Algorithms

Lecture Topic: Matchings in Bipartite Graphs

Anxiao (Andrew) Jiang

Roadmap of this lecture:

1. Matching in Bipartite Graphs

    1.1 Define "Assignment Problem".

    1.2 Concepts useful for finding an optimal solution.

    1.3 Hungarian Algorithm for "Assignment Problem".

The Hungarian algorithm for the assignment problem

Assignment Problem

Input: A complete bipartite graph $G = (V, E)$, where $V = L \cup R$ and $|L| = |R| = n$.

Every edge $(l, r) \in E$ has a weight $w(l, r)$, for $l \in L$ and $r \in R$.

Output: A perfect matching $M^* \subseteq E$ with the maximum total edge weights. That is, let $w(M) = \sum_{(l,r) \in M} w(l, r)$ denote the total weight of the edges in a matching M, then

$w(M^*) = \max\{w(M) \ : \ M \text{ is a pefect matching}\}$.

$n = 3$

$L = \{l_1, l_2, l_3\}$

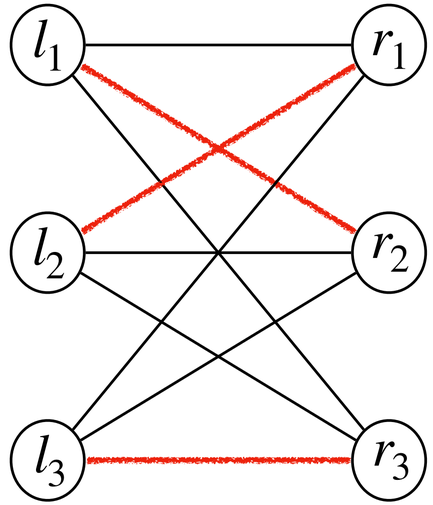
$R = \{r_1, r_2, r_3\}$

each black edge: weight 1

each red edge: weight 2

An optimal perfect matching:

$M^* = \{(l_1, r_2), (l_2, r_1), (l_3, r_3)\}$



Assignment Problem

Input: A complete bipartite graph $G = (V, E)$, where $V = L \cup R$ and $|L| = |R| = n$.

Every edge $(l, r) \in E$ has a weight $w(l, r)$, for $l \in L$ and $r \in R$.

Output: A perfect matching $M^* \subseteq E$ with the maximum total edge weights. That is, let $w(M) = \sum_{(l,r) \in M} w(l, r)$ denote the total weight of the edges in a matching M, then $w(M^*) = \max\{w(M) \ : \ M \text{ is a pefect matching}\}$.

Applications: match jobs and resources with optimal utility, etc.

Solution: Hungarian algorithm

Time complexity: $O(n^4)$, which can be further refined to $O(n^3)$

Quiz questions:

1. What is the difference between the “Assignment Problem” and the “Maximum Bipartite Matching Problem”?

2. What are the applications of the “Assignment Problem”?

Roadmap of this lecture:

1. Matching in Bipartite Graphs

    1.1 Define "Assignment Problem".

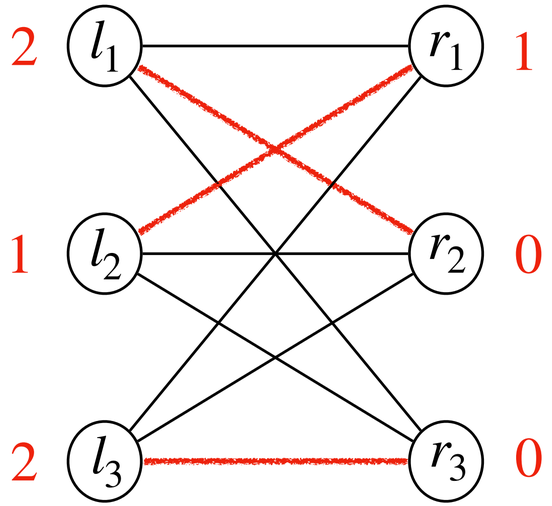
    1.2 Concepts useful for finding an optimal solution.

    1.3 Hungarian Algorithm for "Assignment Problem".

Label: Assign a number $v.h$ to each vertex $v \in V$.

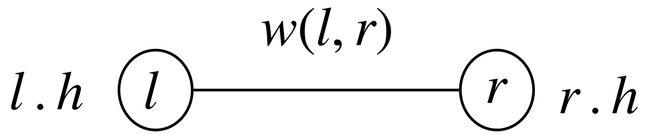
$n = 3$
$L = \{l_1, l_2, l_3\}$
$R = \{r_1, r_2, r_3\}$

each black edge: weight 1
each red edge: weight 2



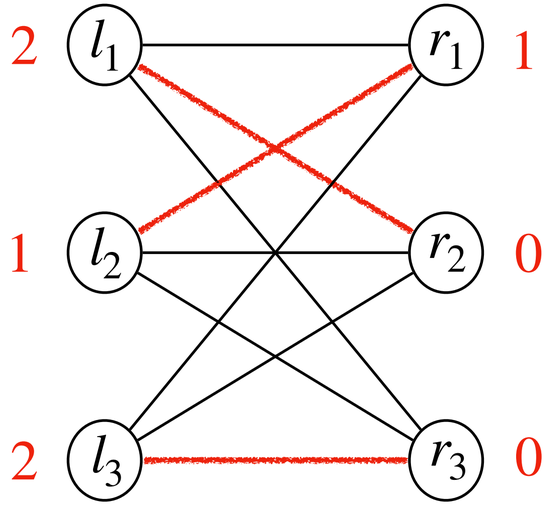
Label: Assign a number $v.h$ to each vertex $v \in V$.

Feasible vertex labeling: h is a feasible vertex labeling of G if $l.h + r.h \geq w(l,r)$ for all $l \in L$ and $r \in R$.



$n = 3$
$L = \{l_1, l_2, l_3\}$
$R = \{r_1, r_2, r_3\}$

each black edge: weight 1
each red edge: weight 2



Label: Assign a number $v.h$ to each vertex $v \in V$.

Feasible vertex labeling: h is a feasible vertex labeling of G if $l.h + r.h \geq w(l,r)$ for all $l \in L$ and $r \in R$.

Default vertex labeling: The following feasible vertex labeling is called the default vertex labeling:

$l.h = \max\{w(l,r) : r \in R\}$ for all $l \in L$,

$r.h = 0$ for all $r \in R$.

For a vertex on the left, its label equals the maximum weight of its edges;

For a vertex on the right, its label equals 0.

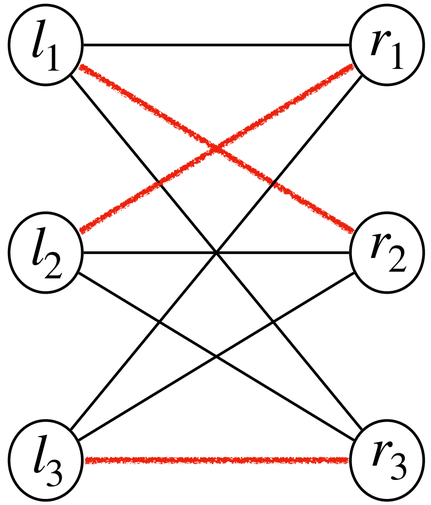
$n = 3$

$L = \{l_1, l_2, l_3\}$

$R = \{r_1, r_2, r_3\}$

each black edge: weight 1

each red edge: weight 2



Label: Assign a number $v.h$ to each vertex $v \in V$.

Feasible vertex labeling: h is a feasible vertex labeling of G if $l.h + r.h \geq w(l,r)$ for all $l \in L$ and $r \in R$.

Default vertex labeling: The following feasible vertex labeling is called the default vertex labeling:

$l.h = \max\{w(l,r) : r \in R\}$ for all $l \in L$,

$r.h = 0$ for all $r \in R$.

For a vertex on the left, its label equals the maximum weight of its edges;

For a vertex on the right, its label equals 0.

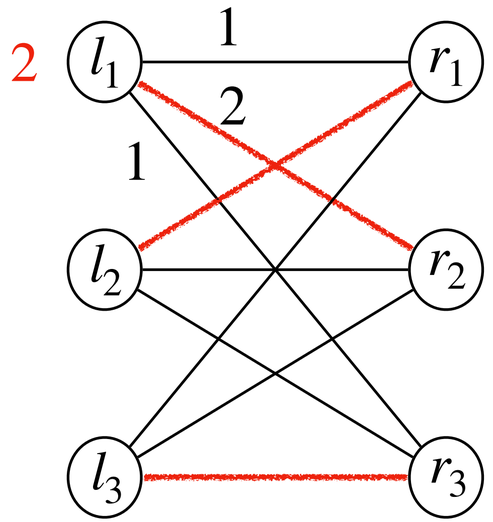
$n = 3$

$L = \{l_1, l_2, l_3\}$

$R = \{r_1, r_2, r_3\}$

each black edge: weight 1

each red edge: weight 2



Label: Assign a number $v.h$ to each vertex $v \in V$.

Feasible vertex labeling: h is a feasible vertex labeling of G if $l.h + r.h \geq w(l,r)$ for all $l \in L$ and $r \in R$.

Default vertex labeling: The following feasible vertex labeling is called the default vertex labeling:

$l.h = \max\{w(l,r) : r \in R\}$ for all $l \in L$,

$r.h = 0$ for all $r \in R$.

For a vertex on the left, its label equals the maximum weight of its edges;

For a vertex on the right, its label equals 0.

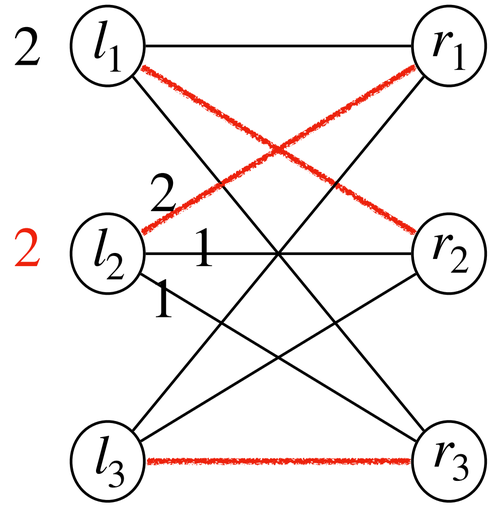
$n = 3$

$L = \{l_1, l_2, l_3\}$

$R = \{r_1, r_2, r_3\}$

each black edge: weight 1

each red edge: weight 2



Label: Assign a number $v.h$ to each vertex $v \in V$.

Feasible vertex labeling: h is a feasible vertex labeling of G if $l.h + r.h \geq w(l,r)$ for all $l \in L$ and $r \in R$.

Default vertex labeling: The following feasible vertex labeling is called the default vertex labeling:

$l.h = \max\{w(l,r) : r \in R\}$ for all $l \in L$,

$r.h = 0$ for all $r \in R$.

For a vertex on the left, its label equals the maximum weight of its edges;

For a vertex on the right, its label equals 0.

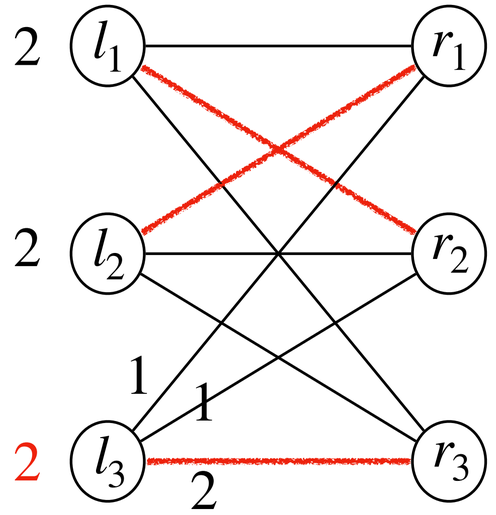
$n = 3$

$L = \{l_1, l_2, l_3\}$

$R = \{r_1, r_2, r_3\}$

each black edge: weight 1

each red edge: weight 2



Label: Assign a number $v.h$ to each vertex $v \in V$.

Feasible vertex labeling: h is a feasible vertex labeling of G if $l.h + r.h \geq w(l,r)$ for all $l \in L$ and $r \in R$.

Default vertex labeling: The following feasible vertex labeling is called the default vertex labeling:

$l.h = \max\{w(l,r) : r \in R\}$ for all $l \in L$,

$r.h = 0$ for all $r \in R$.

For a vertex on the left, its label equals the maximum weight of its edges;

For a vertex on the right, its label equals 0.

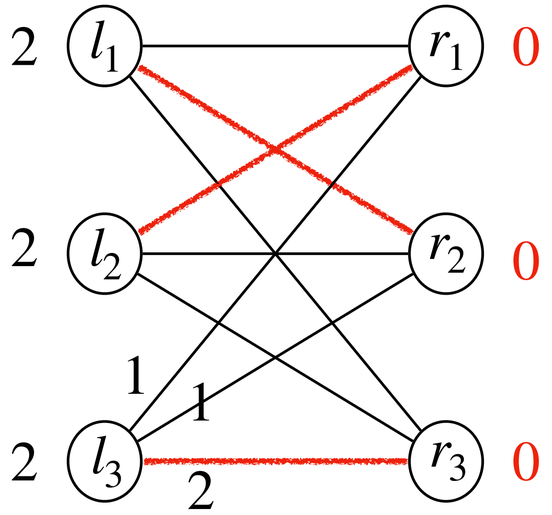
$n = 3$

$L = \{l_1, l_2, l_3\}$

$R = \{r_1, r_2, r_3\}$

each black edge: weight 1

each red edge: weight 2

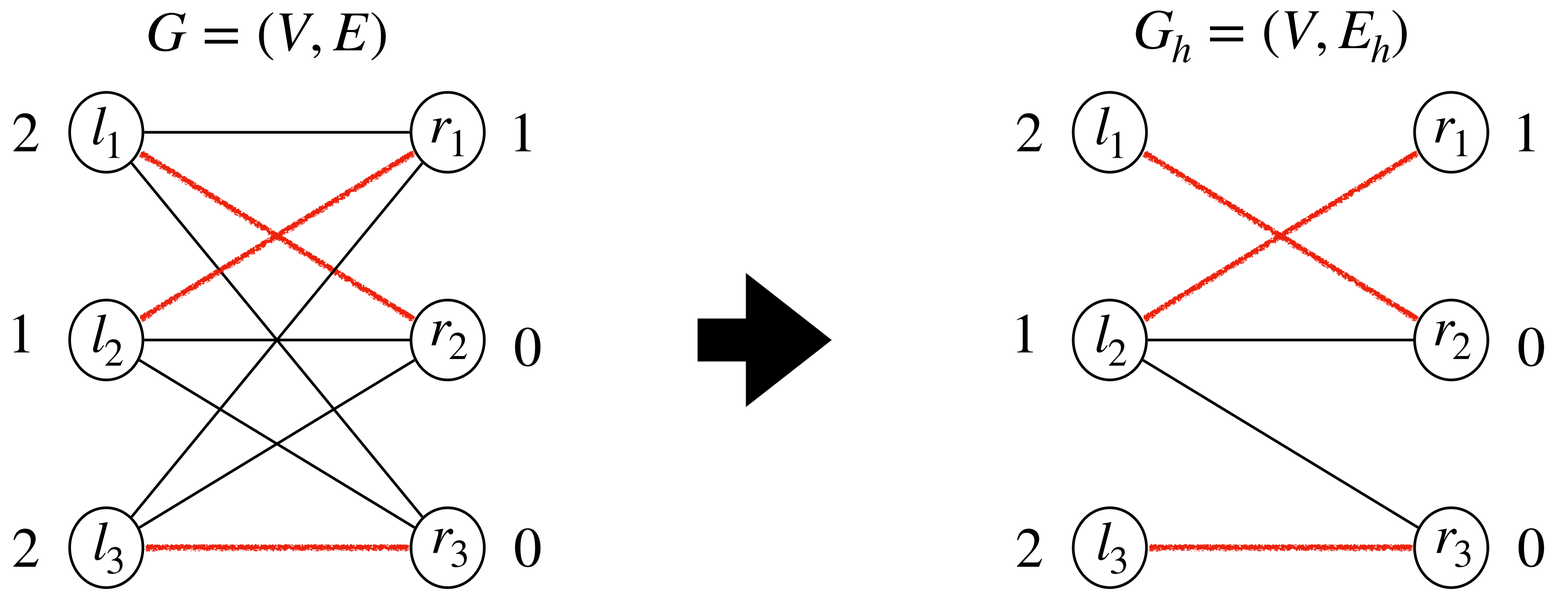


Default vertex labeling: The following feasible vertex labeling is called the default vertex labeling:

$$l.h = \max\{w(l,r) : r \in R\} \text{ for all } l \in L,$$
$$r.h = 0 \quad \text{for all } r \in R.$$

Equality subgraph: Given a feasible vertex labeling h, the equality subgraph $G_h = (V, E_h)$ of G consists of the same vertices as G and the subset of edges

$$E_h = \{(l,r) \in E : l.h + r.h = w(l,r)\}.$$



Theorem: Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

Theorem: Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

To find a perfect matching of maximum weight in G, we just need to find a perfect matching in G_h.

Catch: a perfect matching in G_h needs to exist. So the feasible vertex labeling h needs to be "right".

Theorem: Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

Proof: M^* is also a perfect matching in G, because G and G_h have the same vertex set V.

Theorem: Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

Proof: M^* is also a perfect matching in G, because G and G_h have the same vertex set V.

$$w(M^*) = \sum_{(l,r) \in M^*} w(l, r) = \sum_{(l,r) \in M^*} (l.h + r.h) = \sum_{l \in L} l.h + \sum_{r \in R} r.h$$

Theorem: Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

Proof: M^* is also a perfect matching in G, because G and G_h have the same vertex set V.

$$w(M^*) = \sum_{(l,r)\in M^*} w(l,r) = \sum_{(l,r)\in M^*} (l.h + r.h) = \sum_{l\in L} l.h + \sum_{r\in R} r.h$$

Let M be any perfect matching in G,

$$w(M) = \sum_{(l,r)\in M} w(l,r) \leq \sum_{(l,r)\in M} (l.h + r.h) = \sum_{l\in L} l.h + \sum_{r\in R} r.h$$

Theorem: Let $G = (V, E)$, where $V = L \cup R$, be a complete bipartite graph where each edge $(l, r) \in E$ has weight $w(l, r)$. Let h be a feasible vertex labeling of G and G_h be the equality subgraph of G. If G_h contains a perfect matching M^*, then M^* is an optimal solution to the assignment problem on G.

Proof: M^* is also a perfect matching in G, because G and G_h have the same vertex set V.

$$w(M^*) = \sum_{(l,r)\in M^*} w(l, r) = \sum_{(l,r)\in M^*} (l.h + r.h) = \sum_{l\in L} l.h + \sum_{r\in R} r.h$$

Let M be any perfect matching in G,

$$w(M) = \sum_{(l,r)\in M} w(l, r) \le \sum_{(l,r)\in M} (l.h + r.h) = \sum_{l\in L} l.h + \sum_{r\in R} r.h$$

Thus we have

$$w(M) \le \sum_{l\in L} l.h + \sum_{r\in R} r.h = w(M^*)$$

Quiz questions:

1. What is "feasible vertex labeling"?
2. What is "equality subgraph"?
3. How to tell if a solution to the “Assignment Problem” is optimal?

Roadmap of this lecture:

1. Matching in Bipartite Graphs

    1.1 Define "Assignment Problem".

    1.2 Concepts useful for finding an optimal solution.

    1.3 Hungarian Algorithm for "Assignment Problem".

Basic idea of Hungarian Algorithm:

1. Start with any feasible vertex labeling h and any matching M in the equality subgraph G_h.
2. Keep doing this until we find a perfect matching M in an equality subgraph G_h :

   If there is an M-augmenting path P, update the matching to $M \oplus P$.

   If there is no M-augmenting path, update h, then update G_h accordingly.

Basic idea of Hungarian Algorithm:

1. Start with any feasible vertex labeling h and any matching M in the equality subgraph G_h.
2. Keep doing this until we find a perfect matching M in an equality subgraph G_h :

   If there is an M-augmenting path P, update the matching to $M \oplus P$.

   If there is no M-augmenting path, update h, then update G_h accordingly.

Four questions arise:

1. What initial feasible vertex labeling should the algorithm start with?
   Answer: the default vertex labeling would work.

Basic idea of Hungarian Algorithm:

1. Start with any feasible vertex labeling h and any matching M in the equality subgraph G_h.
2. Keep doing this until we find a perfect matching M in an equality subgraph G_h :

   If there is an M-augmenting path P, update the matching to $M \oplus P$.

   If there is no M-augmenting path, update h, then update G_h accordingly.

Four questions arise:

1. What initial feasible vertex labeling should the algorithm start with?
   Answer: the default vertex labeling would work.
2. What initial matching in G_h should the algorithm start with?
   Short answer: any matching, even an empty matching, but a greedy maximal matching works well.

Basic idea of Hungarian Algorithm:

1. Start with any feasible vertex labeling h and any matching M in the equality subgraph G_h.
2. Keep doing this until we find a perfect matching M in an equality subgraph G_h :

   If there is an M-augmenting path P, update the matching to $M \oplus P$.

   If there is no M-augmenting path, update h, then update G_h accordingly.

Four questions arise:

1. What initial feasible vertex labeling should the algorithm start with?
   Answer: the default vertex labeling would work.
2. What initial matching in G_h should the algorithm start with?
   Short answer: any matching, even an empty matching, but a greedy maximal matching works well.
3. If an M-augmenting path exists in G_h, how to find it?
   Short answer: use a variant of BFS similar to the second phase of the the procedure used in the Hopcroft-Karp Algorithm to find a maximal set of shortest M-augmenting paths.

Basic idea of Hungarian Algorithm:

1. Start with any feasible vertex labeling h and any matching M in the equality subgraph G_h.
2. Keep doing this until we find a perfect matching M in an equality subgraph G_h :

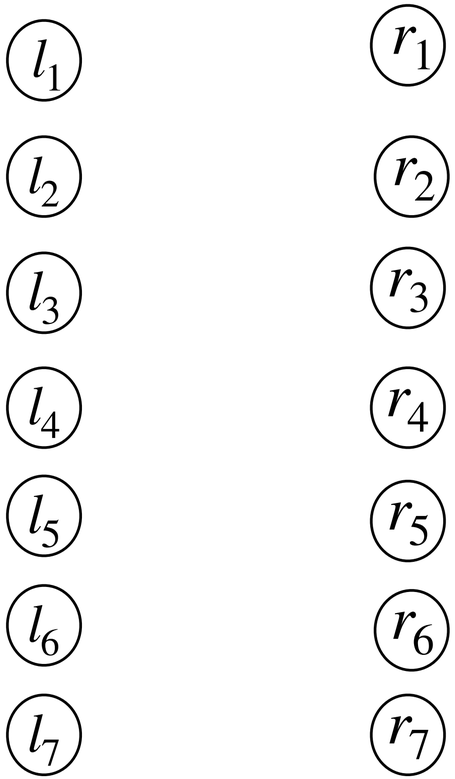
   If there is an M-augmenting path P, update the matching to $M \oplus P$.

   If there is no M-augmenting path, update h, then update G_h accordingly.

Four questions arise:

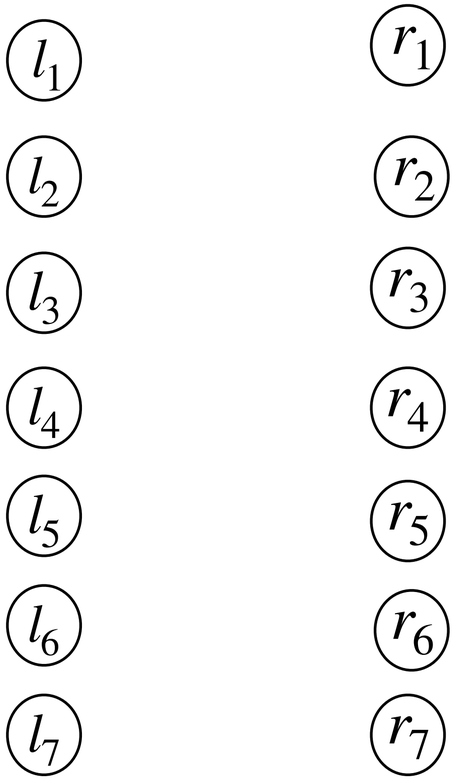
1. What initial feasible vertex labeling should the algorithm start with?
   Answer: the default vertex labeling would work.
2. What initial matching in G_h should the algorithm start with?
   Short answer: any matching, even an empty matching, but a greedy maximal matching works well.
3. If an M-augmenting path exists in G_h, how to find it?
   Short answer: use a variant of BFS similar to the second phase of the the procedure used in the Hopcroft-Karp Algorithm to find a maximal set of shortest M-augmenting paths.
4. What if the search for an M-augmenting path fails?
   short answer: update the feasible vertex labeling to bring in at least one new edge.

h	r_1	r_2	r_3	r_4	r_5	r_6	r_7
l_1	4	10	10	10	2	9	3
l_2	6	8	5	12	9	7	2
l_3	11	9	6	7	9	5	15
l_4	3	9	6	7	5	6	3
l_5	2	6	5	3	2	4	2
l_6	10	8	11	4	11	2	11
l_7	3	4	5	4	3	6	8



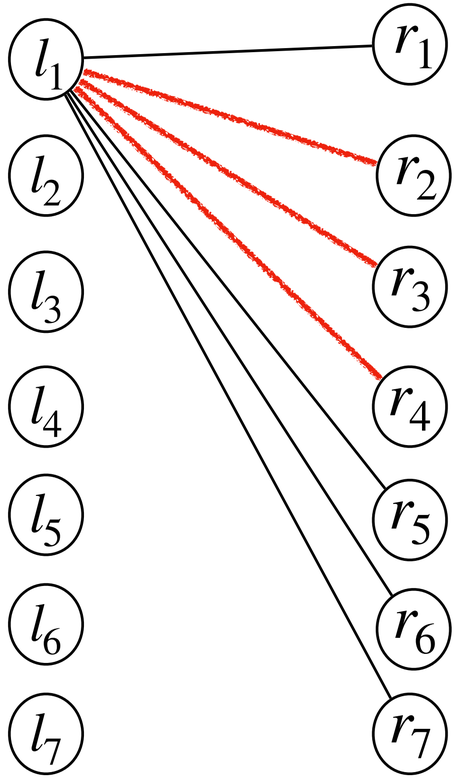
default vertex labeling

	r_1	r_2	r_3	r_4	r_5	r_6	r_7
h	0	0	0	0	0	0	0
l_1	4	10	10	10	2	9	3
l_2	6	8	5	12	9	7	2
l_3	11	9	6	7	9	5	15
l_4	3	9	6	7	5	6	3
l_5	2	6	5	3	2	4	2
l_6	10	8	11	4	11	2	11
l_7	3	4	5	4	3	6	8



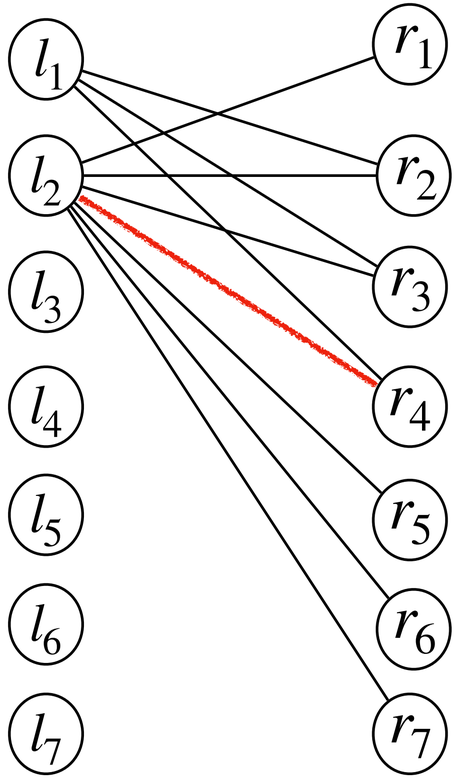
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2		6	8	5	12	9	7	2
l_3		11	9	6	7	9	5	15
l_4		3	9	6	7	5	6	3
l_5		2	6	5	3	2	4	2
l_6		10	8	11	4	11	2	11
l_7		3	4	5	4	3	6	8



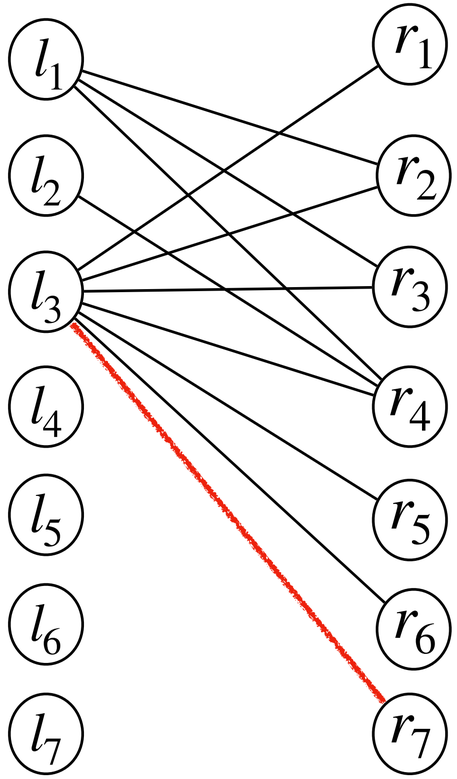
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3		11	9	6	7	9	5	15
l_4		3	9	6	7	5	6	3
l_5		2	6	5	3	2	4	2
l_6		10	8	11	4	11	2	11
l_7		3	4	5	4	3	6	8



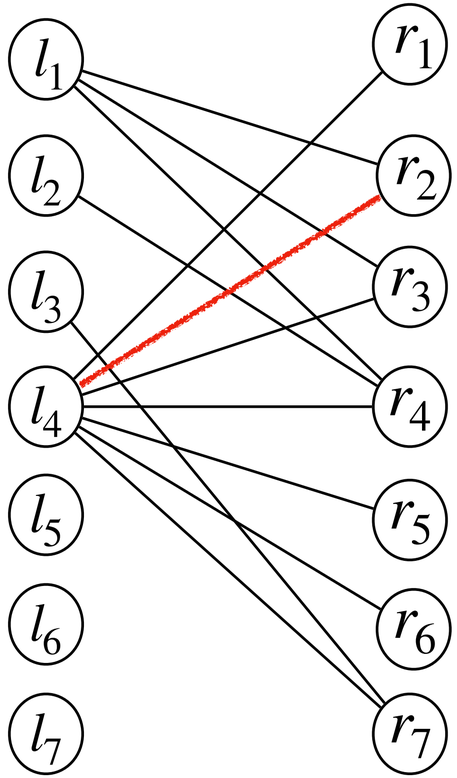
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4		3	9	6	7	5	6	3
l_5		2	6	5	3	2	4	2
l_6		10	8	11	4	11	2	11
l_7		3	4	5	4	3	6	8



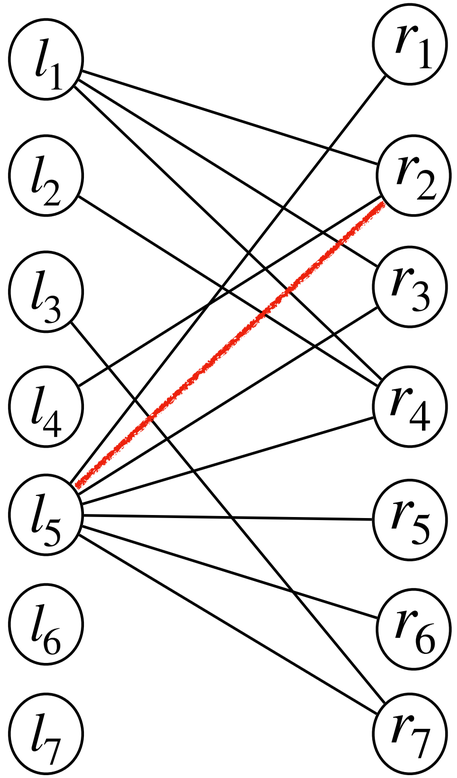
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5		2	6	5	3	2	4	2
l_6		10	8	11	4	11	2	11
l_7		3	4	5	4	3	6	8



