Paths, Trees and Flowers

J. Edmonds, 1965

Anestis Tsakiris

atsakiris@di.uoa.gr

MSc ALMA — Algorithms, Logic and Discrete Mathematics

Algorithms

December 11th, 2025

Introduction to the problem

Graph

A **graph** G is a structure consisting of:

- a finite set of elements called vertices and
- a finite set of elements called edges, such that each edge meets exactly two vertices, called the end-points of the edge.

Matching

Given a graph $G = (V, E)$, a **matching** in G is a subset of its edges $M \subseteq E$ such that no two meet the same vertex.

- In other words: an independent edge set of G.

Exposed vertex

For a pair (G, M), a vertex is called **exposed** if it meets no edge of M.

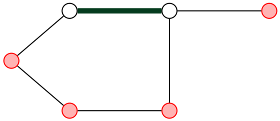
Introduction to the problem

Maximal matching

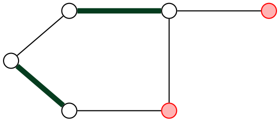
A **maximal matching** M of graph G is a matching that is not a subset of any other matching.

Maximum matching

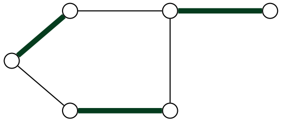
A **maximum matching** M of graph G is a matching of maximum cardinality.



Matching



Maximal matching



Maximum matching

Introduction to the problem

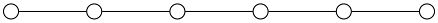
The problem: Given a graph G, we must find a maximum matching M of G.

- Note: We will assume G is connected; if not, then the task simply becomes finding a maximum matching in each of its connected components and adding them together.

Augmenting paths

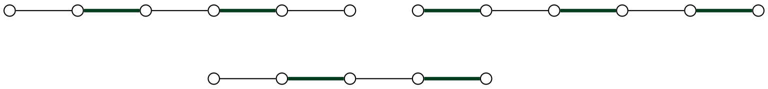
Path

A **path** P in graph G is either a simple vertex or a connected subgraph whose two end-points each meet one edge of P and whose other vertices each meet two edges of P.



Alternating path

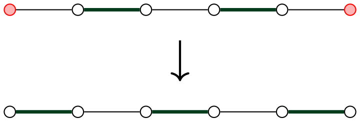
An **alternating path** in (G, M) is a path P such that one edge in $M \cap P$ and one edge in $\overline{M} \cap P$ meets each vertex of P, except for the end-points.



Augmenting paths

Augmenting path

An **augmenting path** is an alternating path A in (G, M) that joins two exposed vertices.



Augmenting paths

Lemma

For any two matchings M_1 and M_2 in G, the components of the subgraph formed by $M_1 \oplus M_2$ are paths and cycles which are alternating for (G, M_1) and for (G, M_2). Each path end-point is exposed for either M_1 or M_2.

Proof:

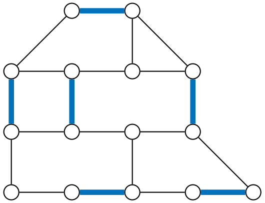
- A vertex of G meets no more than one edge, each, of M_1 and M_2, therefore no more than two edges of $M_1 \oplus M_2$, one in $M_1 \cap \overline{M_2}$ and one in $M_2 \cap \overline{M_1}$.
- An end-point v of a path in $M_1 \oplus M_2$ meeting an end-edge in (let's say) $M_1 \cap \overline{M_2}$ meets no other edge of M_1.
- Hence, if an edge of M_2 meets v, it does not belong to M_1 and so it belongs to $M_1 \oplus M_2$. But then v is not an end-point.
- Therefore, v is exposed for M_2.

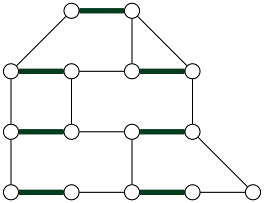
Augmenting paths

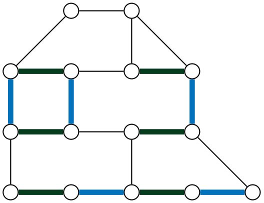
Berge's Theorem

A matching M in G is of maximum cardinality if and only if (G, M) contains no augmenting paths.

