Finding a maximum matching

(in any graph)

To find a maximum matching in a graph $G = (V, E)$, let us start from any matching M.

It might be empty; or constructed with the greedy algorit-hm.

By Berge's theorem:

- If we can find M-extensible paths for any non-maximal M, then we can increase the matching until it becomes maximal.

By our proof of Berge's theorem:

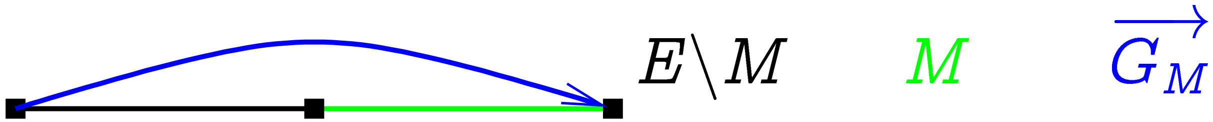
- Increasing the matching M will give us a M-extensible path.

We need to find an M-extensible path.

We are going to search it in the oriented graph $\overrightarrow{G_M}$:

$$V(\overrightarrow{G_M}) = V$$

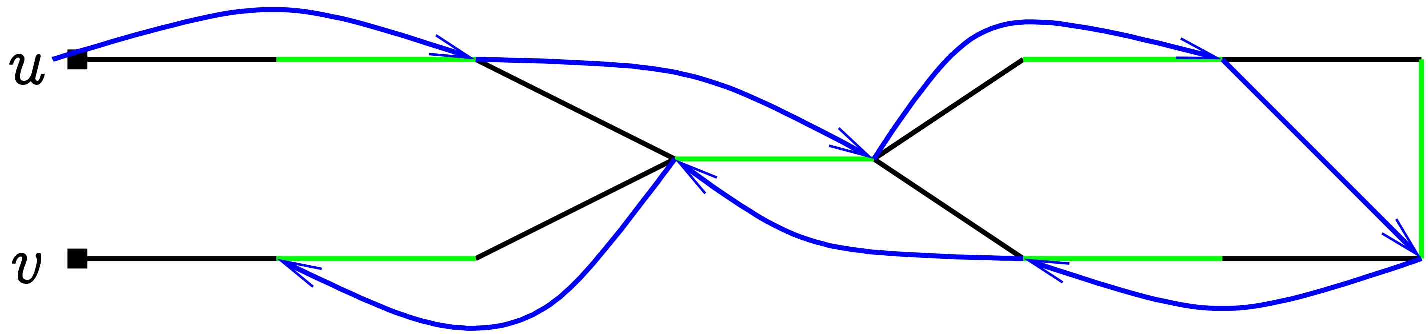
$$E(\overrightarrow{G_M}) = \{(u,w) \mid \exists v \in V : (u,v) \in E \backslash M, (v,w) \in M\} \; .$$



Let $W = \{v \in V \mid \deg_M(v) = 0\}$.

Any directed path in $\overrightarrow{G_M}$ from W to $N(W)$ corresponds to an M-extensible <u>walk</u> (not necessarily a path).

M-extensible walk (not path) from u to v:



But we need to find a path, not a walk...

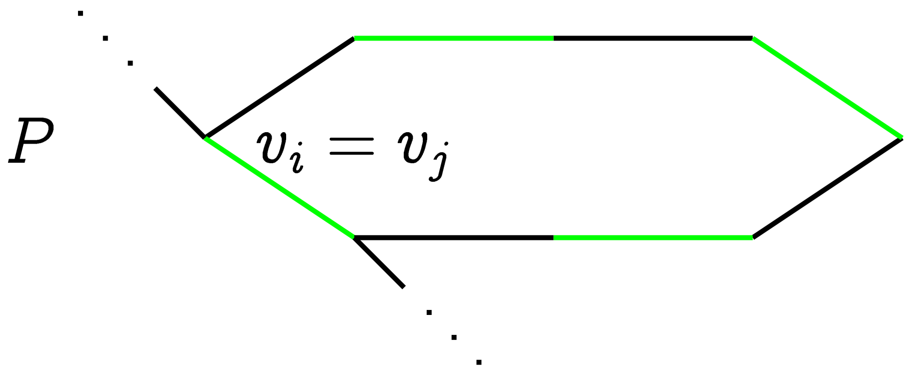
Lemma. Let $P = v_0 — v_1 — \cdots — v_m$ be a minimum-length M-extensible walk from W (i.e. $v_0 \in W$) to some $v = v_m$. One of the following holds:

- P is a path.
- There exist such $0 \leqslant i < j \leqslant m$, that
  (i) $v_i = v_j$;
  (ii) i is even, j is odd
    - meaning that $v_i — v_{i+1}$ and $v_{j-1} — v_j$ are not in M;
  (iii) $v_0, \ldots, v_{j-1}$ are all distinct.

Proof. If P is a path then the lemma holds. Assume P is not a path.

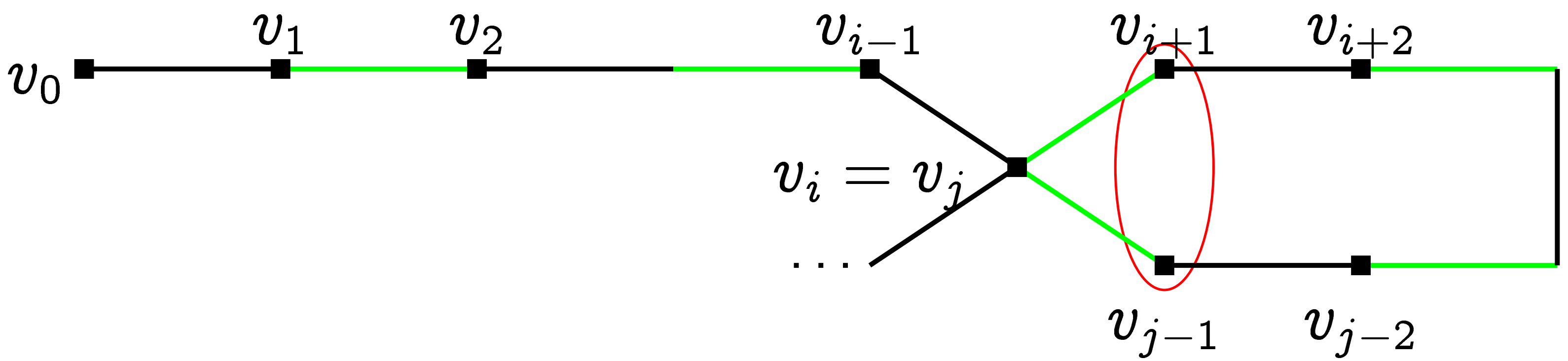
Let i, j be defined by v_j being the first vertex that coincides with some earlier v_i. This choice satisfies (i) and (iii).

If $(j - i)$ were even, then. . .



P would not be of minimum length.

If i would be odd and j would be even, then...

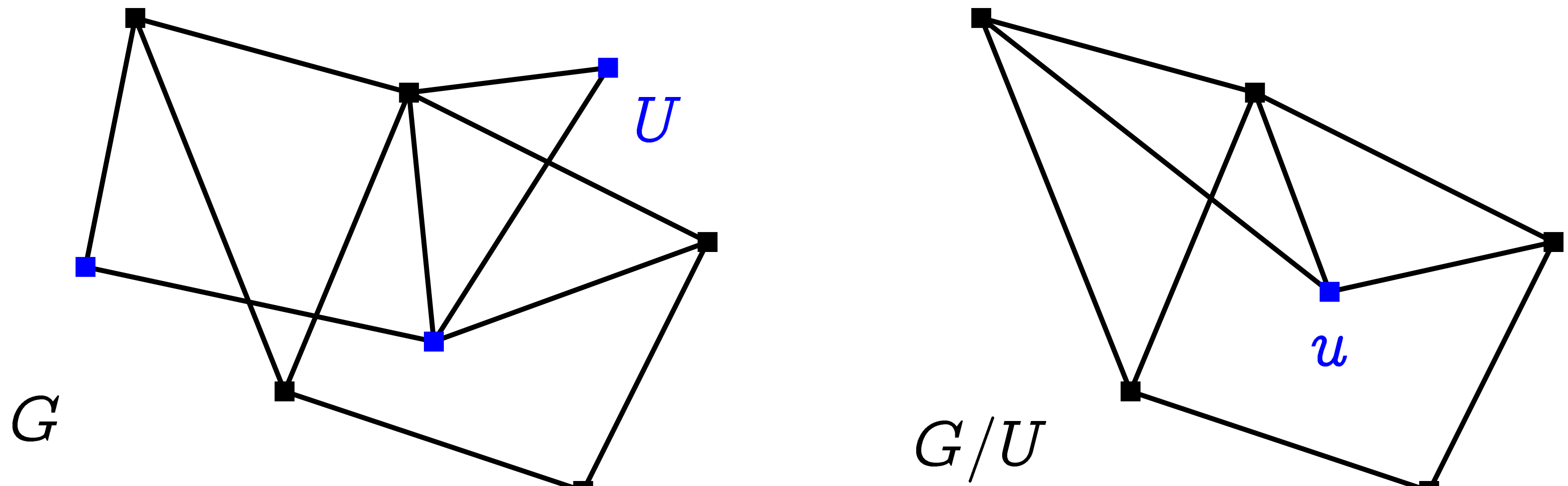


v_{i+1} would equal v_{j-1}.

