LP-Related Algorithms

Hungarian Algorithm
Metric Facility Location

Part I:
Hungarian Algorithm

Problem

[Maximum Weight Perfect Matching (MWPM)]

- Given a weighted complete bipartite graph $G = (A, B, E = A \times B, w: E \to \mathbb{R}_{\geq 0})$, find a maximum weight perfect matching.

- Perfect Matching: all the vertices must be matched!
- Need to assume: $|A| = |B| = n$.

Hungarian Algorithm – High-Level

- Assign a "potential" to each vertex $p: (A \cup B) \to \mathbb{R}_{\geq 0}$
- Throughout the algorithm, maintain:

1. Dominance: $\forall u, v: p(u) + p(v) \geq w(u, v)$
2. Tightness: for any (u, v) selected in the matching M, $p(u) + p(v) = w(u, v)$

Hungarian Algorithm – “at the End of the Day”

- Suppose we have found
  - a matching M with size $|M| = n$, and
  - A potential assignment p such that dominance and tightness hold,
- then we are done!
- **Lemma** (Kuhn & Munkres). If we have a matching M with size $|M| = n$ and a potential assignment p such that dominance and tightness hold, then M is a MWPM.

Proof. For any perfect matching M',

$$w(M) = \sum_{(u,v)\in M} w(u,v) = \sum_{u\in A\cup B} p(u) \geq \sum_{(u,v)\in M'} w(u,v) = w(M')$$

tightness dominance

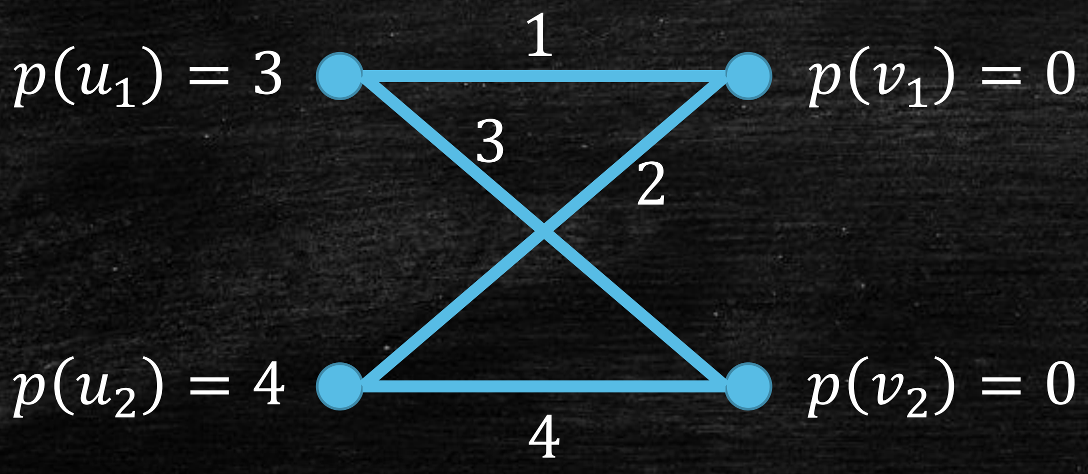
An "Academic Interpretation"

- Each edge (u, v): a research project with cost/value $w(u, v)$
- A: set of female researchers
- B: set of male researchers
- Every female-male pair of researchers can jointly work on a project.
  - Each project only requires a female and a male
  - Each researcher can only work on one project
- $p(u)$: research funding allocated to researcher u
- Dominance: sufficient funding so that researchers can freely paired and work on the project they prefer
- Tightness: just adequate funding so that n most valuable projects can be done
- Objective: properly allocate (minimum) funding to researchers so that their optimal choice is to work out n most valuable projects

Initialization

Initialize:

- $M = \emptyset$
- $\forall u \in A: p(u) = \max_{v \in B} w(u, v)$
- $\forall v \in B: p(v) = 0$



Two Types of Updates

- Update 1: increase $|M|$ with only tight edges.
- Update 2: adjust funding to make more tight edges.

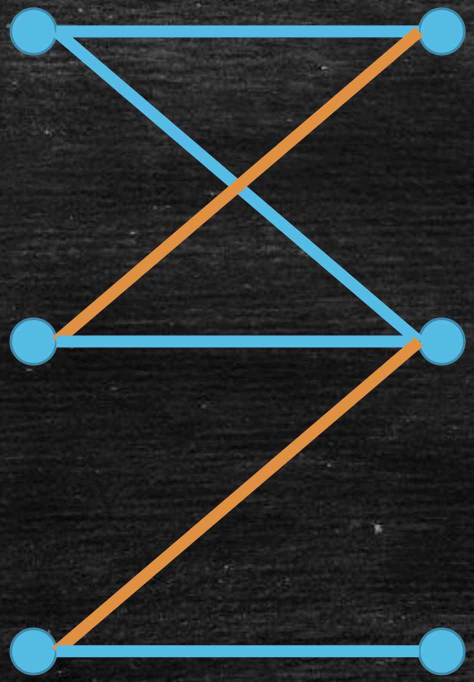
- Hungarian Algorithm:
  - Do 1 while possible.
  - If 1 is impossible, do 2.

- Very Important: throughout the algorithm, dominance and tightness always hold!
  - In particular, M always only contain tight edges.

Augmenting Path

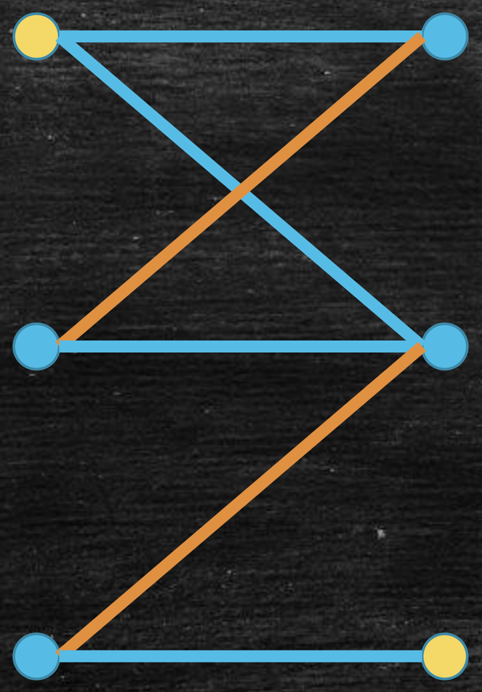
- We will only work on subgraph $G^t = (A^t \cup B^t, E^t)$ with tight edges!
- Alternative path: path with edges alternates between $E^t \setminus M$ and M
- Free vertex: vertex not in M
- Augmenting path: an alternating path with two free vertices as the two endpoints.

Example



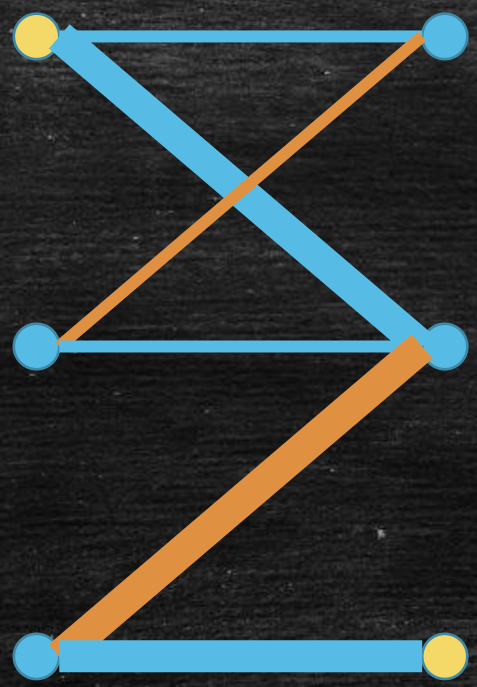
- Orange Edges: Current M

Example



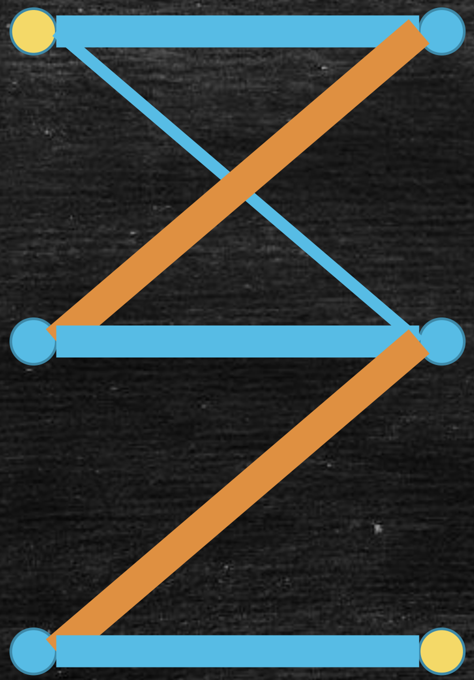
- Yellow Vertices: free vertices

Example



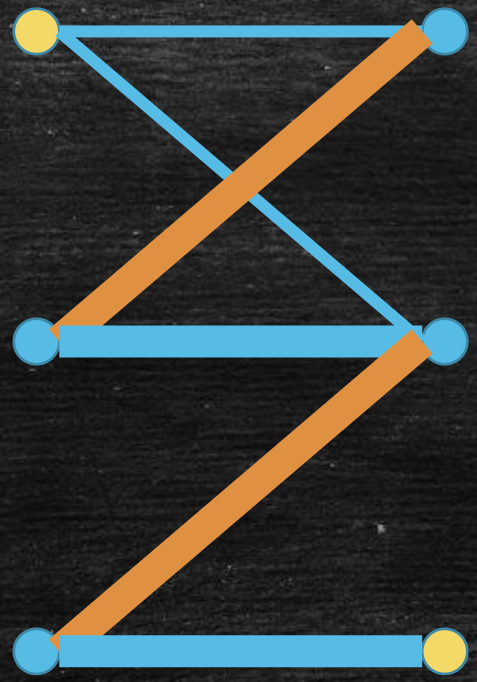
- an augmenting path…

Example



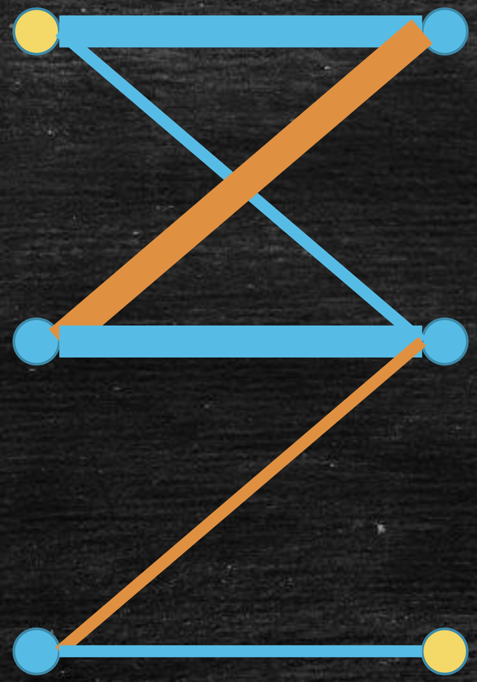
- an augmenting path…

Example



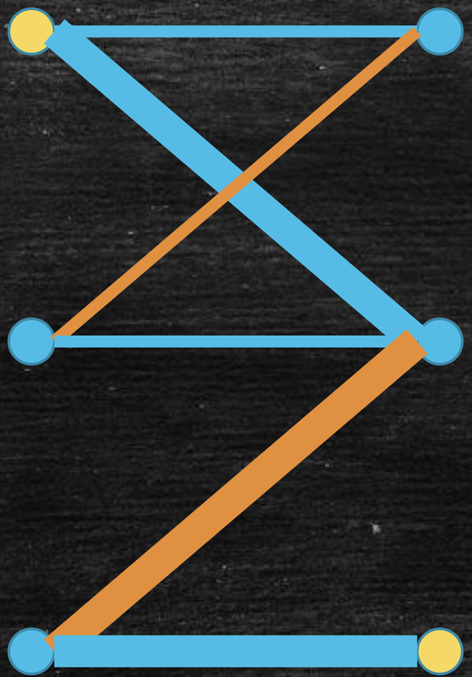
- not an augmenting path…

Example



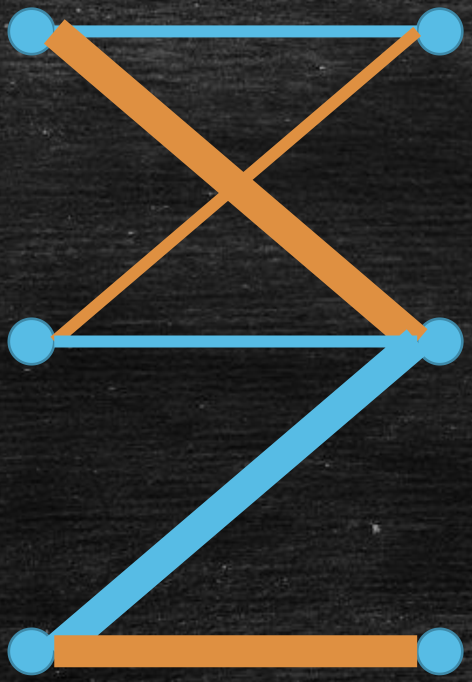
- not an augmenting path…

Increase $|M|$ on an Augmenting Path



- If we have an augmenting path, we can increase $|M|$ by "swopping".

