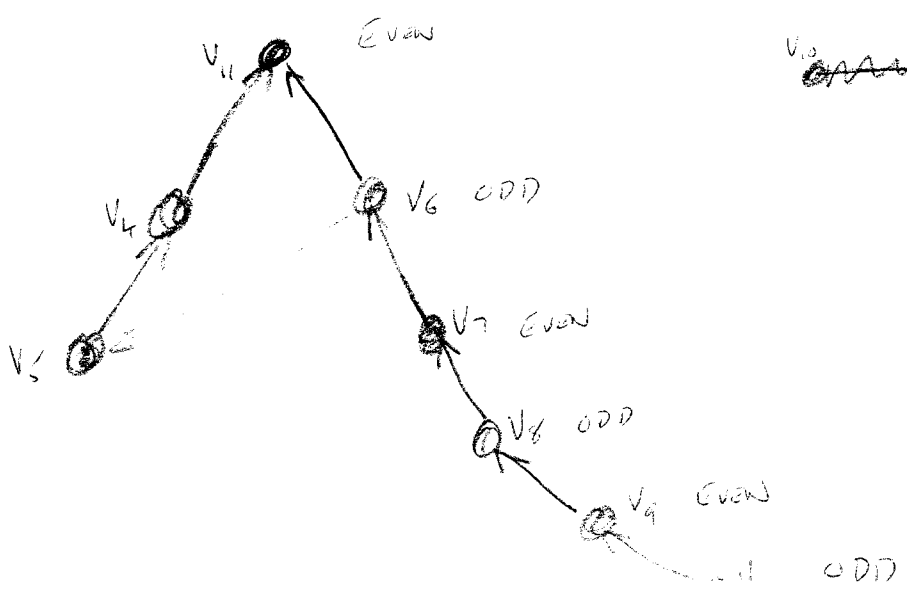
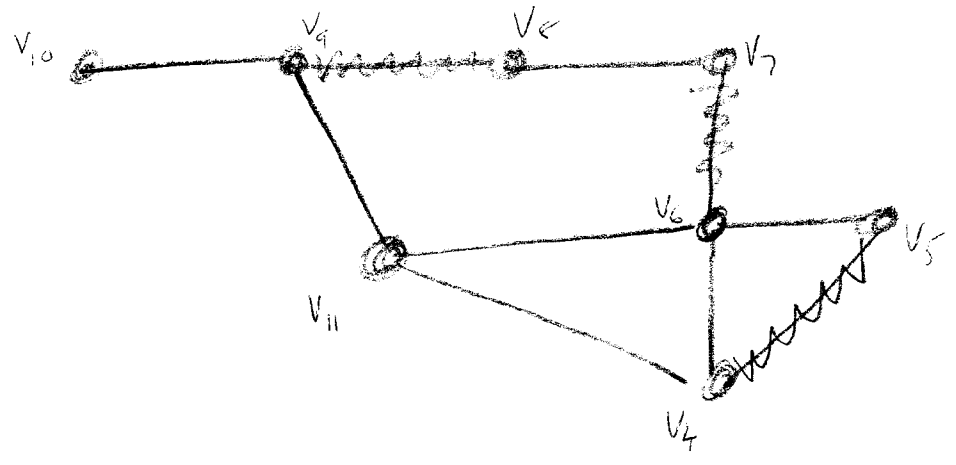
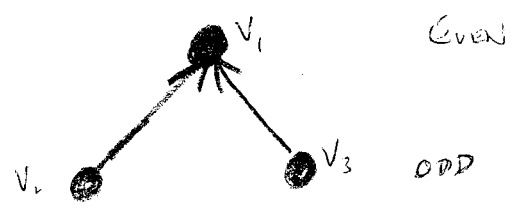
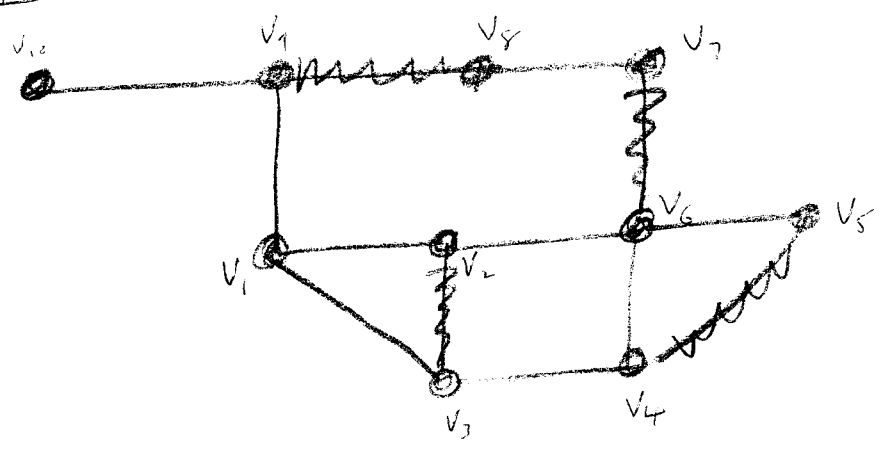
Problem is odd circuit 1, 2, 3:

There are two ~~augmenting~~ alternating paths from 1 to 3, one with even length, the other with odd length.

This ~~also~~ indicates the presence of a blossom: an odd cycle of length $2s+1$ containing s matched edges.

What we do is ~~replace~~ construct an auxiliary graph where the blossom is shrunk to a single ~~edge~~ node. ~~If~~ Any edge out of the blossom is replaced by an edge from the shrunk node to the i in the auxiliary graph.

