A&W 2026

G-13 Timo Stucki, LFW E 13

Week 3

Matchings (with exercise)

Metric TSP with Matching

Colouring (with exercise)

Infos

ETH zürich

Feel free to contact me:

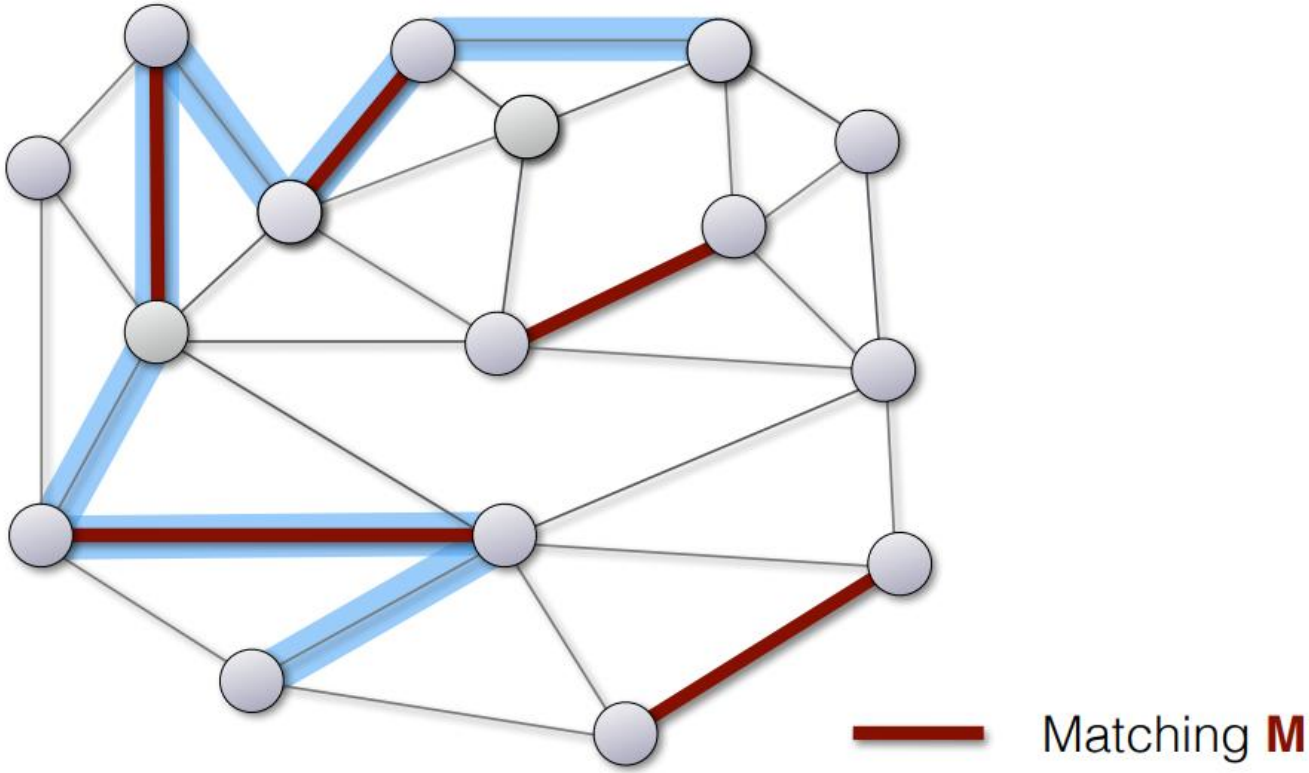
- tistucki@student.ethz.ch
- Discord: timostucki

Material:

