O Algoritmo das Flores de Edmonds

Combinando

Sabrina Teodoro

Setembro 2020

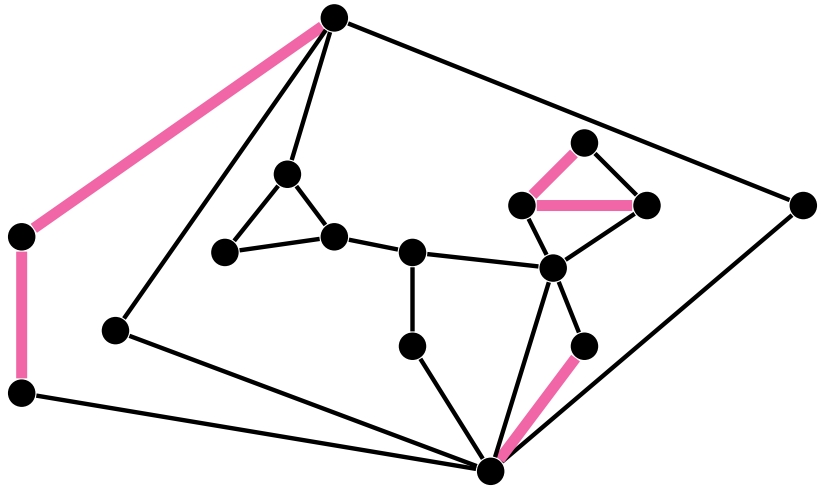
Emparelhamento máximo

Emparelhamento máximo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

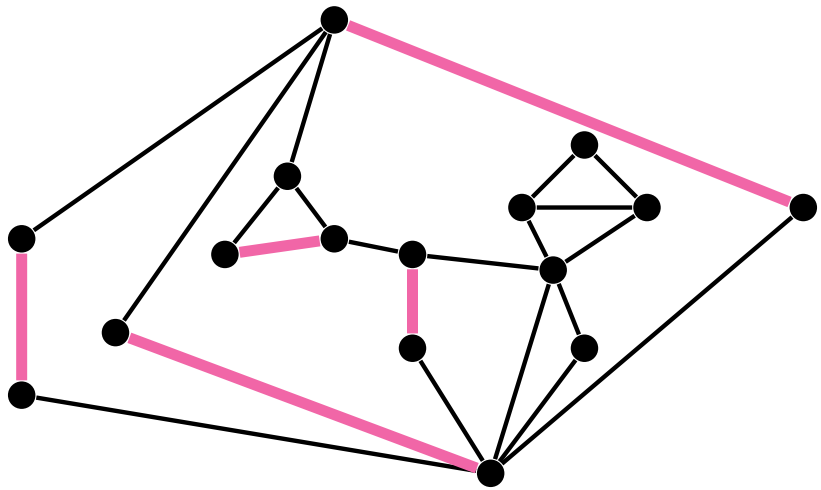
Emparelhamento máximo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.



Emparelhamento máximo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.



Emparelhamento máximo

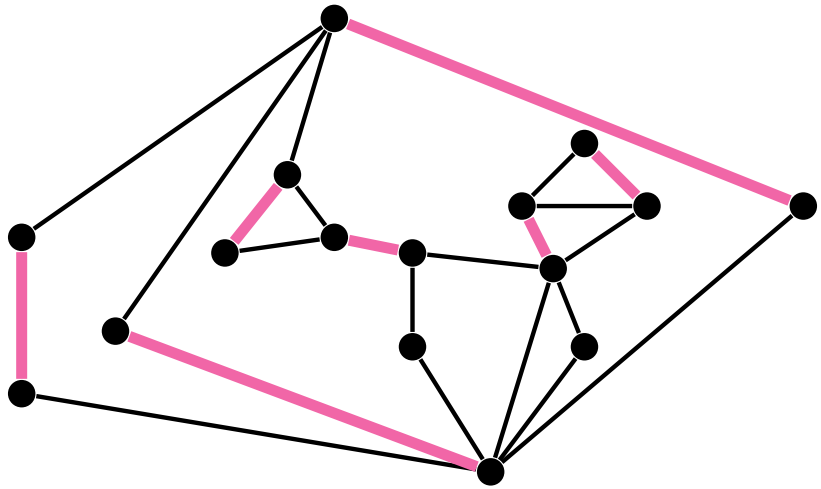
Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

Se $|M| \geq |M'|$ para qualquer emparelhamento M', dizemos que M é um emparelhamento **máximo**.

Emparelhamento máximo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

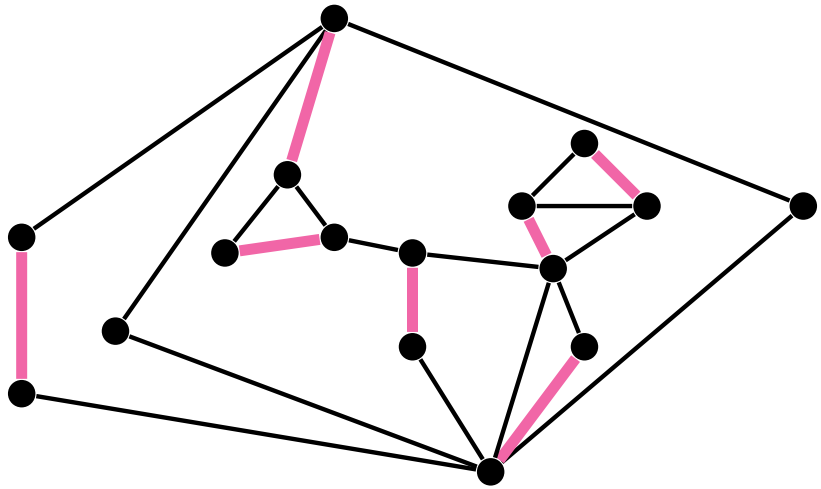
Se $|M| \geq |M'|$ para qualquer emparelhamento M', dizemos que M é um emparelhamento **máximo**.



Emparelhamento máximo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

Se $|M| \geq |M'|$ para qualquer emparelhamento M', dizemos que M é um emparelhamento **máximo**.



Emparelhamento máximo

Um **emparelhamento** (=*matching*) é um conjunto M de arestas duas a duas não adjacentes.

Se $|M| \geq |M'|$ para qualquer emparelhamento M', dizemos que M é um emparelhamento **máximo**.

Problema

Encontrar um emparelhamento máximo em um grafo arbitrário.

A solução desse problema é um importante marco no desenvolvimento da **otimização combinatória**.
Os emparelhamentos máximos também geram uma teoria de grafos bastante interessante e possuem várias aplicações. Uma delas é ao **problema do caixeiro viajante métrico**, em que são utilizados como sub-problema de um algoritmo.

Roteiro

Algoritmo: emparelhamento máximo

Roteiro

Algoritmo: emparelhamento máximo

- Caminho de aumento
- Algoritmo: caminho de aumento

Roteiro

Algoritmo: emparelhamento máximo

- Caminho de aumento
- Algoritmo: caminho de aumento
  - Árvore alternante
  - Algoritmo: extensão de árvore
  - Algoritmo: extensão de caminho de aumento
  - Algoritmo: recursão de flor

Roteiro

Algoritmo: emparelhamento máximo

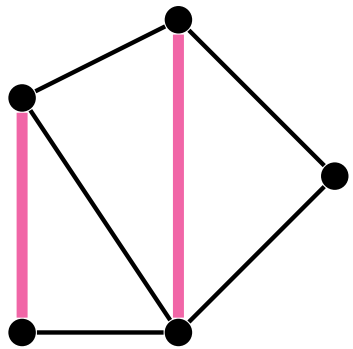
- Caminho de aumento
- Algoritmo: caminho de aumento
  - Árvore alternante
  - Algoritmo: extensão de árvore
  - Algoritmo: extensão de caminho de aumento
  - Algoritmo: recursão de flor
    - Flor
    - Contração de circuito

Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.



Caminho de aumento

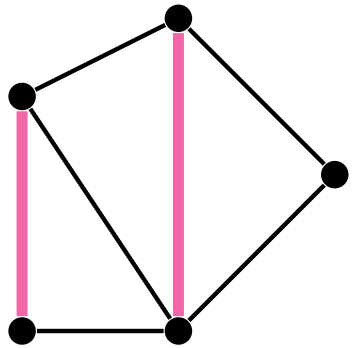
Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

Um caminho é **alternante** ou M-**alternante** se suas arestas estão, alternadamente, em M e em $A \backslash M$.

Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

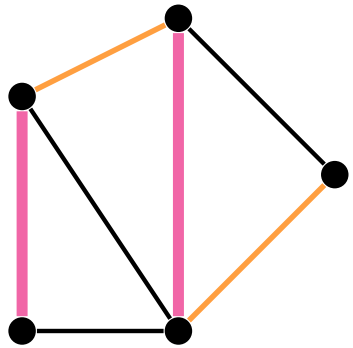
Um caminho é **alternante** ou M-**alternante** se suas arestas estão, alternadamente, em M e em $A \backslash M$.



Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

Um caminho é **alternante** ou M-**alternante** se suas arestas estão, alternadamente, em M e em $A \backslash M$.



Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

Um caminho é **alternante** ou M-**alternante** se suas arestas estão, alternadamente, em M e em $A \backslash M$.

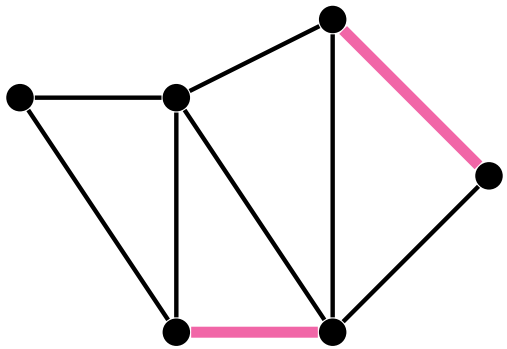
Um caminho **de aumento**, **aumentador** ou M-**aumentador** é um caminho M-alternante que começa e termina em dois vértices expostos distintos.

Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

Um caminho é **alternante** ou M-**alternante** se suas arestas estão, alternadamente, em M e em $A \backslash M$.

Um caminho **de aumento**, **aumentador** ou M-**aumentador** é um caminho M-alternante que começa e termina em dois vértices expostos distintos.

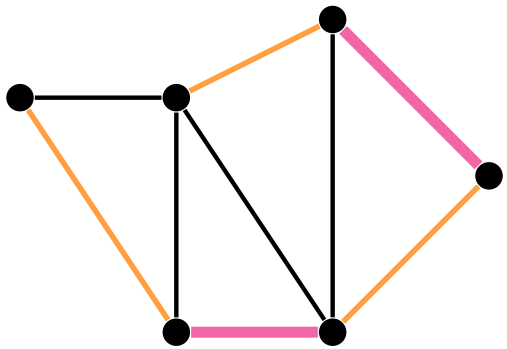


Caminho de aumento

Um emparelhamento M **cobre** um vértice v se alguma aresta de M incide em v. Caso contrário, diz-se que v é **exposto** por M.

Um caminho é **alternante** ou M-**alternante** se suas arestas estão, alternadamente, em M e em $A \backslash M$.

Um caminho **de aumento**, **aumentador** ou M-**aumentador** é um caminho M-alternante que começa e termina em dois vértices expostos distintos.

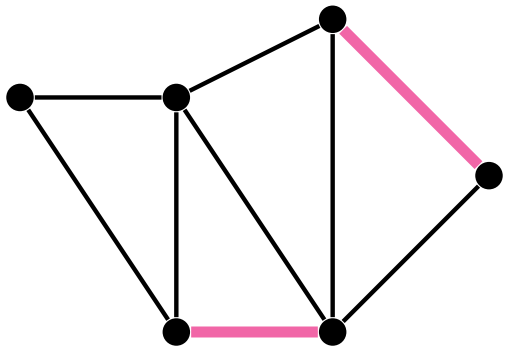


Caminho de aumento

Tomando a diferença simétrica entre um caminho de aumento P e o emparelhamento correspondente M, denotada por $M \oplus A(P)$, obtemos um novo emparelhamento M' tal que $|M'| = |M| + 1$.

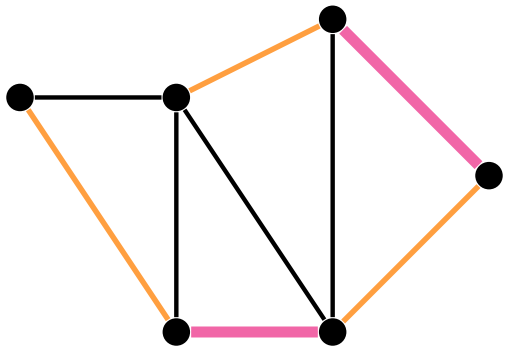
Caminho de aumento

Tomando a diferença simétrica entre um caminho de aumento P e o emparelhamento correspondente M, denotada por $M \oplus A(P)$, obtemos um novo emparelhamento M' tal que $|M'| = |M| + 1$.



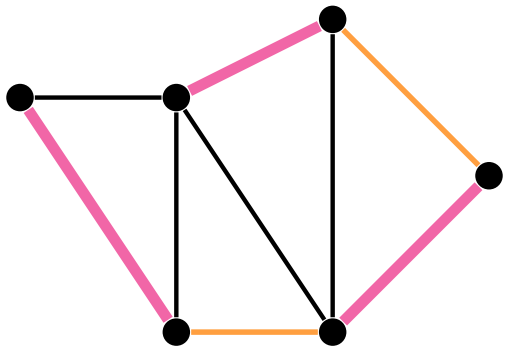
Caminho de aumento

Tomando a diferença simétrica entre um caminho de aumento P e o emparelhamento correspondente M, denotada por $M \oplus A(P)$, obtemos um novo emparelhamento M' tal que $|M'| = |M| + 1$.



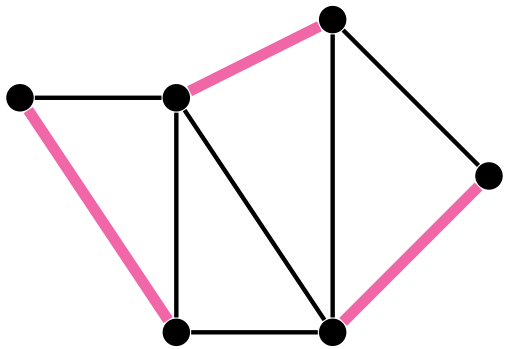
Caminho de aumento

Tomando a diferença simétrica entre um caminho de aumento P e o emparelhamento correspondente M, denotada por $M \oplus A(P)$, obtemos um novo emparelhamento M' tal que $|M'| = |M| + 1$.



Caminho de aumento

Tomando a diferença simétrica entre um caminho de aumento P e o emparelhamento correspondente M, denotada por $M \oplus A(P)$, obtemos um novo emparelhamento M' tal que $|M'| = |M| + 1$.



Algoritmo de Edmonds: emparelhamento máximo

Desenvolvido em 1961 e publicado em 1965 pelo cientista da computação e matemático Jack Edmonds. Conhecido como "algoritmo das flores" (= *"Edmonds' blossom algorithm"*).

Algoritmo 1: EDMONDS:EMP_MAX(G, M)

```
1 faça M ← ∅;
2 faça P ←CAM_AUM(G, M);
3 se P = ∅, então:
4   devolva M;
5 senão:
6   M ← M ⊕ A(P);
7   devolva EMP_MAX(G, M);
```

O lema a seguir garante que o algoritmo anterior encontra, de fato, um emparelhamento máximo.

Lema de Berge

Dado um grafo G e um emparelhamento M, M é um emparelhamento máximo se, e somente se, não existe um caminho M-aumentador em G.

O lema a seguir garante que o algoritmo anterior encontra, de fato, um emparelhamento máximo.

Lema de Berge

Dado um grafo G e um emparelhamento M, M é um emparelhamento máximo se, e somente se, não existe um caminho M-aumentador em G.

($\Rightarrow$) Suponha P um caminho M-aumentador.

O lema a seguir garante que o algoritmo anterior encontra, de fato, um emparelhamento máximo.

Lema de Berge

Dado um grafo G e um emparelhamento M, M é um emparelhamento máximo se, e somente se, não existe um caminho M-aumentador em G.

($\Rightarrow$) Suponha P um caminho M-aumentador. Então $M' = M \oplus A(P)$ é um emparelhamento maior que M. Logo M não é máximo.

O lema a seguir garante que o algoritmo anterior encontra, de fato, um emparelhamento máximo.

Lema de Berge

Dado um grafo G e um emparelhamento M, M é um emparelhamento máximo se, e somente se, não existe um caminho M-aumentador em G.

($\Rightarrow$) Suponha P um caminho M-aumentador. Então $M' = M \oplus A(P)$ é um emparelhamento maior que M. Logo M não é máximo.
($\Leftarrow$) Agora seja M um emparelhamento não máximo. Então existe M' com $|M'| > |M|$.

O lema a seguir garante que o algoritmo anterior encontra, de fato, um emparelhamento máximo.

Lema de Berge

Dado um grafo G e um emparelhamento M, M é um emparelhamento máximo se, e somente se, não existe um caminho M-aumentador em G.

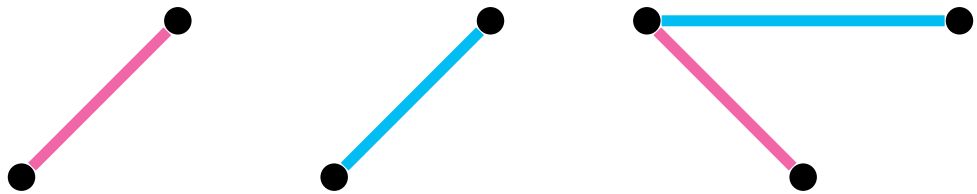
($\Rightarrow$) Suponha P um caminho M-aumentador. Então $M' = M \oplus A(P)$ é um emparelhamento maior que M. Logo M não é máximo.
($\Leftarrow$) Agora seja M um emparelhamento não máximo. Então existe M' com $|M'| > |M|$. Considere o subgrafo induzido pelo conjunto (de arestas) $M \oplus M'$ e note que para cada vértice v temos $1 \leq g(v) \leq 2$.

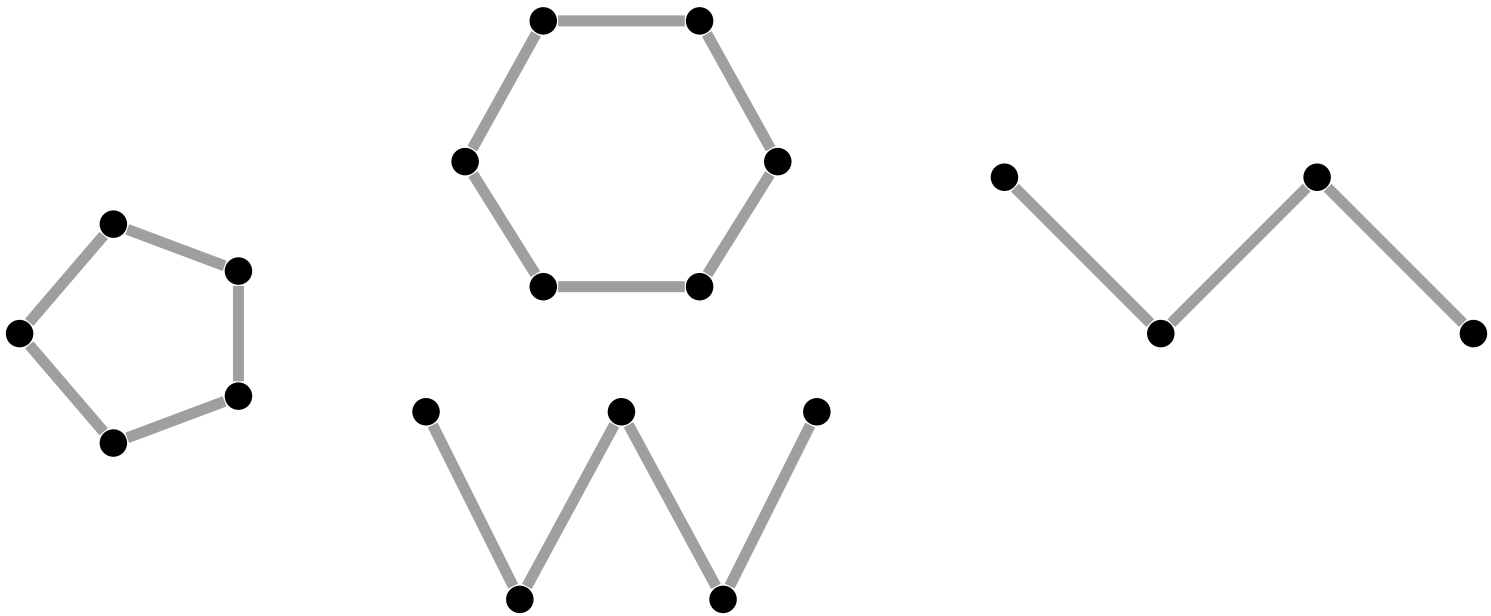
O lema a seguir garante que o algoritmo anterior encontra, de fato, um emparelhamento máximo.

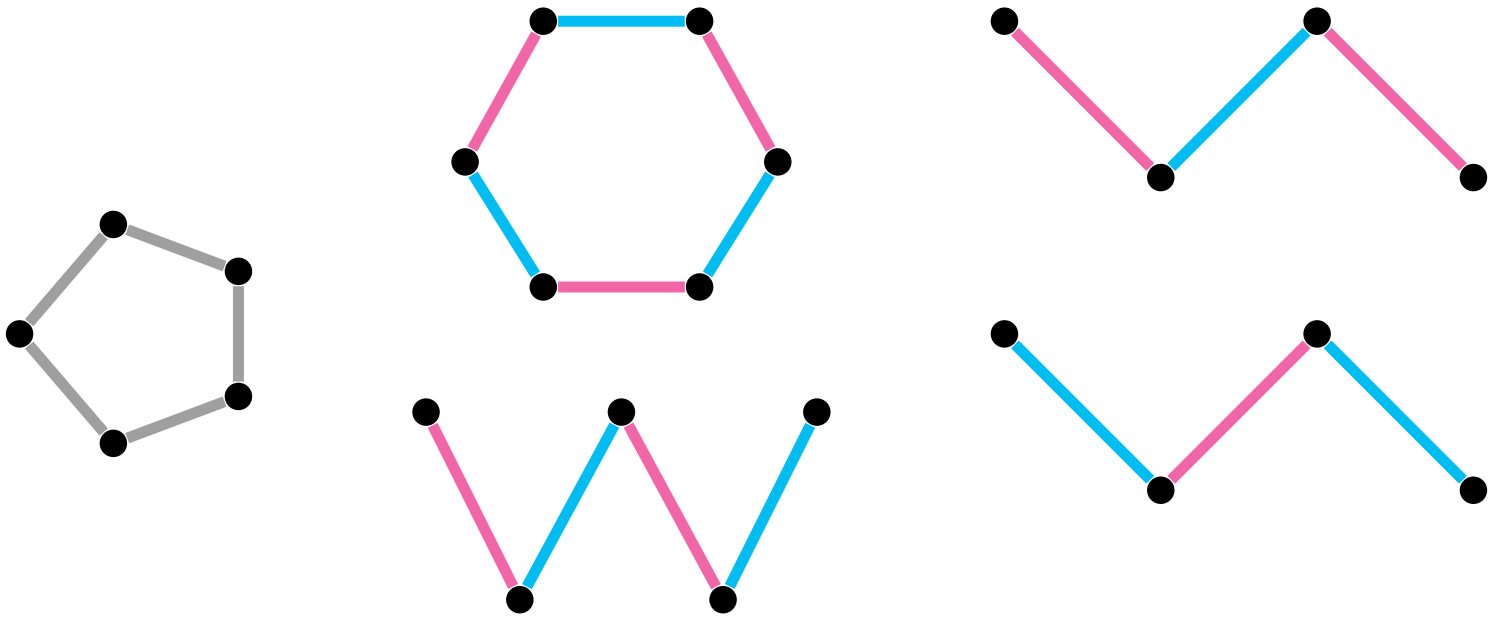
Lema de Berge

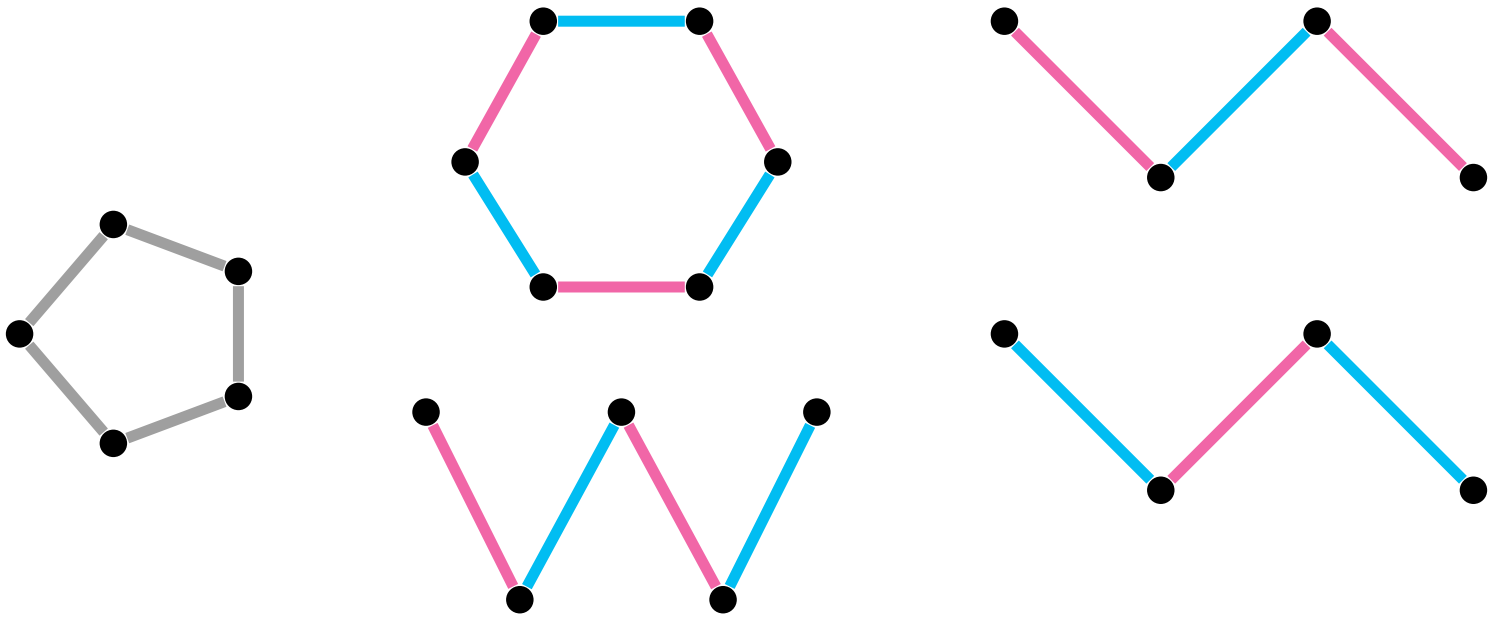
Dado um grafo G e um emparelhamento M, M é um emparelhamento máximo se, e somente se, não existe um caminho M-aumentador em G.

($\Rightarrow$) Suponha P um caminho M-aumentador. Então $M' = M \oplus A(P)$ é um emparelhamento maior que M. Logo M não é máximo.
($\Leftarrow$) Agora seja M um emparelhamento não máximo. Então existe M' com $|M'| > |M|$. Considere o subgrafo induzido pelo conjunto (de arestas) $M \oplus M'$ e note que para cada vértice v temos $1 \leq g(v) \leq 2$.









Logo, existe uma componente conexa do subgrafo que consiste em um caminho ímpar alternado, com arestas inicial e final em M'. Perceba que tal caminho é M-aumentador.

Árvore alternante

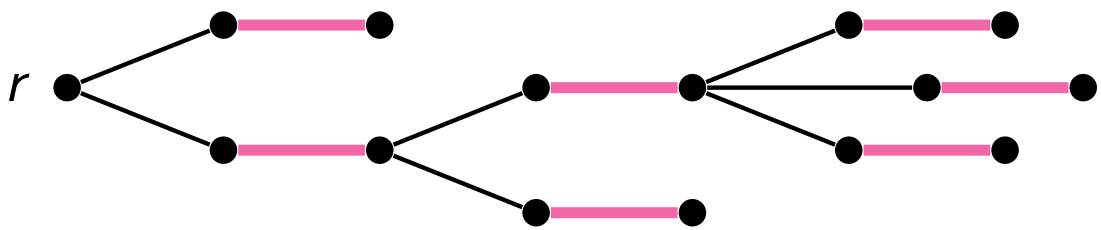
O algoritmo que encontra um caminho de aumento tem como base o conceito a seguir.

Árvore alternante

O algoritmo que encontra um caminho de aumento tem como base o conceito a seguir.

Uma árvore T com raiz r é **alternante** ou M-**alternante** se possui três propriedades:

- r é exposto;
- cada vértice de $T \backslash \{r\}$ é coberto por uma aresta de $M \cap A(T)$;
- para todo vértice v de T, o caminho em T de extremos r e v é M-alternante.



Algoritmo de Edmonds: caminho de aumento

Ideia geral: construir uma floresta F tal que cada vértice exposto v é raiz de uma árvore alternante T_v e mapear caminhos de aumento a partir dessas árvores.

Algoritmo 2: EDMONDS:CAM_AUM(G, M)

```
1  faça F ← ∅;
2  faça V' ← vértices expostos em G;
3  para v ∈ V', faça:
4     T_v ← ({v}, ∅);
5     r(v) ← v;
6     F ← F ∪ T_v;
7  faça A' ← M;
8  para v ∈ V(F), faça:
9    enquanto existir w tal que vw ∈ A\A', faça:
10     se w ∉ V(F), então:
11       EXT_ARV(M, F, v, w);
12     senão:
13       se d(r(w), w) é par, então:
14         se r(v) ≠ r(w), então:
15           P ←EXT_CAM_AUM(F, v, w, r(v), r(w));
16         senão:
17           P ←REC_FLOR(G, M, F, v, w);
18         devolva P;
19      faça A' ← A' ∪ {vw};
20 devolva ∅;
```

Algoritmo 2: EDMONDS:CAM_AUM(G, M)

```
1  faça F ← ∅;
2  faça V' ← vértices expostos em G;
3  para v ∈ V', faça:
4     T_v ← ({v}, ∅);
5     r(v) ← v;
6     F ← F ∪ T_v;
7  faça A' ← M;
8  para v ∈ V(F), faça:
9    enquanto existir w tal que vw ∈ A\A', faça:
10     se w ∉ V(F), então:
11       EXT_ARV(M, F, v, w);
12     senão:
13       se d(r(w), w) é par, então:
14         se r(v) ≠ r(w), então:
15           P ←EXT_CAM_AUM(F, v, w, r(v), r(w));
16         senão:
17           P ←REC_FLOR(G, M, F, v, w);
18         devolva P;
19      faça A' ← A' ∪ {vw};
20 devolva ∅;
```

Algoritmo de Edmonds: extensão de árvore

Convém, incialmente, marcar as arestas do emparelhamento atual M e, após cada iteração, a aresta recém-analisada.

Para cada vértice v da floresta F, olhamos para uma aresta não marcada vw que incide em v.

Algoritmo de Edmonds: extensão de árvore

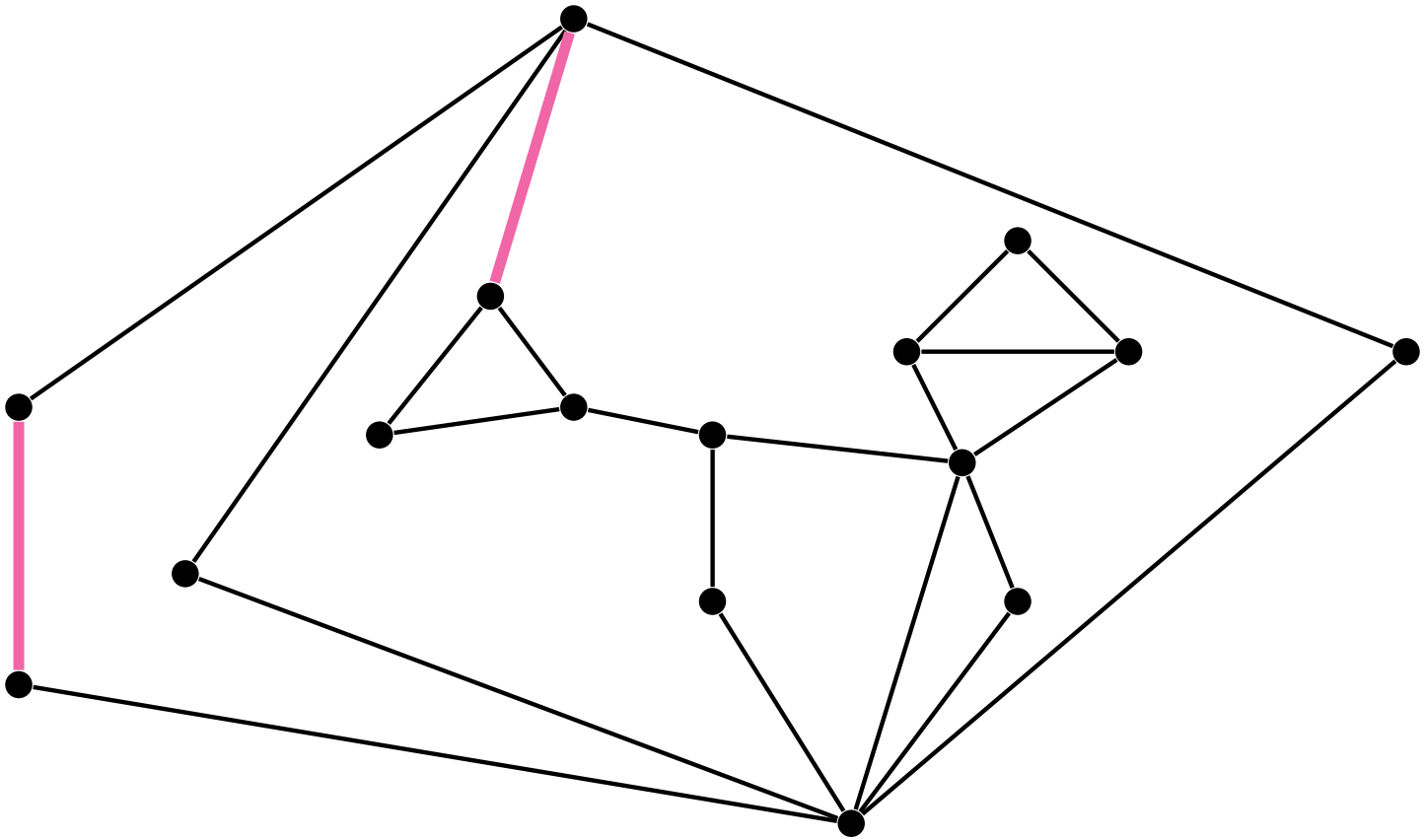
Convém, incialmente, marcar as arestas do emparelhamento atual M e, após cada iteração, a aresta recém-analisada.

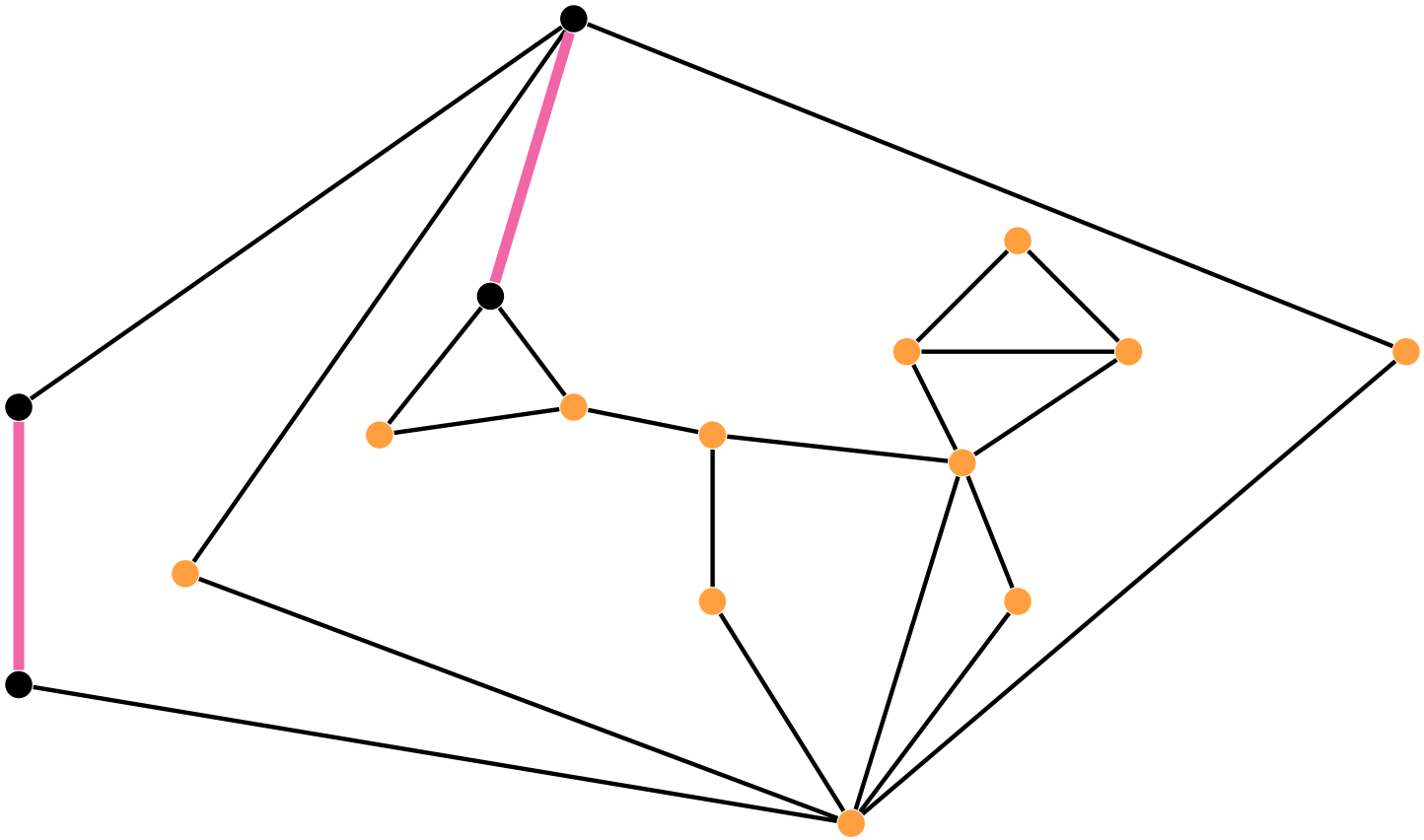
Para cada vértice v da floresta F, olhamos para uma aresta não marcada vw que incide em v.

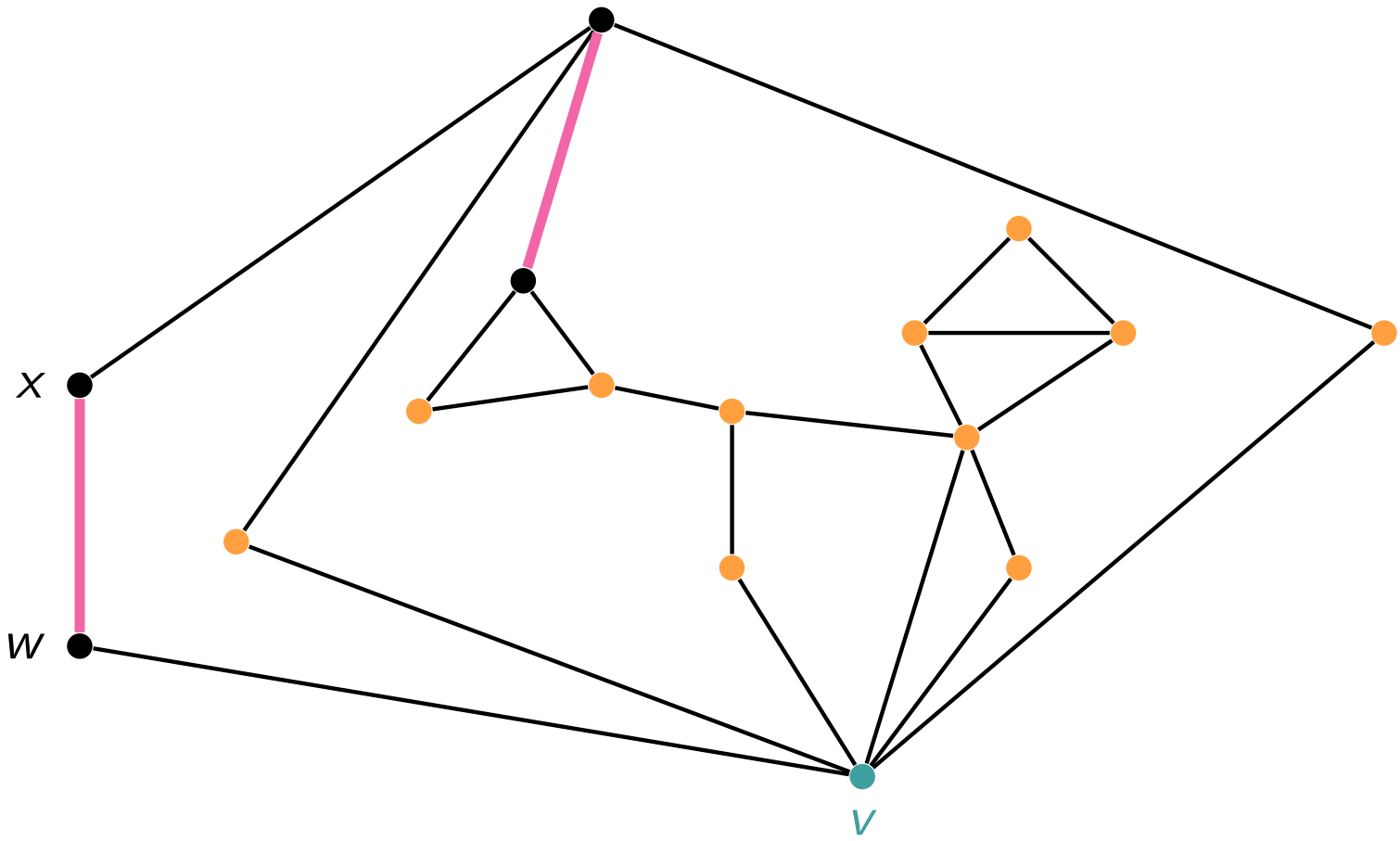
- Se $w \notin F$, então w é extremo de uma aresta wx de M. Adicionamos os vértices w, x e as arestas vw, wx à F.