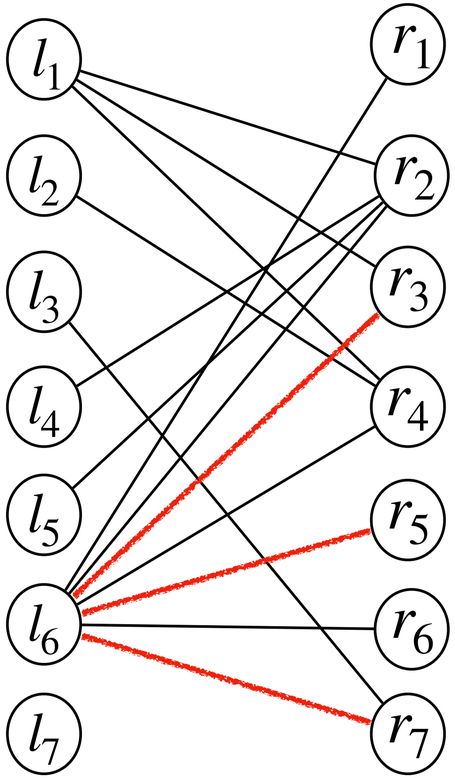
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6		10	8	11	4	11	2	11
l_7		3	4	5	4	3	6	8



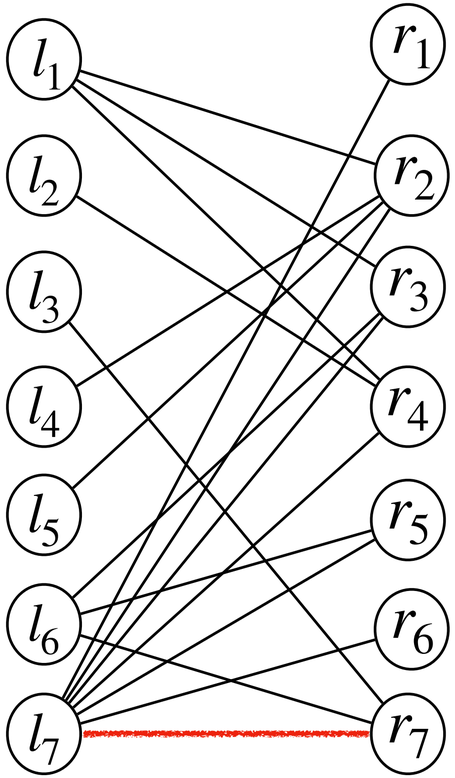
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7		3	4	5	4	3	6	8



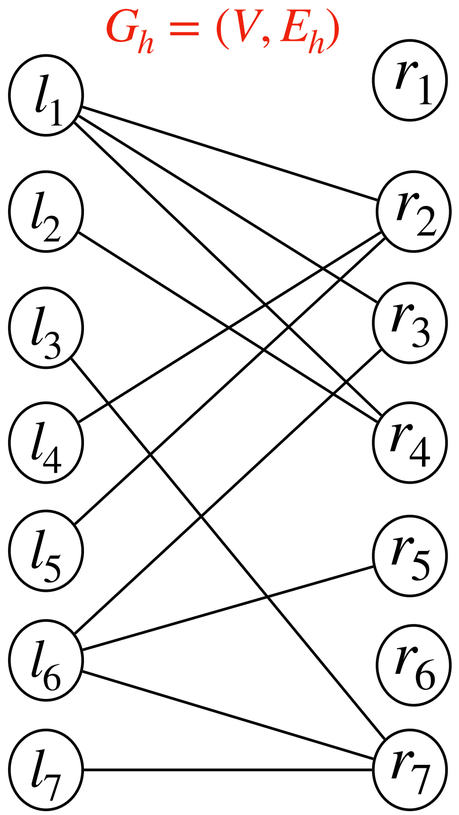
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



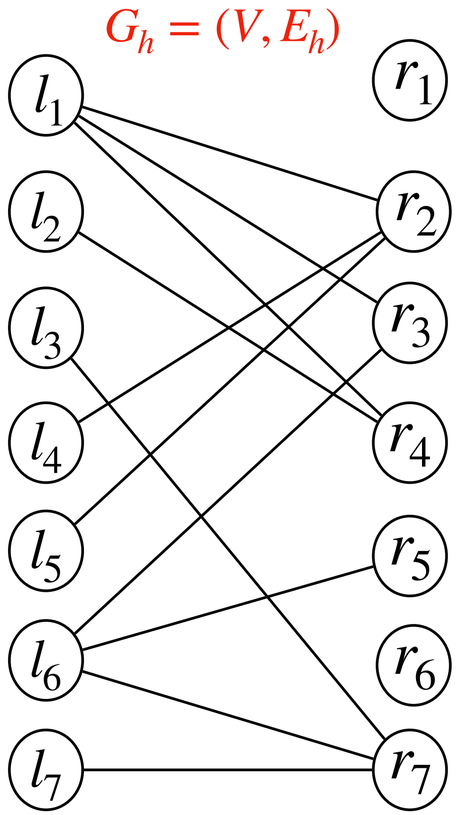
default vertex labeling

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



equality subgraph

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

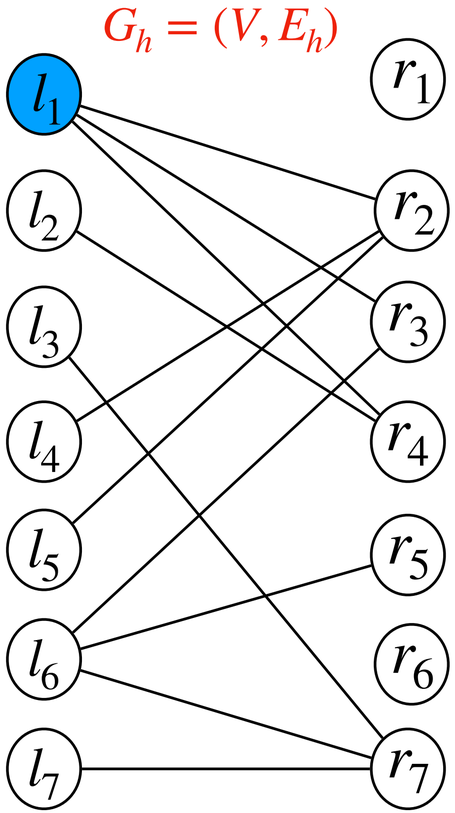


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

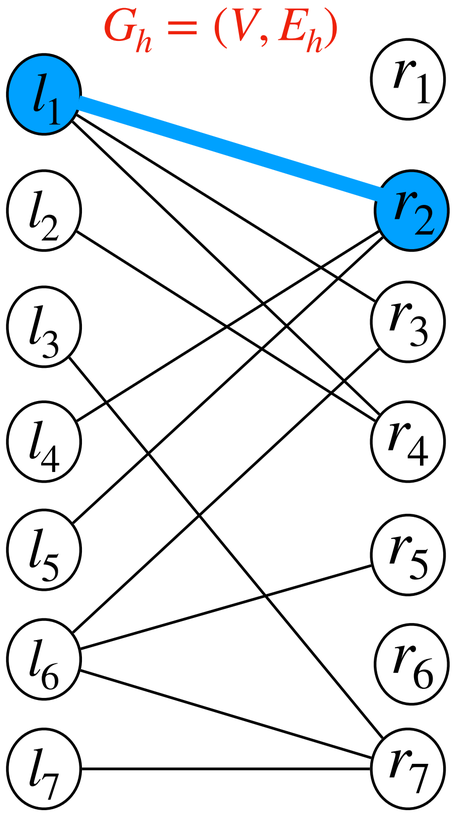


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

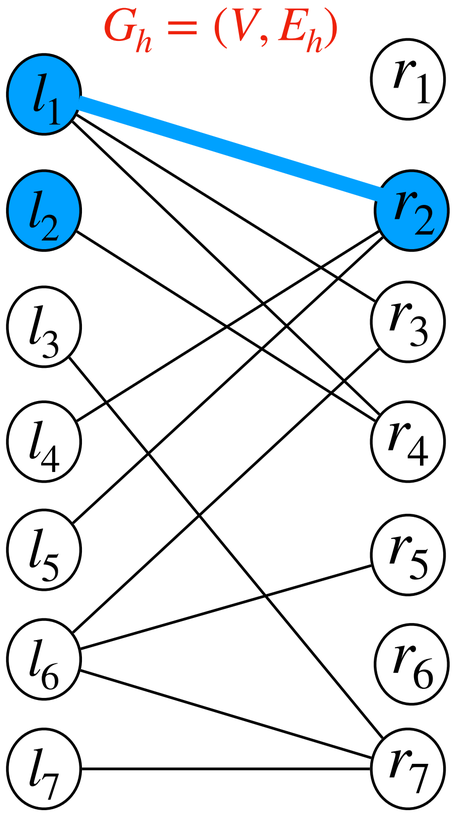


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

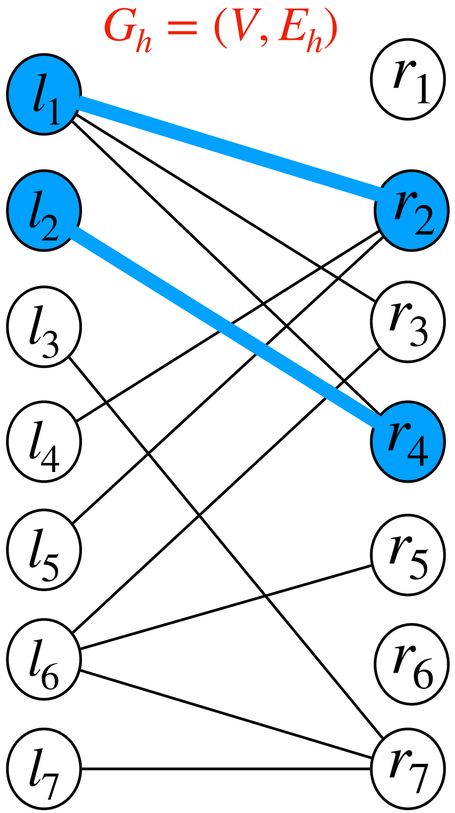


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

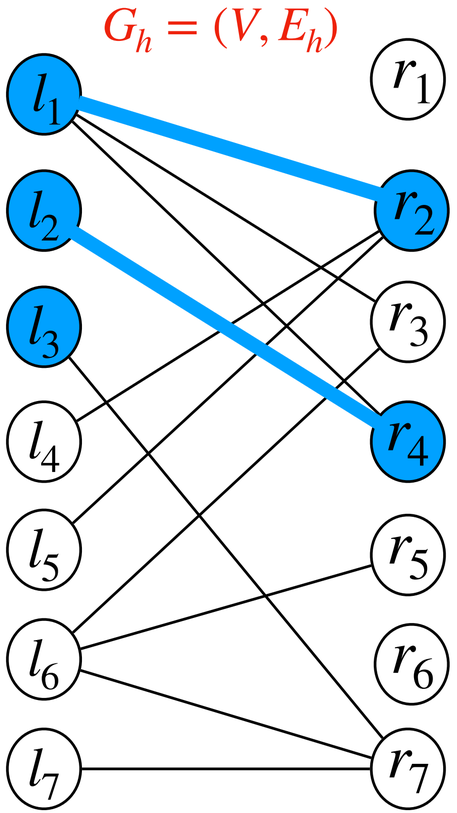


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

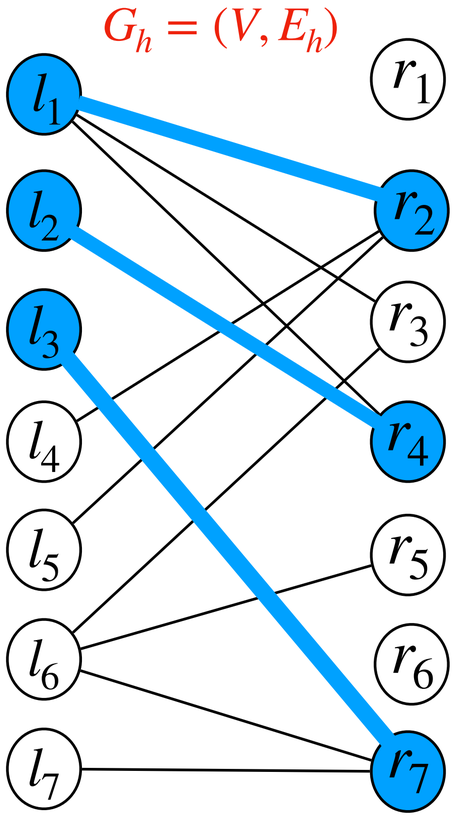


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

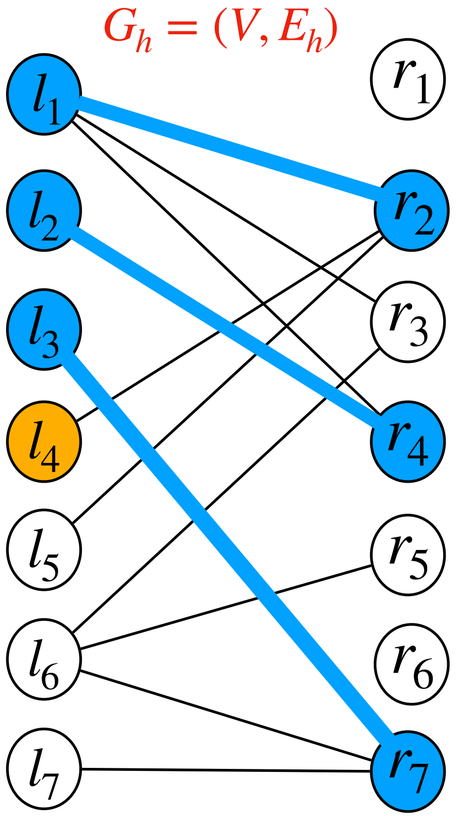


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

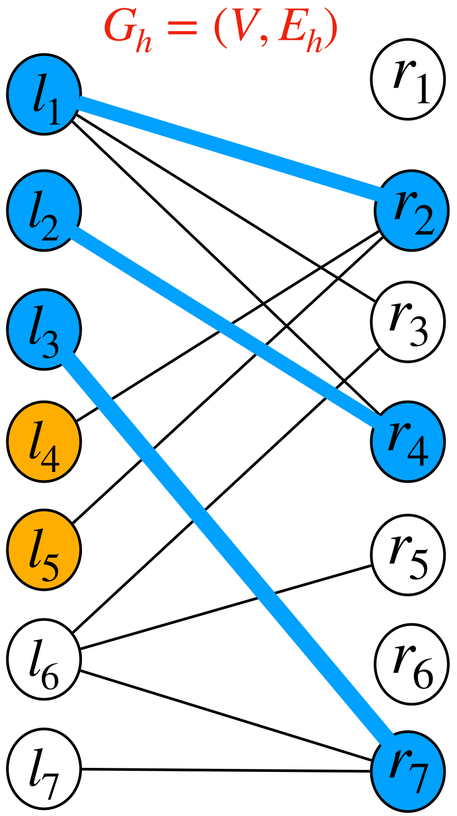


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

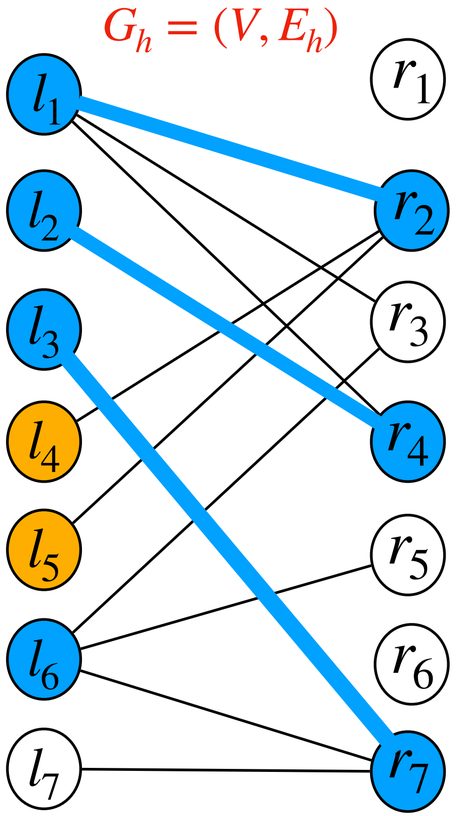


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

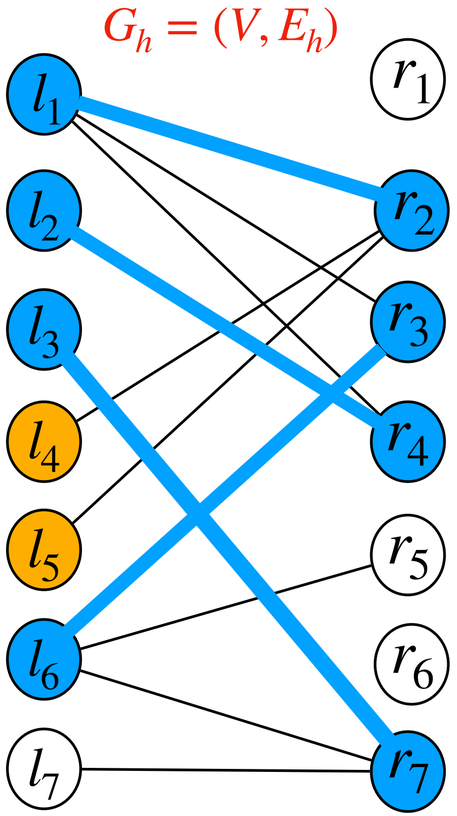


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

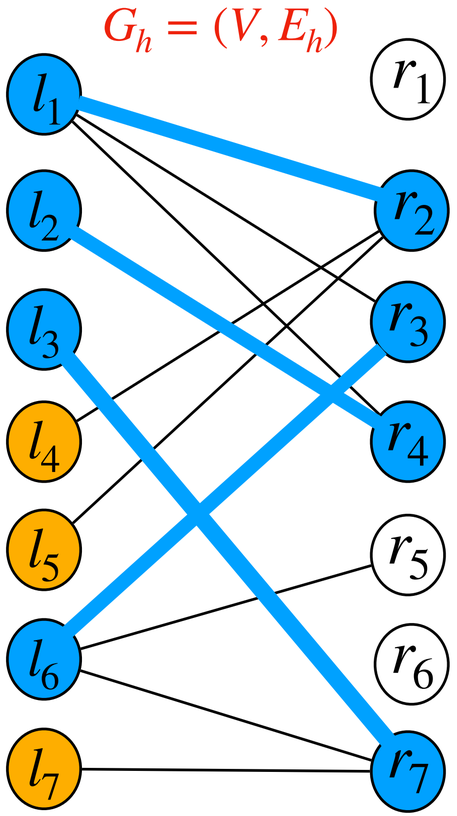


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

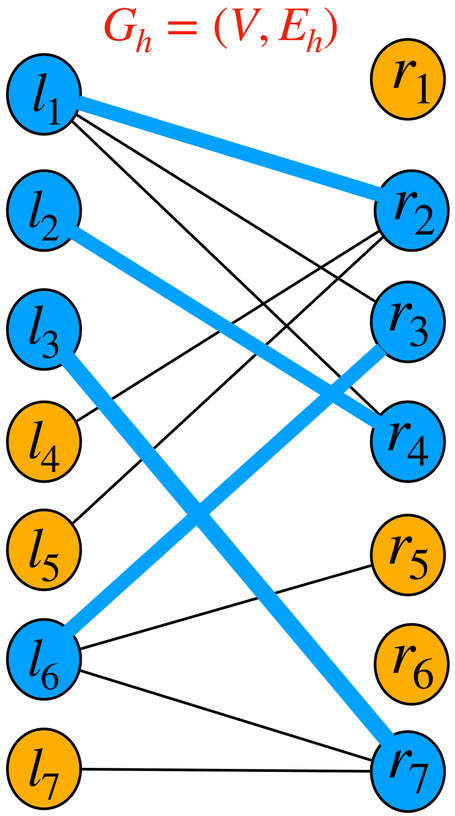


Greedy Maximal Bipartite Matching

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



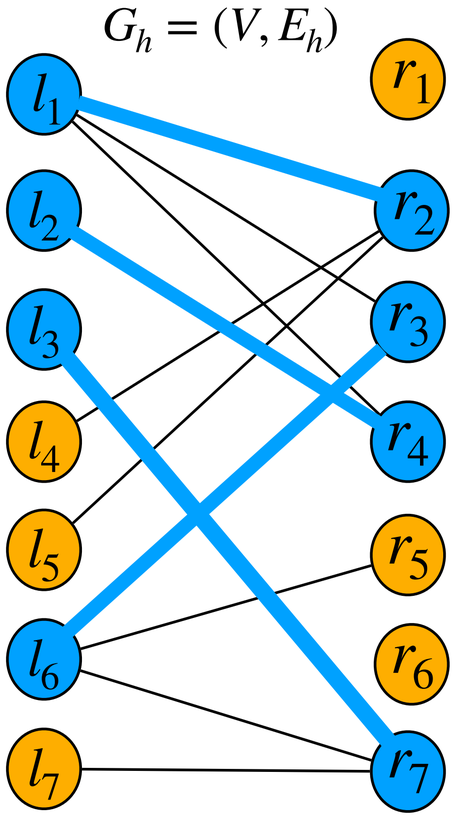
Greedy Maximal Bipartite Matching

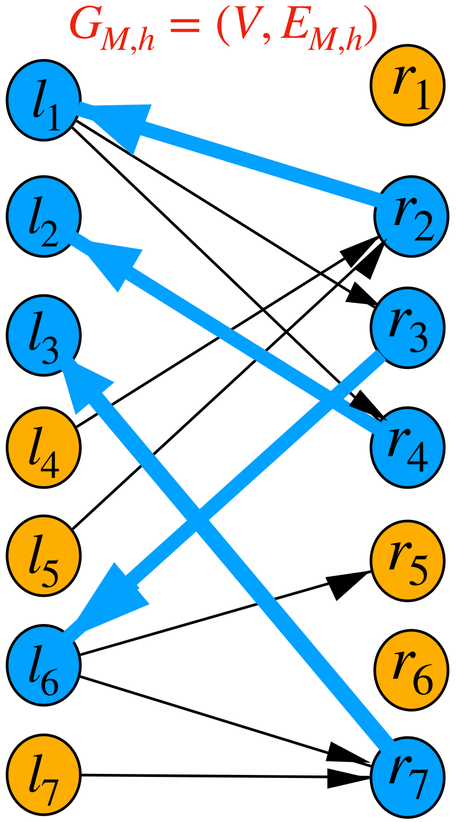
$M = \{(l_1, r_2),$
$(l_2, r_4),$
$(l_3, r_7),$
$(l_6, r_3)\}$

Greedy-Bipartite-Matching(G)

1. $M = \emptyset$
2. for each vertex $l \in L$
3. if l has an unmatched neighbor in R
4. choose any such unmatched neighbor $r \in R$
5. $M = M \cup \{(l, r)\}$
6. return M

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

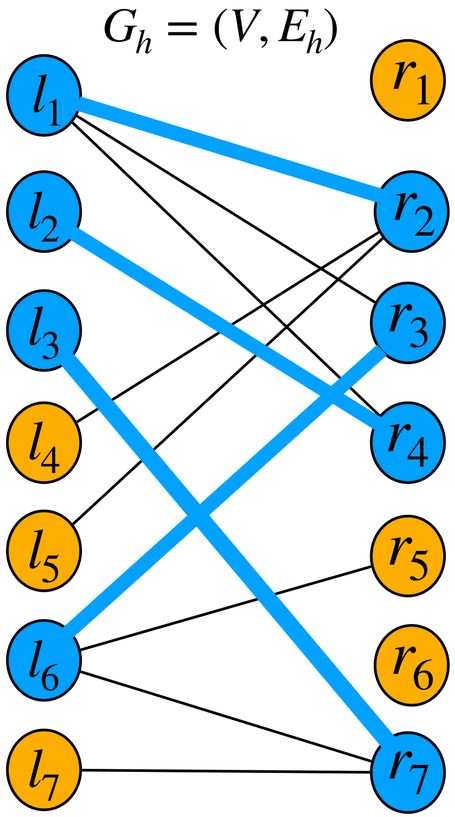
Similar to the Hopcroft-Karp algorithm.

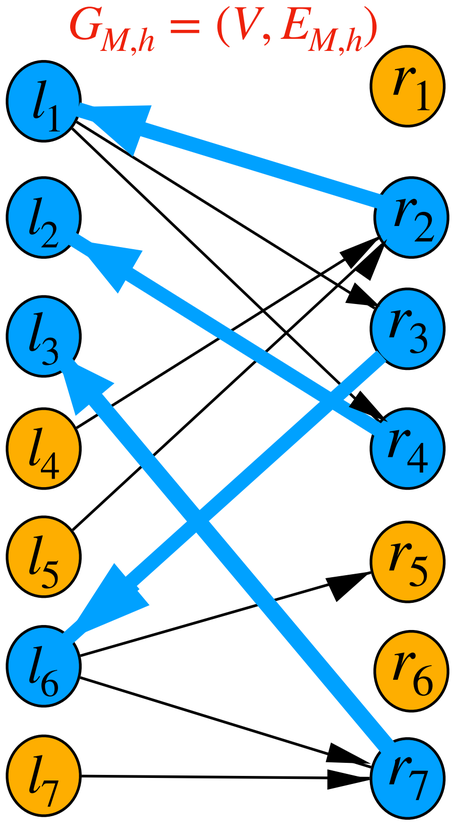
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

All the edges not in the matching M are from left to right.

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

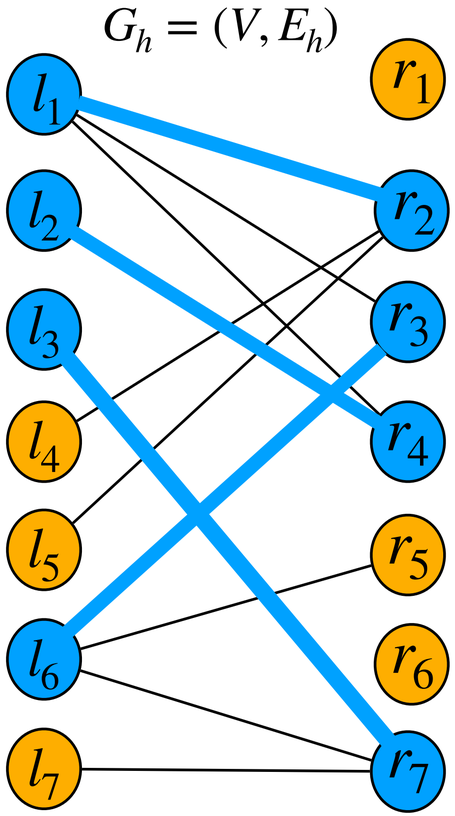
All the edges in the matching M are from right to left;

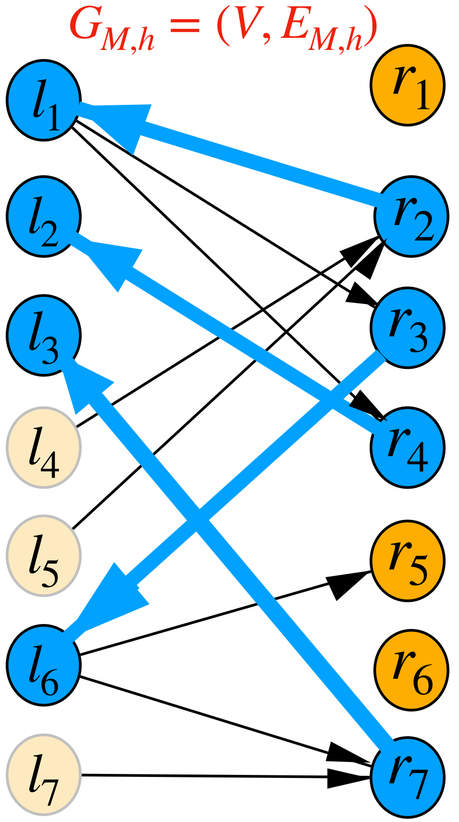
All the edges not in the matching M are from left to right.

Use BFS to find one M-augmenting path in $G_{M,h}$

not a maximal set of M-augmenting paths.

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

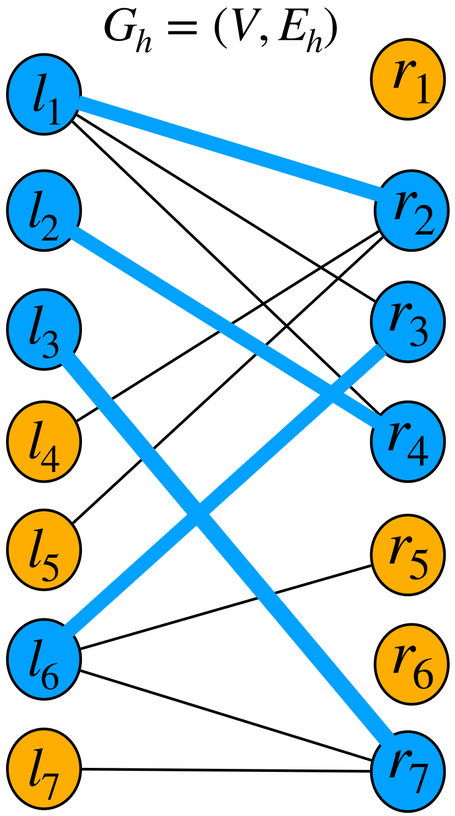
All the edges in the matching M are from right to left;

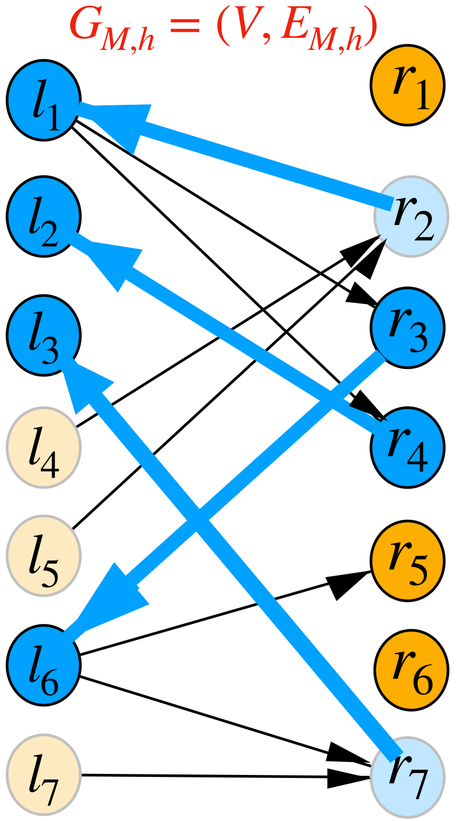
All the edges not in the matching M are from left to right.