Proof:
$(\Rightarrow)$ If there is an augmenting path A, we can "flip" it, removing k edges from the matching and adding $k + 1$ new ones. Therefore, we obtain a matching larger than M.
$(\Leftarrow)$ If M is not maximum, there exists a matching M^* larger than M. Some component of $M \oplus M^*$ must contain more M^*-edges than M-edges. For this to happen, the component must be a path with M^*-edges at both its ends, therefore it is an augmenting path for (G, M).


Matching M ($|M| = 6$)


Better matching M^* ($|M^*| = 7$)


$M \oplus M^*$

Kuhn's algorithm (1955)

Kuhn's algorithm for bipartite graphs (Hungarian method)

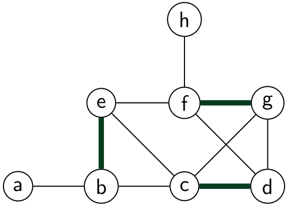
Given a bipartite graph G and an arbitrary matching M, we look for augmenting paths in G in the following way.

- Starting from any exposed vertex v, we explore the vertices of G (i.e. using DFS/BFS), specifically switching between unmatched and matched edges (in essence we build an alternating tree with v as root).
- When we encounter an exposed vertex u (obviously via an unmatched edge), we have found a (v, u)-augmenting path. We flip the path's edges in and out of the matching.
- If we hit a dead-end, we stop exploring from this root and pick a different exposed vertex as root of the alternating tree (without revisiting any vertices we already visited during this attempt).
- We repeat until no more augmenting paths can be found starting from any exposed vertex.

- Since there are no more augmenting paths in the graph, we conclude (by Berge's Theorem) that the matching is of maximum cardinality.
- This works in bipartite graphs, because they contain *no odd cycles*: If during the graph traversal we loop back to an already explored vertex, the path will be of even length and therefore nothing can be done there to improve the matching (a local dead end).

Kuhn's algorithm (1955)

- Time complexity: $O(n^3)$.
- When attempting to augment a path starting from exposed vertex v, we essentially alternate between vertices of even distance from v (which we'll call outer vertices from now on) and vertices of odd distance from v (which we'll call inner). When we have reached an exposed vertex of odd distance from v, then we can conclude that we have found an augmenting path.
  - The issue with this in general graphs: Some vertices will appear as both outer and inner during the graph traversal. Even if we allow this to be possible, it could possibly make the algorithm reach exponential time complexity.
  - The issue was already known. Edmond found a way to solve it by "ignoring" it.



Alternating trees

Tree

Equivalent definitions of a **tree**:

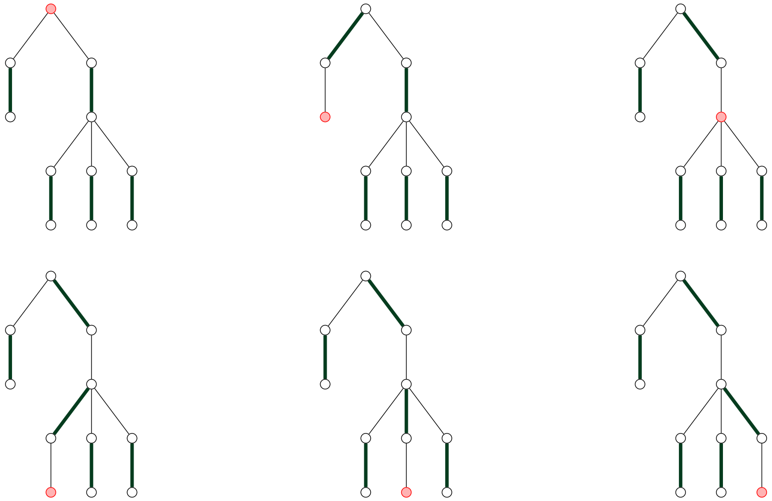
- A connected graph with no cycles.
- A graph T every pair of whose vertices is joined by exactly one path in T
- A connected graph with one more vertex than edges.

Alternating tree

An **alternating tree** J is a tree each of whose edges joins an *inner* vertex to an *outer* vertex, where an inner vertex of J meets exactly two edges of J.