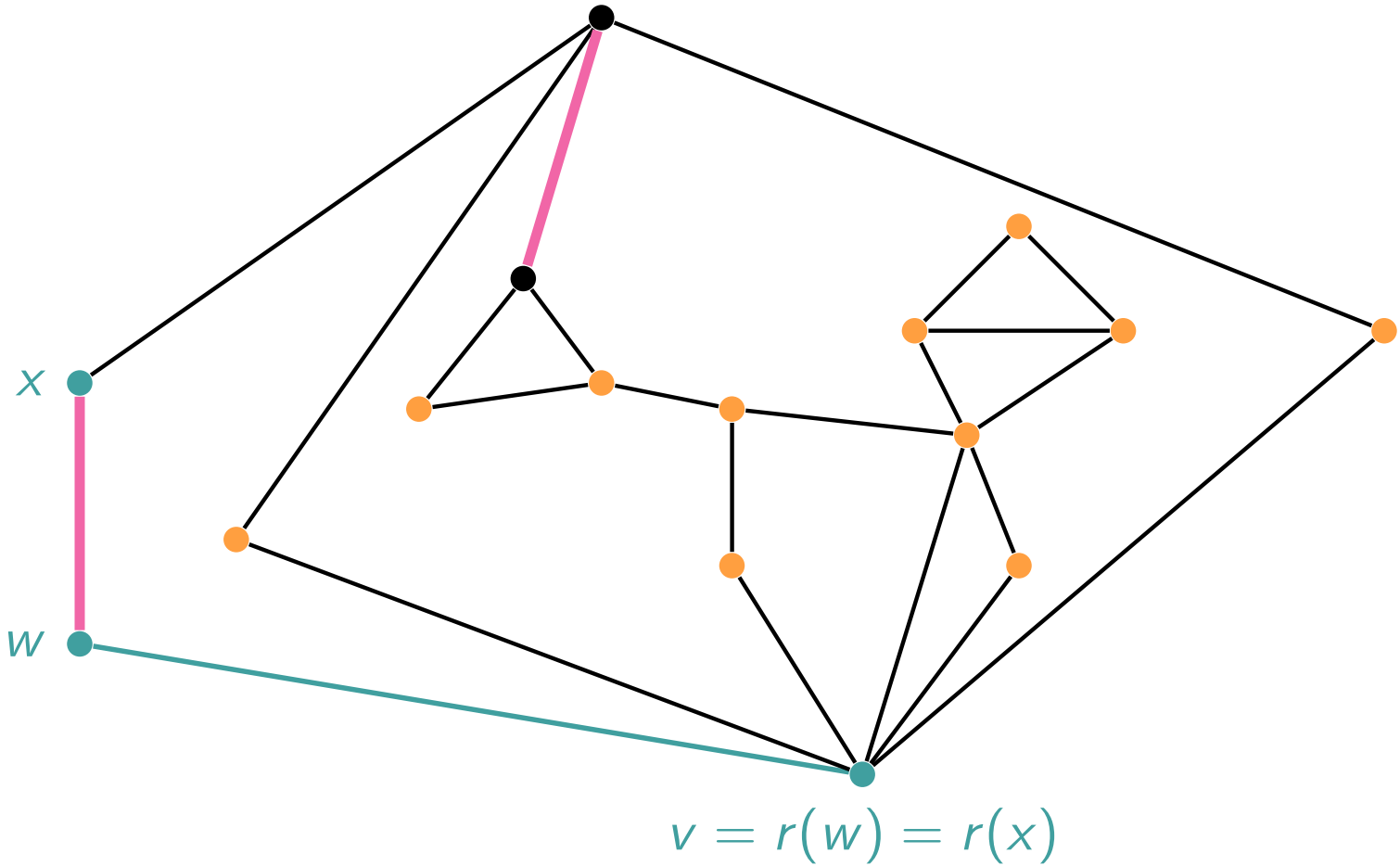
Algoritmo 3: EDMONDS:EXT_ARV(M, F, v, w)

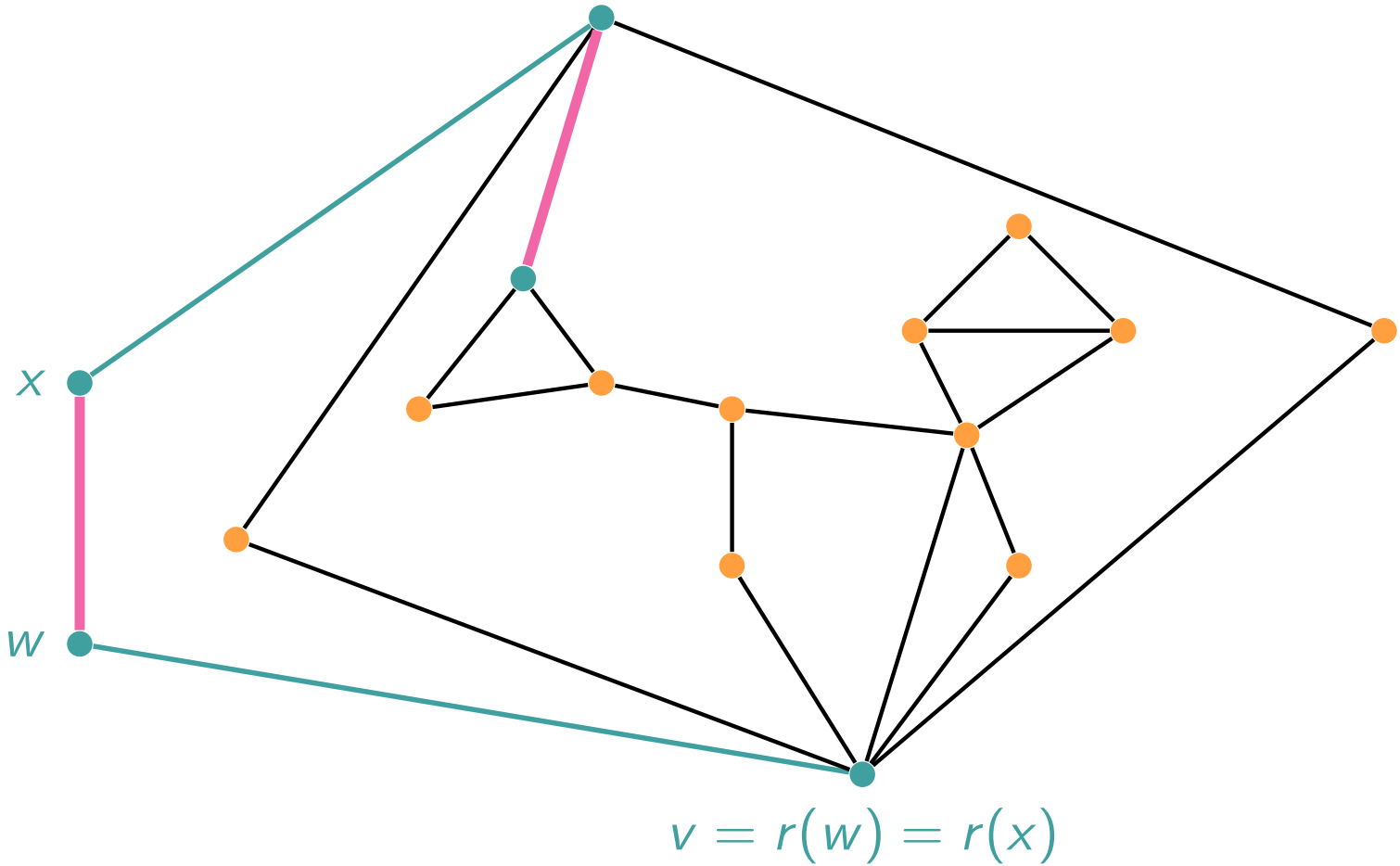
1 faça $x \leftarrow$ vértice adjacente a w em M;
2 faça $T_v \leftarrow T_v \cup (\{w, x\}, \{vw, wx\})$;
3 faça $V' \leftarrow V' \cup \{x\}$;
4 faça $r(w), r(x) \leftarrow r(v)$;






x
w
v


x
w
$v = r(w) = r(x)$


x
w
$v = r(w) = r(x)$

Algoritmo 2: EDMONDS:CAM_AUM(G, M)

```
1  faça F ← ∅;
2  faça V' ← vértices expostos em G;
3  para v ∈ V', faça:
4     T_v ← ({v}, ∅);
5     r(v) ← v;
6     F ← F ∪ T_v;
7  faça A' ← M;
8  para v ∈ V(F), faça:
9    enquanto existir w tal que vw ∈ A\A', faça:
10     se w ∉ V(F), então:
11       EXT_ARV(M, F, v, w);
12     senão:
13       se d(r(w), w) é par, então:
14         se r(v) ≠ r(w), então:
15           P ←EXT_CAM_AUM(F, v, w, r(v), r(w));
16         senão:
17           P ←REC_FLOR(G, M, F, v, w);
18         devolva P;
19      faça A' ← A' ∪ {vw};
20 devolva ∅;
```

Algoritmo de Edmonds: extensão de caminho de aumento

- Se $w \in F$, olhamos para a raiz $r(w)$ de w.

Algoritmo de Edmonds: extensão de caminho de aumento

- Se $w \in F$, olhamos para a raiz $r(w)$ de w.
  - Se a distância $d(w, r(w))$ é ímpar, a estratégia abaixo não funciona. Logo, nada fazemos.

Algoritmo de Edmonds: extensão de caminho de aumento

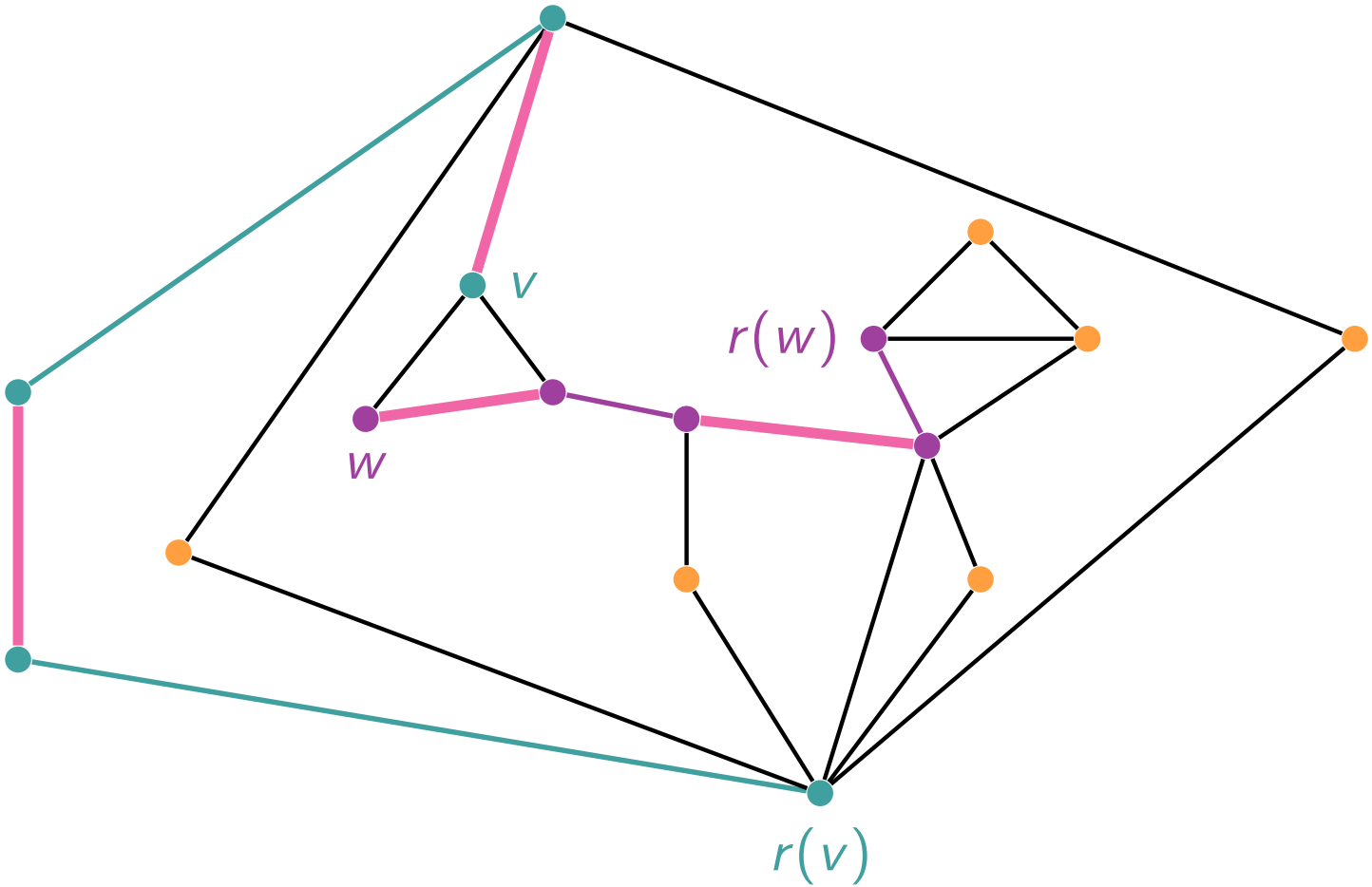
- Se $w \in F$, olhamos para a raiz $r(w)$ de w.
  - Se a distância $d(w, r(w))$ é ímpar, a estratégia abaixo não funciona. Logo, nada fazemos.
  - Se $d(w, r(w))$ é par, o caminho $C(w, r(w))$ de extremos w e $r(w)$ é alternante e sua aresta wx pertence à M.
    De modo análogo, $C(r(v), v)$ é alternante e $uv \in M$.

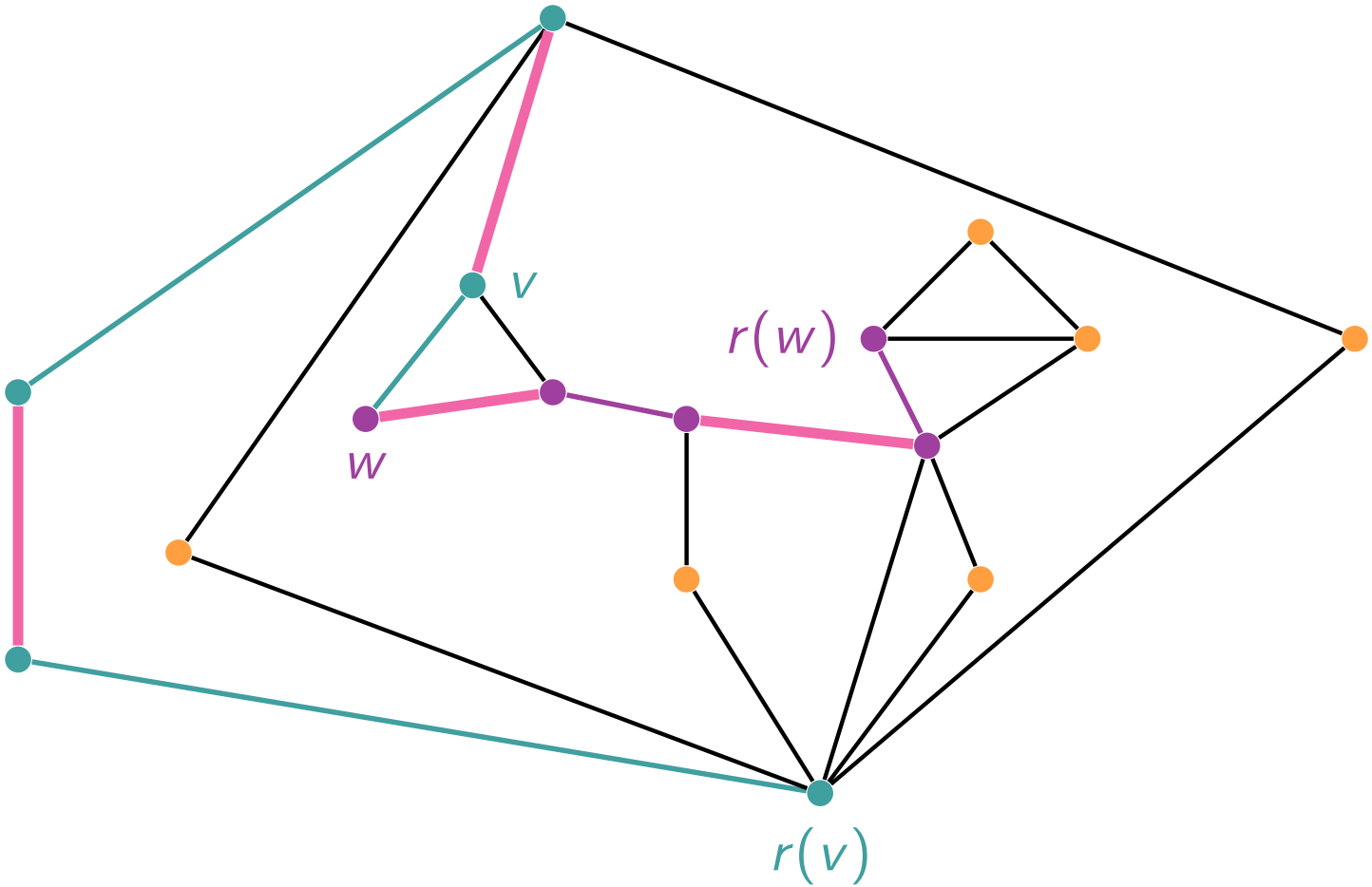
Algoritmo de Edmonds: extensão de caminho de aumento

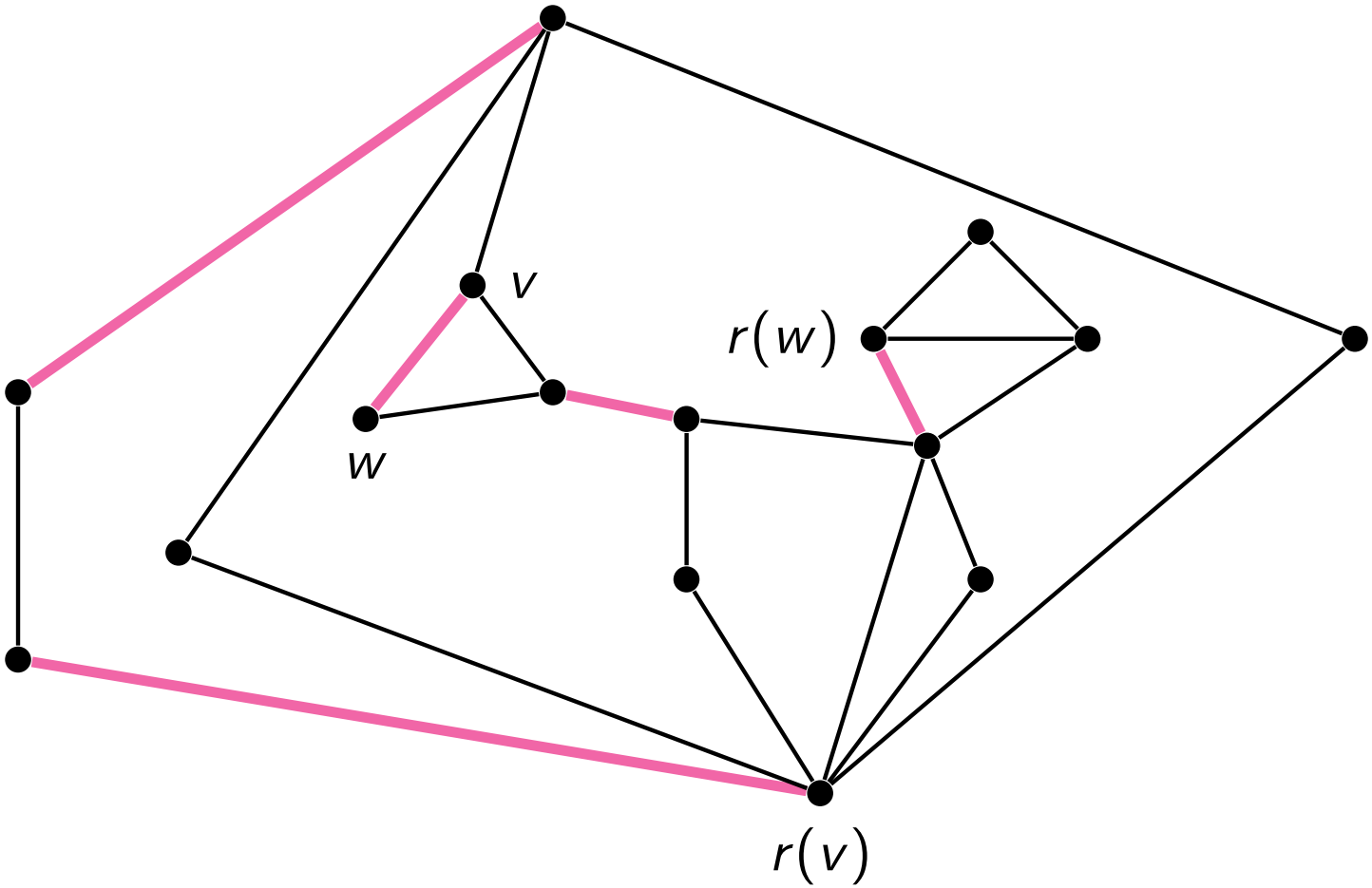
- Se $w \in F$, olhamos para a raiz $r(w)$ de w.
  - Se a distância $d(w, r(w))$ é ímpar, a estratégia abaixo não funciona. Logo, nada fazemos.
  - Se $d(w, r(w))$ é par, o caminho $C(w, r(w))$ de extremos w e $r(w)$ é alternante e sua aresta wx pertence à M.
    De modo análogo, $C(r(v), v)$ é alternante e $uv \in M$.
    - Se $r(w) \neq r(v)$, podemos unir os dois caminhos alternantes com vw e obter um caminho de aumento.

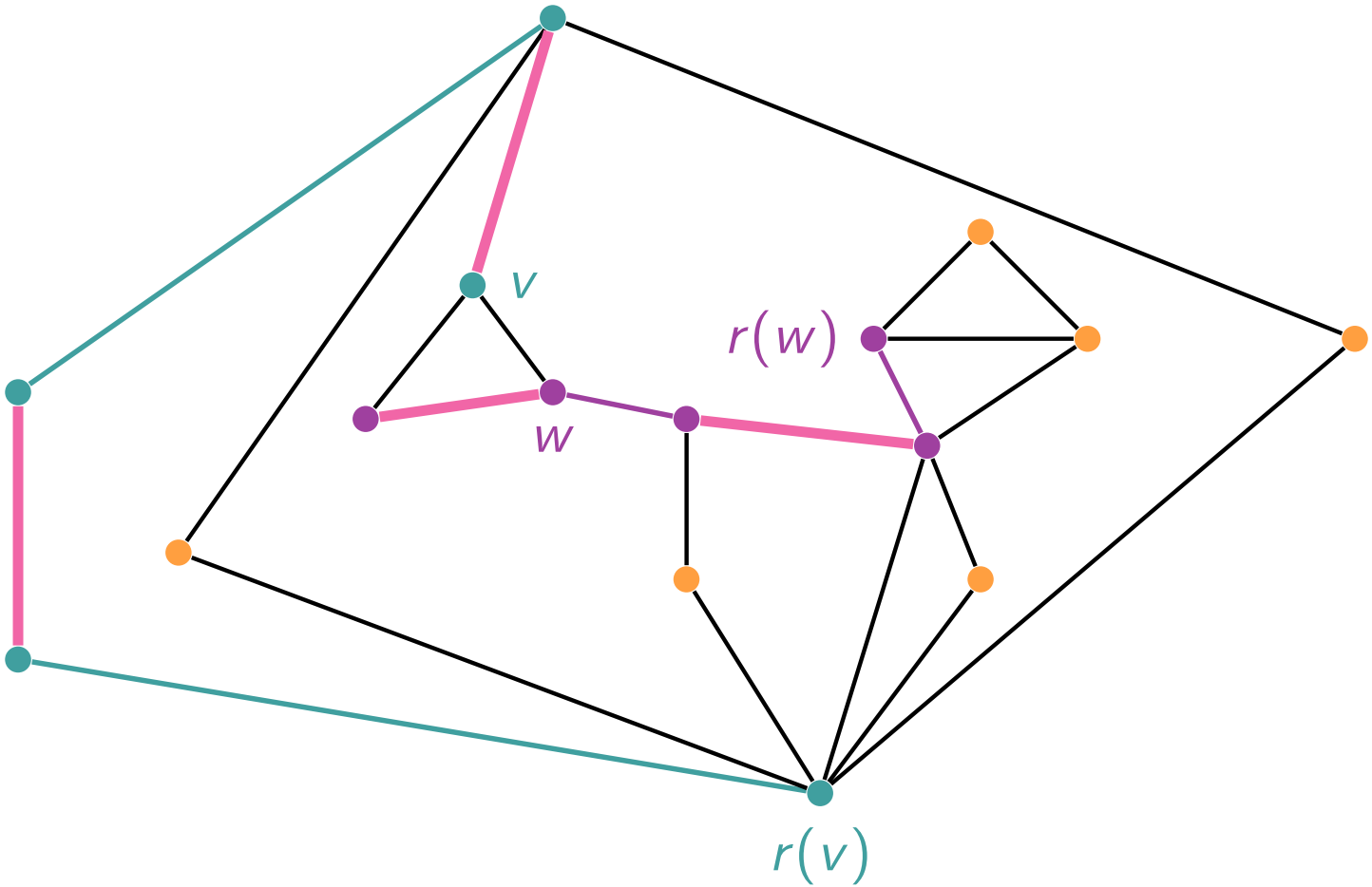
Algoritmo 4: EDMONDS:EXT_CAM_AUM$(F, v, w, r, (v), r(w))$

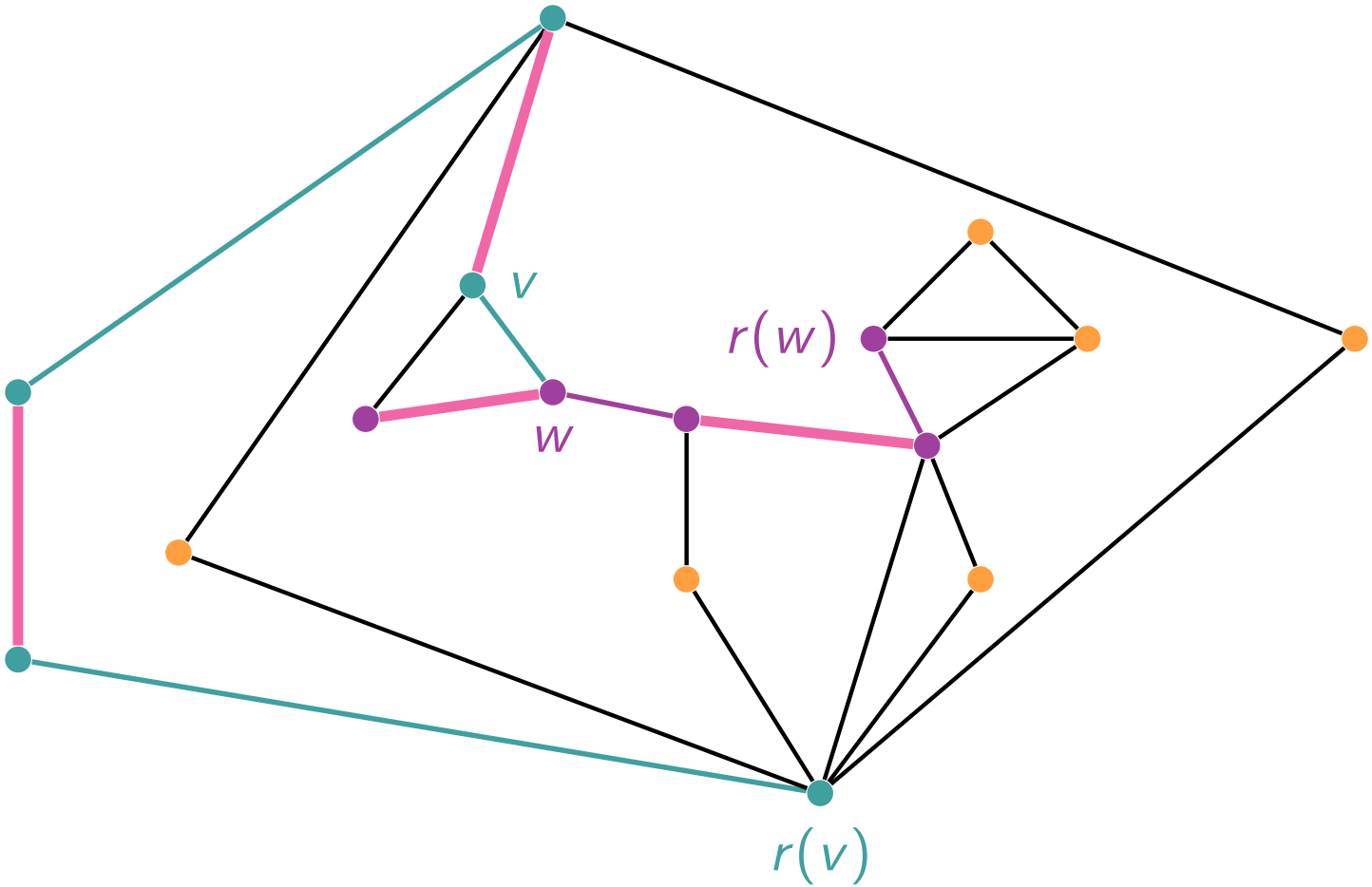
```
1 faça P_v ← C(r(v), r, F);
2 faça P_w ← C(w, r(w), F);
3 faça P ← P_v ∪ {vw} ∪ P_w;
4 devolva P;
```


v
r(w)
w
r(v)


v
r(w)
w
r(v)


v
r(w)
w
r(v)


v
r(w)
w
r(v)


v
r(w)
w
r(v)

Algoritmo 2: EDMONDS:CAM_AUM(G, M)

```
1  faça F ← ∅;
2  faça V' ← vértices expostos em G;
3  para v ∈ V', faça:
4     T_v ← ({v}, ∅);
5     r(v) ← v;
6     F ← F ∪ T_v;
7  faça A' ← M;
8  para v ∈ V(F), faça:
9    enquanto existir w tal que vw ∈ A\A', faça:
10     se w ∉ V(F), então:
11       EXT_ARV(M, F, v, w);
12     senão:
13       se d(r(w), w) é par, então:
14         se r(v) ≠ r(w), então:
15           P ←EXT_CAM_AUM(F, v, w, r(v), r(w));
16         senão:
17           P ←REC_FLOR(G, M, F, v, w);
18         devolva P;
19      faça A' ← A' ∪ {vw};
20 devolva ∅;
```

Flor ($=$*blossom*)

O caso mais interessante é $d(r(w), w)$ par e $r(v) = r(w)$, que justifica o nome do algoritmo e motiva a teoria a seguir.

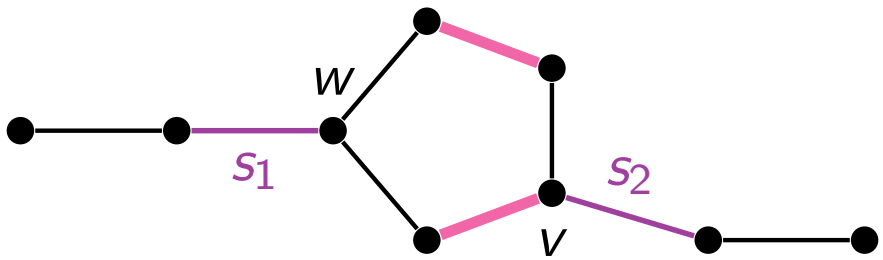
Flor (=*blossom*)

O caso mais interessante é $d(r(w), w)$ par e $r(v) = r(w)$, que justifica o nome do algoritmo e motiva a teoria a seguir.

Uma **flor** B é um circuito alternado de tamanho ímpar.

O único vértice w da flor no qual incidem duas arestas não emparelhadas é chamado **base** de B.

Denotamos por **haste** uma aresta s não pertencente a B incidente em um vértice v da flor.



Contração de circuito

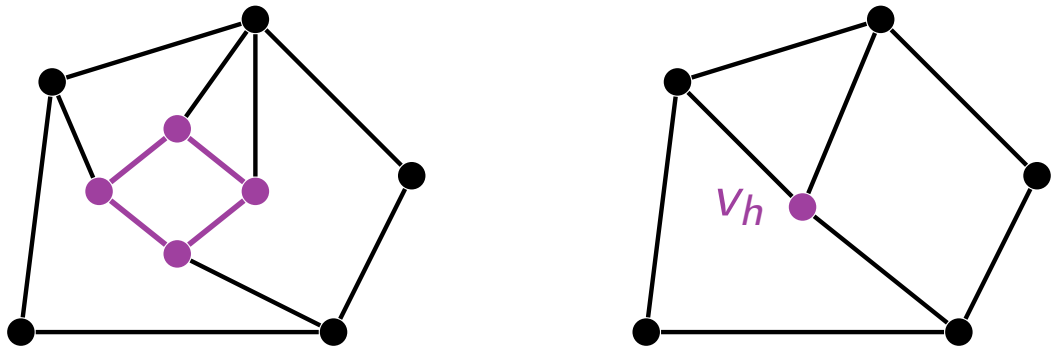
Seja H um circuito em um grafo G. A **contração** de H é a operação que produz um novo grafo G' tal que:

- $V(G') = (V(G) \setminus V(H)) \cup \{v_h\}$;

Contração de circuito

Seja H um circuito em um grafo G. A **contração** de H é a operação que produz um novo grafo G' tal que:

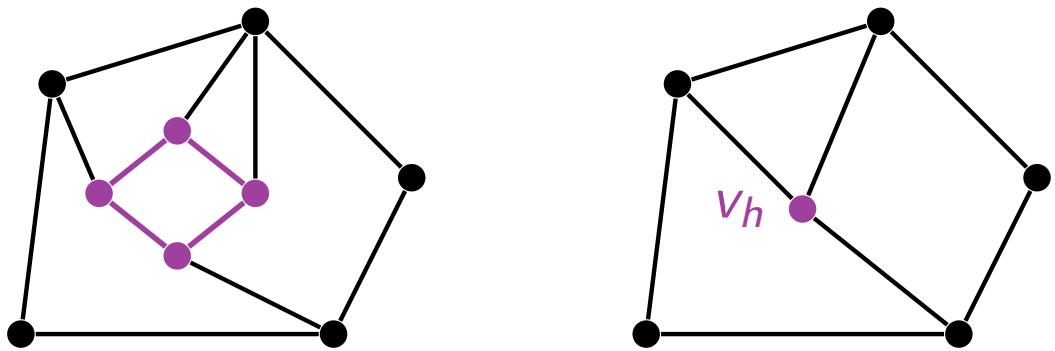
- $V(G') = (V(G) \setminus V(H)) \cup \{v_h\}$;
- para cada aresta vw de G com $v \in V(G) \setminus V(H)$, se $w \in V(G) \setminus V(H)$, então vw é aresta de G', e se $w \in V(H)$, então vv_h é aresta de G'.



Contração de circuito

Seja H um circuito em um grafo G. A **contração** de H é a operação que produz um novo grafo G' tal que:

- $V(G') = (V(G) \setminus V(H)) \cup \{v_h\}$;
- para cada aresta vw de G com $v \in V(G) \setminus V(H)$, se $w \in V(G) \setminus V(H)$, então vw é aresta de G', e se $w \in V(H)$, então vv_h é aresta de G'.



O grafo G' obtido a partir de G por uma sequência de contrações de flores B_i é chamado grafo **derivado**.

Algoritmo de Edmonds: recursão de flor

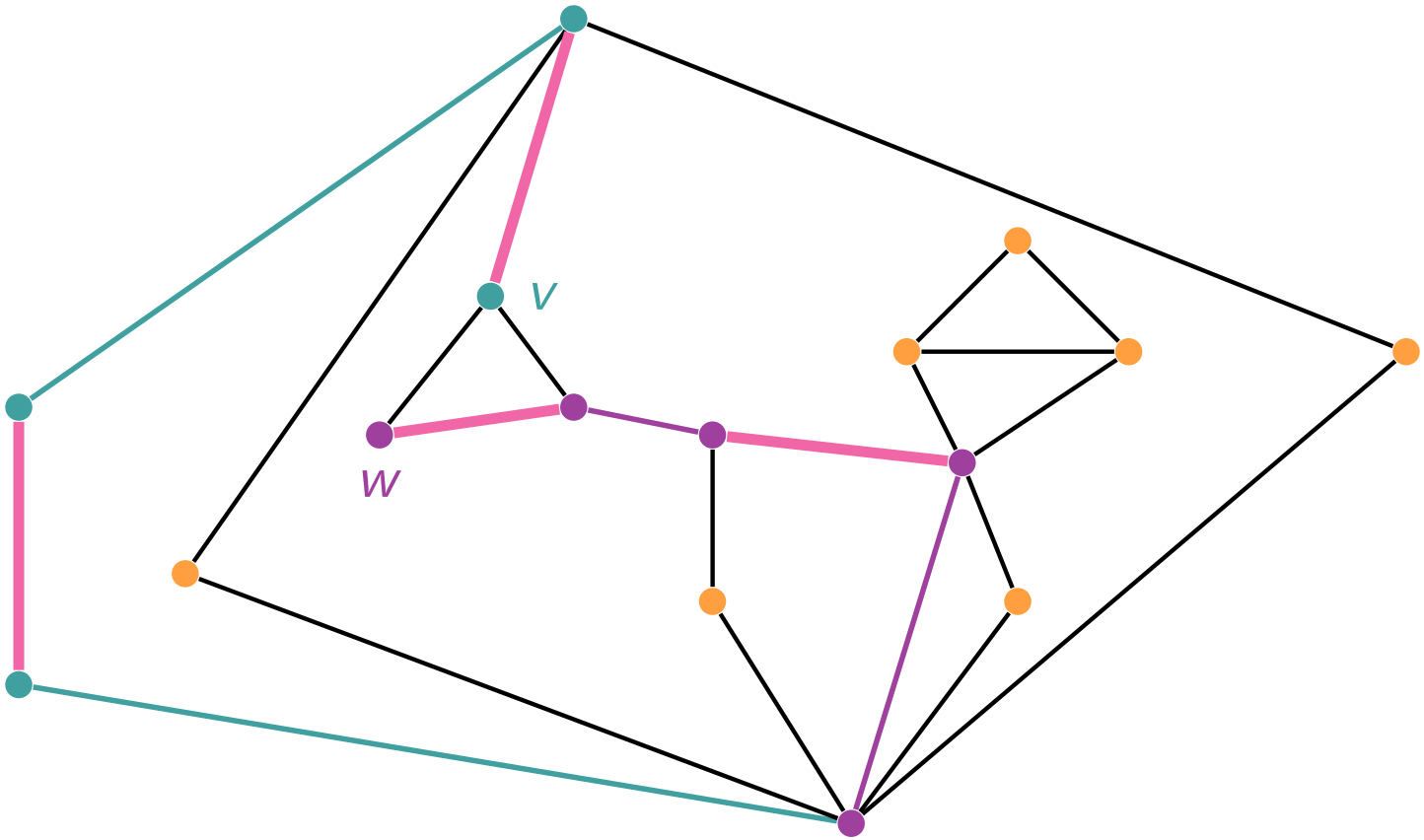
Ideia geral: contrair cada flor em seu vértice base v_{b_i} e continuar o mapeamento dos caminhos de aumento.