Increase $|M|$ on an Augmenting Path

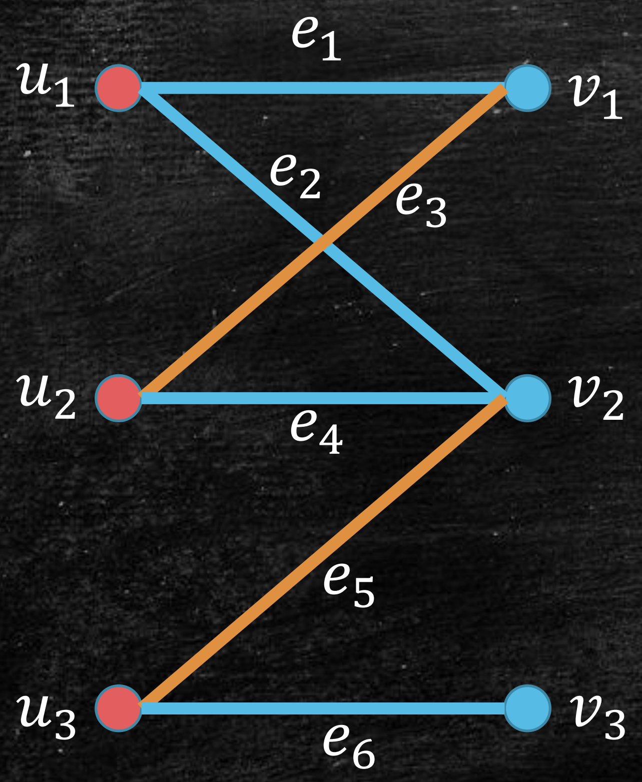


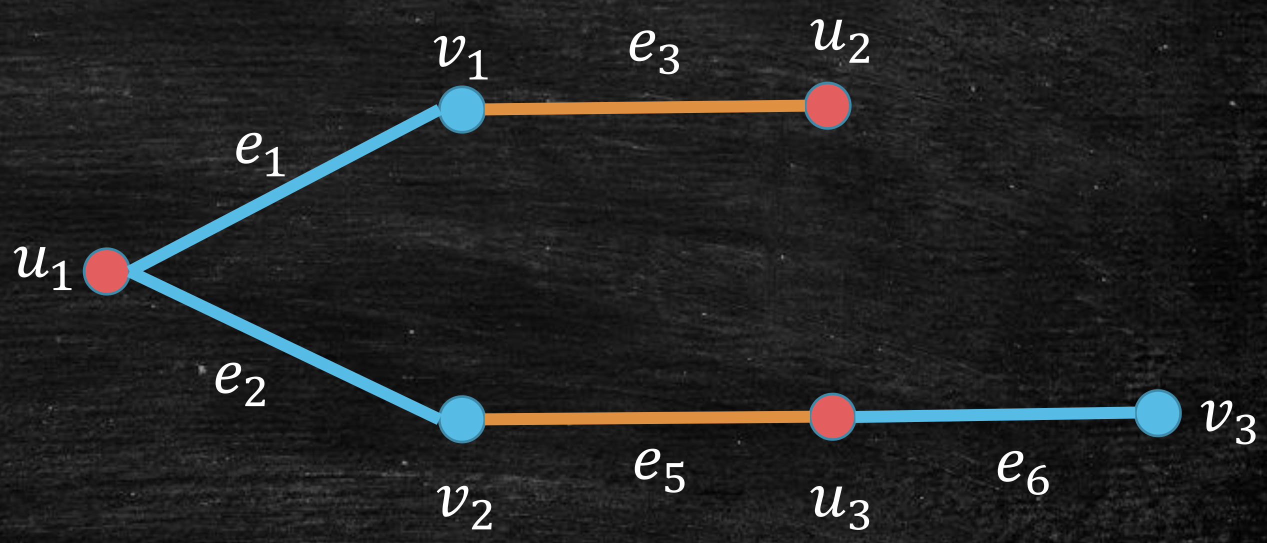
- If we have an augmenting path, we can increase $|M|$ by "swopping".

Reachable Set S and Search Graph

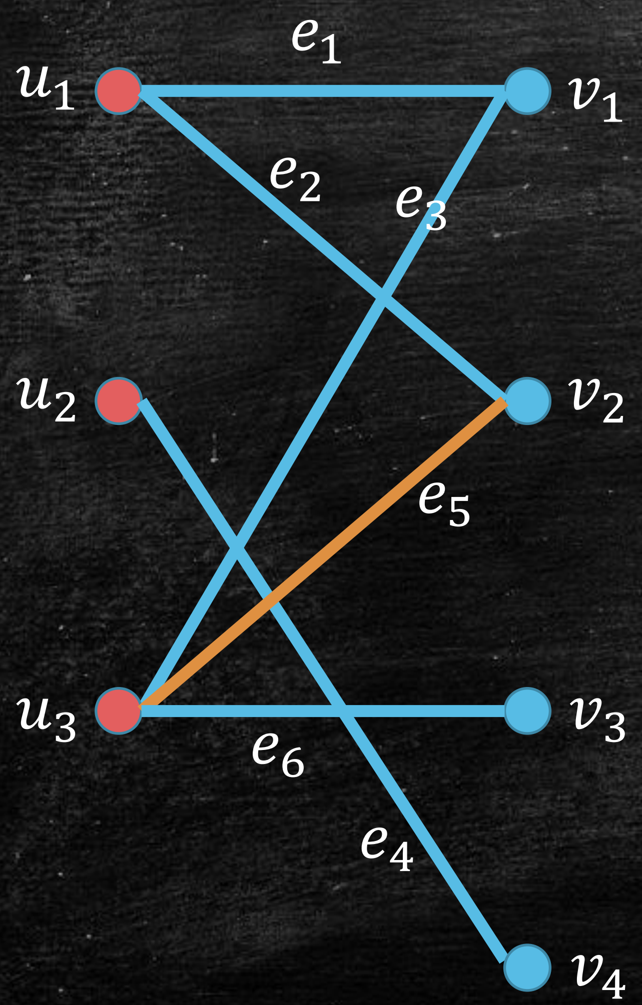
- Initialize $S = A' \cup B'$ with $A' = B' = \emptyset$
- Start by including all free vertices of A to A'.
- If $u \in A'$, add all v with $(u, v) \in E^t$ to B'.
- If $v \in B'$, add u to A' where $(u, v) \in M$.
- This is like a “alternative version” of BFS (or DFS, which also works).

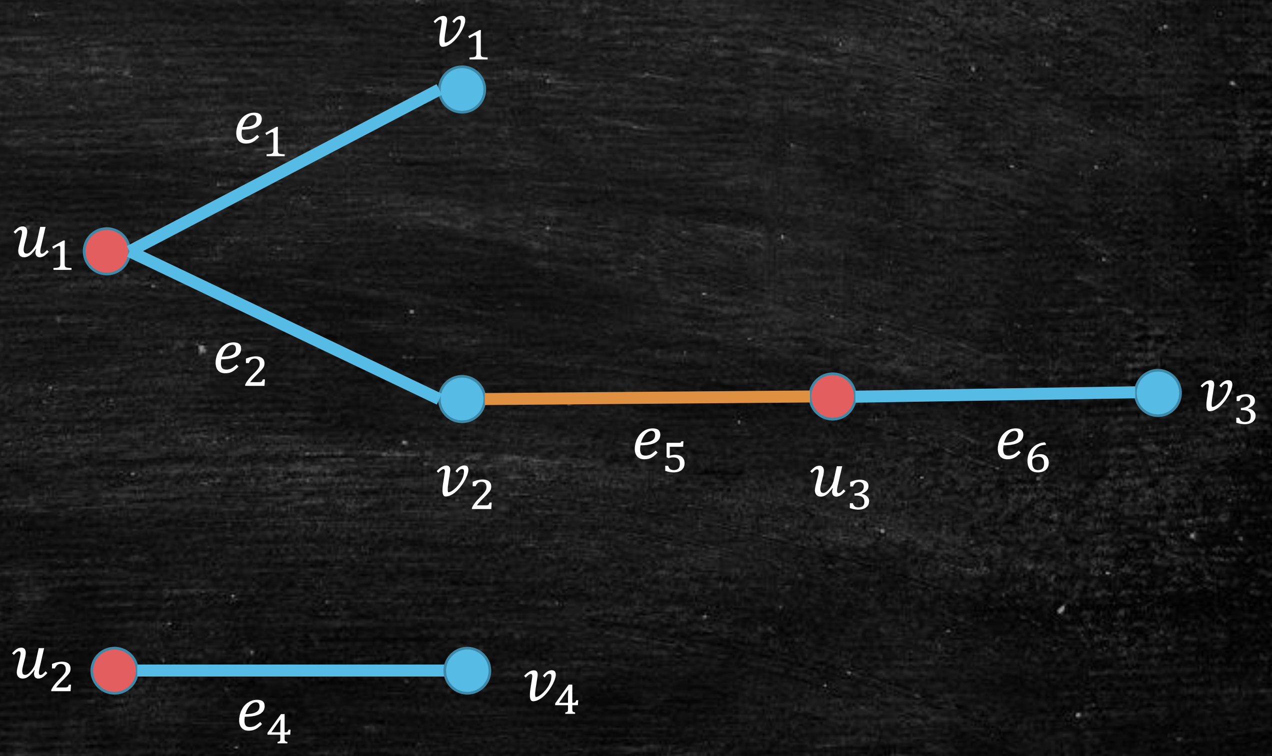
Search Graph – Example 1





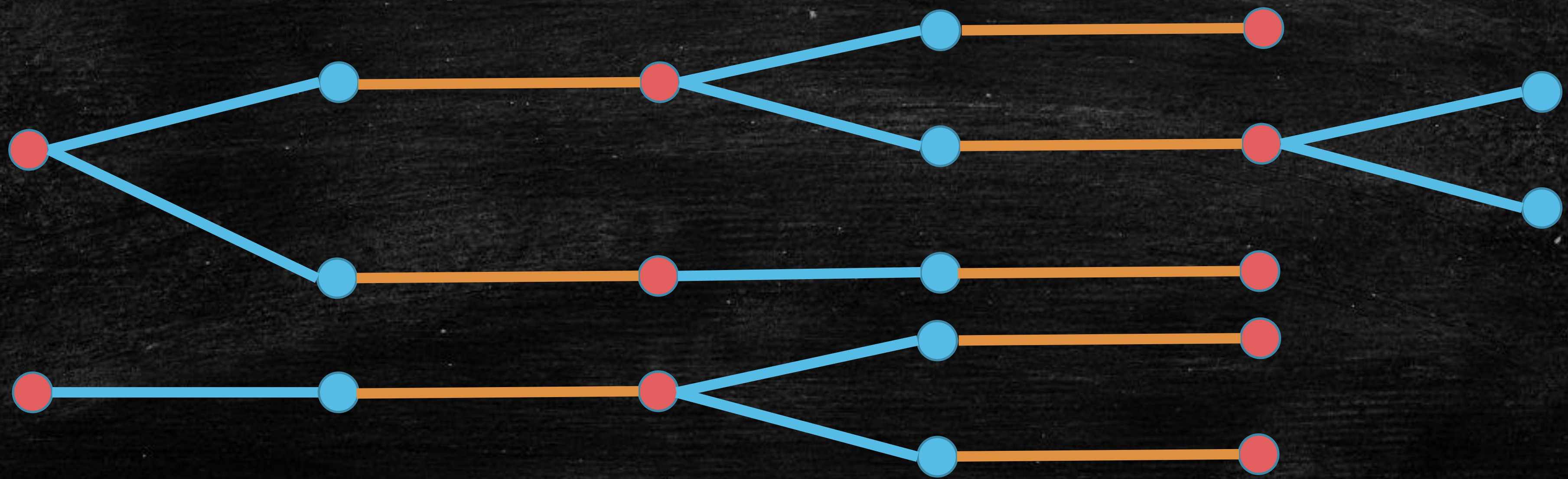
Search Graph – Example 2





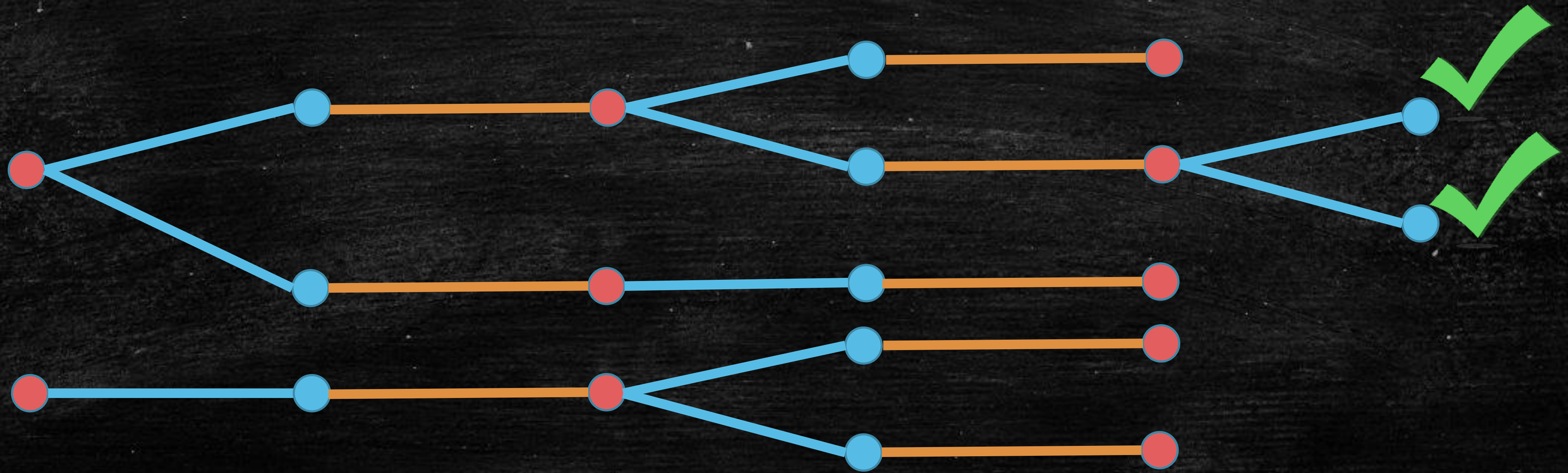
Search Graph

- A forest span on reachable set $S = A' \cup B'$
- All roots are free vertices in A.
- Edges on each path alternates between $E^t \setminus M$ and M.
- All middle vertices are not free.
- Vertices on each path alternates between females and males.



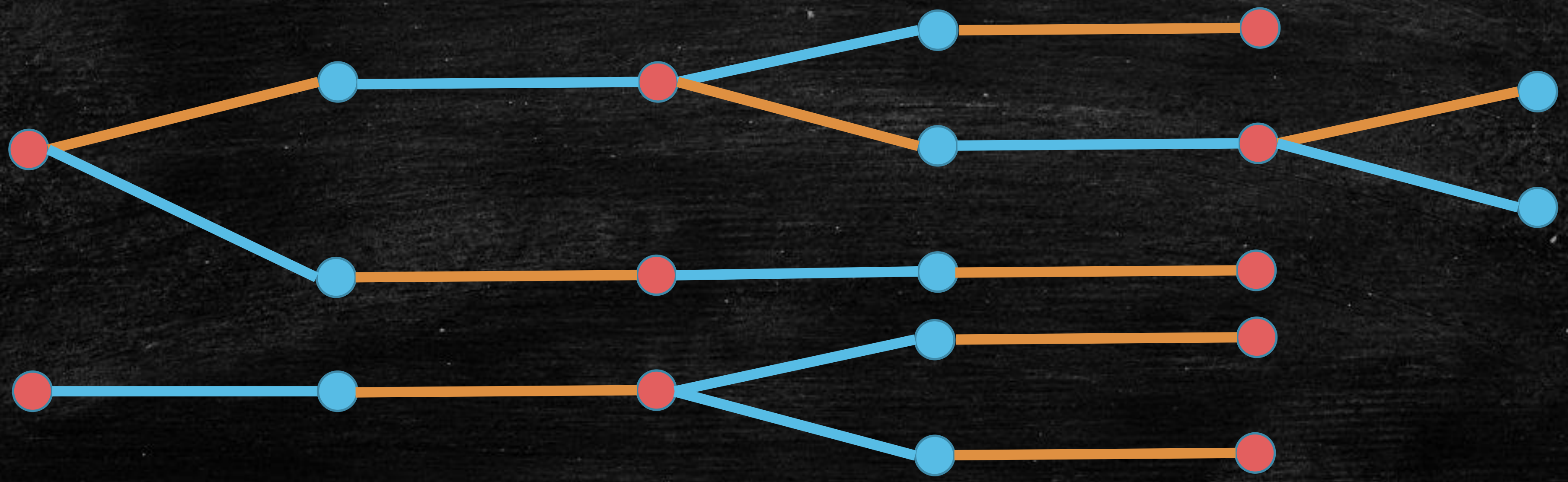
Update 1: increase M using tight edges.