Use BFS to find one M-augmenting path in $G_{M,h}$

build a breadth-first forest $F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

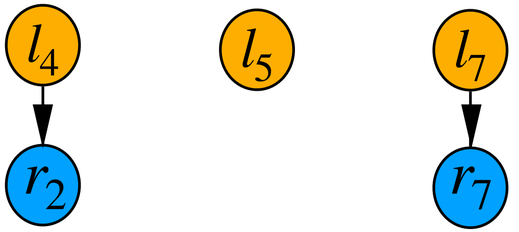
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

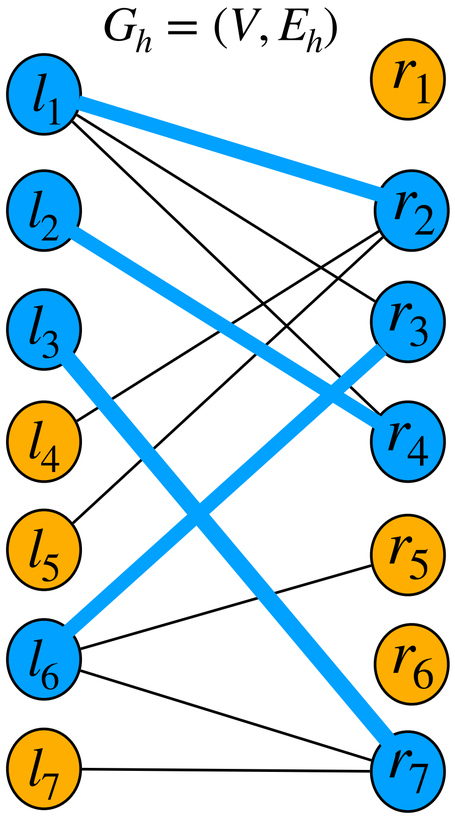
All the edges not in the matching M are from left to right.

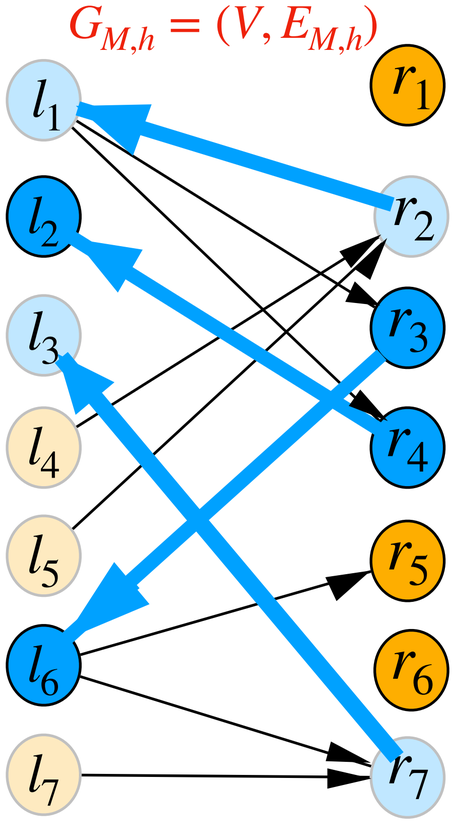
Use BFS to find one M-augmenting path in $G_{M,h}$

build a breadth-first forest $F = (V_F, E_F)$



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

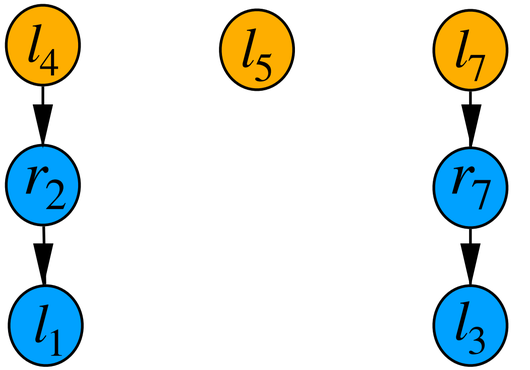
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

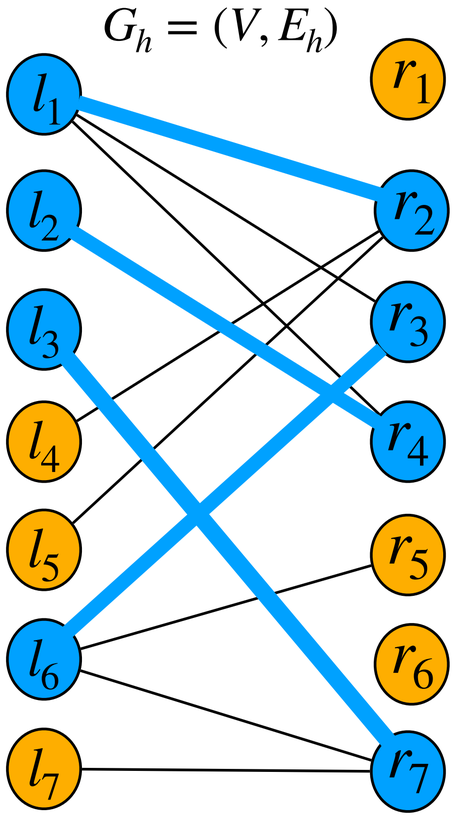
All the edges not in the matching M are from left to right.

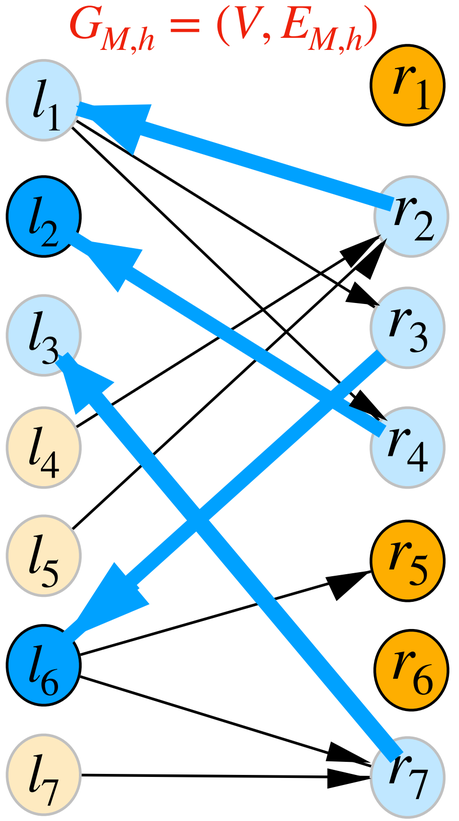
Use BFS to find one M-augmenting path in $G_{M,h}$

build a breadth-first forest $F = (V_F, E_F)$



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

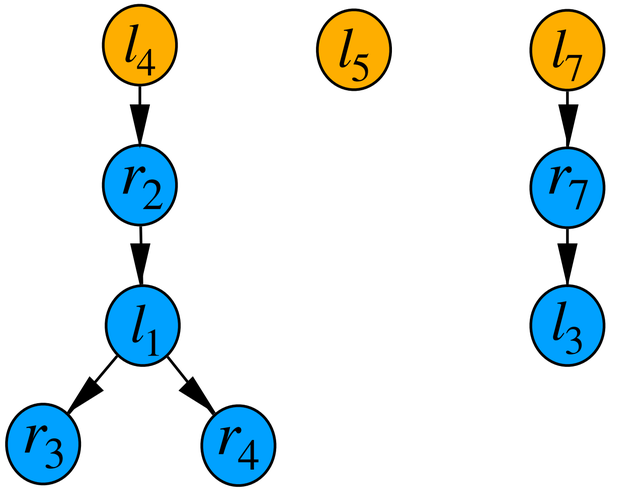
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

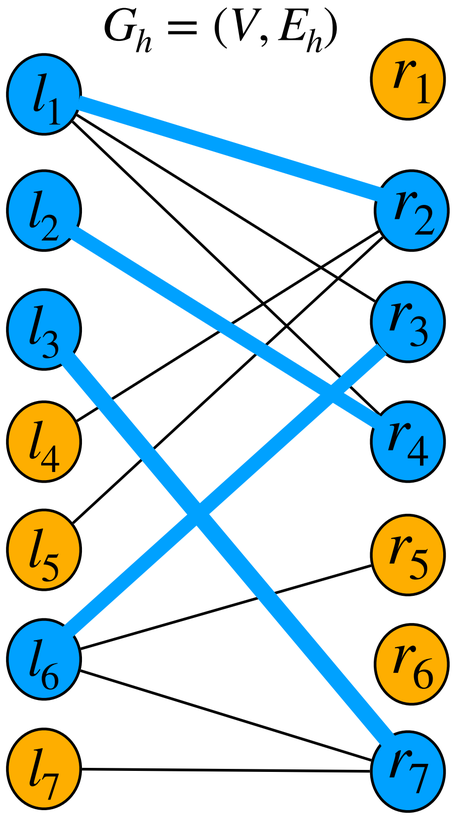
All the edges not in the matching M are from left to right.

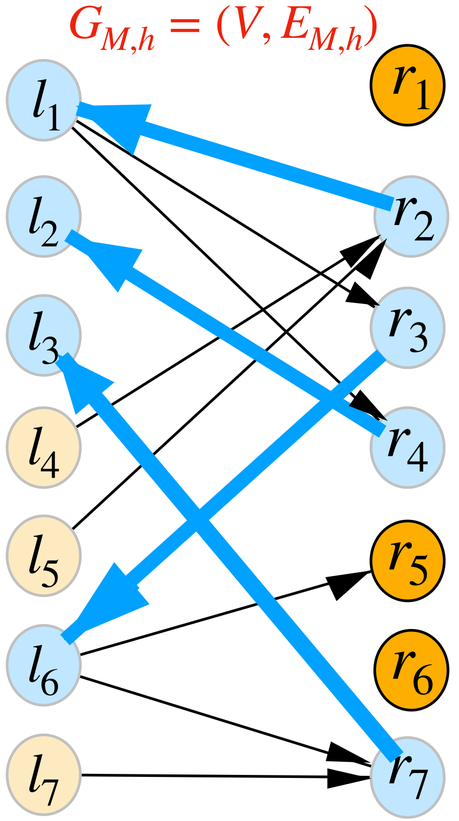
Use BFS to find one M-augmenting path in $G_{M,h}$

build a breadth-first forest $F = (V_F, E_F)$



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

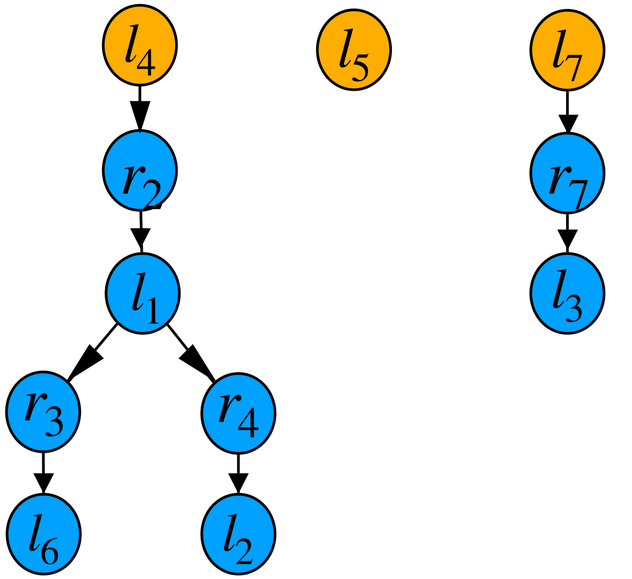
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

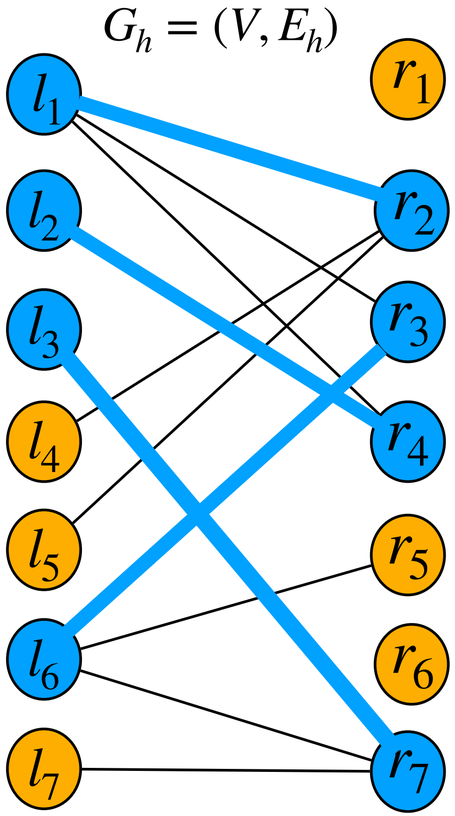
All the edges not in the matching M are from left to right.

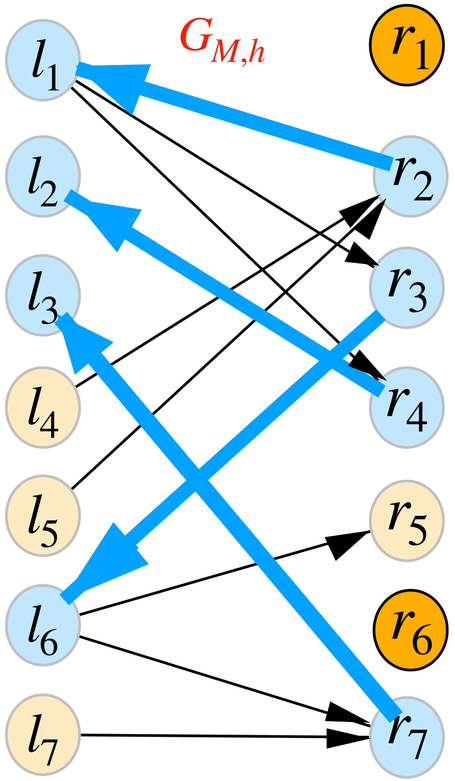
Use BFS to find one M-augmenting path in $G_{M,h}$

build a breadth-first forest $F = (V_F, E_F)$



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

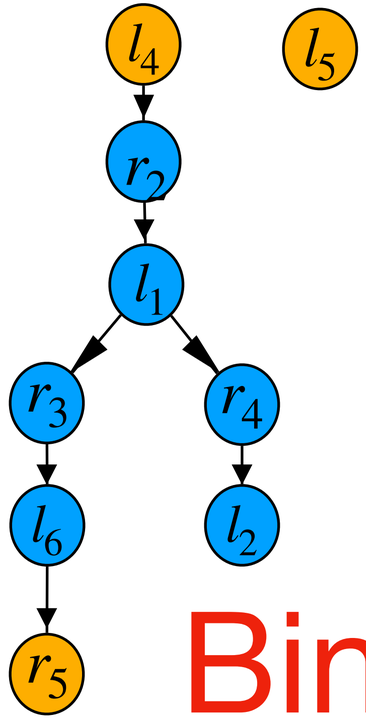
Similar to the Hopcroft-Karp algorithm.

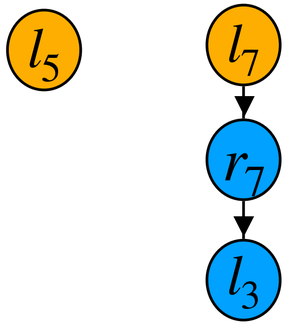
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

All the edges not in the matching M are from left to right.

Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$

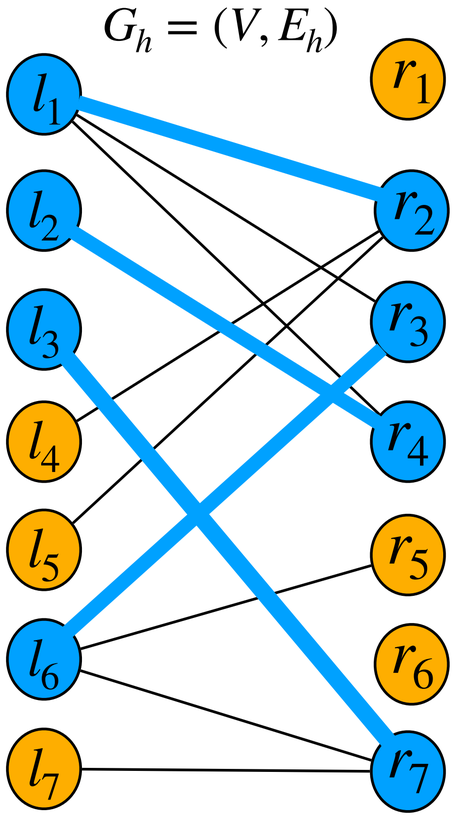


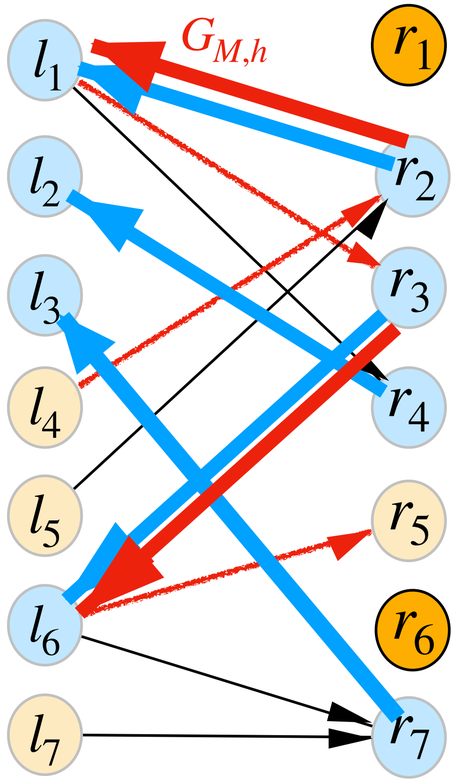


$F = (V_F, E_F)$

Bingo!

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

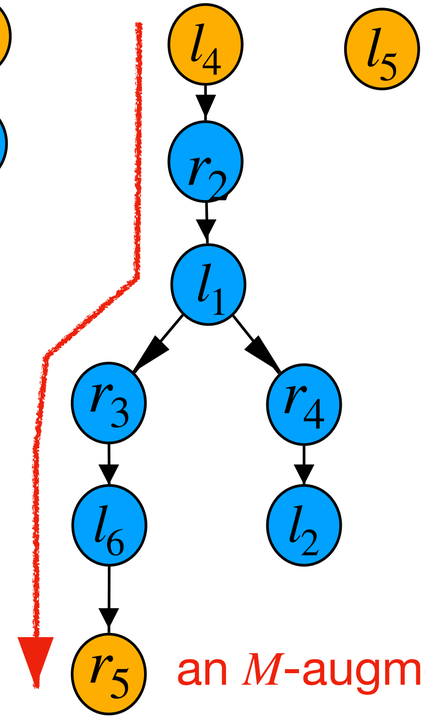
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

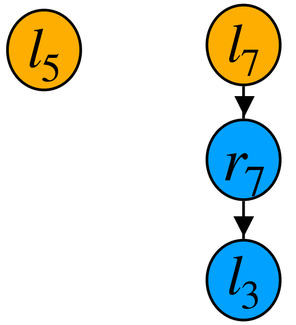
All the edges in the matching M are from right to left;

All the edges not in the matching M are from left to right.

Use BFS to find one M-augmenting path in $G_{M,h}$

build a breadth-first forest $F = (V_F, E_F)$



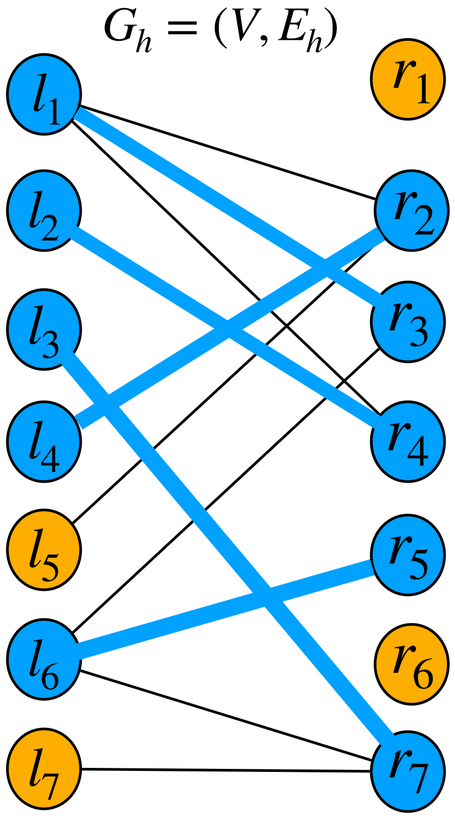


$F = (V_F, E_F)$

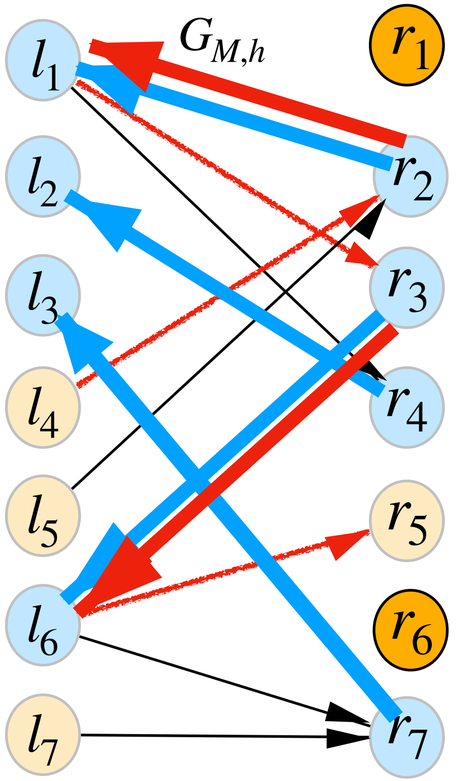
an M-augmenting path P

Update the matching to $M \oplus P$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



$M \oplus P$



Find an M-augmenting path in G_h

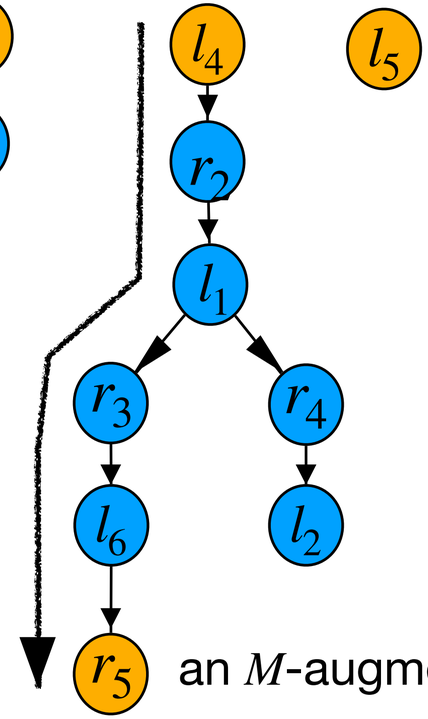
Similar to the Hopcroft-Karp algorithm.

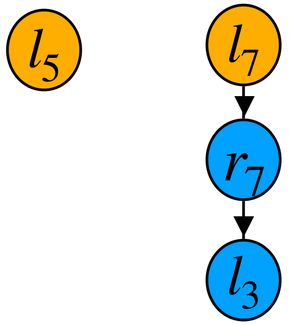
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

All the edges in the matching M are from right to left;

All the edges not in the matching M are from left to right.

Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$



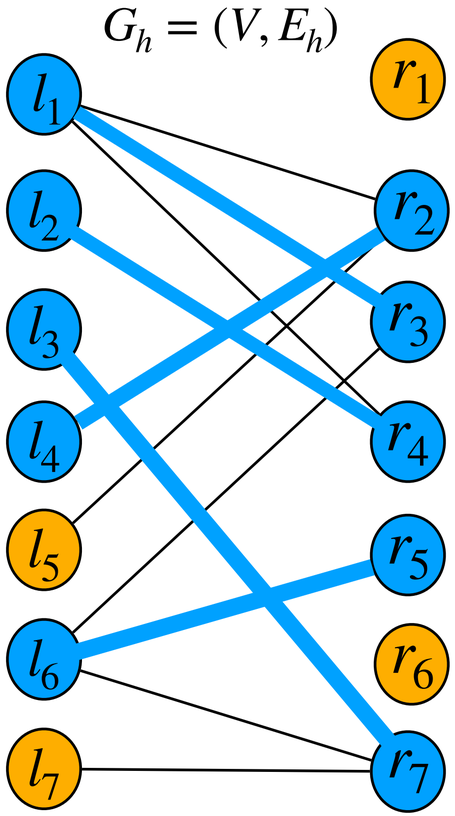


$F = (V_F, E_F)$

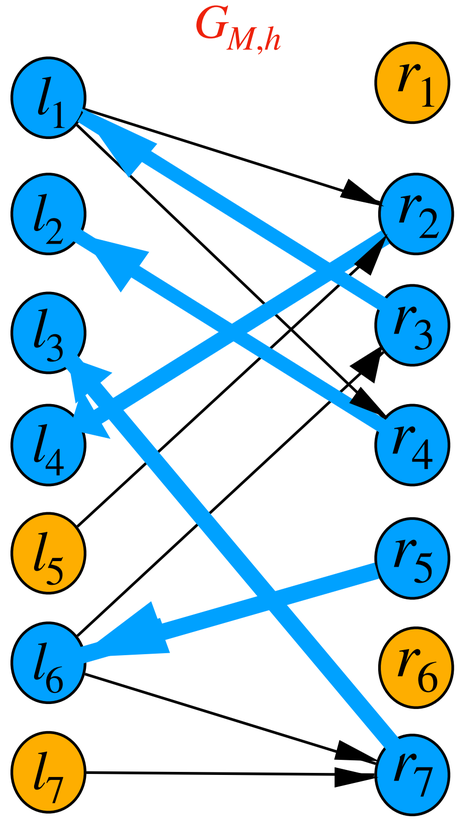
an M-augmenting path P

Update the matching to $M \oplus P$

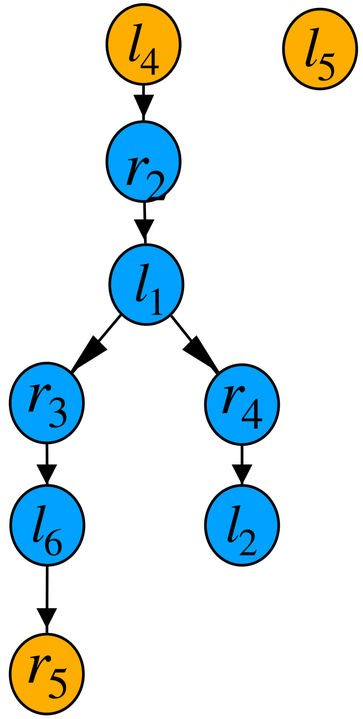
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

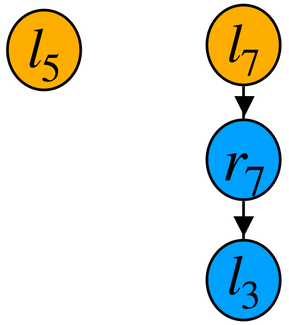


$M \oplus P$



Update $G_{M,h} = (V, E_{M,h})$





$F = (V_F, E_F)$

Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

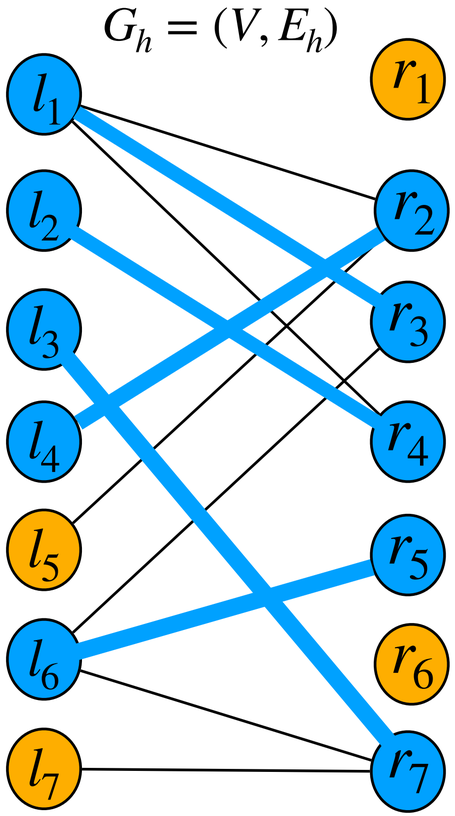
Turn G_h into a directed graph $G_{M,h} = (V, E_{M,h})$:

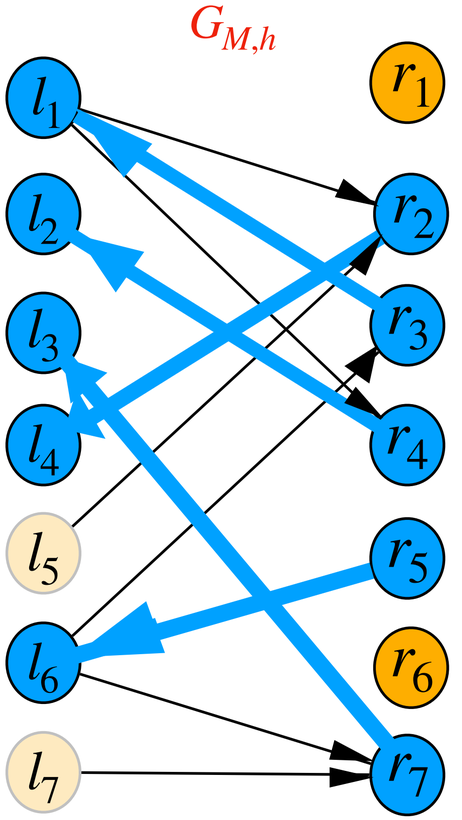
All the edges in the matching M are from right to left;

All the edges not in the matching M are from left to right.

Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





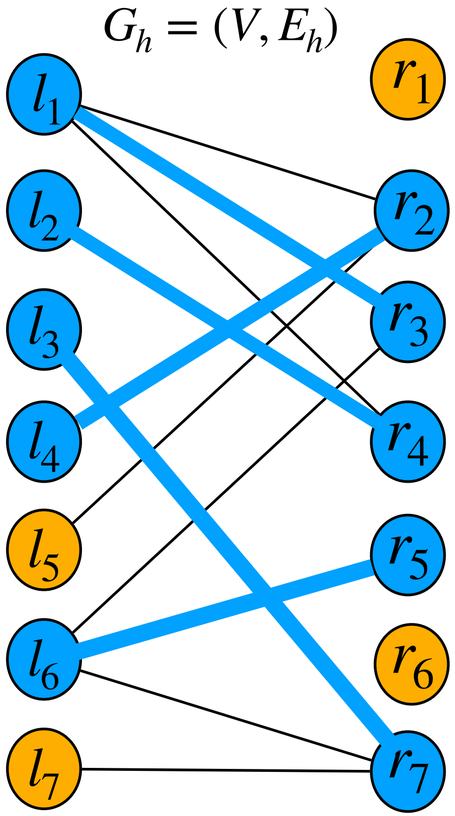
Find an M-augmenting path in G_h

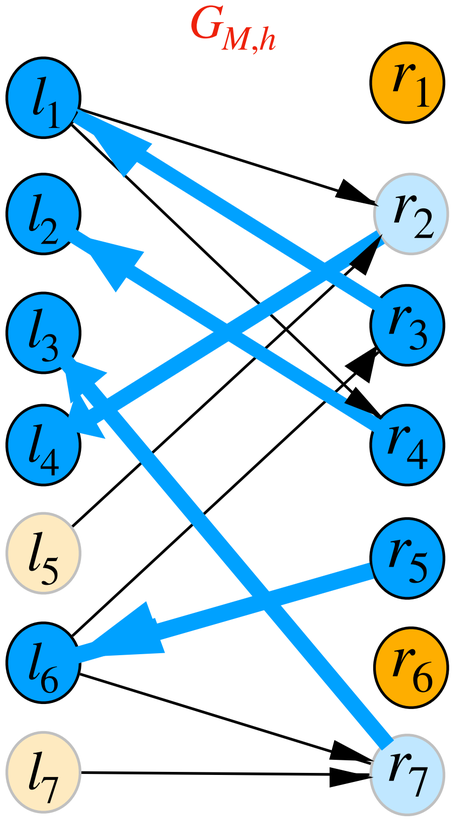
Similar to the Hopcroft-Karp algorithm.

Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$

l_5 l_7

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

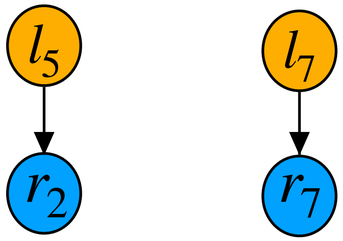




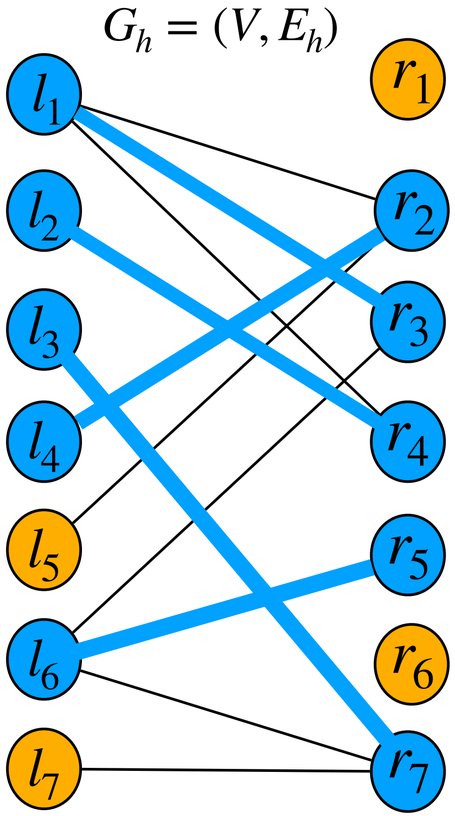
Find an M-augmenting path in G_h

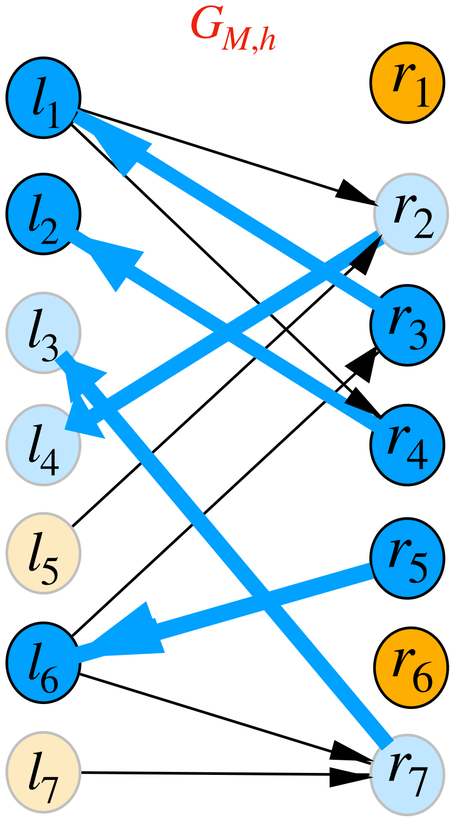
Similar to the Hopcroft-Karp algorithm.

Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

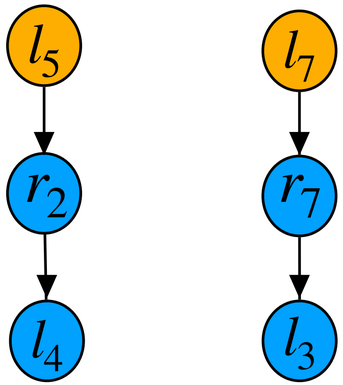




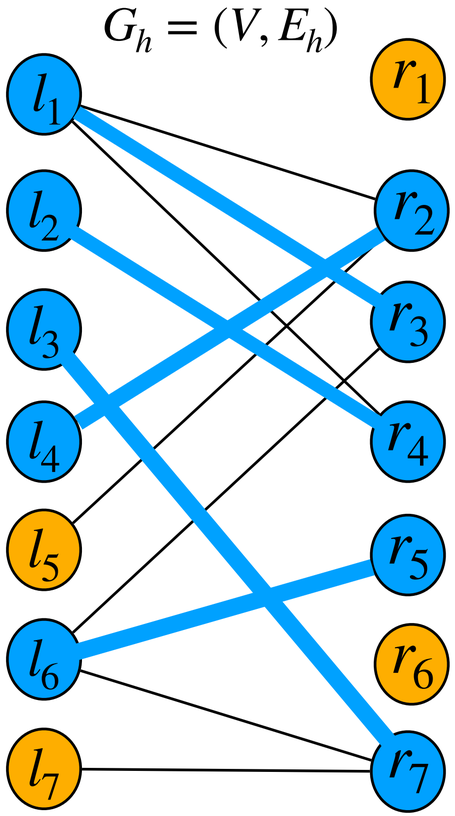
Find an M-augmenting path in G_h

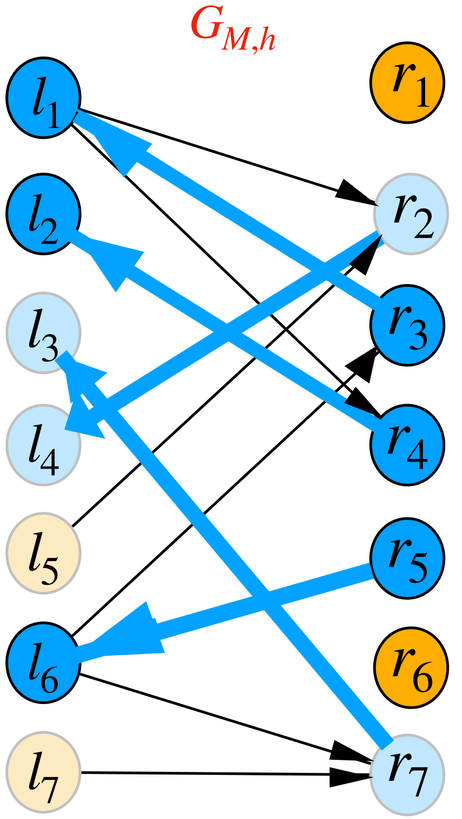
Similar to the Hopcroft-Karp algorithm.

Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



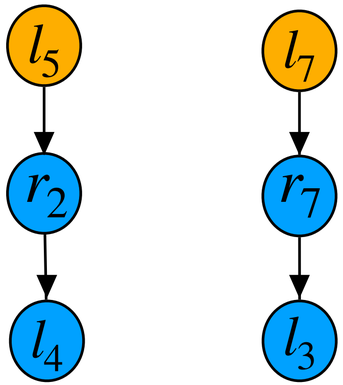


Find an M-augmenting path in G_h

Similar to the Hopcroft-Karp algorithm.

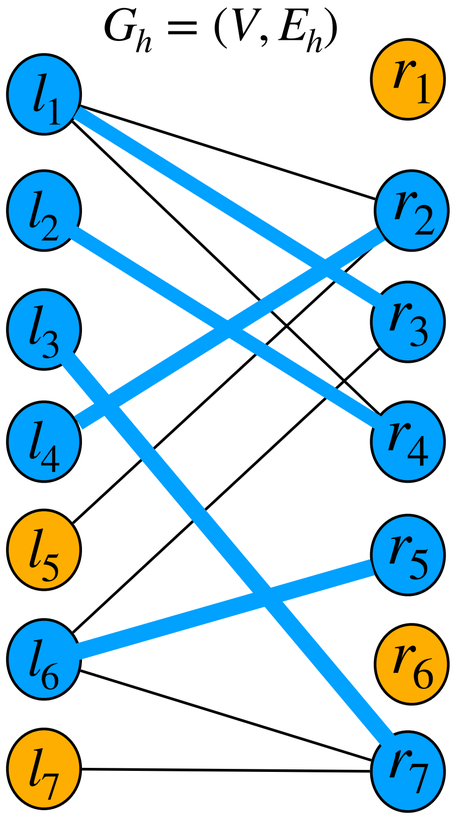
Use BFS to find one M-augmenting path in $G_{M,h}$
build a breadth-first forest $F = (V_F, E_F)$

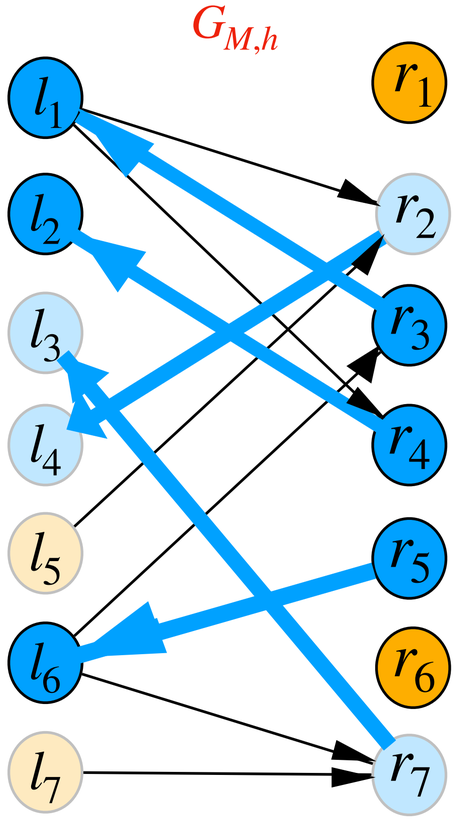
The search for an M-augmenting path failed!



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

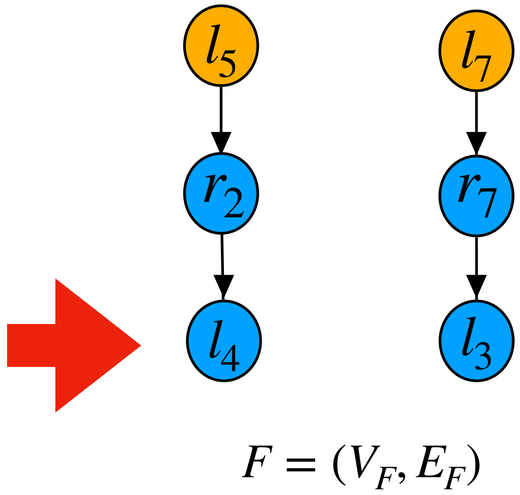




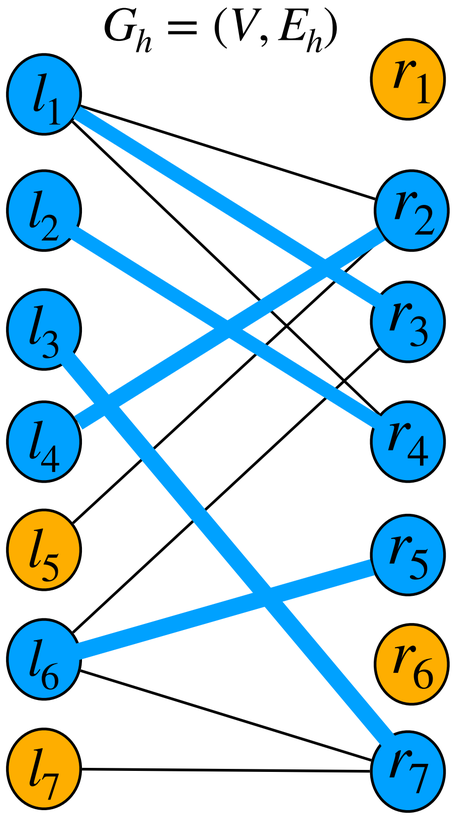
When the search for an M-augmenting path fails

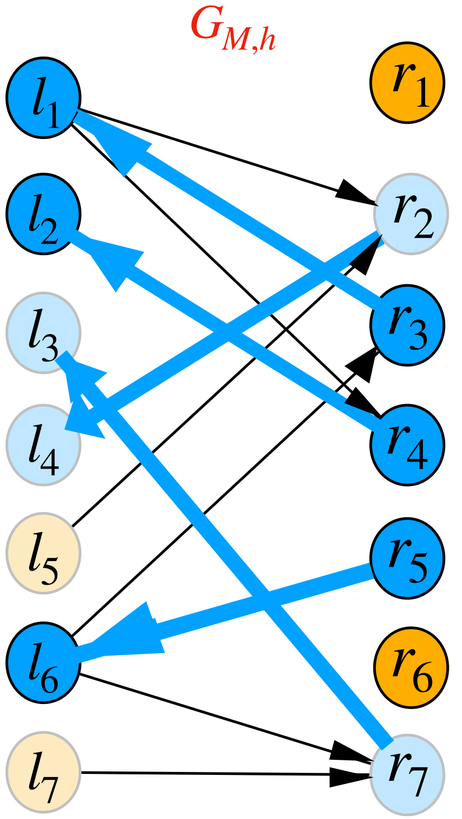
The last layer of vertices must be vertices on the left side.

Why?



		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



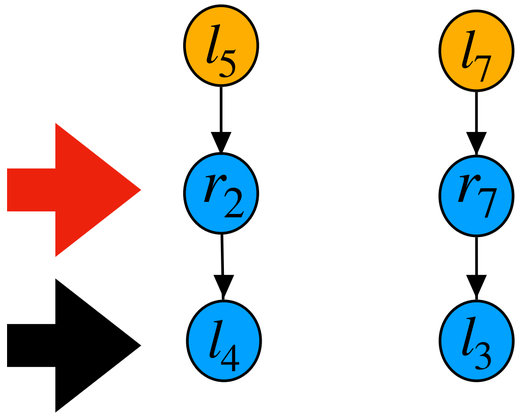


When the search for an M-augmenting path fails

For matched vertices on the right side,
they lead to matched vertices on the left side.

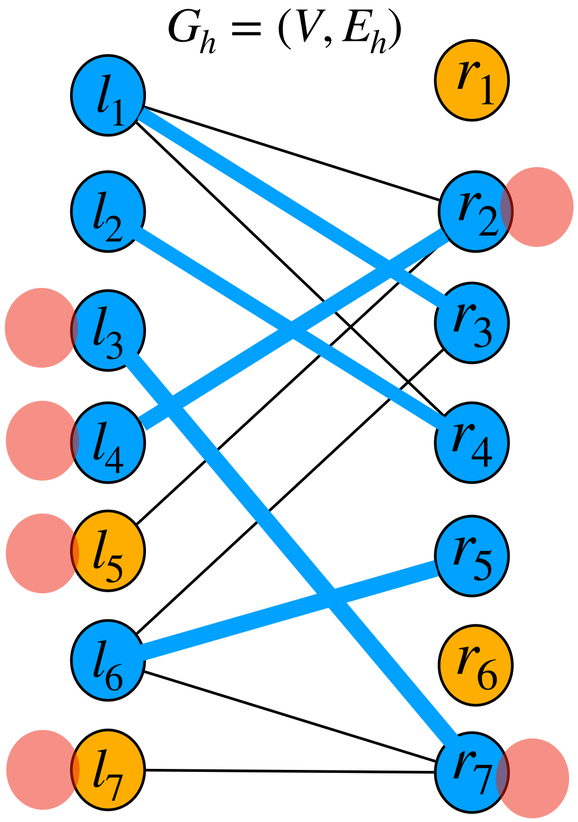
The last layer of vertices must be vertices on the left side.

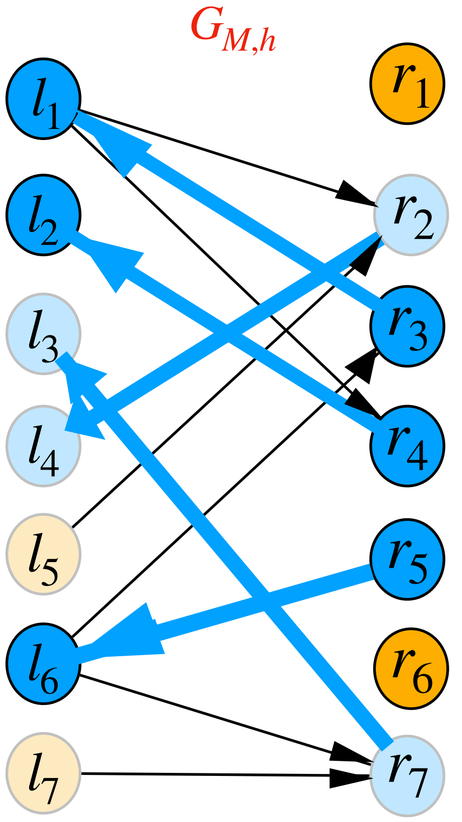
Why?



$F = (V_F, E_F)$

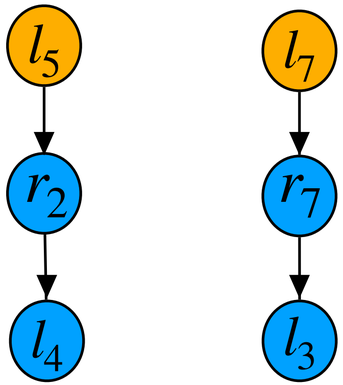
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





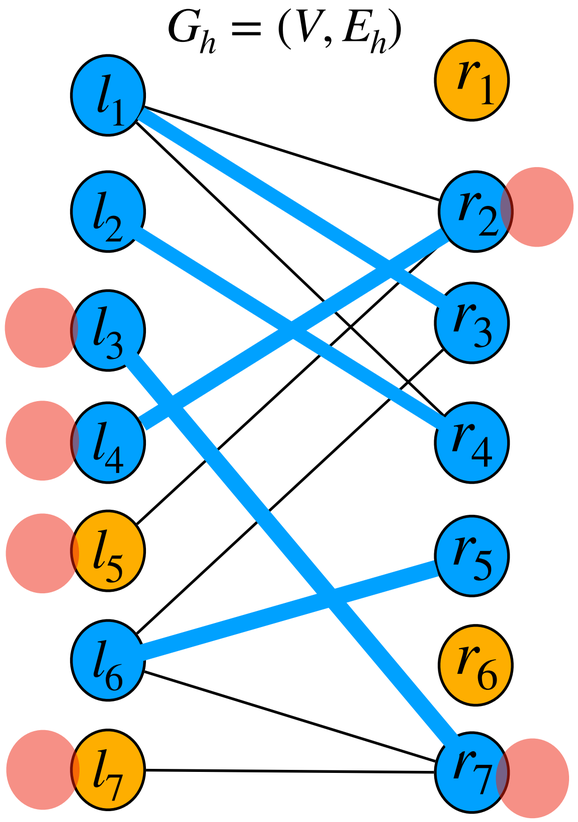
When the search for an M-augmenting path fails

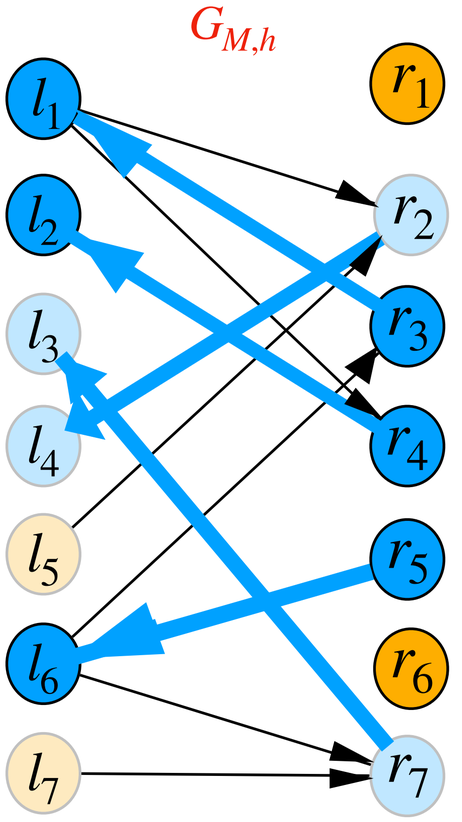
$F_L = L \cap V_F$ left vertices in BFS forest



$F = (V_F, E_F)$

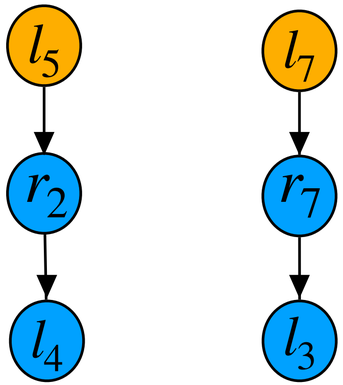
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





When the search for an M-augmenting path fails

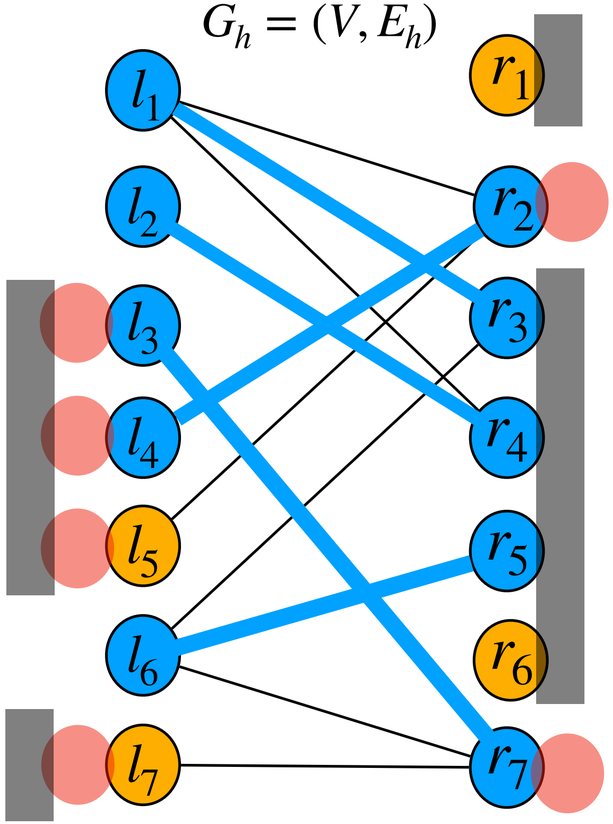
$F_L = L \cap V_F$ $F_R = R \cap V_F$ right vertices in BFS forest



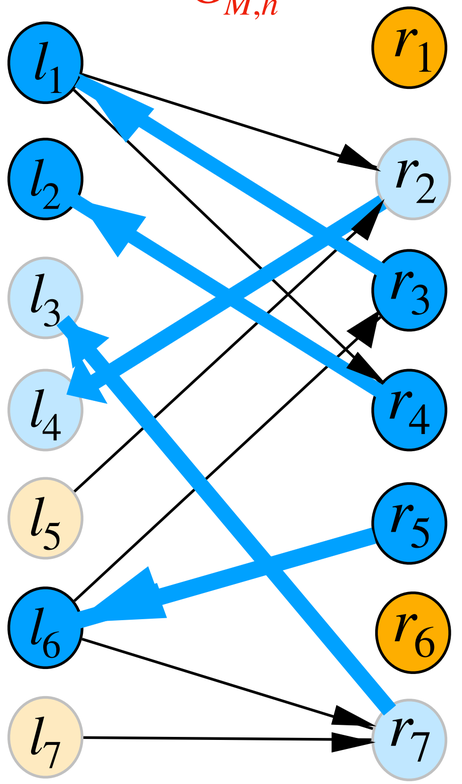
$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8

$G_h = (V, E_h)$



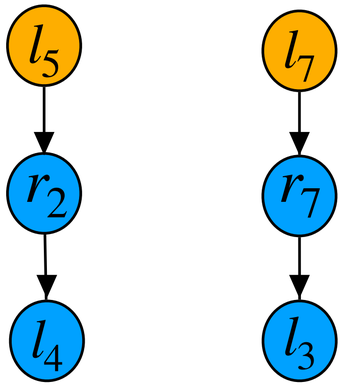
$G_{M,h}$



When the search for an M-augmenting path fails

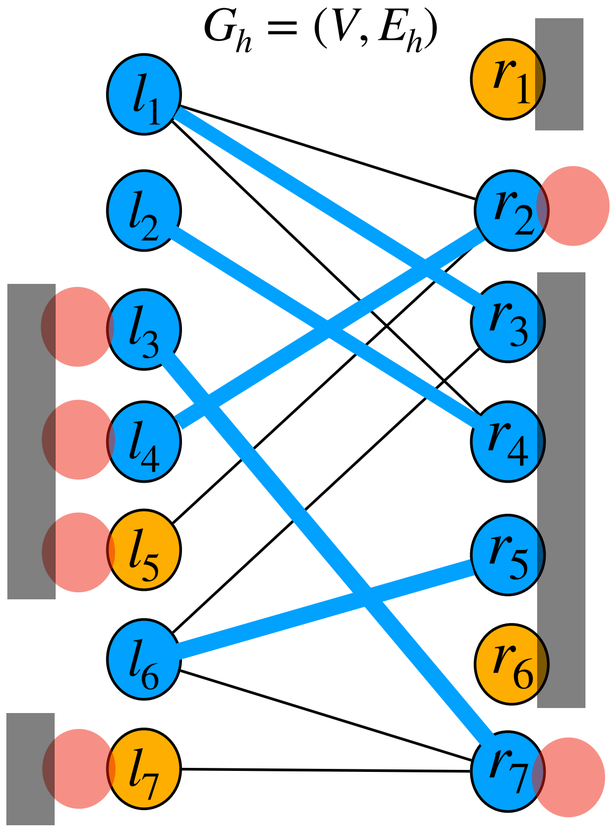
$F_L = L \cap V_F \qquad F_R = R \cap V_F$

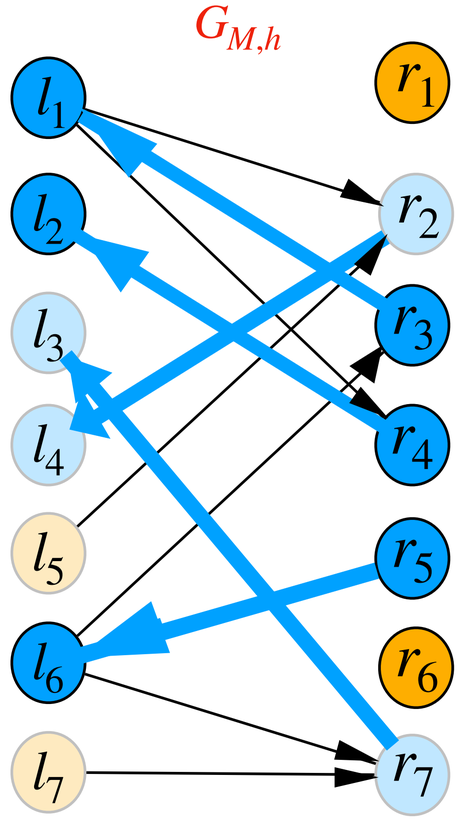
$\delta = \min\{l.h + r.h - w(l,r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8



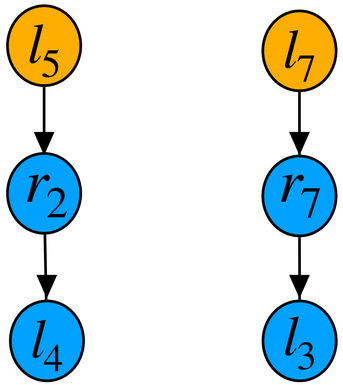


When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad F_R = R \cap V_F$

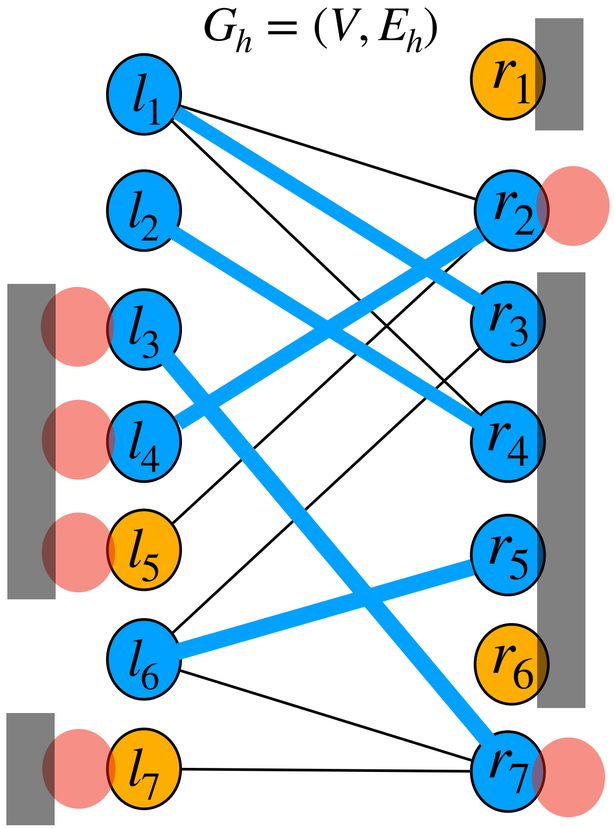
$\delta = \min\{l.h + r.h - w(l,r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

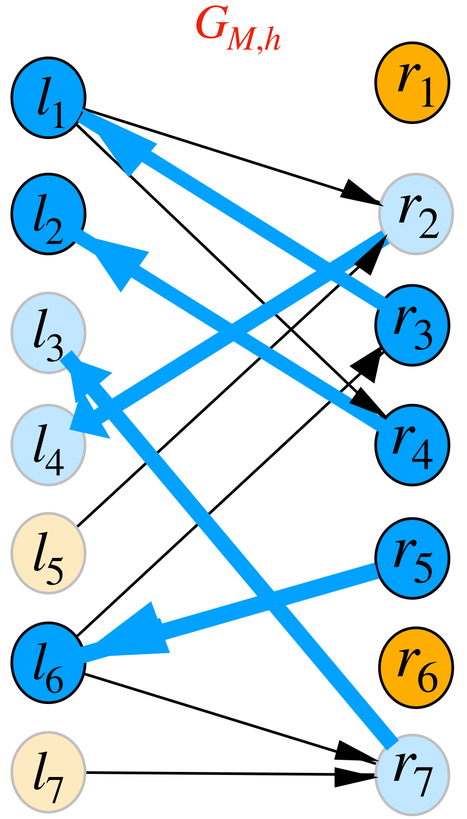
minimum "gap" between edge weight and labeling



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





When the search for an M-augmenting path fails

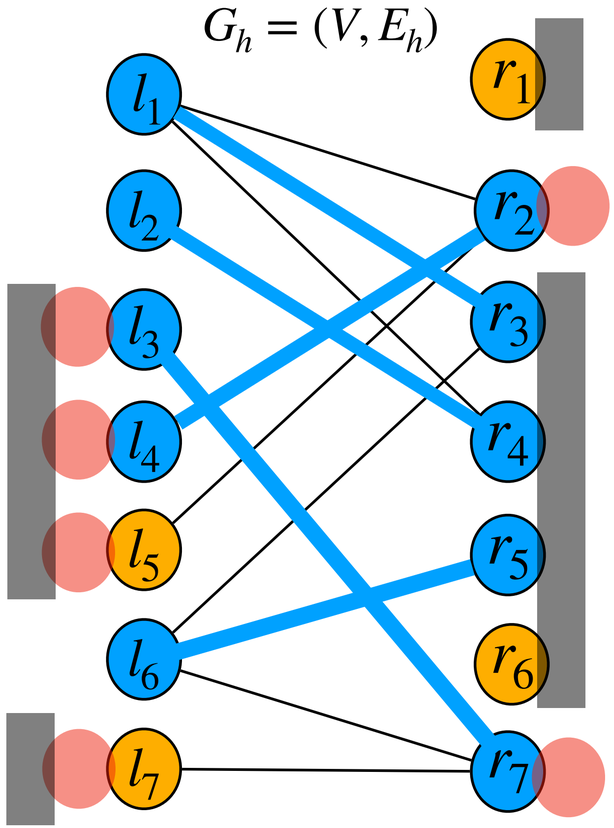
$F_L = L \cap V_F$ $\quad$ $F_R = R \cap V_F$

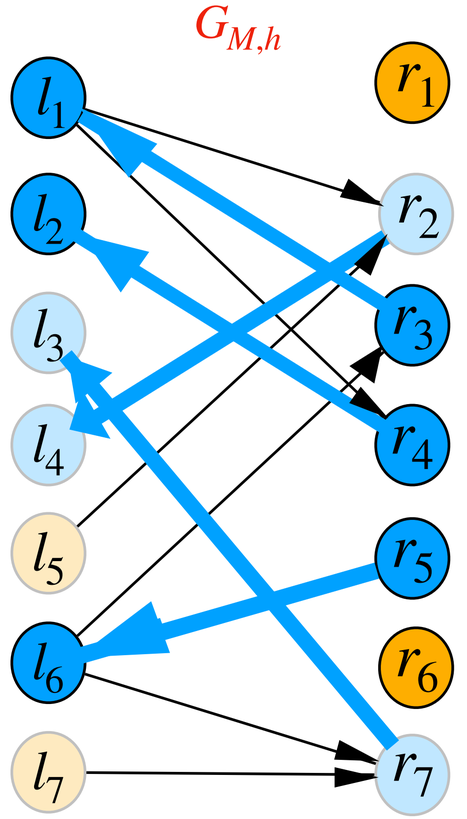
$\delta = \min\{l.h + r.h - w(l,r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

minimum "gap" between edge weight and labeling

Note: $\delta > 0$, because otherwise the edge causing δ would have been in G_h and its right endpoint would have been in F (whether it is matched or not, and whether the edge is in the match or not).