- timostucki.com

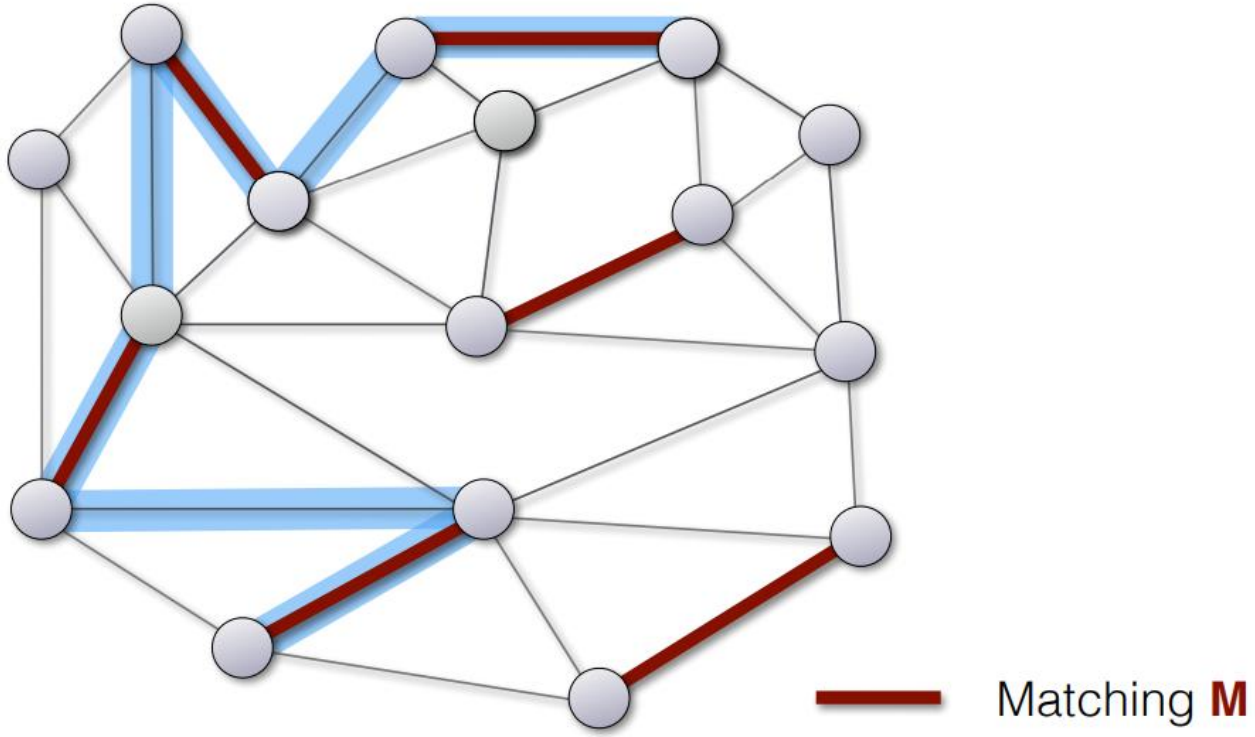
Matchings

Matchings



Ein **M-augmentierender Pfad P** ist ein Pfad, der abwechselnd Kanten aus **M** und nicht aus **M** enthält und der in von **M** nicht überdeckten Knoten beginnt und endet.

Matchings



Ein **M-augmentierender Pfad P** ist ein Pfad, der abwechselnd Kanten aus **M** und nicht aus **M** enthält und der in von **M** nicht überdeckten Knoten beginnt und endet.

⇒ durch *Tauschen* entlang M können wir das Matching vergrössern:

M' := **M** ⊕ **P**

Matchings

Satz 1.48 (Satz von Berge). Ist M ein Matching in einem Graphen $G = (V, E)$, das nicht kardinalitätsmaximal ist, so existiert ein augmentierender Pfad zu M.

→ We can make an algorithm out of this!

