

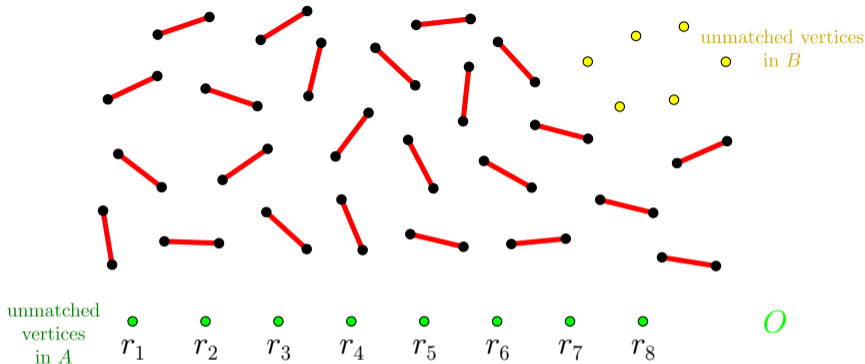
Matching algorithms

(Hungarian method + blossom algorithm)

Graph theory

University of Szeged

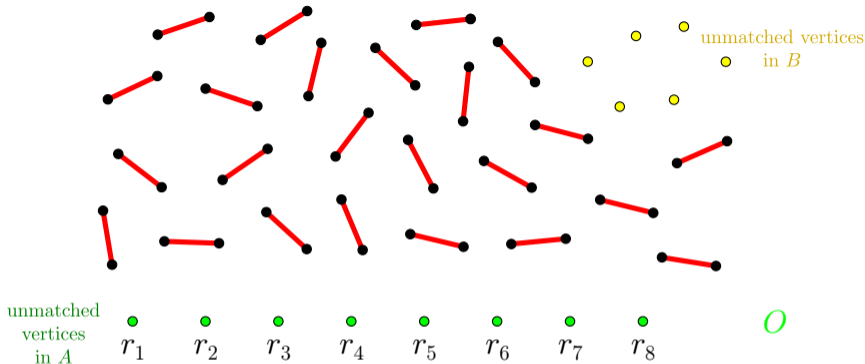
Szeged, 2019.



HUNGARIAN METHOD

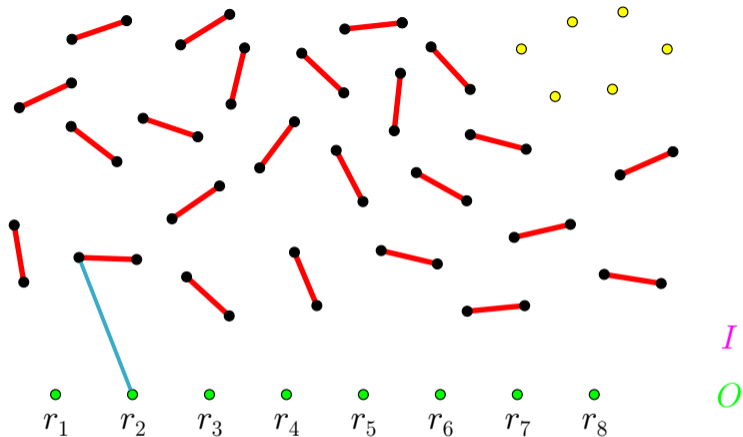
INPUT: A **bipartite** graph G (with color classes A and B) and a matching M in G which does not cover A .

OUTPUT: An augmenting path for M , or „ M is of maximum size”.

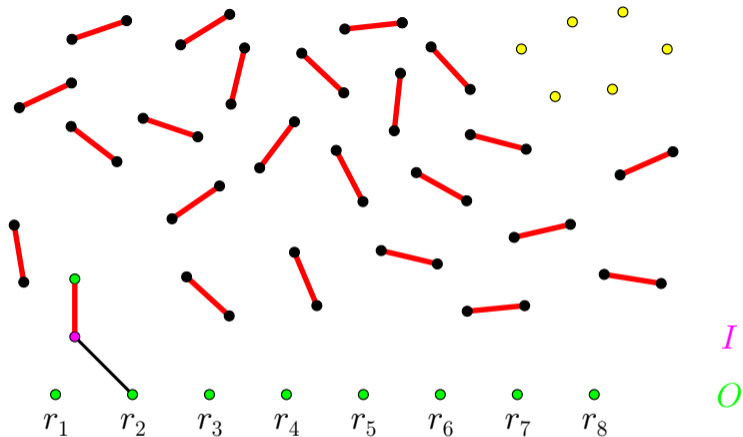


Sketch of the algorithm: Starting from the unmatched points of A (as roots), build a forest from the alternating paths of G . The forest is constructed by a „greedy method”, which will be discussed on the next slide.