This contradicts the choice of v_j. $\square$

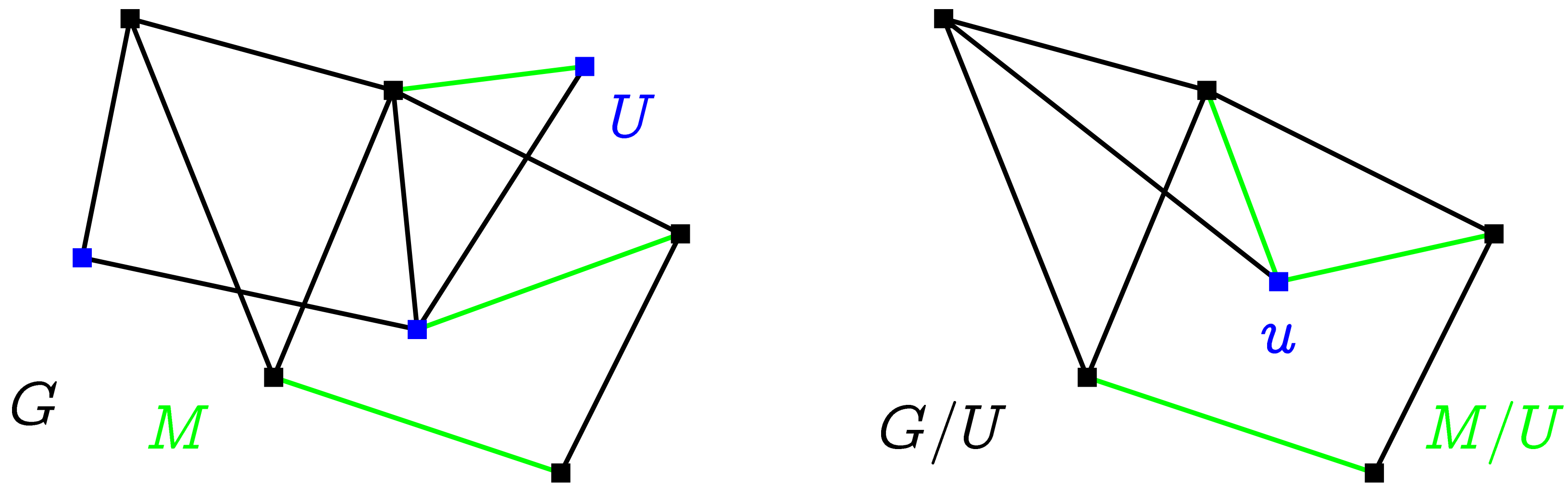
Let $G = (V, E)$ be a simple graph and $U \subseteq V$. The *contraction (kokkutõmbamine)* of U in G gives us the simple graph G/U, where

- instead of vertices of U, we have a single new vertex u;
- all neighbours of U are connected to u.



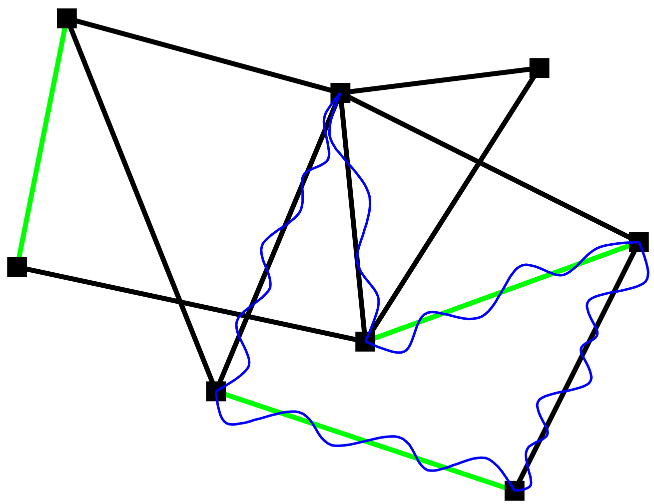
Also define:

- If $H \leqslant G$, then $G/H = G/V(H)$.
- If $M \subseteq E(G)$ ja $U \subseteq V(G)$, then M/U is the set of edges of the graph $(V(G), M)/U$.



Let M be a matching in $G = (V, E)$. A cycle $C \leqslant G$ is *M-blossom (M-õis)*, if

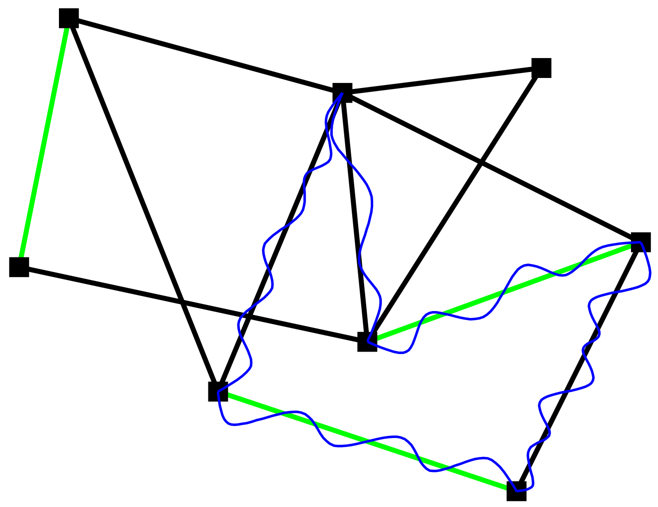
- $|V(C)| = 2k + 1$ for some $k \in \mathbb{N}$;
- $|E(C) \cap M| = k$.
- C passes through a vertex not covered by M.

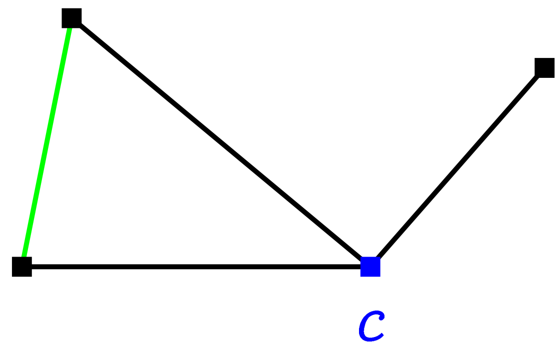


Theorem. Let M be a matching in $G = (V, E)$. Let C be an M-blossom. M is a maximal matching in G iff M/C is a maximal matching in G/C.

Proof. Let $c \in V(G/C)$ be the vertex that C was contracted to.

M/C does not cover C, because no edge in M is between $V(C)$ and $V(G) \backslash V(C)$.

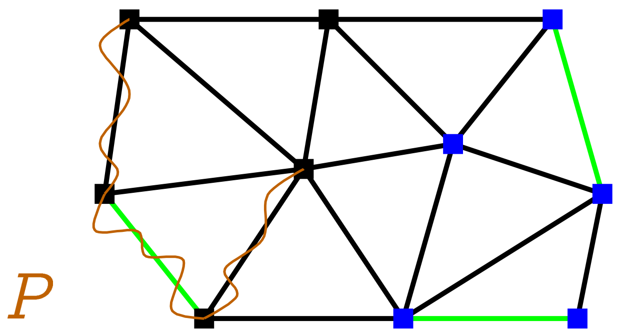


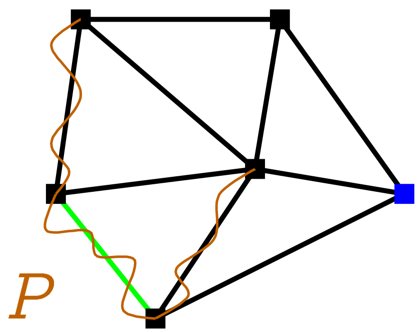


Proof by contradiction:

1\. M **not maximal** $\Rightarrow$ M/C **not maximal.**

Let P be a M-extensible path in G. If P does not intersect C, then it is a M/C-extensible path in G/C.

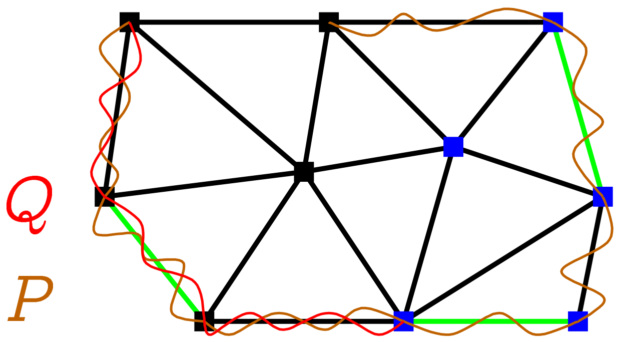


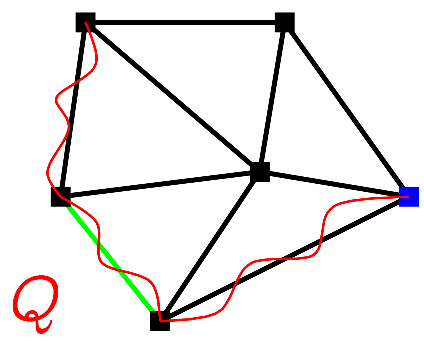


If P intersects C, then at least one of its endpoints v is outside C.

- Because C contains only one vertex not covered by M.

Let Q be a subpath of P from v to the first vertex in C. Then Q is M/C-extensible in G/C.



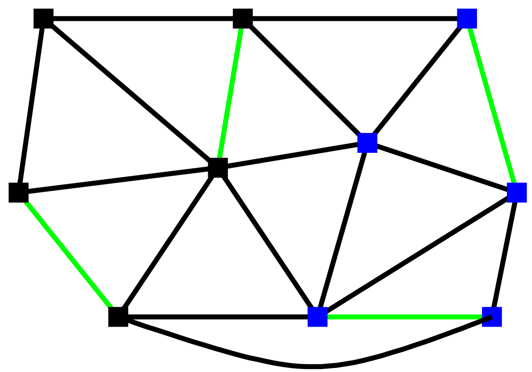


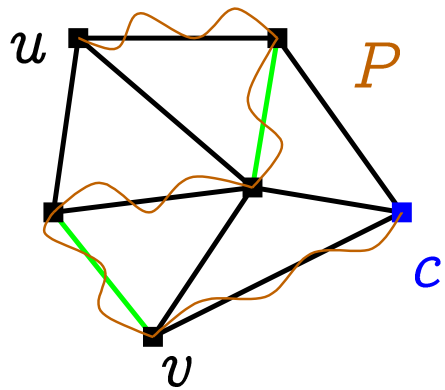
2. M/C **not maximal** $\Rightarrow$ M **not maximal.**

Let P be a M/C-extensible path in G/C. If it does not contain c, then it is also M-extensible in G.