- If a path on the search graph ends at a free vertex, we have an augmenting path.



Update 1: increase $|M|$ using tight edges.

- If a path on the search graph ends at a free vertex, we have an augmenting path.
- We can do “swopping”, which increases $|M|$

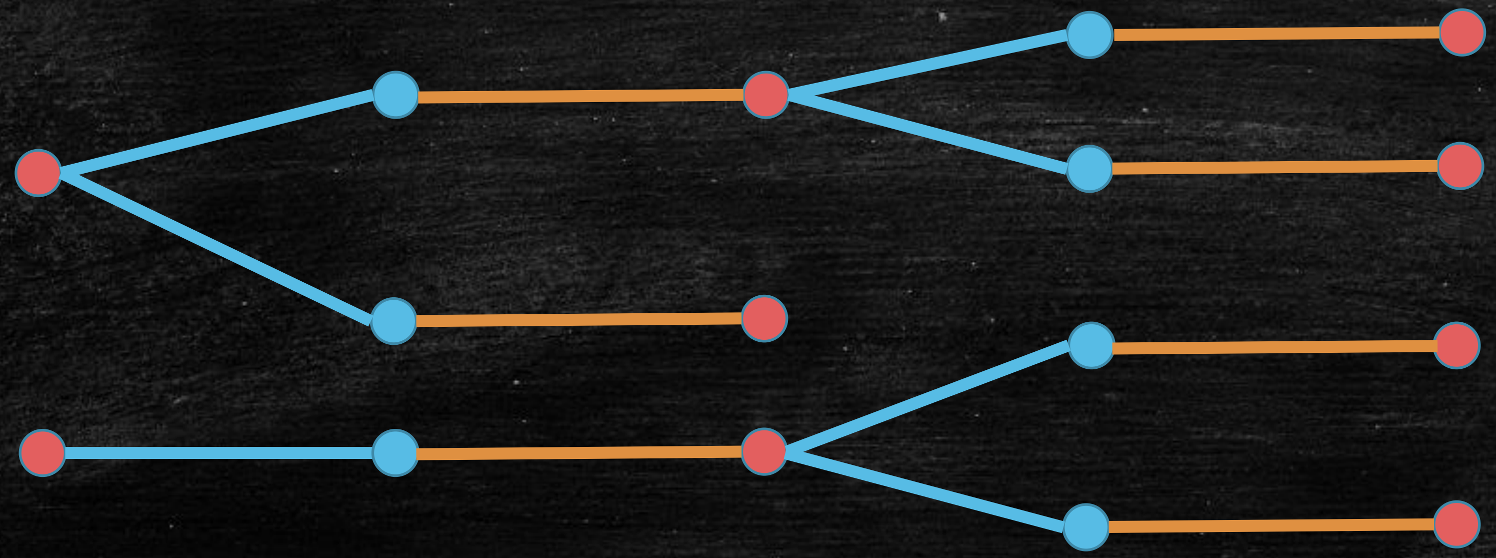


Update 1: increase $|M|$ using tight edges.

- If a path on the search graph ends at a free vertex, we have an augmenting path.
- We can do "swopping", which increases $|M|$.
- Then, start over for another "Update 1".

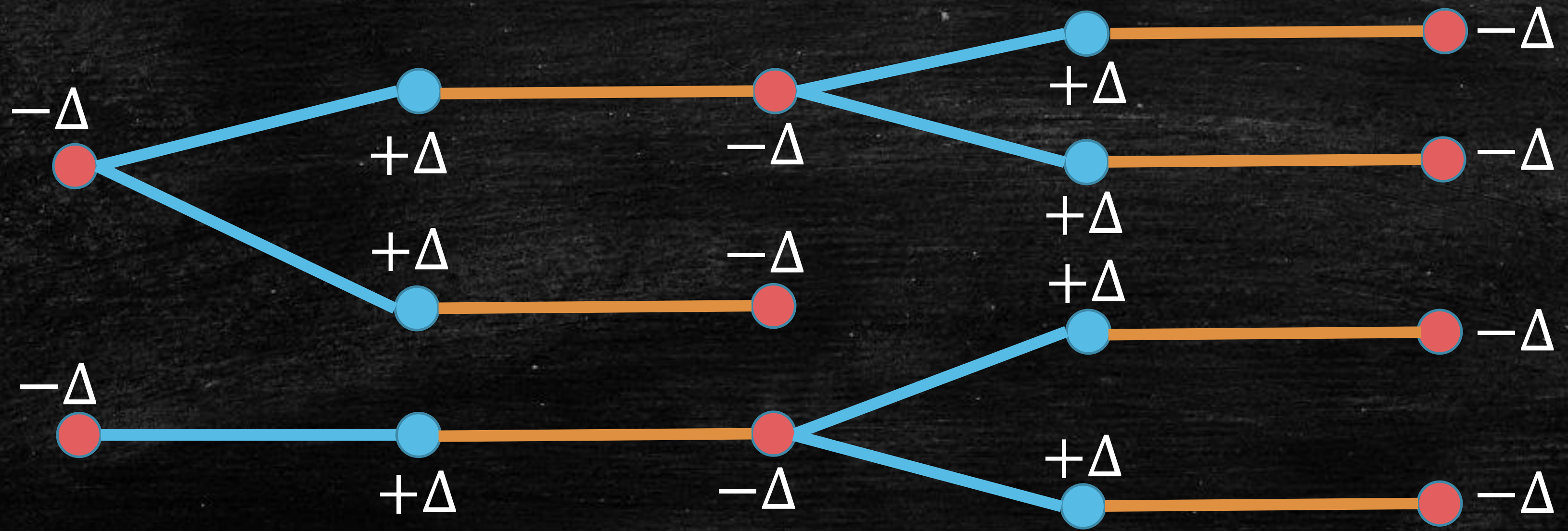
Update 2: adjust "funding" p

- When all endpoints are not free, they should be "females".



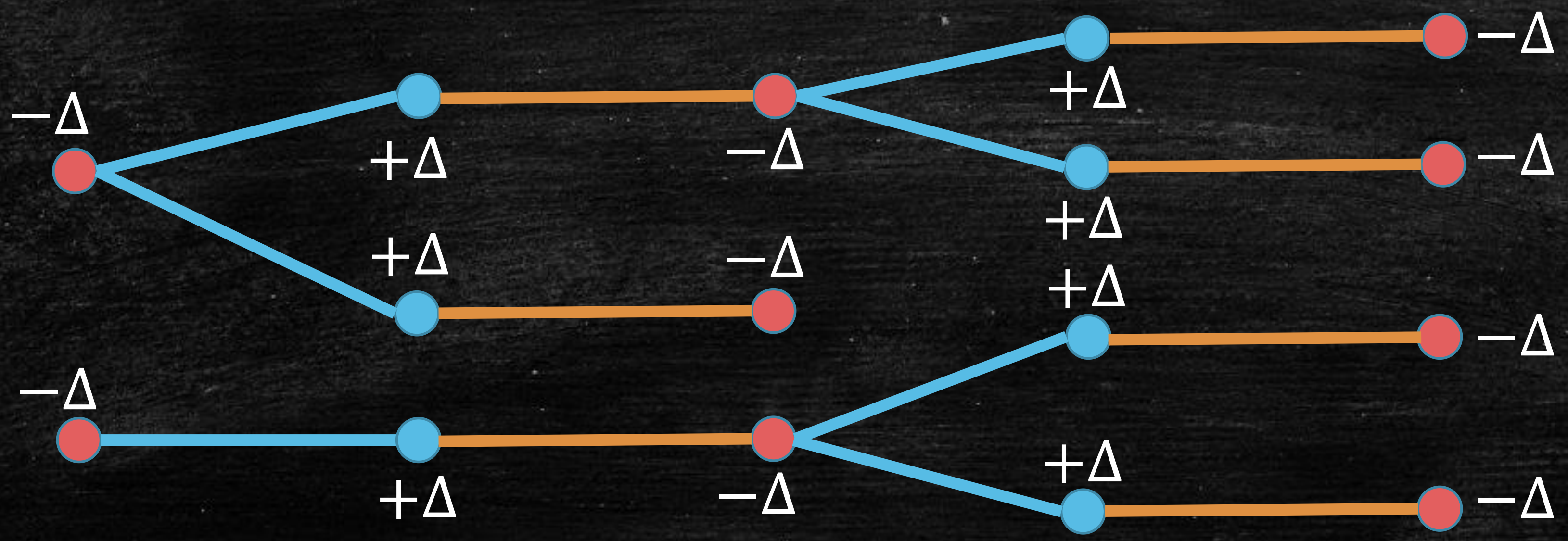
Update 2: adjust "funding" p

- When all endpoints are not free, they should be "females".
- Choose a "suitable" $\Delta > 0$ and adjust the "funding" as shown.



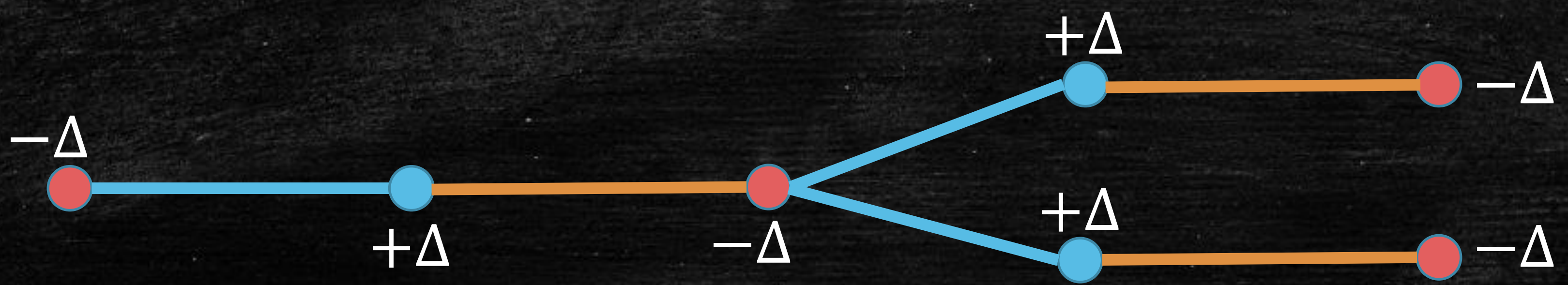
Update 2: adjust "funding" p

- When all endpoints are not free, they should be "females".
- Choose a "suitable" $\Delta > 0$ and adjust the "funding" as shown.
- The tight edges remains tight.



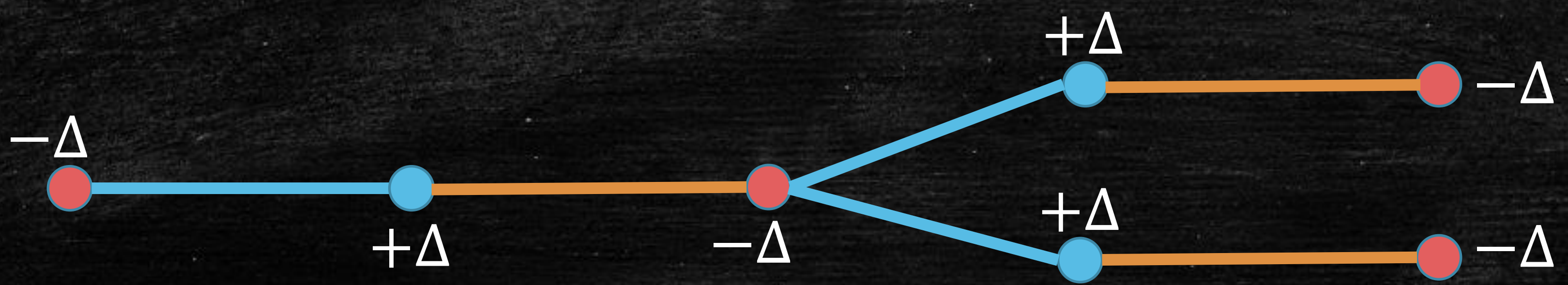
Update 2: adjust "funding" p

- When all endpoints are not free, they should be "females".
- Choose a "suitable" $\Delta > 0$ and adjust the "funding" as shown.
- The tight edges remains tight.
- Three types of "loose" edges (u, v):
  - 1) $u \in A \setminus A^t, v \in B \setminus B^t$ 2) $u \in A \setminus A^t, v \in B^t$ 3) $u \in A^t, v \in B \setminus B^t$



Update 2: adjust “funding” p

- When all endpoints are not free, they should be “females”.
- Choose a “suitable” $\Delta > 0$ and adjust the “funding” as shown.
- The tight edges remains tight.
- Three types of “loose” edges (u, v):
  - 1) $u \in A \setminus A^t, v \in B \setminus B^t$ 2) $u \in A \setminus A^t, v \in B^t$ 3) $u \in A^t, v \in B \setminus B^t$
- “Dominance” clearly continue to holds for type 1) and 3)



Update 2: adjust "funding" p

- When all endpoints are not free, they should be "females".
- Choose a "suitable" $\Delta > 0$ and adjust the "funding" as shown.
- The tight edges remains tight.
- Three types of "loose" edges (u, v):
  - 1) $u \in A \setminus A^t, v \in B \setminus B^t$ 2) $u \in A \setminus A^t, v \in B^t$ 3) $u \in A^t, v \in B \setminus B^t$
- "Dominance" clearly continue to holds for type 1) and 3)
- For 2), we choose Δ just enough to "tighten" a loose edge while guaranteeing dominance.