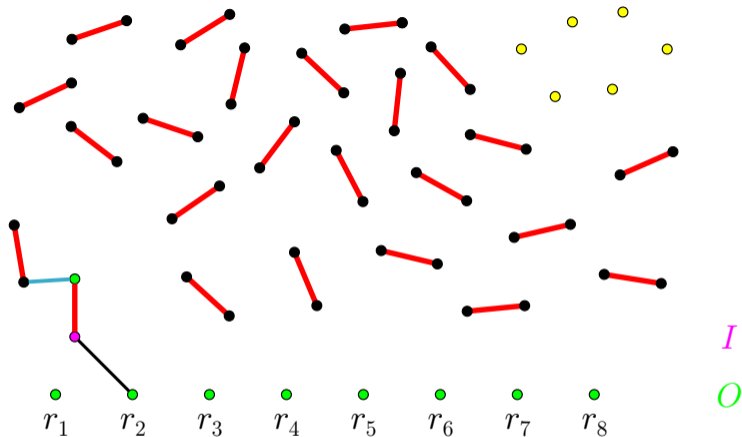
We define two sets of vertices, the sets of inner and outer vertices, I and O . Initially, $O := \{r_1, \dots, r_8\}$ and $I := \emptyset$.



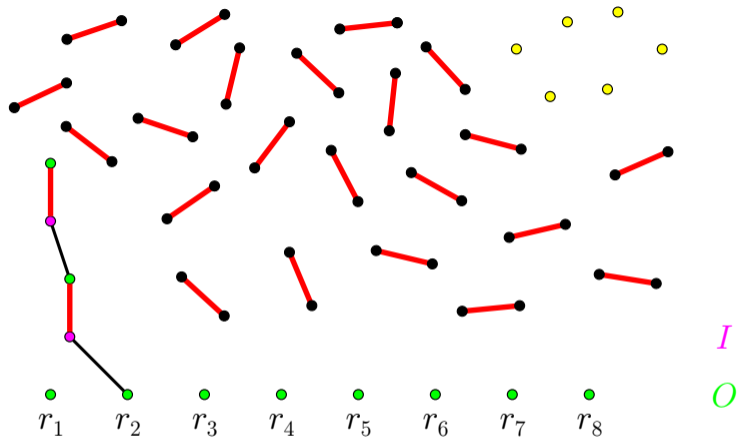
Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv'). v is designated as **inner** vertex, v' is designated as **outer** vertex.



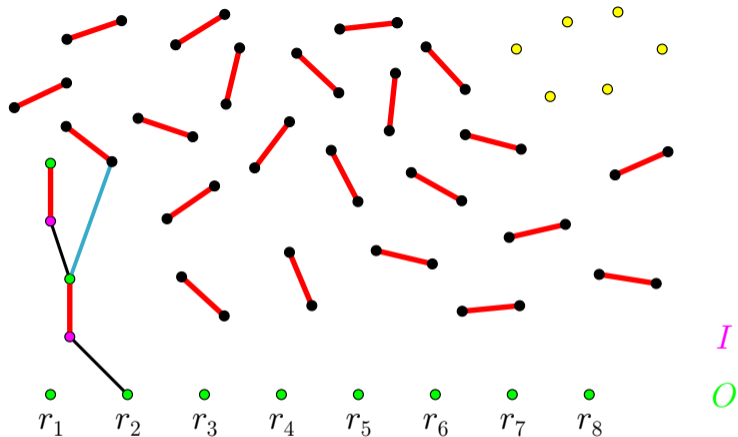
Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv'). v is designated as **inner** vertex, v' is designated as **outer** vertex.