If P contains c, then c is one of the end-vertices of P. Let

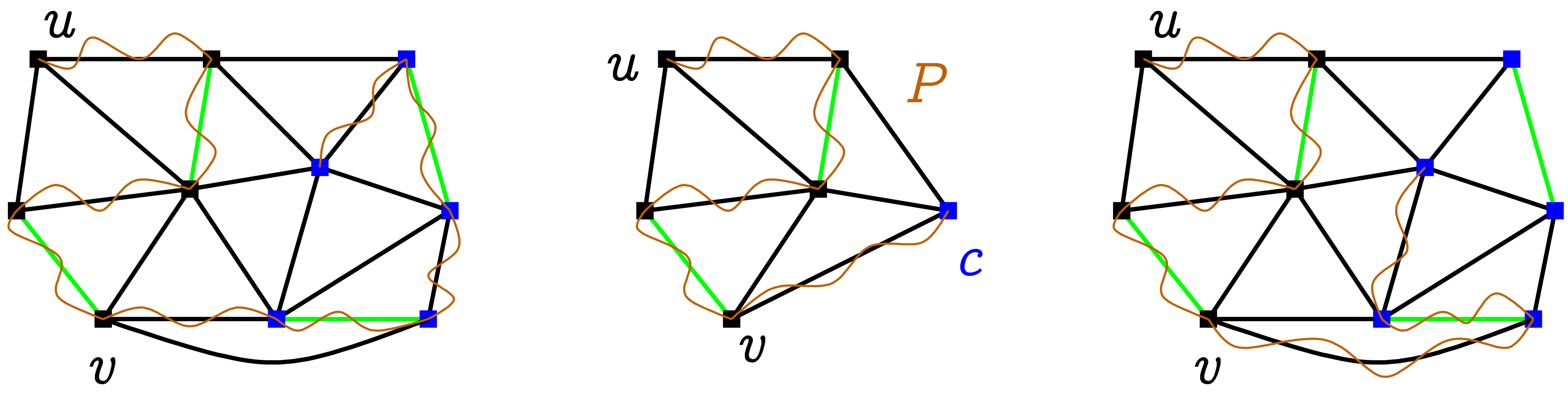
- v be c's neighbour on P;
- u be the other end-vertex of P.





Construct a M-extensible path in G by

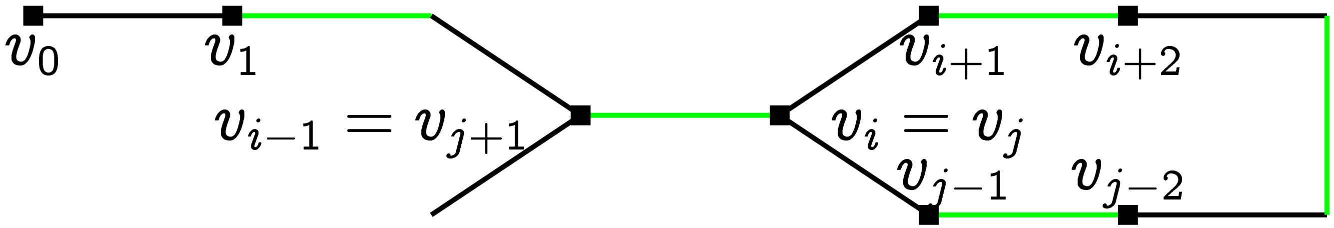
- Going from u to v along P;
- stepping from v to some vertex in C;
- going along C from that vertex to the vertex not covered by M. $\square$



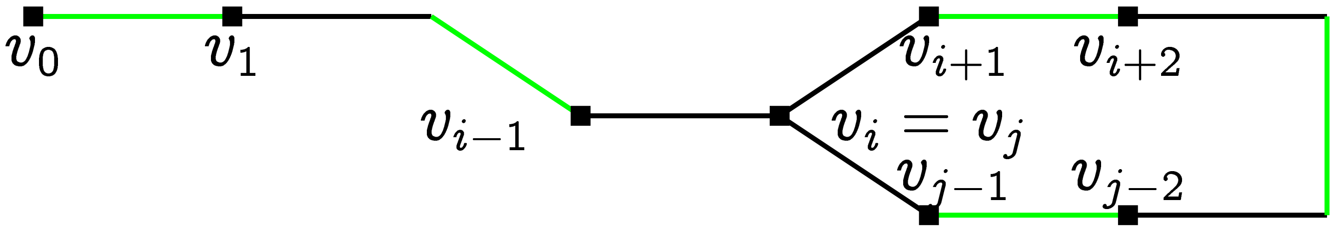
Algorithm for increasing the matching M in G by an edge:

1. Find the minimum-length M-extensible walk P from W to W.
   - Find the shortest directed path from W to $N(W)$ in $\overrightarrow{G_M}$.
     - Do a breadth-first traversal of $\overrightarrow{G_M}$.
2. If no such P exists, then M is maximal. Stop.
3. If P is a path, then return $M \triangle E(P)$.
   - $A \triangle B = (A \backslash B) \cup (B \backslash A)$.

4. If $P = v_0 — v_1 — \cdots — v_m$ is not a path then let v_j be the first vertex, such that $\exists i < j : v_i = v_j$.



5. Let $M := M \triangle \{v_0 — v_1, v_1 — v_2, \ldots, v_{i-1} — v_i\}$.



M remains a matching because only $\deg_M(v_0)$ increased.

$C = v_i — v_{i+1} — \cdots — v_j$ is a M-blossom.

6. Recursively invoke the algorithm for M/C and G/C.

7. If M/C is maximal, then M is maximal. Stop.

8. If a matching N was returned, then

   - If $\deg_N(c) = 0$, then return

$$(N \cap E(G\backslash C)) \cup (M \cap E(C)) \ .$$

- If $\deg_N(c) = 1$ then return

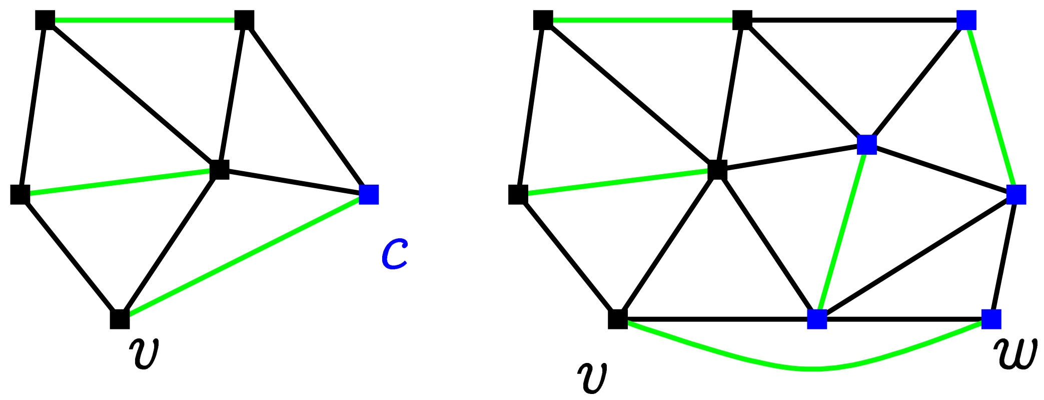
$$(N \cap E(G \backslash C)) \cup \{v — w\} \cup M_C^w$$

where

  – v is the vertex, such that $\{v, c\} \in N$;

  – $w \in V(C)$ is a neighbour of v in G;

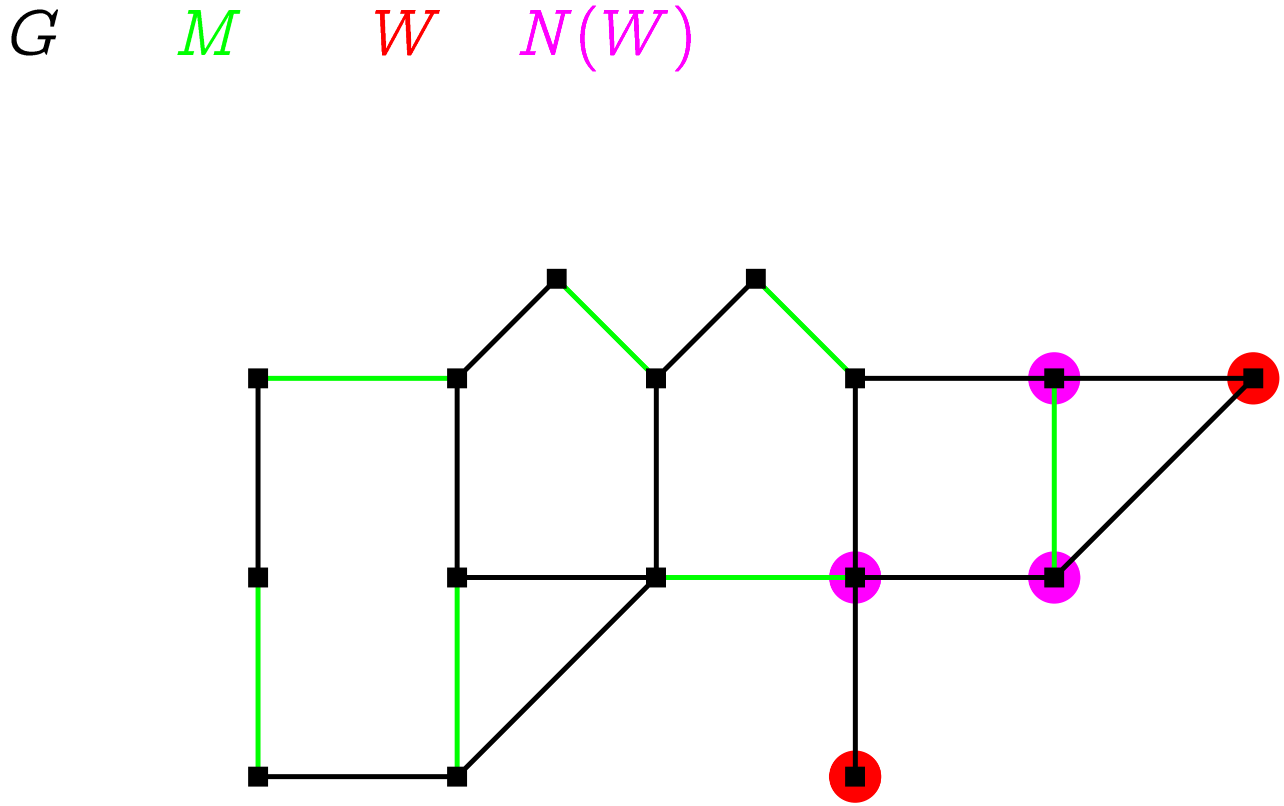
  – M_C^w is the maximum matching in C not covering w.