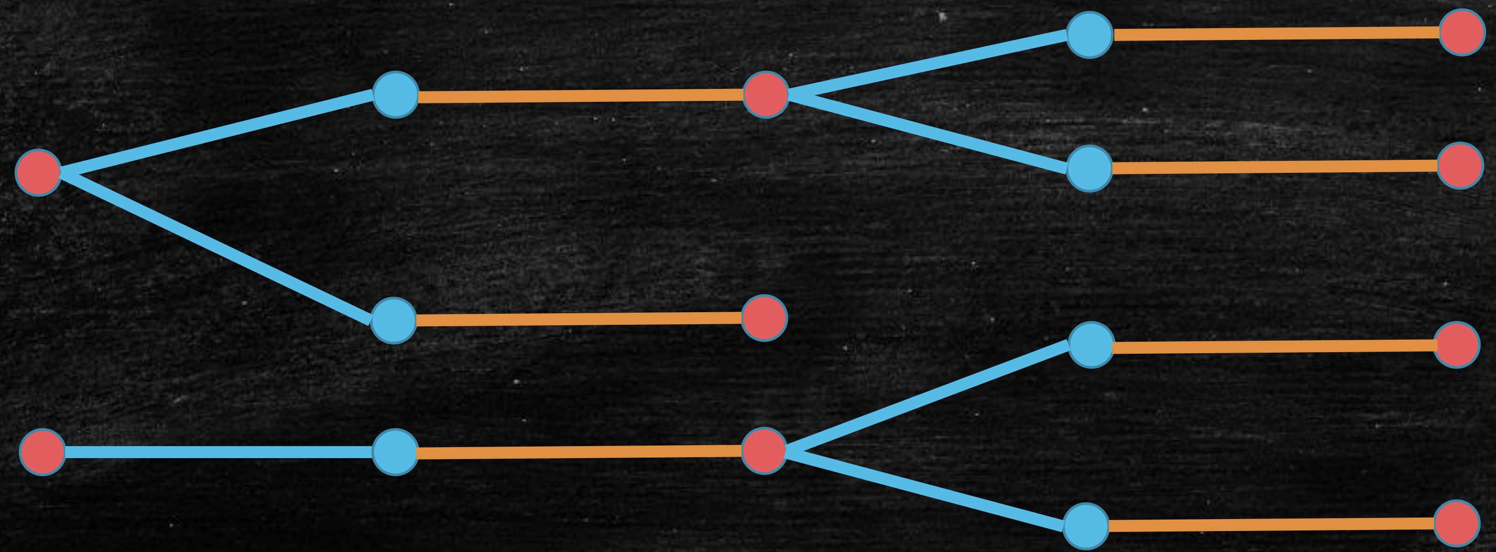
Update 2: adjust "funding" p

- When all endpoints are not free, they should be "females".
- Choose a "suitable" $\Delta > 0$ and adjust the "funding" as shown.
- The tight edges remains tight.
- Three types of "loose" edges (u, v):
  - 1) $u \in A \setminus A^t, v \in B \setminus B^t$ 2) $u \in A \setminus A^t, v \in B^t$ 3) $u \in A^t, v \in B \setminus B^t$
- "Dominance" clearly continue to holds for type 1) and 3)
- For 2), we choose Δ just enough to "tighten" a loose edge while guaranteeing dominance.

$$\Delta = \min_{u \in A'} \text{slack}[u] \qquad \text{where} \quad \text{slack}[u] = \min_{v \in B \setminus B'} (p(u) + p(v) - w(u, v))$$

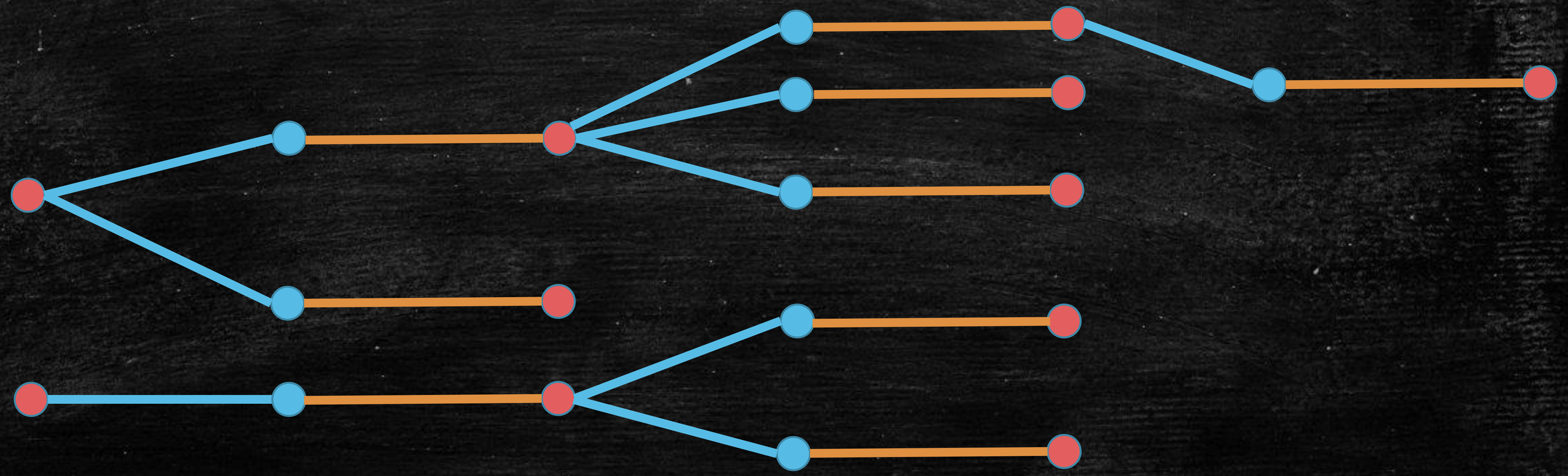
Update 2: adjust "funding" p

- After Δ-adjustment, continue to explore the search graph.



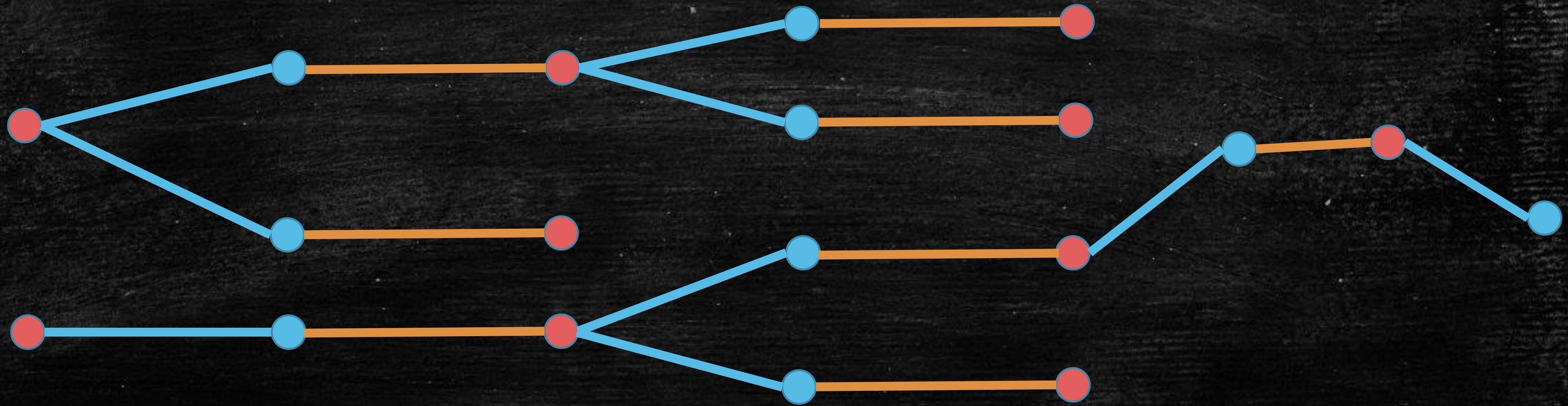
Update 2: adjust "funding" p

- After Δ-adjustment, continue to explore the search graph.
- If still no augmenting path, do another "Update 2"



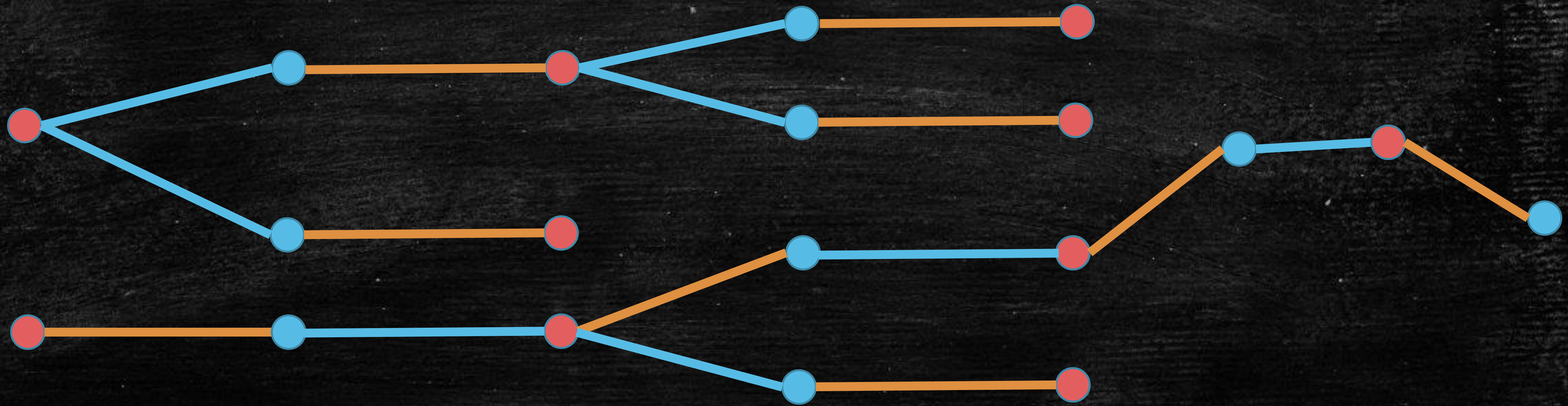
Update 2: adjust "funding" p

- After Δ-adjustment, continue to explore the search graph.
- If still no augmenting path, do another "Update 2"
- Otherwise, do "Update 1"



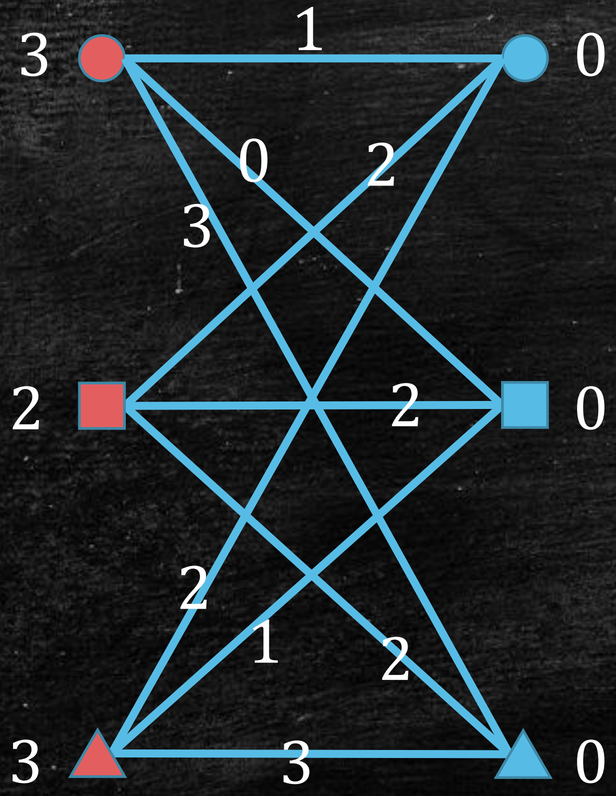
Update 2: adjust "funding" p

- After Δ-adjustment, continue to explore the search graph.
- If still no augmenting path, do another "Update 2"
- Otherwise, do "Update 1"



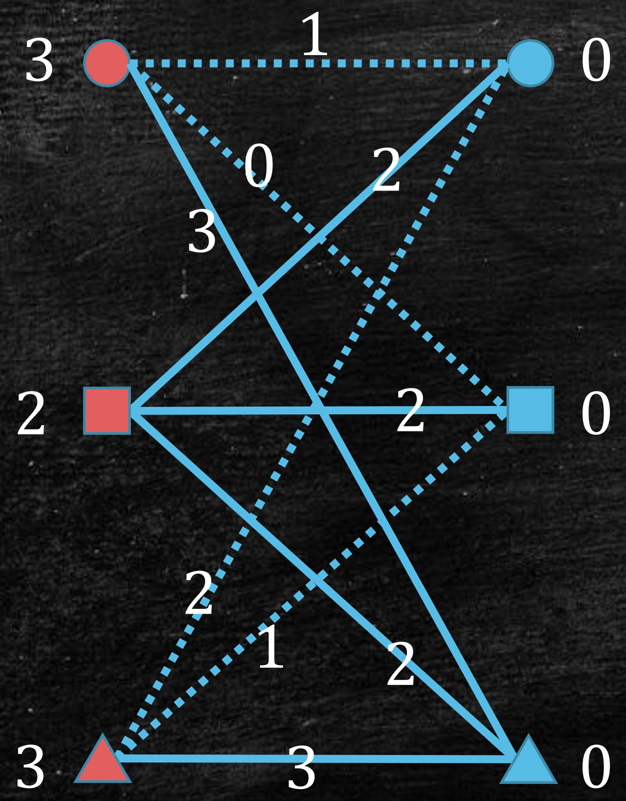
Hungarian Algorithm – Example

- Initialization



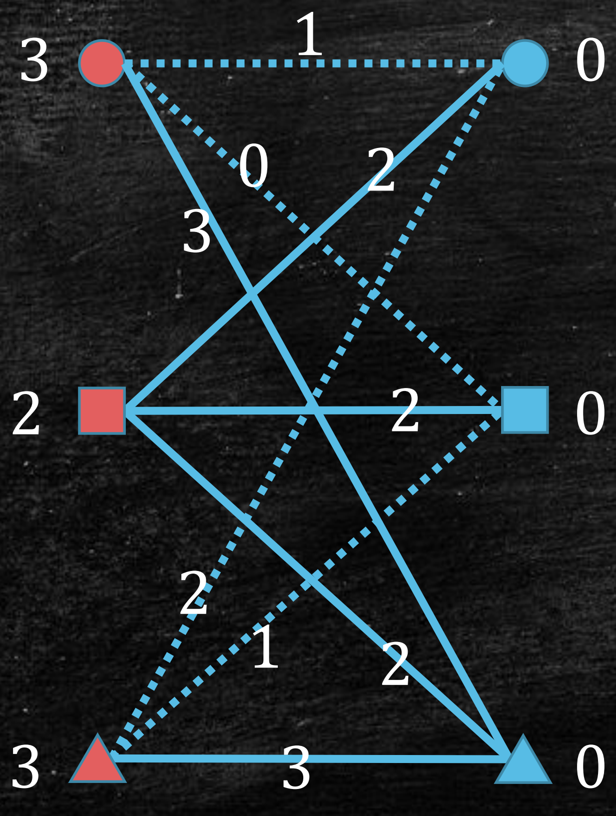
Hungarian Algorithm – Example

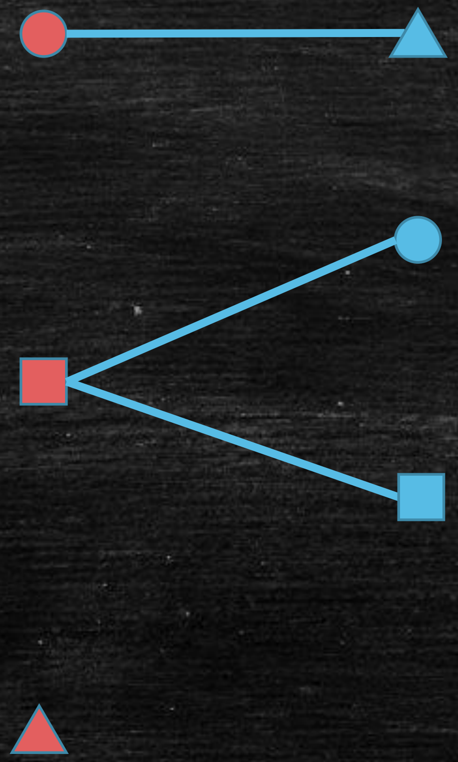
- Solid Edges are tight.