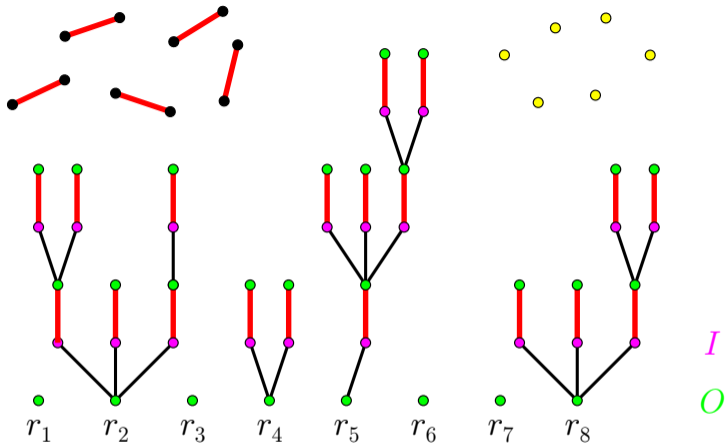
Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv'). v is designated as **inner** vertex, v' is designated as **outer** vertex.



Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv'). v is designated as **inner** vertex, v' is designated as **outer** vertex.

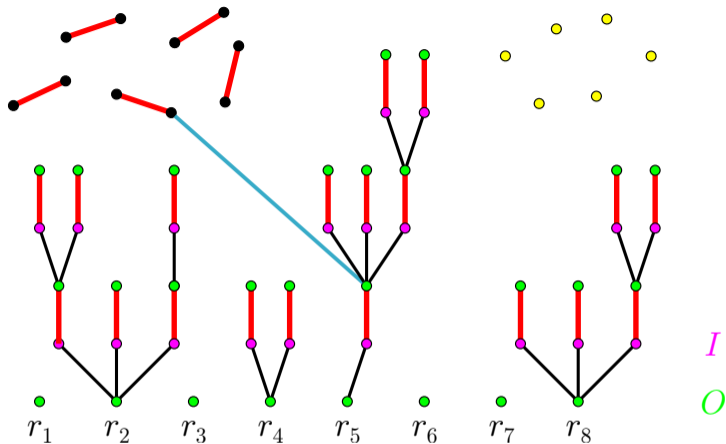


Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv'). v is designated as **inner** vertex, v' is designated as **outer** vertex.



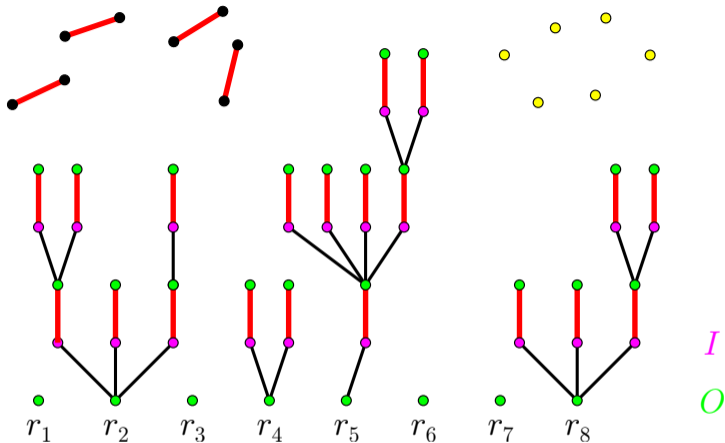
Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv').

Remark. It is always true that $O \subseteq A$ and $I \subseteq B$.



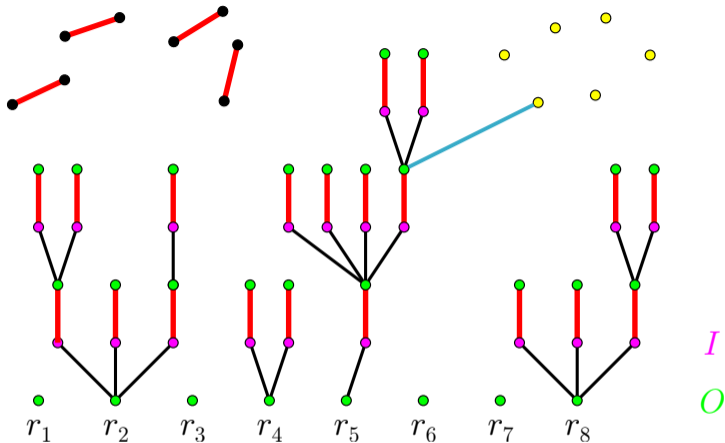
Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv').

Remark. It is always true that $O \subseteq A$ and $I \subseteq B$.