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0	0	0	0	0	0
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15	11	9	6	7	9	5	15
l_4	9	3	9	6	7	5	6	3
l_5	6	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8	3	4	5	4	3	6	8





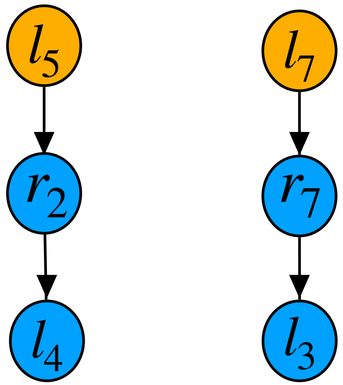
When the search for an M-augmenting path fails

$F_L = L \cap V_F \qquad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l,r) : l \in F_L \text{ and } r \in R - F_R\} > 0$

Update the vertex labeling:

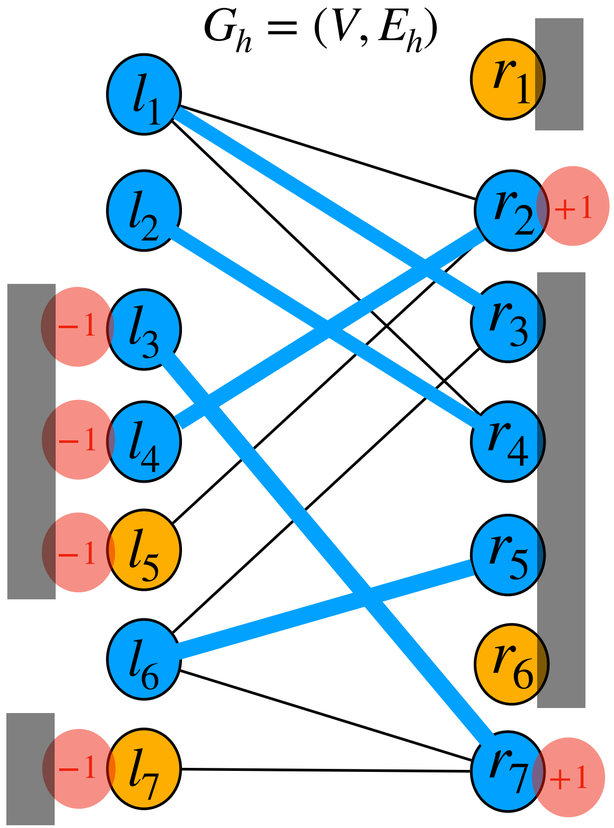
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$

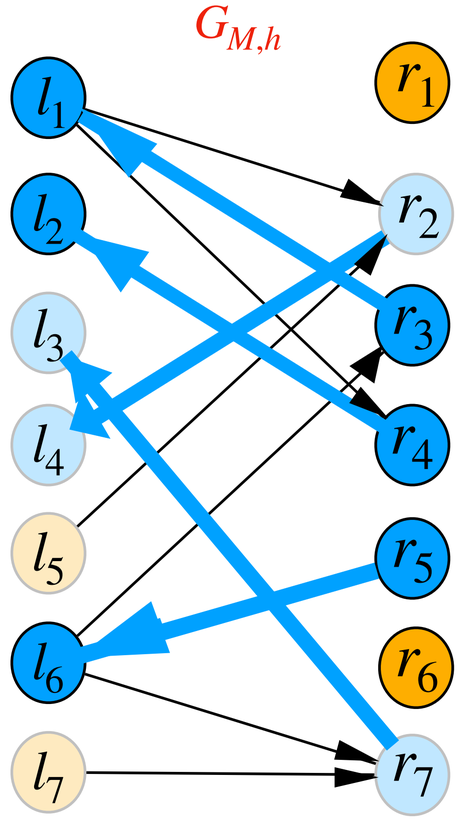


$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$





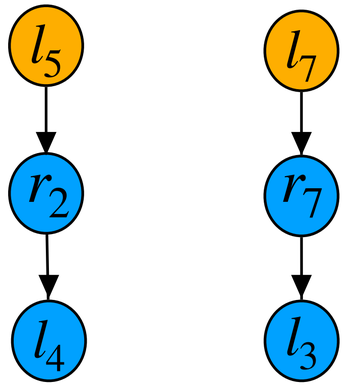
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad$ $F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l,r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

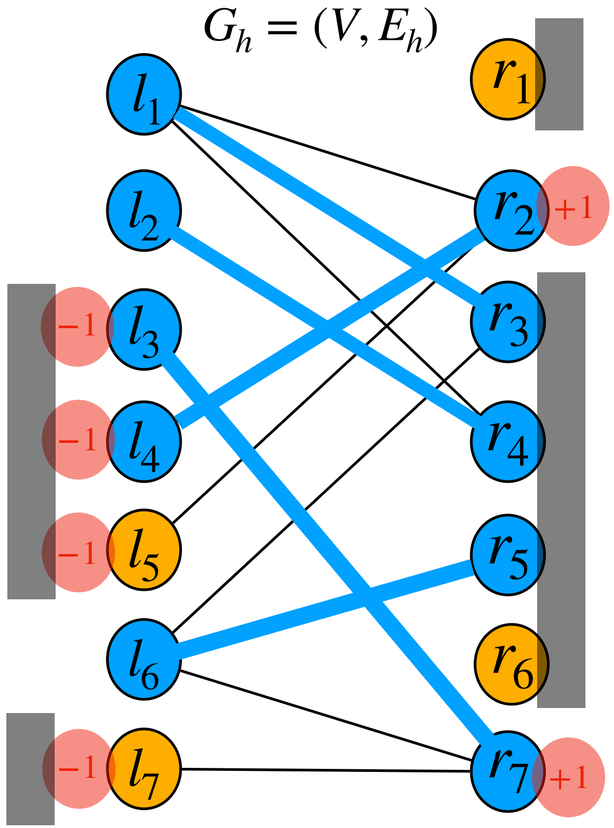
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$

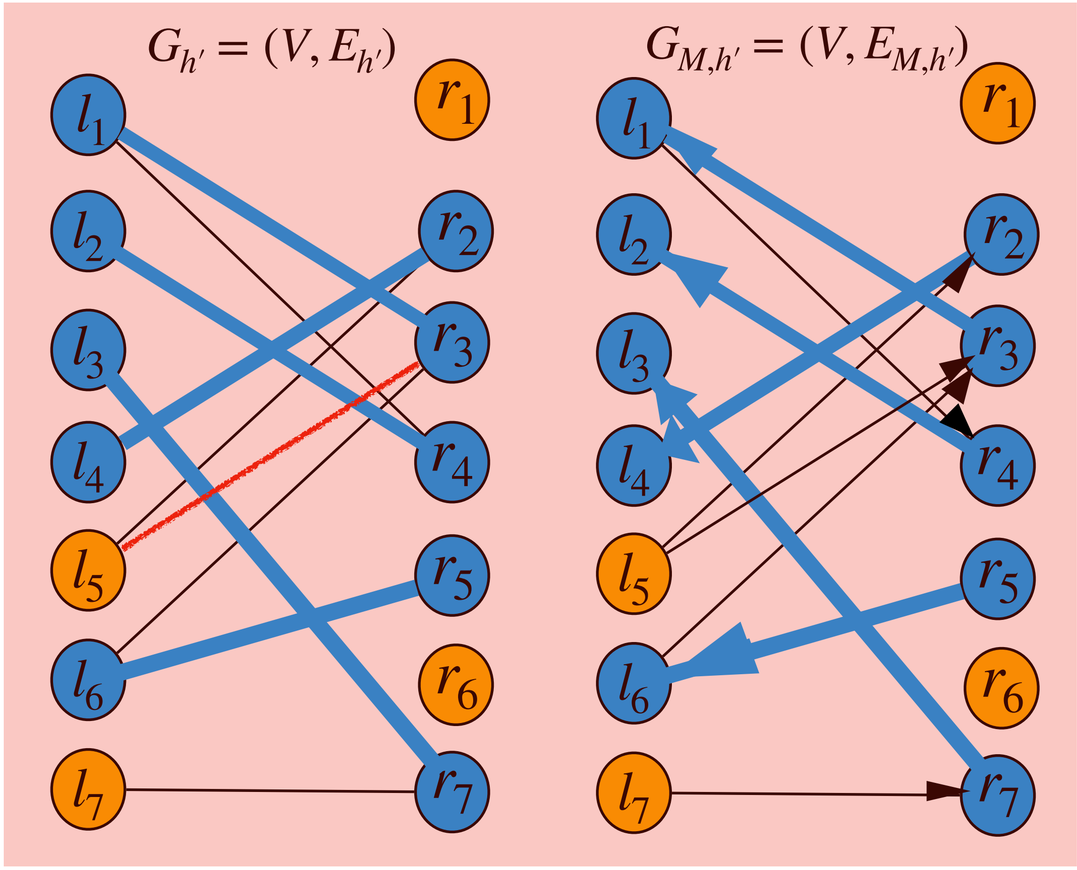


$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$$\delta = 1$$





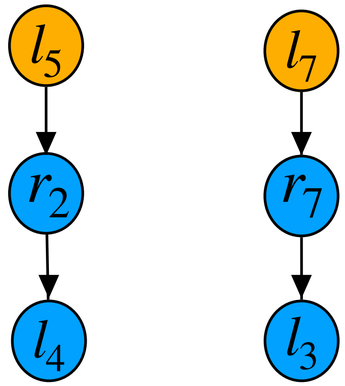
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad$ $F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l,r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

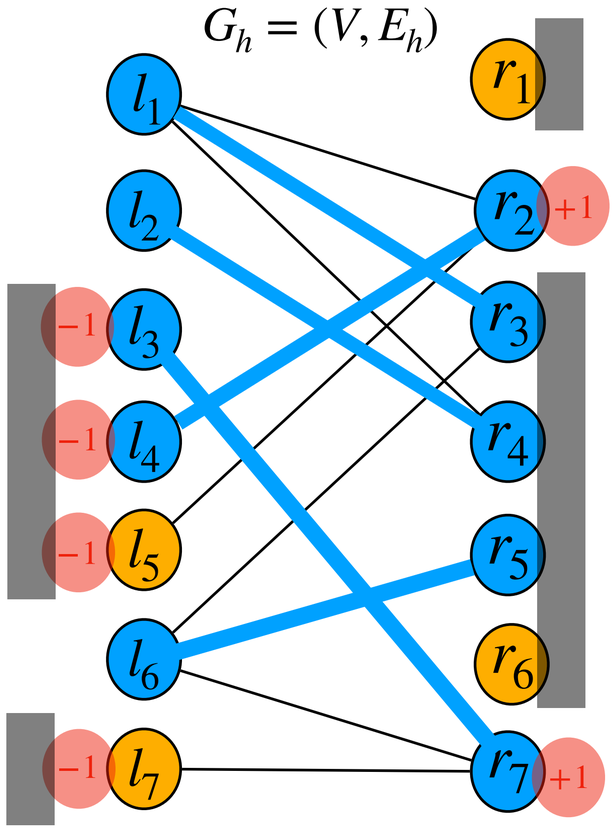
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



Lemma:

1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

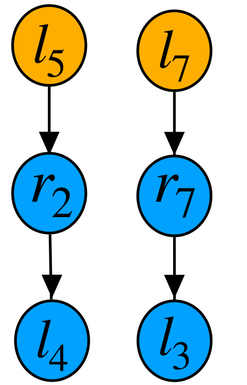
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l, r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

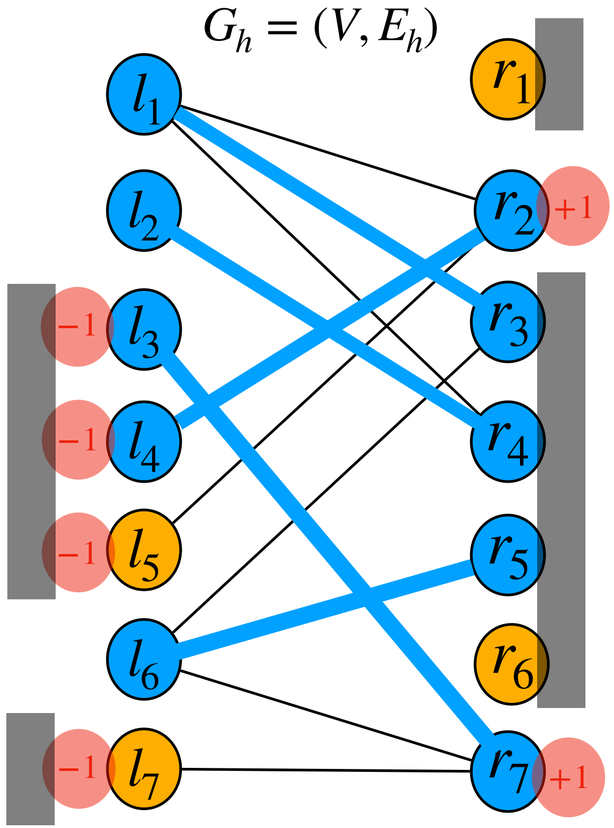
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



Lemma:

1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof: $\forall (l, r),\ l.h + r.h \geq w(l, r)$.

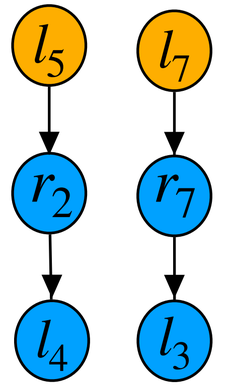
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l, r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

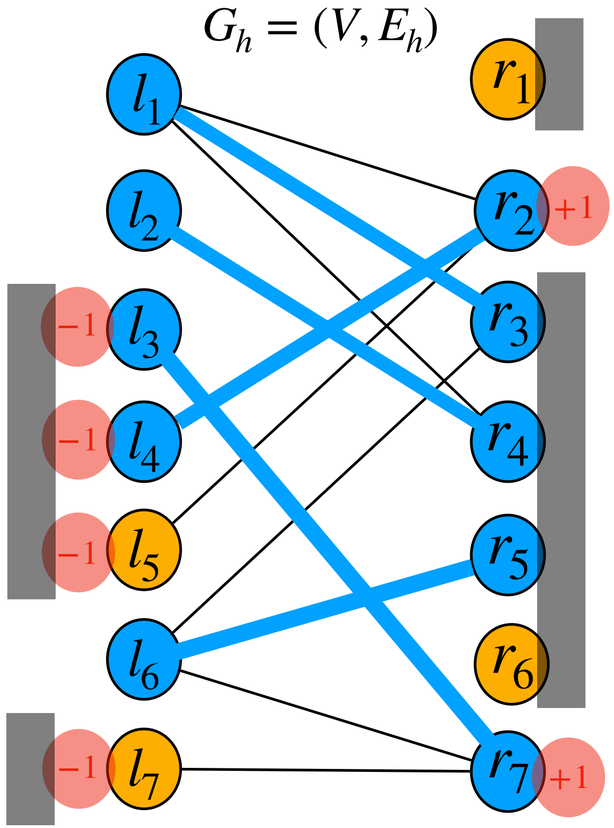
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0+1	0	0	0	0	0+1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



Lemma:

1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof: $\forall (l, r),\ \ l.h + r.h \geq w(l, r)$.

If $l.h' + r.h' < w(l, r)$

→ $l \in F_L$ and $r \in R - F_R$

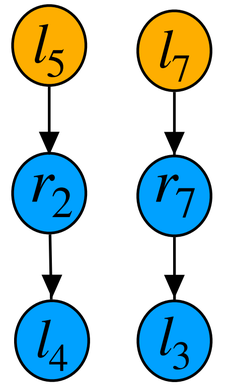
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\qquad$ $F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l, r) \ :\ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

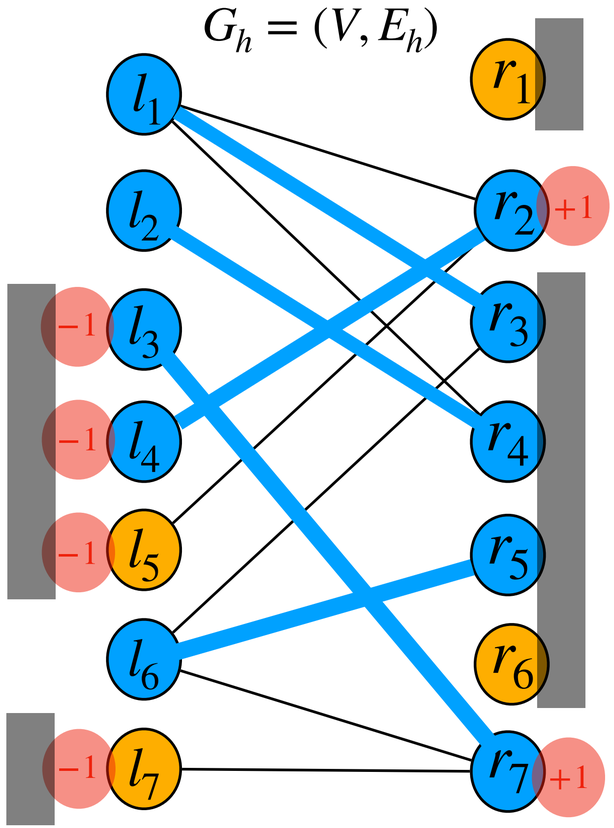
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



Lemma:

1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof: $\forall (l, r),\ \ l.h + r.h \geq w(l, r)$.

If $l.h' + r.h' < w(l, r)$

→ $l \in F_L$ and $r \in R - F_R$

→ contradiction by definition of δ

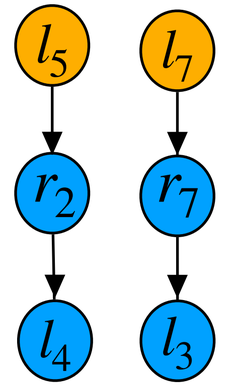
When the search for an M-augmenting path fails

$F_L = L \cap V_F \qquad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l, r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

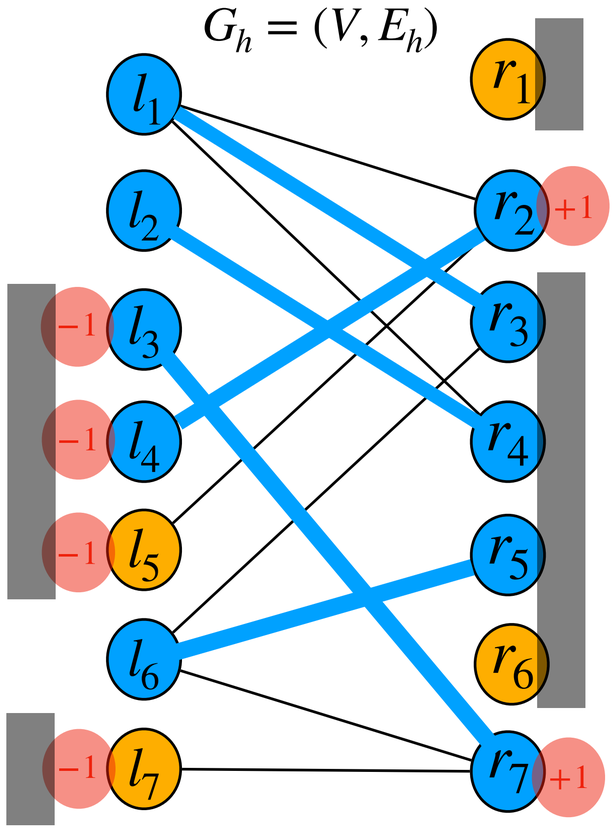
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



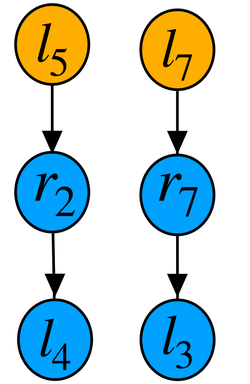
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l,r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

Lemma:

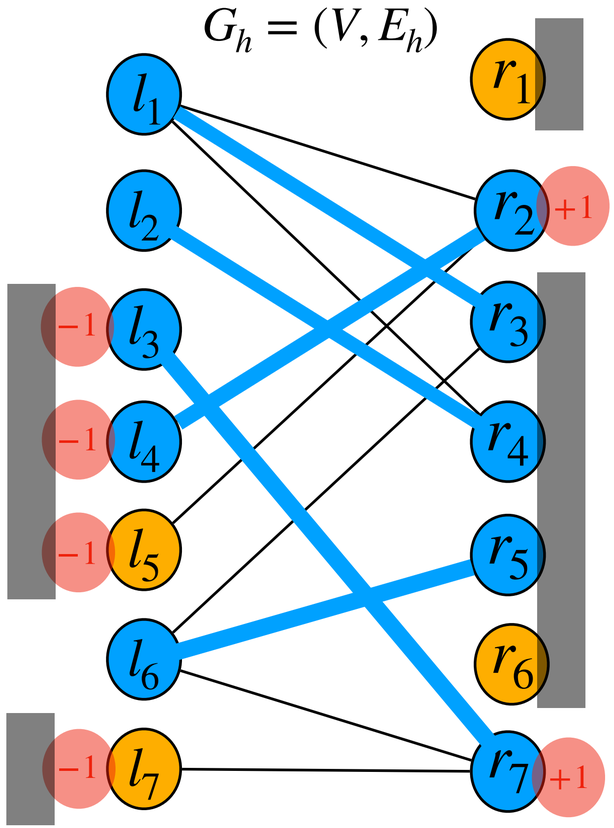
1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof: Let (u, v) be (l, r) or (r, l).

$$\begin{aligned} & l.h' + r.h' \\ = \ & (l.h - \delta) + (r.h + \delta) \\ = \ & l.h + r.h \\ = \ & w(l, r) \end{aligned}$$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



Lemma:

1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof:

l and r are either both in the forest F,

or both outside F.

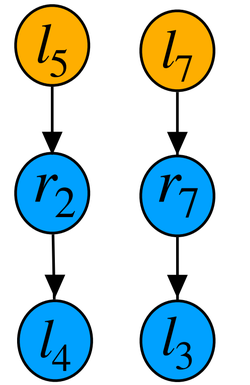
When the search for an M-augmenting path fails

$F_L = L \cap V_F \qquad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l, r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

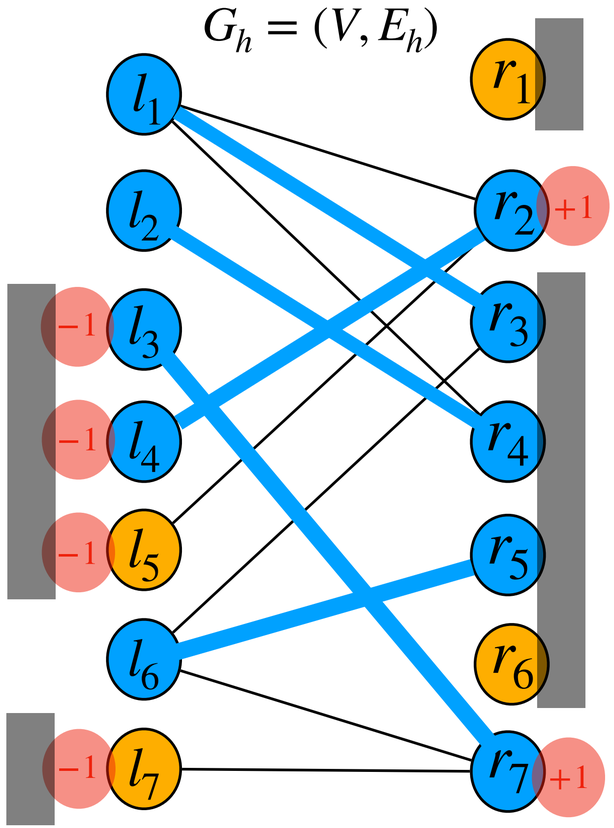
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$



$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$



Lemma:

1. h' is a feasible vertex labeling.
2. If (u, v) is an edge in the breadth-first forest F for $G_{M,h}$, then $(u, v) \in E_{M,h'}$.
3. If (l, r) belongs to the matching M for G_h, then $(l, r) \in E_{M,h'}$.
4. There exist vertices $l \in F_L$ and $r \in R - F_R$ such that $(l, r) \notin E_{M,h}$ but $(l, r) \in E_{M,h'}$.

Proof:

The edge that caused δ.

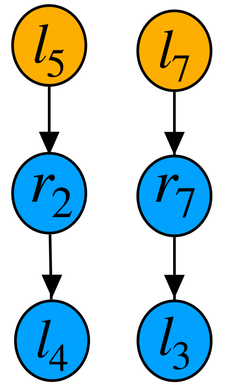
When the search for an M-augmenting path fails

$F_L = L \cap V_F$ $\quad F_R = R \cap V_F$

$\delta = \min\{l.h + r.h - w(l, r) \ : \ l \in F_L \text{ and } r \in R - F_R\}$

Update the vertex labeling:

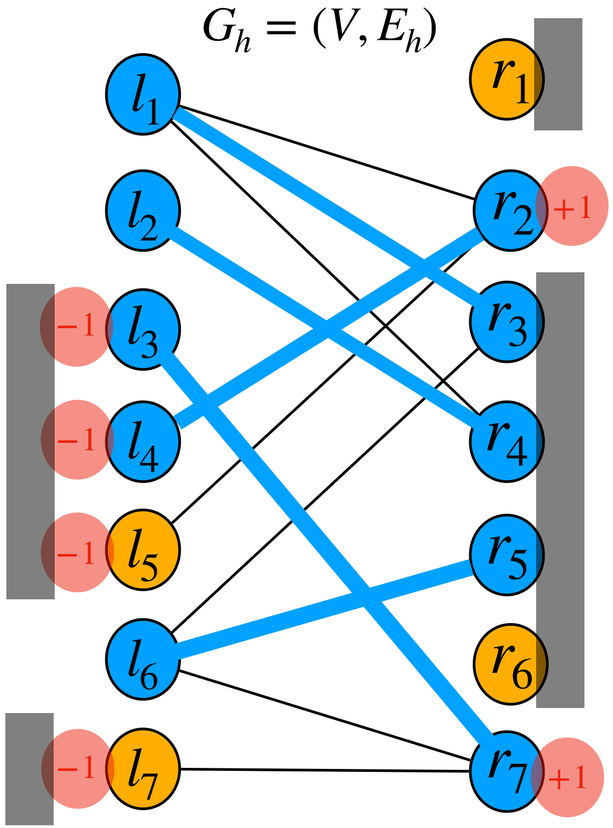
$$v.h' = \begin{cases} v.h - \delta & \text{if } v \in F_L \\ v.h + \delta & \text{if } v \in F_R \\ v.h & \text{otherwise } (v \in V - V_F) \end{cases}$$

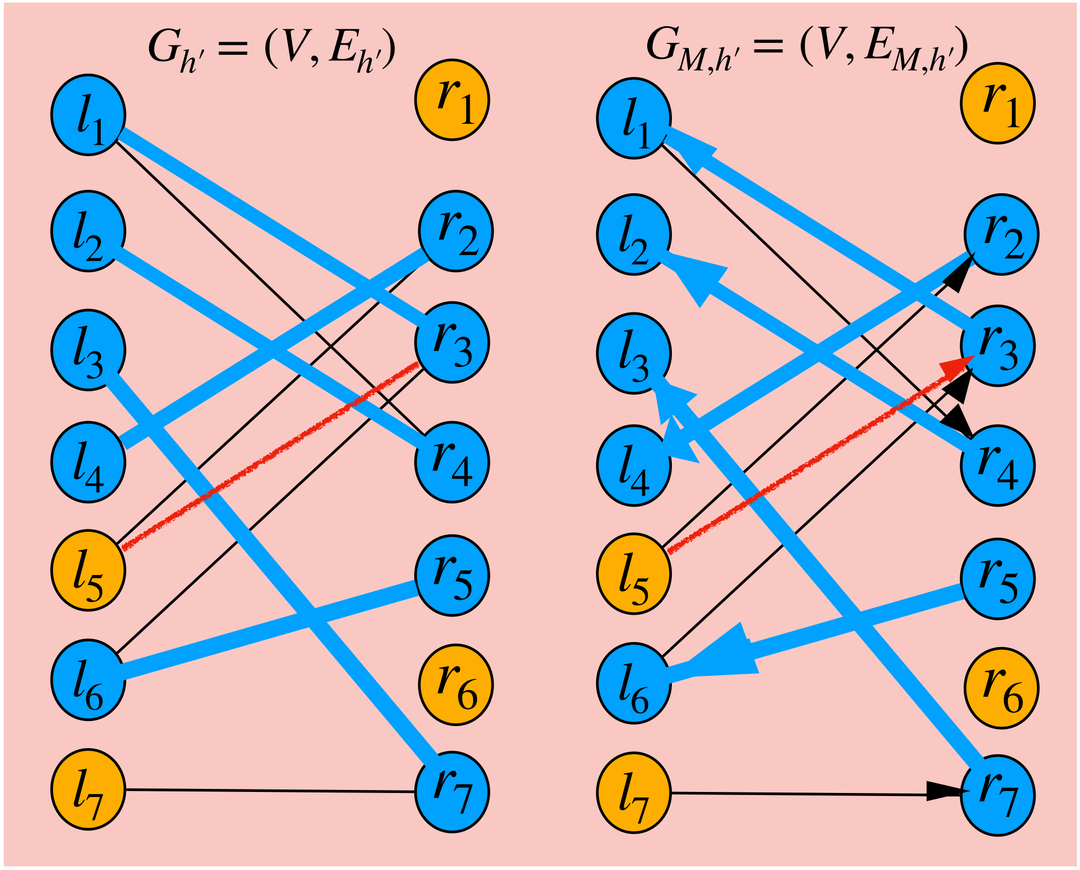


$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

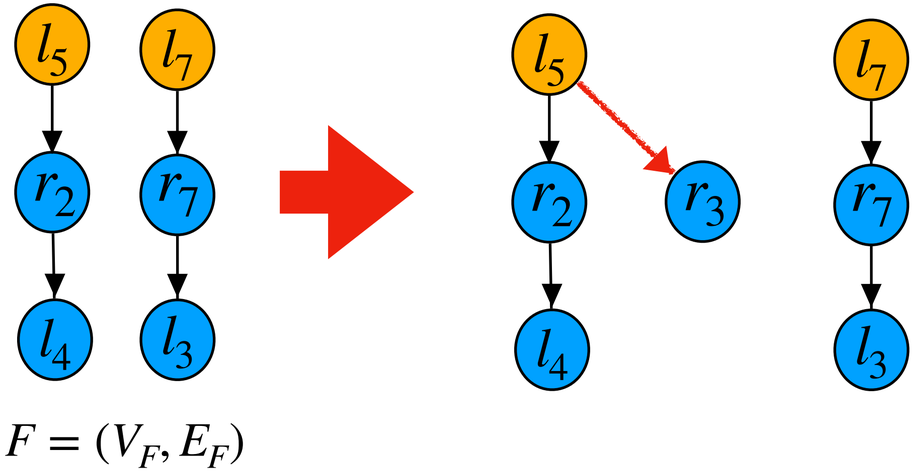
$\delta = 1$





When the search for an M-augmenting path fails

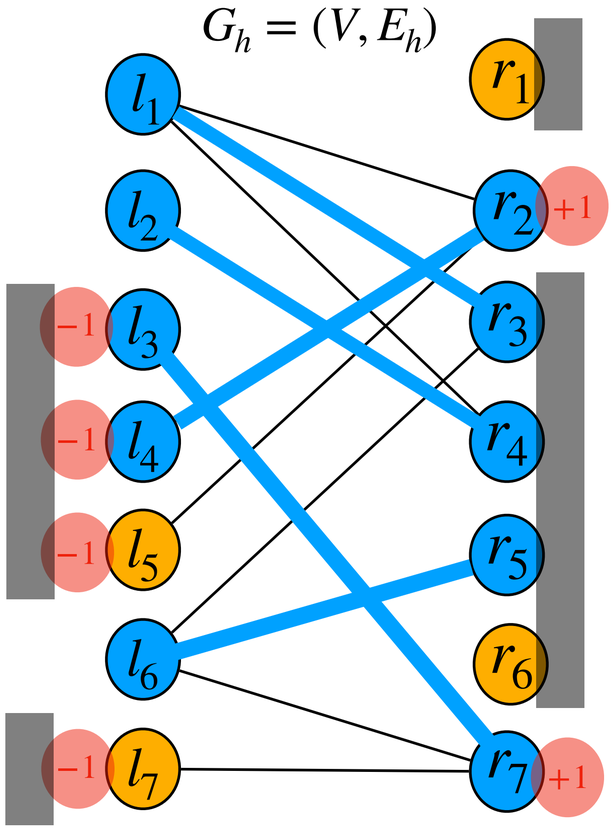
The matching remains the same,
but the breadth-first forest has become larger.