Hungarian Algorithm – Example

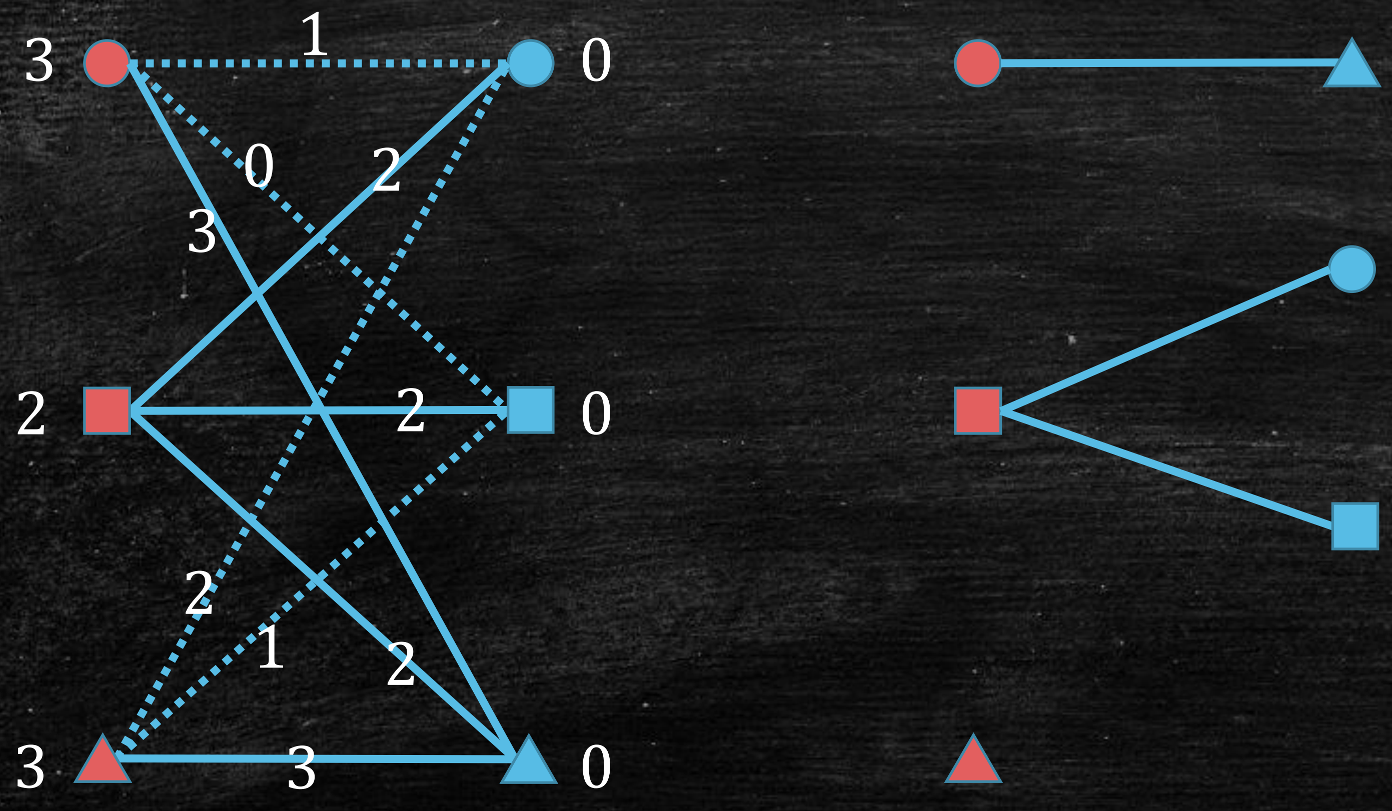
- Construct the search graph





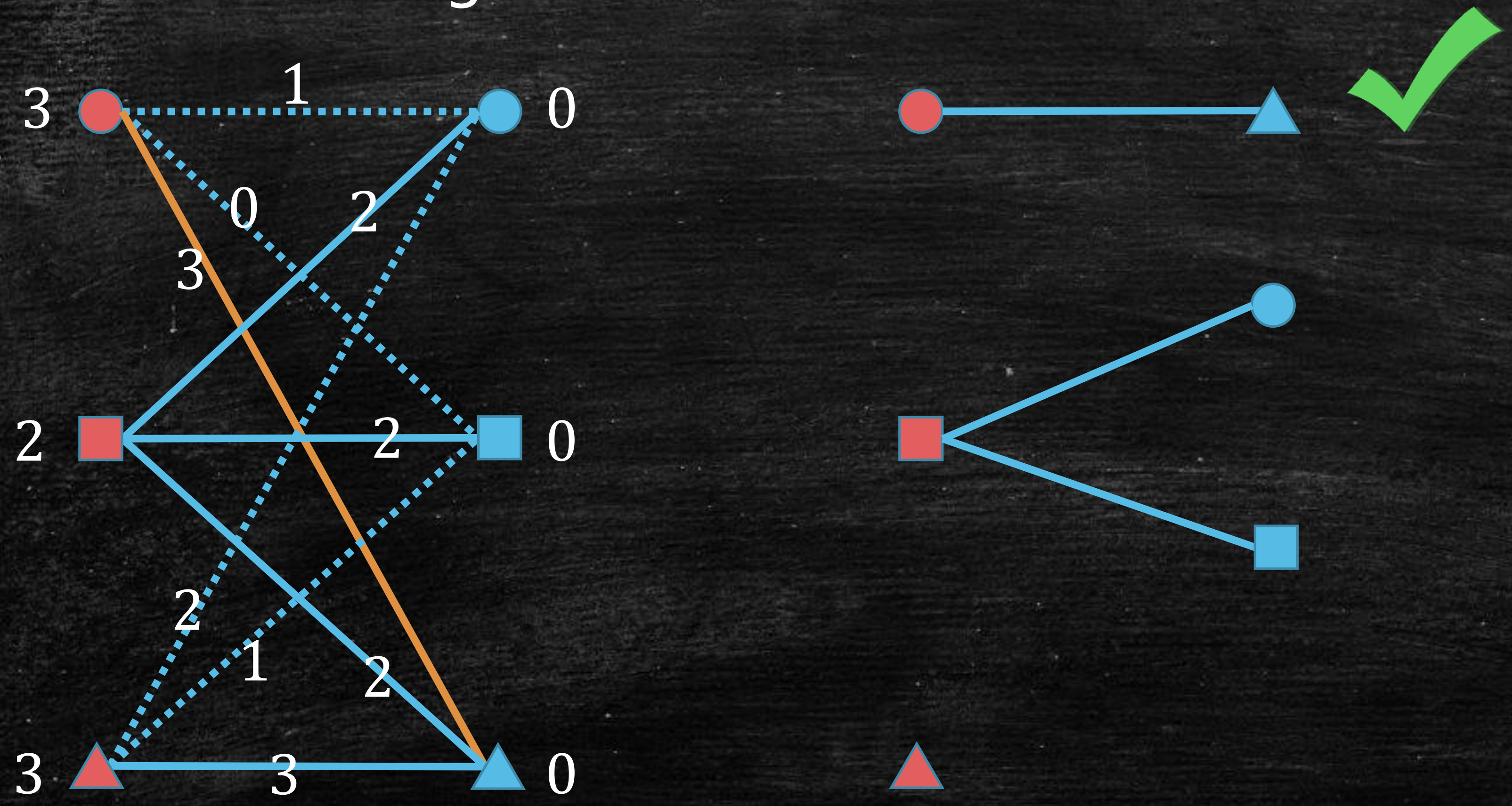
Hungarian Algorithm – Example

- Three augmenting paths, choose an arbitrary one (say, 1st)