Algoritmo de Edmonds: recursão de flor

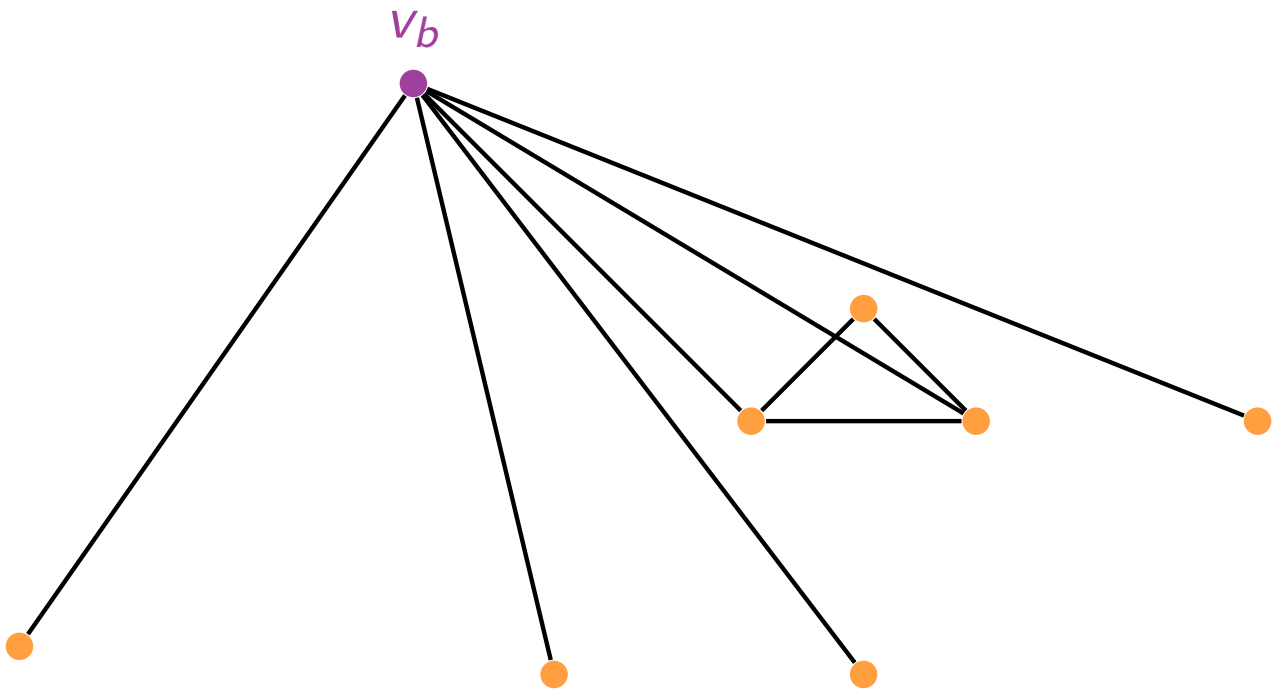
Ideia geral: contrair cada flor em seu vértice base v_{b_i} e continuar o mapeamento dos caminhos de aumento.
Se um caminho encontrado contiver algum v_{b_i}, desfazemos a respectiva contração e atravessamos a flor estendida ($=$*lifted*) de modo que o caminho permaneça de aumento.

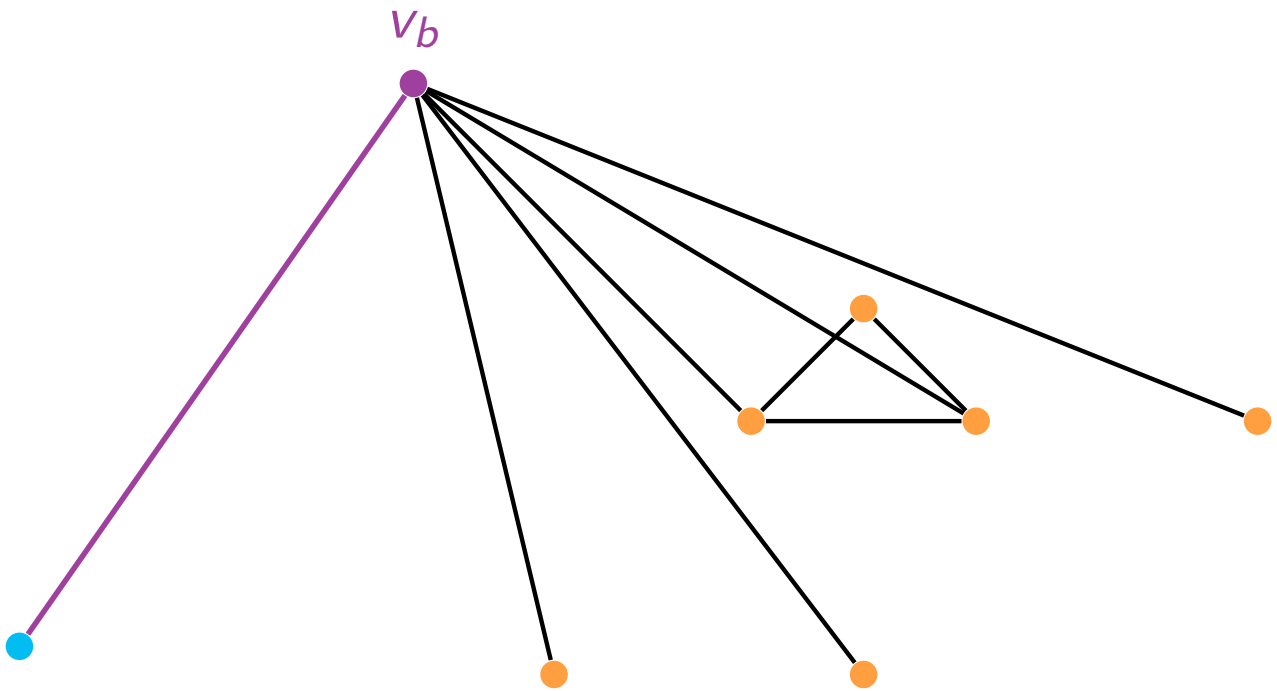
Algoritmo 5: EDMONDS:REC_FLOR(G, M, F, v, w)

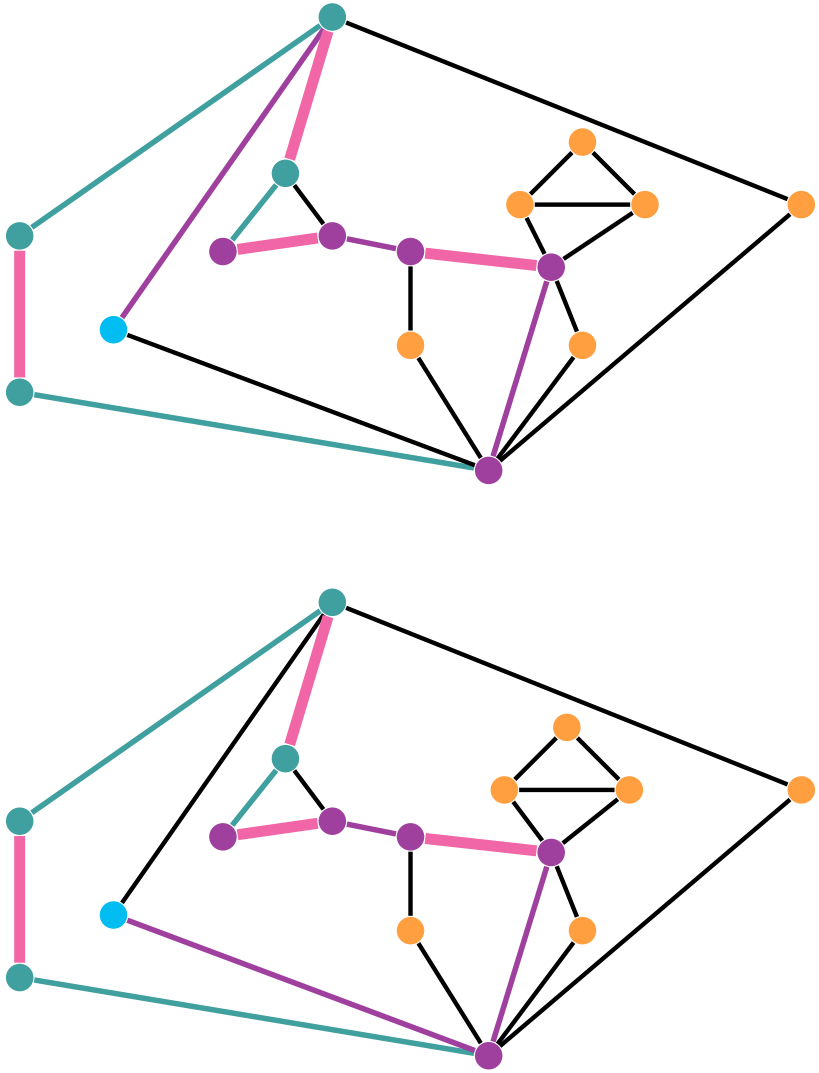
```
1 faça B ← C(v, w, F) ∪ vw;
2 faça G' ← G com B contraída em w;
3 faça M' ← M com B contraída em w;
4 faça P' ←CAM_AUM(G', M');
5 se w ∈ P', então:
6   P ← P' com w estendido para B;
7   devolva P;
8 senão:
9   devolva P';
```

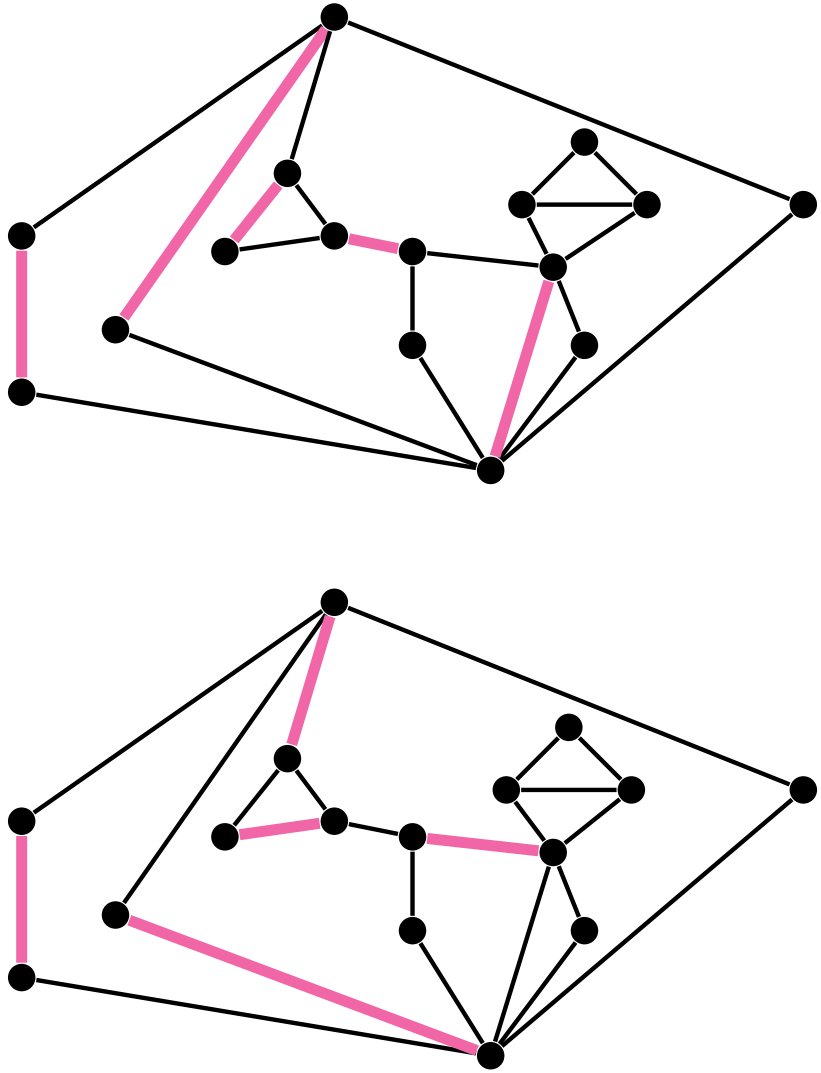


$r(v) = r(w)$


v_b


v_b





O seguinte teorema garante que o último passo do algoritmo de recursão de flor é possível.

Teorema

Seja G' o grafo derivado de G. Então G tem um caminho de aumento se, e somente se, G' tem um caminho de aumento.

O seguinte teorema garante que o último passo do algoritmo de recursão de flor é possível.

Teorema

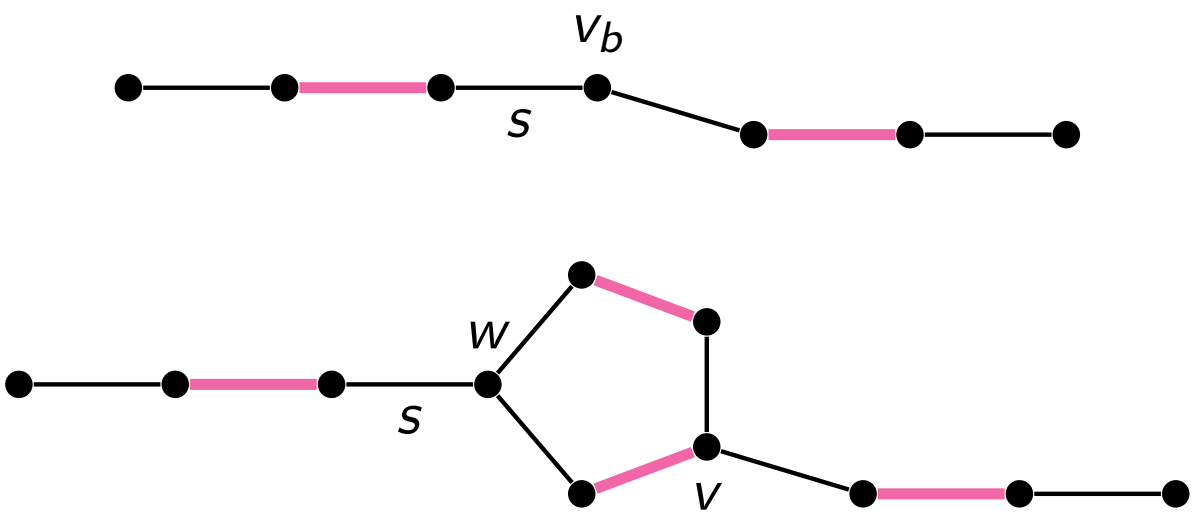
Seja G' o grafo derivado de G. Então G tem um caminho de aumento se, e somente se, G' tem um caminho de aumento.

($\Leftarrow$) Considere um caminho de aumento P' em G' que atravessa uma flor contraída v_b (caso contrário, o caminho é o mesmo em G e em G', e o teorema vale trivialmente). Temos dois casos:

Caso 1: Se v_b é um extremo de P', então a flor estendida B possui uma única haste s incidente.

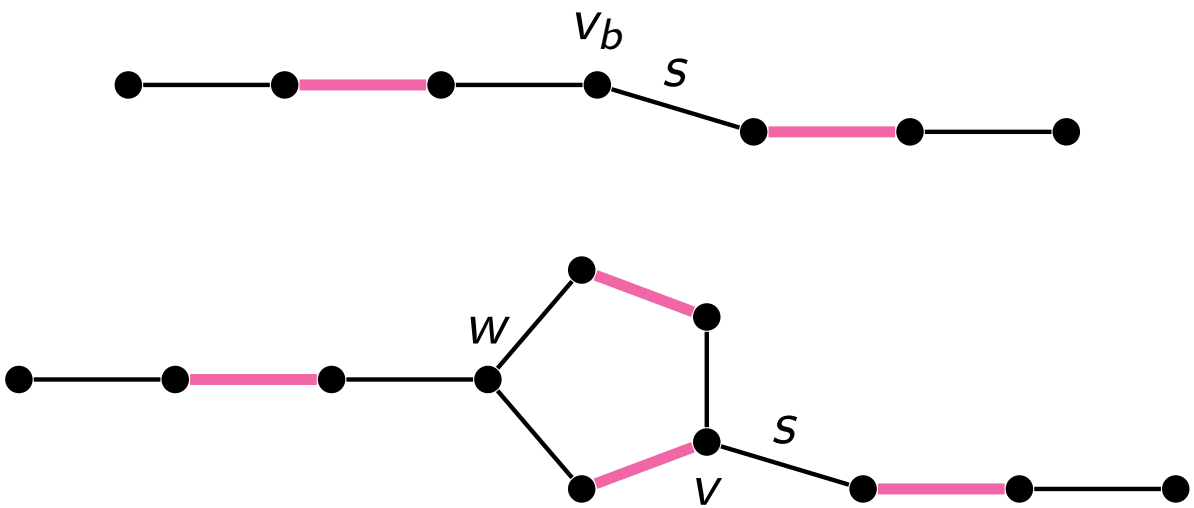
Caso 1: Se v_b é um extremo de P', então a flor estendida B possui uma única haste s incidente. Note que $s \notin M$ uma vez que é uma aresta final (por definição, não emparelhada) do caminho de aumento P'.

Caso 1: Se v_b é um extremo de P', então a flor estendida B possui uma única haste s incidente. Note que $s \notin M$ uma vez que é uma aresta final (por definição, não emparelhada) do caminho de aumento P'.
Se s incide na base w de B, então P' termina em w e obtemos o caminho de aumento P em G (basta trocar v_b por w em P').

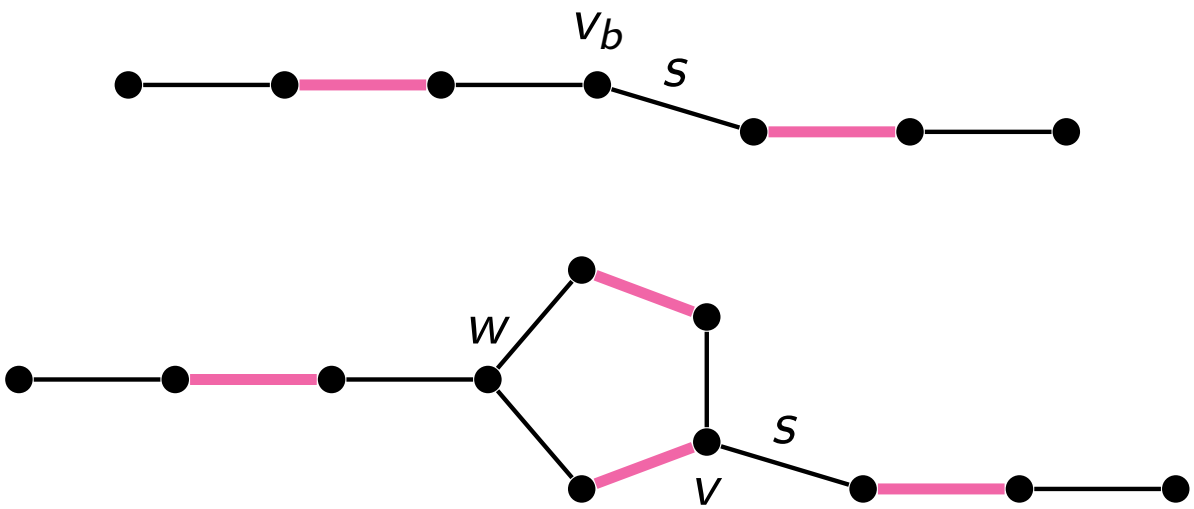


Se s não incide na base, mas em um vértice v de B, podemos estender P'. Como v não é base, uma das arestas de B incidentes em v estão em M.

Se s não incide na base, mas em um vértice v de B, podemos estender P'. Como v não é base, uma das arestas de B incidentes em v estão em M. Estendemos P' nessa direção, adicionando pares de arestas *matched-unmatched* até atingir w.



Se s não incide na base, mas em um vértice v de B, podemos estender P'. Como v não é base, uma das arestas de B incidentes em v estão em M. Estendemos P' nessa direção, adicionando pares de arestas *matched-unmatched* até atingir w.



Como adicionamos apenas arestas alternadas, o caminho estendido P é aumentador em G.

Considere o seguinte

Lema auxiliar

Se v_b não é um extremo de P', uma das hastes deve incidir na base w.

Considere o seguinte

Lema auxiliar

Se v_b não é um extremo de P', uma das hastes deve incidir na base w.

Note que exatamente uma das hastes, digamos s, está emparelhada.

Considere o seguinte

Lema auxiliar

Se v_b não é um extremo de P', uma das hastes deve incidir na base w.

Note que exatamente uma das hastes, digamos s, está emparelhada. Afirmamos que s deve incidir em w.

Considere o seguinte

Lema auxiliar

Se v_b não é um extremo de P', uma das hastes deve incidir na base w.

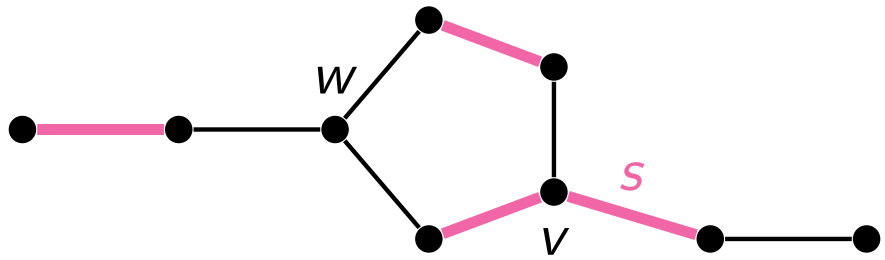
Note que exatamente uma das hastes, digamos s, está emparelhada. Afirmamos que s deve incidir em w. De fato, suponha que s incida em um vértice $v \neq w$.

Considere o seguinte

Lema auxiliar

Se v_b não é um extremo de P', uma das hastes deve incidir na base w.

Note que exatamente uma das hastes, digamos s, está emparelhada. Afirmamos que s deve incidir em w. De fato, suponha que s incida em um vértice $v \neq w$.



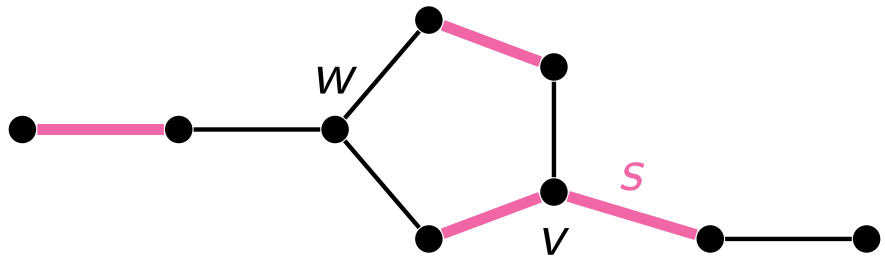
Então v é extremo de duas arestas de M, o que contradiz o fato de M ser um emparelhamento.

Considere o seguinte

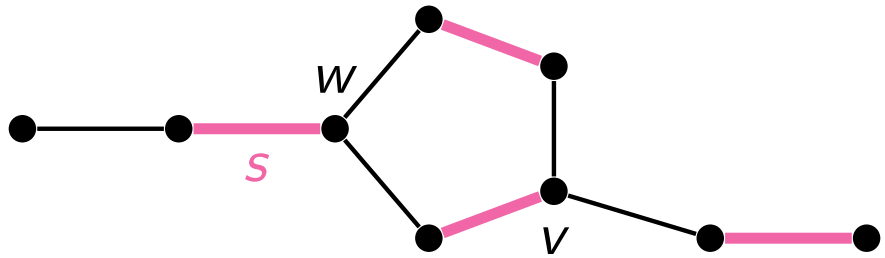
Lema auxiliar

Se v_b não é um extremo de P', uma das hastes deve incidir na base w.

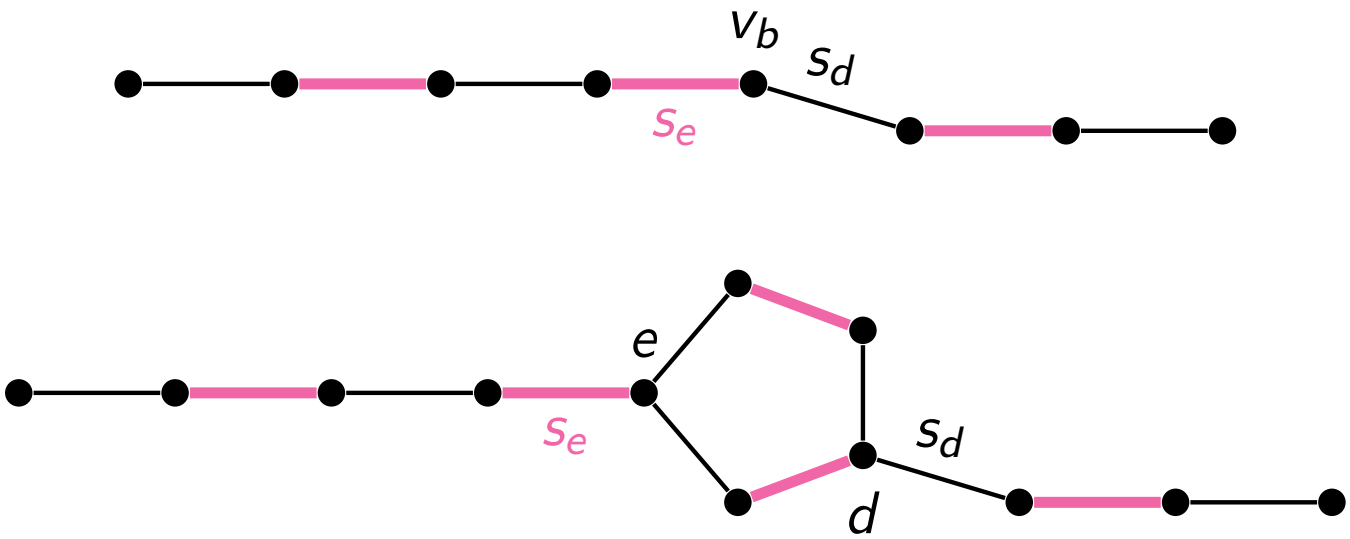
Note que exatamente uma das hastes, digamos s, está emparelhada. Afirmamos que s deve incidir em w. De fato, suponha que s incida em um vértice $v \neq w$.



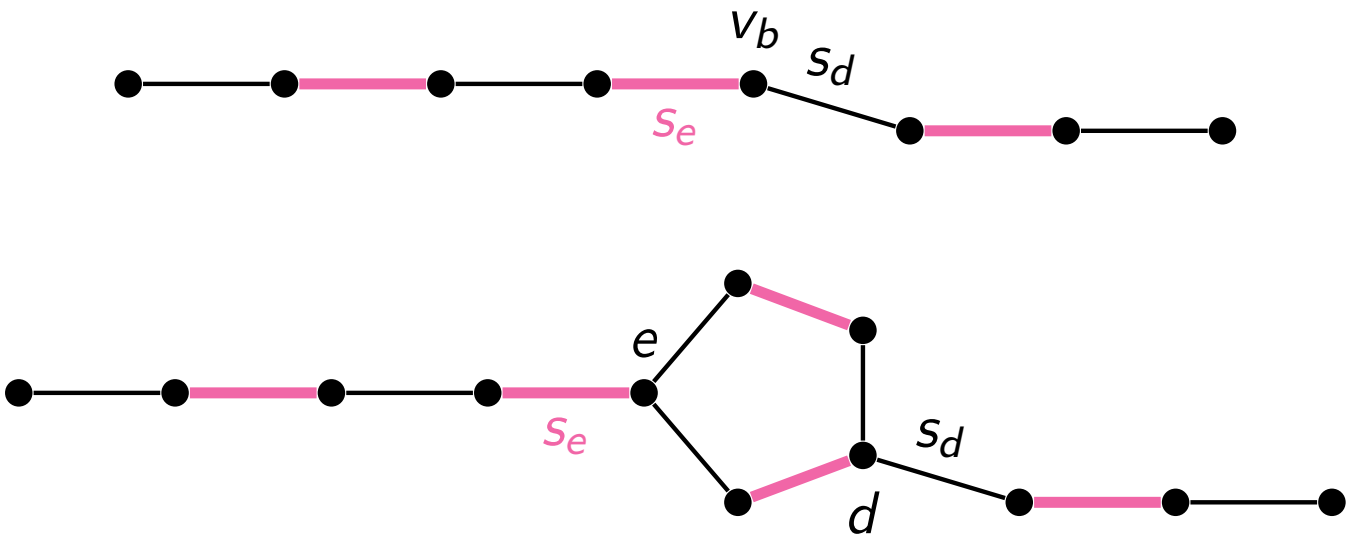
Então v é extremo de duas arestas de M, o que contradiz o fato de M ser um emparelhamento. Portanto, s deve incidir na base w.



Caso 2: Se v_b não é um extremo de P', sejam d, e os vértices nos quais incidem as hastes direita s_d e esquerda s_e, respectivamente.

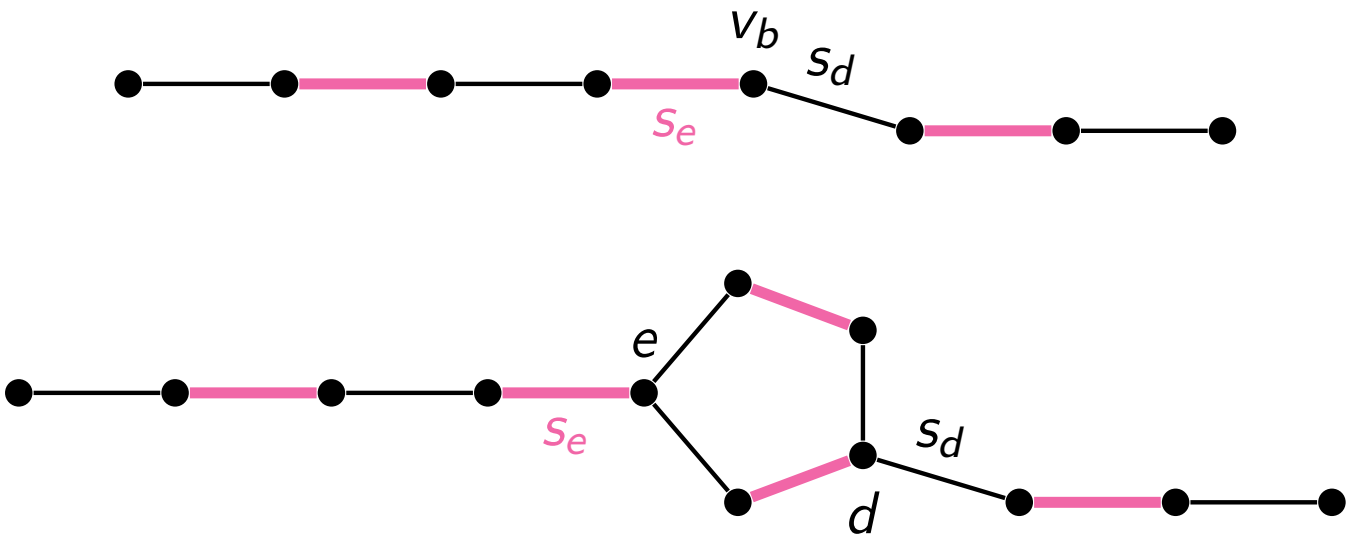


Caso 2: Se v_b não é um extremo de P', sejam d, e os vértices nos quais incidem as hastes direita s_d e esquerda s_e, respectivamente.



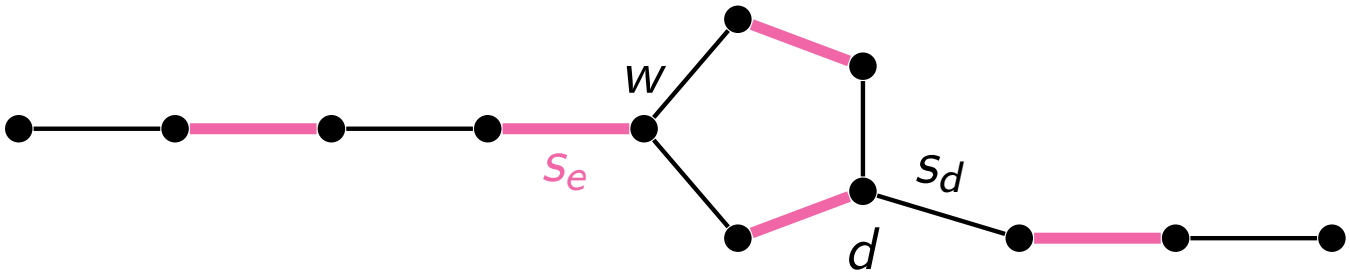
Veja que, como B é um circuito ímpar, sempre existe um caminho par C de extremos d, e.

Caso 2: Se v_b não é um extremo de P', sejam d, e os vértices nos quais incidem as hastes direita s_d e esquerda s_e, respectivamente.

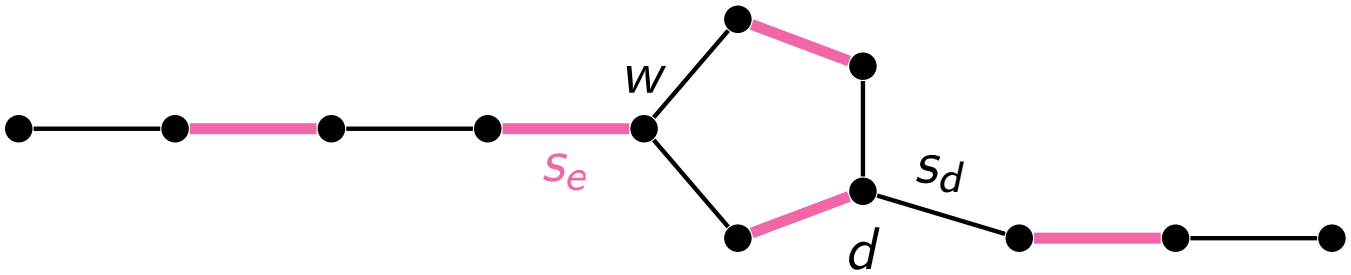


Veja que, como B é um circuito ímpar, sempre existe um caminho par C de extremos d, e. Daí, podemos obter P percorrendo P' até e, depois C até d, seguido do restante de P'.

Pelo lema auxiliar, a haste emparelhada deve incidir na base w de B. Suponha, sem perda de generalidade, $w = e$.

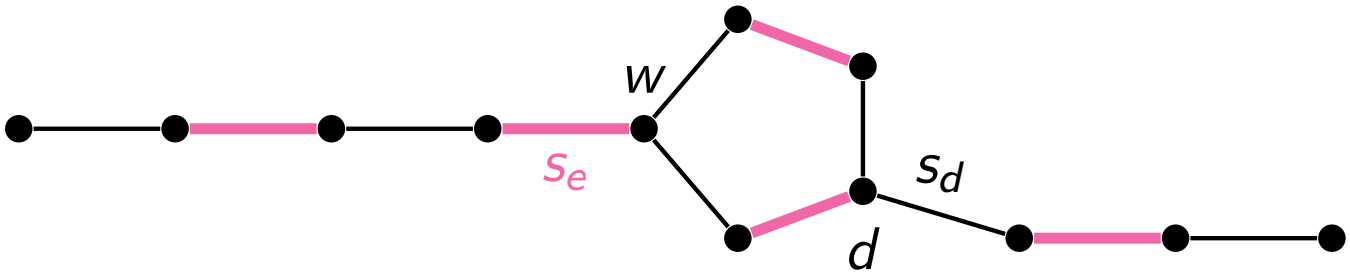


Pelo lema auxiliar, a haste emparelhada deve incidir na base w de B. Suponha, sem perda de generalidade, $w = e$.



Disso é garantido que, ao percorrer C de e para d, incluimos pares de arestas *unmatched-matched*. Logo, P é aumentador em G.

Pelo lema auxiliar, a haste emparelhada deve incidir na base w de B. Suponha, sem perda de generalidade, $w = e$.

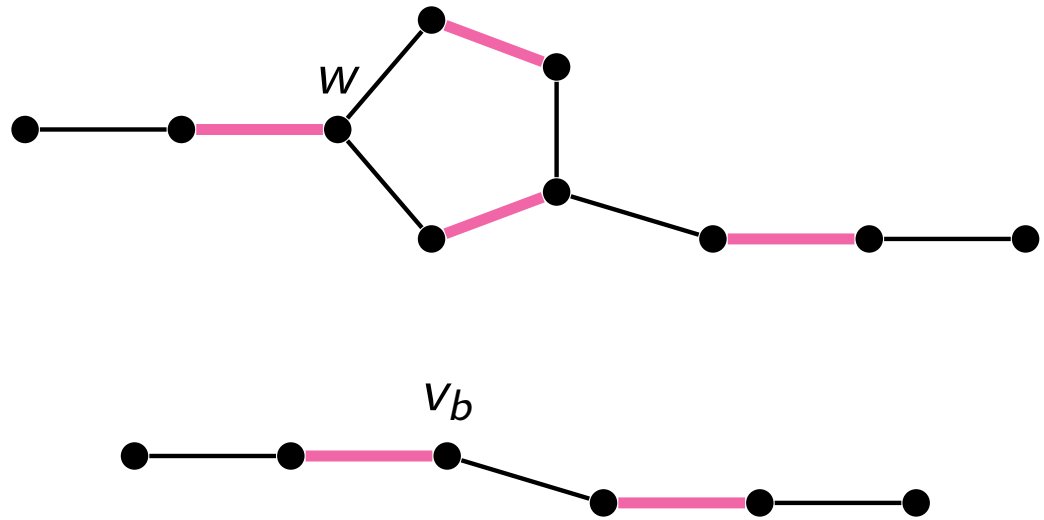


Disso é garantido que, ao percorrer C de e para d, incluimos pares de arestas *unmatched-matched*. Logo, P é aumentador em G.

Portanto, se G' tem um caminho de aumento, então G tem um caminho de aumento.

($\Rightarrow$) Note que se as arestas de uma flor B não são parte de um caminho de aumento P em G, então $P' = P$ também é de aumento em G'.

($\Rightarrow$) Note que se as arestas de uma flor B não são parte de um caminho de aumento P em G, então $P' = P$ também é de aumento em G'.
Se P inclui arestas de B, contraindo B em v_b, eliminamos uma quantidade par de arestas de P (caso contrário, o lema auxiliar seria contradito). Daí o caminho P' obtido é aumentador em G'. Isso completa a demonstração.