Complexity:

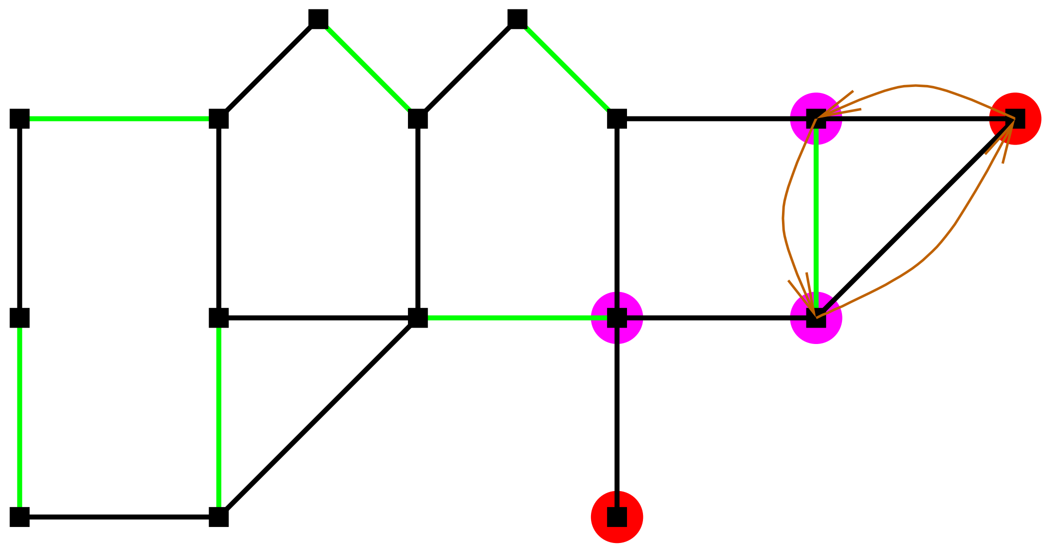
- To find a maximal matching, the previous algorithm has to be called up to $|V|/2$ times.
- During one execution of the algorithm:
  - The walk P can be found in time $O(|E|)$. The matc-hing M can be updated in time $O(|E|)$.
  - The recursion depth is $O(|V|)$.

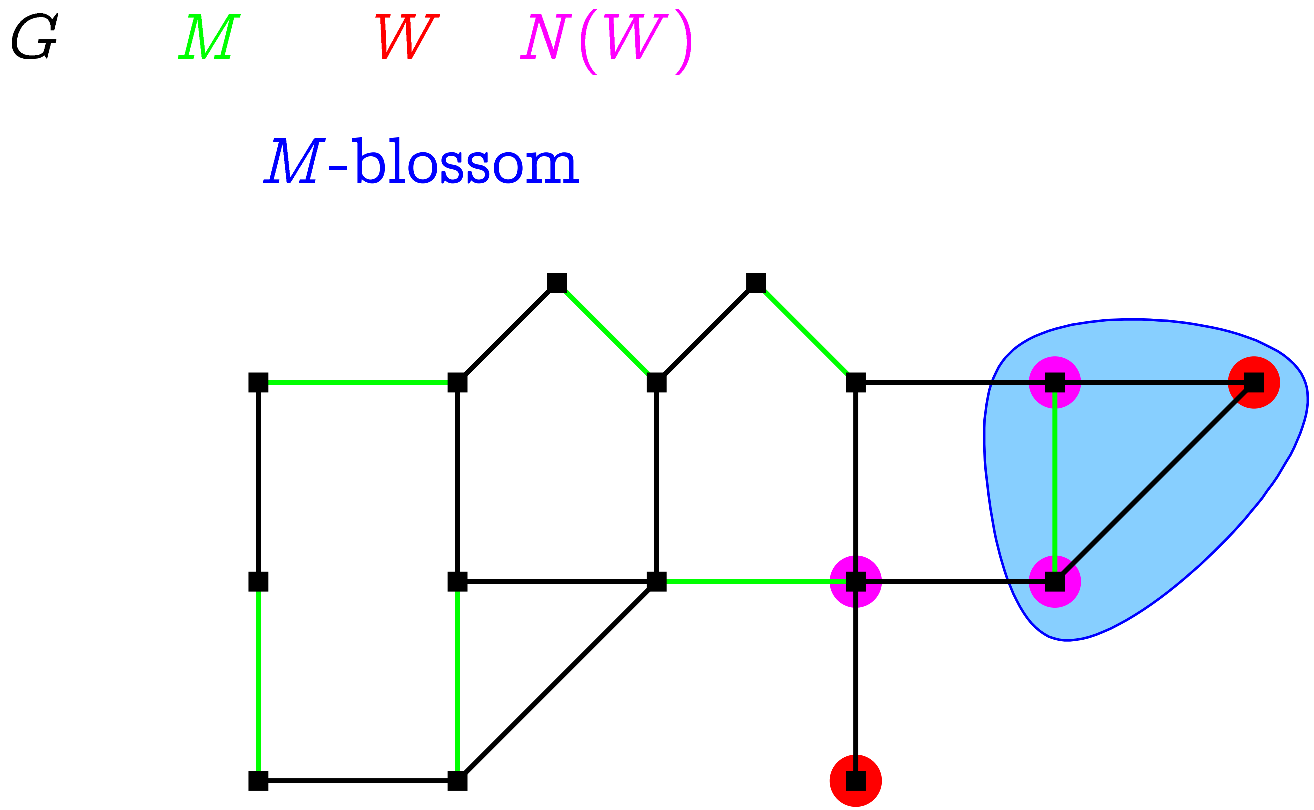
  One execution requires $O(|V| \cdot |E|)$ time altogether.
- Maximal matching can be found in time $O(|V|^2 \cdot |E|)$.