- For each outer vertex v of an alternating tree J there is a unique maximum matching of J which leaves v exposed and the only exposed vertex in J. Every maximum matching of J is one of these.

Alternating trees



Alternating trees

Planted tree

A **planted tree** $J = J(M)$ of G for a matching M is an alternating tree in G such that $M \cap J$ is a maximum matching of J and such that the vertex r in J which is exposed for $M \cap J$ is also exposed for M. That is, all matching edges which meet J are in J. Vertex r is called the root of J.

- In planted tree $J(M)$, every alternating path $P(M)$, which has outer vertex v and the matching edge to v at one of its ends, is a subpath of the alternating path $P_v(M)$ in $J(M)$ which joins v to the root.

Augmenting tree

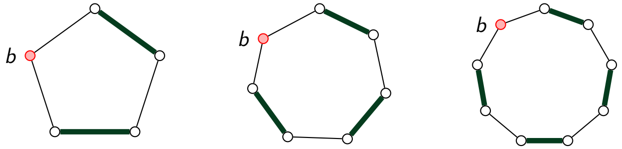
An **augmenting tree** $J_A = J_A(M)$ in (G, M) is a planted tree $J(M)$ plus an edge e of G such that one end-point of e is an outer vertex v_1 of J and the other end-point v_2 is exposed and not in J.

- The path in J_A that joins v_2 to the root of J is an augmenting path.

Blossoms and flowers

Blossom

For each vertex b of an odd cycle B there is a unique maximum matching of B which leaves b exposed. A **blossom** B in (G, M) is an odd cycle for which $M \cap B$ is a maximum matching in B with say vertex b exposed for $M \cap B$.

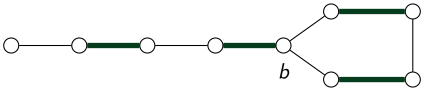


Stem

A **stem** in (G,M) is either an exposed vertex or an alternating path with an exposed vertex at one end (root) and a matching edge at the other end (tip).

Flower

A **flower** F consists of a blossom and a stem which intersect only at the tip of the stem (the vertex b).



Blossoms and flowers

Flowered tree

A **flowered tree** J_F in (G, M) is a planted tree J plus an edge e of G which joins a pair of outer vertices of J.

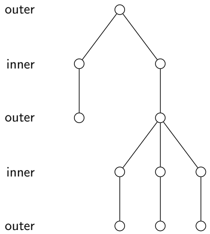
- The union of e and the two paths which join its outer-vertex end-points to the root of J is a flower F.

Hungarian tree

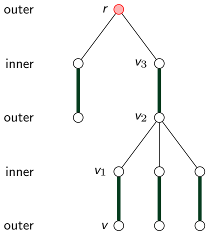
A **Hungarian tree** H in graph G is an alternating tree whose outer vertices are joined by edges of G only to its inner vertices.

- Essentially, an alternating tree that cannot be extended because no outer vertices have any edges that could lead to an augmenting or a flowered tree.

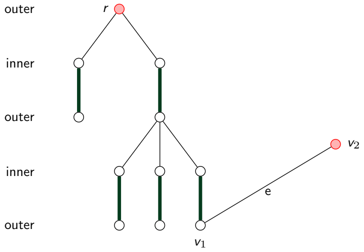
Examples of types of ”trees”



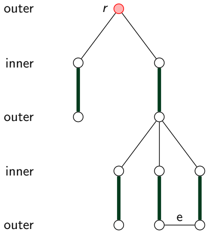
(a) Alternating tree



(b) Planted tree



(c) Augmenting tree

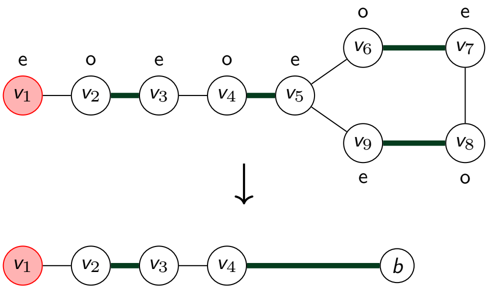


(d) Flowered tree

Blossoms and flowers

Main idea: We use the principles of Kuhn's algorithm for the general graph case too. However, when we identify a blossom B, we shrink it to a pseudo-vertex b and run the algorithm on the new graph $G' = G/B$. Any edges that connected to a vertex of B from outside will now connect to b.