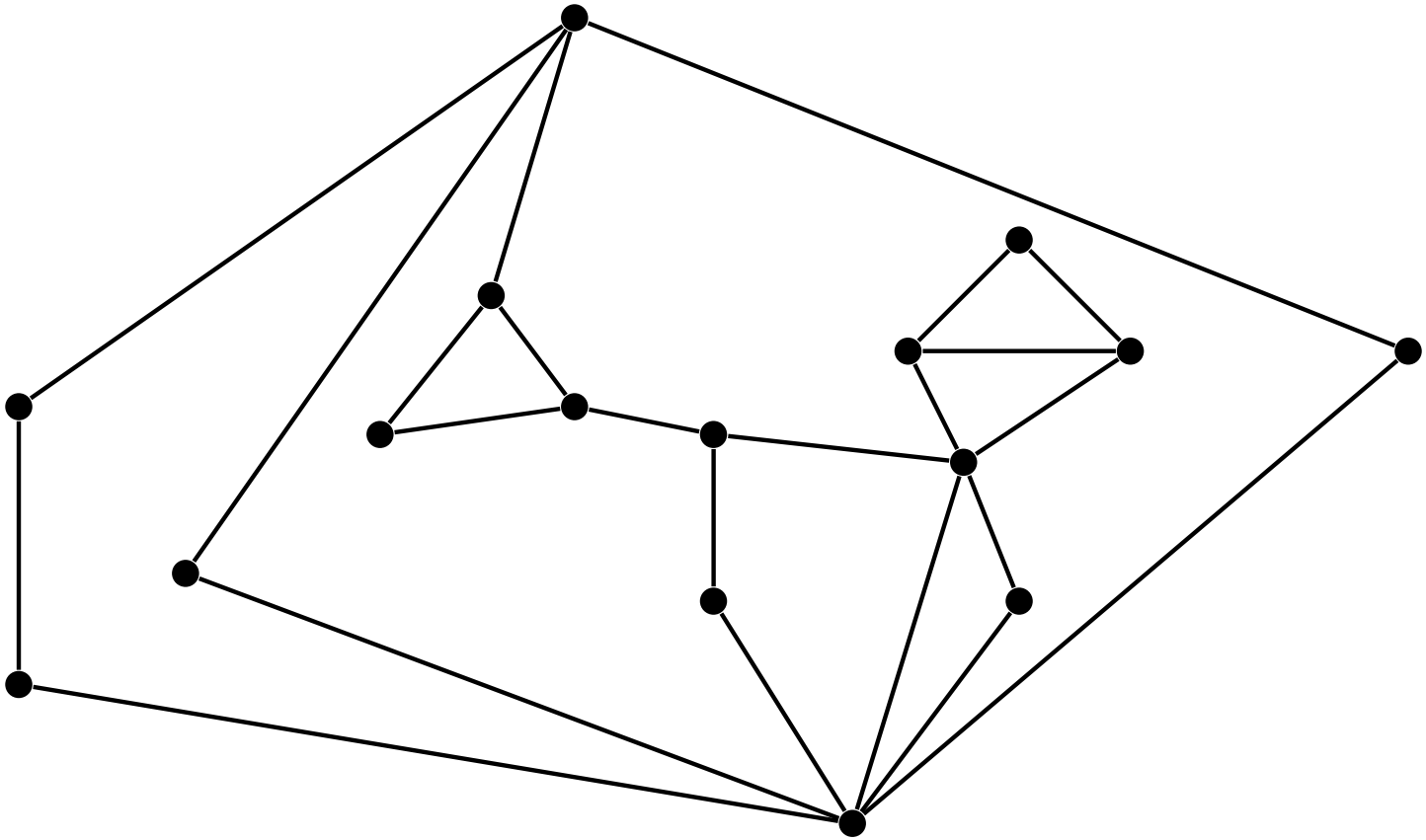
Algoritmo das flores de Edmonds

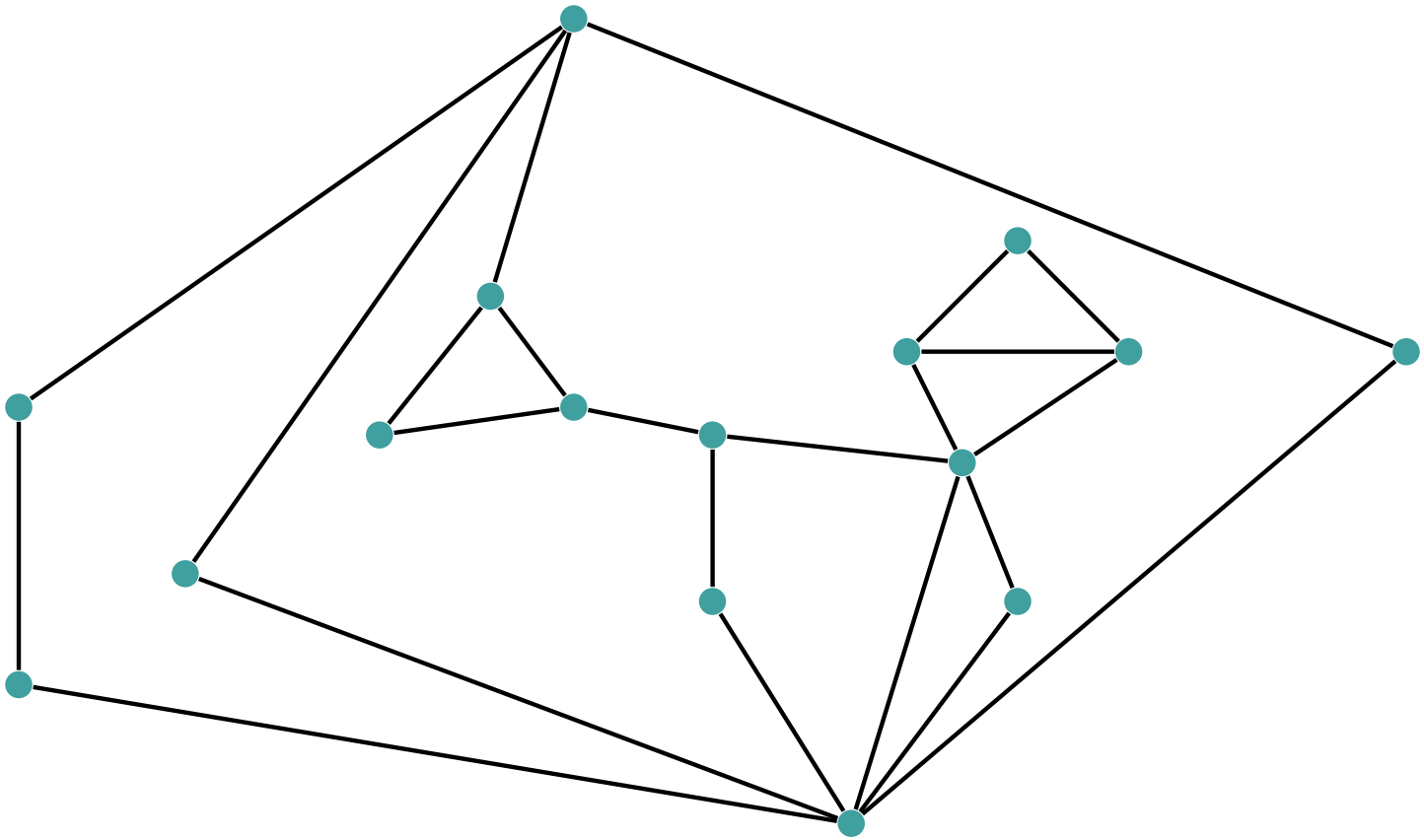
Algoritmo 1: EDMONDS:EMP_MAX(G, M)

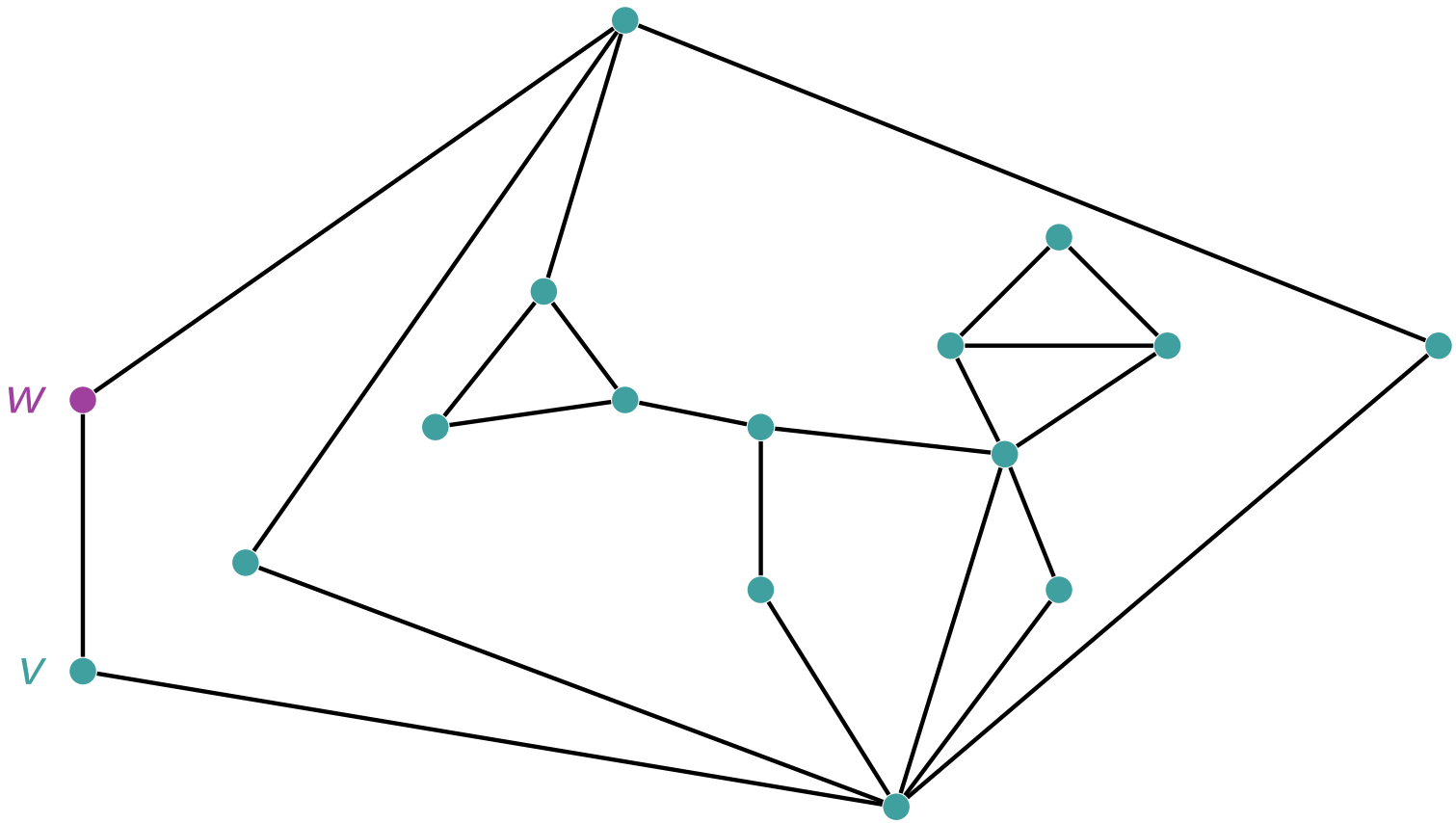
```
1 faça M ← ∅;
2 faça P ←CAM_AUM(G, M);
3 se P = ∅, então:
4    devolva M;
5 senão:
6    M ← M ⊕ A(P);
7    devolva EMP_MAX(G, M);
```

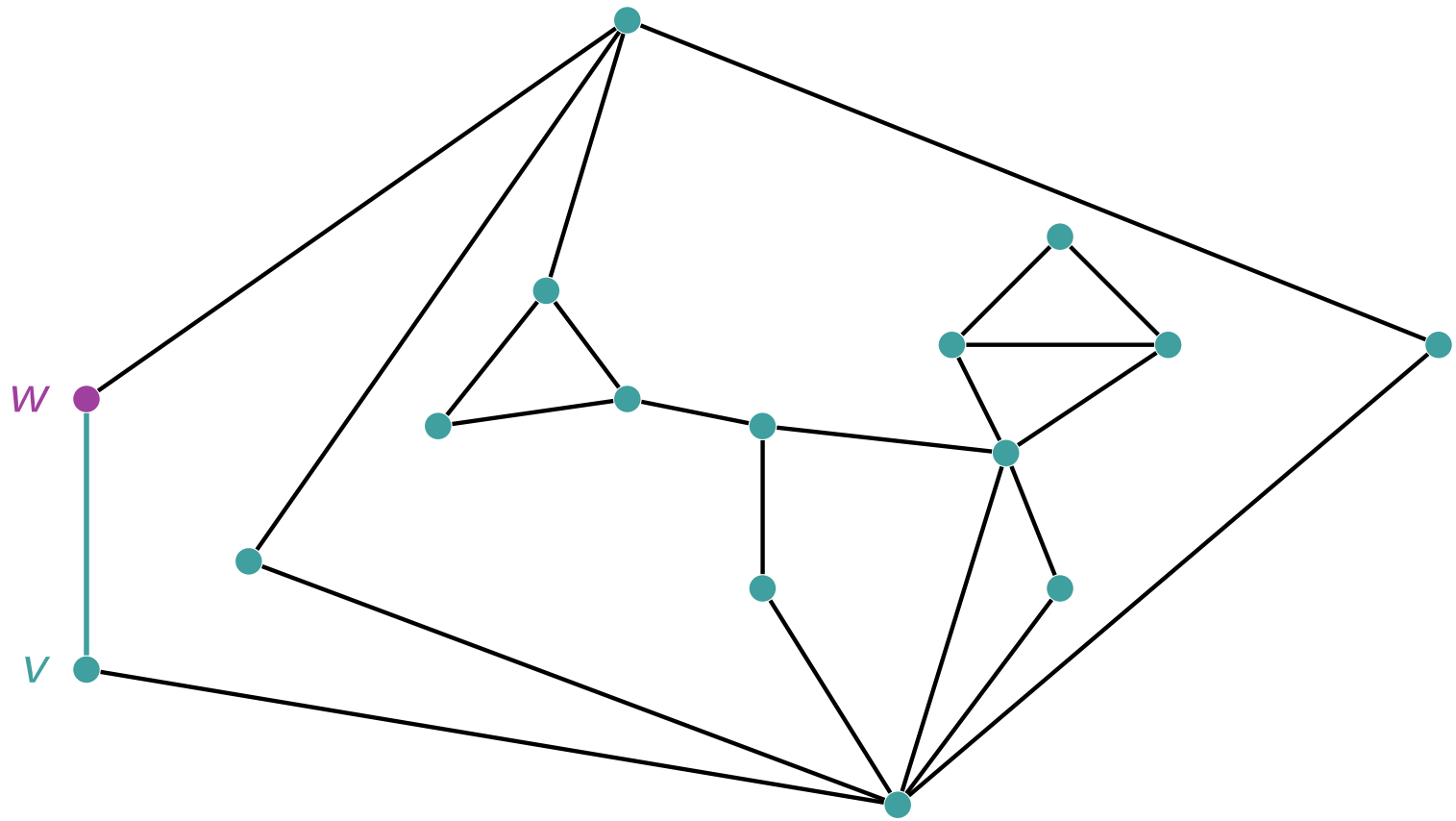
Algoritmo 2: EDMONDS:CAM_AUM(G, M)

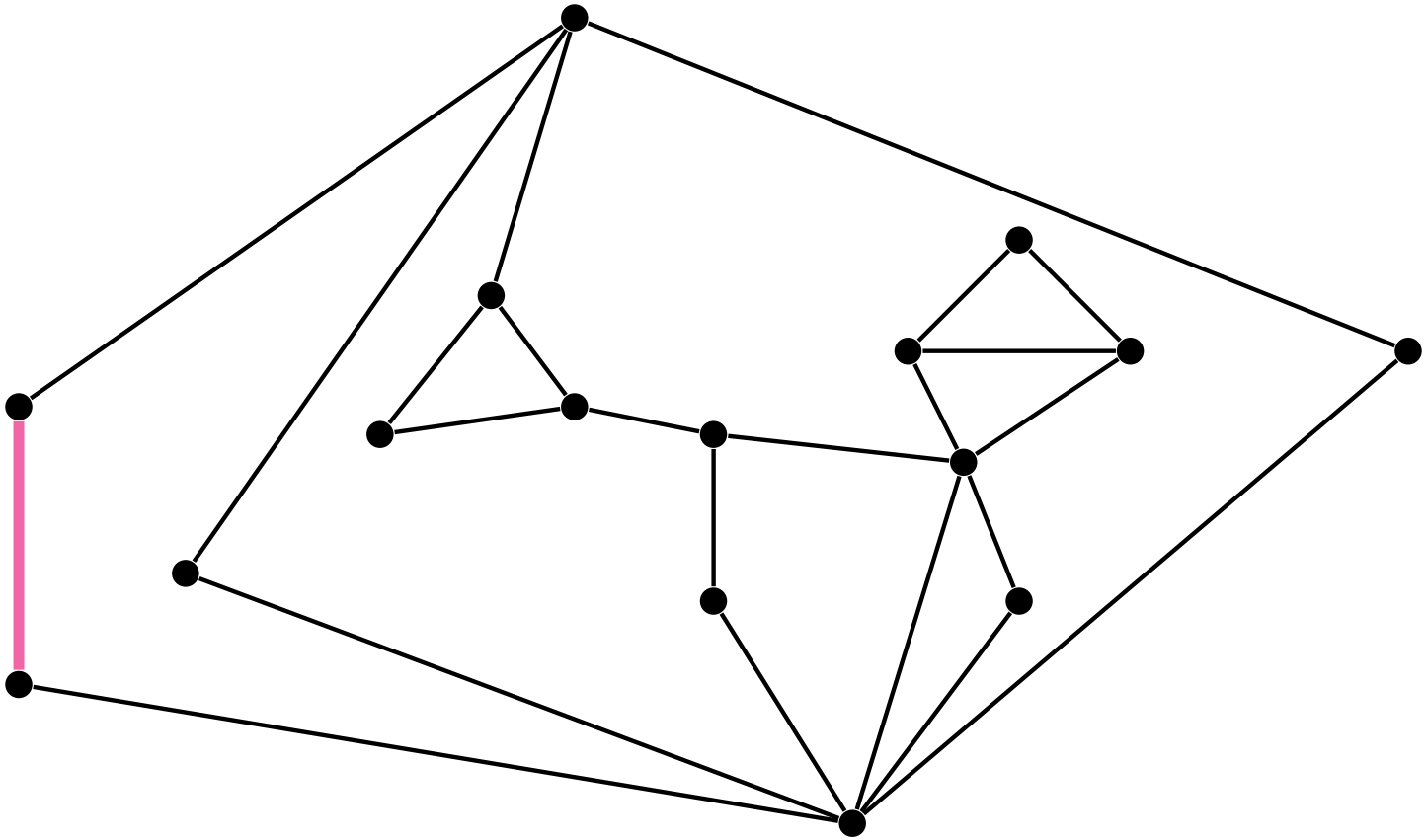
```
1  faça F ← ∅;
2  faça V' ← vértices expostos em G;
3  para v ∈ V', faça:
4     T_v ← ({v}, ∅);
5     r(v) ← v;
6     F ← F ∪ T_v;
7  faça A' ← M;
8  para v ∈ V(F), faça:
9    enquanto existir w tal que vw ∈ A\A', faça:
10     se w ∉ V(F), então:
11       EXT_ARV(M, F, v, w);
12     senão:
13       se d(r(w), w) é par, então:
14         se r(v) ≠ r(w), então:
15           P ←EXT_CAM_AUM(F, v, w, r(v), r(w));
16         senão:
17           P ←REC_FLOR(G, M, F, v, w);
18         devolva P;
19      faça A' ← A' ∪ {vw};
20 devolva ∅;
```