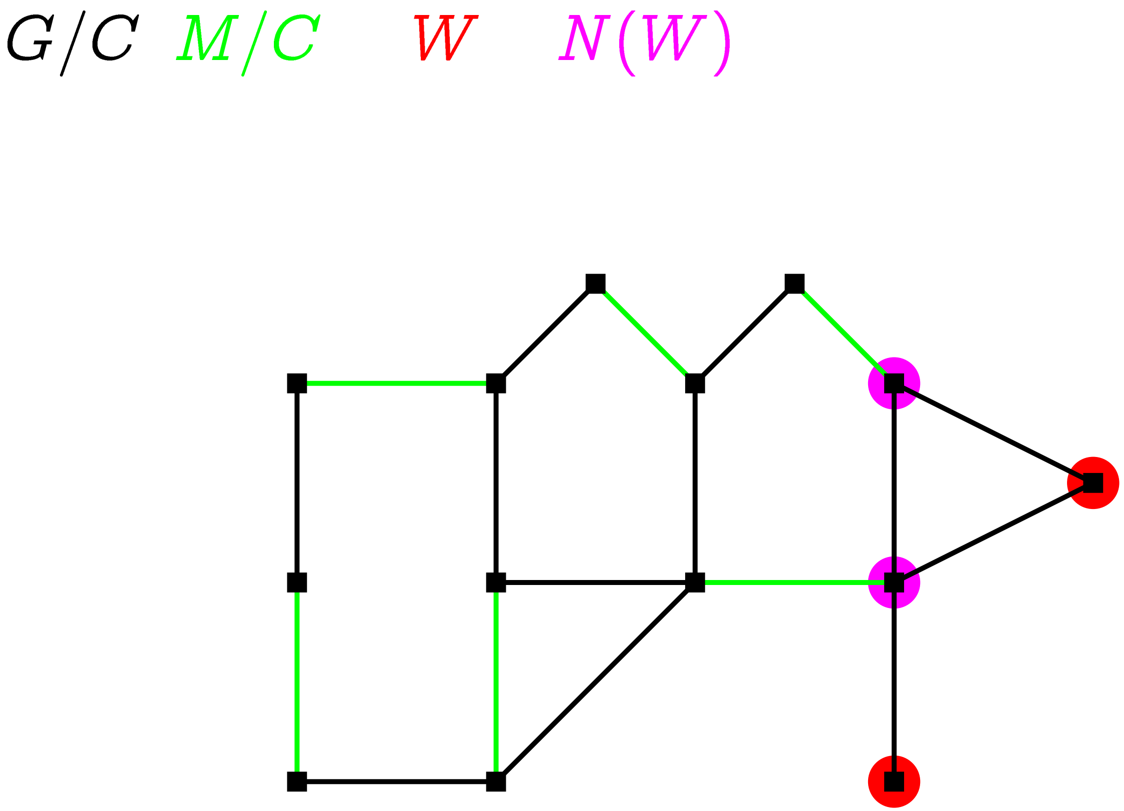

G
M
W
$N(W)$

G M W $N(W)$

Shortest M-extensible walk

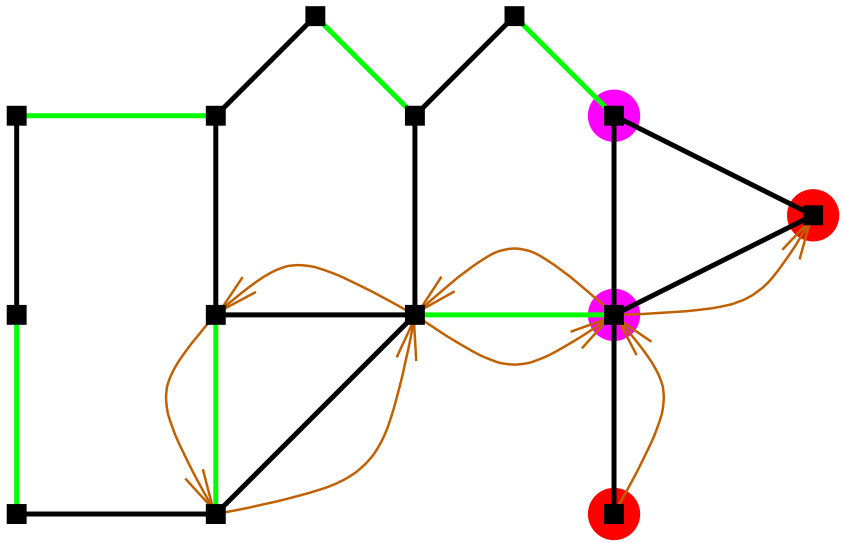


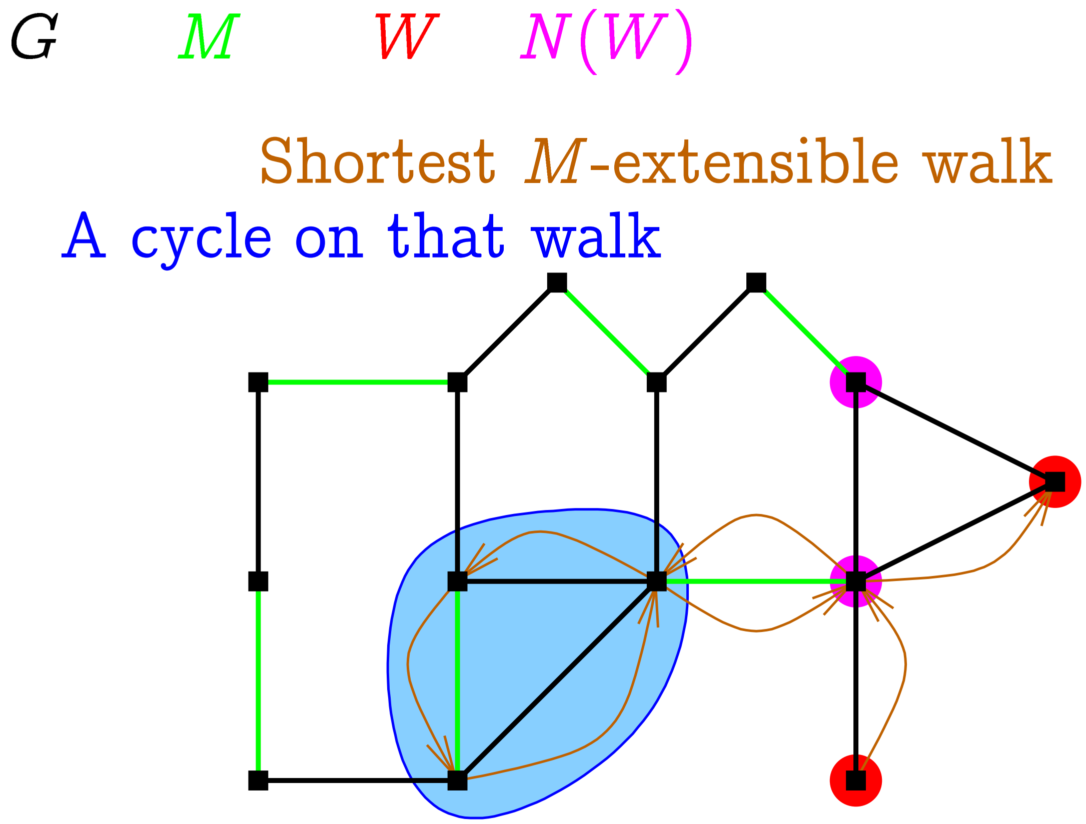

G
M
W
$N(W)$
M-blossom


G/C
M/C
W
$N(W)$

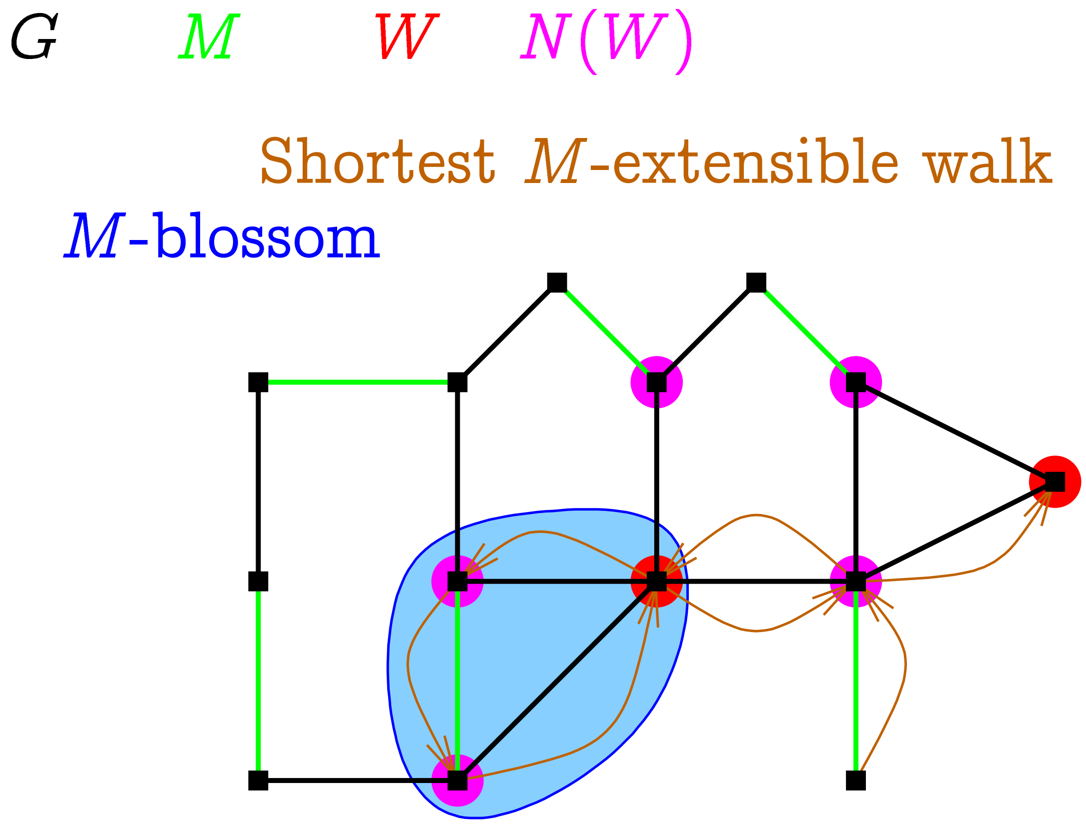
G M W $N(W)$

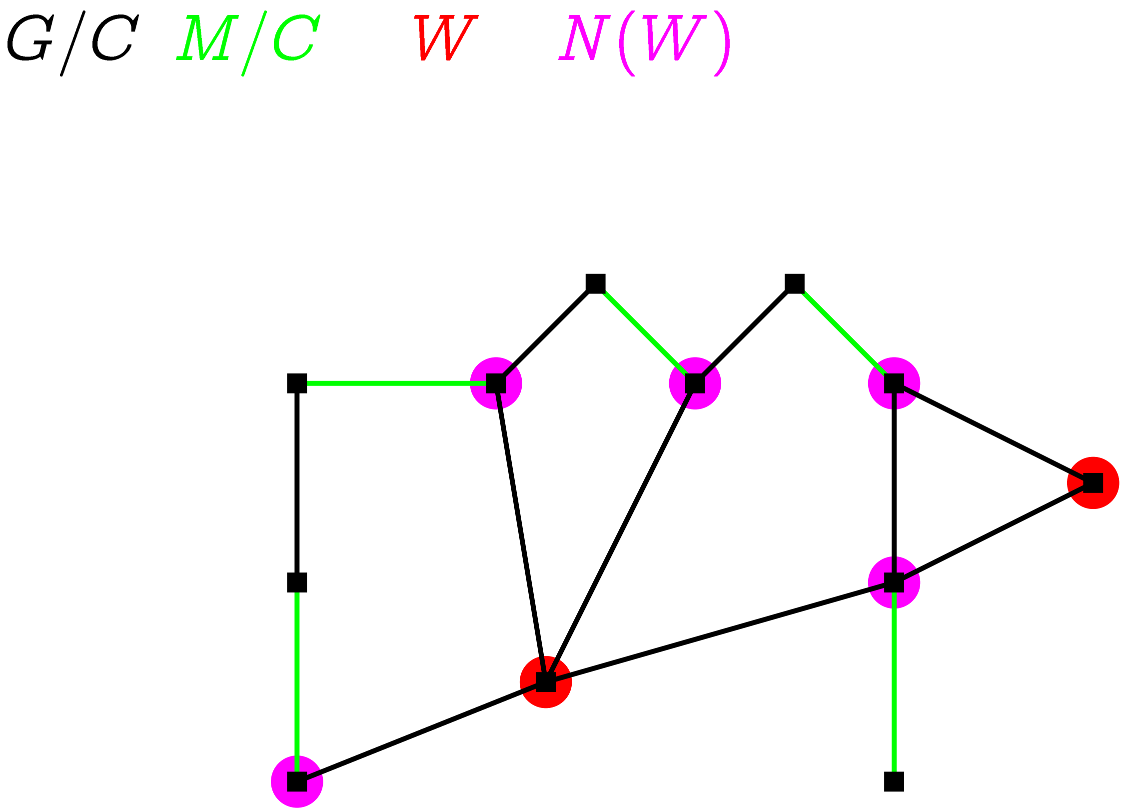
Shortest M-extensible walk





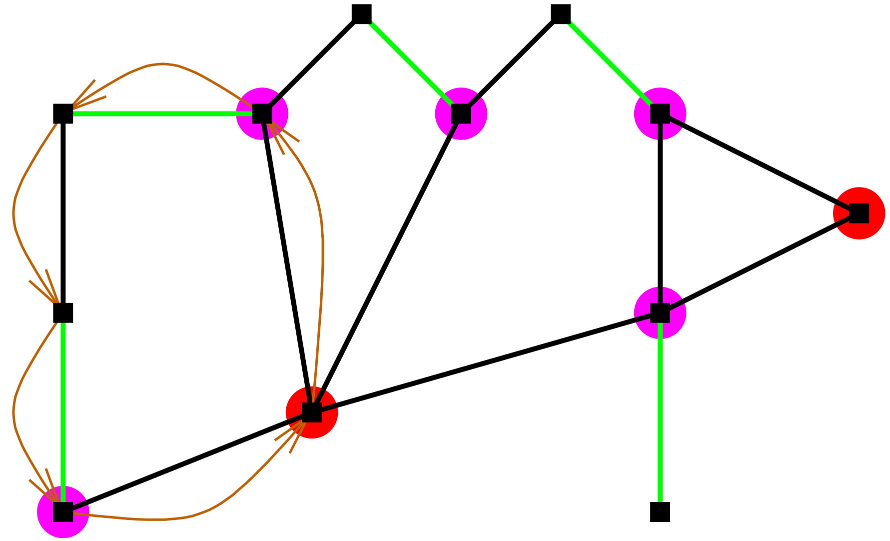
G
M
W
$N(W)$
Shortest M-extensible walk
A cycle on that walk