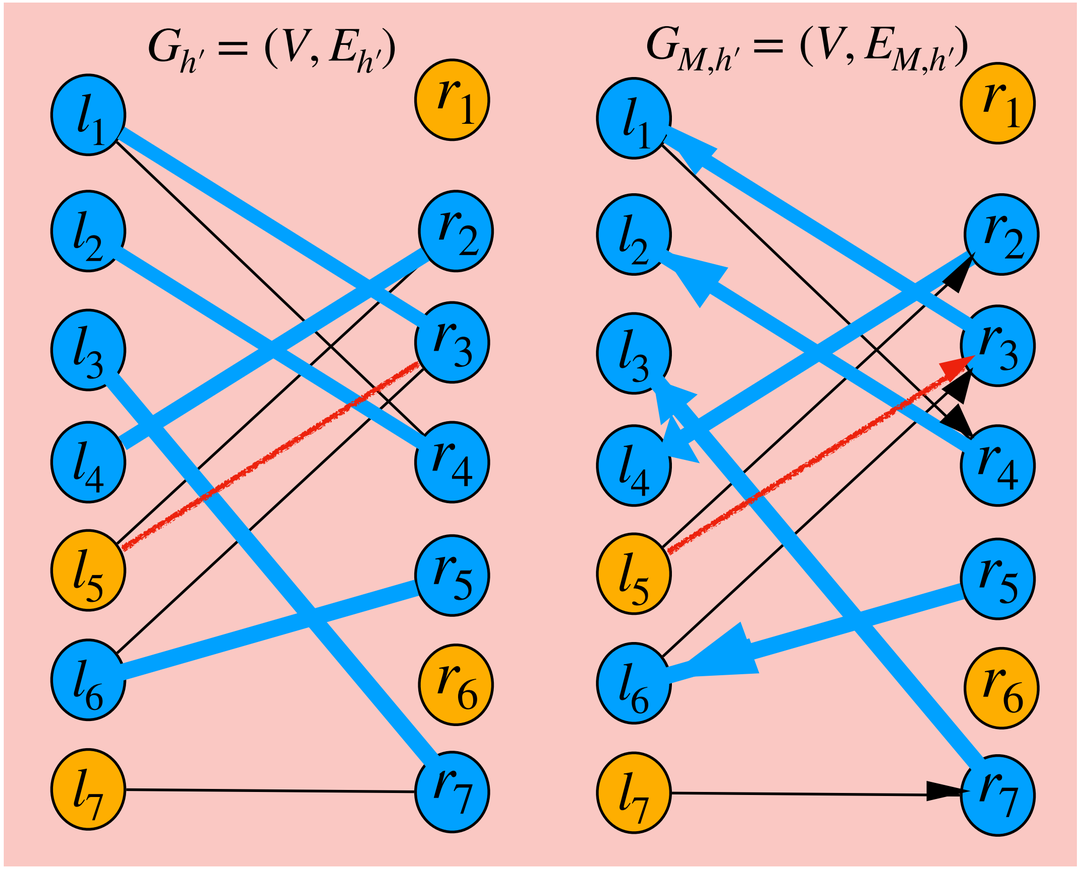


$F = (V_F, E_F)$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	0 + 1	0	0	0	0	0 + 1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	15 − 1	11	9	6	7	9	5	15
l_4	9 − 1	3	9	6	7	5	6	3
l_5	6 − 1	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	8 − 1	3	4	5	4	3	6	8

$\delta = 1$

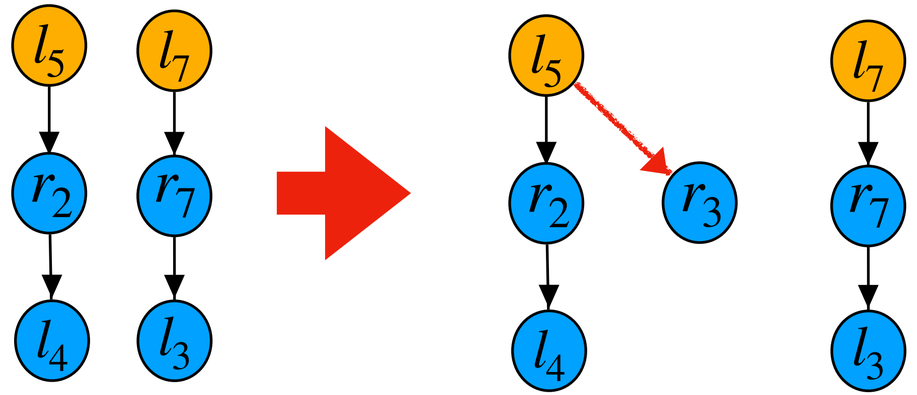




When the search for an M-augmenting path fails

The matching remains the same,

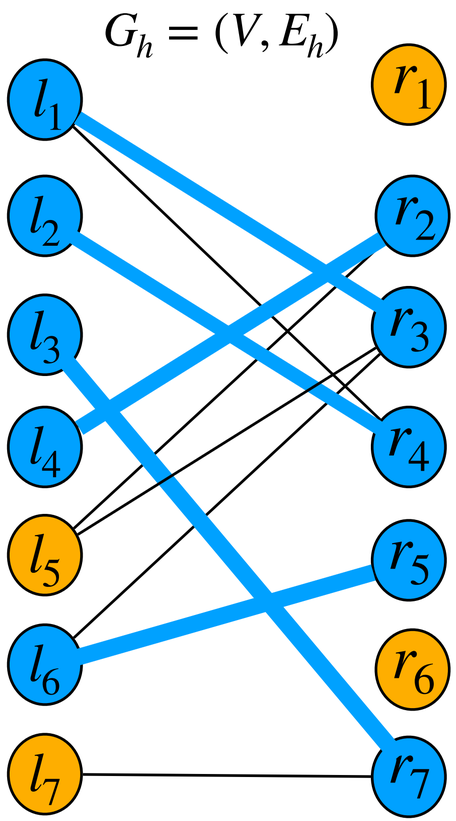
but the breadth-first forest has become larger.

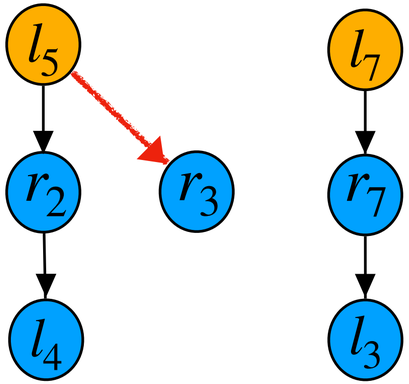


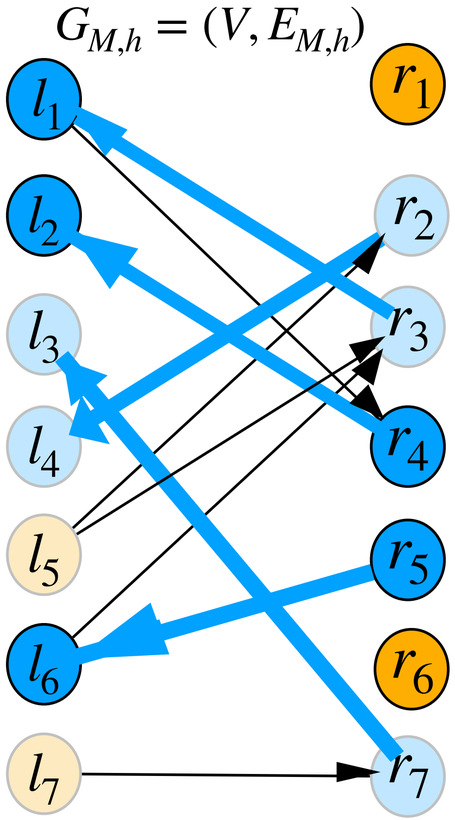
$F = (V_F, E_F)$

If we keep enlarging the forest, eventually it will reach an unmatched vertex on the right, so we will find an M-augmenting path, which will in turn make the matching larger. So eventually, we will find a perfect matching in the equality subgraph, which is an optimal solution.

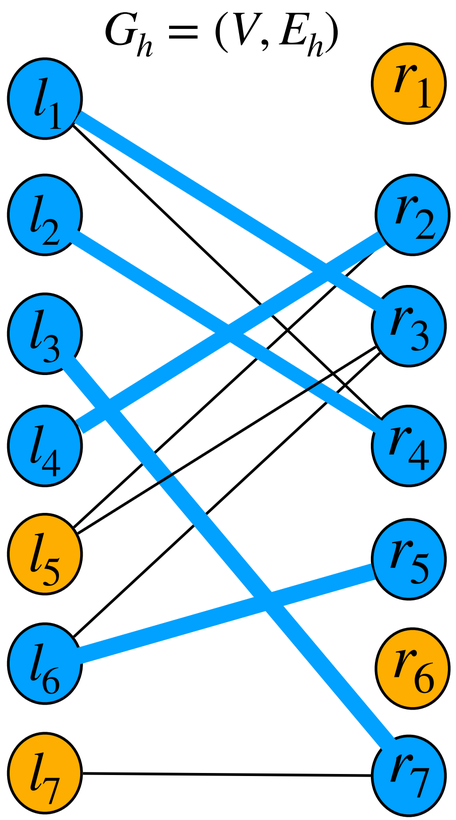
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	1	0	0	0	0	1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	14	11	9	6	7	9	5	15
l_4	8	3	9	6	7	5	6	3
l_5	5	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	7	3	4	5	4	3	6	8

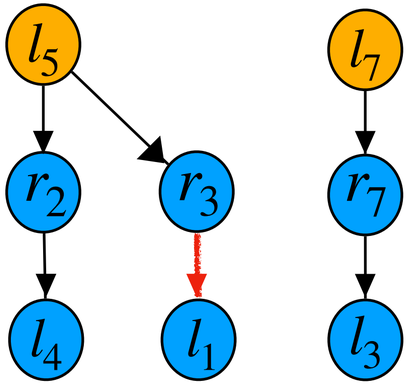


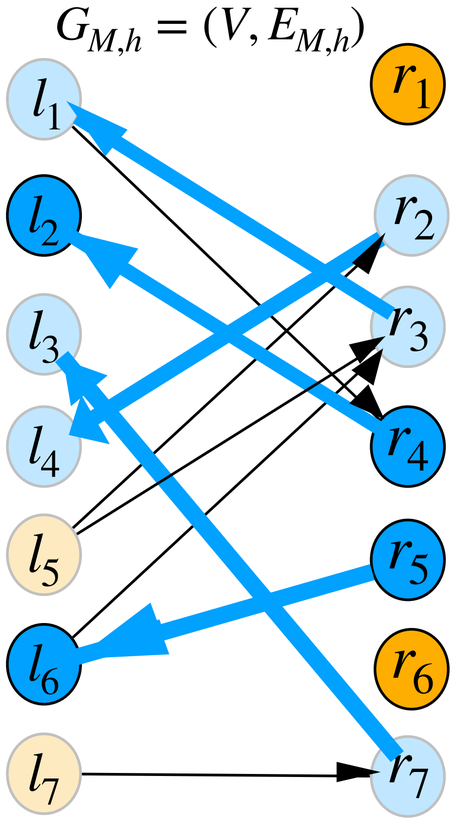




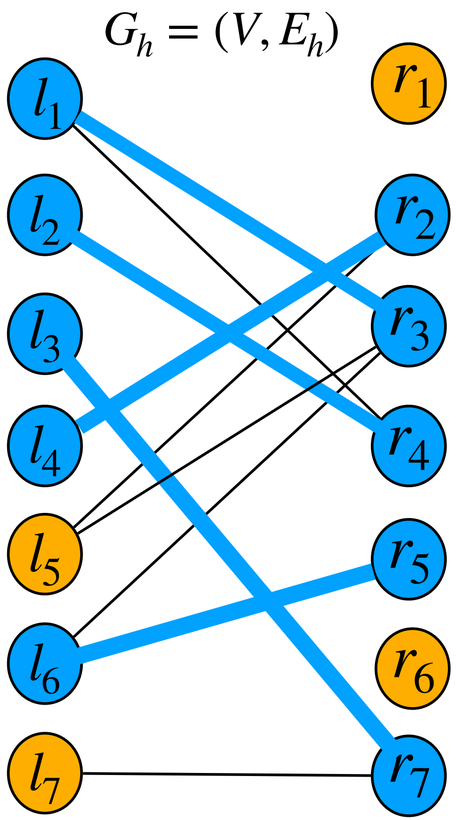
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	1	0	0	0	0	1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	14	11	9	6	7	9	5	15
l_4	8	3	9	6	7	5	6	3
l_5	5	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	7	3	4	5	4	3	6	8

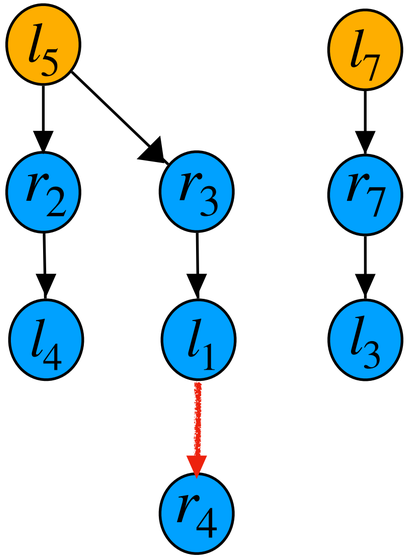


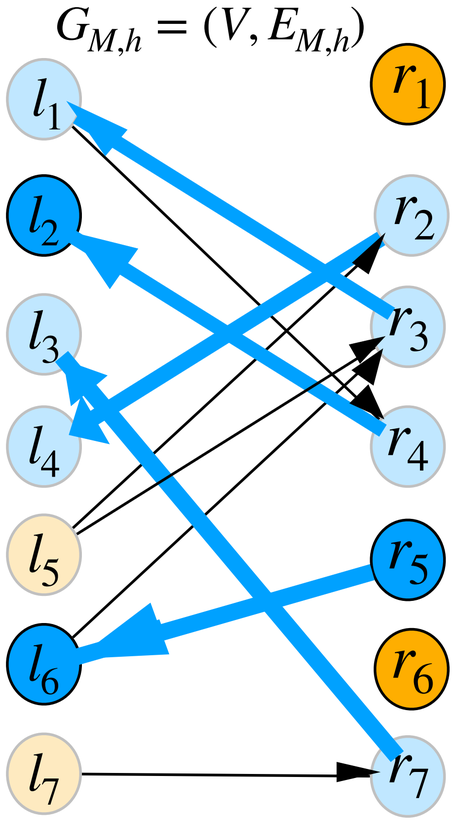




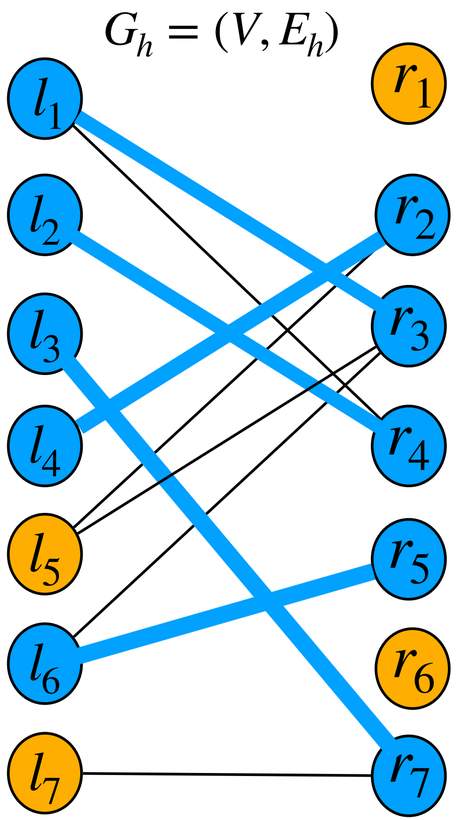
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	1	0	0	0	0	1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	14	11	9	6	7	9	5	15
l_4	8	3	9	6	7	5	6	3
l_5	5	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	7	3	4	5	4	3	6	8

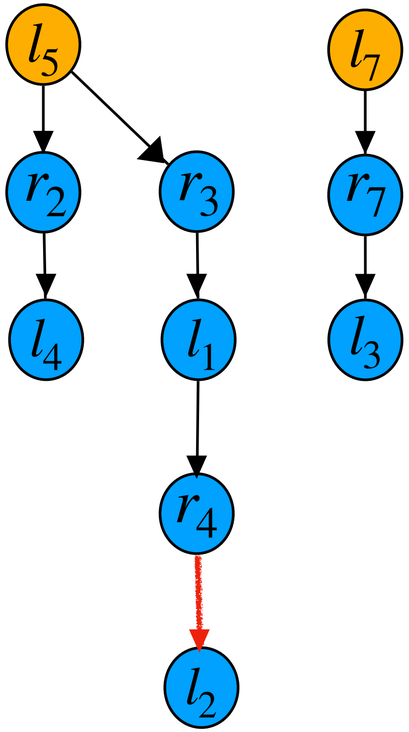


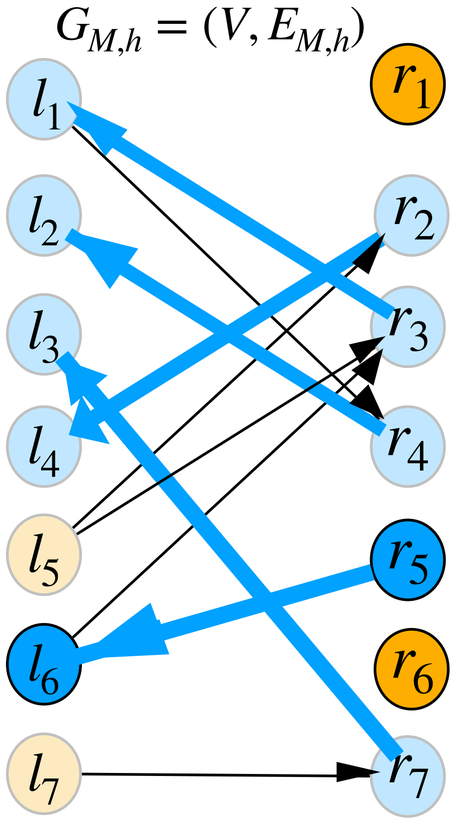




		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	1	0	0	0	0	1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	14	11	9	6	7	9	5	15
l_4	8	3	9	6	7	5	6	3
l_5	5	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	7	3	4	5	4	3	6	8

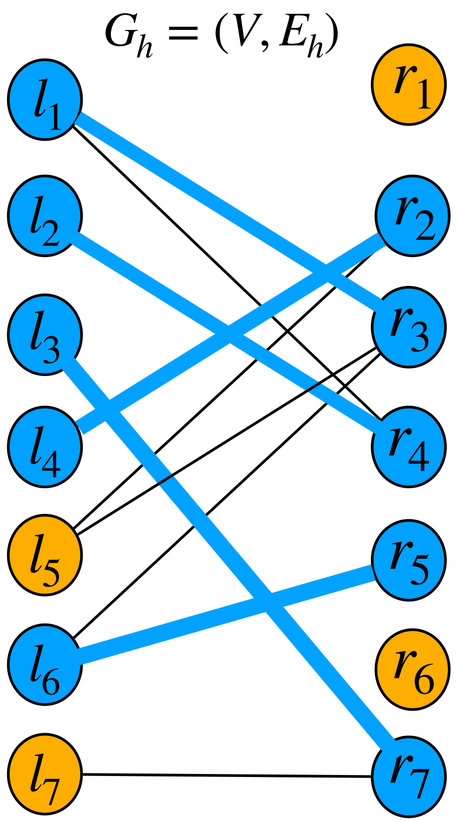


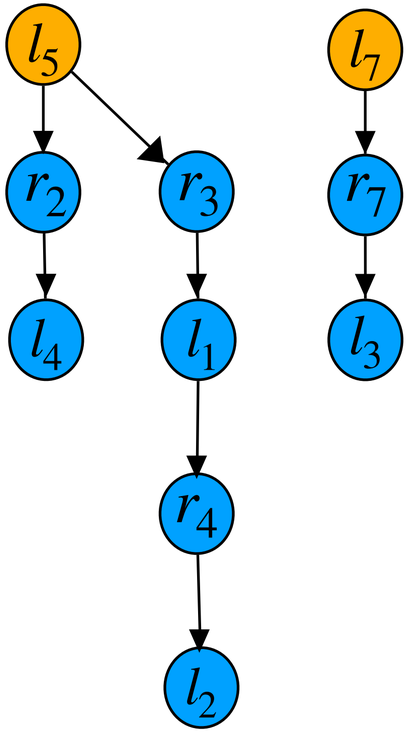


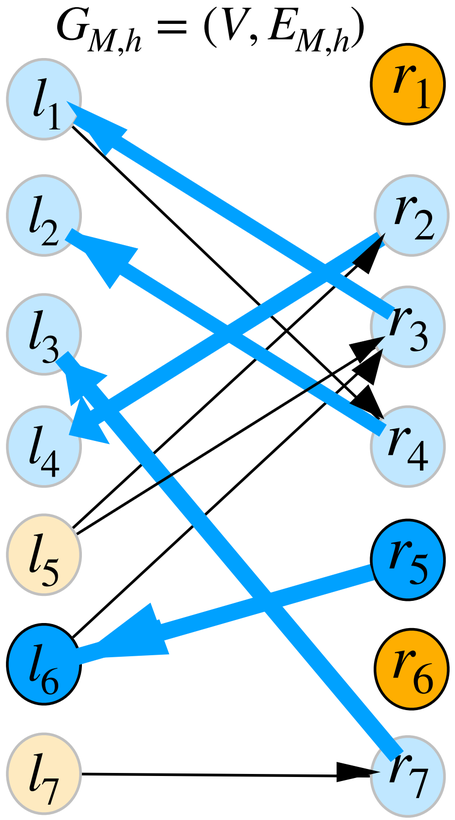


		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	1	0	0	0	0	1
l_1	10	4	10	10	10	2	9	3
l_2	12	6	8	5	12	9	7	2
l_3	14	11	9	6	7	9	5	15
l_4	8	3	9	6	7	5	6	3
l_5	5	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	7	3	4	5	4	3	6	8

$$\delta = 1$$

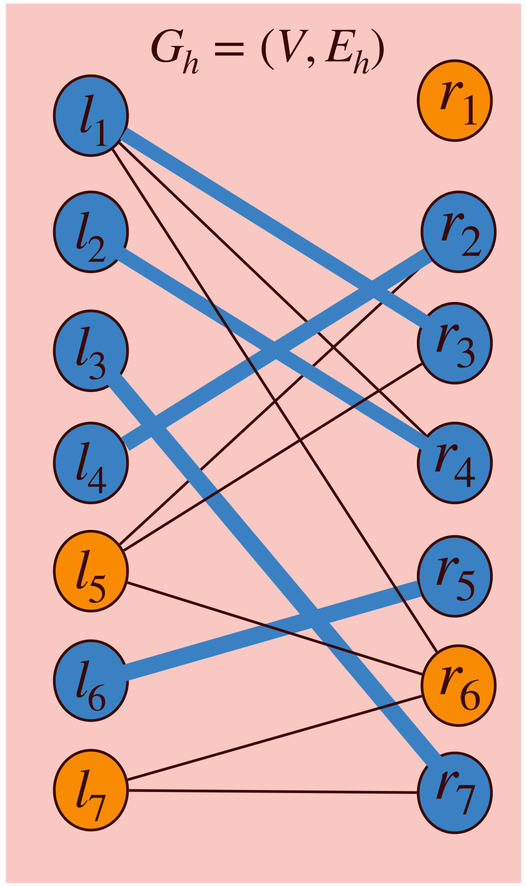


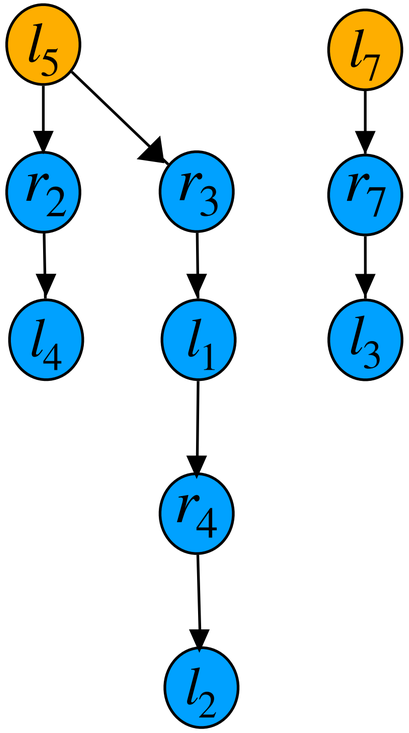


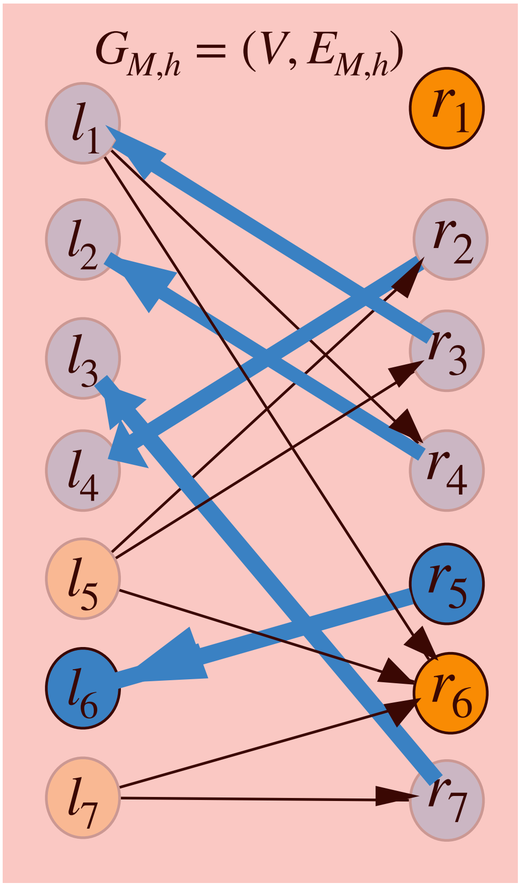


		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

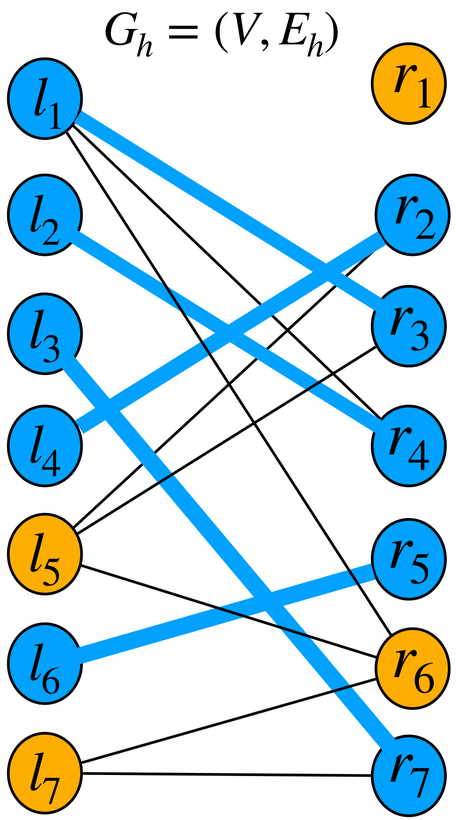
$$\delta = 1$$

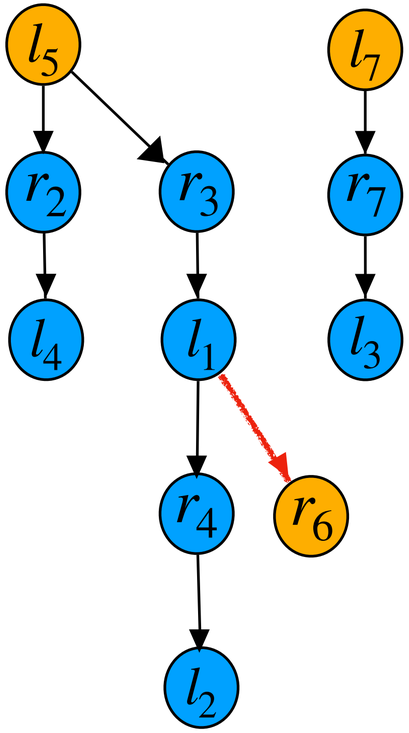


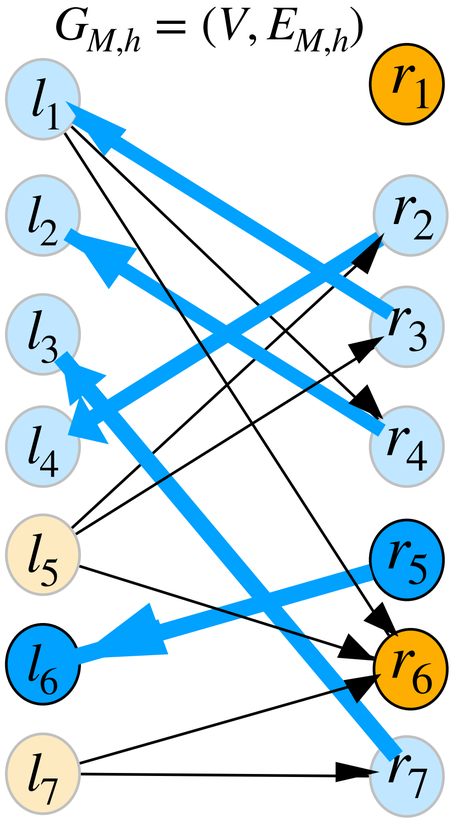




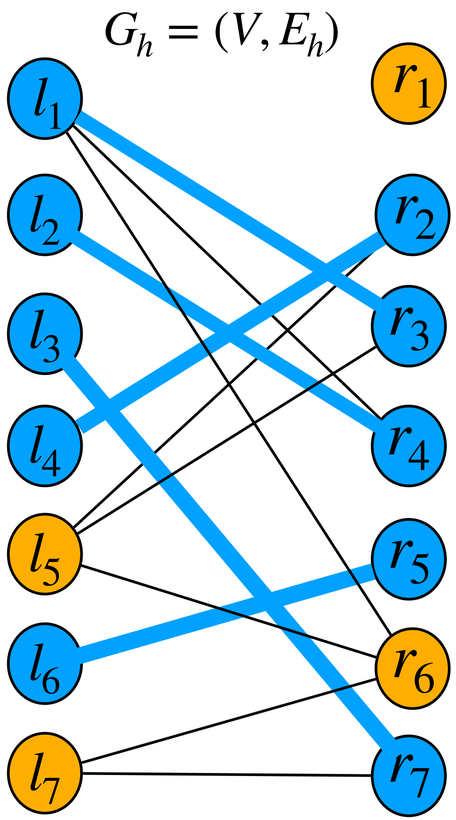
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