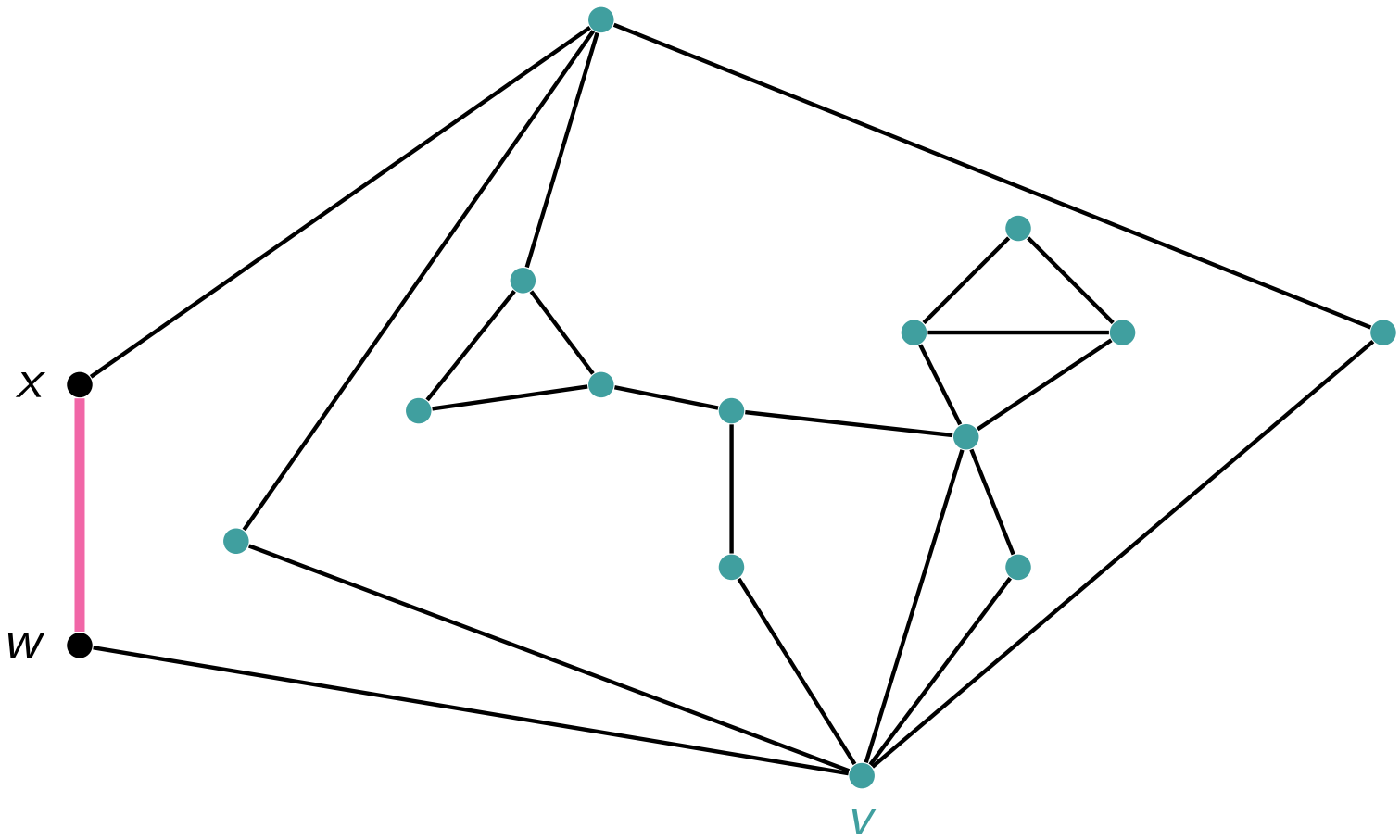


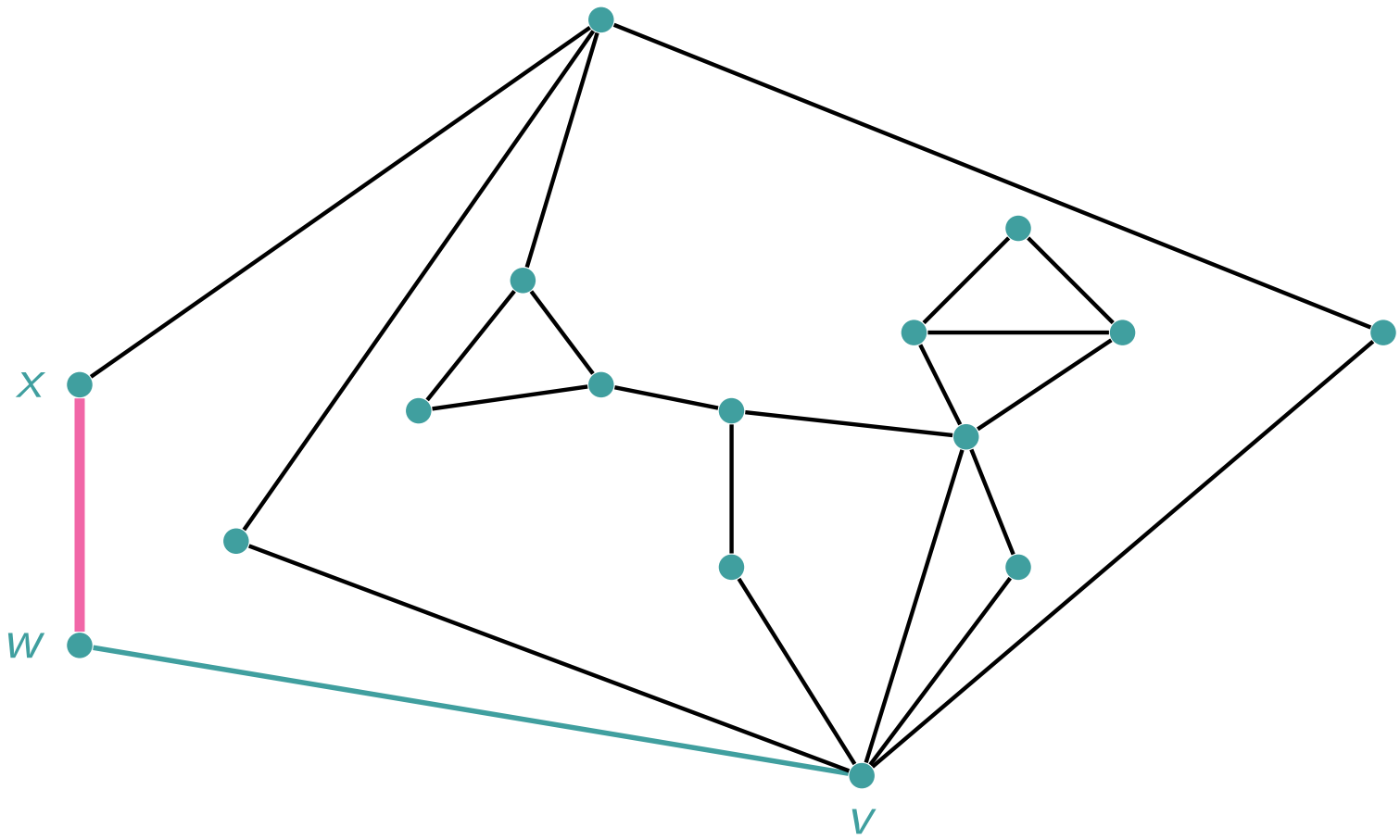


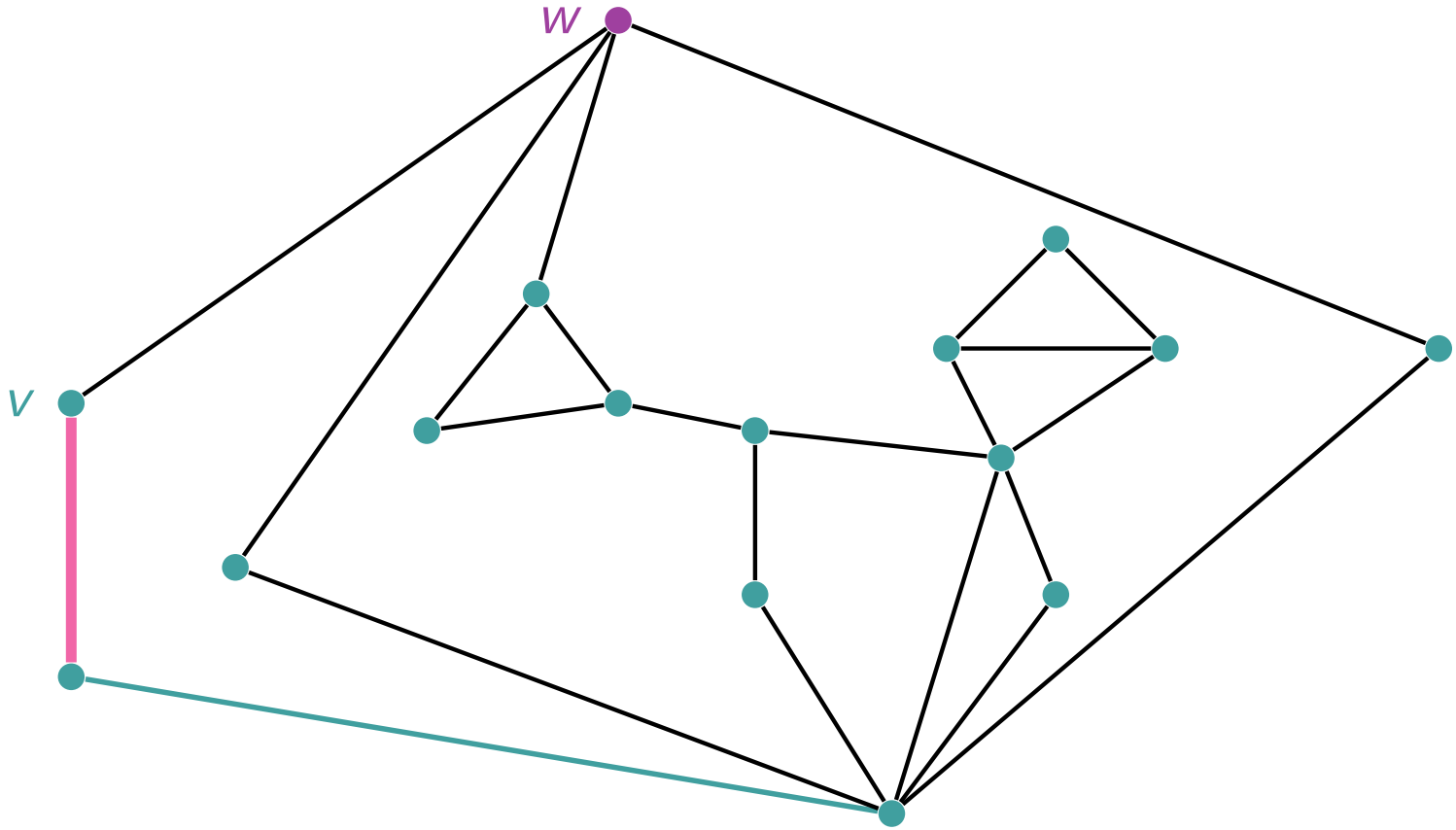

w
v

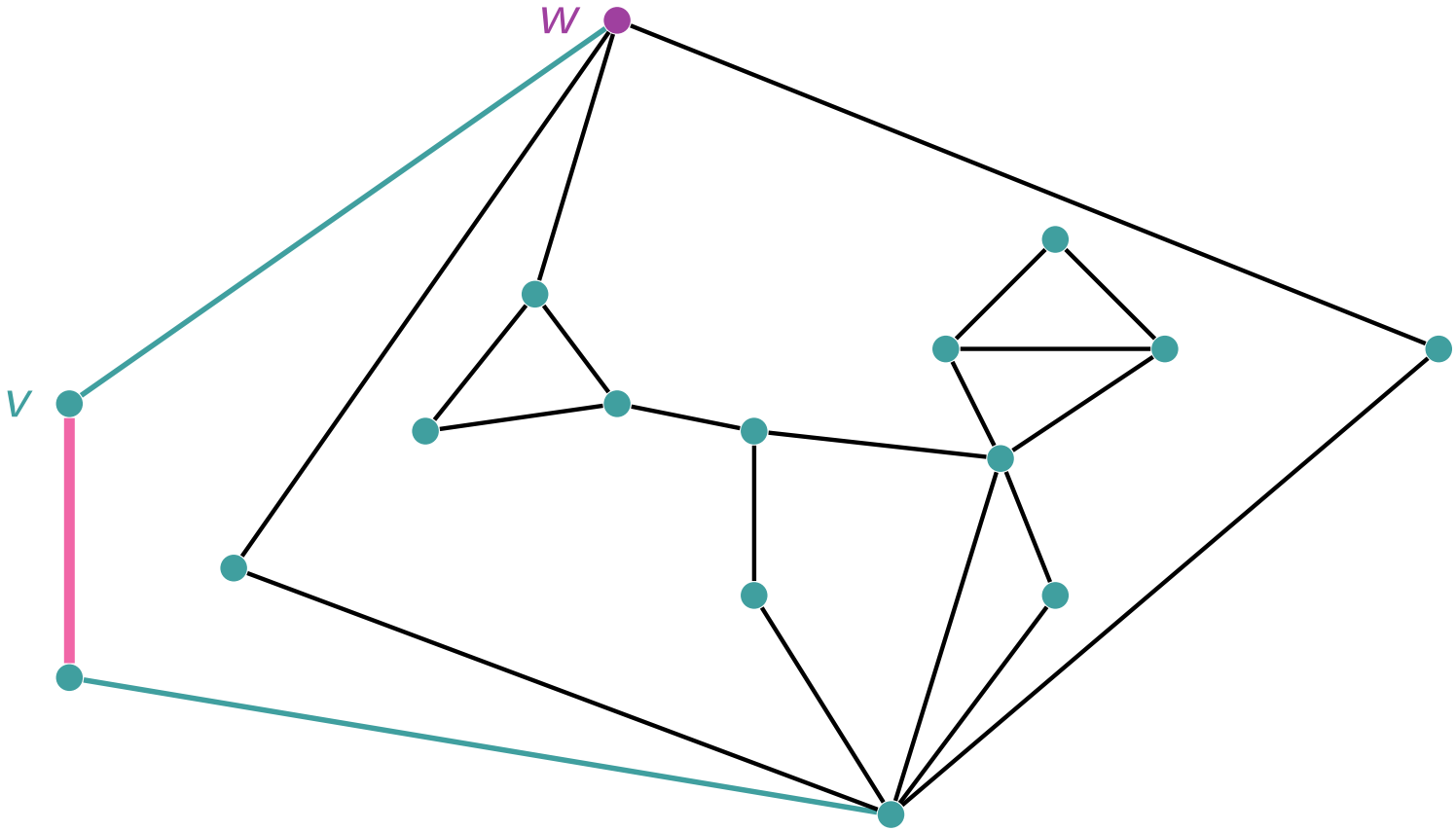

w
v

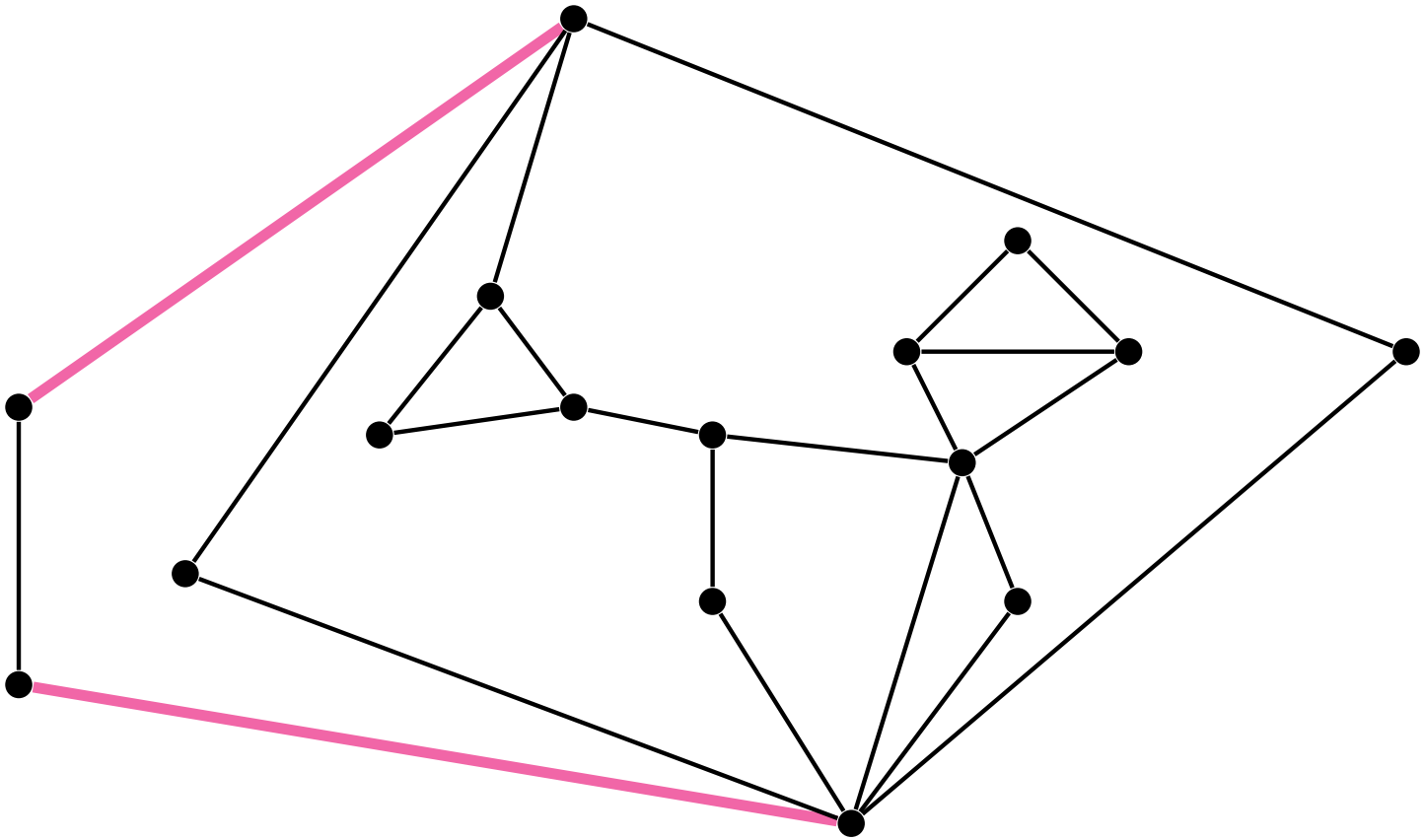


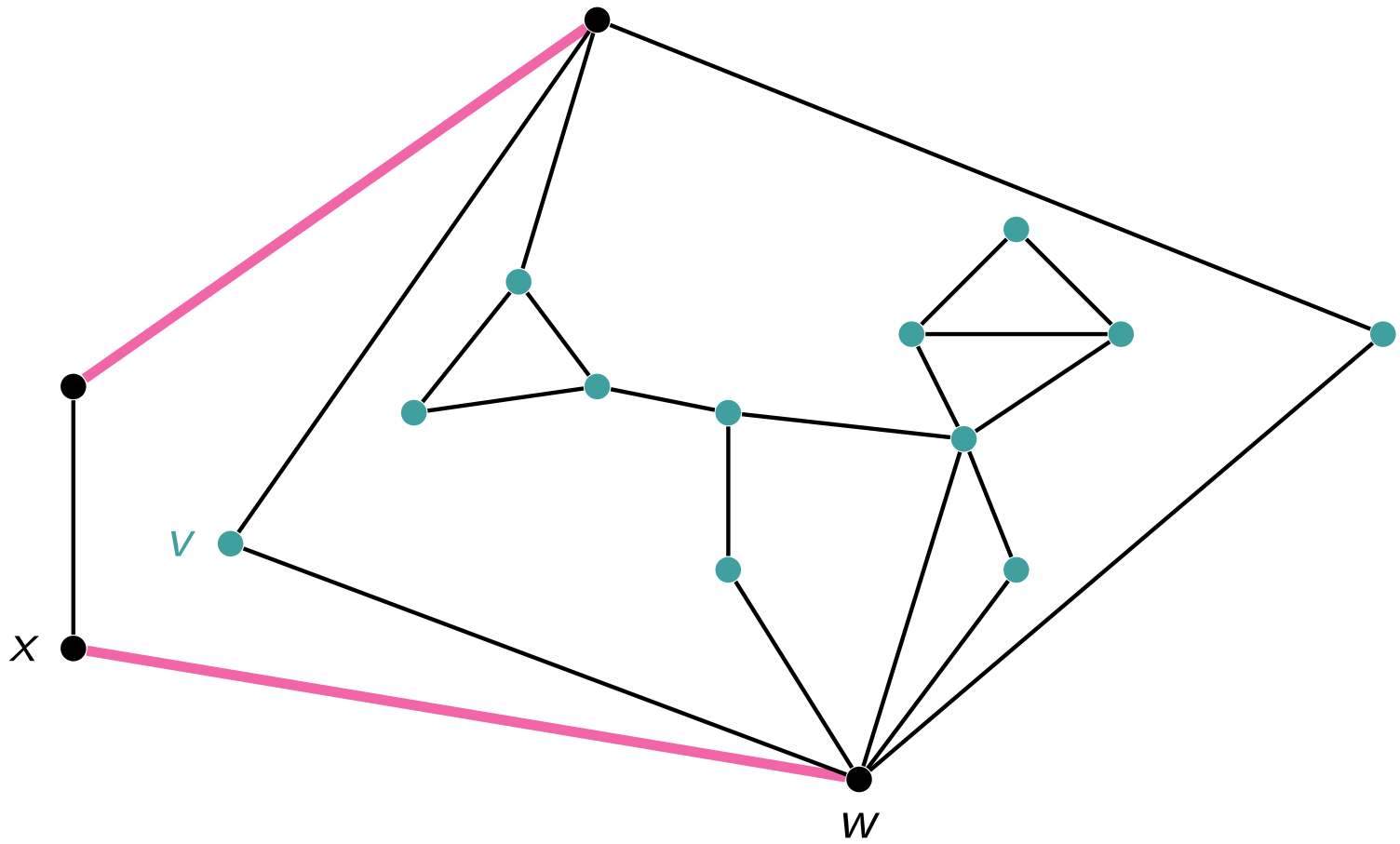

x
w
v

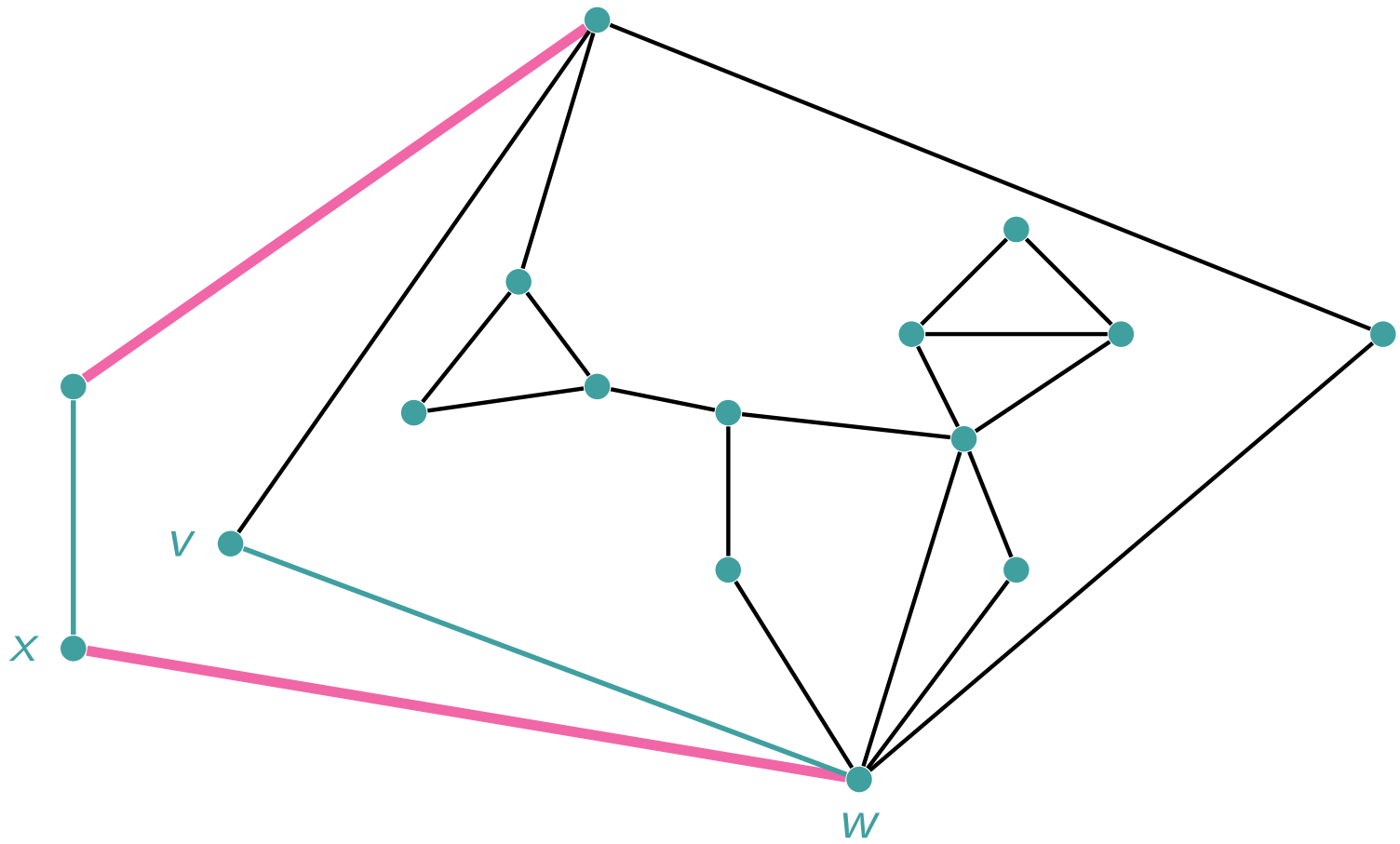

x
w
v

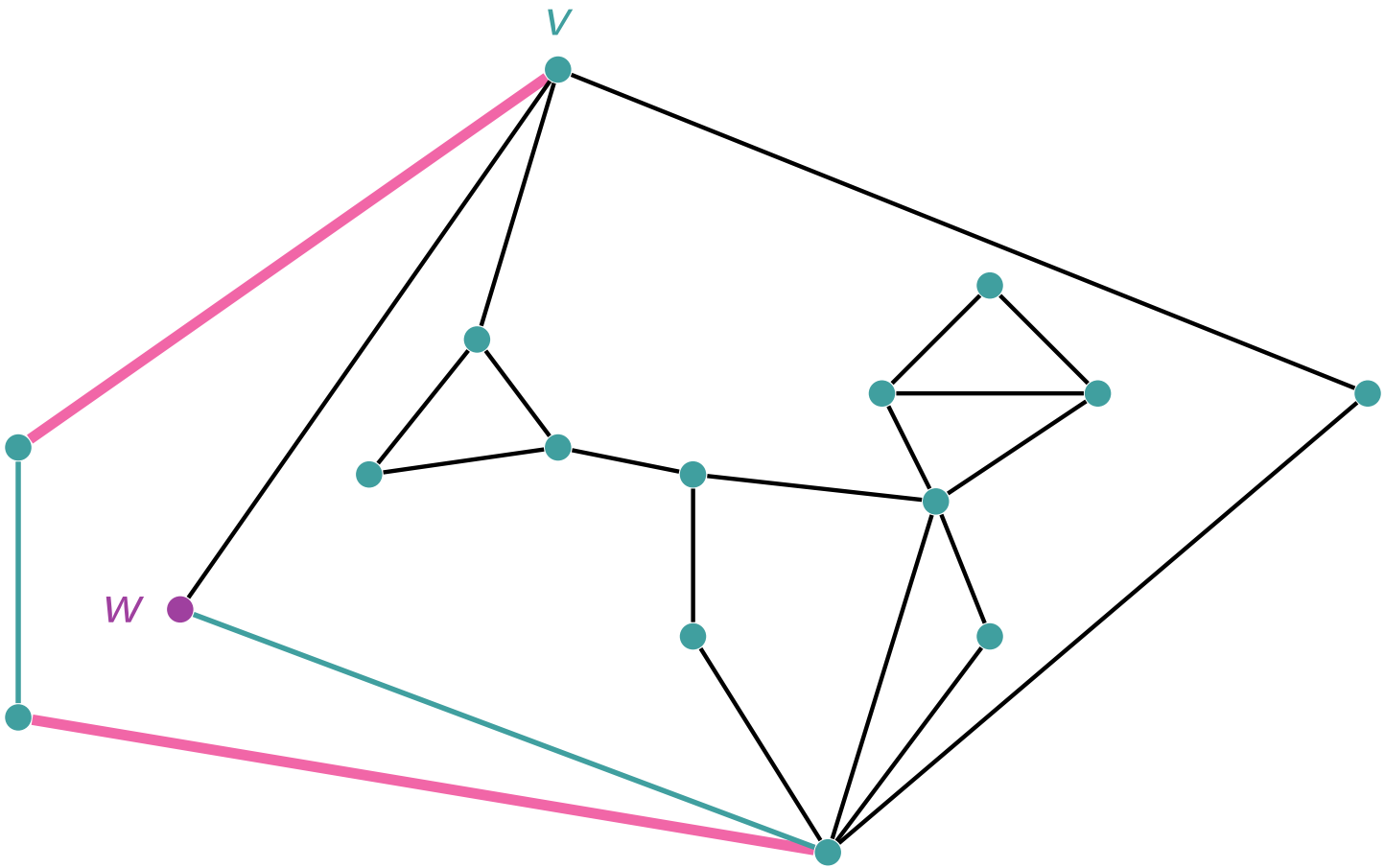

w
v

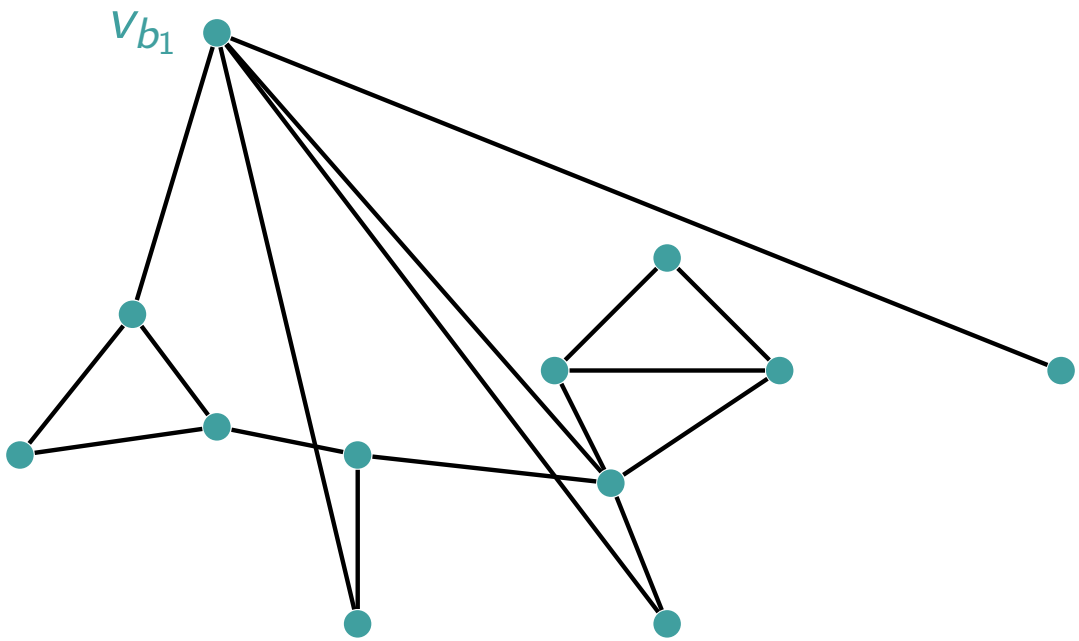

w
v

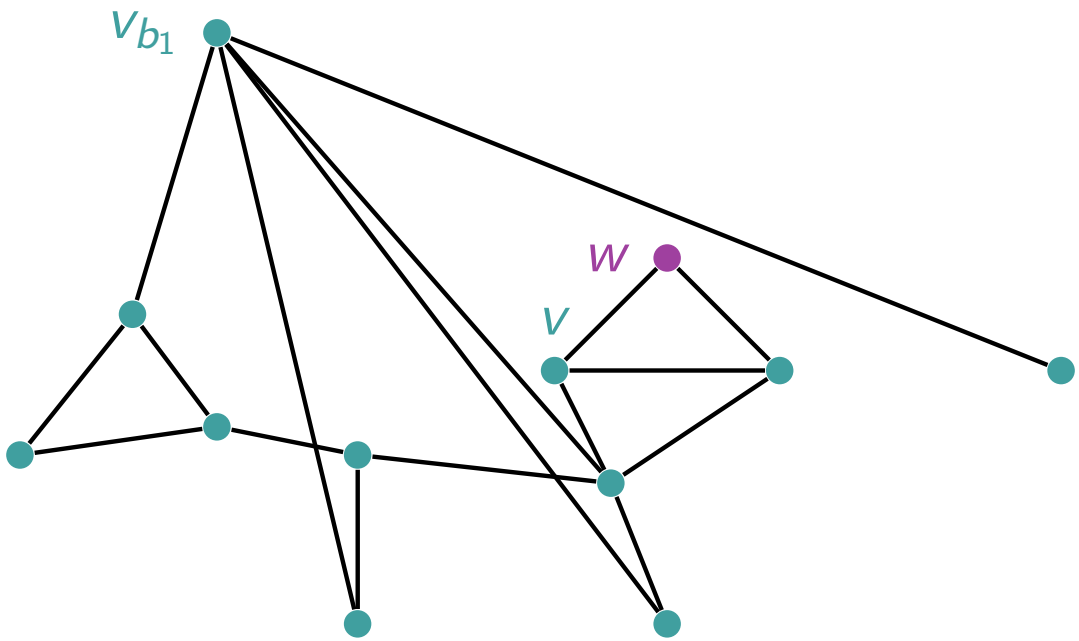


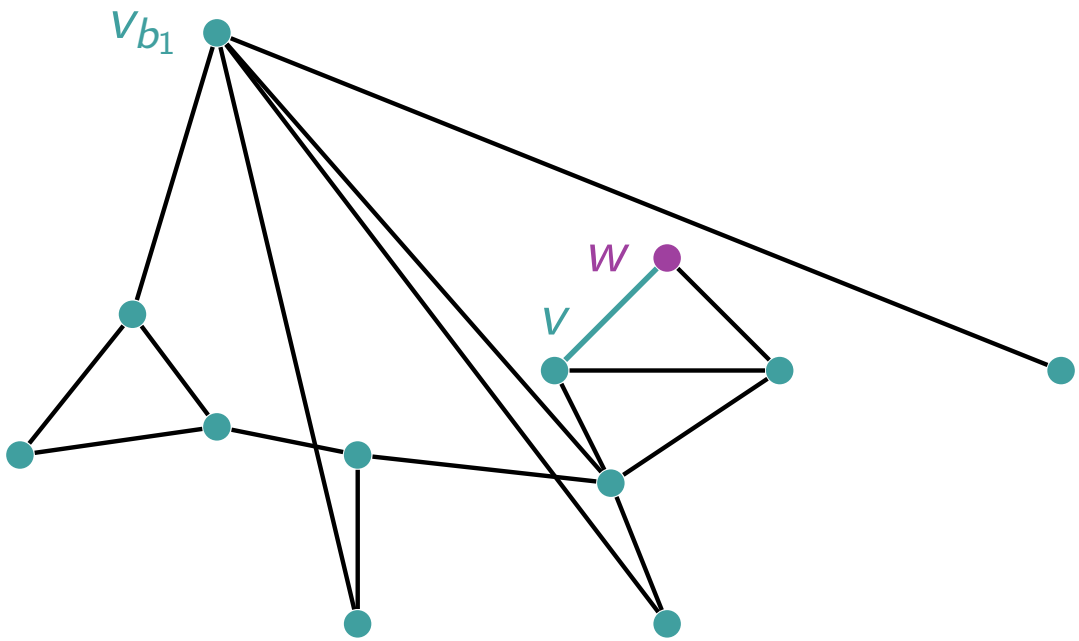

v
x
w

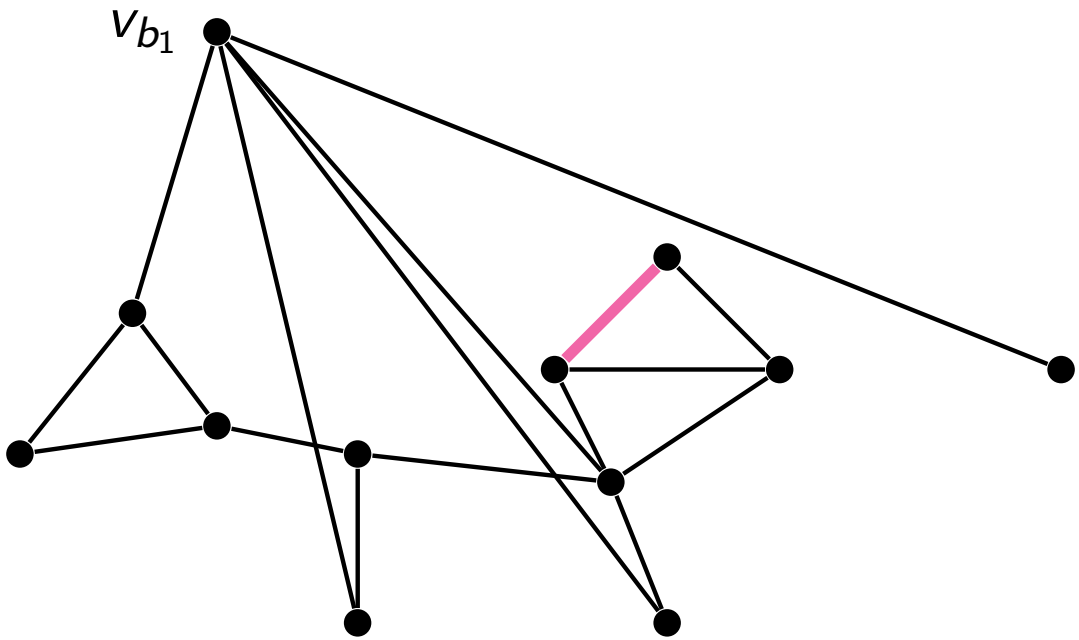

v
x
w

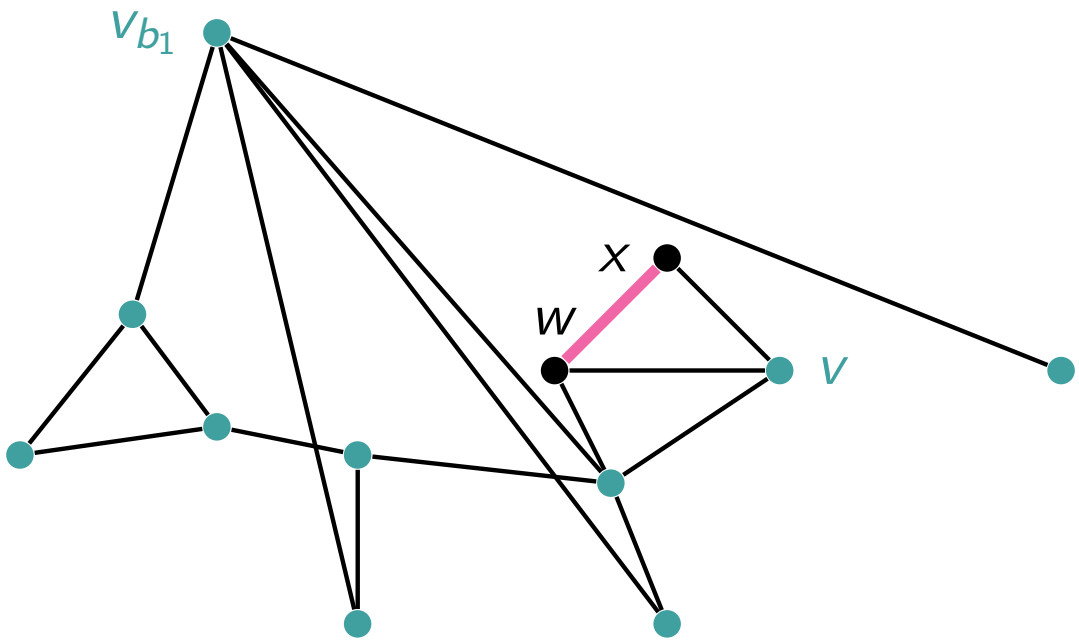

v
w

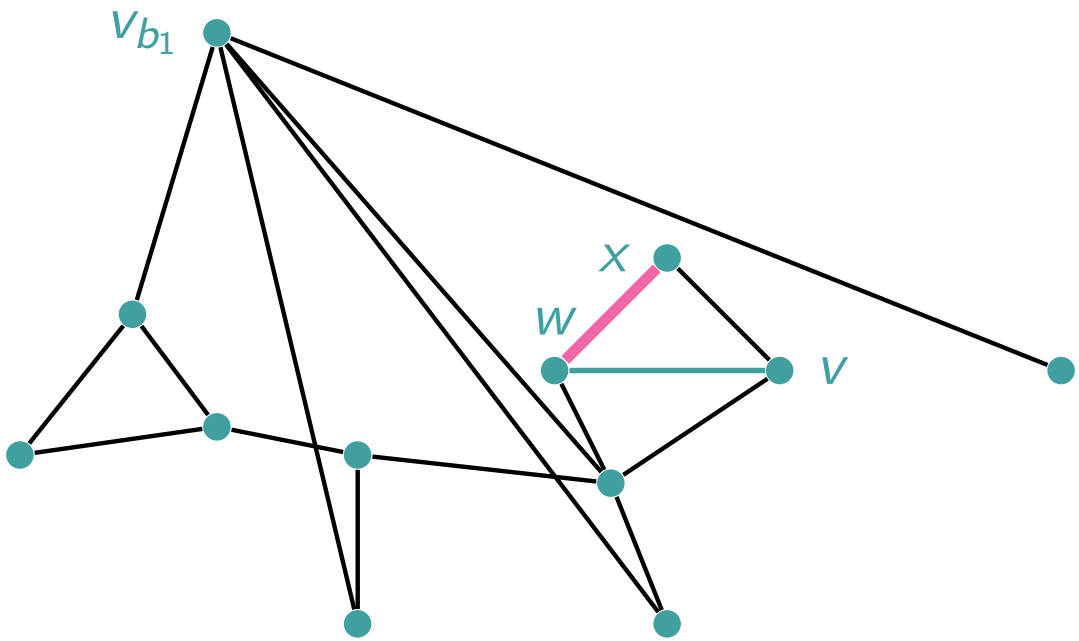

v_{b_1}

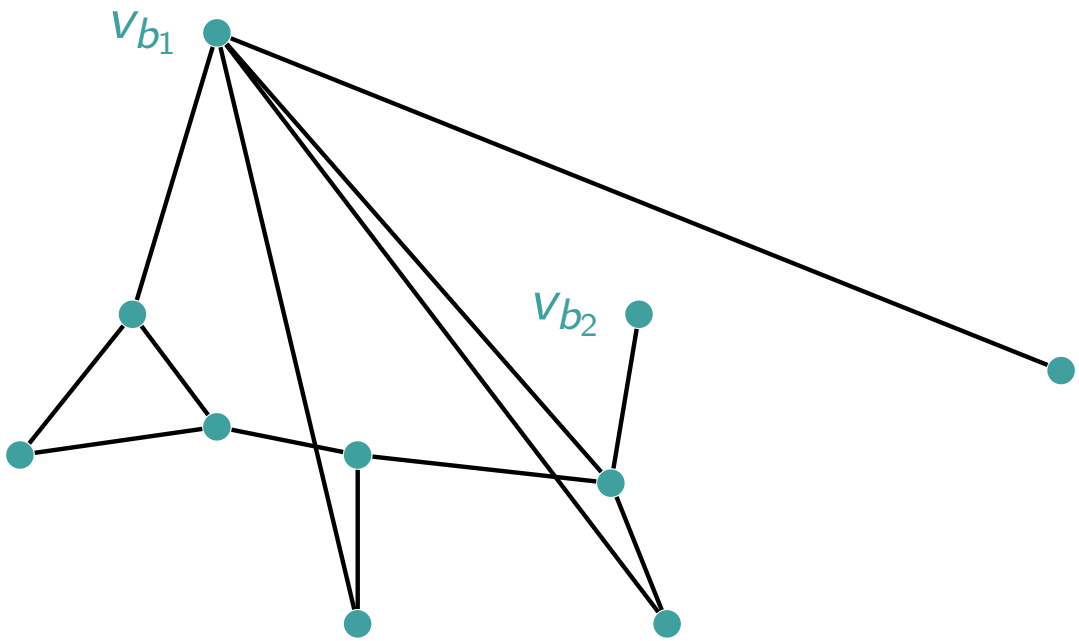

v_{b_1}
w
v

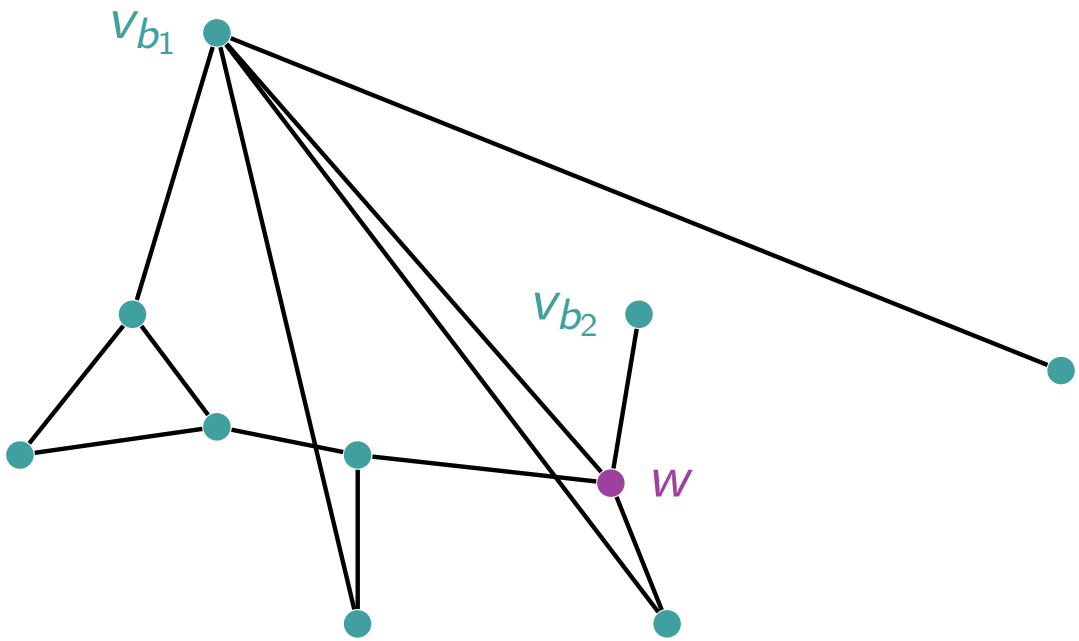

v_{b_1}
w
v

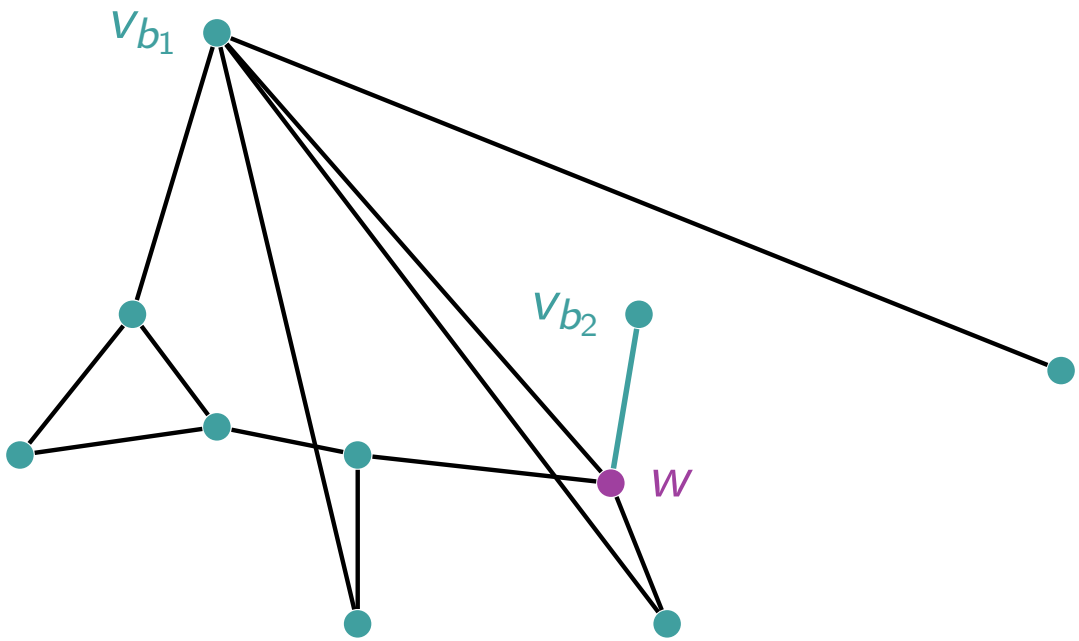

v_{b_1}

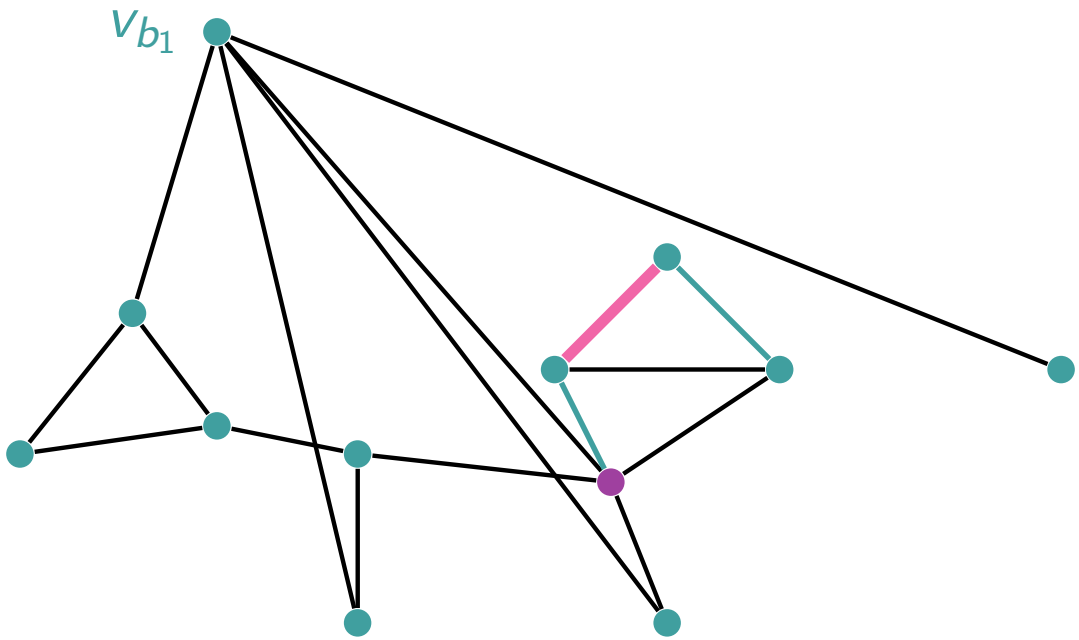

v_{b_1}
x
w
v