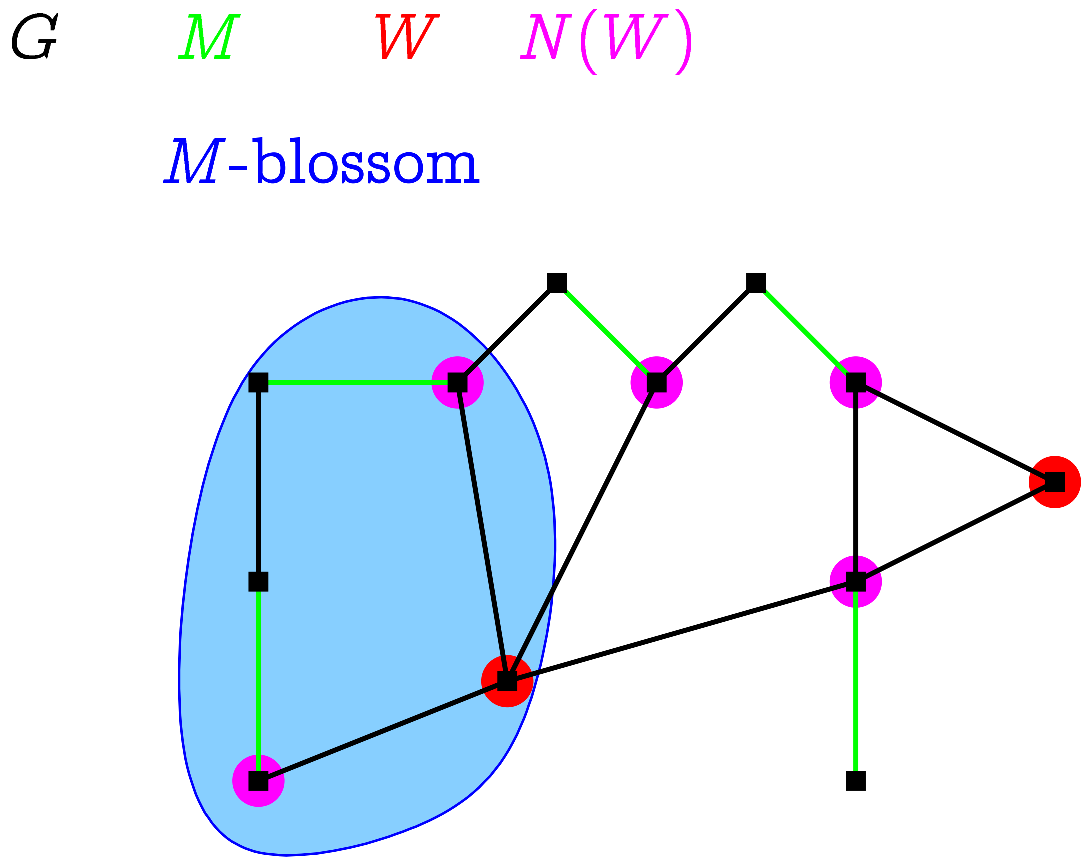

G
M
W
$N(W)$
Shortest M-extensible walk
M-blossom


G/C M/C W $N(W)$

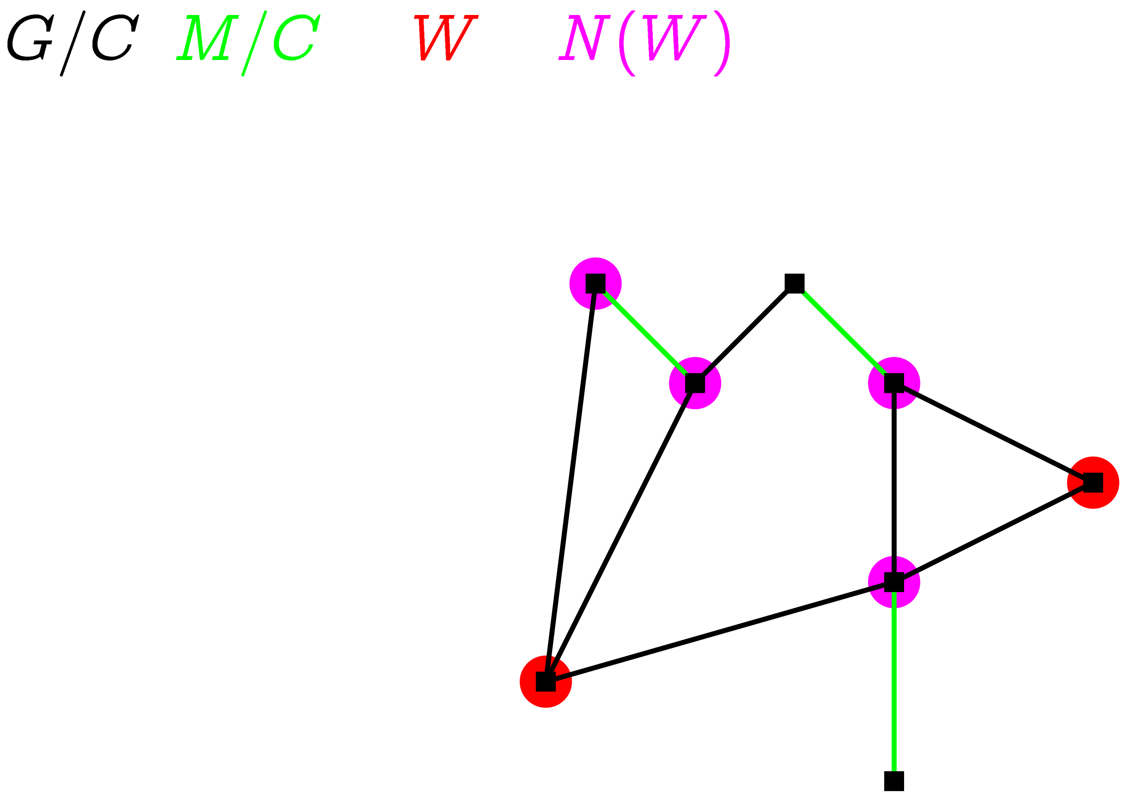
G M W $N(W)$

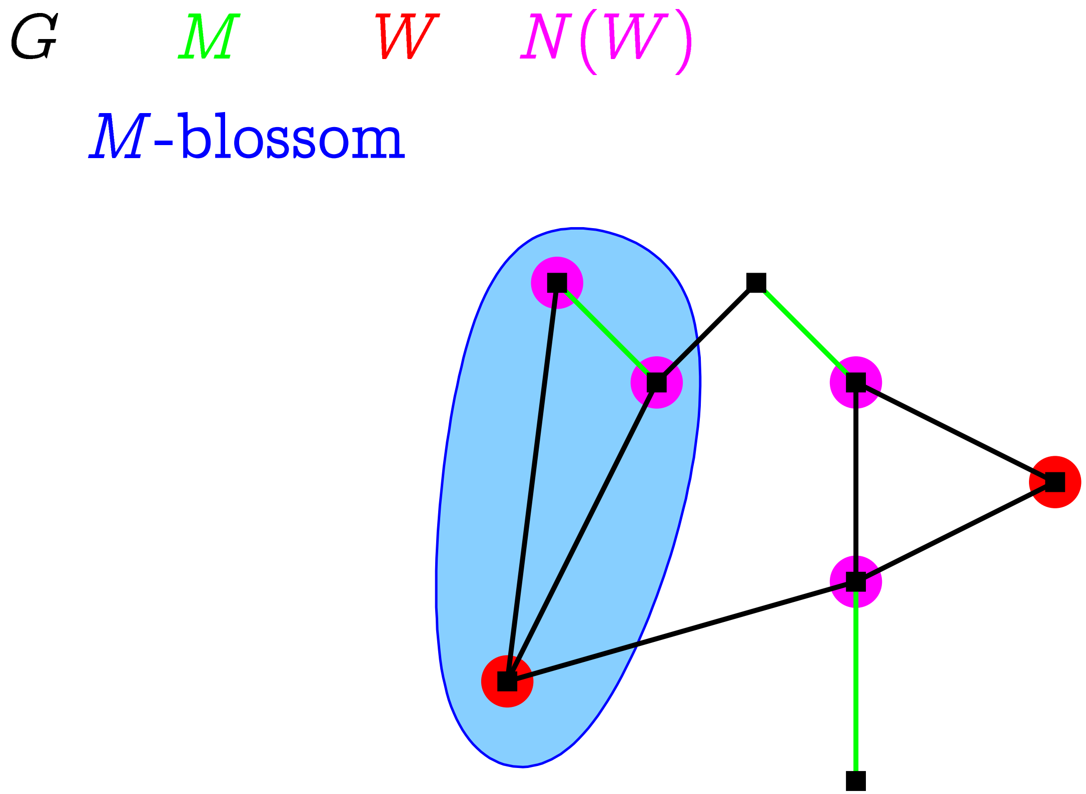
Shortest M-extensible walk



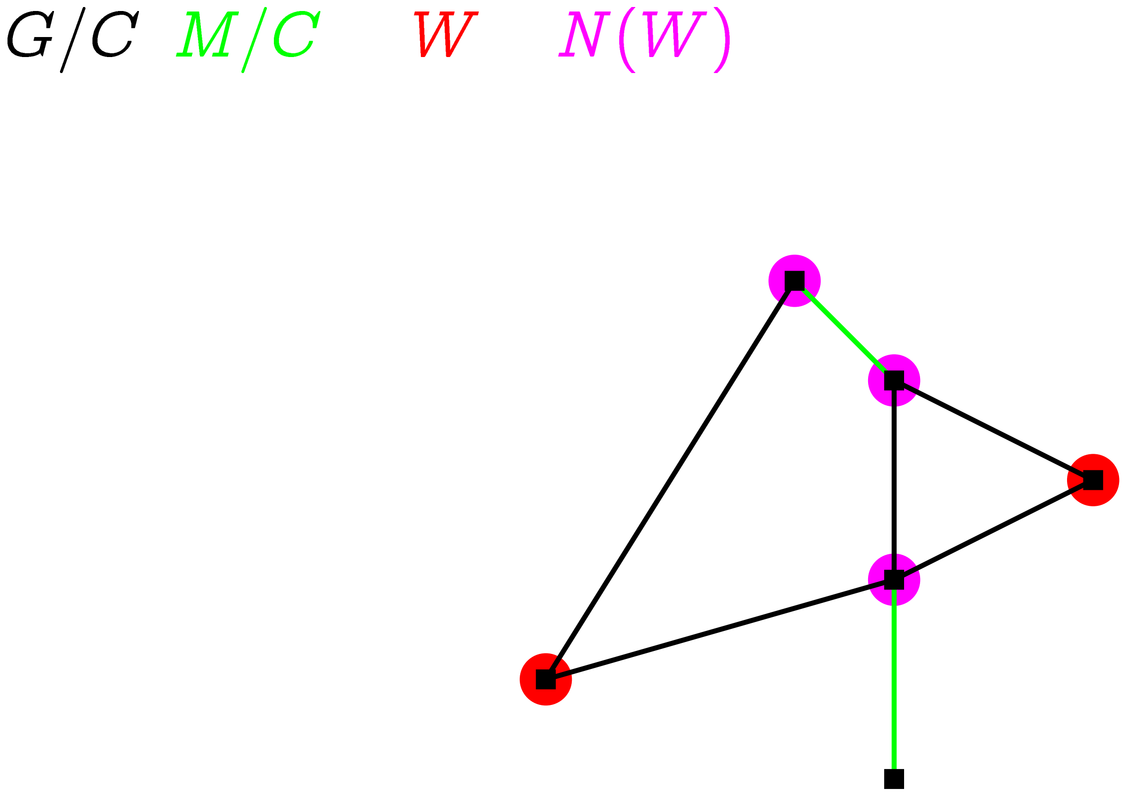