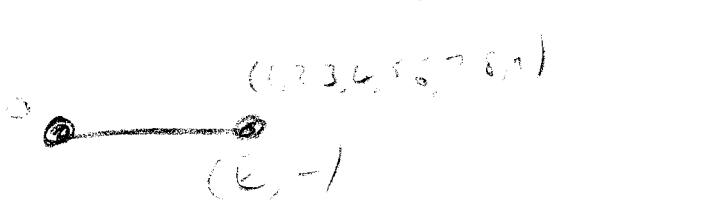
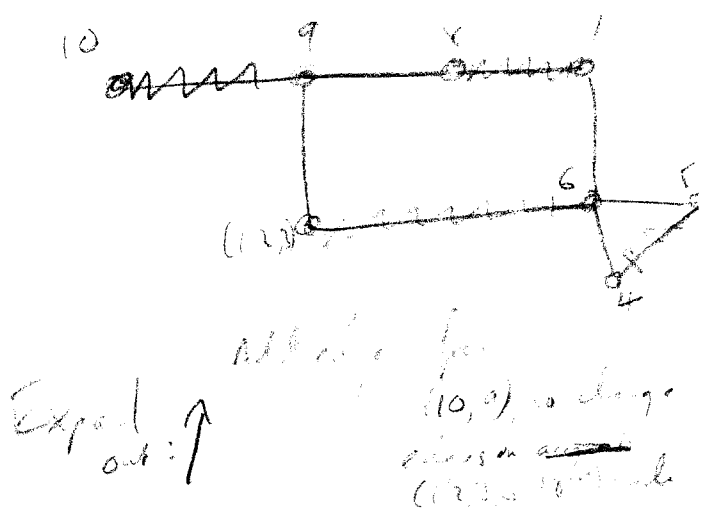
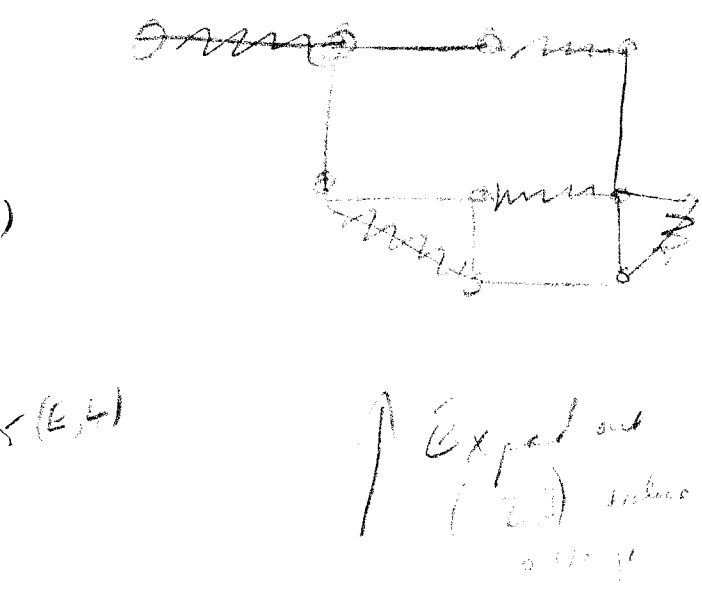
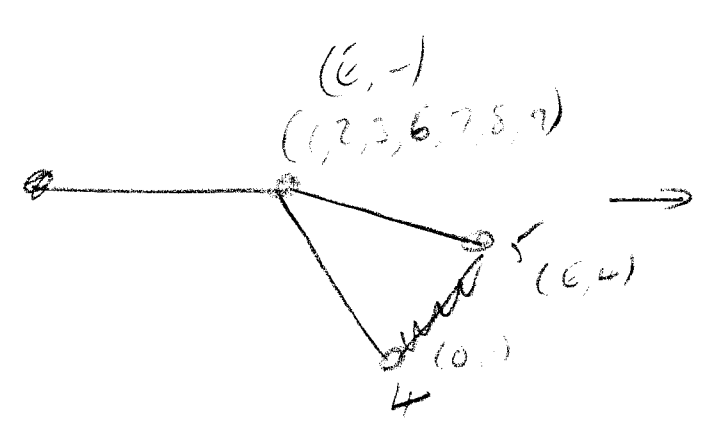
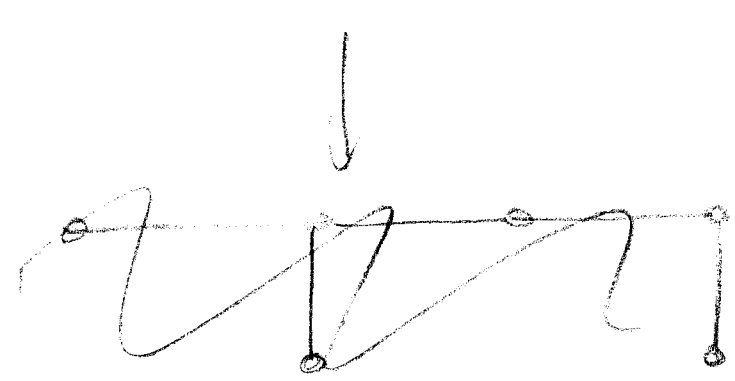
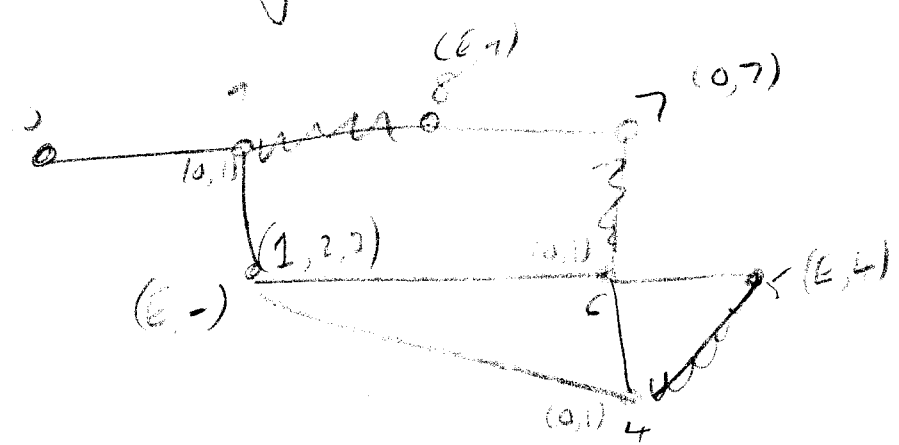
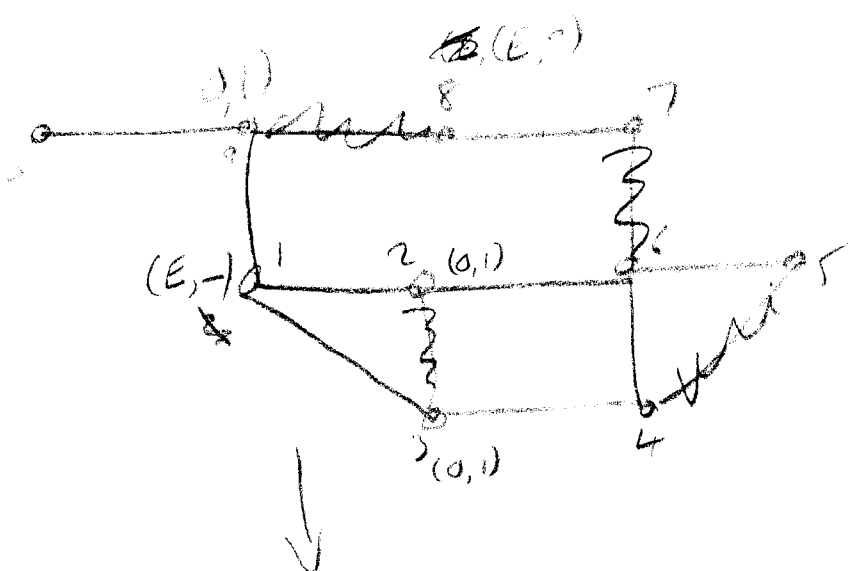
Example