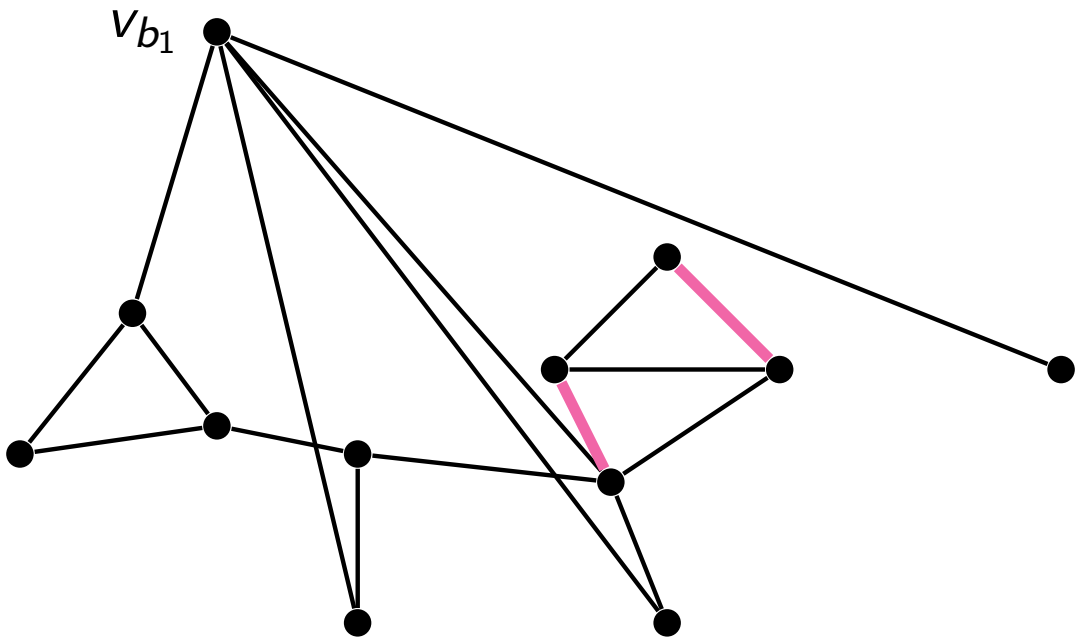

v_{b_1}
x
w
v

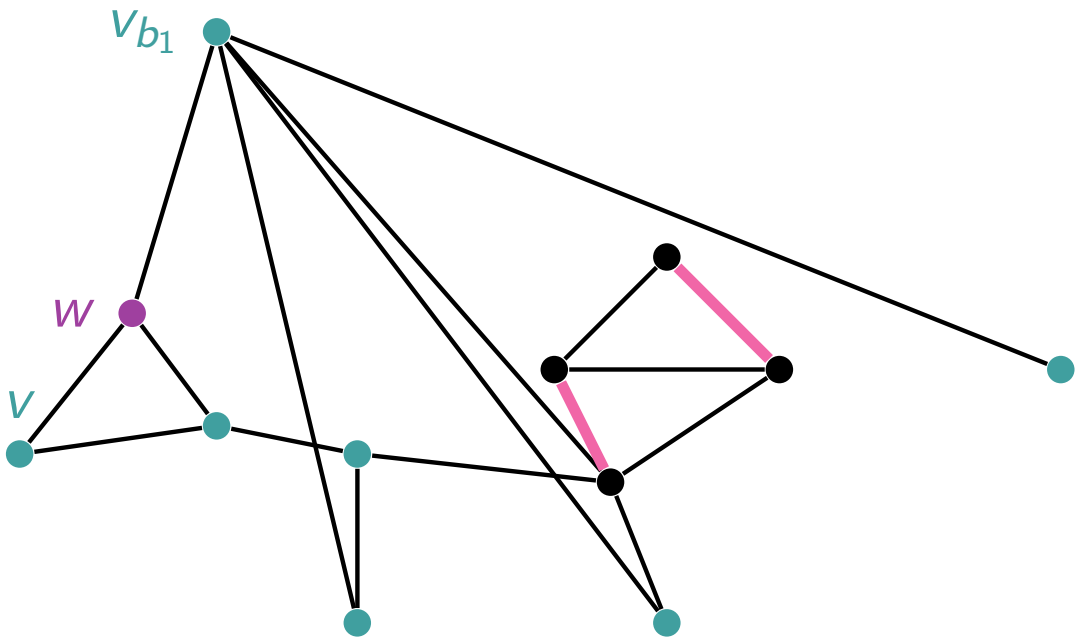

v_{b_1}
v_{b_2}

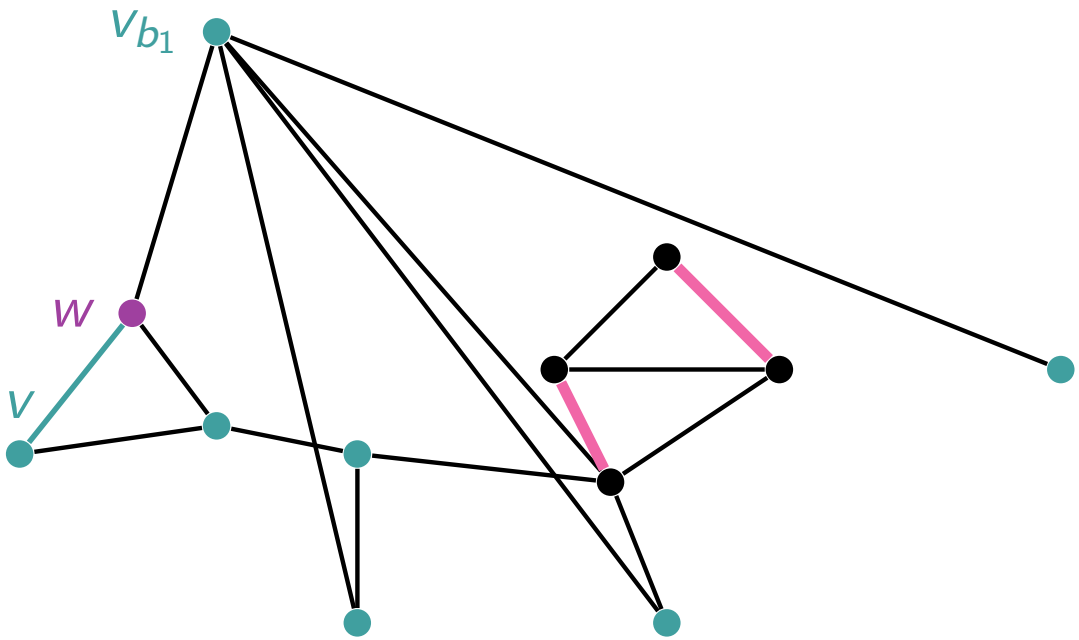

v_{b_1}
v_{b_2}
w

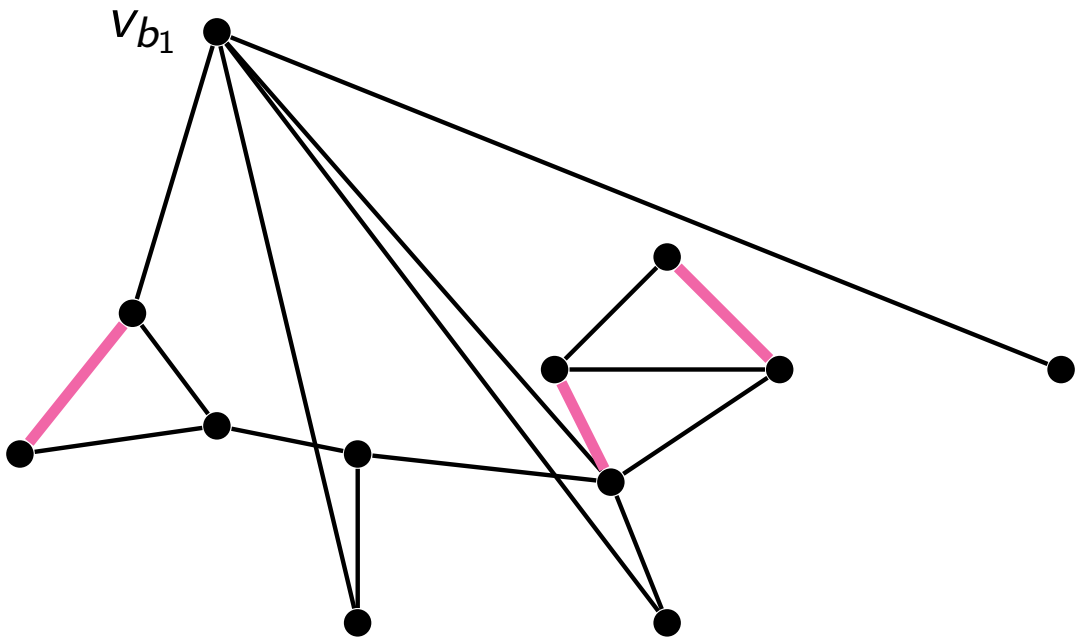

v_{b_1}
v_{b_2}
w

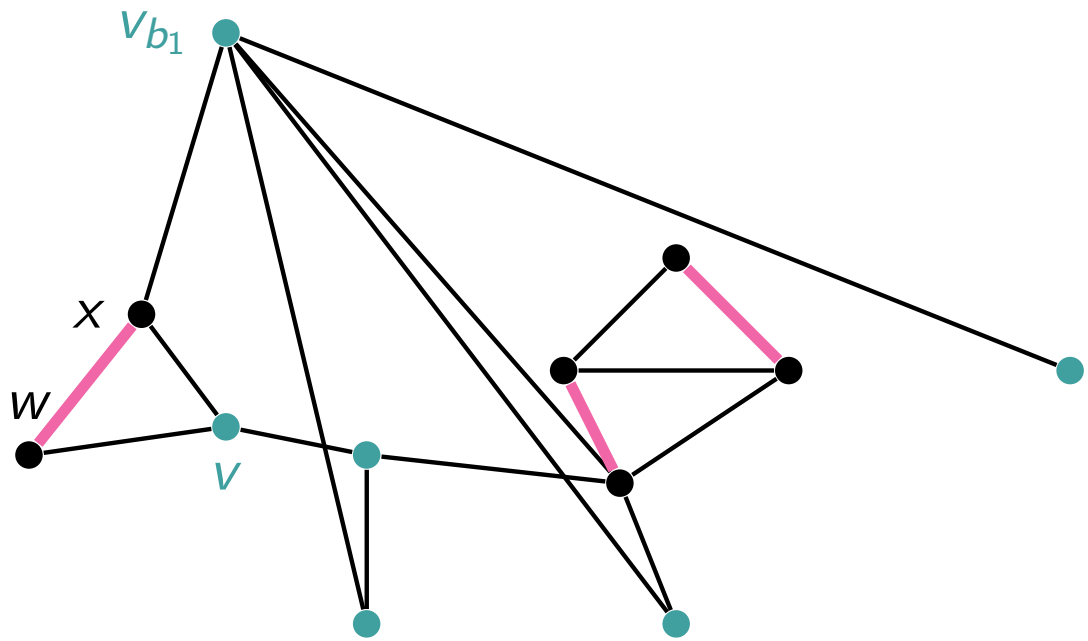

v_{b_1}

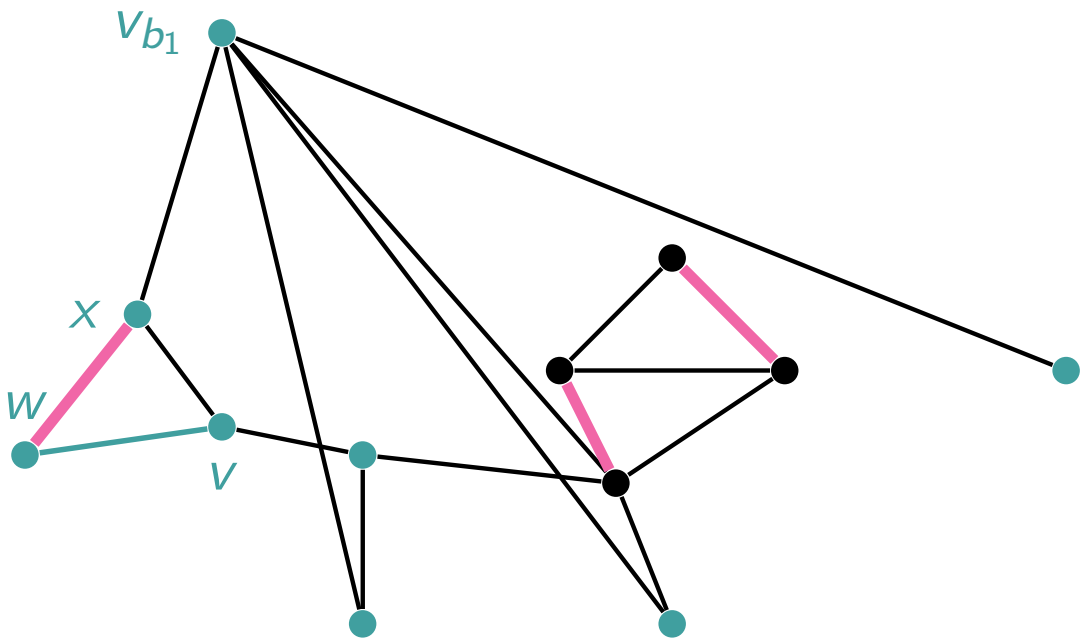

v_{b_1}

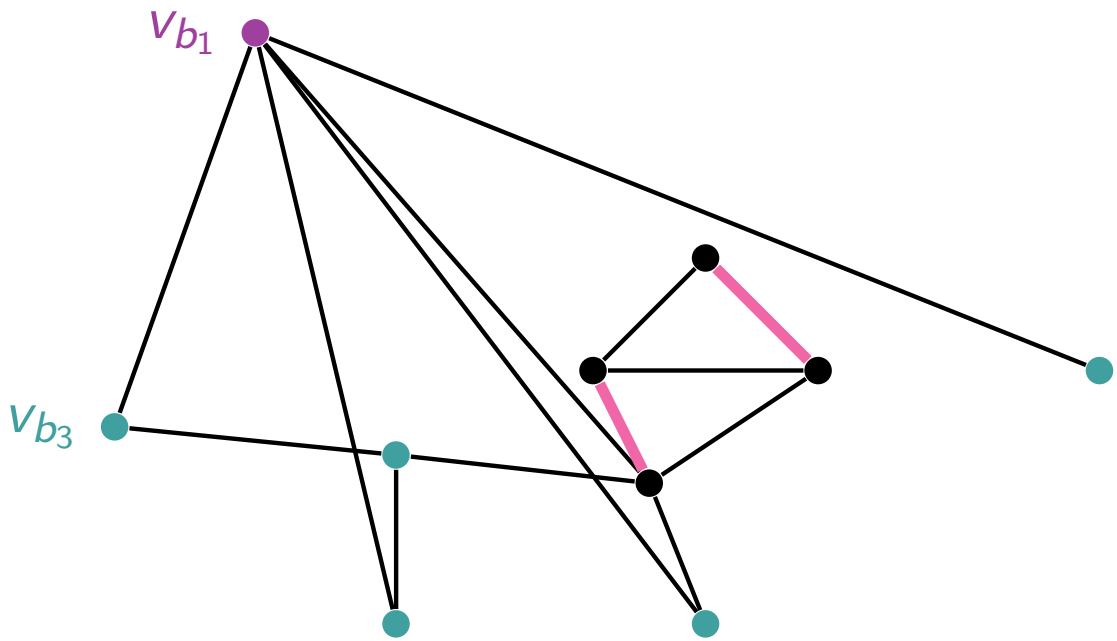

v_{b_1}
w
v

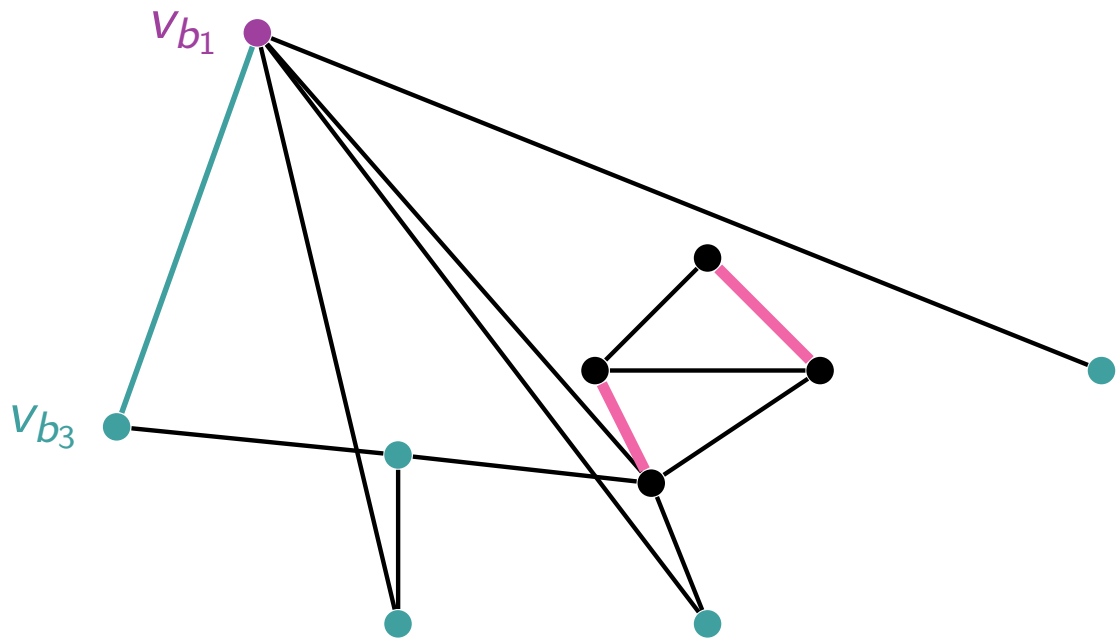

v_{b_1}
w
v

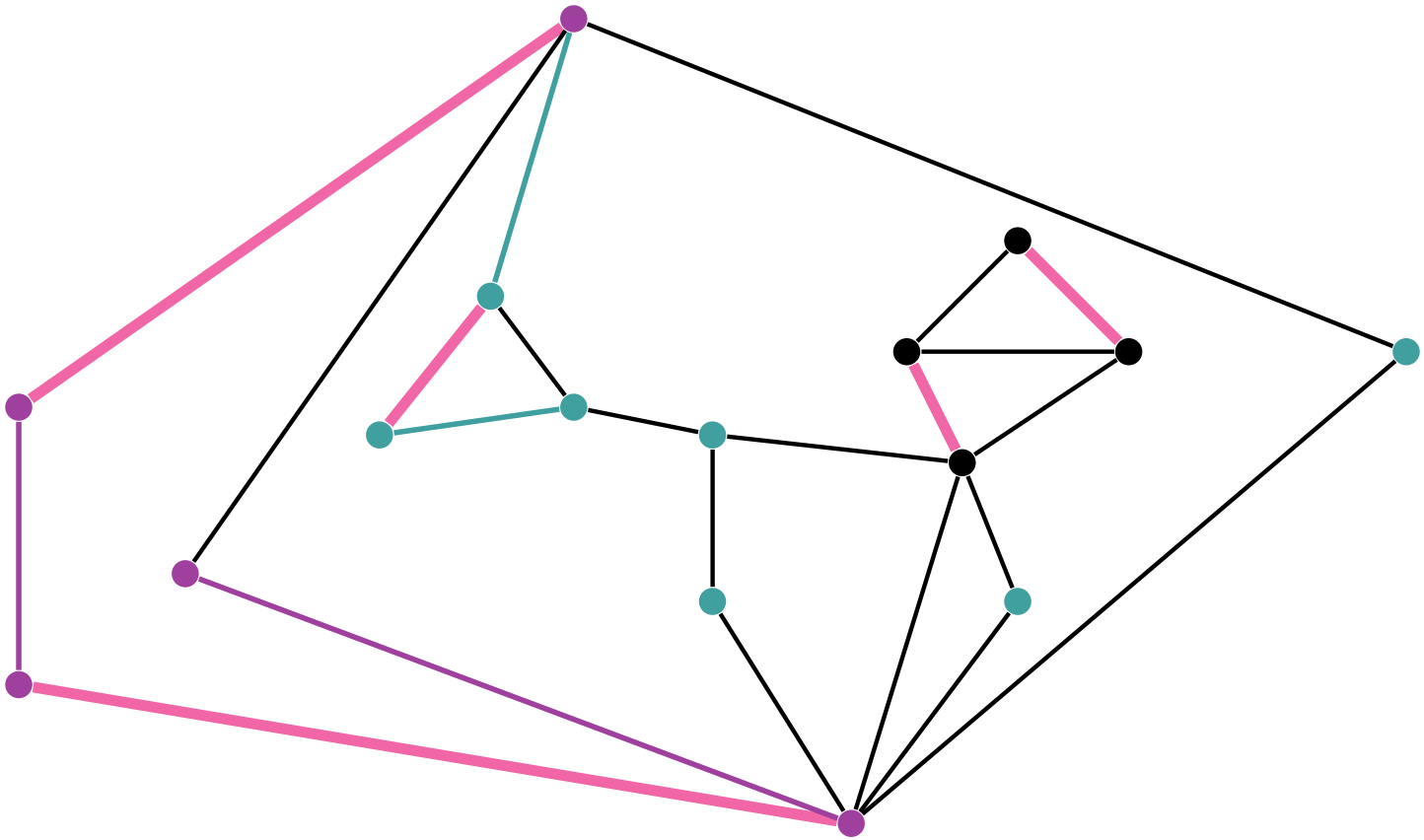

v_{b_1}

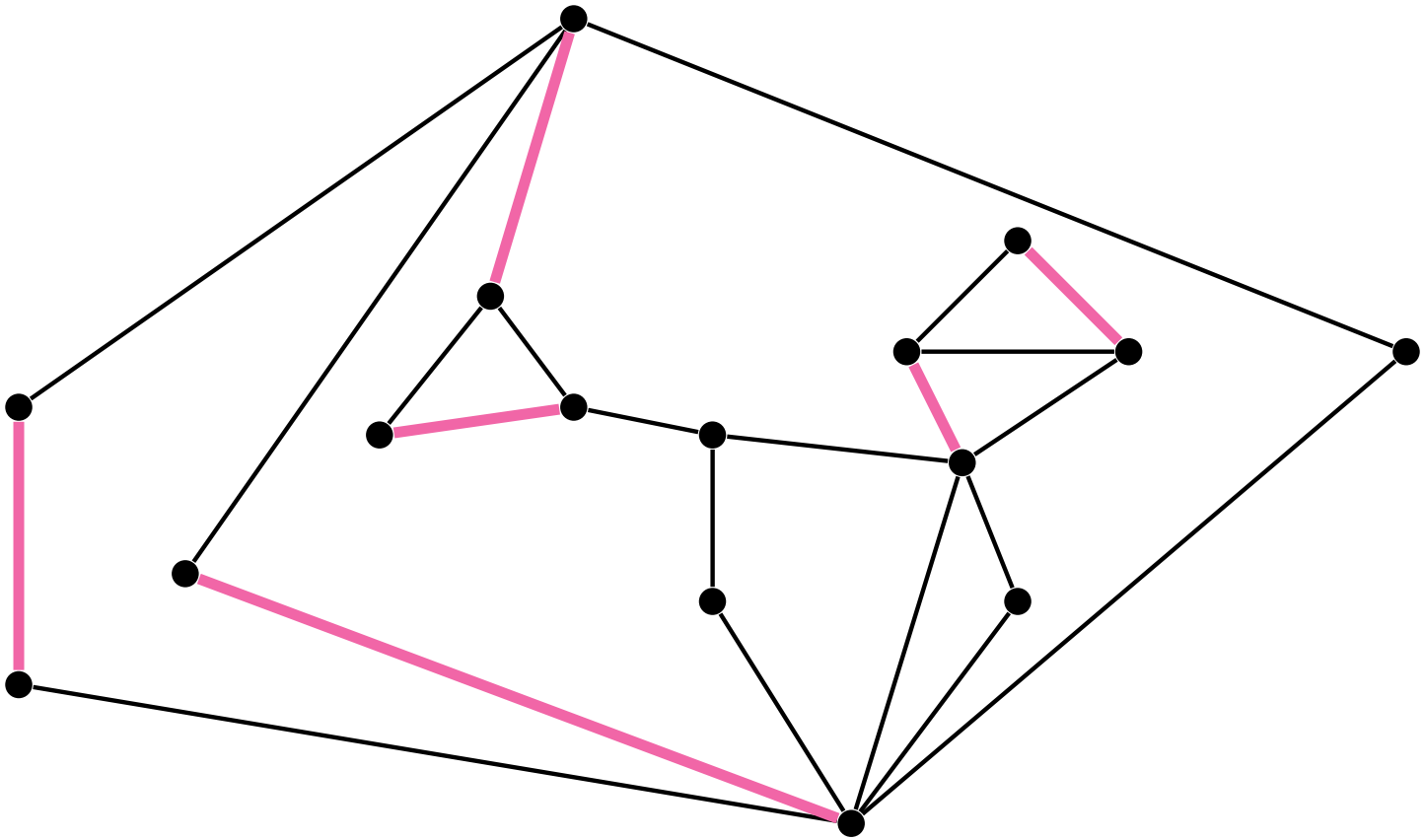

v_{b_1}
x
w
v

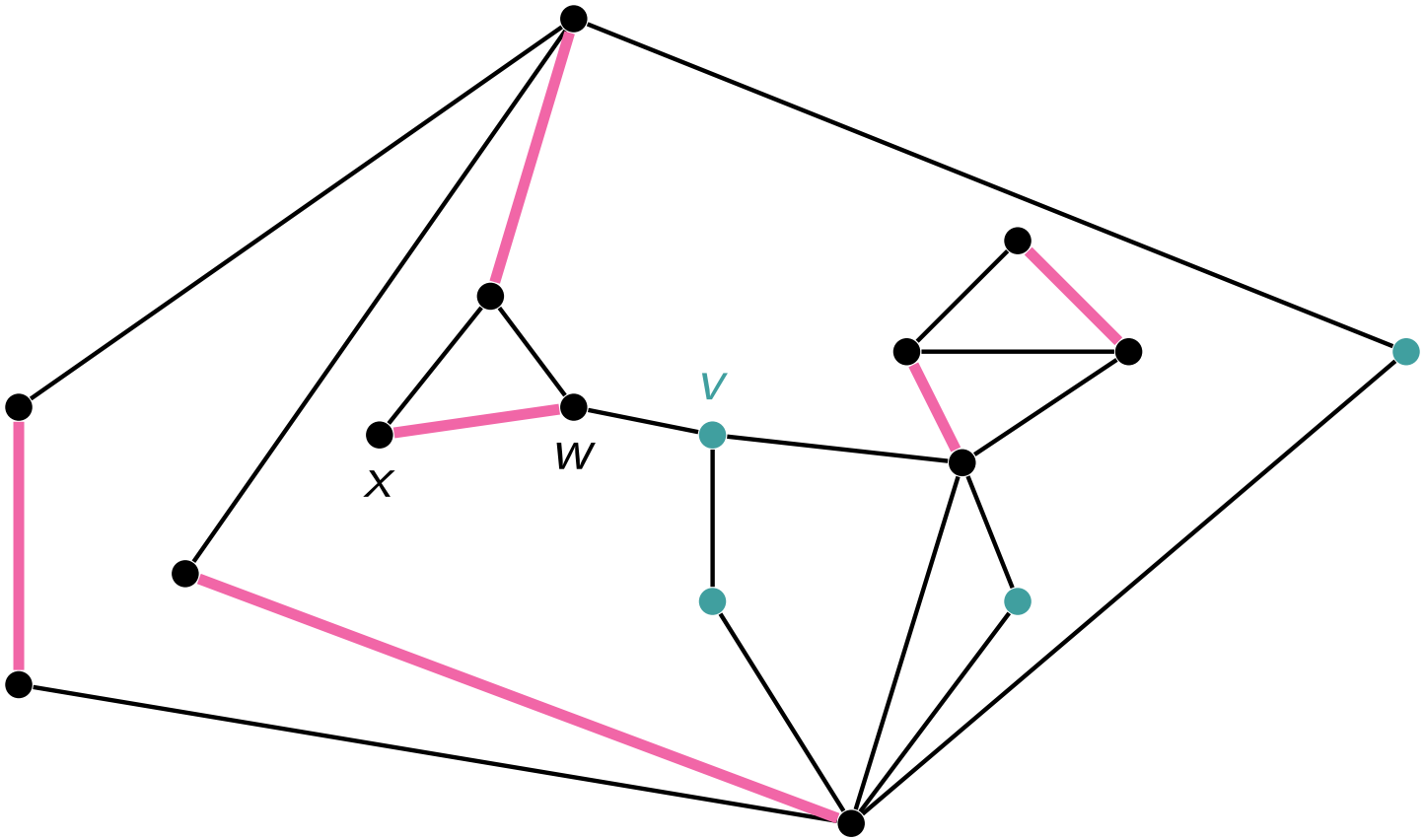

v_{b_1}
x
w
v

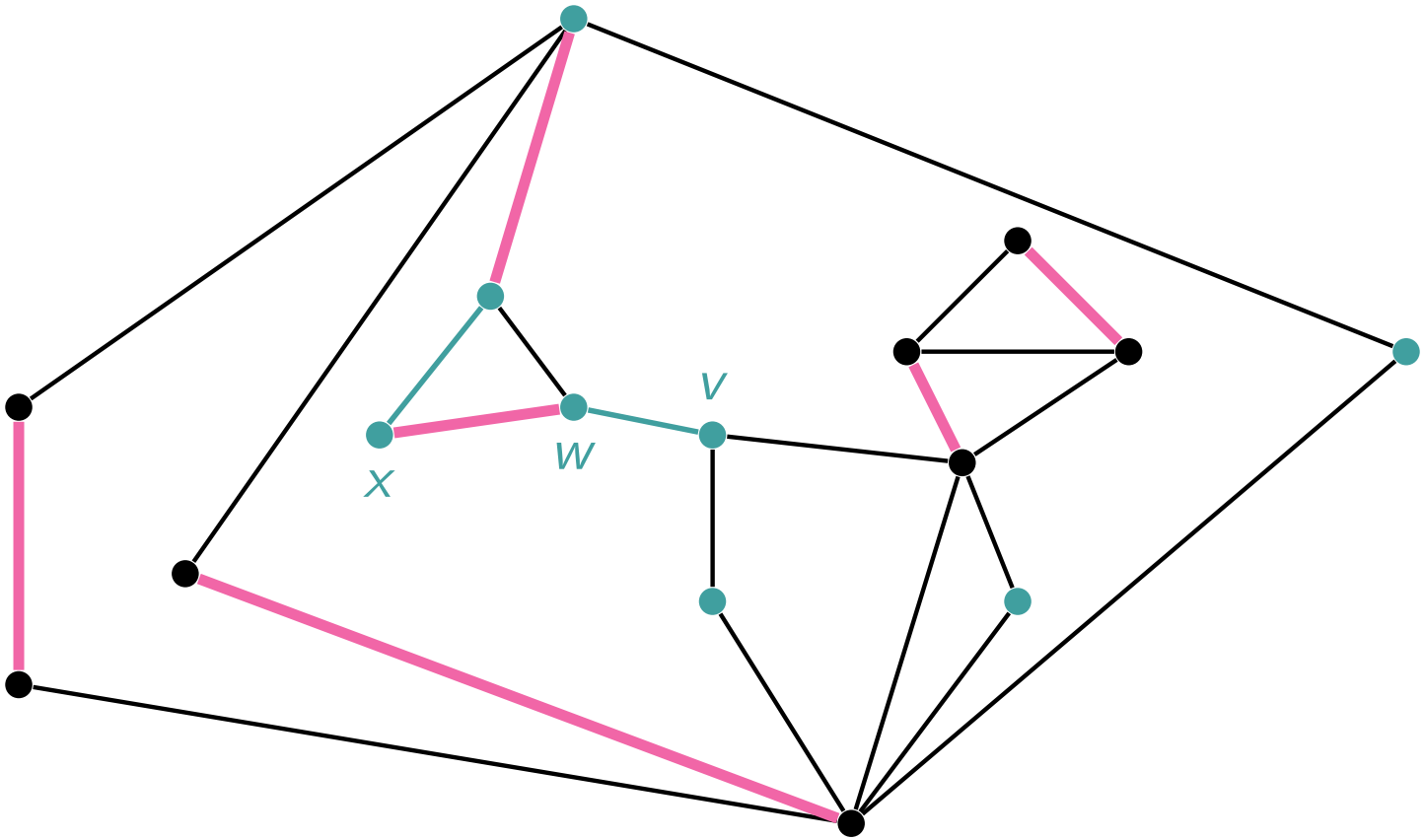

v_{b_1}
v_{b_3}

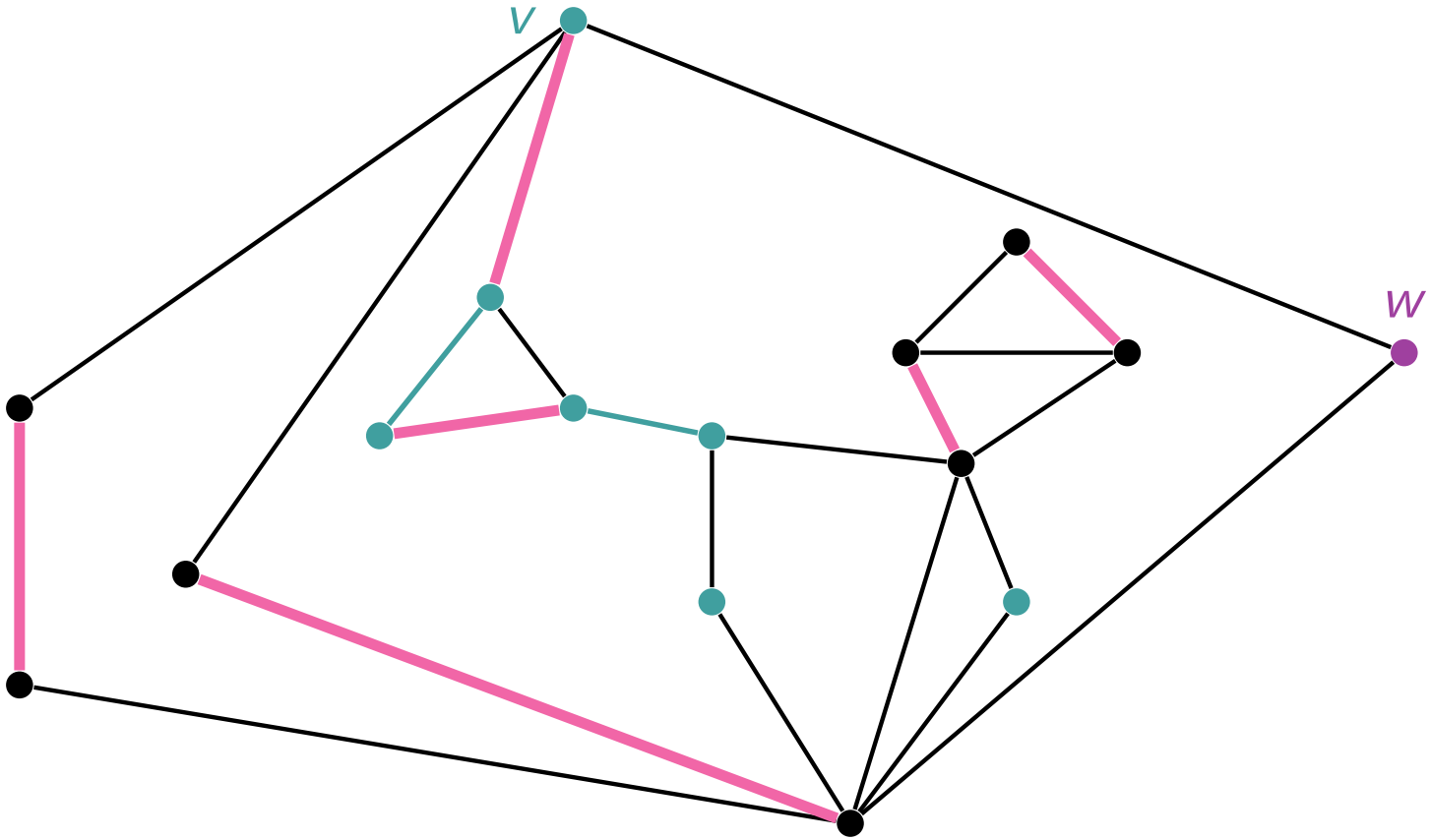

v_{b_1}
v_{b_3}

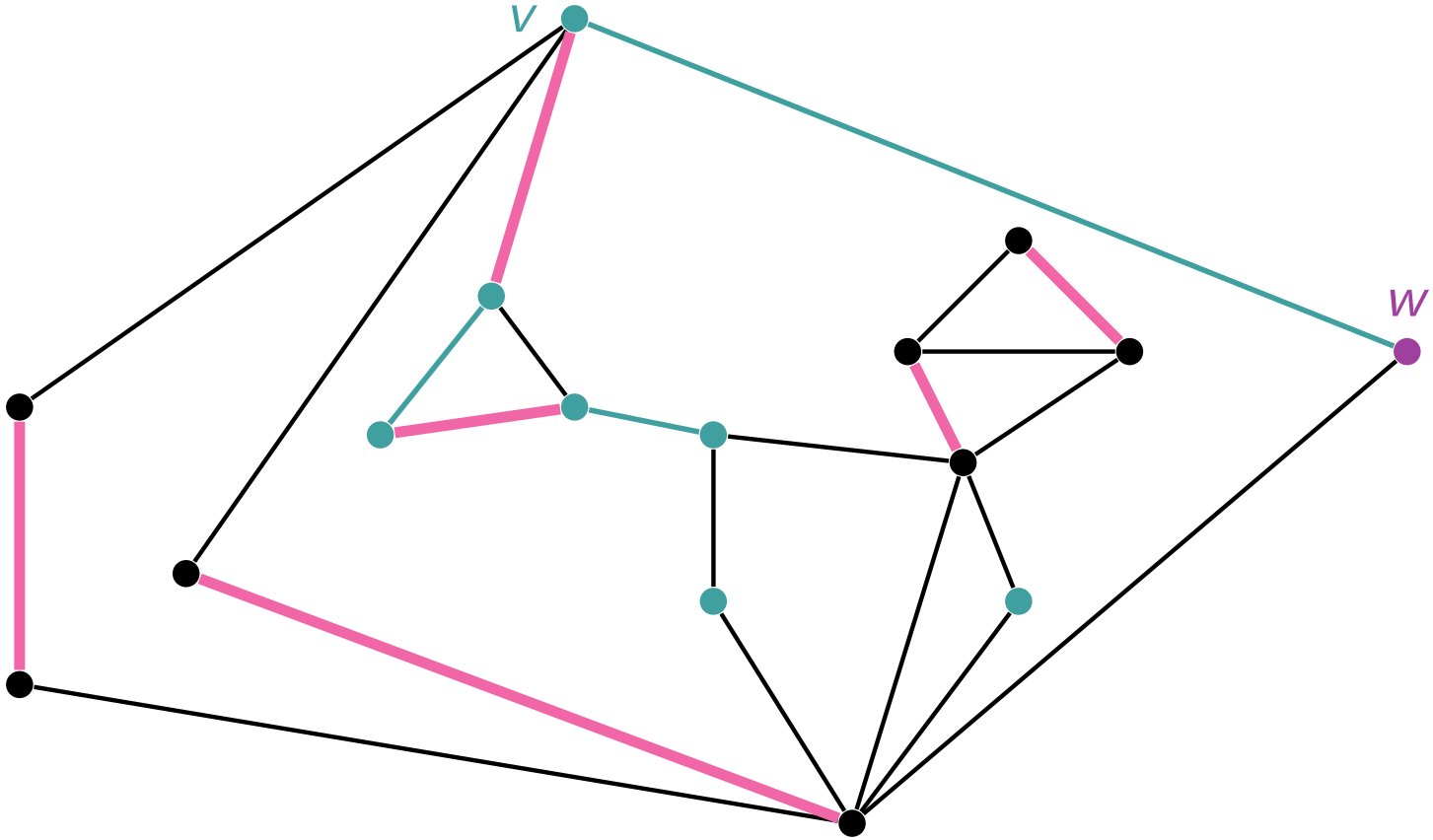


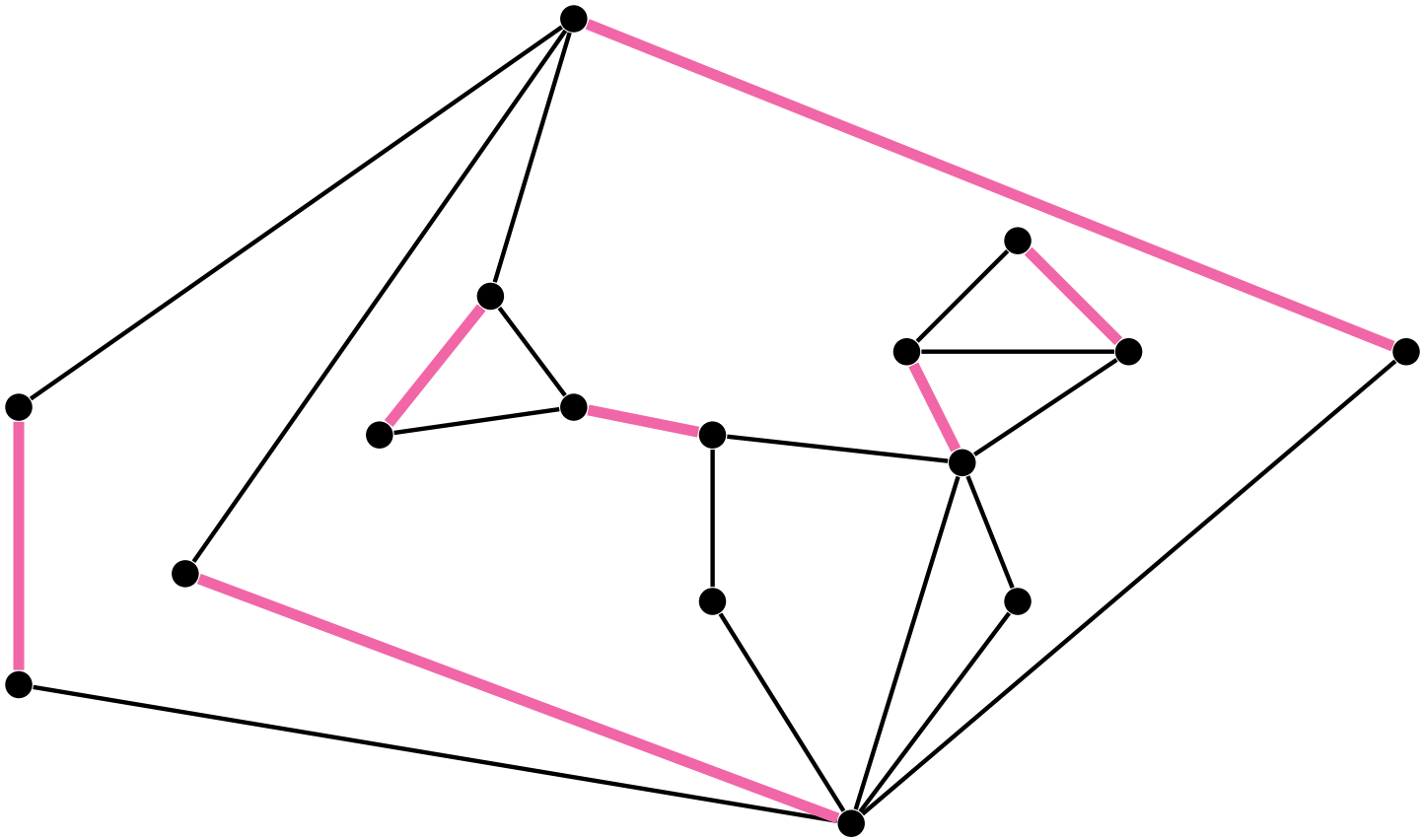


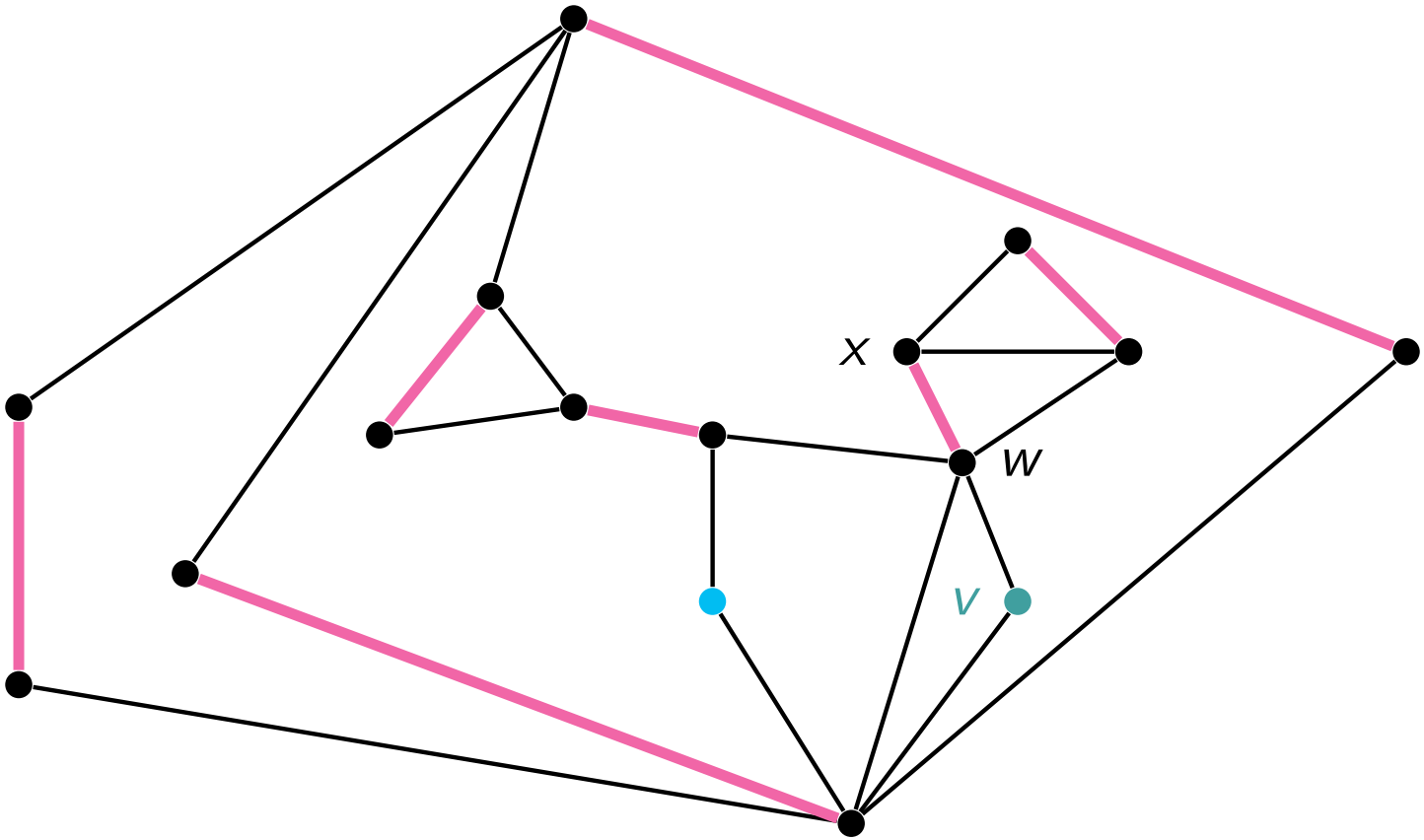

v
w
x

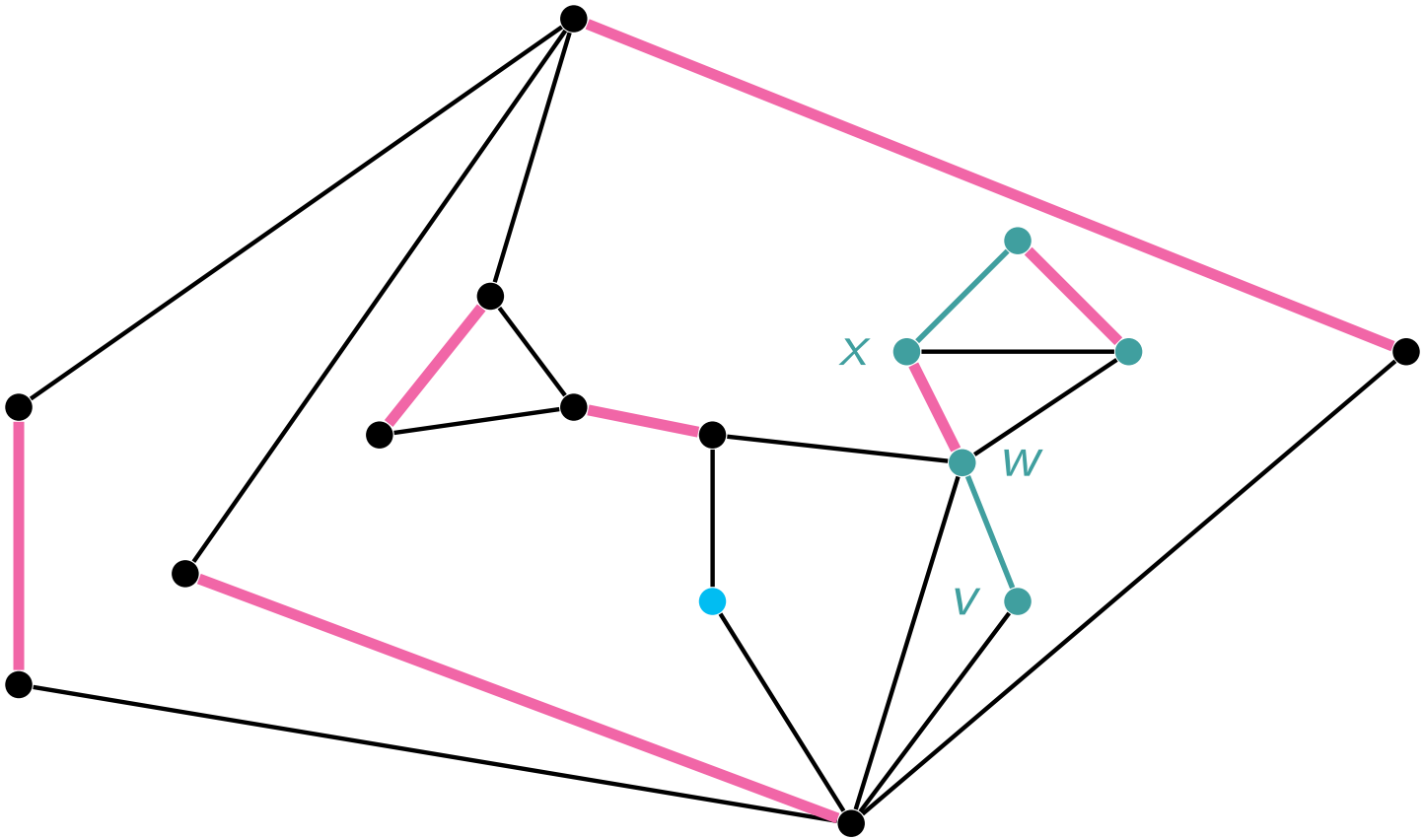