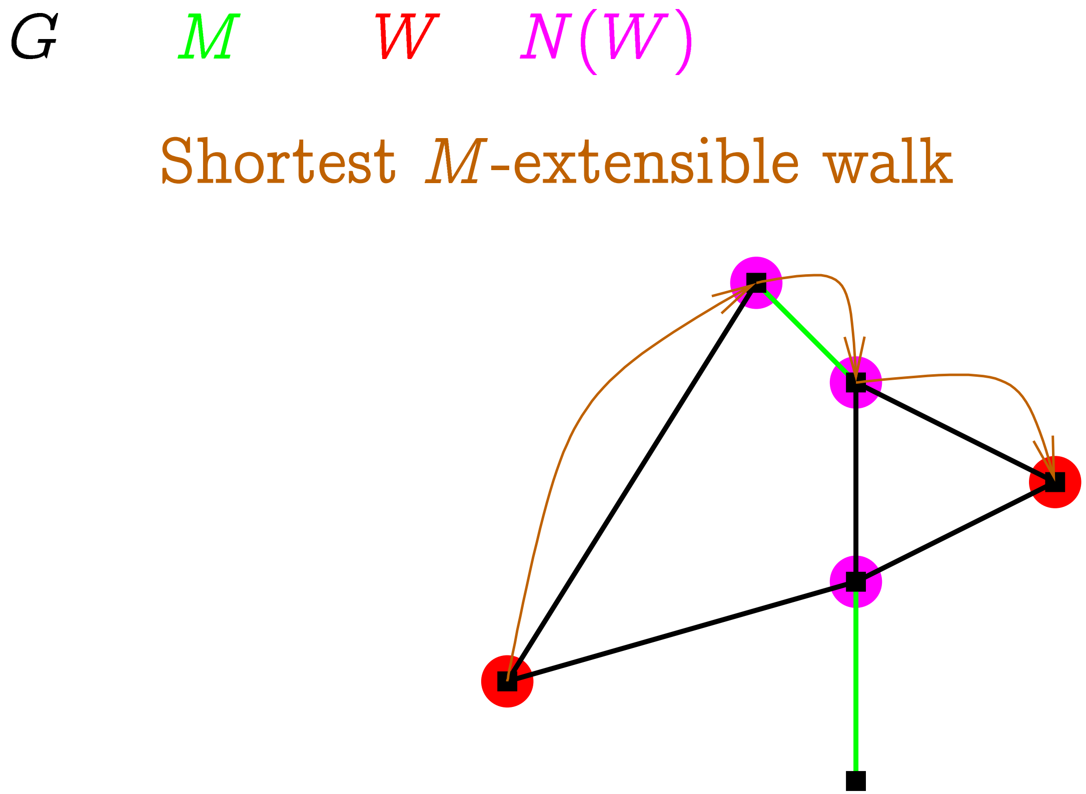
G
M
W
$N(W)$
M-blossom


G/C
M/C
W
$N(W)$

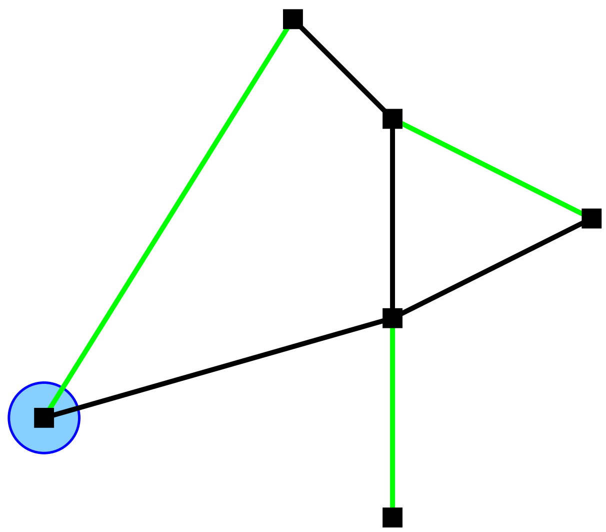


G
M
W
$N(W)$
M-blossom

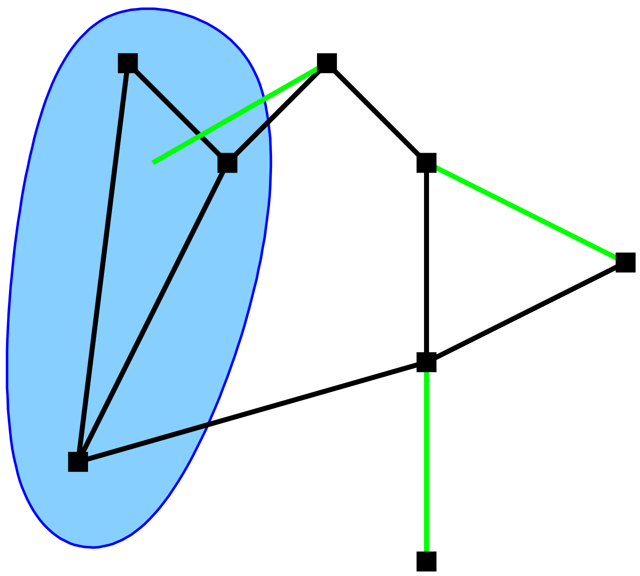

G/C
M/C
W
$N(W)$


G M W $N(W)$
Shortest M-extensible walk

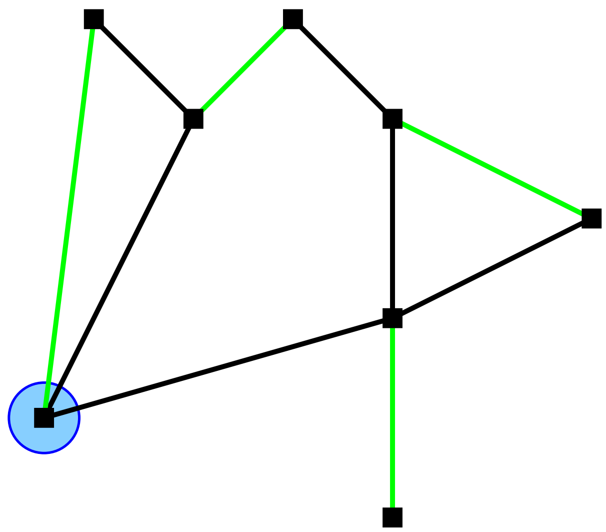