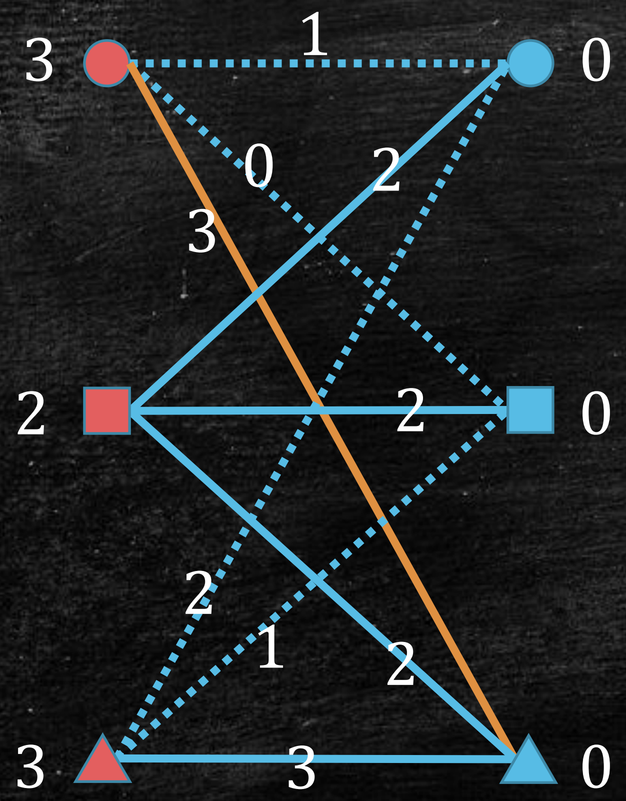
Hungarian Algorithm – Example

- Add the edge to M



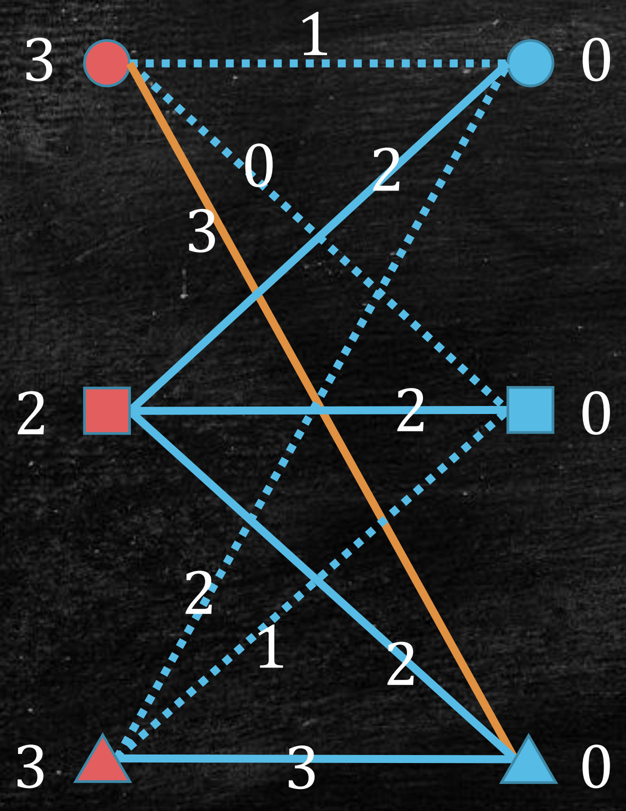
Hungarian Algorithm – Example

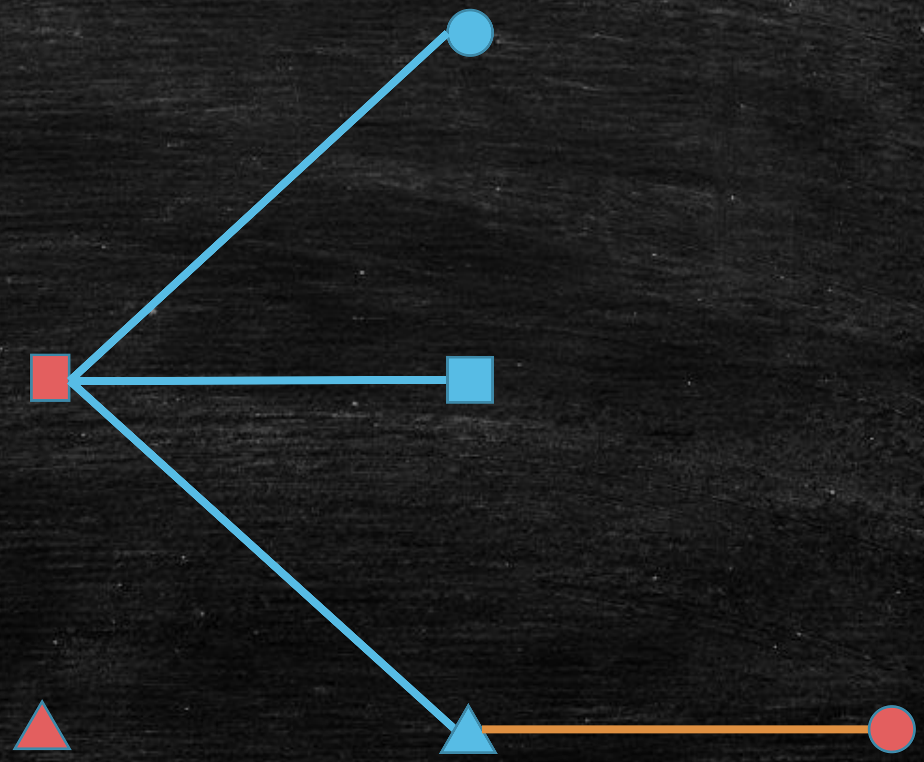
- Start over…



Hungarian Algorithm – Example

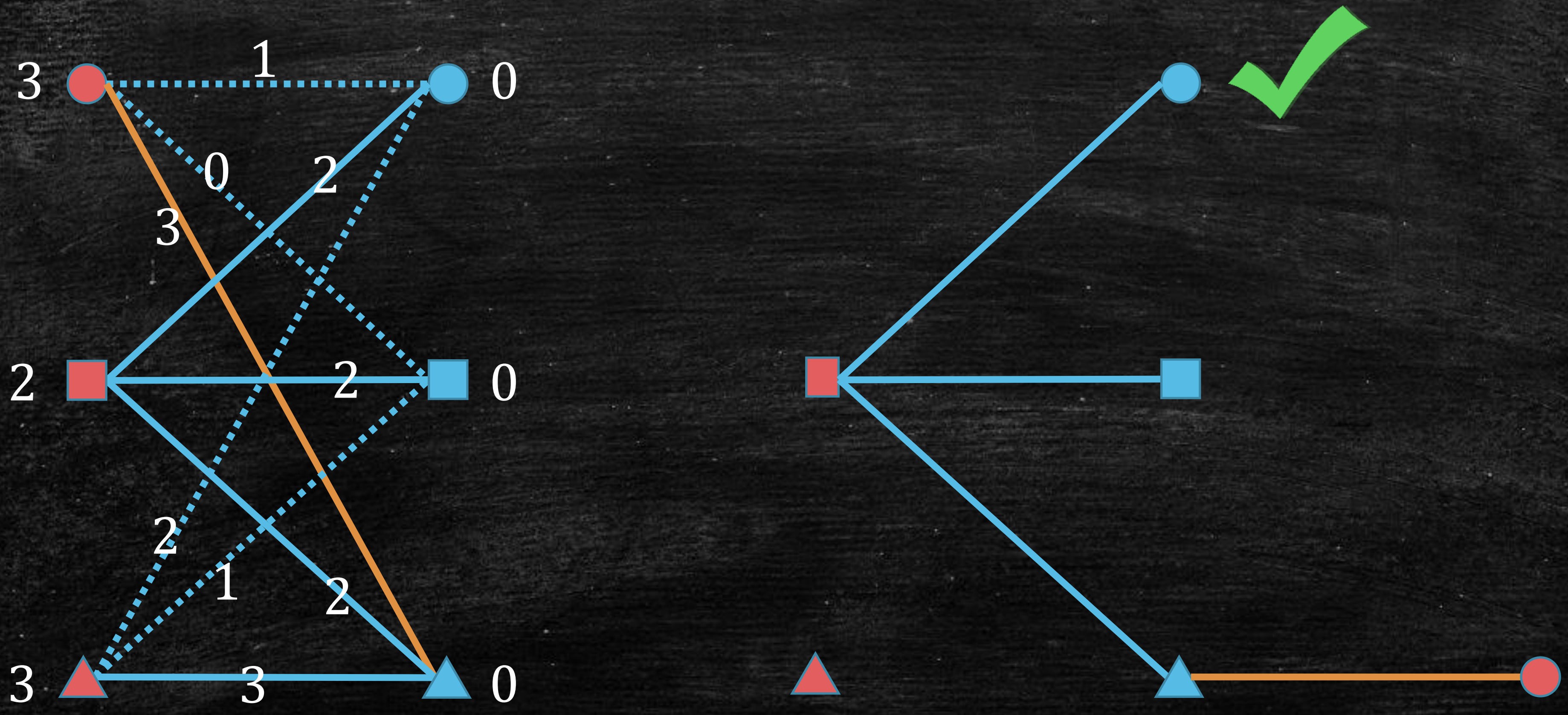
- and construct the search graph again.





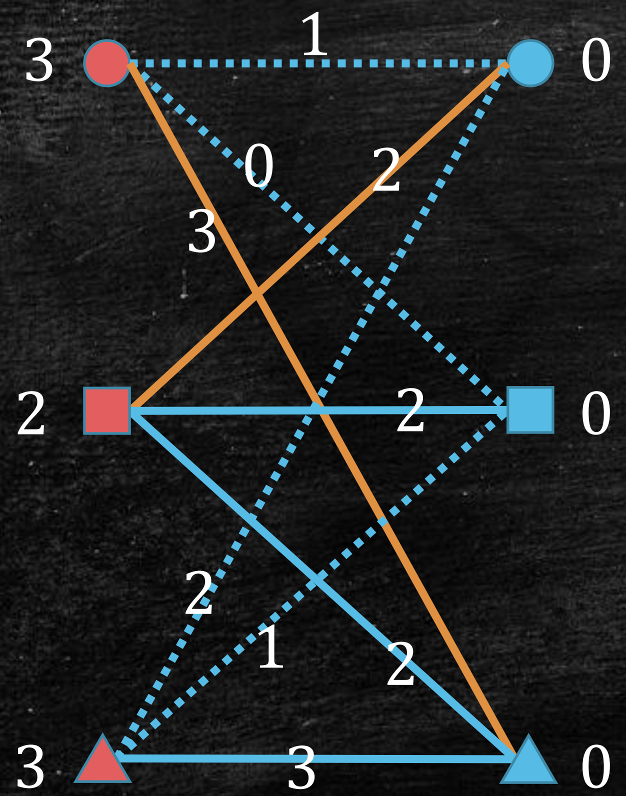
Hungarian Algorithm – Example

- Two augmenting paths, choose an arbitrary one.