v
w
x

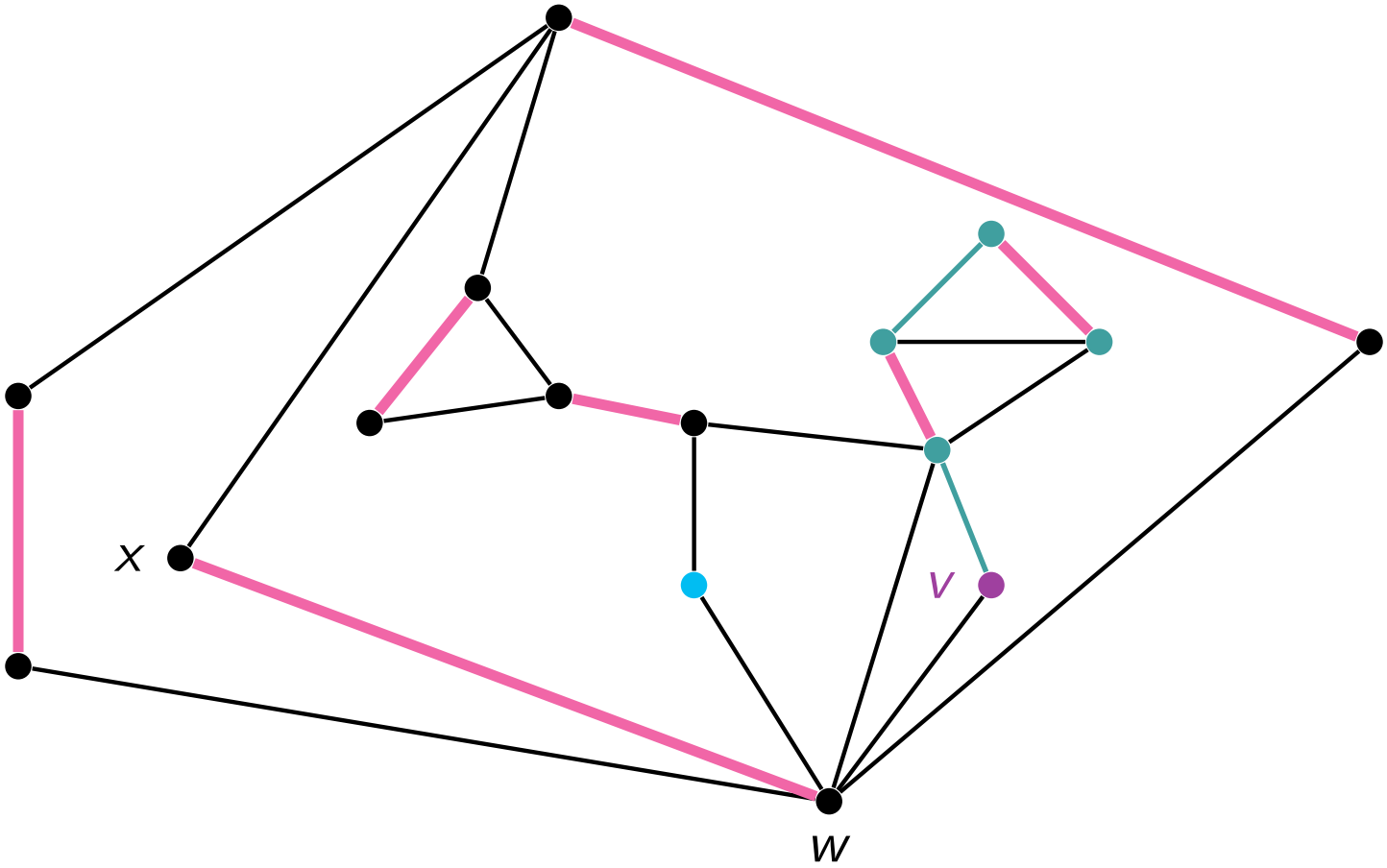

v
w

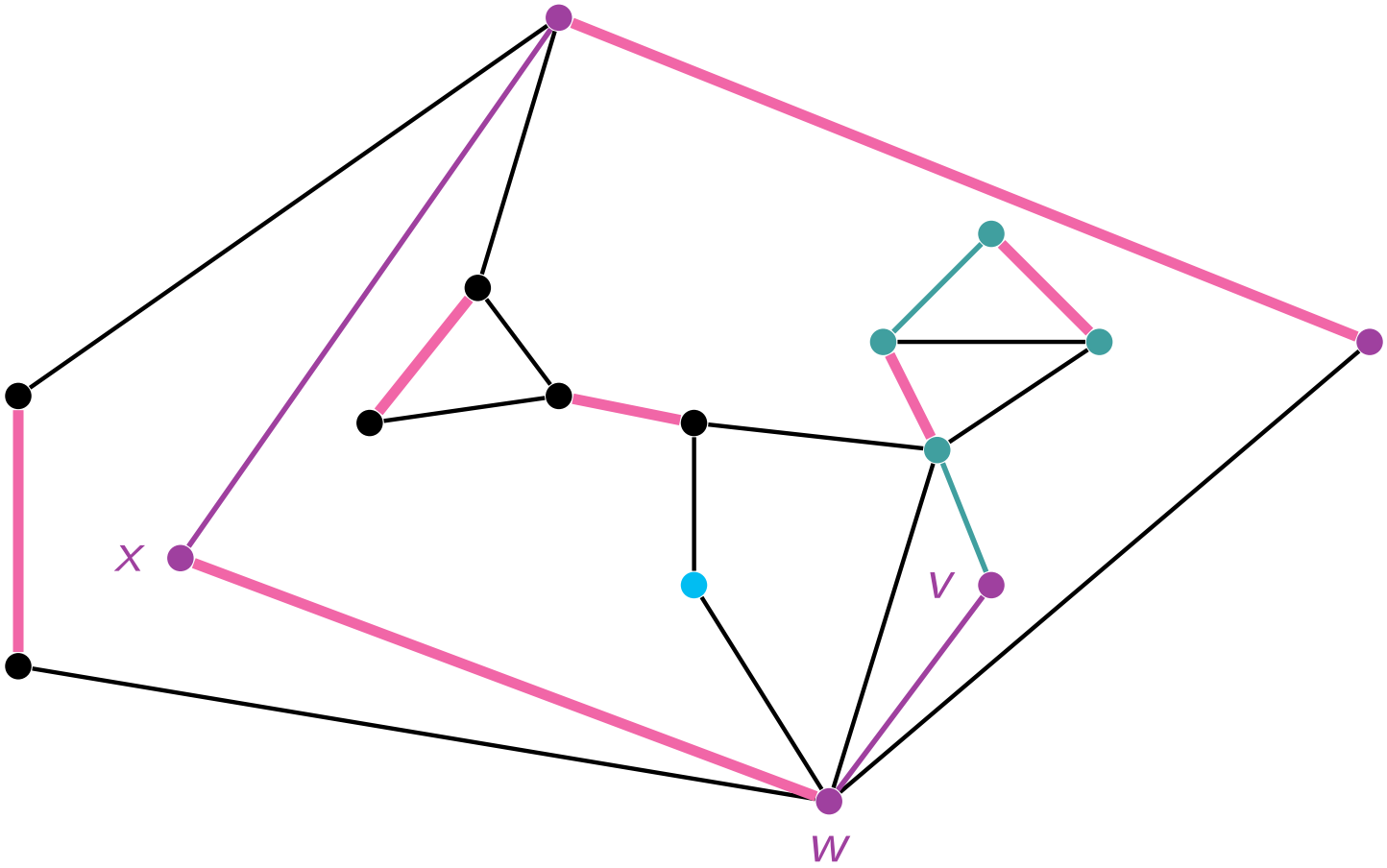

v
w

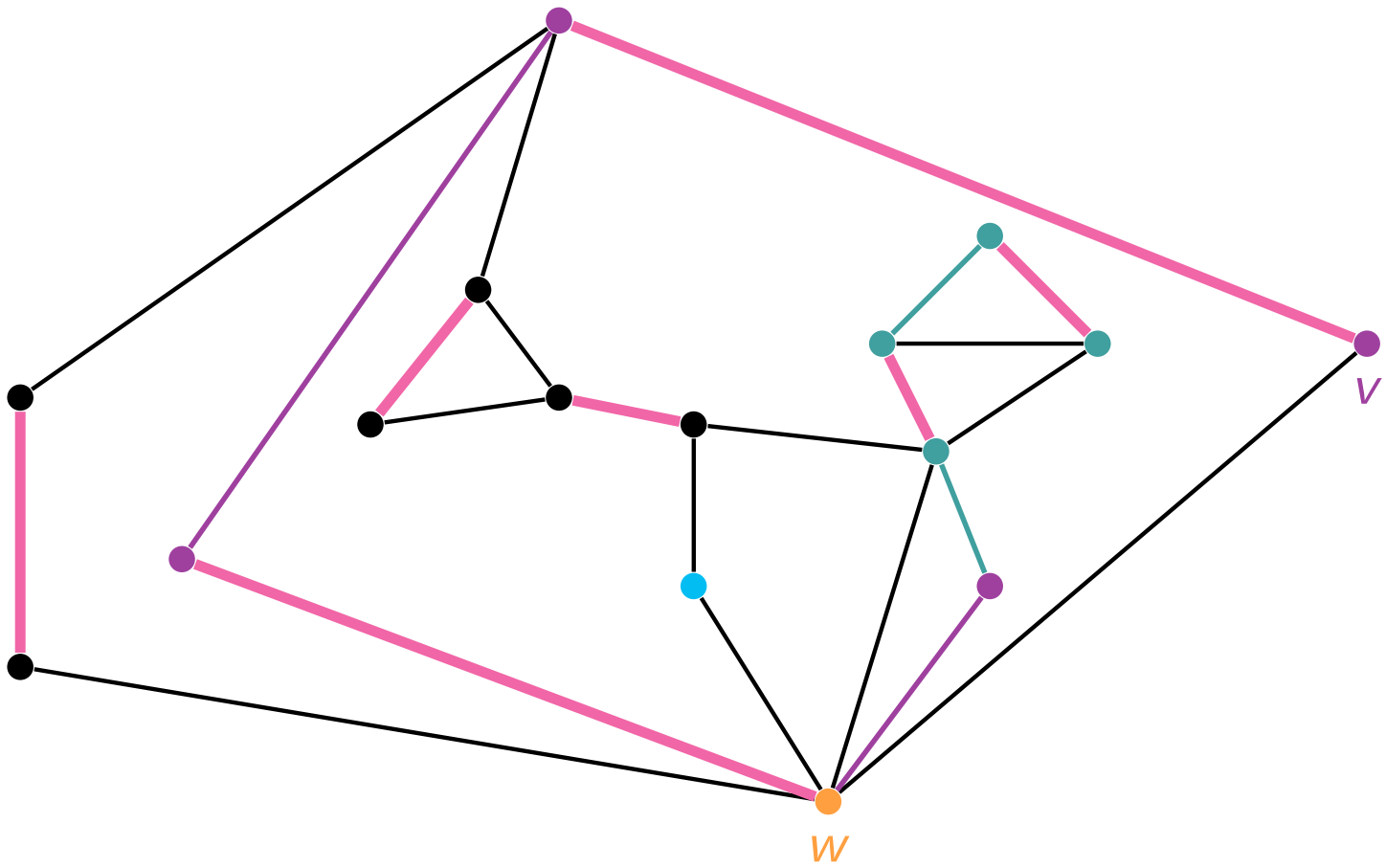


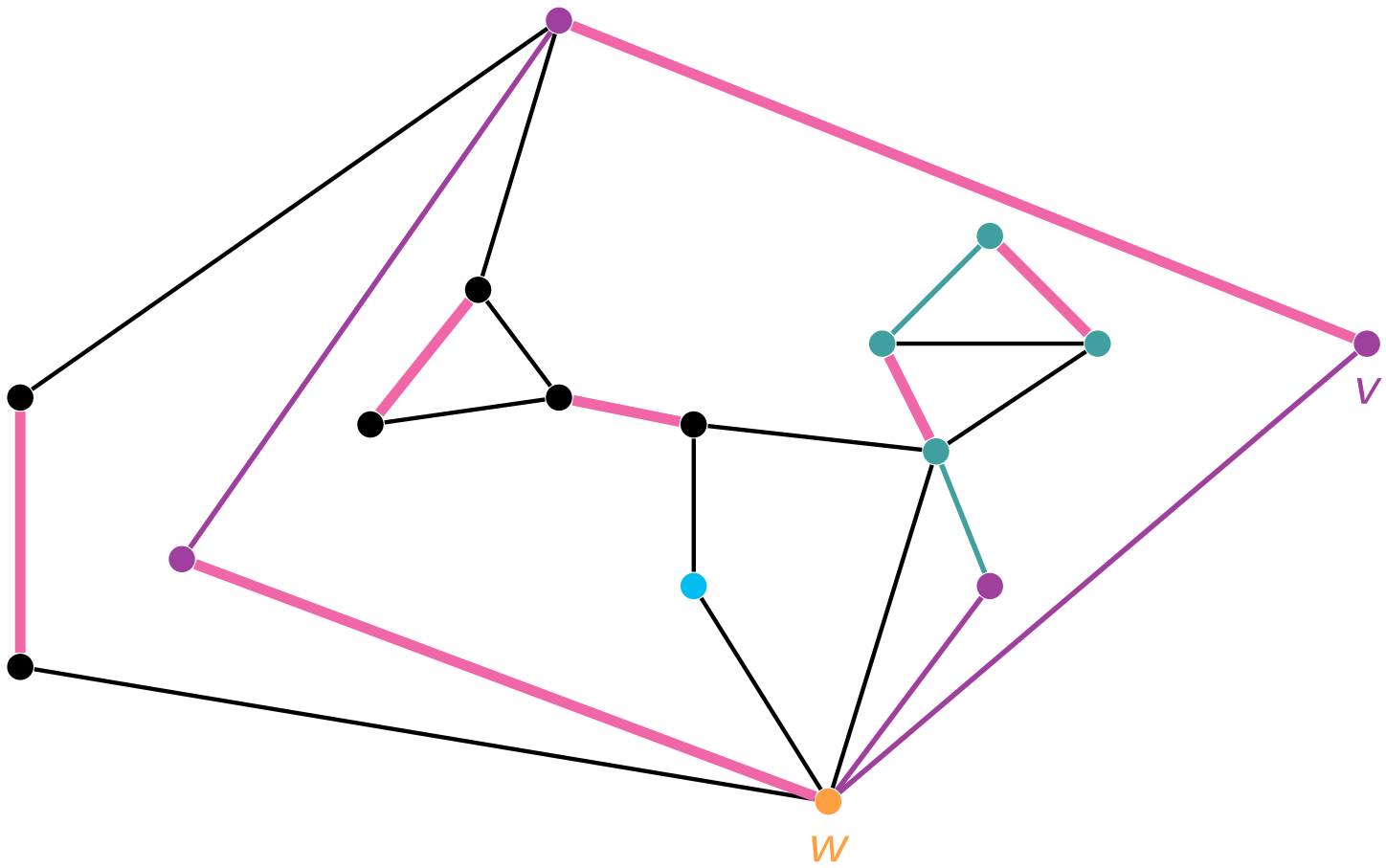

x
w
v

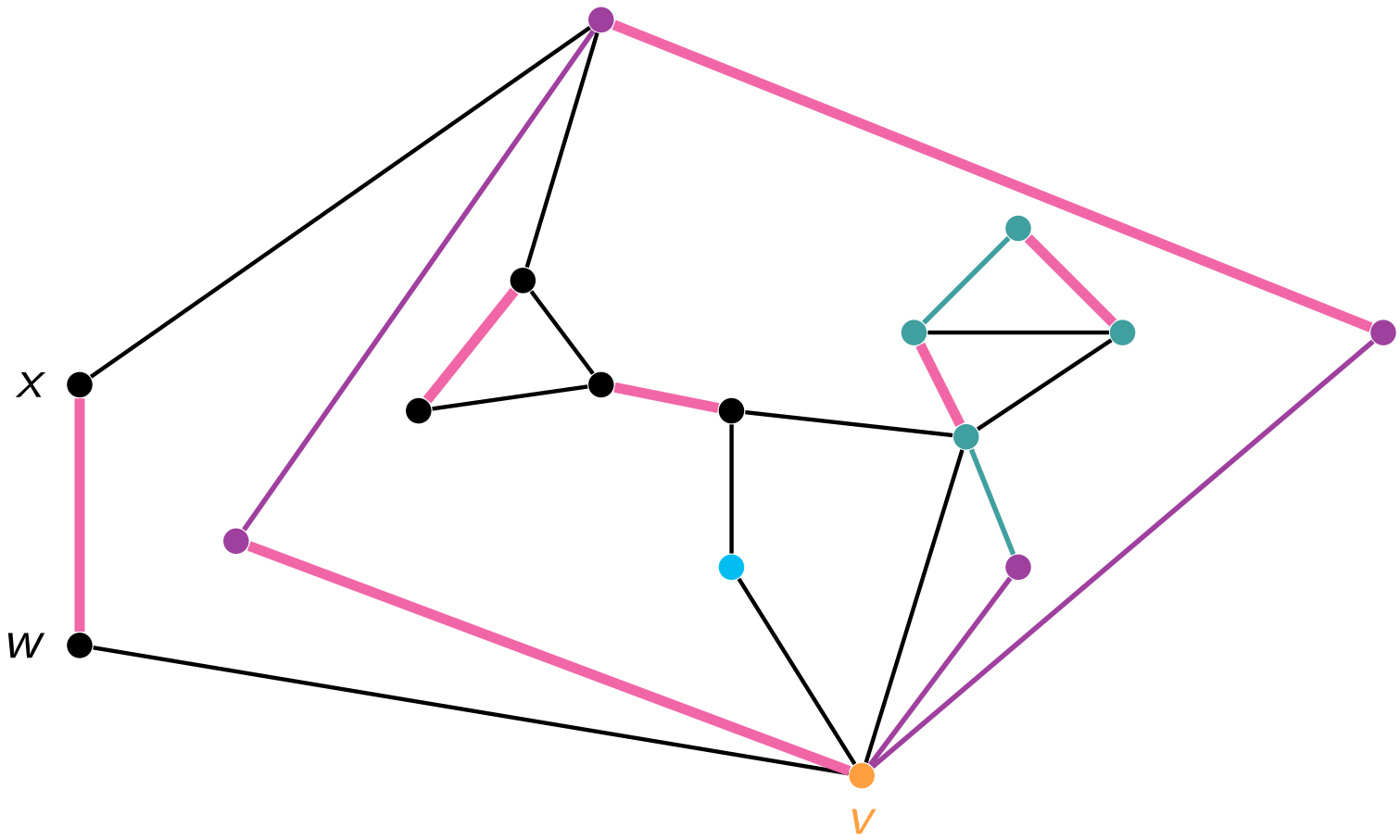

x
w
v

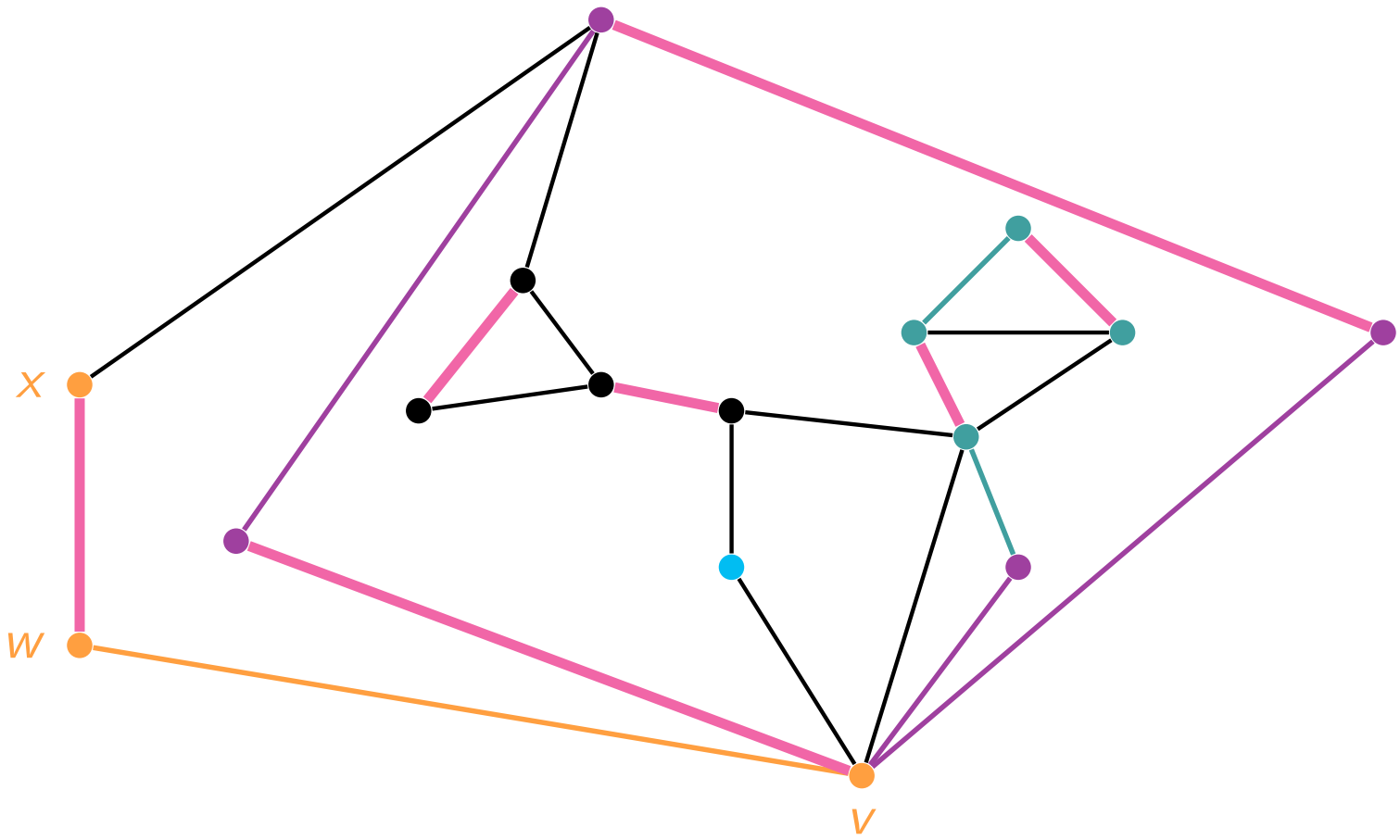

x
v
w

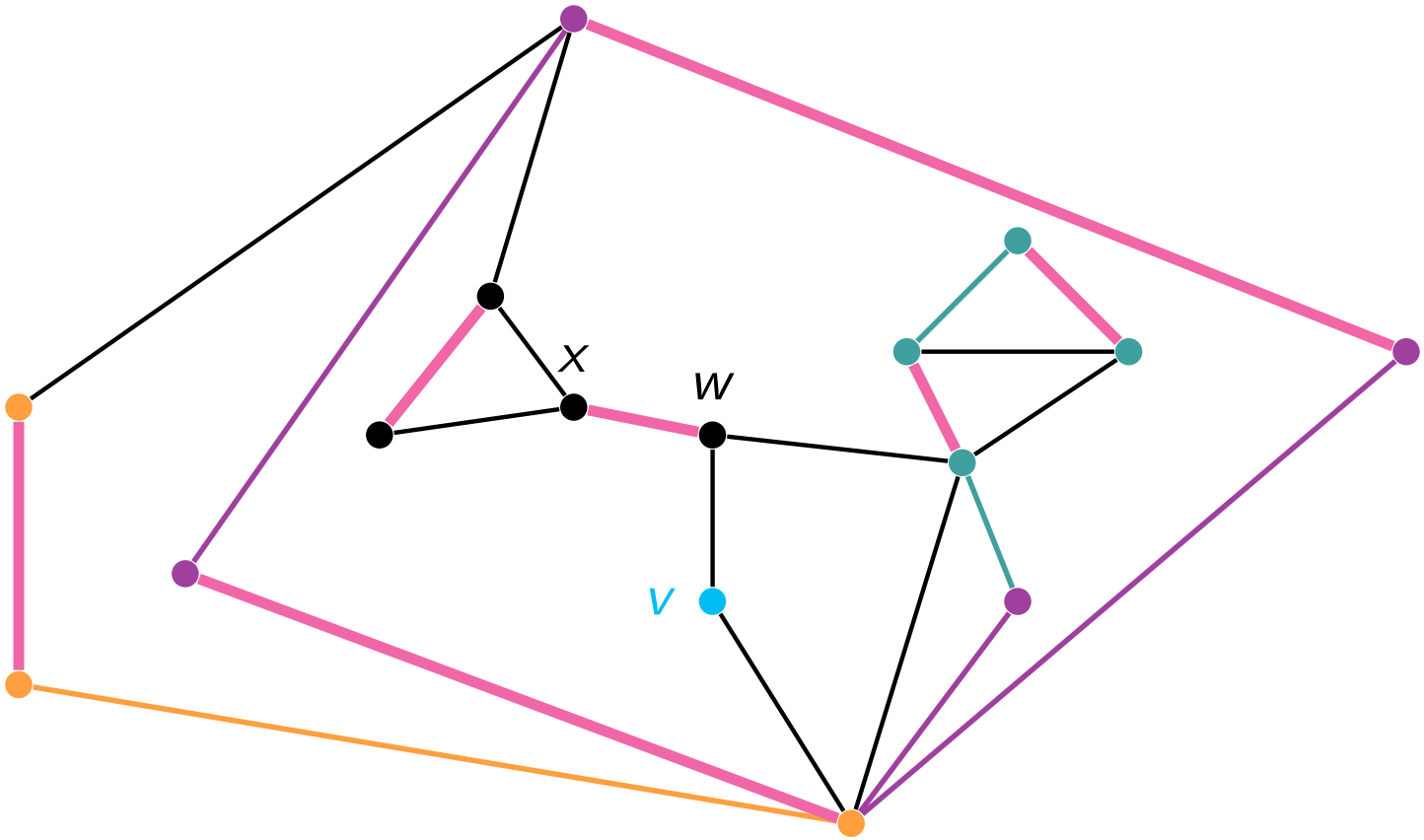

x
v
w

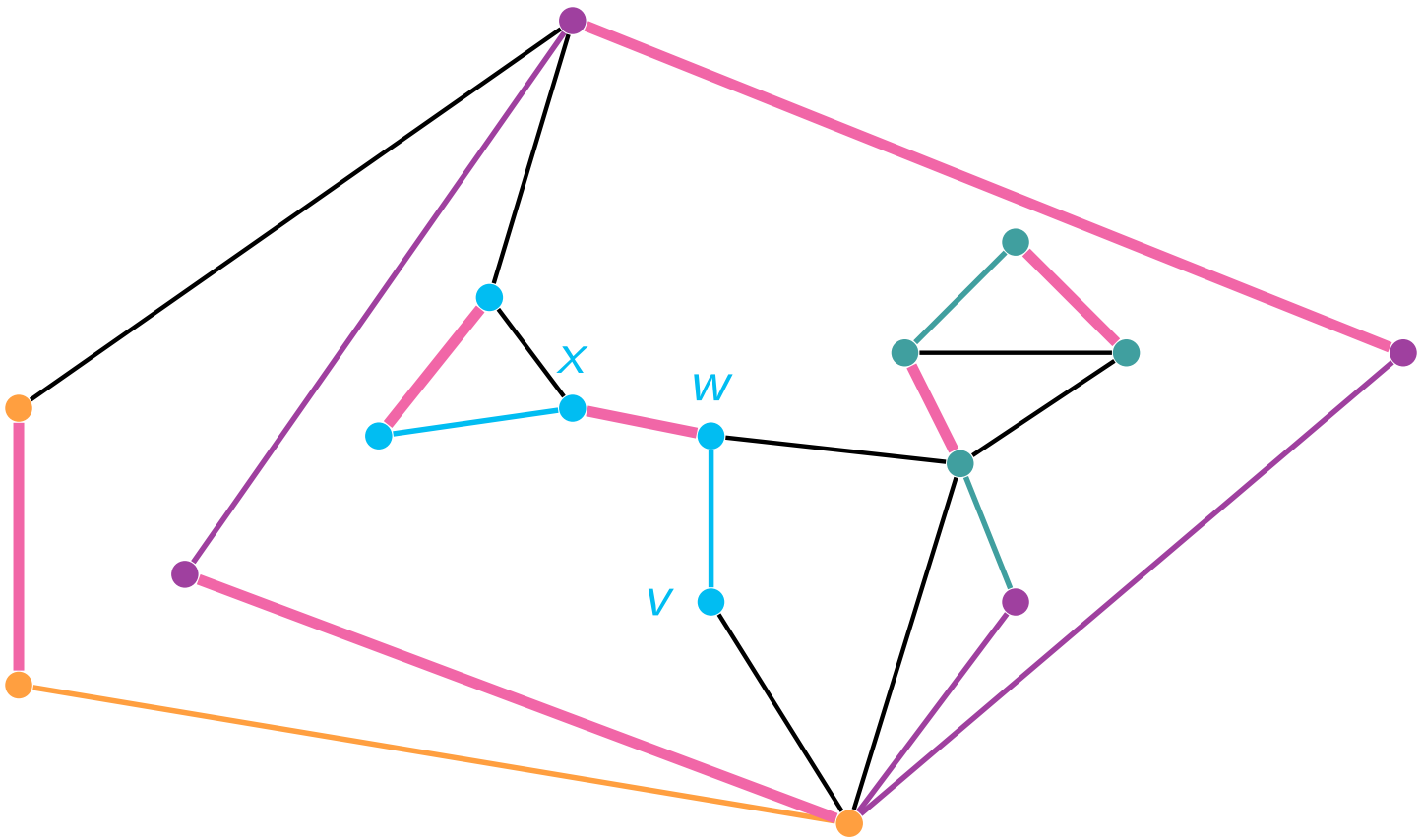

v
w

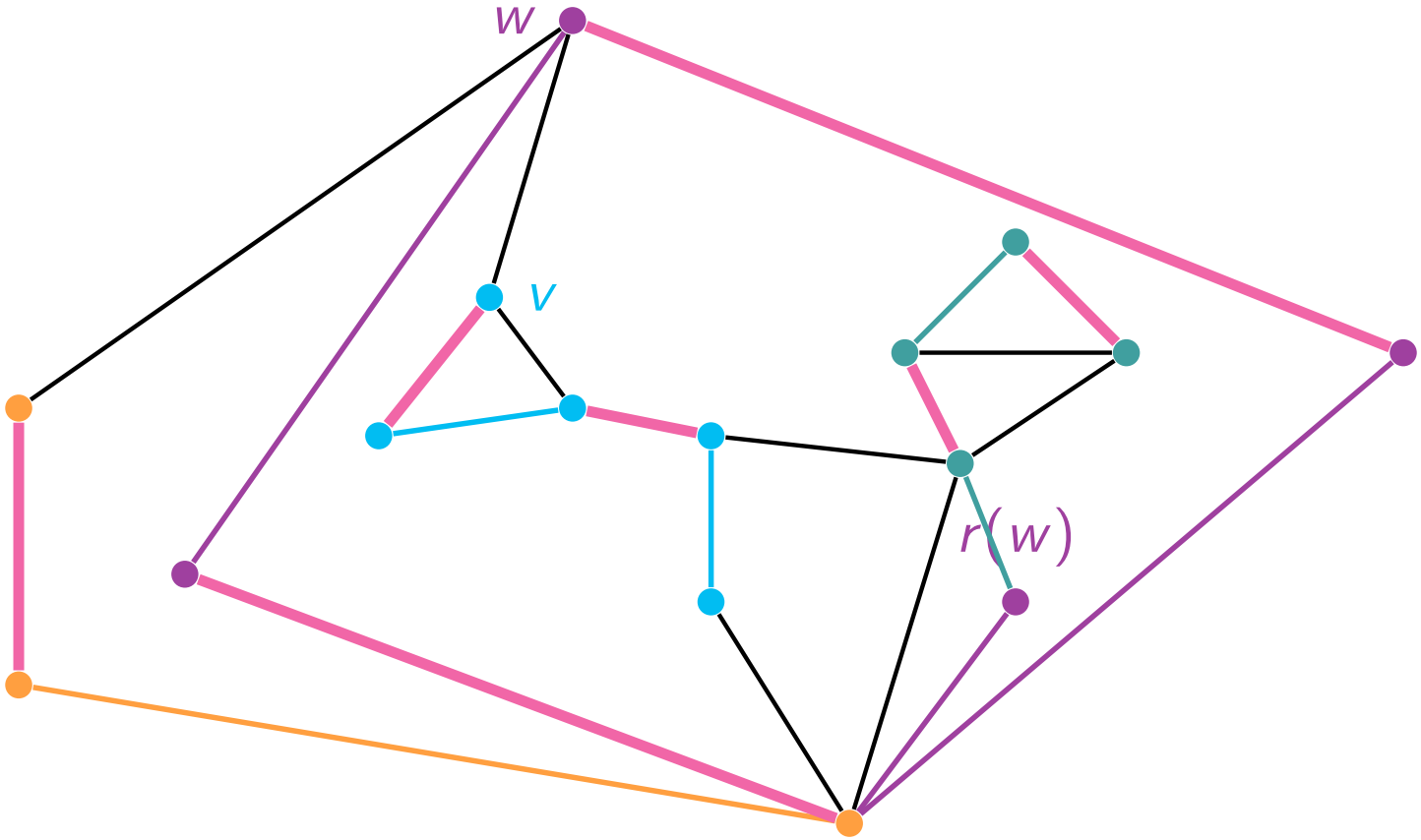

v
w

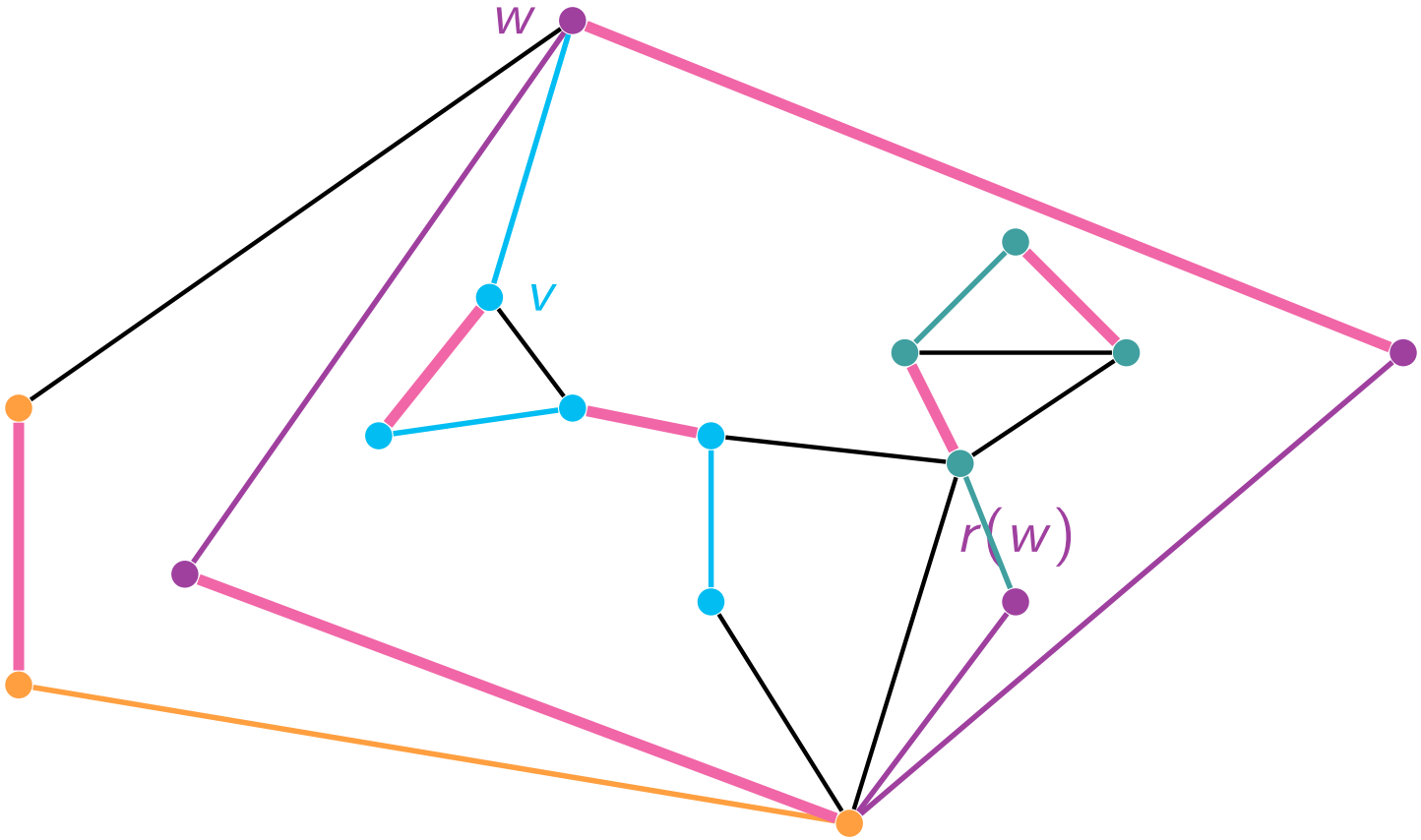

x
w
v

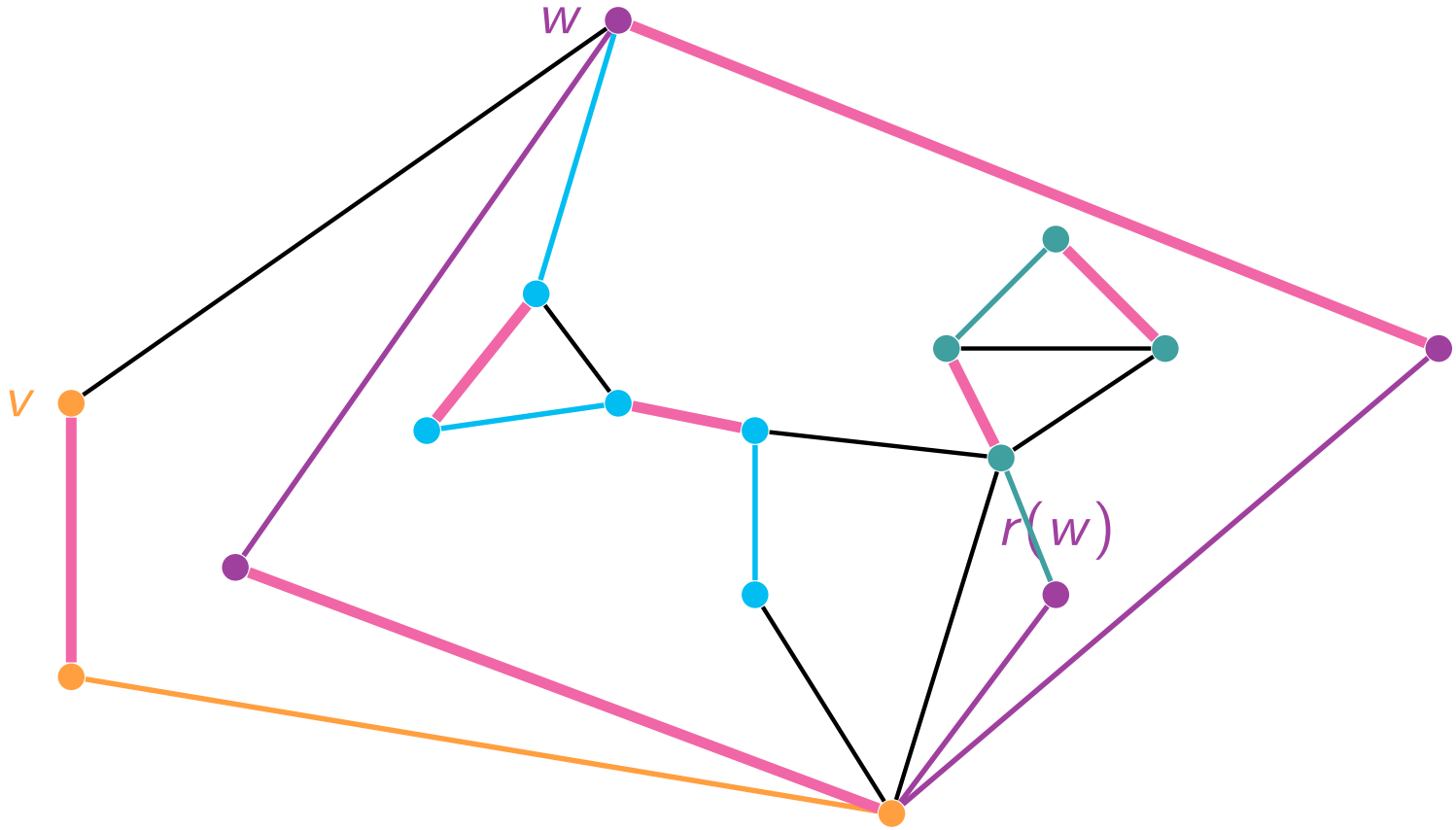

x
w
v

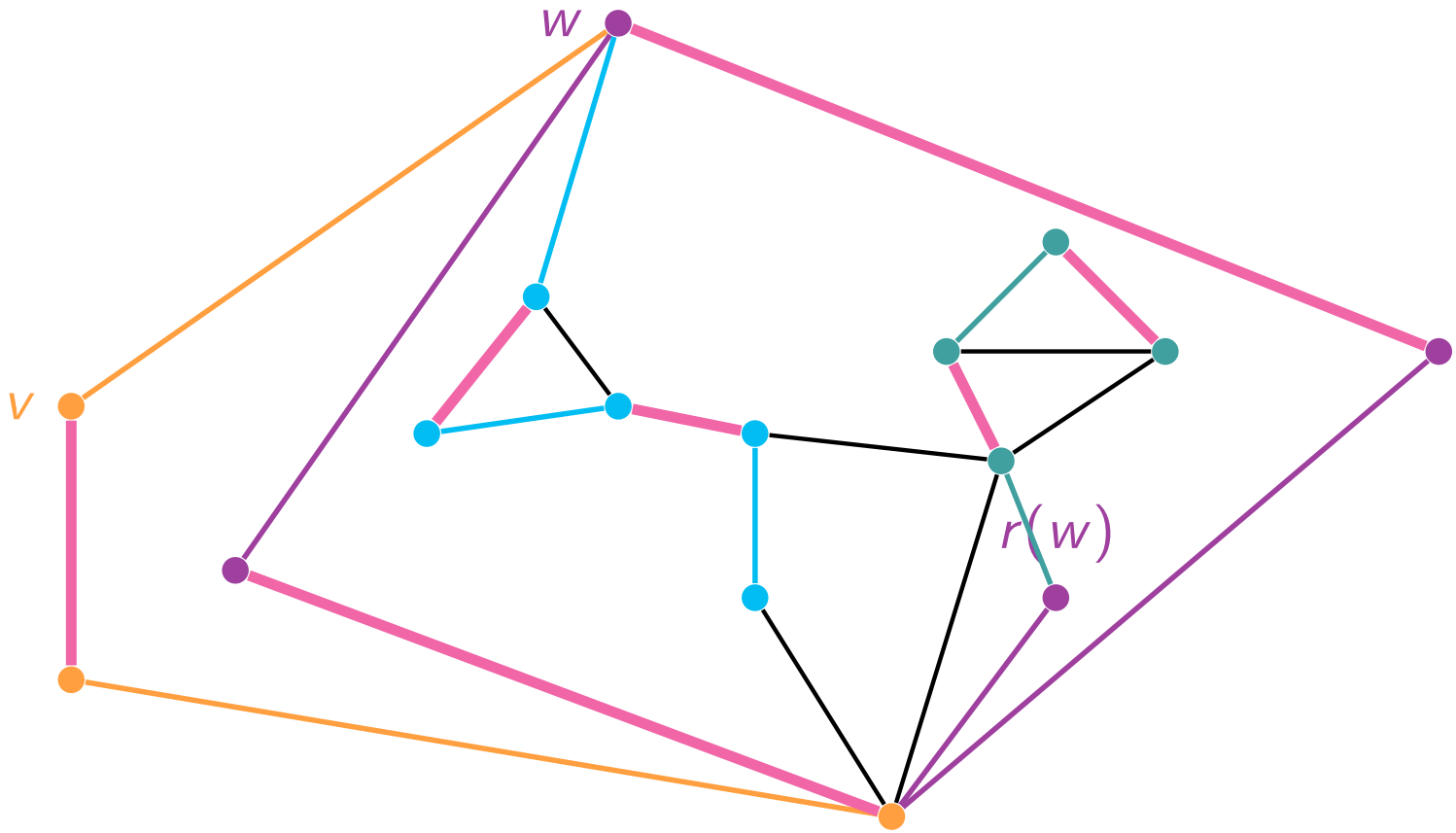

x
w
v

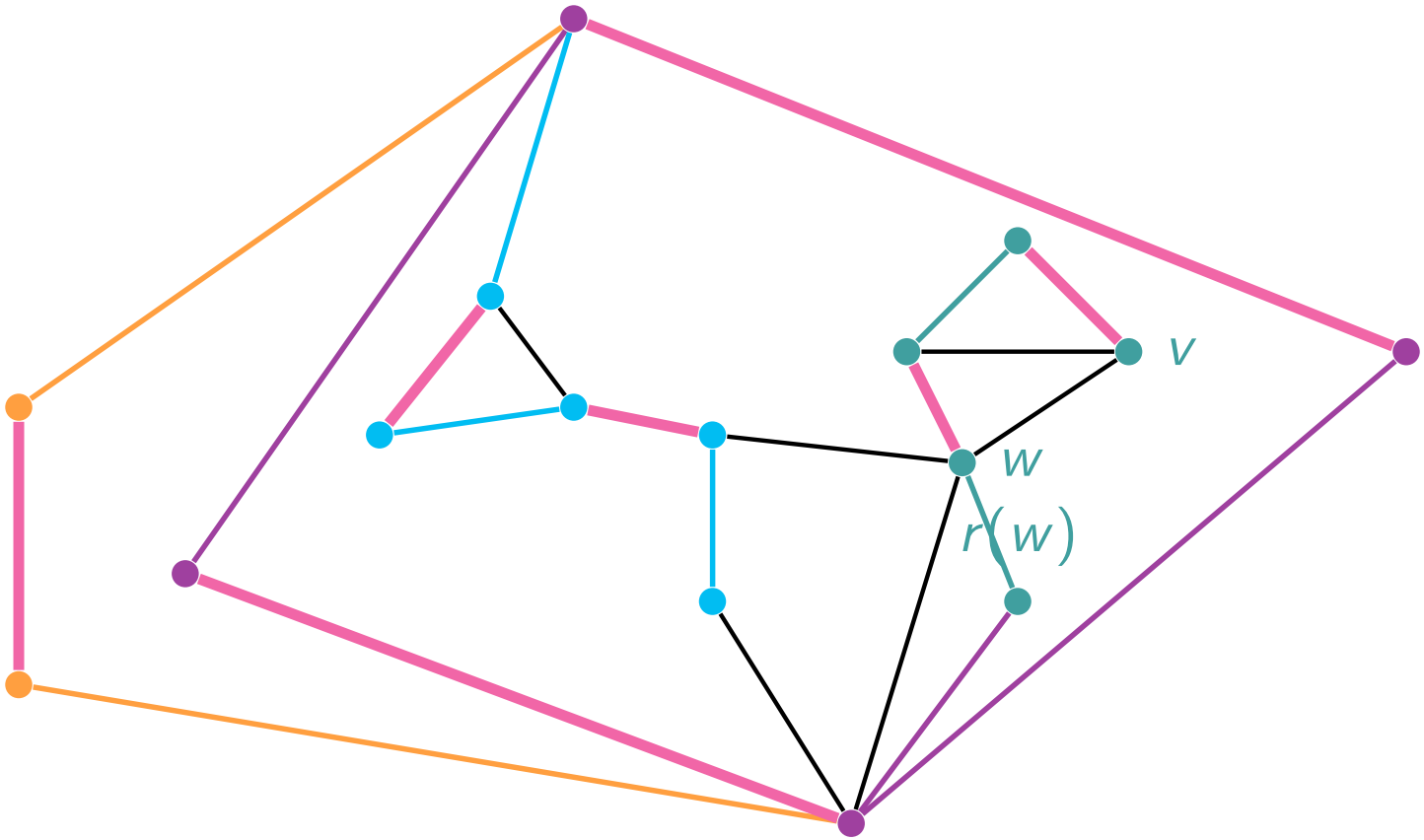

x
w
v

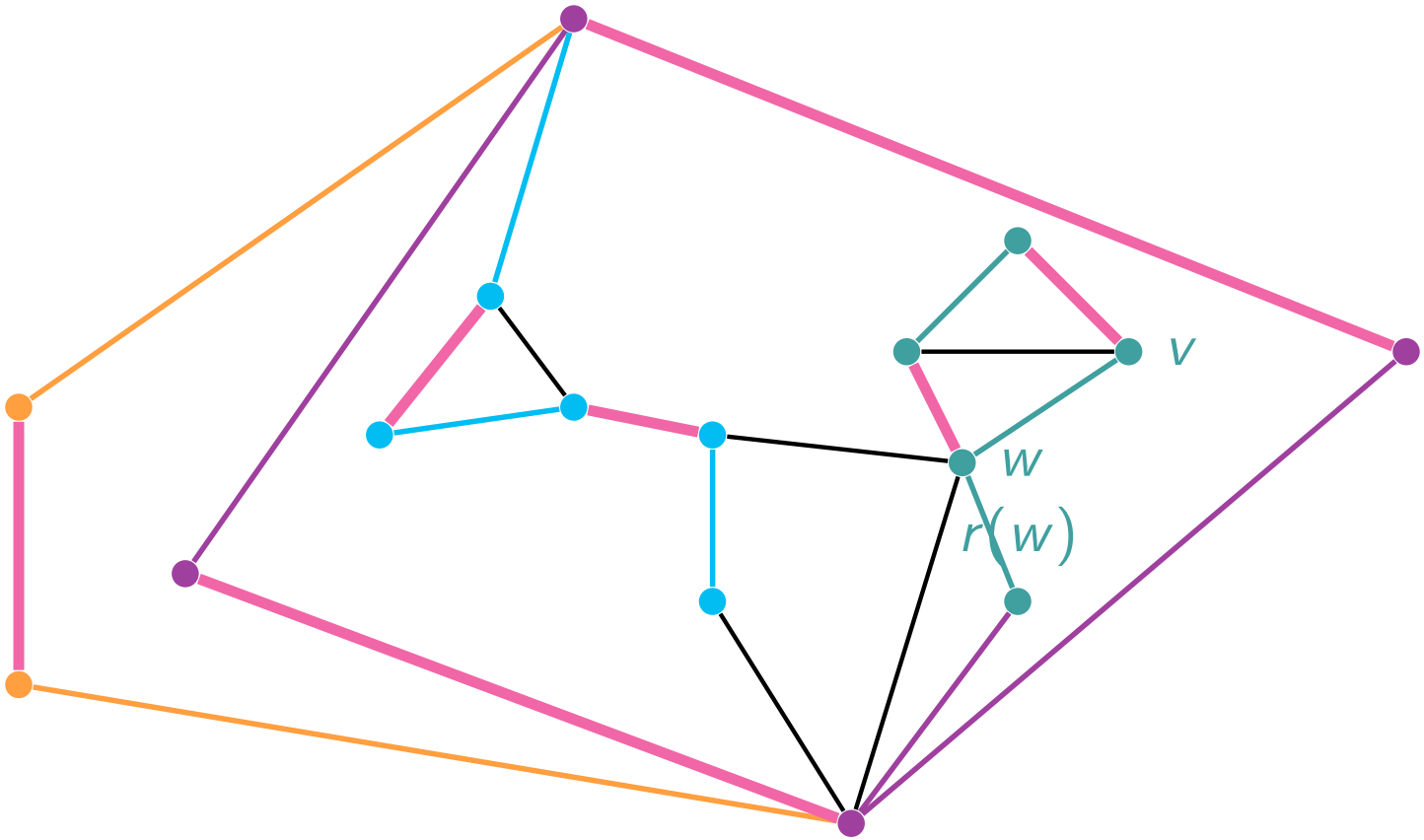

w
v
r(w)

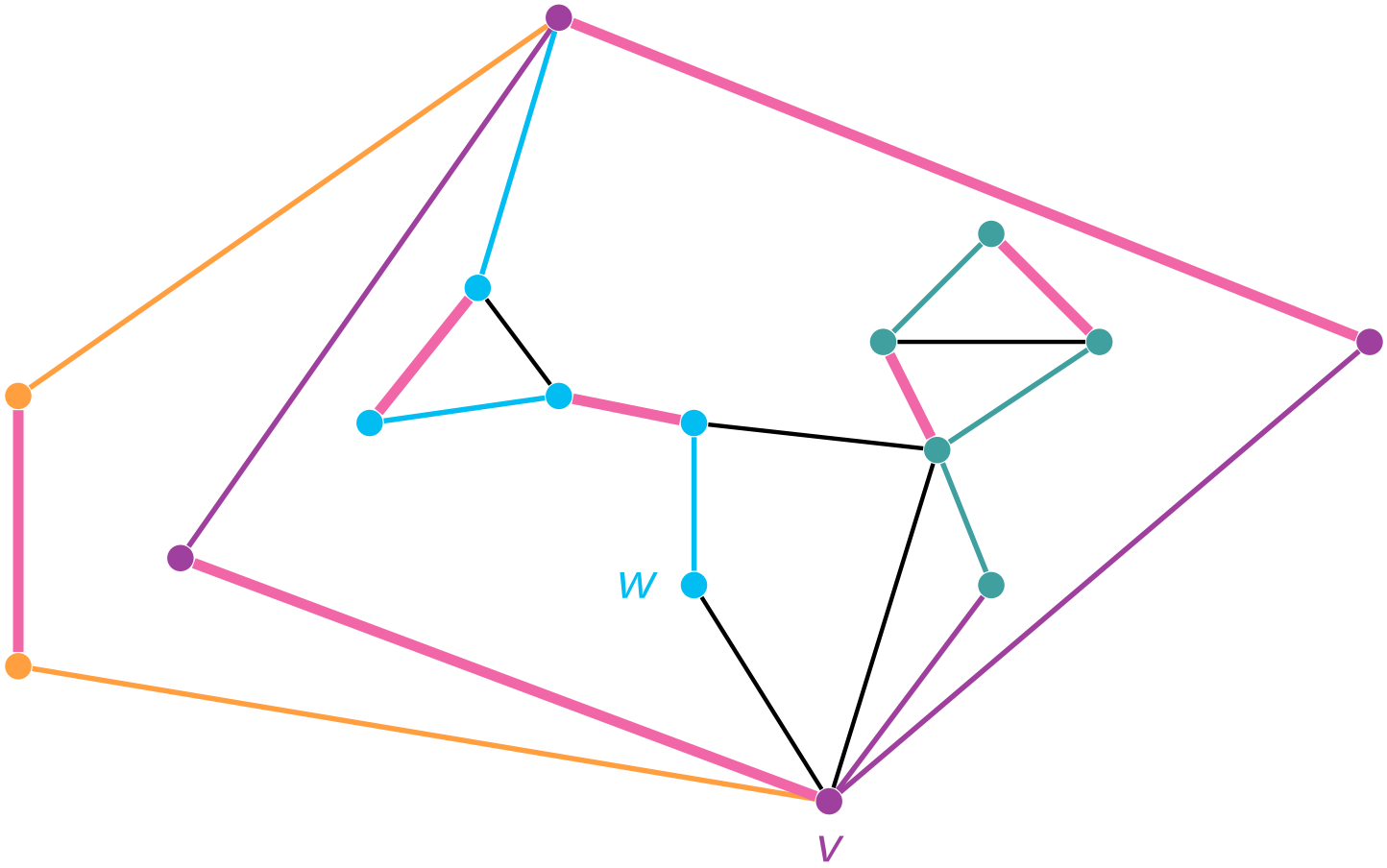

w
v
r(w)

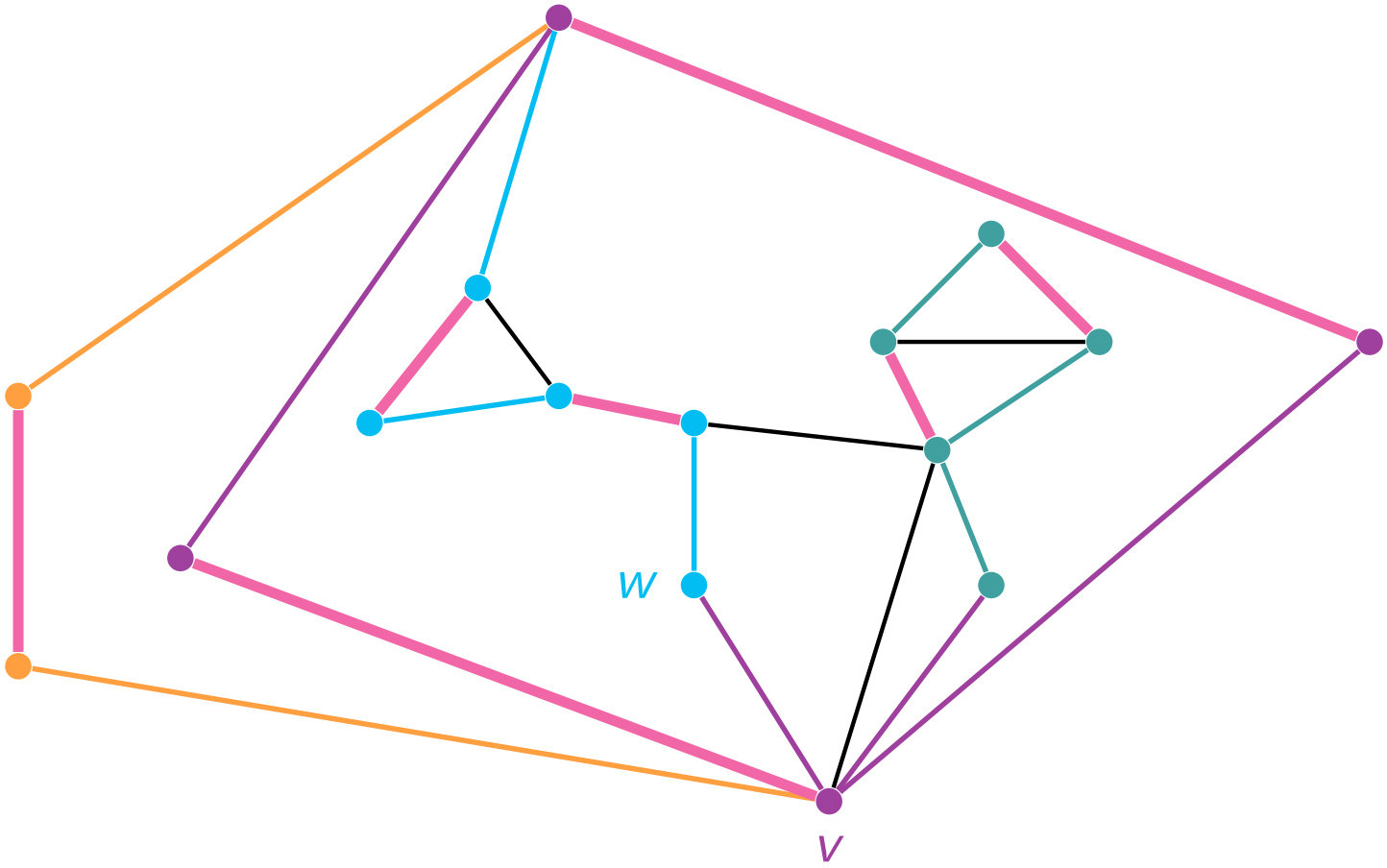

w
v
r(w)

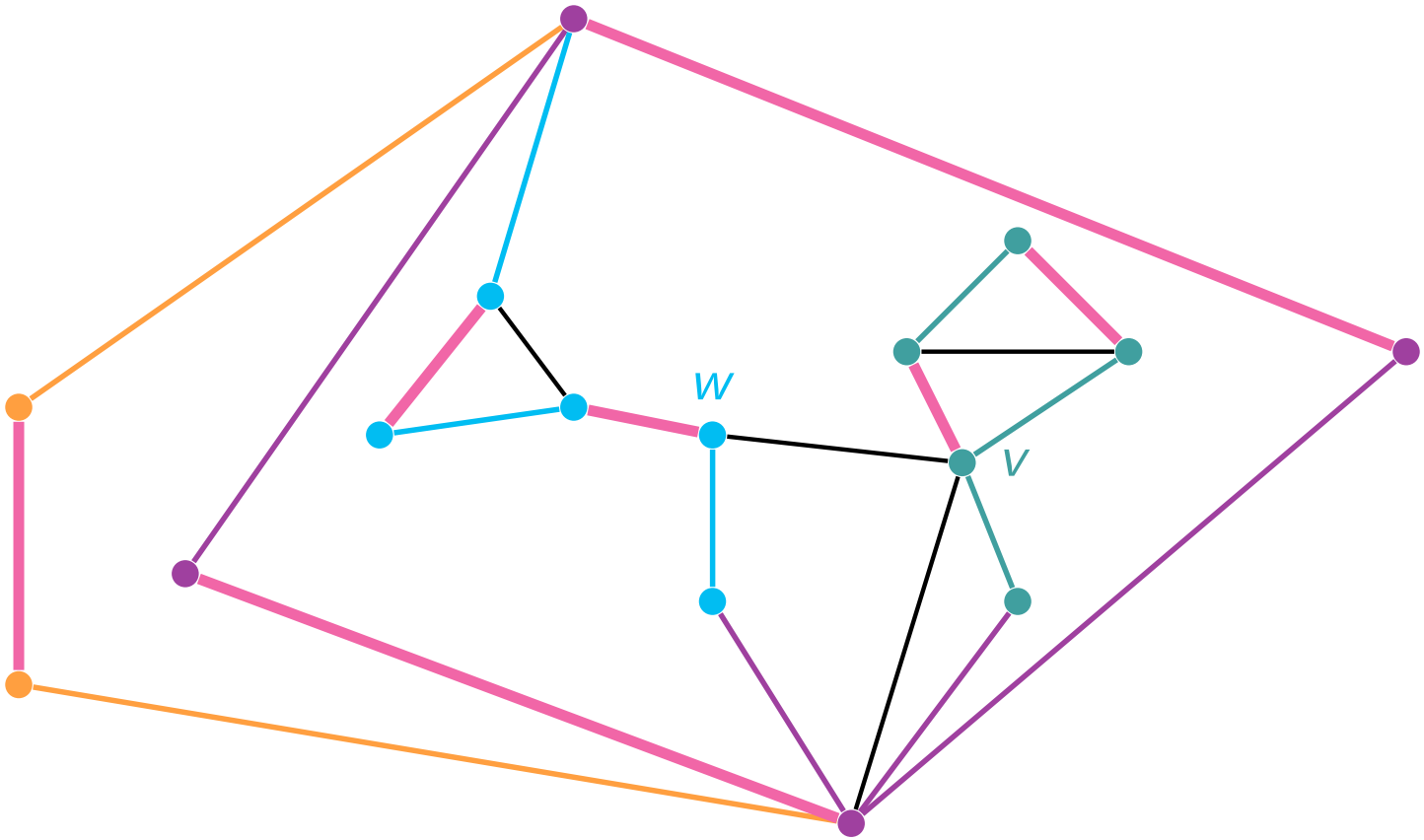

w
v
r(w)

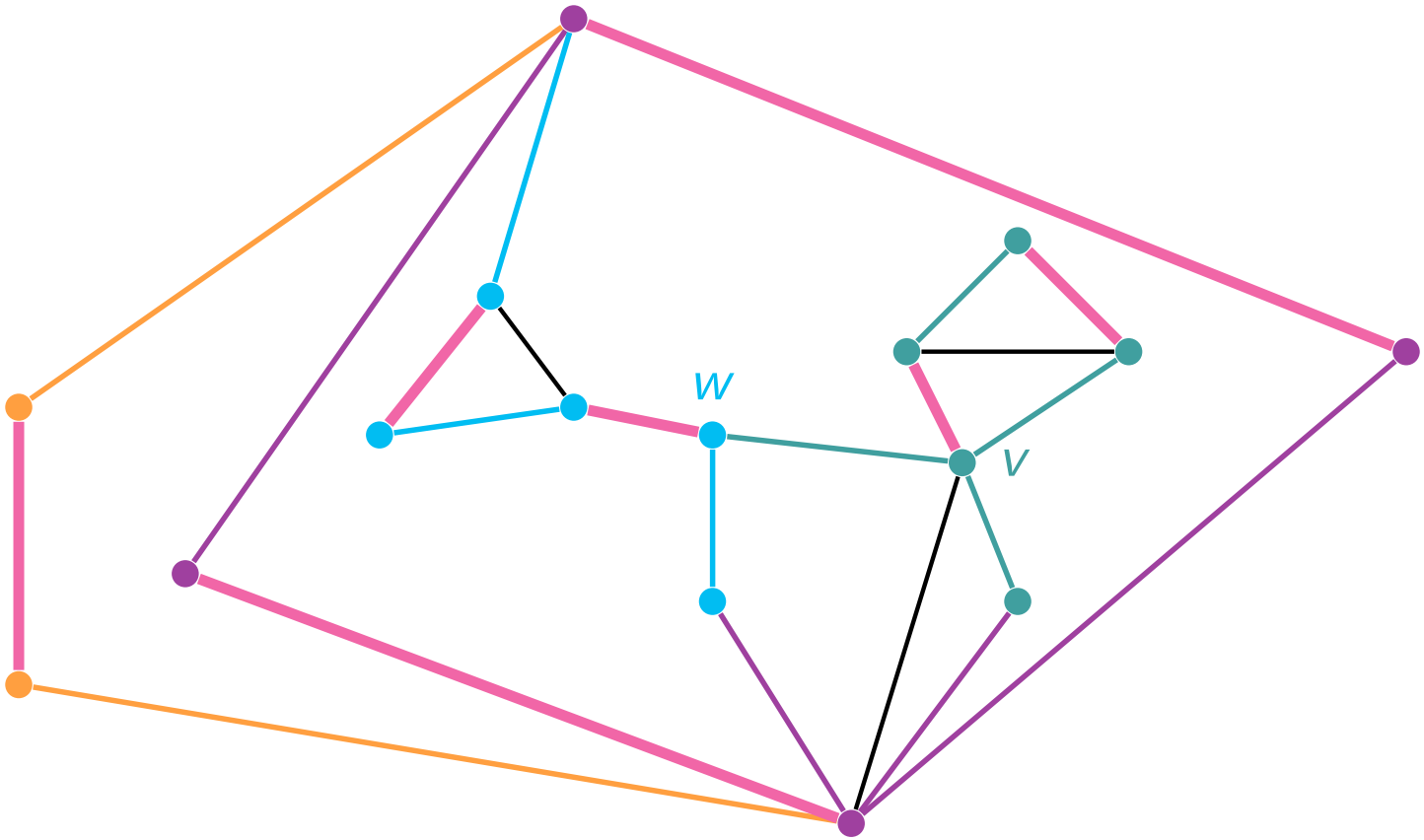