- How to locate a blossom: When scanning for an augmenting path, we alternate between outer vertices (even distance from starting vertex) and inner vertices (odd distance). If we reach an outer vertex which has a neighbor that has also been marked as outer, then we have located a blossom.



Blossoms and flowers

Blossom Shrinking Theorem

$G' = G/B$ contains an augmenting path if and only if G does.

Proof:

($\Leftarrow$) Suppose there is an augmenting path P for (G, M).

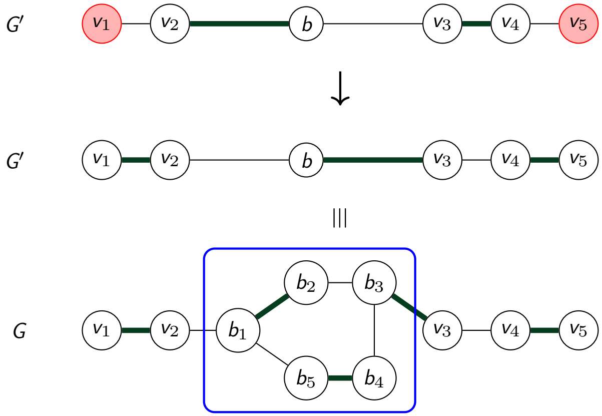
- If P does not pass through B, then P exists also in G'.
- If P passes through B (let's say entering B at v_{in} and exiting at v_{out}), it must include an even-length (v_{in}, v_{out})-path inside B that is alternating for M. After shrinkage, B becomes a single vertex b and the resulting path will still be alternating and have exposed vertices as end-points.

($\Rightarrow$) Suppose there is an augmenting path P' for (G', M').

- If P' does not pass through b, then P' exists also in G.
- If P' passes through b (e.g. $\ldots - v_1 - b - v_2 - \ldots$), then we can create an even-length alternating (v_1, v_2)-path inside B (in G) that still manages a maximum matching inside the blossom. Therefore, after expanding the blossom, we will still have an alternating path with exposed end-points in G.

Blossoms and flowers

- When an augmenting path is found that contains pseudo-vertex b, we can expand the blossom, augmenting the path by having it pass through the appropriate path in the cycle.
    - For better efficiency, we can also wait until the end of the algorithm before expanding the blossom.



The main idea behind the algorithm

Claim: For a matching M in a graph G, an exposed vertex is a planted tree. Any planted tree $J(M)$ in G can be extended either to an augmenting tree, or to a flowered tree, or to a Hungarian tree (merely by looking at most once at each of the edges in G which join vertices of the final tree).

Proof: An exposed vertex satisfies the definition of a planted tree. Given a planted tree J and a set D (perhaps empty) of edges in G which are not in J but which join outer to inner vertices of J.

- If no outer vertex of J meets an edge not in $D \cup J$, then J is Hungarian.

Suppose outer vertex v_1 meets an edge $e = \{v_1, v_2\}$ that is not in $D \cup J$.

- If v_2 is an inner vertex of J, we can enlarge D by adjoining e.
- If v_2 is an outer vertex of J, then $e \cup J$ is a flowered tree.
- If v_2 is exposed and not in J, then $e \cup J$ is an augmenting tree.
- If v_2 is not exposed and not in J, the M-edge e_2 which meets v_2 is not in J (so its other end-point v_3 is not in J by the definition of planted tree). Therefore, we can extend J to a larger planted tree with new inner vertex v_2 and new outer vertex v_3 by adjoining edges e and e_2.

For any J and D, one of the above holds. Therefore, given a planted tree and by looking at any edge in G at most once, we can reach either an augmenting tree, a flowered tree, or a Hungarian tree.