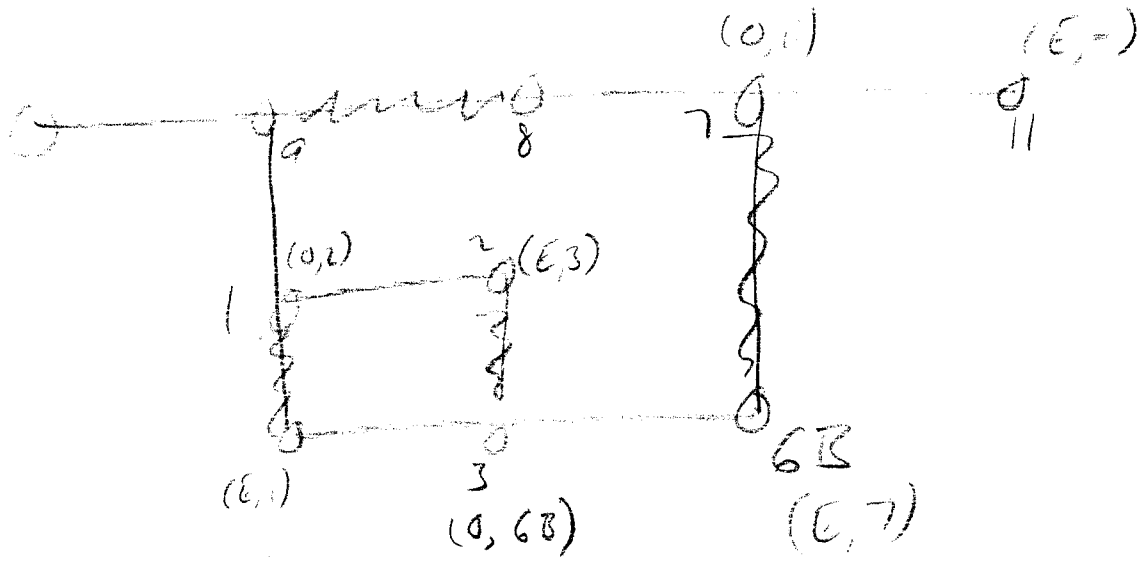
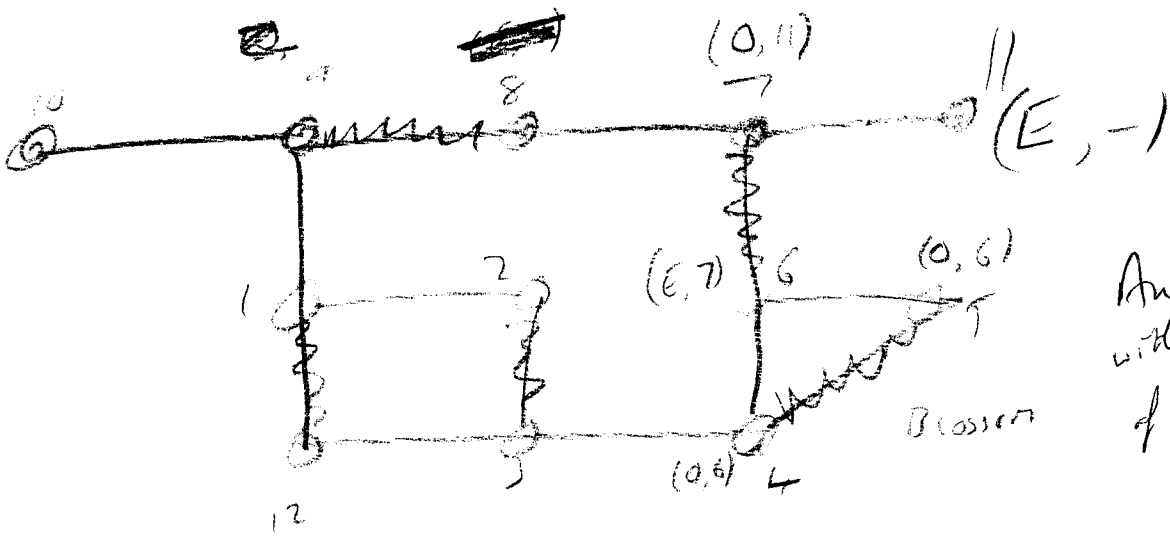