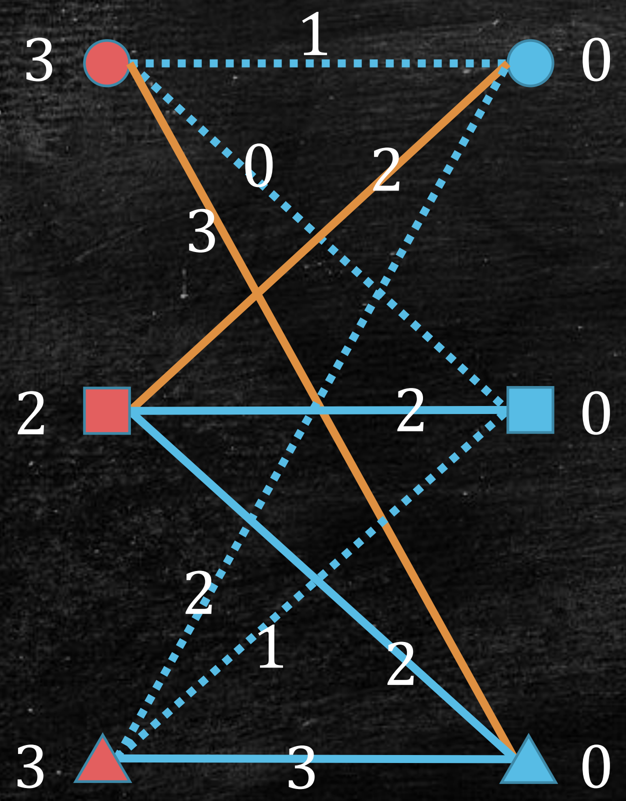
Hungarian Algorithm – Example

- Update M and start over…



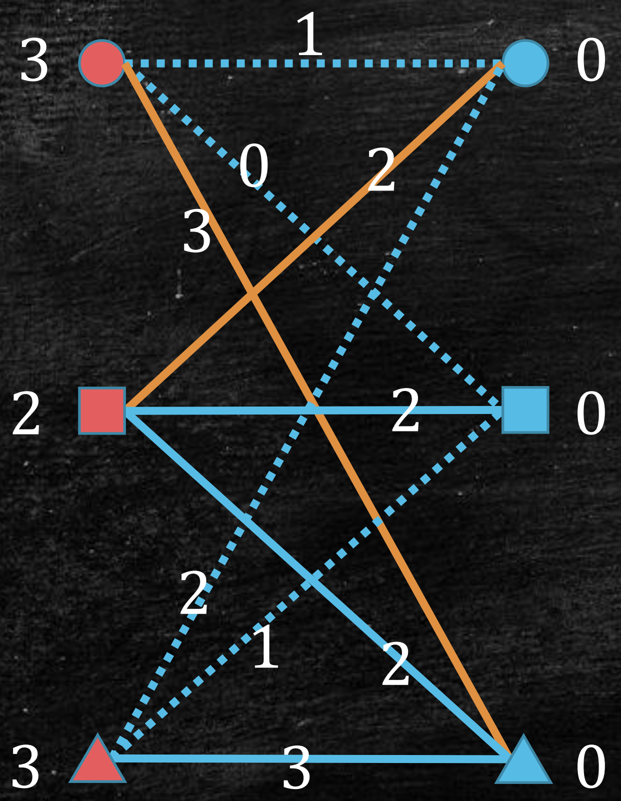
Hungarian Algorithm – Example

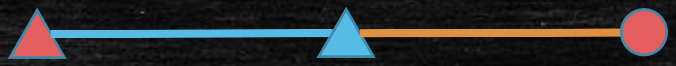
- Construct the search graph



Hungarian Algorithm – Example

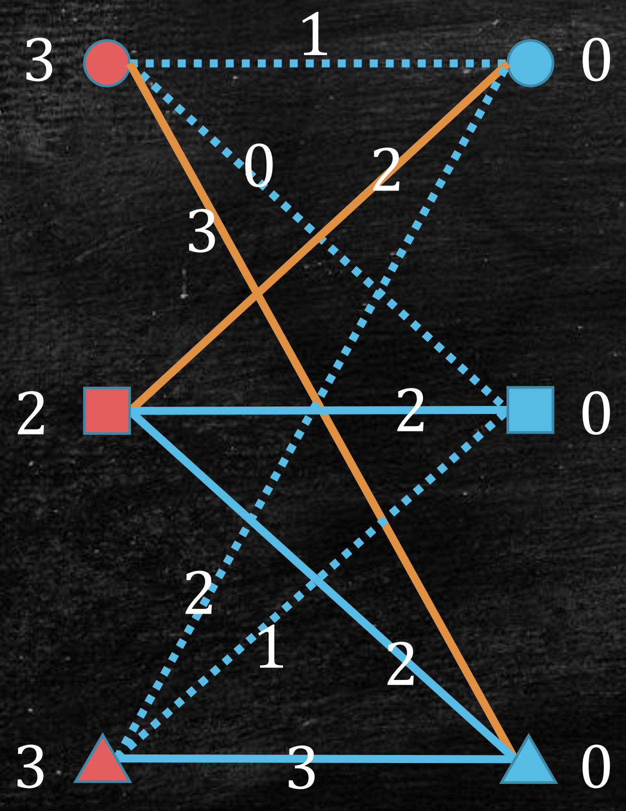
- No more augmenting path





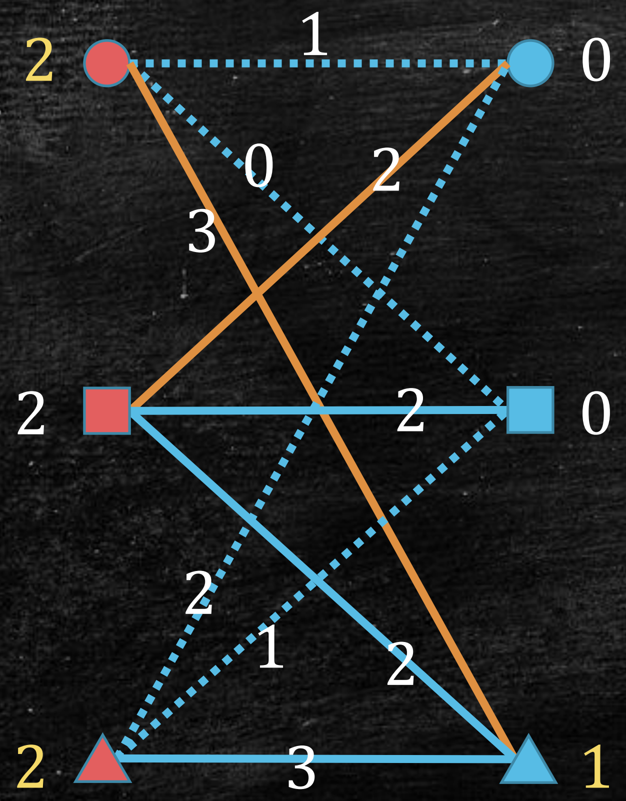
Hungarian Algorithm – Example

- Circle has slackness 2 and Triangle has slackness 1, so we choose $\Delta = 1$



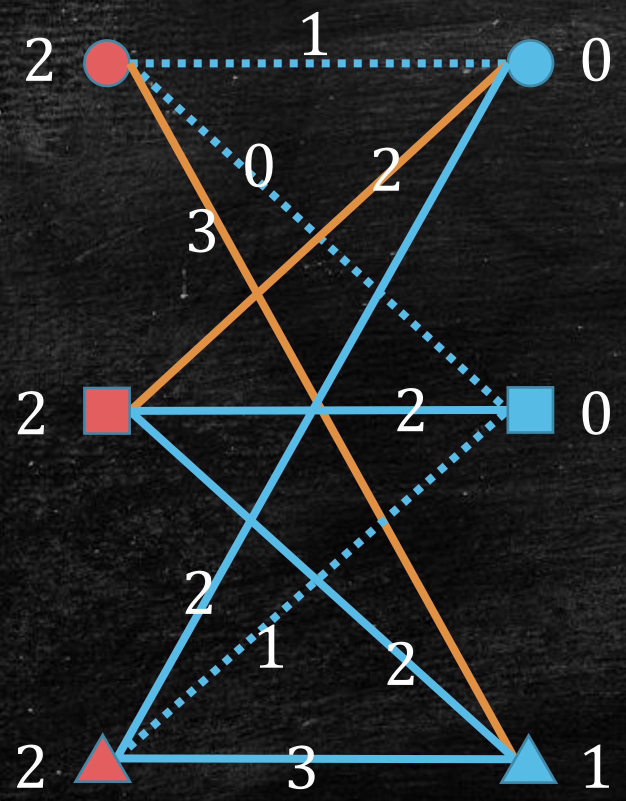
Hungarian Algorithm – Example

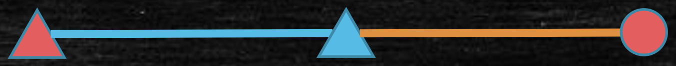
- Update p



Hungarian Algorithm – Example

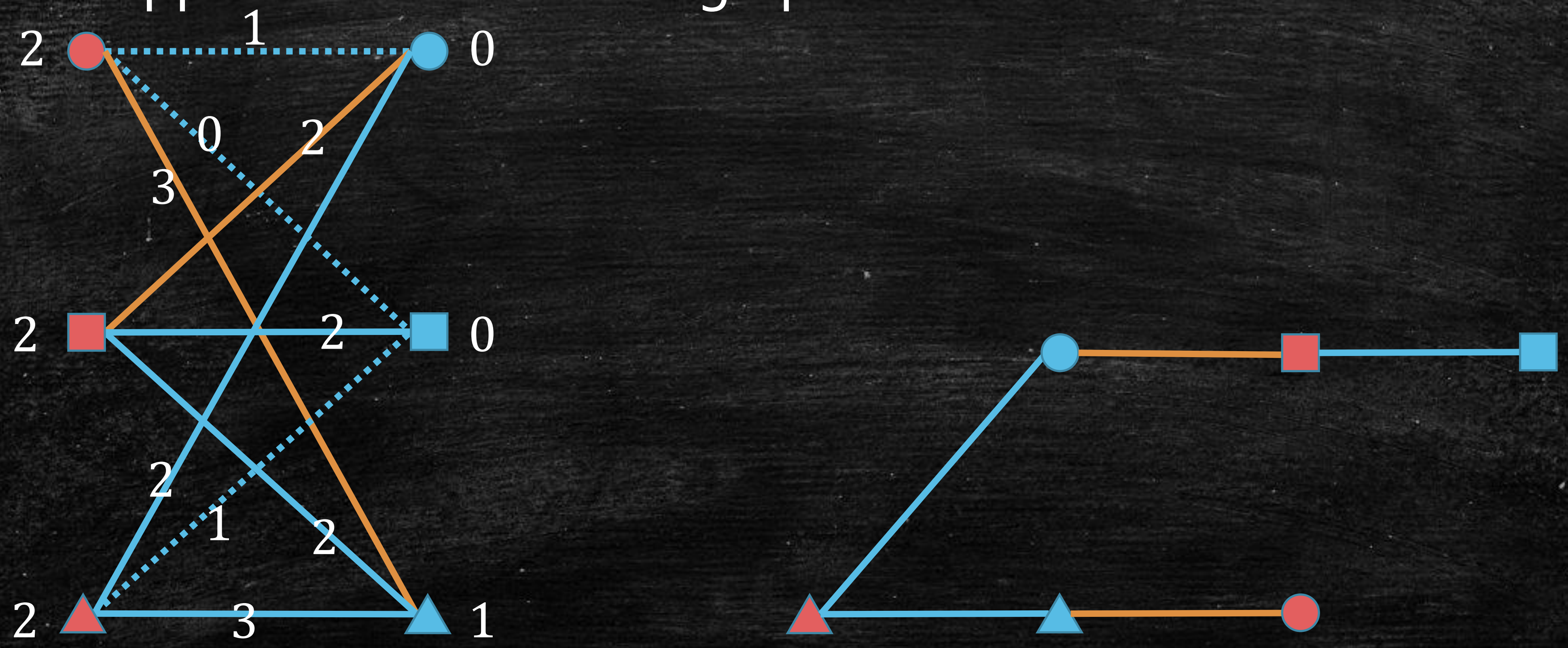
- We see one more tight edge Triangle-Circle.





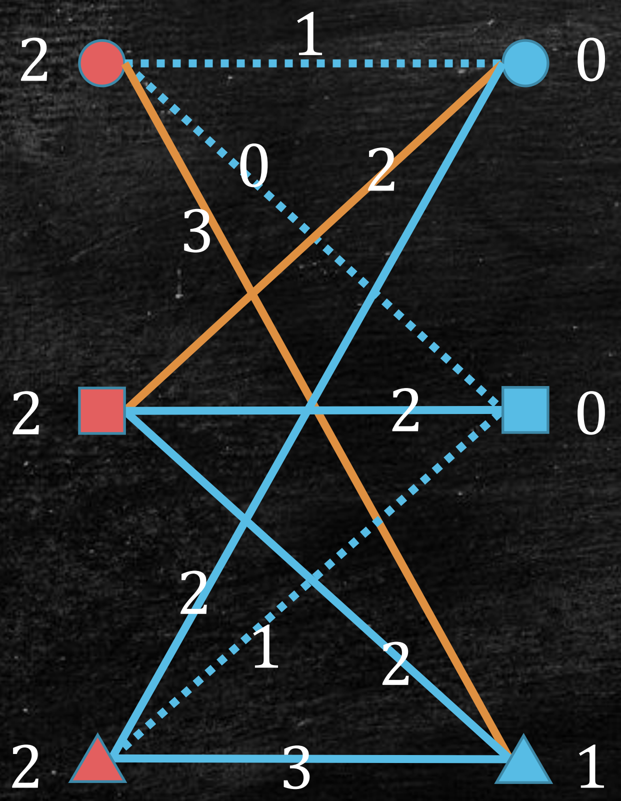
Hungarian Algorithm – Example

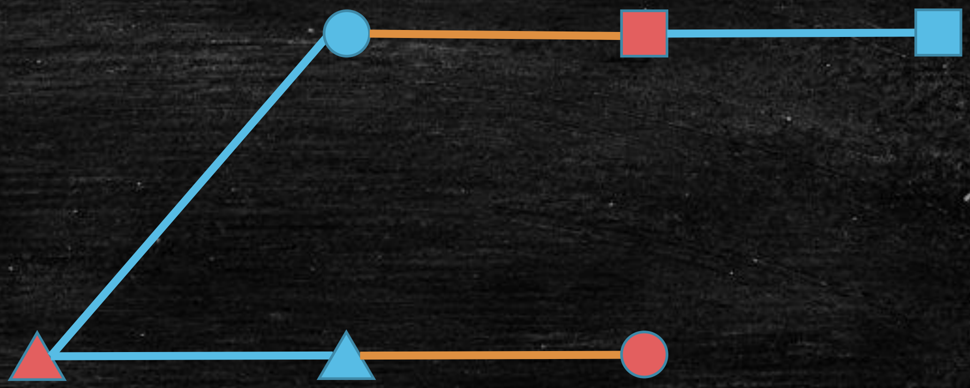
- We see one more tight edge Triangle-Circle.
- Append it in the search graph



Hungarian Algorithm – Example

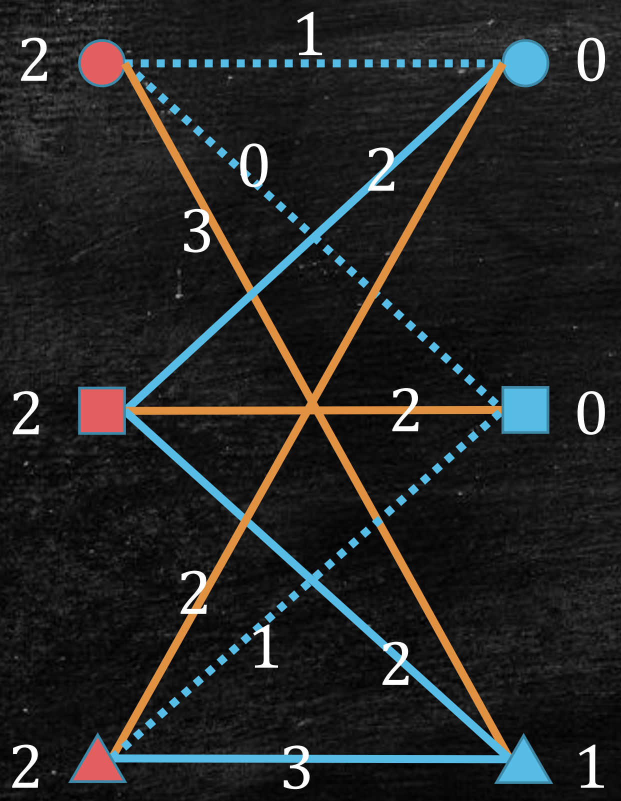
- Now we have one more augmenting path

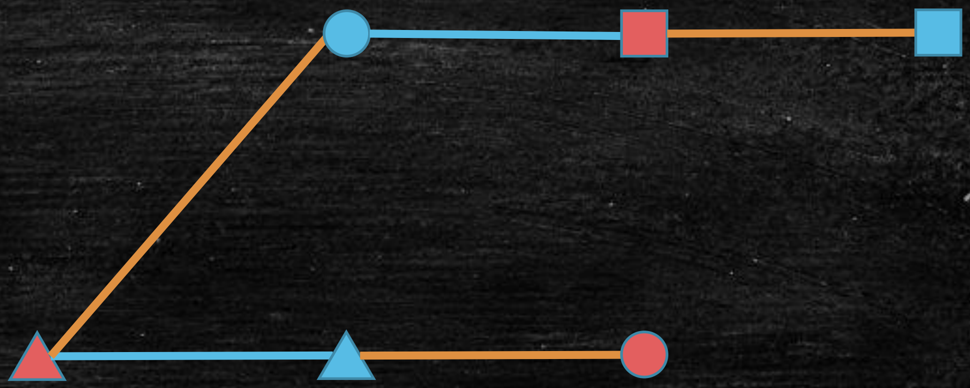




Hungarian Algorithm – Example

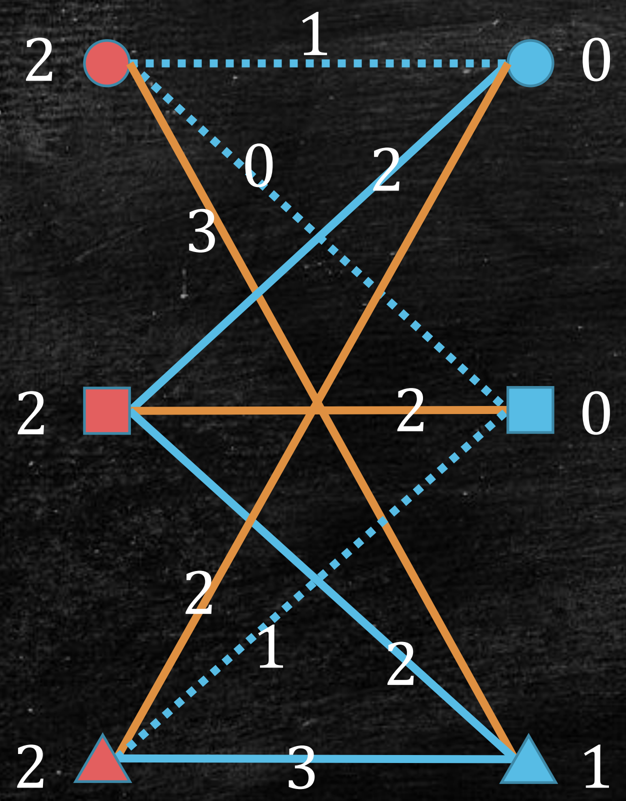
- Update M





Hungarian Algorithm – Example

- Now $M = 3$. We are done!



Correctness

- As long as M is not perfect, we can always do either Update 1 or Update 2.
- Dominance and Tightness hold all the time.
- At the end, **Lemma** (Kuhn & Munkres) implies we have a MWPM.

Time Complexity

- Number of “Update 1”: $O(n)$
- Time complexity of each “Update 1”: $O(n^2)$
- Overall time complexity for all “Update 1”: $O(n^3)$

Time Complexity

- Compute time complexity for those “Update 2” between every two “Update 1”.

For those intermediate “Update 2” between two “update 1”,

- Overall time for search graph: $O(n^2)$
- Overall time for updating p: $O(n^2)$
  - Each update takes $O(n)$ time, and
  - there can be at most n intermediate “Update 2” between two “Update 1”. (why?)
- Overall time for computing Δ in all intermediate “Update 2”: $O(n^2)$
  - We will prove it later…
- Since there are at most n “Update 1”, overall time for all “Update 2”: $O(n^3)$
- Overall time complexity for Hungarian Algorithm: $O(n^3) + O(n^3) = O(n^3)$

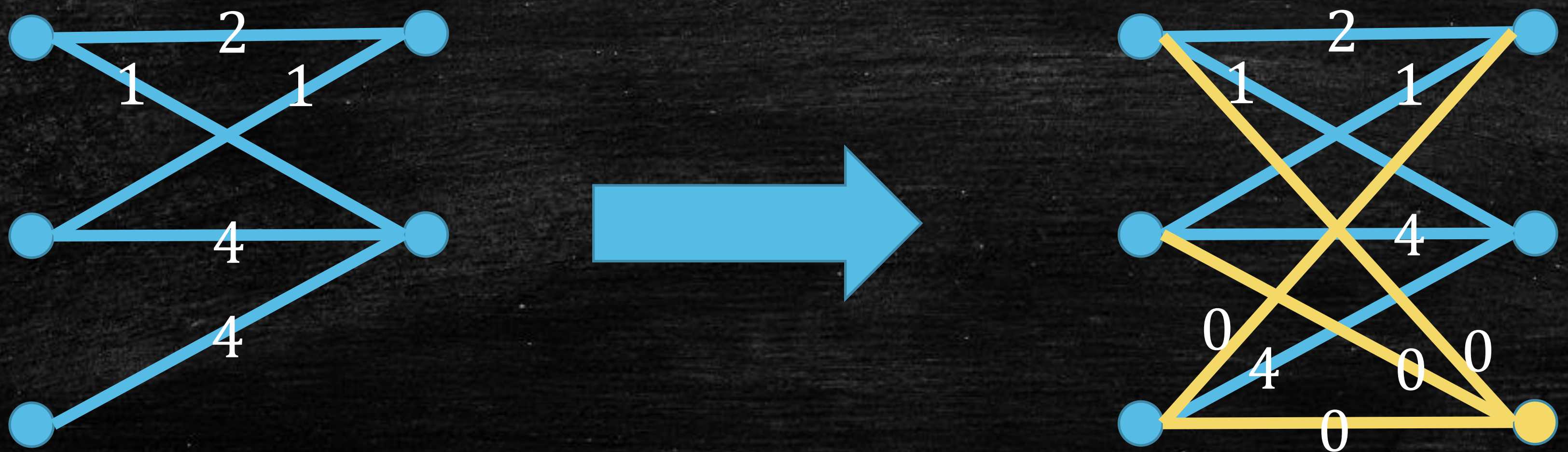
Overall time for computing Δ in all intermediate "Update 2": $O(n^2)$

- We maintain $\text{slack}[u]$ for each $u \in A$ throughout the algorithm
- Compute $\Delta = \min_{u \in A'} \text{slack}[u]$: $O(n^2)$
  - Each search graph expansion takes $O(n)$ time.
  - Search graph can expand at most n times.
- Each time the search graph expands, two types of updates for $\text{slack}[u]$:
  - Easy update: $\text{slack}[u] \leftarrow \text{slack}[u] - \Delta$ if $\text{slack}[u] - \Delta > 0$
  - Advanced update: check every neighbor of u to update $\text{slack}[u]$ if $\text{slack}[u] - \Delta = 0$
- Time for all easy updates: $O(n^2)$
  - Each update $O(1)$; at most $O(n)$ updates for each expansion; at most n expansions.
- Time for all advanced updates: $O(n^2)$
  - Each update $O(n)$
  - an advanced update corresponds to an edge in M added into the search graph (why?)
  - so there are at most n advanced updates

Similar Problems

Similar problems that can be solved by Hungarian Algorithm:

- Minimum Weight Perfect Matching:
  - Just negate the weights of all edges (and add a large number to make them non-negative)
- Maximum Weight Matching:
  - Add vertices and zero-weight edges



History for Hungarian Algorithm

- Invented by Harold Kuhn in 1955.
- Kuhn names it “Hungarian Method” as it is based on two Hungarian mathematicians Dénes Kőnig and Jenő Egerváry.
- James Munkres proves that the algorithm is polynomial time.
- Thus, the algorithm is also called Kuhn-Munkres algorithm.
- Jack Edmonds and Richard Karp: reduce the time complexity from $O(n^4)$ to $O(n^3)$.