G M



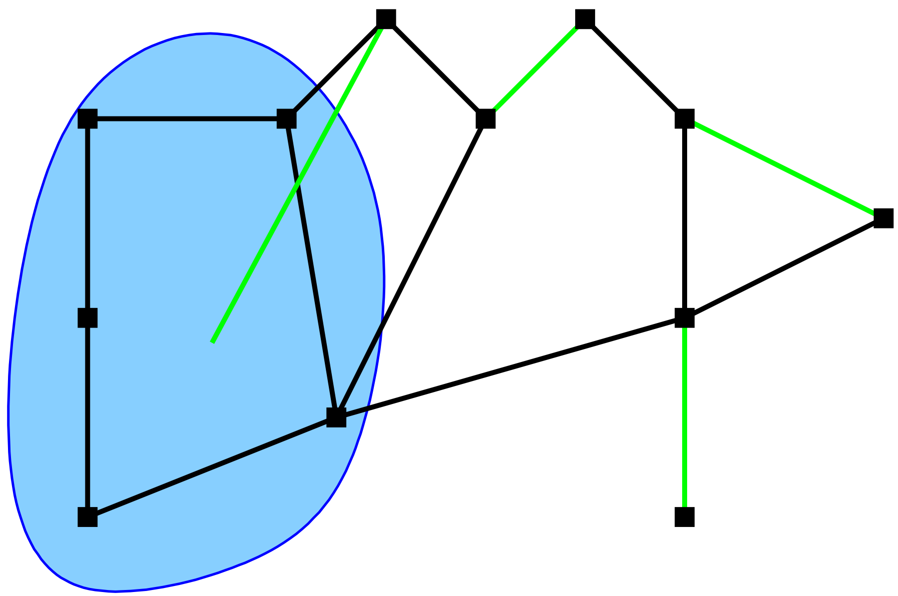
G M



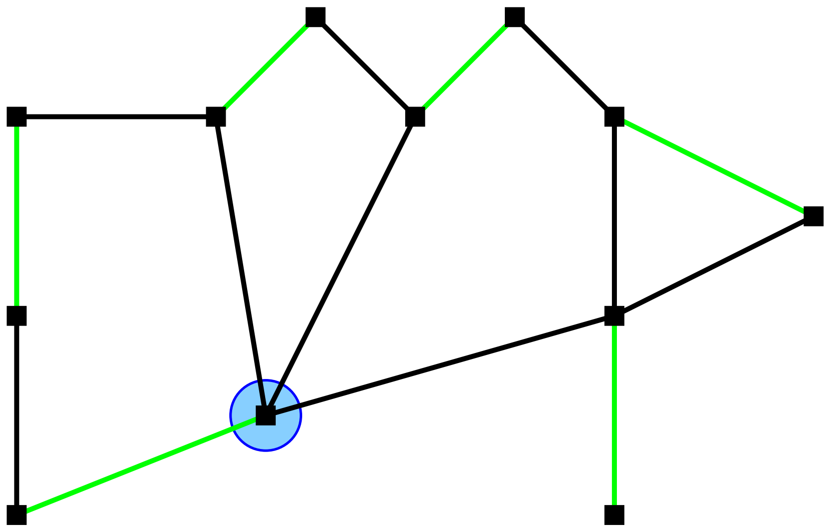
G M



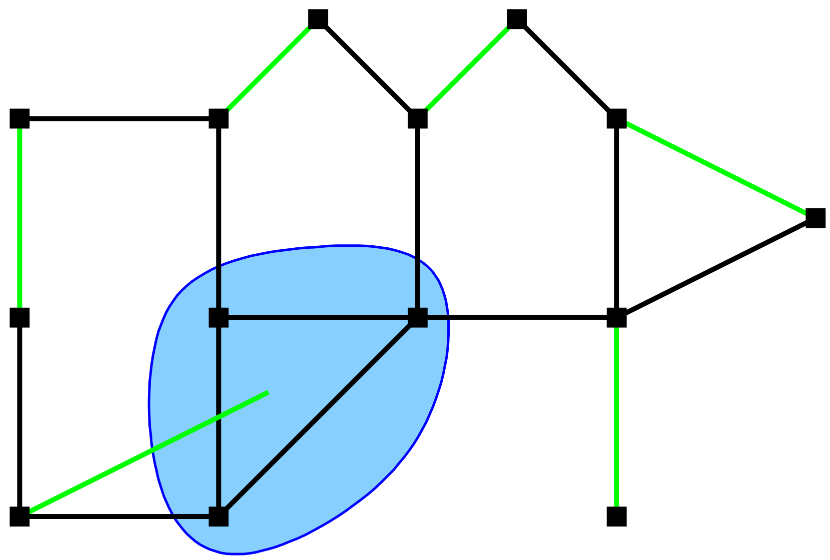
G M

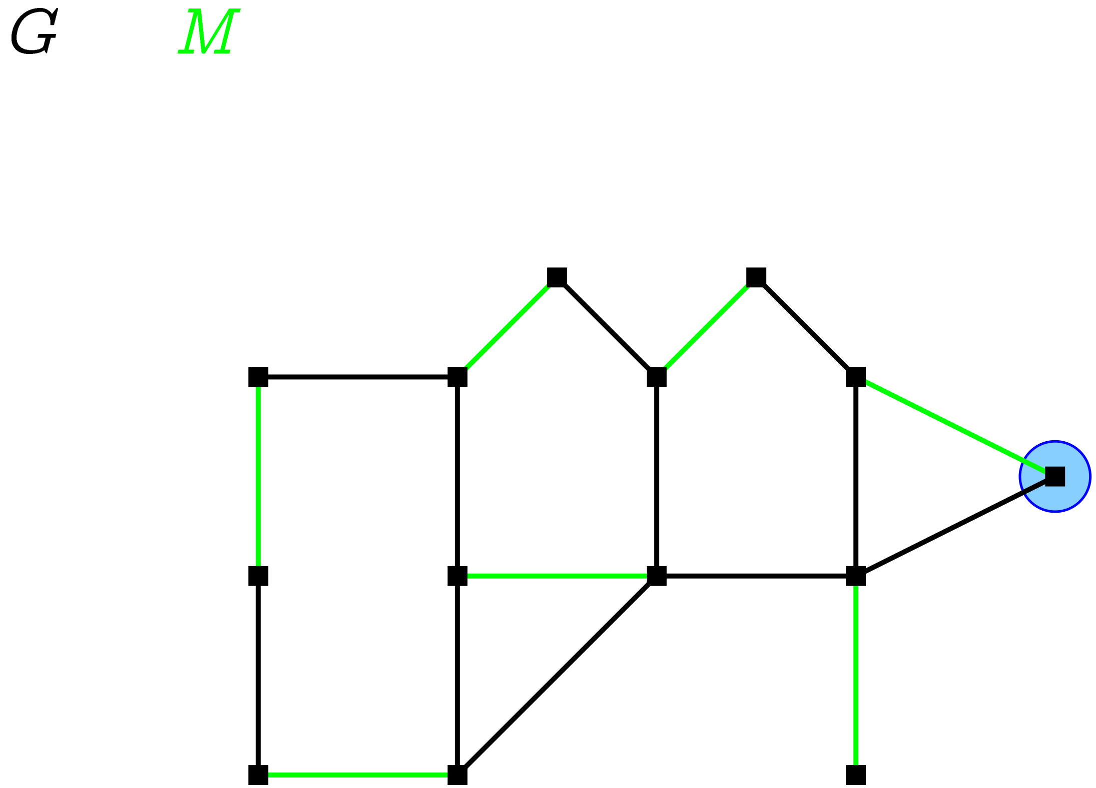


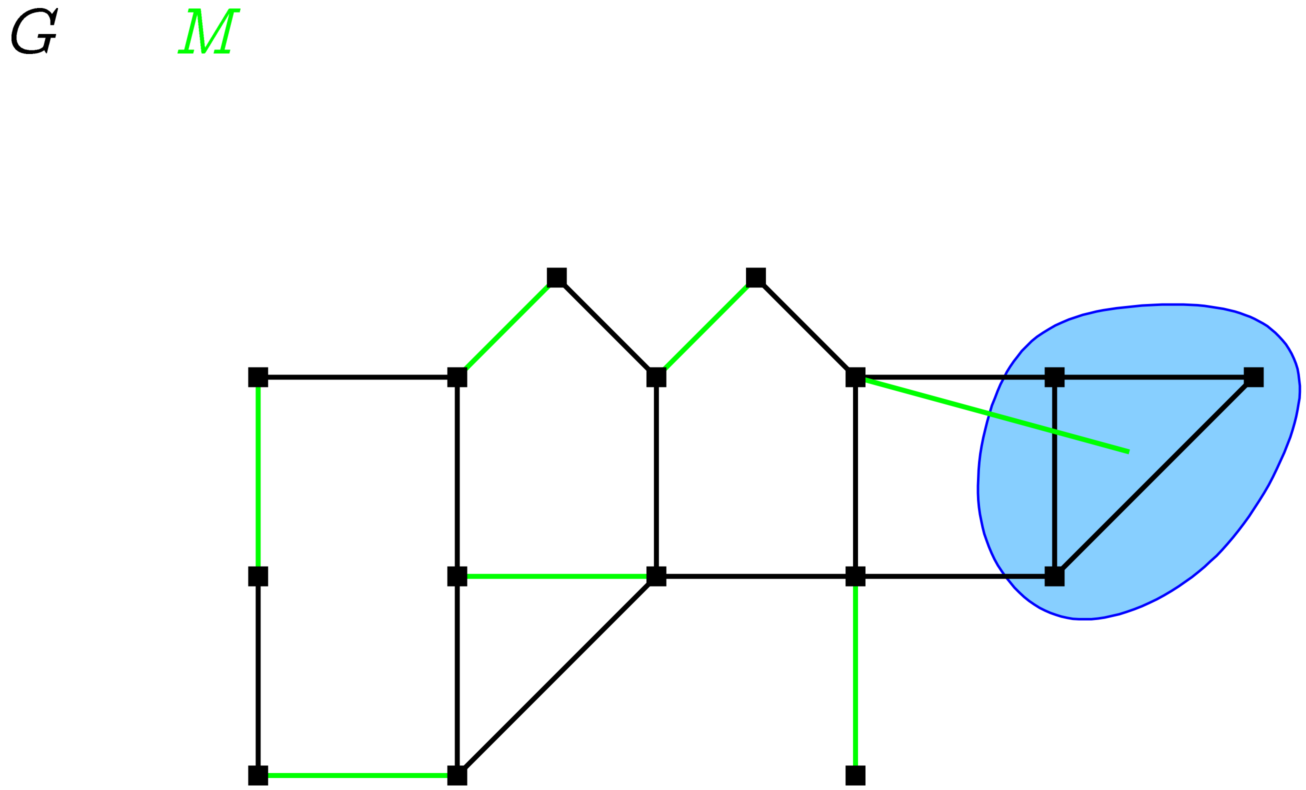
G M



G M




G
M


G
M

G M