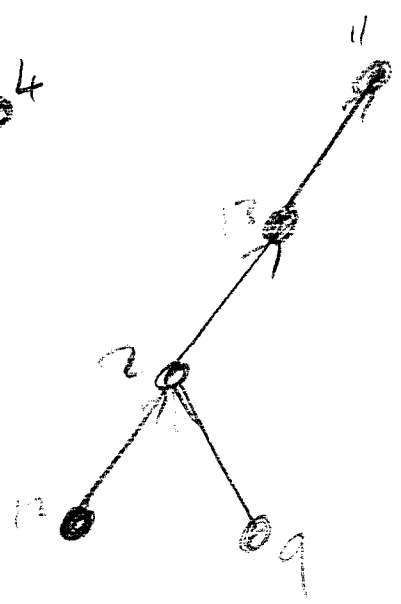
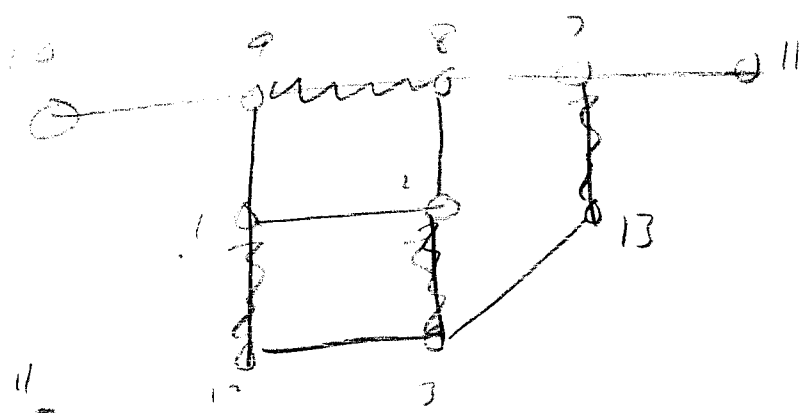
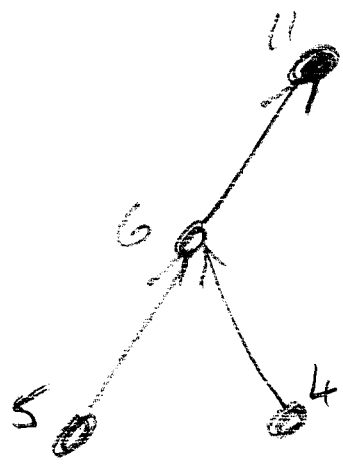
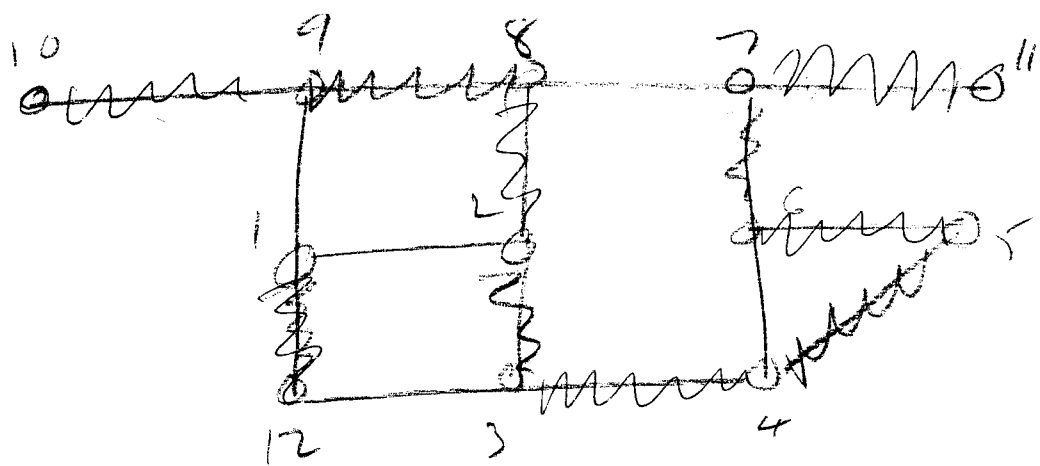
So, if we encounter a blossom, we shrink it and continue the alternating path, if possible.