Primal-Dual Method

- Hungarian Algorithm "anticipates" primal-dual method.

MWPM (primal)

$$\text{maximize} \quad \sum_{(u,v)} w_{uv} \cdot x_{uv}$$

$$\text{subject to} \quad \forall u: \sum_{v} x_{uv} = 1$$

$$\forall (u,v): x_{uv} \geq 0$$

minimizing "funding" (dual)

$$\text{minimize} \quad \sum_{u \in A \cup B} p_u$$

$$\text{subject to} \quad \forall (u,v): p_u + p_v \geq w_{uv}$$

Part II: Metric Facility Location

Metric Facility Location

- A complete positively weighted undirected graph $G = (V, E, d: E \to \mathbb{R}^+)$
  - Weights with triangle inequality: $d(u, v) + d(v, w) \geq d(u, w)$
- Vertices partitioned to $V = F \cup C$:
  - F: set of possible locations for building facilities
  - C: set of locations for clients
- Building a facility $i \in F$ requires a building cost f_i.
- Connecting a client $j \in C$ to a facility $i \in F$ requires a connection cost $d_{ij} = d(i, j)$.
- Objective: open facilities $S \subseteq F$ minimizing the overall cost

$$\sum_{i \in S} f_i + \sum_{j \in C} d(j, S)$$

IP Formulation

- $x_i \in \{0,1\}$: whether facility at i is open
- $y_{ij} \in \{0,1\}$: whether client j is connected to facility i
- Overall cost: $\sum_{i \in S} f_i x_i + \sum_{j \in C} d_{ij} y_{ij}$
- Each client j must be connected: $\sum_{i \in F} y_{ij} = 1$
- The facility i must be open if being connected: $y_{ij} \le x_i$

IP Formulation

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in S} f_i x_i + \sum_{i \in F, j \in C} d_{ij} y_{ij} & \\
\text{subject to} \quad & \sum_{i \in F} y_{ij} = 1 & \forall j \in C \\
& y_{ij} \leq x_i & \forall i \in F, j \in C \\
& y_{ij} \in \{0,1\} & \forall i \in F, j \in C \\
& x_i \in \{0,1\} & \forall i \in F
\end{aligned}
$$

LP Relaxation

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in S} f_i x_i + \sum_{i \in F, j \in C} d_{ij} y_{ij} \\
\text{subject to} \quad & \sum_{i \in F} y_{ij} = 1 && \forall j \in C \\
& y_{ij} \leq x_i && \forall i \in F, j \in C \\
& 0 \leq y_{ij} \leq 1 && \forall i \in F, j \in C \\
& 0 \leq x_i \leq 1 && \forall i \in F
\end{aligned}$$

LP Relaxation

- Let $\{x_i, y_{ij}\}$ be the optimal LP solution.
- Let $F^* = \sum_{i\in F} f_i x_i$ be the LP building cost.
- Let $D^* = \sum_{i\in F, j\in C} d_{ij} y_{ij}$ be the LP connection cost.
- Our objective: construct an integral solution S and compare its cost to the optimal LP cost $F^* + D^*$
- Let $L_j = \sum_{i\in F} d_{ij} y_{ij}$ be the LP connection cost for j.
- L_j can be viewed as the "weighted-average distance" to all facilities.

Balls

- $B(v, r)$: the set of all vertices within distance r from vertex v
  - A "ball" centered at v with radius r
- Choose a parameter $\alpha > 1$ (to be decided later)
- For each client j, let $B_j = B(j, \alpha L_j)$
- B_j should contain most "mass" of facilities connected to j, if α is large.

Algorithm

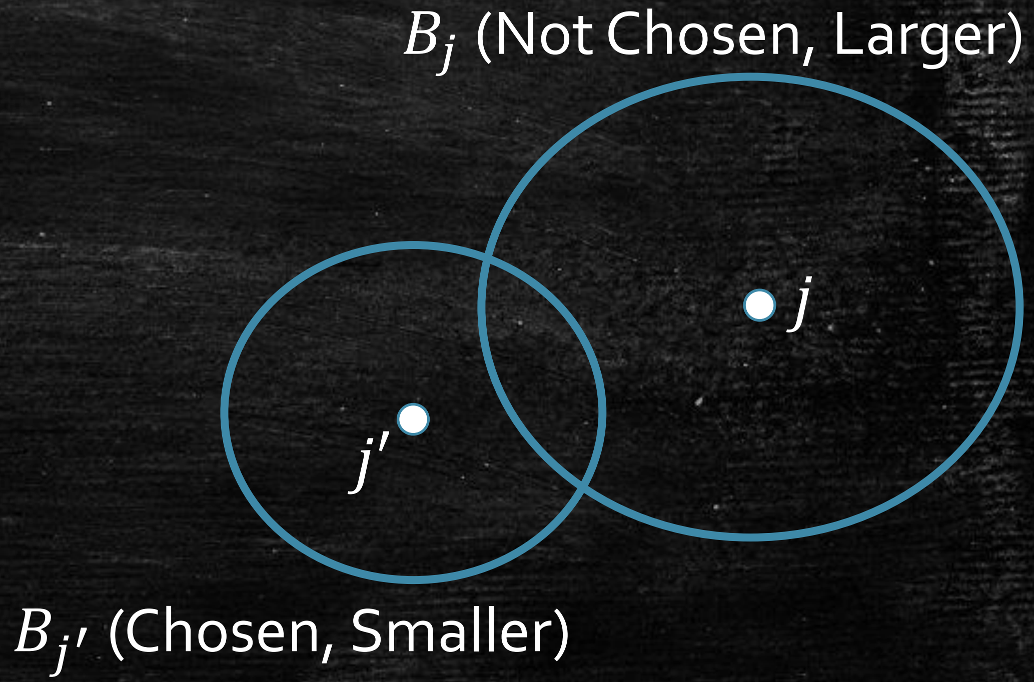
1. Sort the clients as $L_1 \leq L_2 \leq \cdots \leq L_{|C|}$
2. Greedily choose a maximal subset of disjoint balls in order $1, 2, \ldots, |C|$. Let $I \subseteq C$ encode the chosen balls $\{B_j : j \in I\}$.
3. Build the cheapest facility $\pi(j)$ in each chosen ball.
4. Return S containing all the opened facilities.

Bound the Connection Cost

- For each $j \in I$, the connection cost is at most αL_j
  - Since $\pi(j)$ is opened in $B_j = B(j, \alpha L_j)$

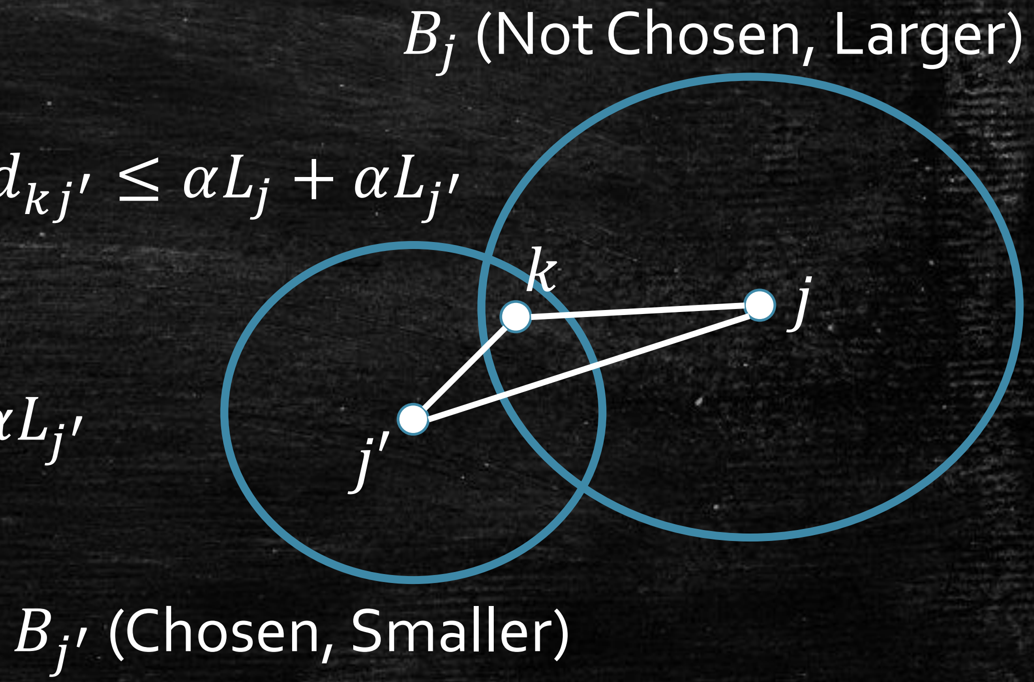
Bound the Connection Cost

- For each $j \in I$, the connection cost is at most αL_j
- For each $j \notin I$, there exists $j' \in I$ with
  - $j' < j$
  - $L_{j'} \le L_j$
  - $B_{j'} \cap B_j \neq \emptyset$



Bound the Connection Cost

- For each $j \in I$, the connection cost is at most αL_j
- For each $j \notin I$, there exists $j' \in I$ with
  - $j' < j$
  - $L_{j'} \le L_j$
  - $B_{j'} \cap B_j \neq \emptyset$
- For $k \in B_{j'} \cap B_j$, we have $d_{jj'} \le d_{jk} + d_{kj'} \le \alpha L_j + \alpha L_{j'}$
- Connection cost for j is at most
- $d_{\pi(j')j} \le d_{\pi(j')j'} + d_{jj'} \le \alpha L_{j'} + \alpha L_j + \alpha L_{j'}$
- $L_{j'} \le L_j \quad \Longrightarrow d_{\pi(j')j} \le 3\alpha L_j$



Bound the Connection Cost

- Connection cost for each $j \in C$: at most $3\alpha L_j$
- Overall Connection Cost is at most:

$$\sum_{j \in C} 3\alpha L_j = 3\alpha \sum_{j \in C} \sum_{i \in F} d_{ij} y_{ij} = 3\alpha D^*$$

Bound the Building Cost

- B_j should contain most "mass" of facilities connected to j, if α is large.
- In particular, $\sum_{k \in B_j} y_{kj} \geq 1 - \frac{1}{\alpha}$:
  - O.w., $\sum_{k \notin B_j} y_{kj} > \frac{1}{\alpha}$, and
  - $L_j = \sum_{i \in F} d_{ij} y_{ij} \geq \sum_{k \notin B_j} d_{ij} y_{kj} > \alpha L_j \sum_{k \notin B_j} y_{kj} > L_j$, a contradiction!

Bound the Building Cost

- B_j should contain most "mass" of facilities connected to j, if α is large.
- In particular, $\sum_{k\in B_j} y_{kj} \geq 1 - \frac{1}{\alpha}$
- For each opened facility $\pi(j)$,

$$f_{\pi(j)} = \min_{k\in F\cap B_j} f_k \leq \frac{\sum_{k\in B_j} f_k y_{kj}}{\sum_{k\in B_j} y_{kj}} \leq \frac{1}{1-\frac{1}{\alpha}} \sum_{k\in B_j} f_k y_{kj}$$

min<weighted-average yellow

Bound the Building Cost

- B_j should contain most "mass" of facilities connected to j, if α is large.
- In particular, $\sum_{k\in B_j} y_{kj} \geq 1 - \frac{1}{\alpha}$
- For each opened facility $\pi(j)$,

$$f_{\pi(j)} = \min_{k\in F\cap B_j} f_k \leq \frac{\sum_{k\in B_j} f_k y_{kj}}{\sum_{k\in B_j} y_{kj}} \leq \frac{1}{1-\frac{1}{\alpha}} \sum_{k\in B_j} f_k y_{kj}$$

- Overall Building Cost:

$$\sum_{j\in I} f_{\pi(j)} \leq \frac{1}{1-\frac{1}{\alpha}} \sum_{j\in I} \sum_{k\in B_j} f_k y_{kj} \leq \frac{1}{1-\frac{1}{\alpha}} \sum_{j\in I} \sum_{k\in B_j} f_k x_k \leq \frac{1}{1-\frac{1}{\alpha}} F^*$$

orangeLP constraintballs are disjoint

Summarizing

- Overall Connection Cost $\leq 3\alpha D^*$
- Overall Building Cost $\leq \frac{1}{1-\frac{1}{\alpha}} F^*$
- Overall Cost $\leq 3\alpha D^* + \frac{1}{1-\frac{1}{\alpha}} F^* = \max\left(\frac{\alpha}{\alpha-1}, 3\alpha\right)(F^* + D^*)$
- $\alpha = \frac{4}{3} \quad \Longrightarrow \quad$ Overall Cost $\leq 4(F^* + D^*)$
- LP optimum $F^* + D^*$ is a lower bound to OPT
- We have a 4-approximation algorithm!

Results for Metric Facility Location

- This algorithm (Greedy + LP-relaxation): 4-approximation
- Primal-Dual Schema: 3-approximation
- [Li, 2011] 1.488-approximation
- [Guha & Khuller, 1999] 1.463-approximation is NP-hard
  - Reduction from Set Cover
  - At that time, the $(1-o(1))\ln n$ inapproximability for Set Cover is only known to be based on $\mathbf{NP} \not\subseteq \mathrm{DTIME}\left(n^{O(\log\log n)}\right)$
  - Thus, Guha & Khuller concludes 1.463 inapproximability for Metric Facility Location based on $\mathbf{NP} \not\subseteq \mathrm{DTIME}\left(n^{O(\log\log n)}\right)$
  - But now we know 1.463-approximation is NP-hard since $(1-o(1))\ln n$ NP-hardness of approximation is now known for Set Cover