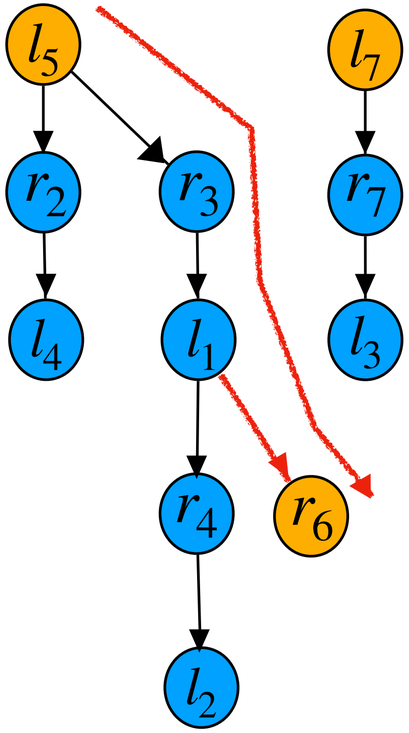


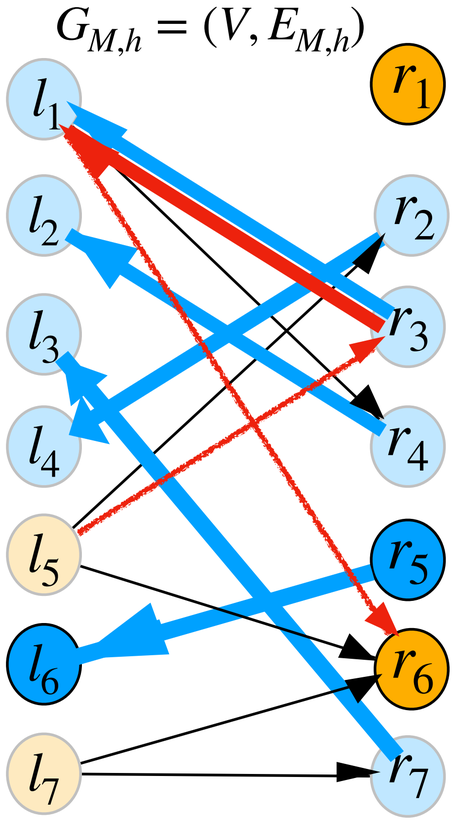




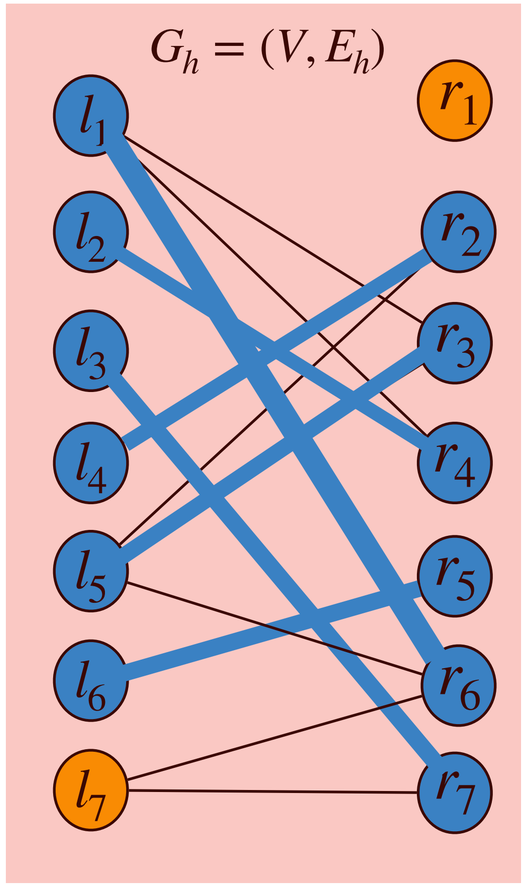
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

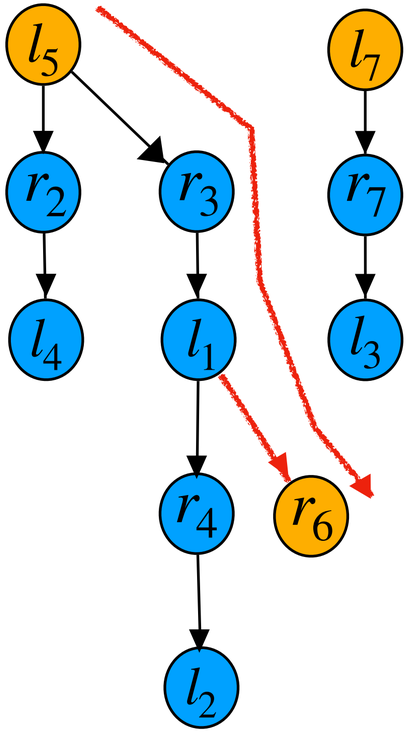


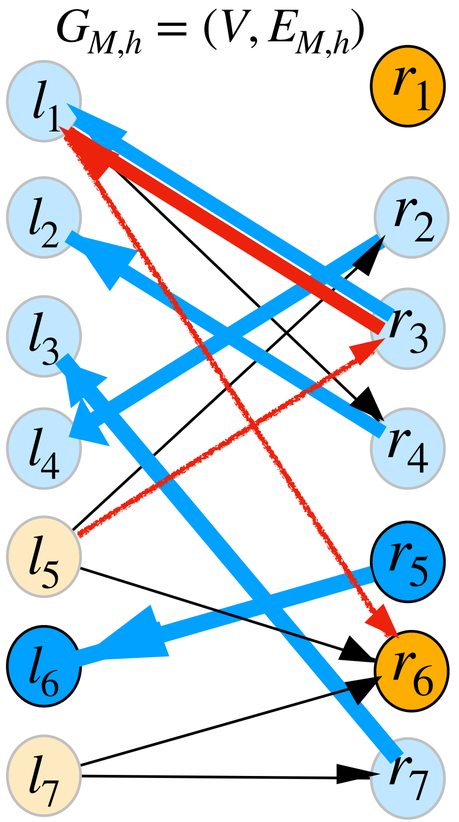




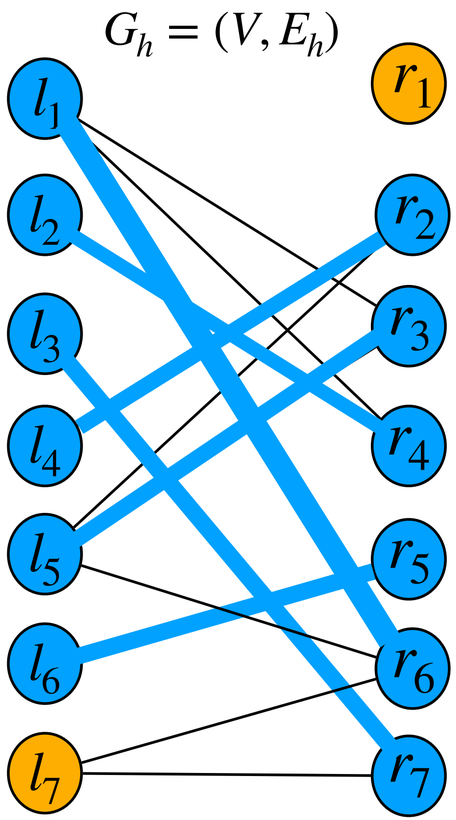
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

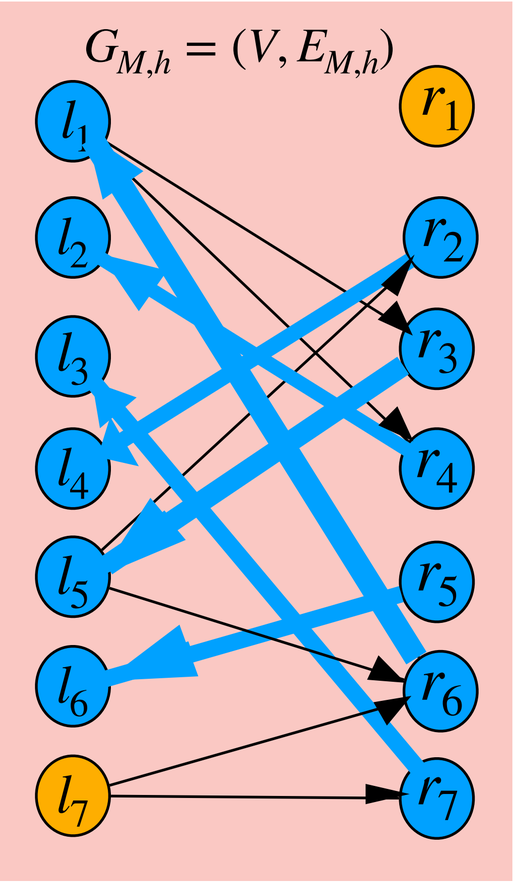




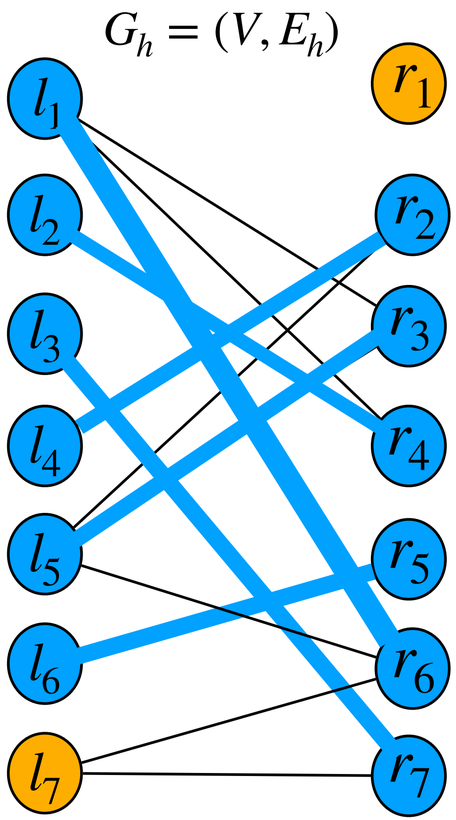


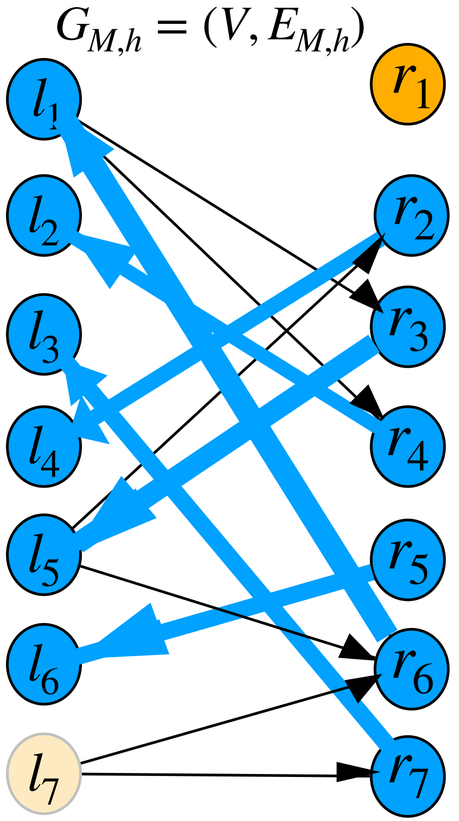
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8



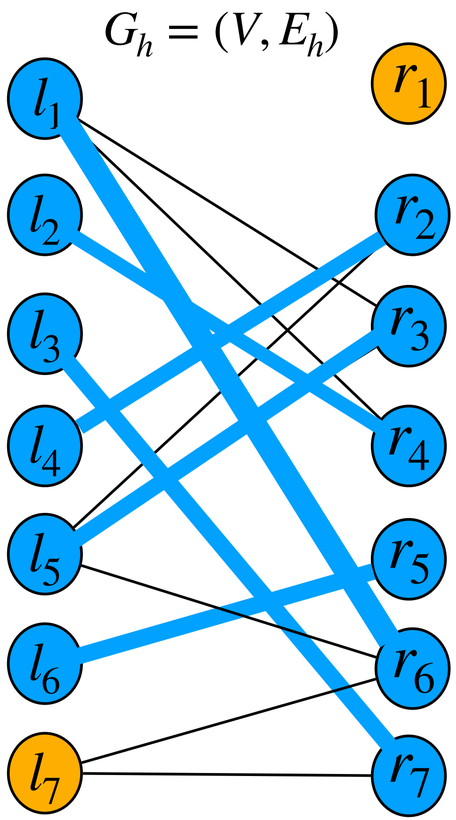


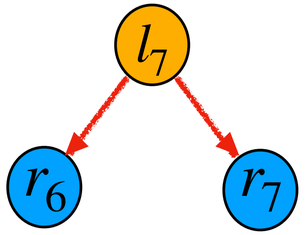
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

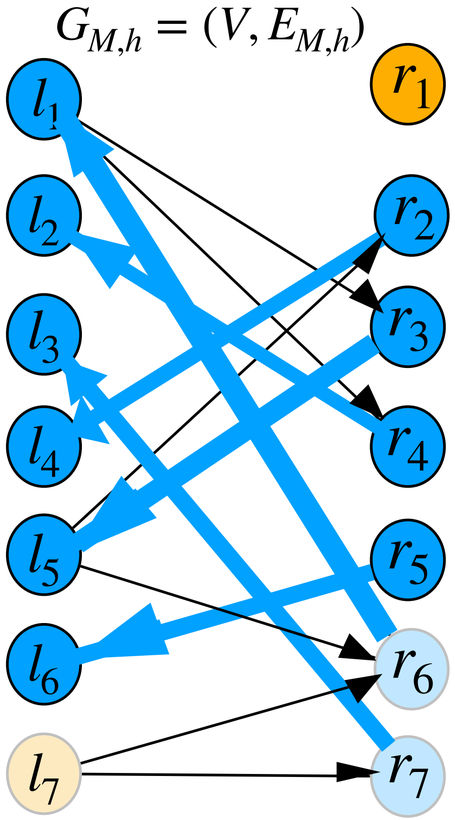




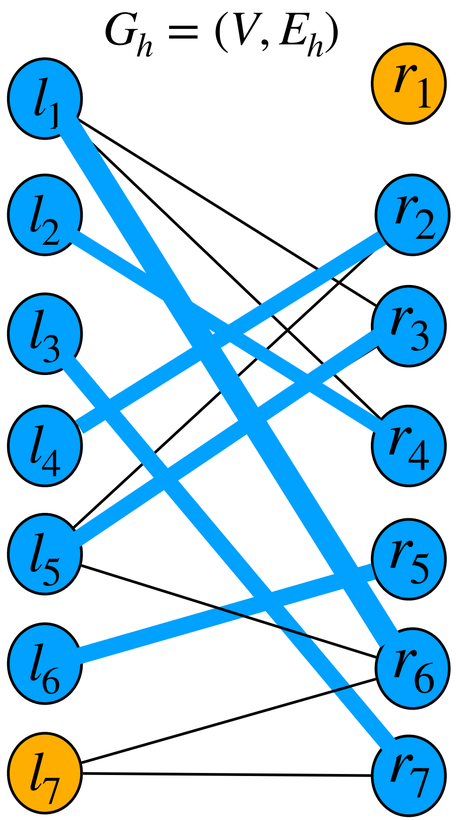
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

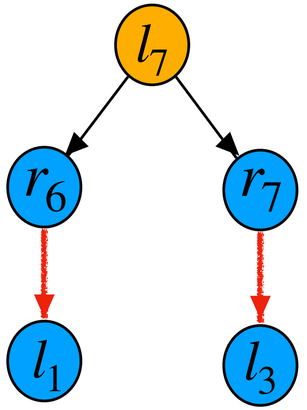


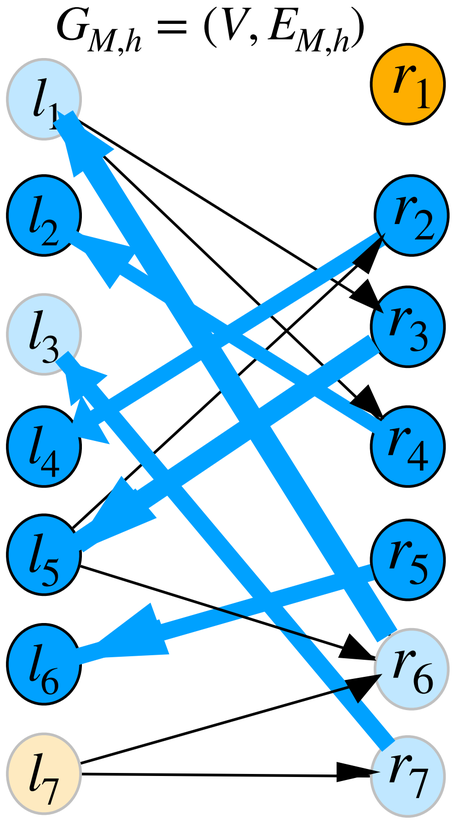




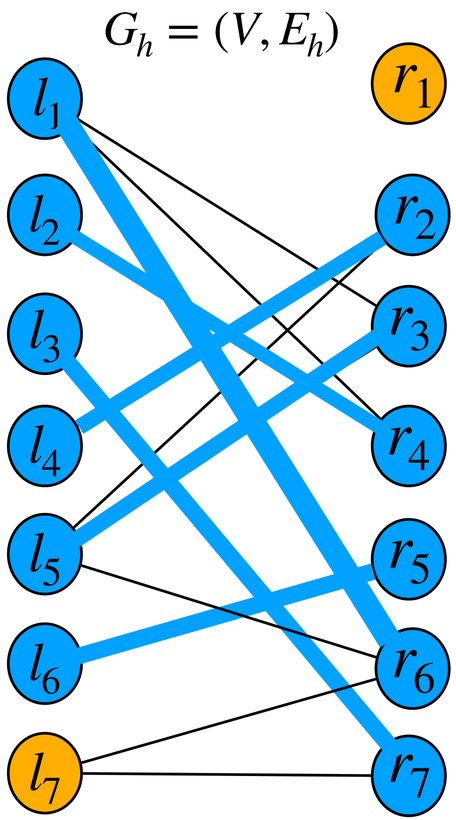
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

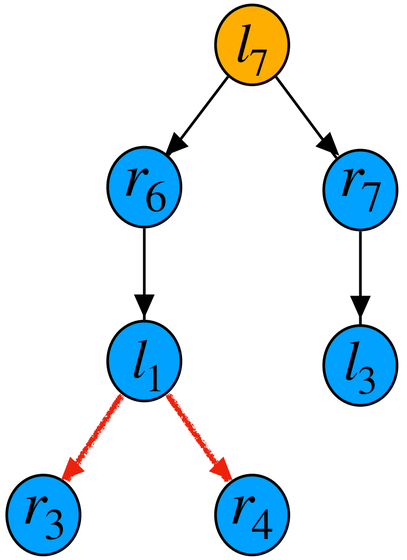


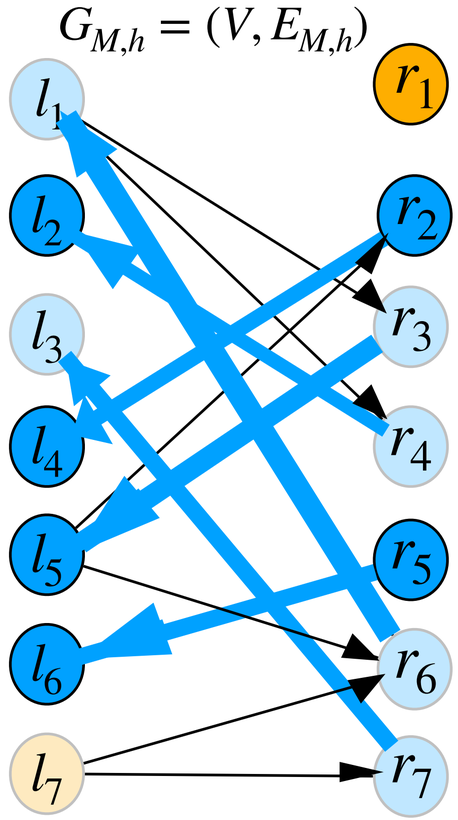




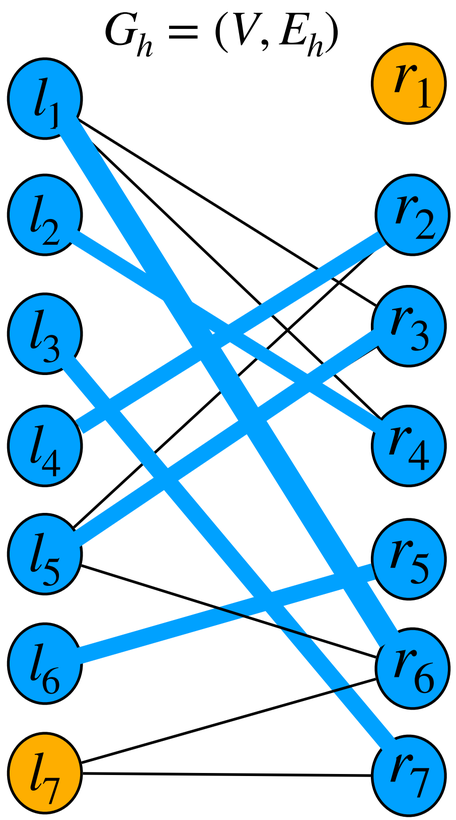
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

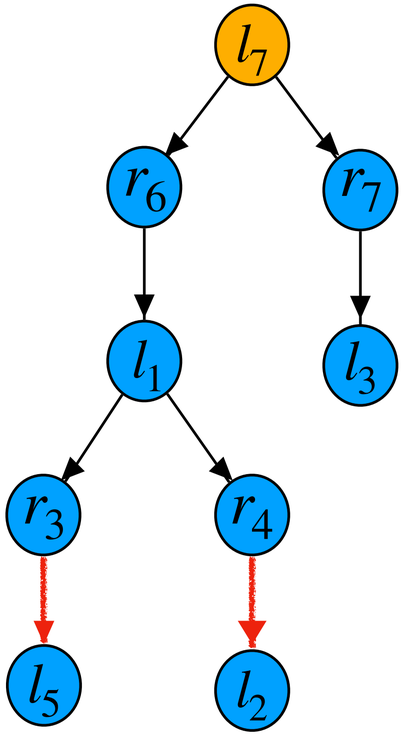


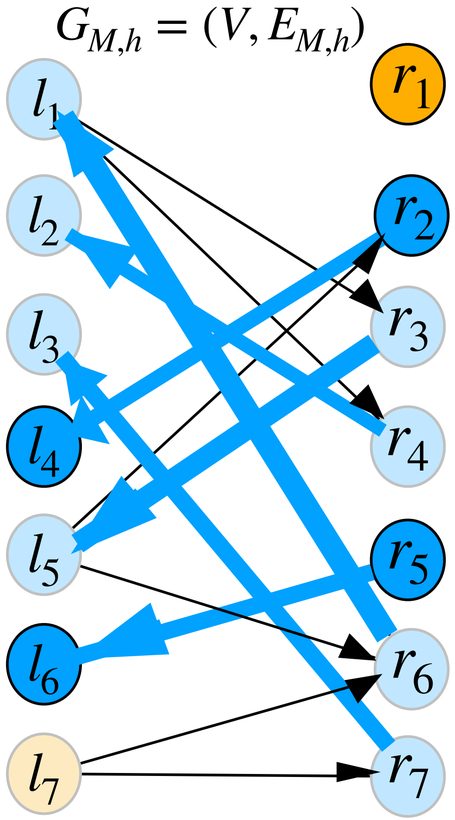




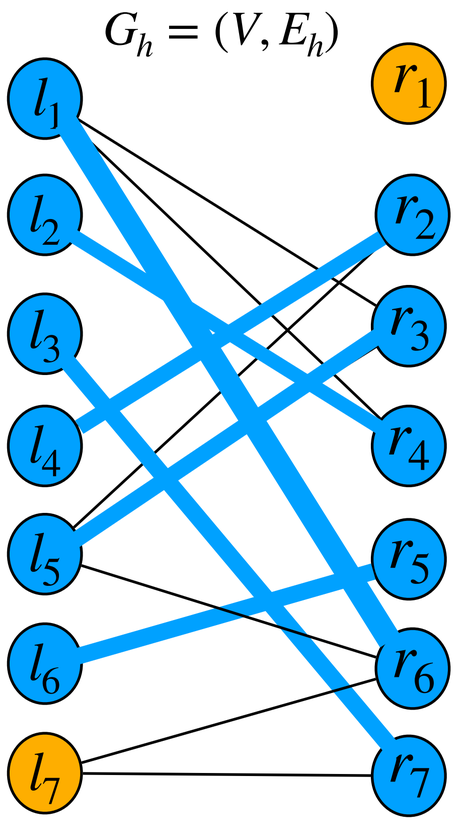
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

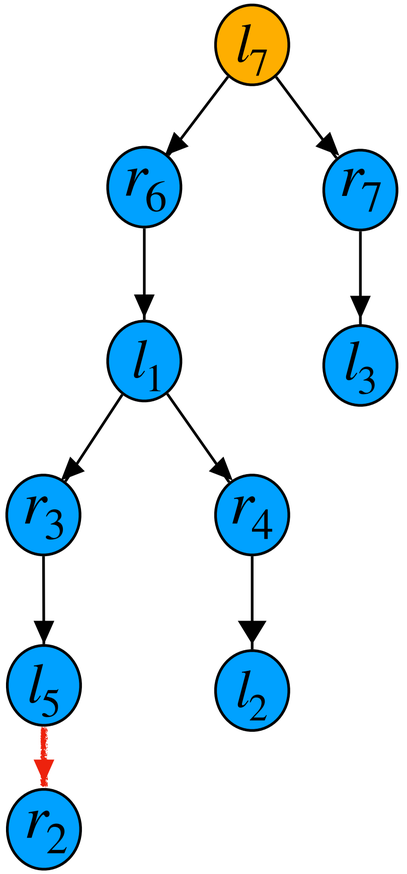


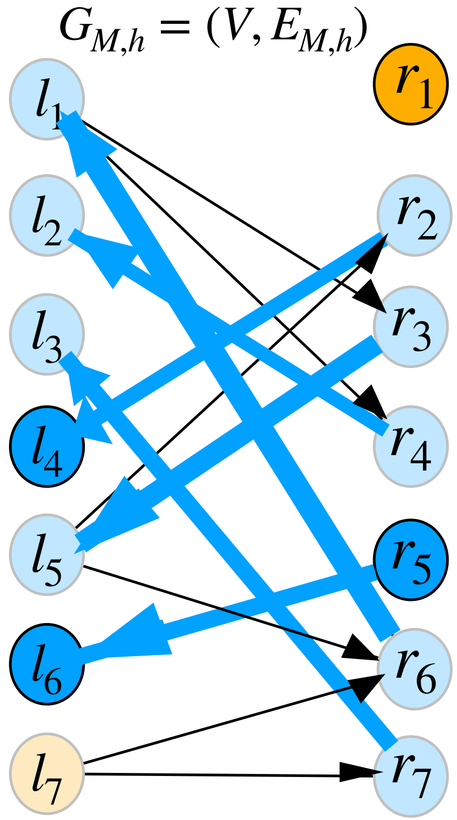




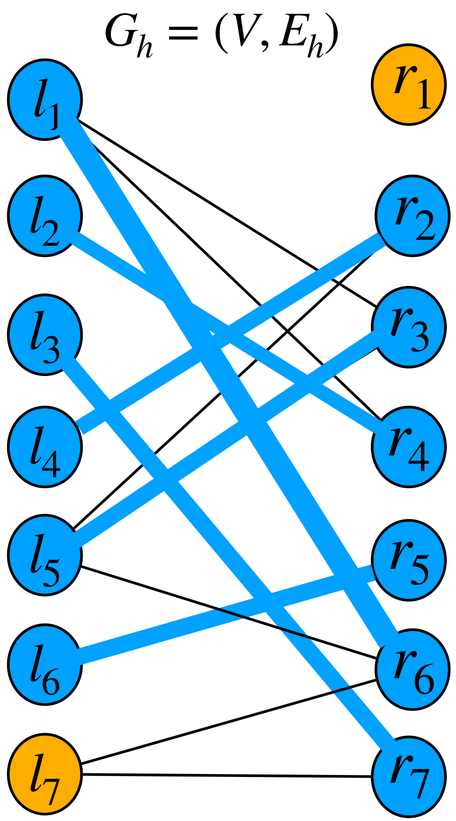
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