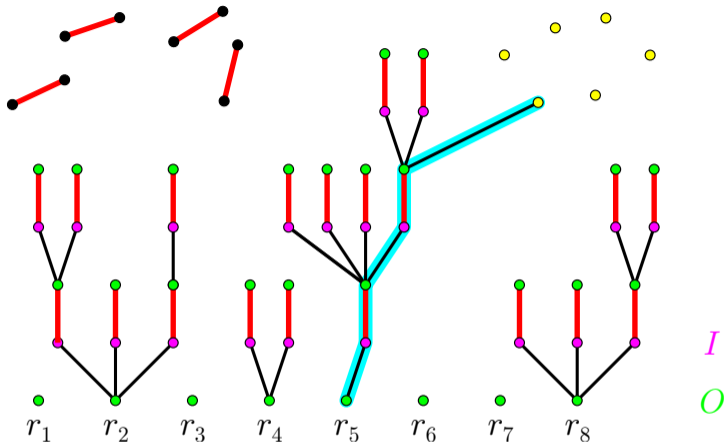


Greedy expansion step: If some **outer** vertex u is adjacent to some **matched** vertex v outside the forest, then add v to the forest, together with its pair v' in M (and edges uv and vv').

Remark. It is always true that $O \subseteq A$ and $I \subseteq B$.

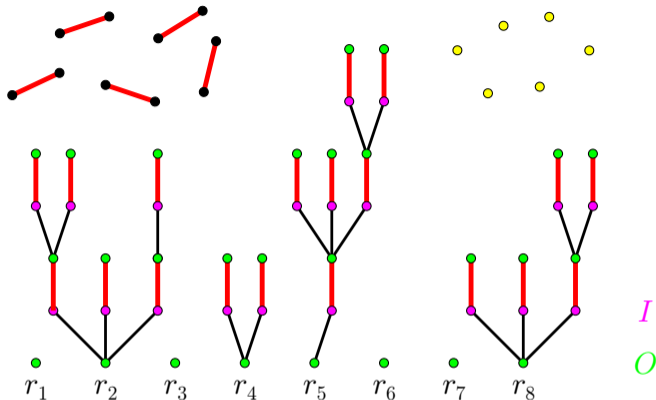


Successful search: If some **outer** vertex is adjacent to some **unmatched** vertex, then we found an augmenting path. STOP.



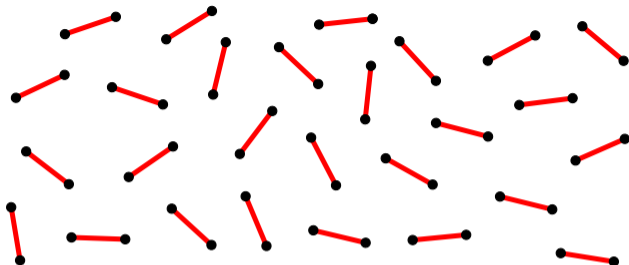
Successful search: If some **outer** vertex is adjacent to some **unmatched** vertex, then we found an augmenting path. STOP.

OUTPUT: The obtained augmenting path.



Unsuccessful search: If no augmenting path can be found, and the forest cannot be extended, i.e. if every **outer** vertex is adjacent to only (inner) vertices of the forest, then STOP.

OUTPUT: „ M is of maximum size, i.e. $\nu(G) = |M|$. This is justified by the König blocking set O .”



unmatched
vertices

r_1

r_2

r_3

r_4

r_5

r_6

r_7

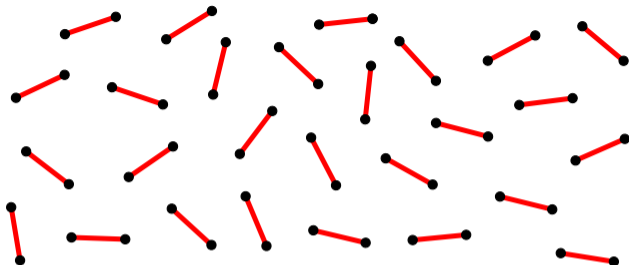
r_8

O

BLOSSOM ALGORITHM

INPUT: An arbitrary graph G + a non-perfect matching M in G .