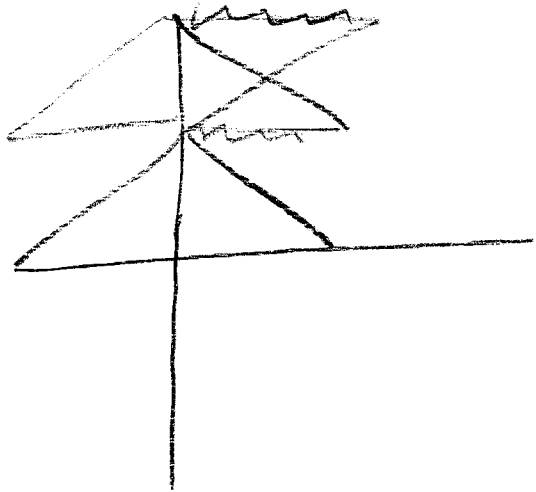
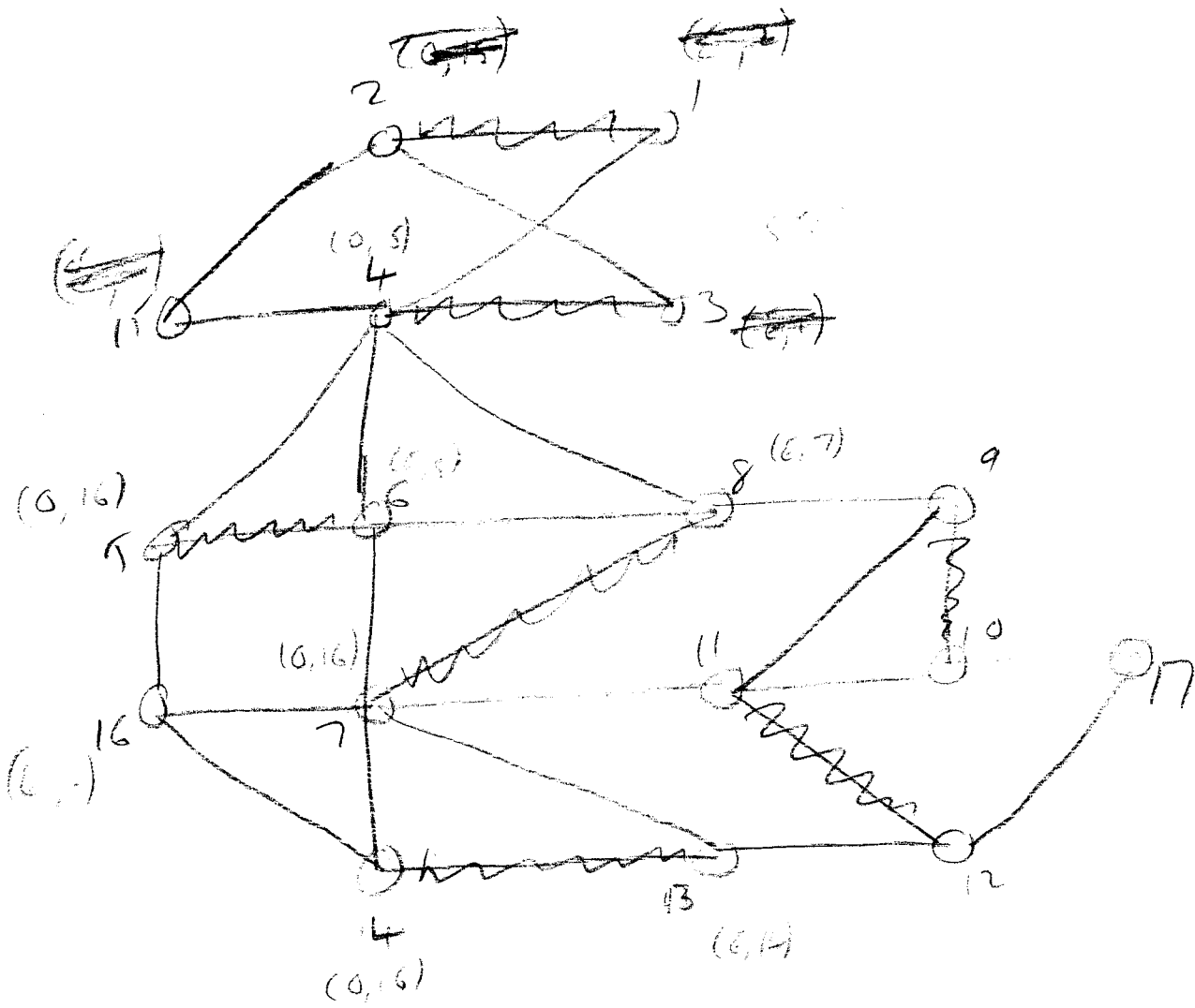
Shrinking does not destroy or create any augmenting paths.