The algorithm in simple terms

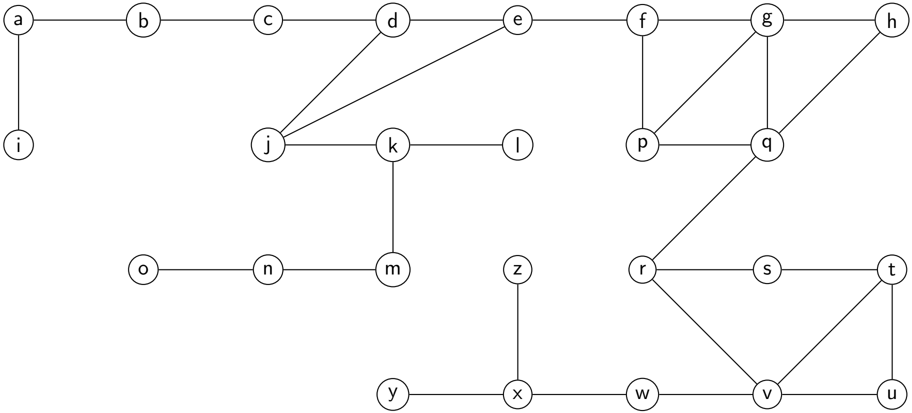
Edmond's Blossom Algorithm

Given a general graph G and a matching M (let it be $M = \emptyset$ at first), we look for augmenting paths in G in the following way.

- We start building a planted tree starting from an exposed vertex r as root (using DFS/BFS, alternating between outer-inner vertices).
    - If we identify an augmenting tree (i.e. we encounter an edge between an outer vertex and an exposed vertex outside the current tree), we "flip" its augmenting path to get a larger matching. Then we restart the search.
    - If we identify a flowered tree (i.e. we encounter an edge between two outer vertices of the current tree), we shrink its blossom B and run the algorithm on graph $G' = G/B$, using matching M' which we obtain from current M by removing the edges of M inside B.
    - If a planted tree becomes Hungarian (there are no more edges that can expand it), we start building a planted tree from a different exposed vertex, without revisiting vertices already visited during this iteration.
- If during an iteration every vertex is visited without finding an augmenting or a flowered tree, then no augmenting path exists. We expand the shrunk blossoms one by one appropriately and terminate. M is maximum.

- Since the number of possible augmentations and the number of blossoms to shrink are finite, the algorithm will terminate. Due to the Blossom Shrinking Theorem and Berge's Theorem, the algorithm will return a maximum matching for G.

Example



Time complexity

We run the algorithm on a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges.

- Algorithm makes up to $\frac{n}{2} = O(n)$ calls for augmentation.
- There are at most m edges to explore in each augmentation attempt:
  - In the case of a blossom shrinkage, there is no good guarantee of how many edges we might need to visit, therefore this might take $O(m)$.
  - In the case of adding an edge to the forest or augmenting a path, again we may need to examine each edge of the graph, therefore $O(m)$.
  - Note: In case we hit a local dead-end, the current augmentation attempt has not finished. We can start growing an alternating tree from another exposed vertex without revisiting the ones we already checked.
- A blossom shrinkage reduces the number of vertices by at least 2. If we implement the algorithm such that we shrink blossoms and only expand them at the end (so that we don't accidentally re-shrink the same blossom), we will have at most $\frac{n}{2}$ shrinkings. Therefore, there may be up to $O(n)$ blossom recursive calls.

$\rightarrow$ Complexity of the original algorithm: $O(n) \cdot [O(m) \cdot O(n)] = \mathbf{O(m \cdot n^2)}$.

Later improvements of the algorithm:

- $O(n^3)$ by Gabow (1976).
- $O(m \cdot \sqrt{n})$ by Micali, Vazirani (1980).

References

- Edmonds, J. (1965). *Paths, Trees, and Flowers.* Canadian Journal of Mathematics, 17, 449–467.
- Kuhn, H. W. (1955). *The Hungarian Method for the Assignment Problem.* Naval Research Logistics Quarterly, 2(1–2), 83–97.
- Gabow, H. N. (1976). *An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs.* Journal of the ACM, 23(2), 221–234.
- Micali, S., & Vazirani, V. V. (1980). *An $O(\sqrt{|V|}\,|E|)$ Algorithm for Finding Maximum Matching in General Graphs.* Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS), 17–27.
- Shoemaker, A., & Vare, G. (2016). *Edmonds' Blossom Algorithm.* Stanford CME 323 Project Report. Available at: `https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/shoemaker_vare.pdf`

Thank you for your attention!