OUTPUT: An augmenting path for M , or „ M is of max. size”.



unmatched
vertices

r_1

r_2

r_3

r_4

r_5

r_6

r_7

r_8

O

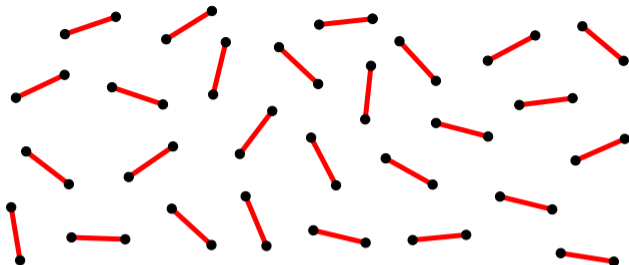
BLOSSOM ALGORITHM

INPUT: An arbitrary graph G + a non-perfect matching M in G .

OUTPUT: An augmenting path for M , or „ M is of max. size”.

Now the roots of the „forest of alternating paths” are **all** the unmatched vertices in G .

So there will be no unmatched vertices outside the forest!



unmatched
vertices

r_1

r_2

r_3

r_4

r_5

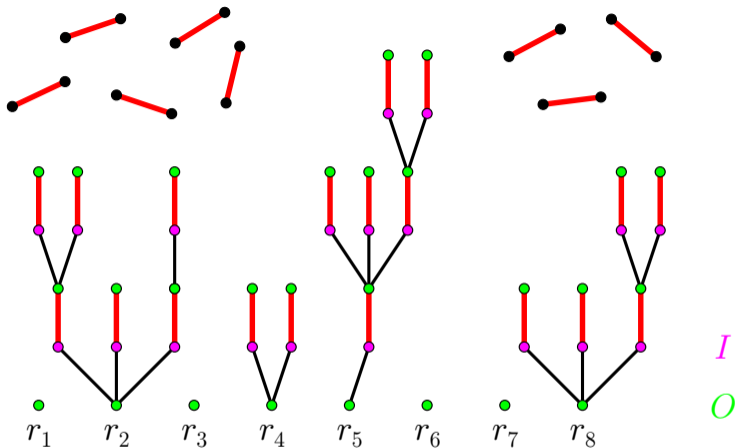
r_6

r_7

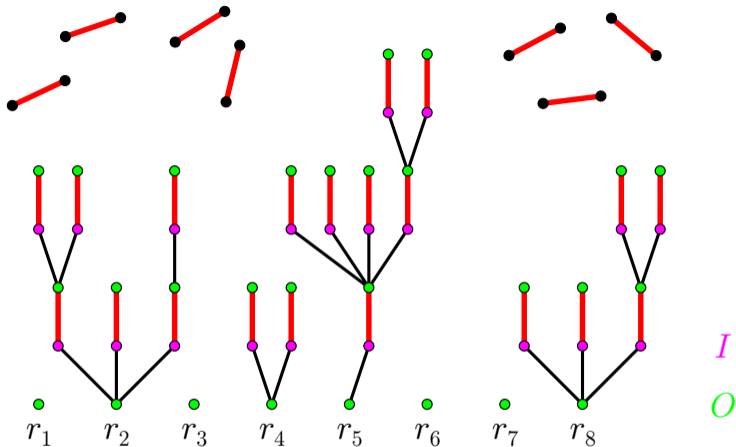
r_8

O

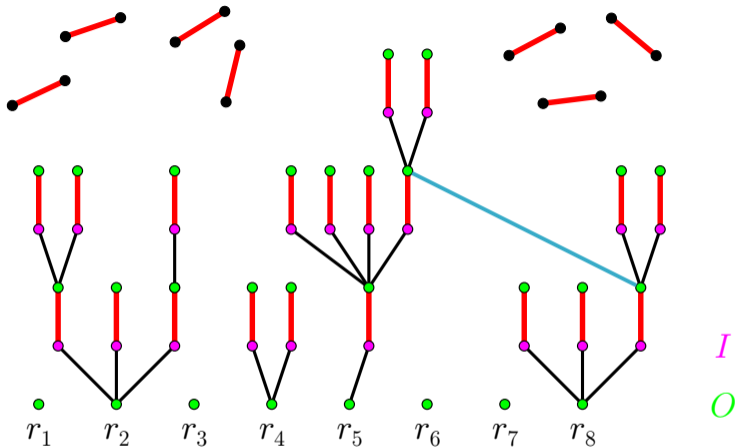
On the next slides we discuss the 3 possible types of steps of blossom algorithm. The „default” step is the greedy expansion step seen in Hungarian method (Type 1).