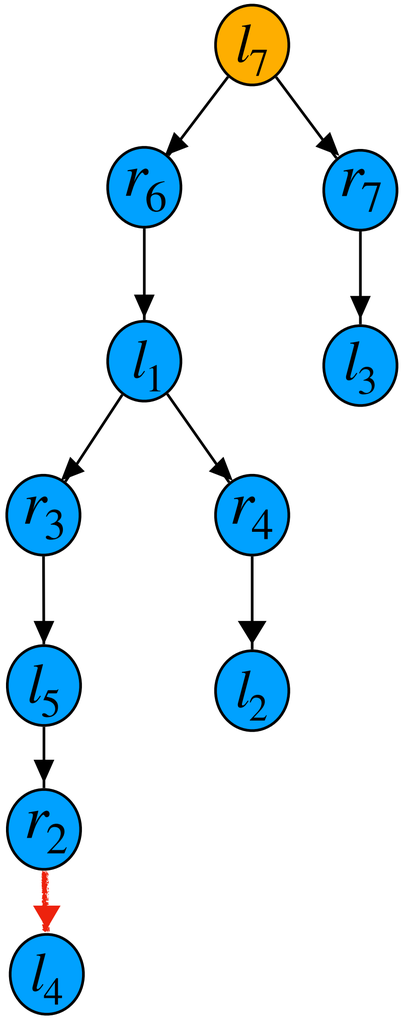


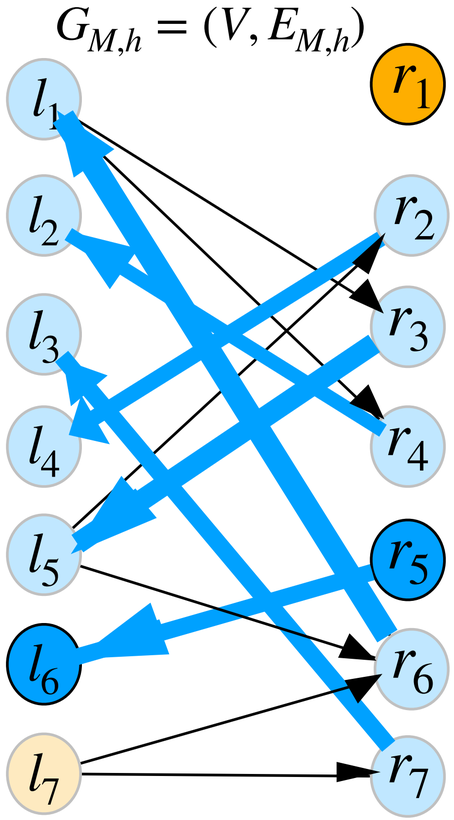




		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

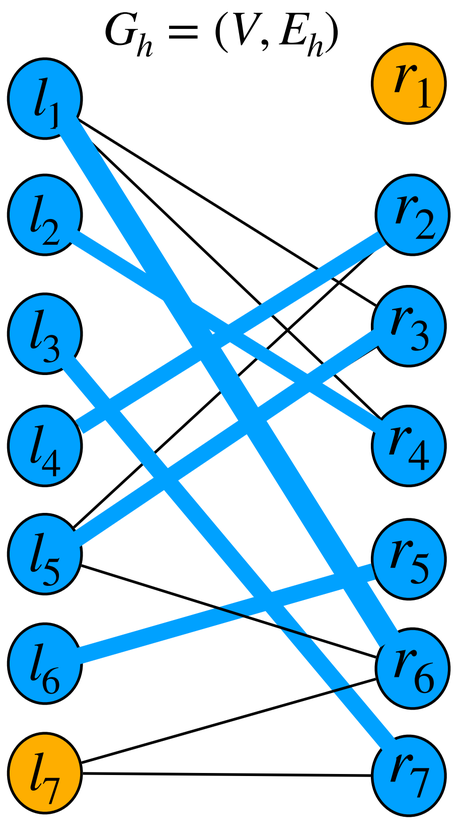


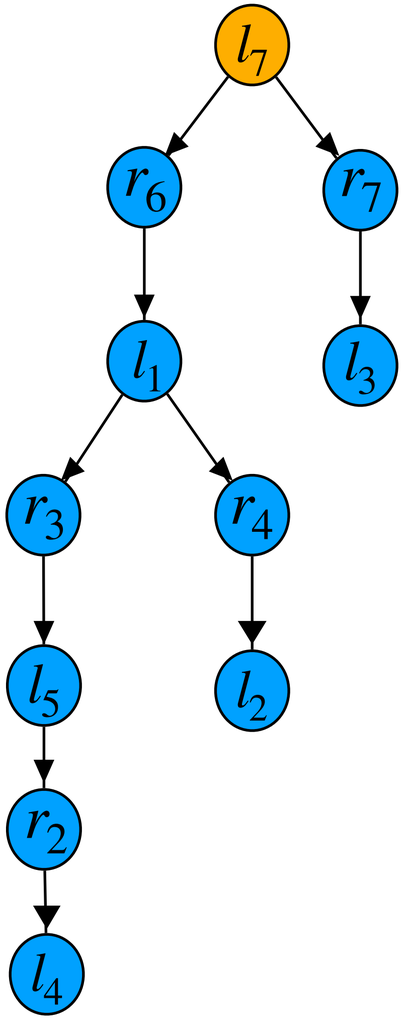


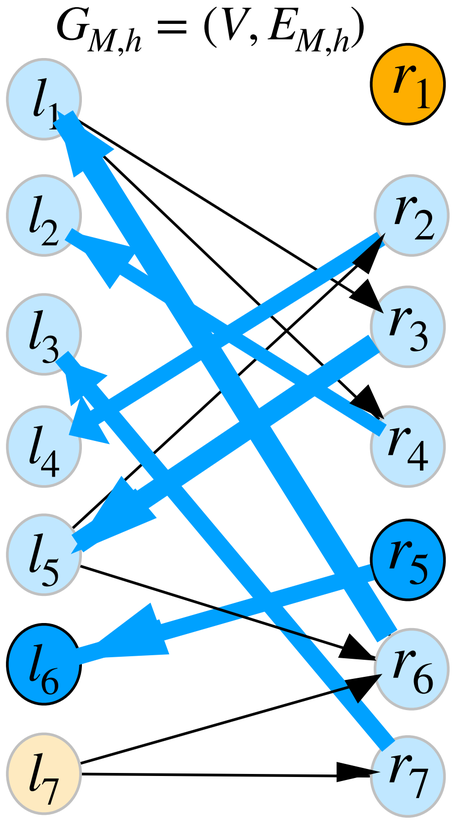


		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	2	1	1	0	0	2
l_1	9	4	10	10	10	2	9	3
l_2	11	6	8	5	12	9	7	2
l_3	13	11	9	6	7	9	5	15
l_4	7	3	9	6	7	5	6	3
l_5	4	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	6	3	4	5	4	3	6	8

$$\delta = 2$$

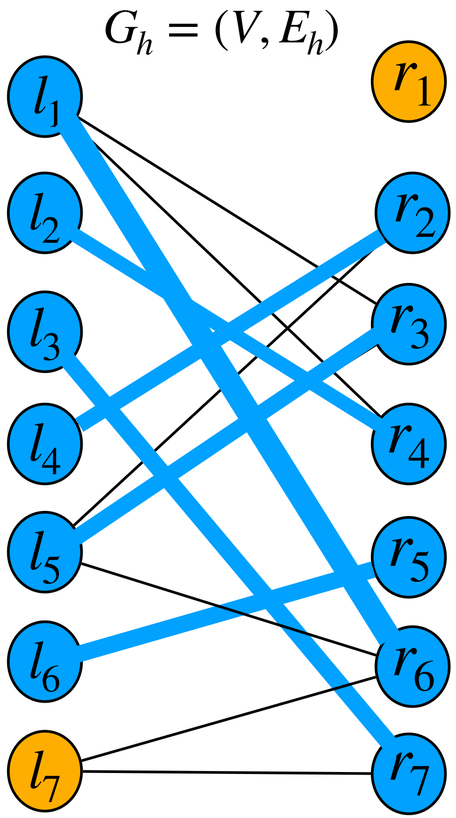


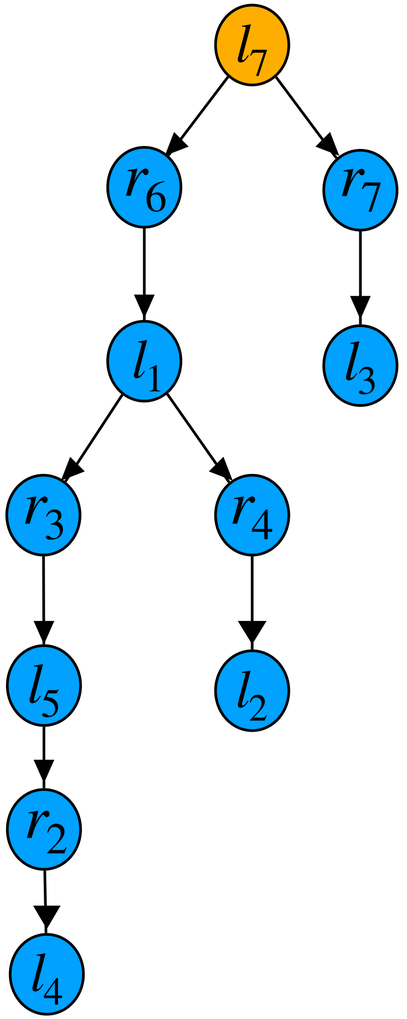


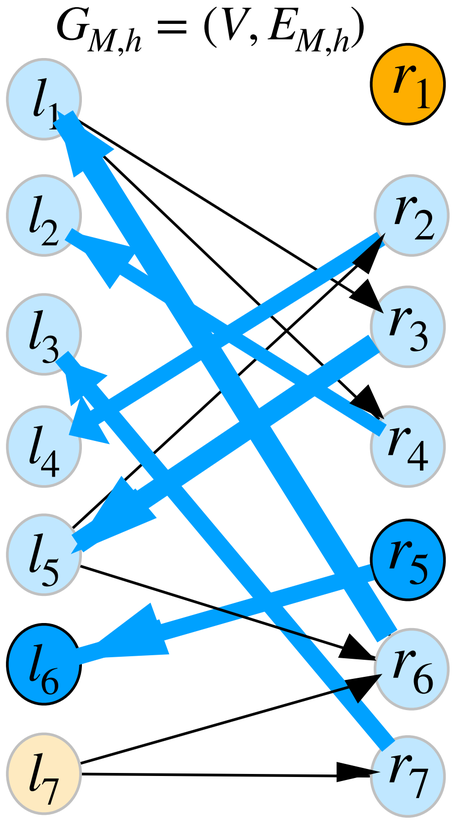


	r_1	r_2	r_3	r_4	r_5	r_6	r_7
h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8

$$\delta = 2$$

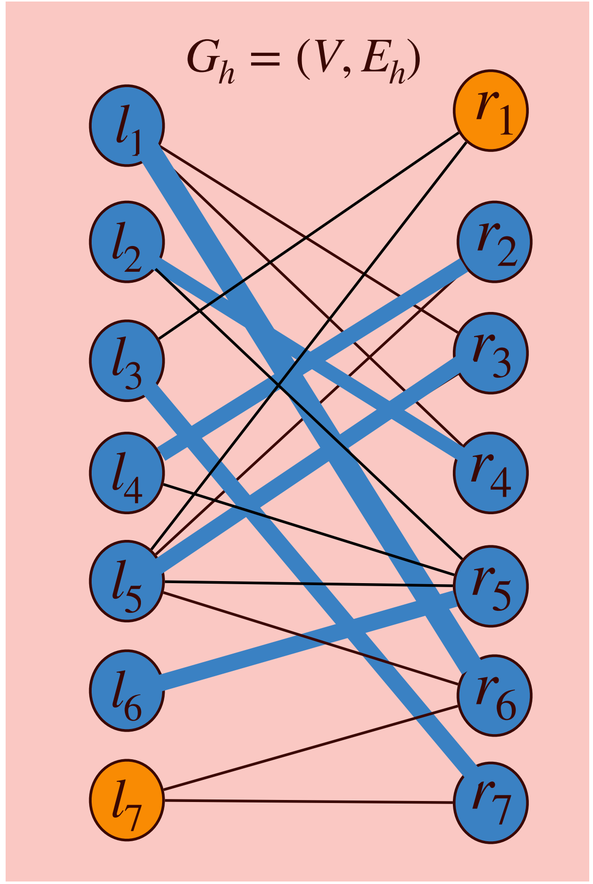


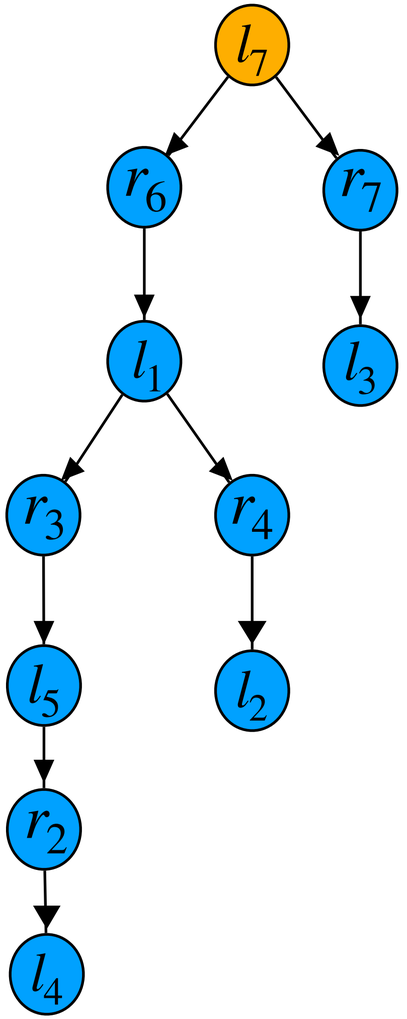


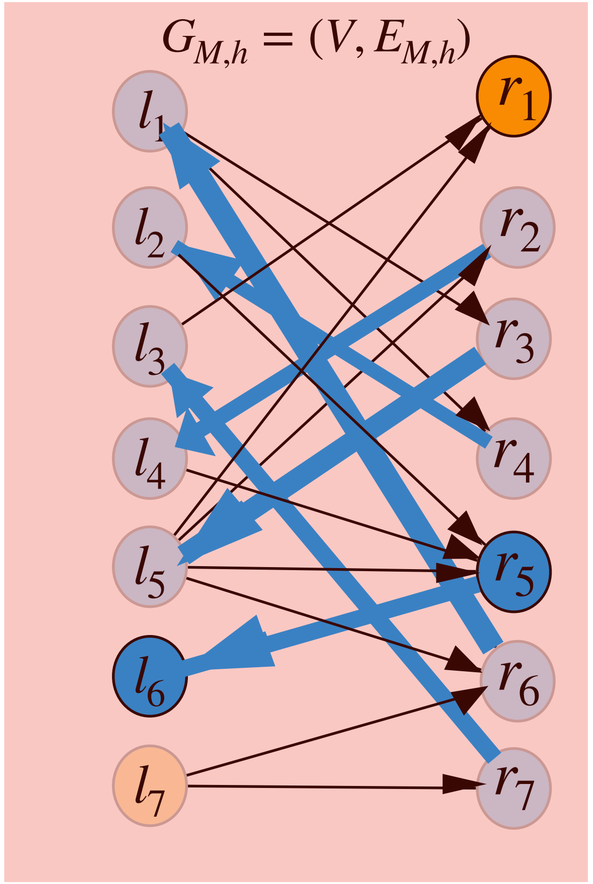


		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8

$$\delta = 2$$

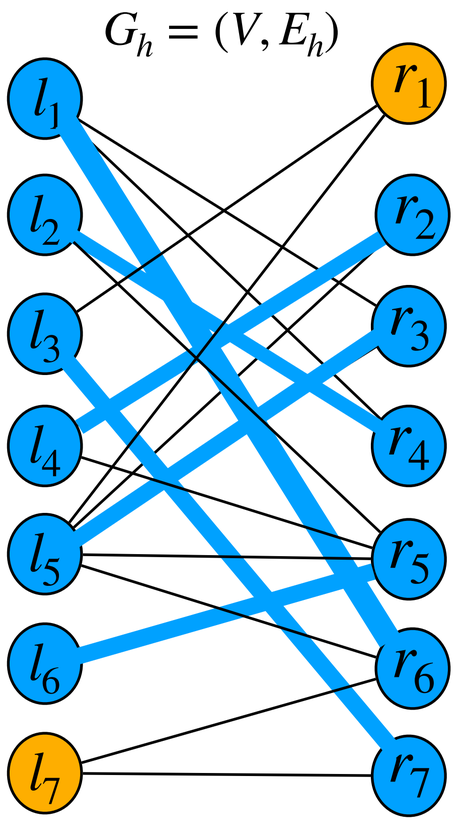


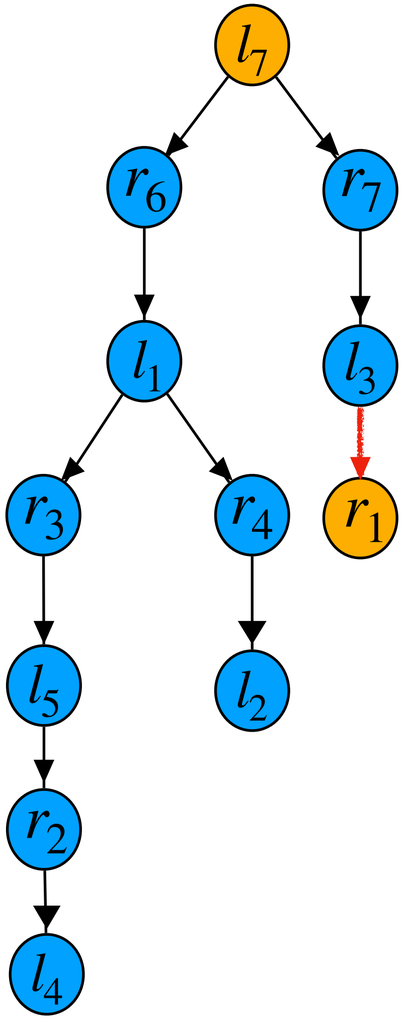


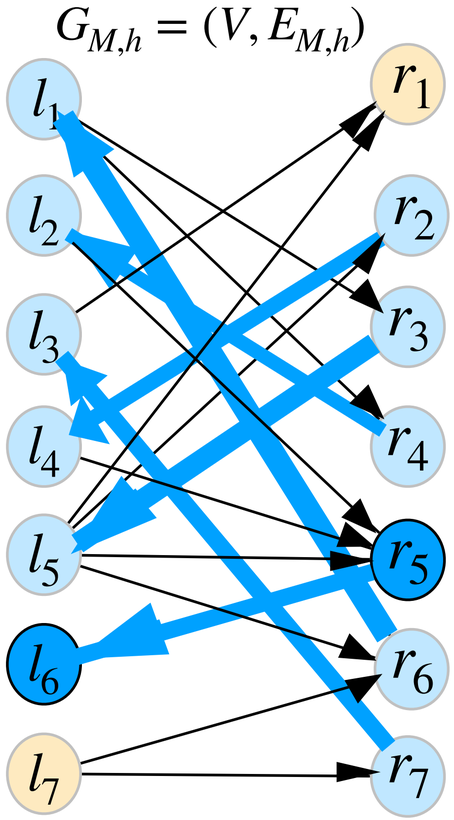


		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8

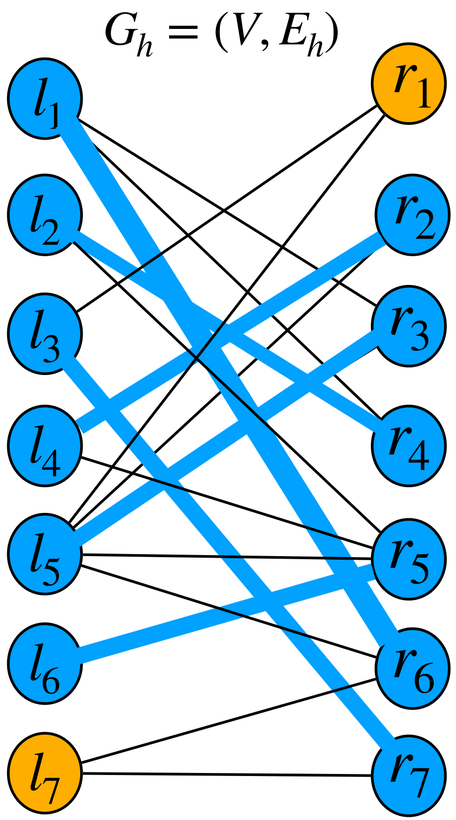
$$\delta = 2$$

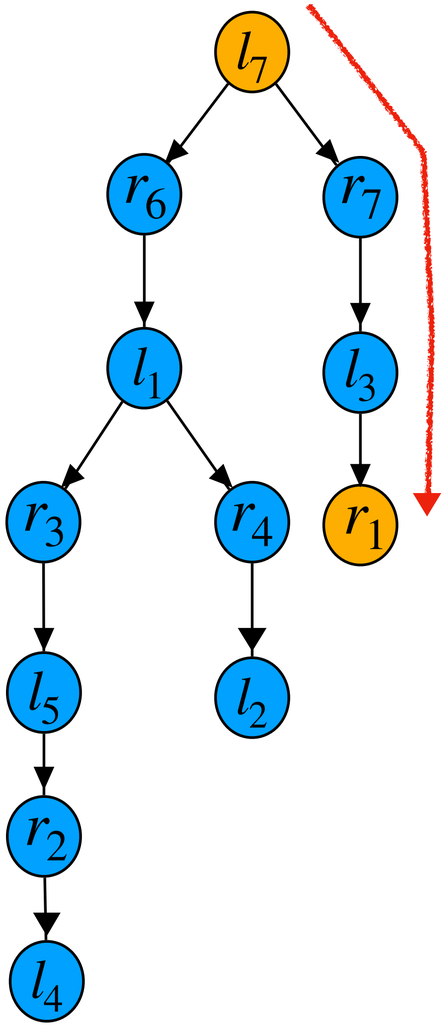


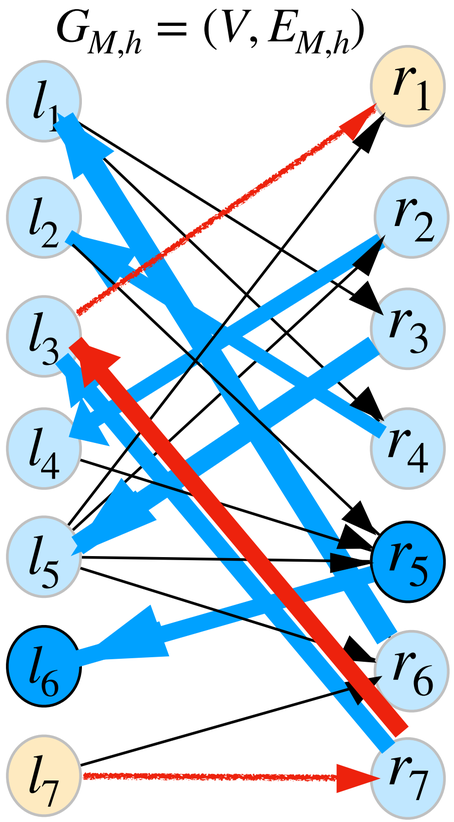




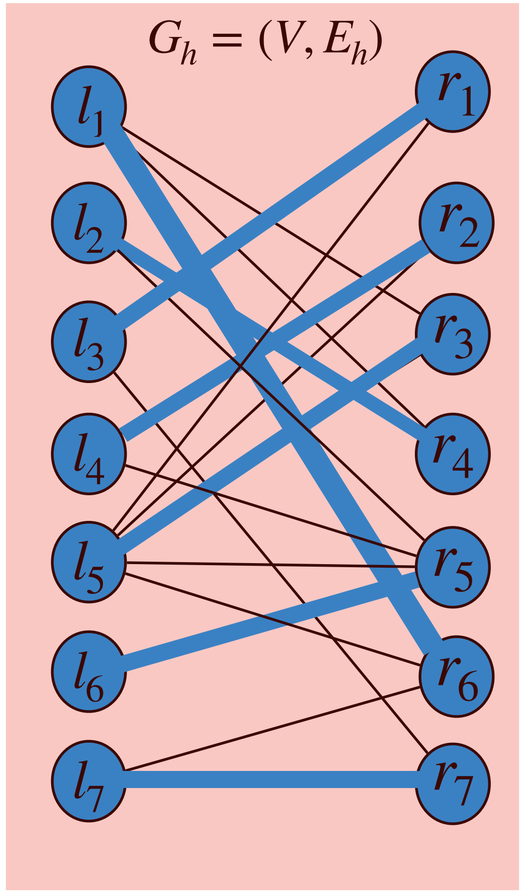
		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8

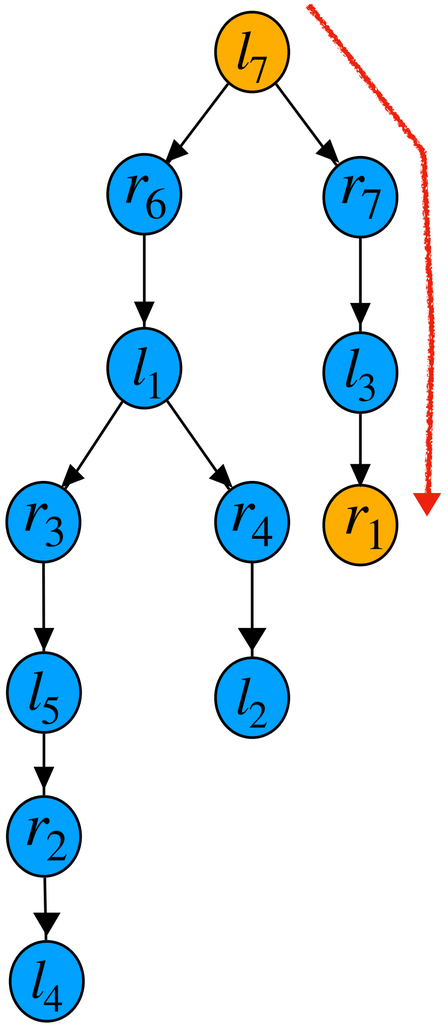


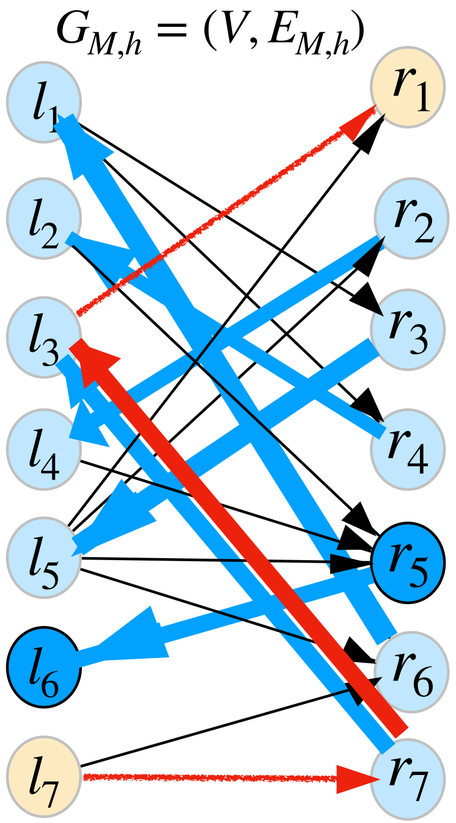




		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8





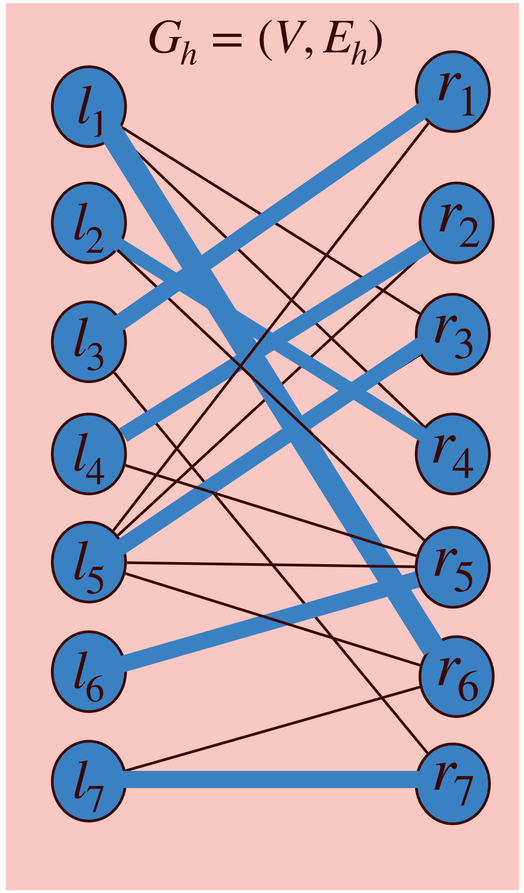


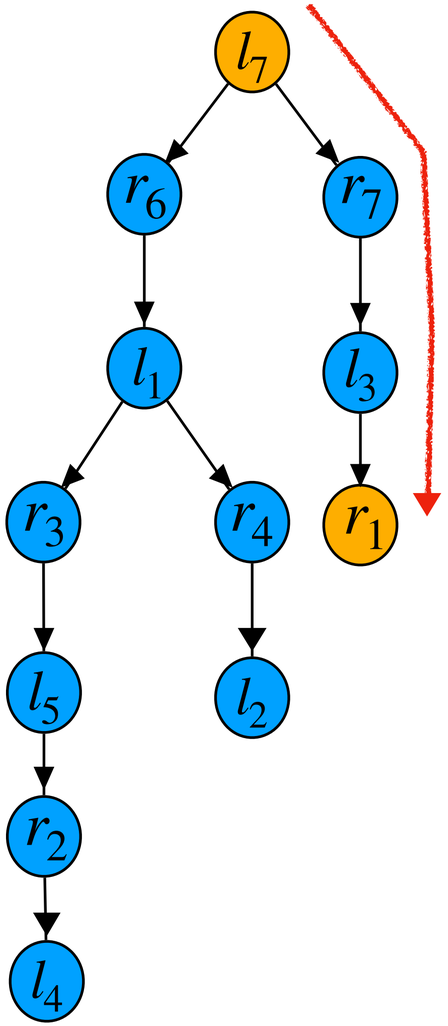
Perfect matching!

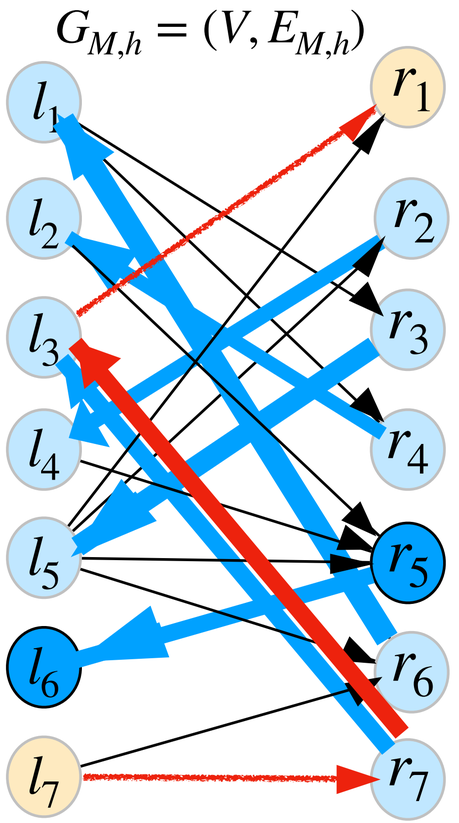
maximum weight of perfect matching

$= 9 + 12 + 11 + 9 + 5 + 11 + 8 \ = 65$

		r_1	r_2	r_3	r_4	r_5	r_6	r_7
	h	0	4	3	3	0	2	4
l_1	7	4	10	10	10	2	9	3
l_2	9	6	8	5	12	9	7	2
l_3	11	11	9	6	7	9	5	15
l_4	5	3	9	6	7	5	6	3
l_5	2	2	6	5	3	2	4	2
l_6	11	10	8	11	4	11	2	11
l_7	4	3	4	5	4	3	6	8







Perfect matching!

maximum weight of perfect matching

$= 9 + 12 + 11 + 9 + 5 + 11 + 8 = 65$

sum of all vertex labels

$= (7 + 9 + 11 + 5 + 2 + 11 + 4) + (0 + 4 + 3 + 3 + 0 + 2 + 4)$

$= 65$

Basic idea of Hungarian Algorithm:

1. Start with any feasible vertex labeling h and any matching M in the equality subgraph G_h.
2. Keep doing this until we find a perfect matching M in an equality subgraph G_h :

   If there is an M-augmenting path P, update the matching to $M \oplus P$.

   If there is no M-augmenting path, update h, then update G_h accordingly.

Time complexity: $O(n^4)$, where $n = |L| = |R| = \frac{|V|}{2}$

The algorithm can be refined to have time complexity $O(n^3)$

Quiz questions:

1. What is the main idea of the Hungarian Algorithm?

2. How do we know that the Hungarian Algorithm outputs an optimal solution?

3. How did we find the time complexity of the Hungarian Algorithm?