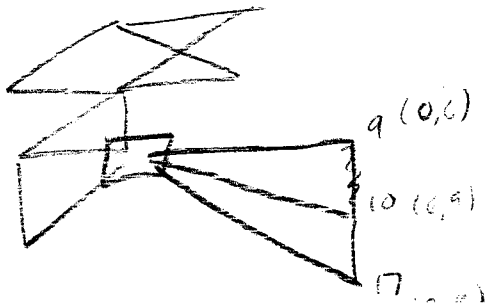
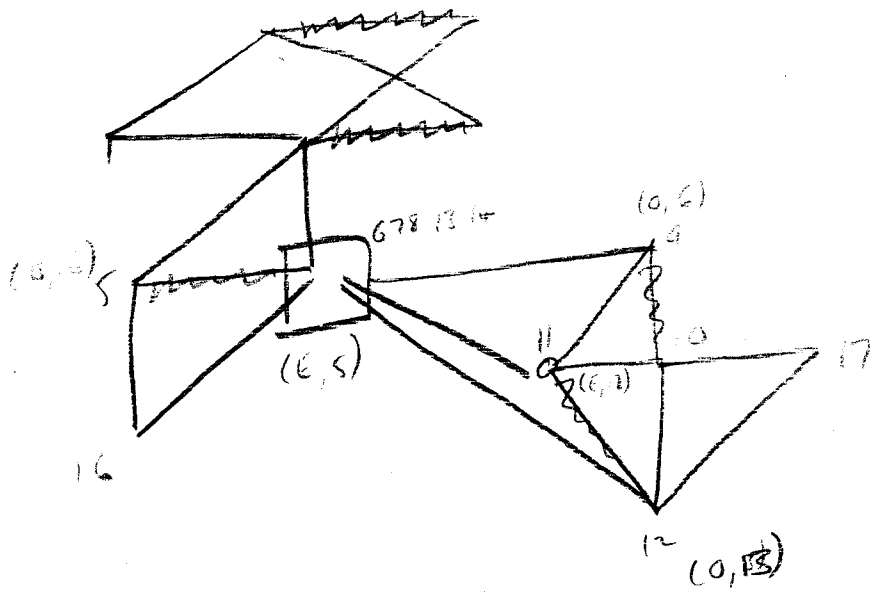
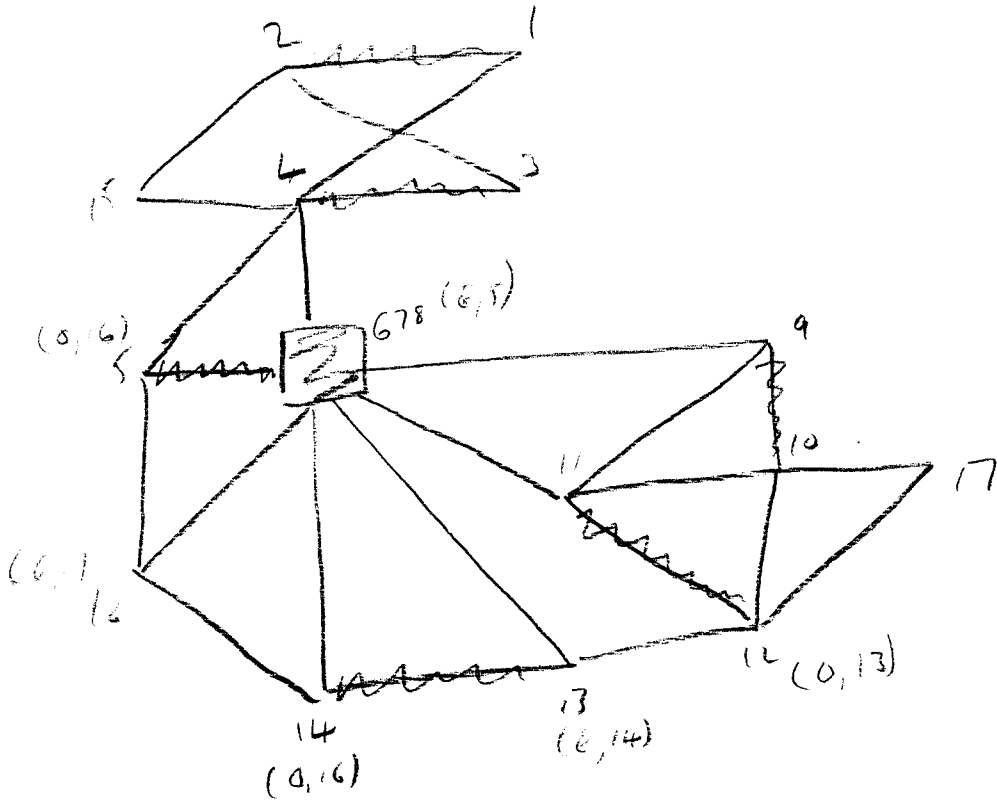
Let v and w be nodes











Weighted matching

Given weights $w_{ij} \geq 0$ on edges (v_i, v_j)

Want to find a matching which has largest weight $w(M) = \sum_{e \in M} w_e$.

(Special case of unweighted matching problem: take $w_{ij} = 1 \forall (i,j) \in E$.)

~~Can change problem to perfect matching problem~~ (Bipartite case also.)

Can assume graph is complete: give edges not in E weight 0.

Since $w_{ij} \geq 0$, optimal matching will be a perfect matching.

Take cost $c_{ij} = W - w_{ij}$ where $W > w_{ij} \forall i,j$.

The matching problem $\max \sum_{e \in M} w_e$
M matching

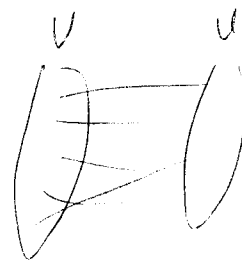
is equivalent to the perfect matching problem

$\min \sum_{e \in M} c_e$
M perfect matching.

Let $x_{ij} = \begin{cases} 1 & \text{if edge } e \text{ is in } M \\ 0 & \text{o/w} \end{cases}$

Perfect matching problem is
 $\min \sum_{i,j} c_{ij} x_{ij}$
s.t. $\sum_{j=1}^n x_{ij} = 1$

Bipartite case: Two sets of vertices V, U :



$$\text{Let } x_{ij} = \begin{cases} 1 & \text{if } (v_i, u_j) \in M \\ 0 & \text{o/w} \end{cases}$$

The perfect matching problem is:

$$\begin{aligned} \min \quad & \sum_{ij} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^n x_{ij} = 1 \quad \forall i \\ & \sum_{i=1}^m x_{ij} = 1 \quad \forall j \\ & x_{ij} \text{ binary } \forall i, j. \end{aligned}$$