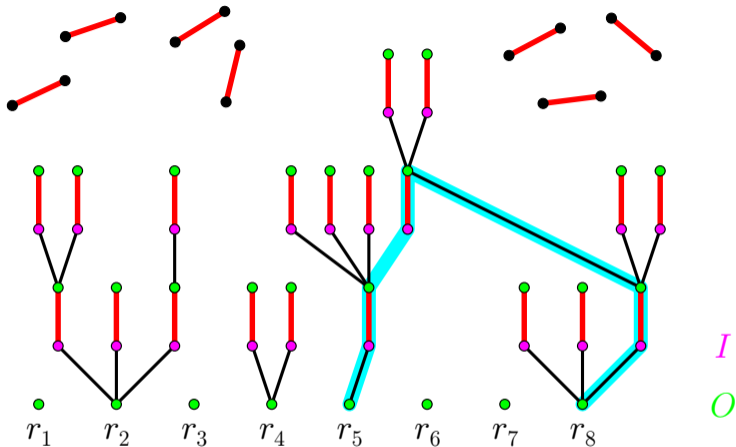
The „default” step is the greedy expansion step seen in Hungarian method (Type 1). For example, starting from the roots, after 19 greedy expansion steps we might obtain the forest seen in the figure.



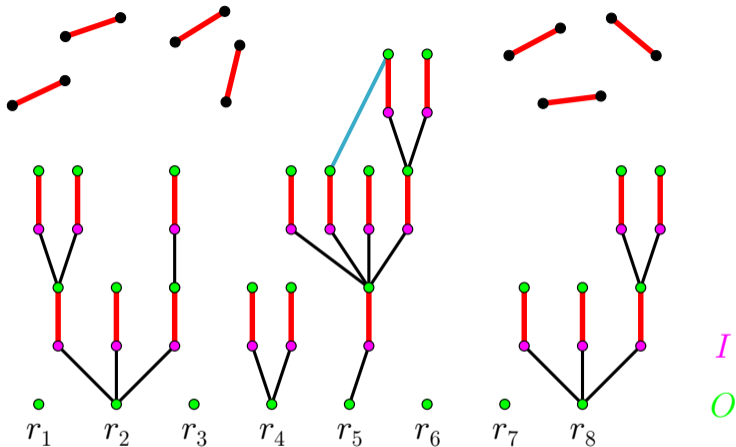
Type 1 (greedy expansion): If some **outer** vertex u is adjacent to some vertex v **outside the forest**, then extend the forest with v together with its pair in M , in the same way as in the bipartite case (including the definition of inner and outer vertices).



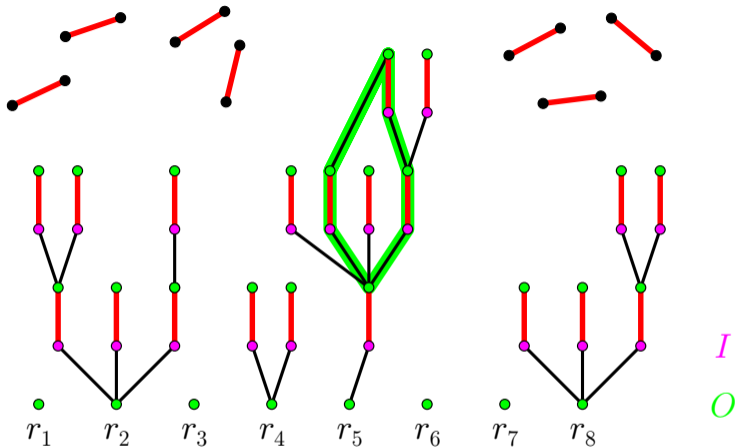
Type 2 (successful search): If some **outer** vertex is adjacent to some **outer** vertex of an **other** component of the forest, then an augmenting path is found (in the actual graph, cf. Type 3). STOP.