v
w
r(w)

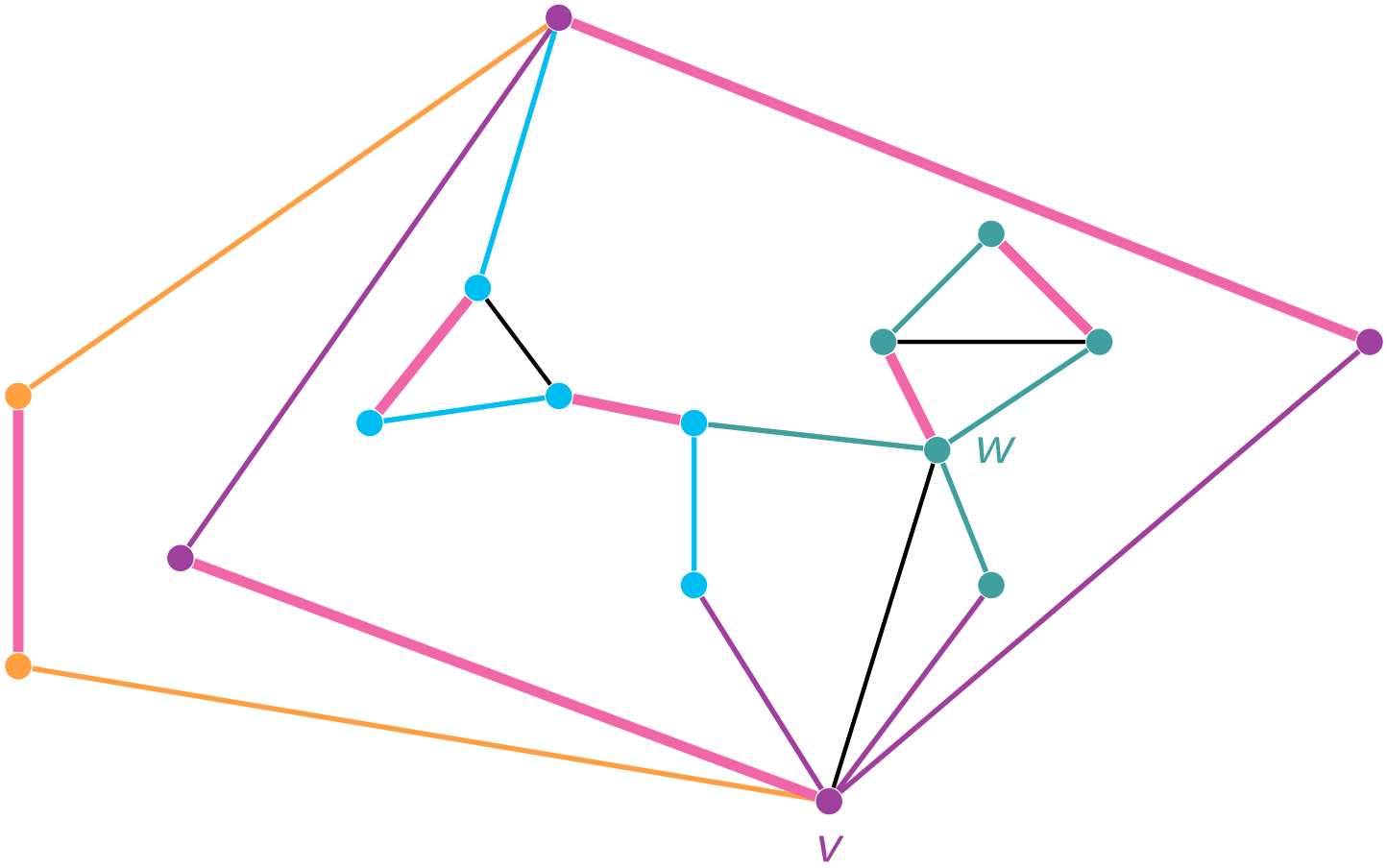

v
w
r(w)

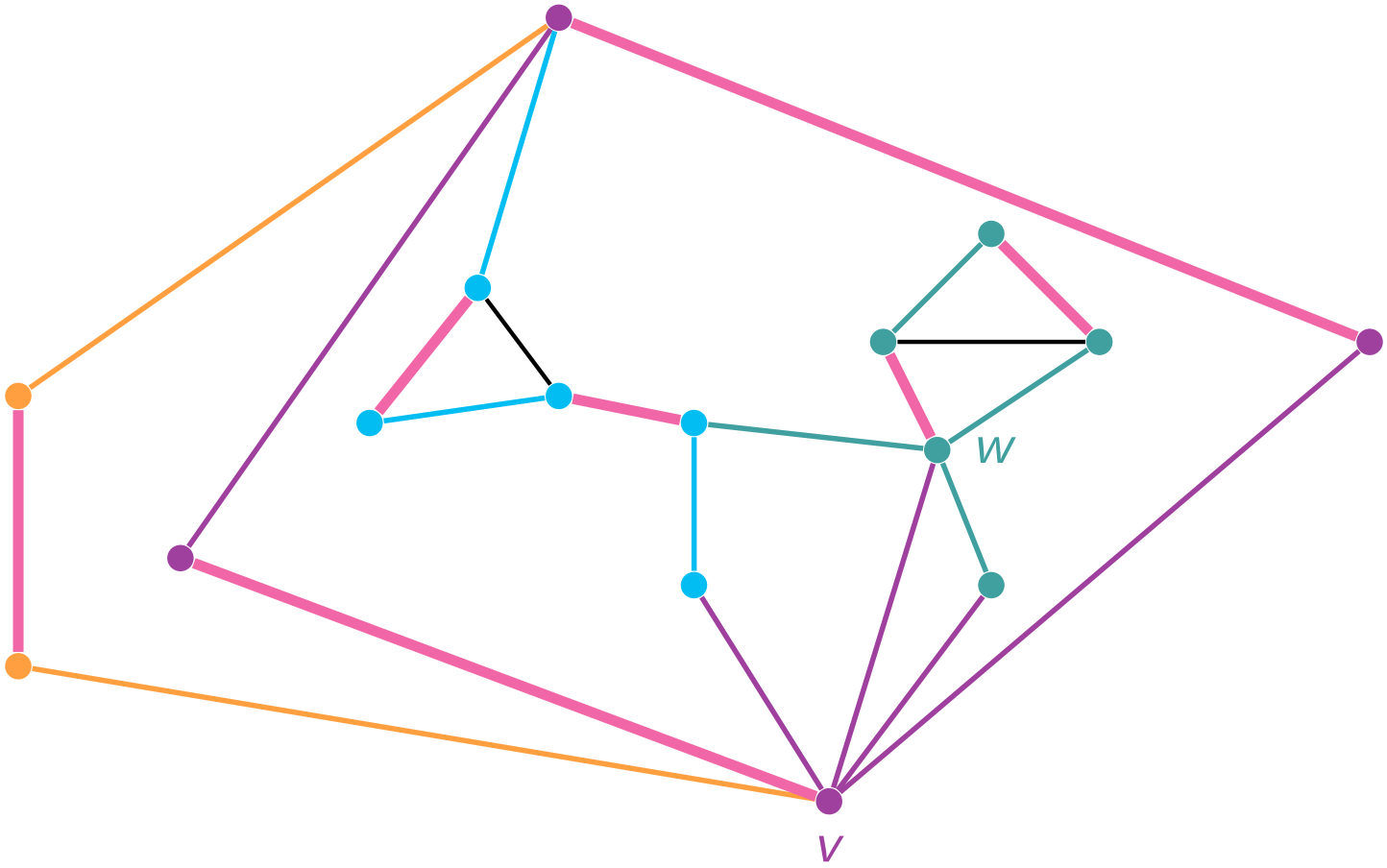

w
v

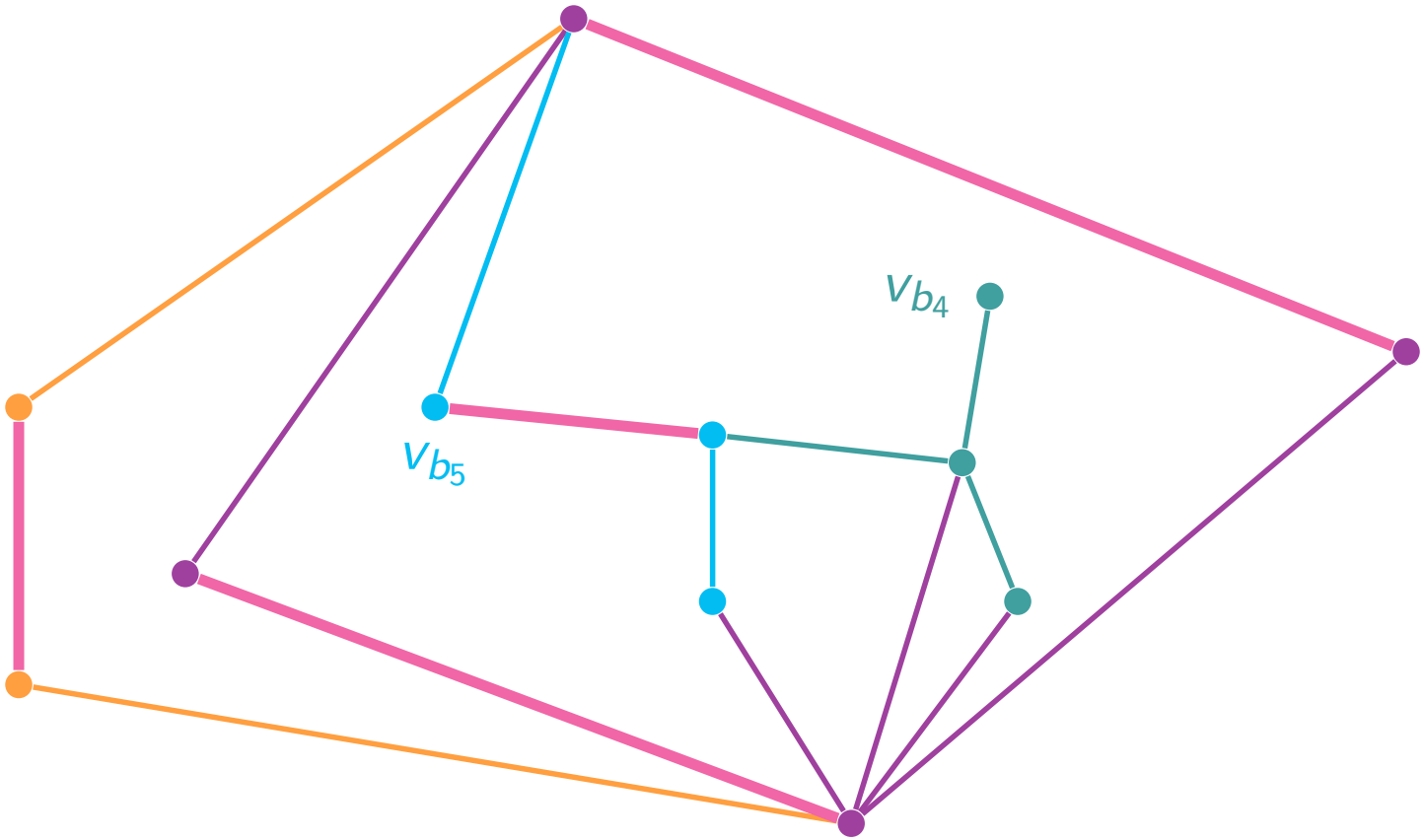

w
v

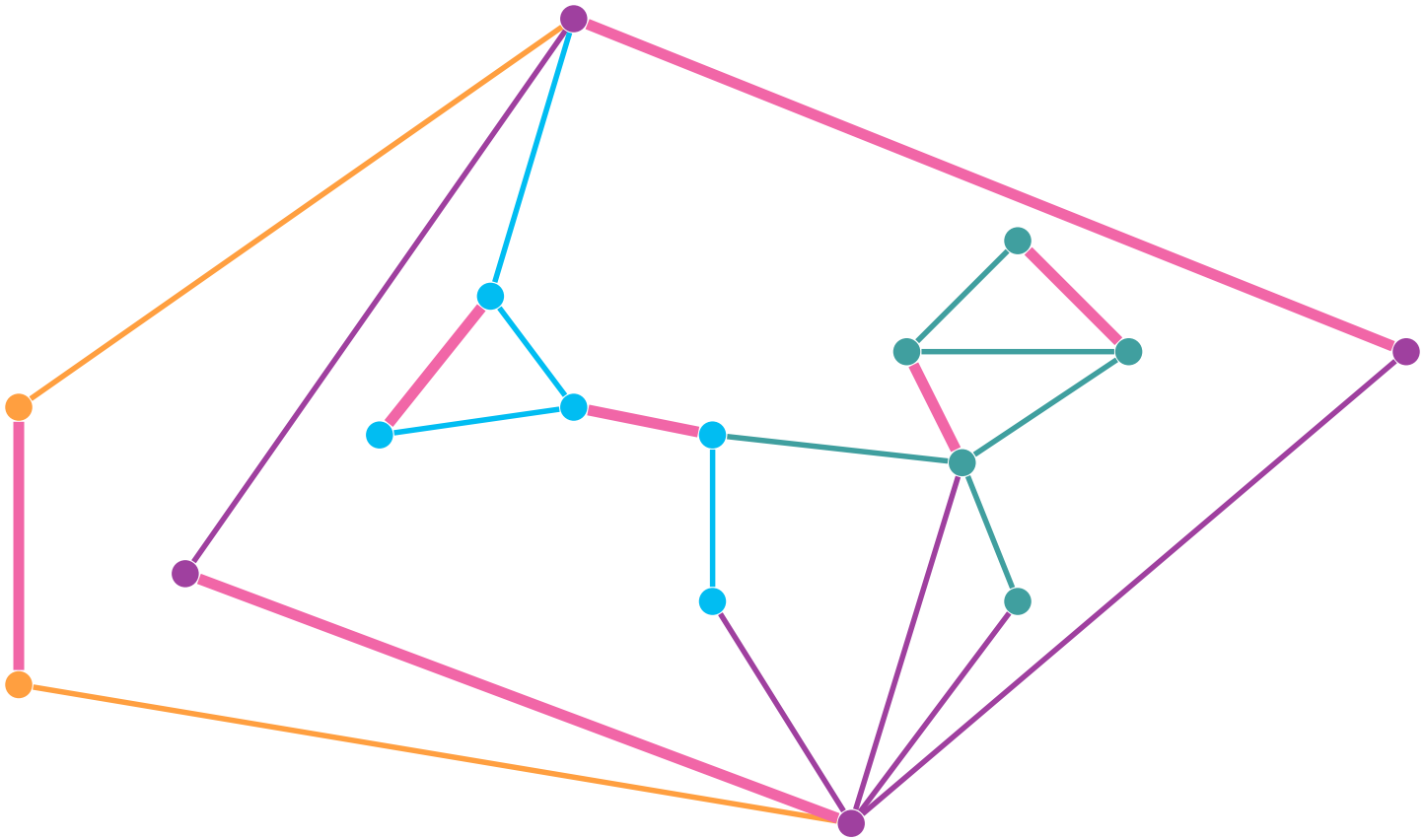

w
v

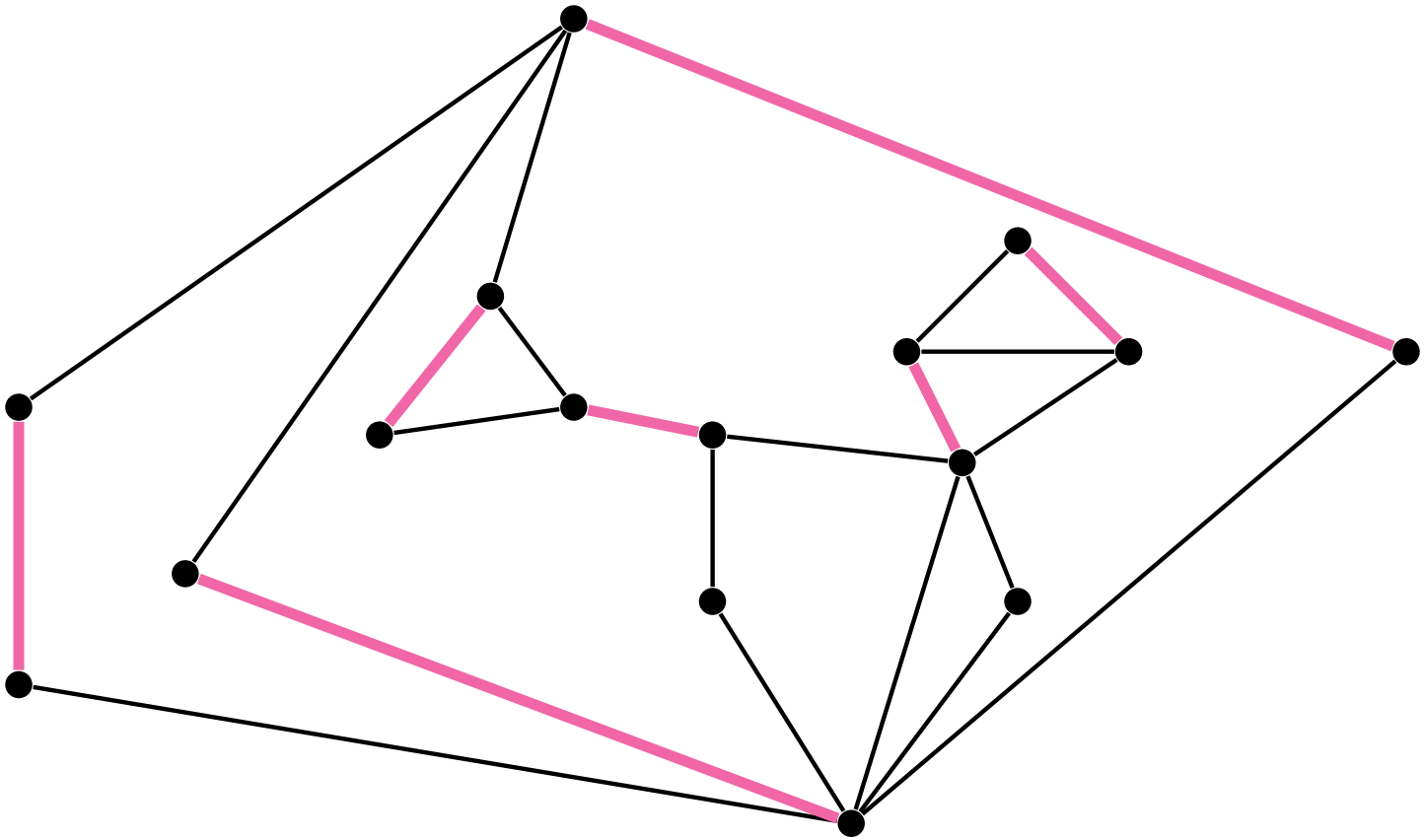

w
v


w
v


w
v


v_{b_4}
v_{b_5}





Referências

Blossom algorithm. Disponível em:
https://en.wikipedia.org/wiki/Blossom_algorithm

Paulo Feofiloff. *Curso de Otimização Combinatória*. Disponível em:
https://www.ime.usp.br/~pf/otimizacao-combinatoria/

Amy Shoemaker e Sagar Vare. *Edmonds' Blossom Algorithm*. Disponível em: https://stanford.edu/~rezab/classes/cme323/S16/

Jack Edmonds. *Paths, Trees, and Flowers*. Disponível em:
https://www.cambridge.org/core/journals/canadian-journal-of-mathematics/article/paths-trees-and-flowers/08B492B72322C4130AE800C0610E0E21

Reinhard Diestel. *Graph Theory*. Springer, New York, 2000.