Solution to this is solution to

$$\begin{aligned} \min \quad & \sum_{ij} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^n x_{ij} = 1 \\ & \sum_{i=1}^m x_{ij} = 1 \\ & x_{ij} \geq 0 \quad \forall ij \end{aligned} \quad (P)$$

Dual (D):

$$\begin{aligned} \max \quad & \sum_i \alpha_i + \sum_j \beta_j \\ \text{s.t.} \quad & \alpha_i + \beta_j \leq c_{ij} \end{aligned} \quad (D)$$

$$\text{C.S.} \quad x_{ij}(c_{ij} - \alpha_i - \beta_j) = 0.$$

Hungarian algorithm:

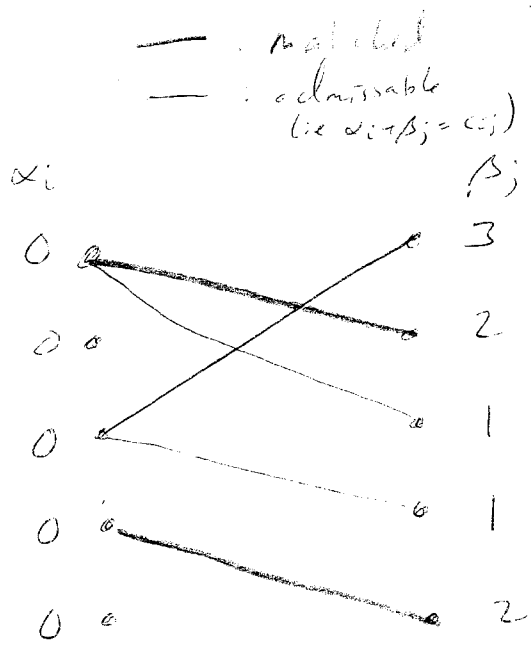
Start with (D)-feas, comp slackness.

Work to satisfy (P)-feas by using augmenting paths, and adjusting dual variables.

Example:

Table of costs:

		U				
		u_1	u_2	u_3	u_4	u_5
V	v_1	7	2	1	9	4
	v_2	9	6	9	5	5
	v_3	3	8	3	1	8
	v_4	7	9	4	2	2
	v_5	8	4	7	4	8



Initial solution: $\alpha_i = 0 \forall i$, $\beta_j = \min_i \{c_{ij}\}$

x_{ij} can be one if $\alpha_i + \beta_j = c_{ij}$.

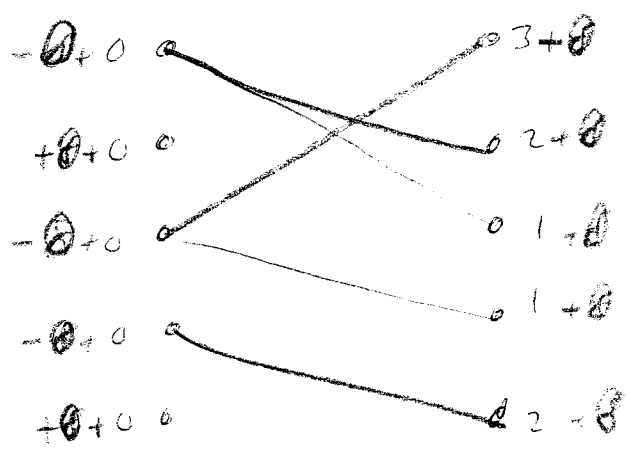
Now perturb weights in order

to attempt to match

vertices v_2 and v_5 :

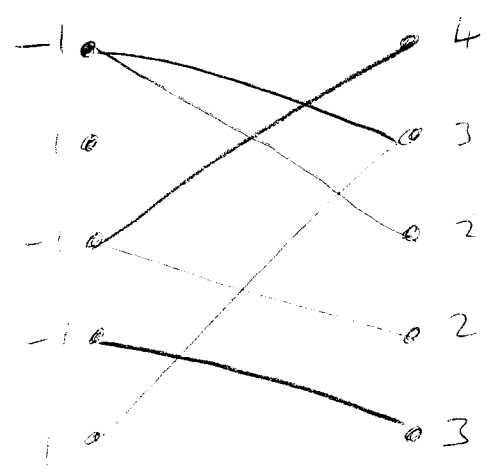
Trying to make

$\alpha_i + \beta_j = c_{ij}$
 or $\alpha_i + \beta_j = c_{ij}$
 for some j .

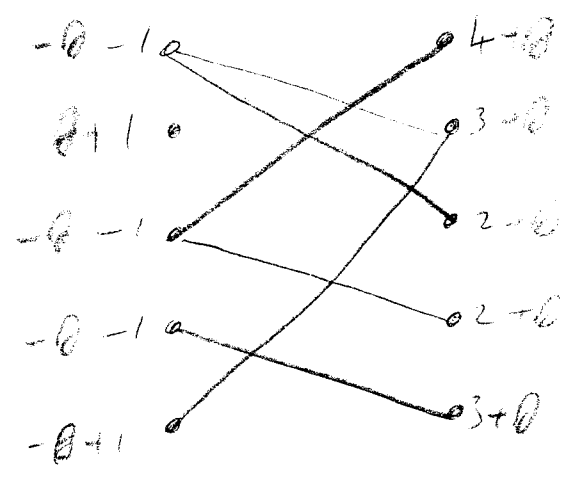


Largest possible θ is 1, constrained by edge ~~(v_5, u_2)~~ (v_5, u_2)
 (Need to keep $\alpha_i + \beta_j \leq c_{ij} \forall i, j$).