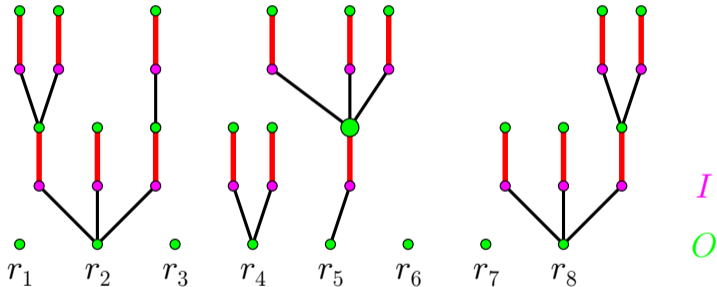
Type 2 (successful search): If some **outer** vertex is adjacent to some **outer** vertex of an **other** component of the forest, then an augmenting path is found (in the actual graph, cf. Type 3). STOP.



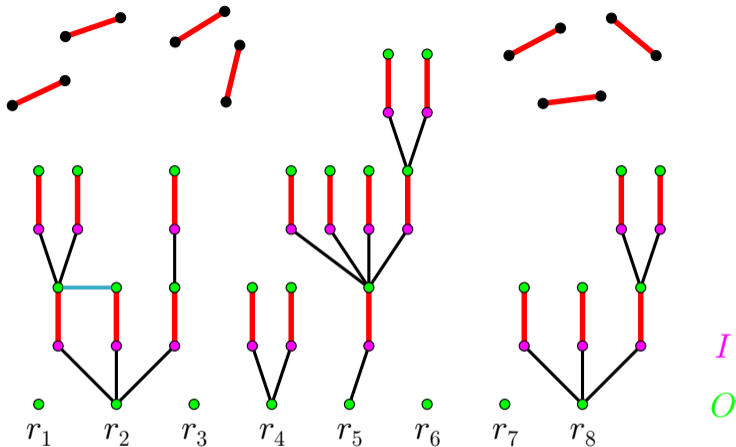
Type 3 (cycle contraction): If there is an edge between two **outer** vertices **in the same component** of the forest, then contract the corresponding odd cycle, and continue with the obtained graph. The contracted cycle becomes an **outer** vertex.



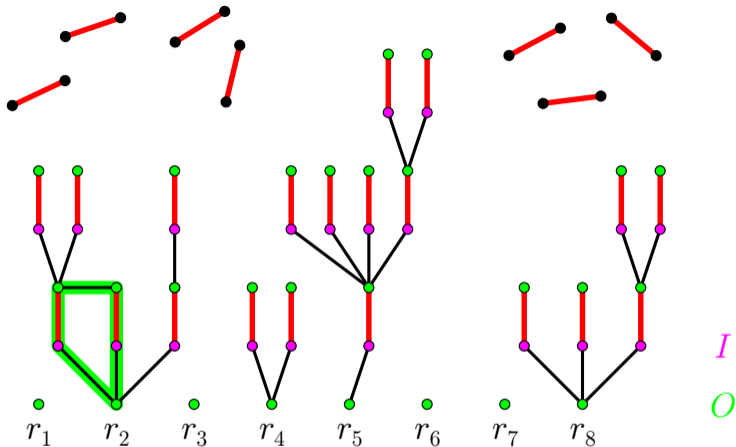
Type 3 (cycle contraction): If there is an edge between two **outer** vertices **in the same component** of the forest, then contract the corresponding odd cycle, and continue with the obtained graph. The contracted cycle becomes an **outer** vertex.



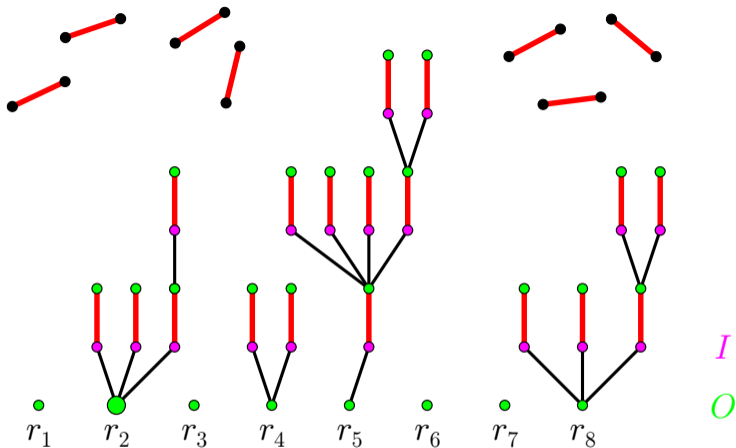
Type 3 (cycle contraction): If there is an edge between two **outer** vertices **in the same component** of the forest, then contract the corresponding odd cycle, and continue with the obtained graph. The contracted cycle becomes an **outer** vertex.



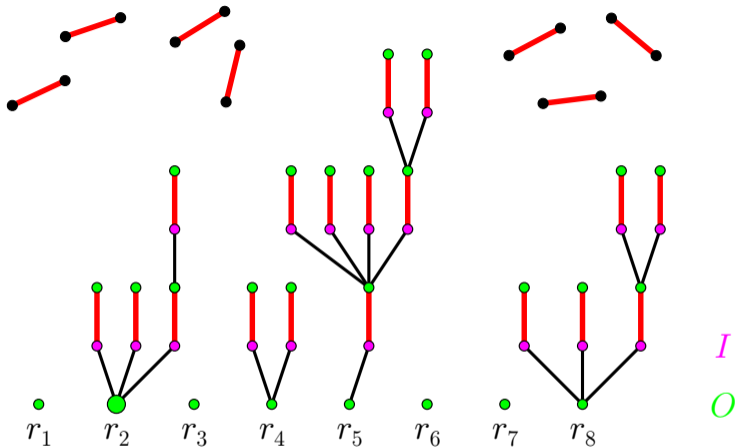
Type 3 (another example): If there is an edge between two **outer** vertices in the same component of the forest, then contract the corresponding odd cycle, and continue with the obtained graph. The contracted cycle becomes an **outer** vertex.



Type 3 (another example): If there is an edge between two **outer** vertices in the same component of the forest, then contract the corresponding odd cycle, and continue with the obtained graph. The contracted cycle becomes an **outer** vertex.



Type 3 (another example): If there is an edge between two **outer** vertices **in the same component** of the forest, then contract the corresponding odd cycle, and continue with the obtained graph. The contracted cycle becomes an **outer** vertex.



Remark: Steps of types 1-3 can be performed in any order. However, cycle contraction is complicated, so in practice a step of type 3 is recommended to perform only if the other two types of steps cannot be performed (and then try with types 1-2 again).

It may happen, that the contraction step must be performed many times before getting stuck.

It may happen, that the contraction step must be performed many times before getting stuck.

The two possible outcomes:

- 1. Successful search:** If we found an augmenting path in the (possibly) contracted actual graph (see Type 2), then this augmenting path can be transformed back to an augmenting path of the original graph G and matching M .
OUTPUT: The obtained augmenting path in G .

It may happen, that the contraction step must be performed many times before getting stuck.

The two possible outcomes:

1. Successful search: If we found an augmenting path in the (possibly) contracted actual graph (see Type 2), then this augmenting path can be transformed back to an augmenting path of the original graph G and matching M .

OUTPUT: The obtained augmenting path in G .

2. Unsuccessful search: If no augmenting path was found, and we are in stuck, because all **outer** vertices have **inner** neighbors only, then the original matching M is of maximum size in the original graph* G , so $\nu(G) = |M|$.

OUTPUT: „The matching M has maximum size in G .“

The two possible outcomes:

1. Successful search: If we found an augmenting path in the (possibly) contracted actual graph (see Type 2), then this augmenting path can be transformed back to an augmenting path of the original graph G and matching M .

OUTPUT: The obtained augmenting path in G .

2. Unsuccessful search: If no augmenting path was found, and we are in stuck, because all **outer** vertices have **inner** neighbors only, then the original matching M is of maximum size in the original graph* G , so $\nu(G) = |M|$.

OUTPUT: „The matching M has maximum size in G .“

* In case of unsuccessful search, the “preimage” of I in the original graph G is a Tutte-blocking set, which proves that $\nu(G) \leq |M|$. (Why?) This TBS can be also added to the OUTPUT as proof.

The blossom algorithm is fast (polynomial time), so it can be used in practice. If we start from an arbitrary matching of G (e.g. from one arbitrary edge, considered as trivial matching), then by performing the blossom algorithm repeatedly, we always get a larger size matching, until we reach to a perfect matching or a maximal size non-perfect matching (in case of unsuccessful search). So **the value of $\nu(G)$ can be determined efficiently.**

Moreover, not only the value of $\nu(G)$ is calculated in this way, but the algorithm constructs a maximum size matching with a Tutte blocking set proving the maximality. So the correctness of the algorithm can be verified without knowing/understanding the algorithm itself!

In case of bipartite graphs the same holds for the Hungarian method.