So now (v_5, u_2) becomes admissible:

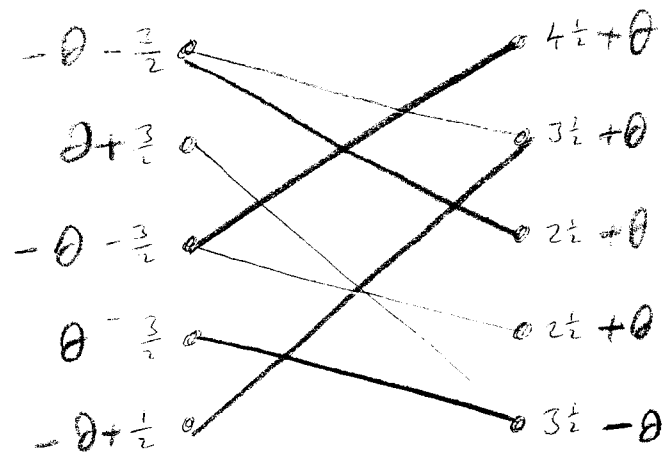


Augmenting path
 v_5, u_2, v_1, u_2



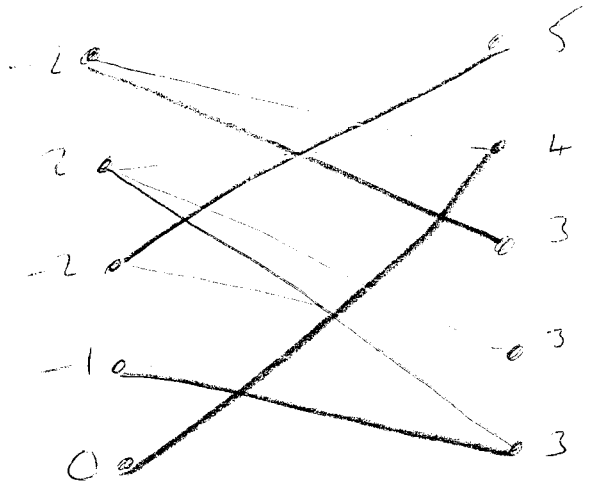
Try to get admissible edge out of v_2 .

Largest possible θ is $\frac{1}{2}$; constrained by edge (v_2, u_5)



No augmenting path, although there is an alternating path to v_4 . So if we can get admissible edge from either v_2 or v_4 , will be helpful.

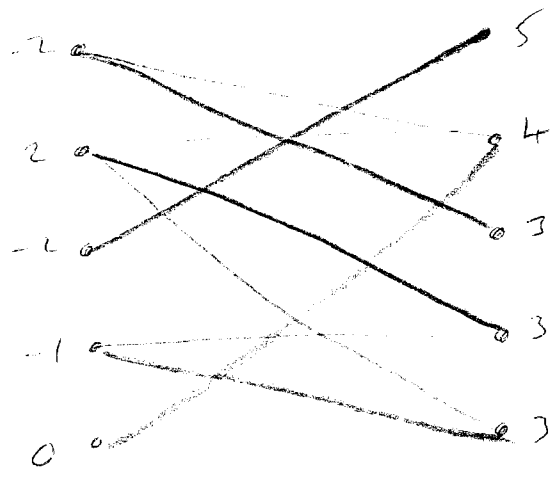
Largest possible θ is $\frac{1}{2}$; constrained by edges $(v_2, u_1), (v_2, u_4), (v_4, u_4)$



Now have augmenting paths
for v_2 :

$$v_2, u_4 \leftarrow$$

$$v_2, u_5, v_4, u_4.$$



Primal cost is

$$1 + 5 + 3 + 2 + 4 = 15$$

Dual cost is

$$-2 + 2 - 2 - 1 + 0 + 5 + 4 + 3 + 3 + 3 = 15 \checkmark$$

Non bipartite case:

$$x_e = \begin{cases} 1 & \text{if edge } e \in M \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ij} = x_{ji}, \quad x_{ii} = 0$$

~~is~~

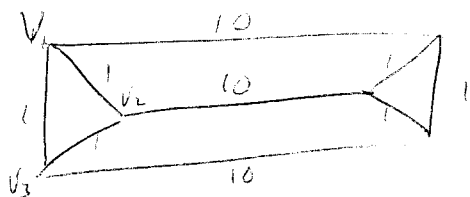
Perfect matching problem is:

$$\begin{aligned} \min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} = 1 \quad \forall i \in V \quad (\text{IP}) \\ & x_{ij} \text{ binary.} \end{aligned}$$

Not equivalent to

$$\begin{aligned} \min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} = 1 \quad \forall i \in V \\ & x_{ij} \geq 0 \end{aligned}$$

E.g.:



Optimal p.m. has weight 12

Optimal soln to LP relaxation has value 3.

Problem is overconstrained odd sets.

Consider $S = \{v_1, v_2, v_3\}$

A Matching can use at most one edge from the edges in this odd set

However, the LP soln gives total weight $\frac{3}{2}$ to edges in $E(S)$.

So introduce odd set / blossom constraints:

$$\sum_{e \in E(S)} x_e \leq \frac{|S|-1}{2} \quad \forall \text{ odd sets } S, |S| \geq 3.$$

Theorem ~~Any~~ optimal soln to (IP) is an optimal soln to

$$\min \sum c_{ij} x_{ij}$$

$$\text{s.t. } \sum_j x_{ij} = 1 \quad \forall i \in V$$

$$\sum_{\substack{ij \in E \\ i < j}} x_{ij} \leq \frac{|S|-1}{2} \quad \forall \text{ odd sets } S \subseteq V, |S| \geq 3$$

$$x_{ij} \geq 0.$$

//

Now there are $2^{m-1} - m$ odd sets - exponential number.

But by exploiting blossoms, Edmonds gave an algorithm which requires time polynomial in m .