Matchings

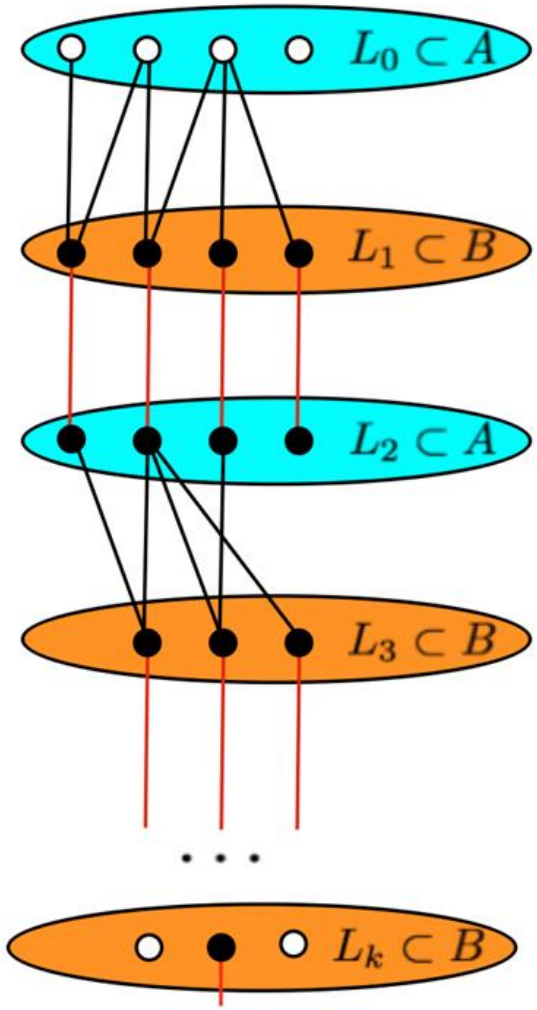
AUGMENTING_PATH $(G = (A \uplus B, E), M)$

1: $L_0 := \{\text{unüberdeckte Knoten in } A\}$
2: Markiere alle Knoten aus L_0 als besucht.
3: **if** $L_0 = \emptyset$ **then**
4: **return** M ist maximal
5: **for all** $i = 1$ **to** n **do**
6: **if** i ungerade **then**
7: $L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } E \setminus M\}$
8: **else**
9: $L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } M\}$
10: Markiere alle Knoten aus L_i als besucht.
11: **if** L_i enthält unüberdeckten Knoten v **then**
12: Finde Pfad P von L_0 nach v durch backtracking
13: **return** P // *terminiert Algorithmus*
14: **return** M ist schon maximal

Matchings

AUGMENTING_PATH $(G = (A \uplus B, E), M)$

1: $L_0 := \{\text{unüberdeckte Knoten in } A\}$
2: Markiere alle Knoten aus L_0 als besucht.
3: **if** $L_0 = \emptyset$ **then**
4: **return** M ist maximal
5: **for all** $i = 1$ **to** n **do**
6: **if** i ungerade **then**
7: $L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } E \setminus M\}$
8: **else**
9: $L_i := \{\text{unbesuchte Nachbarn von } L_{i-1} \text{ via Kanten in } M\}$
10: Markiere alle Knoten aus L_i als besucht.
11: **if** L_i enthält unüberdeckten Knoten v **then**
12: Finde Pfad P von L_0 nach v durch backtracking
13: **return** P // *terminiert Algorithmus*
14: **return** M ist schon maximal



Matchings

- For 1 iteration: $O(|E|)$
- In total after at most n/2 iterations: $O(|V||E|)$

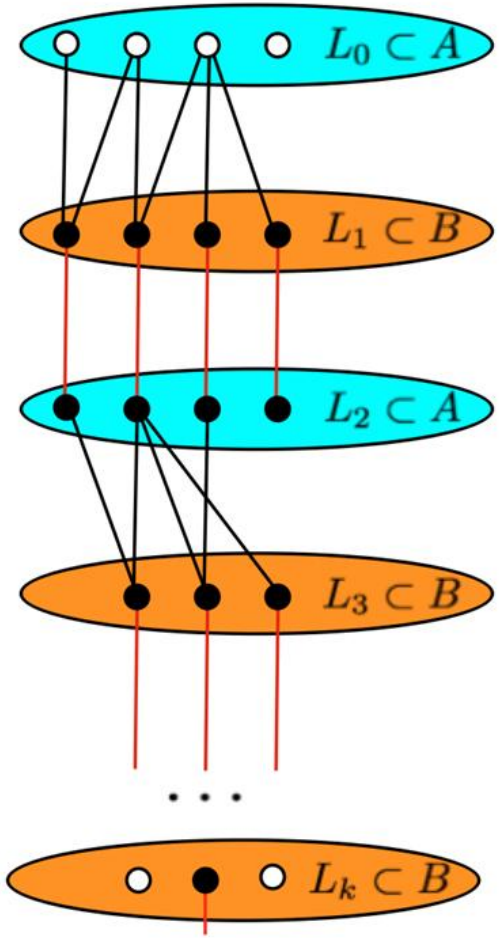
(since a maximum matching has at most n/2 edges)

Matchings

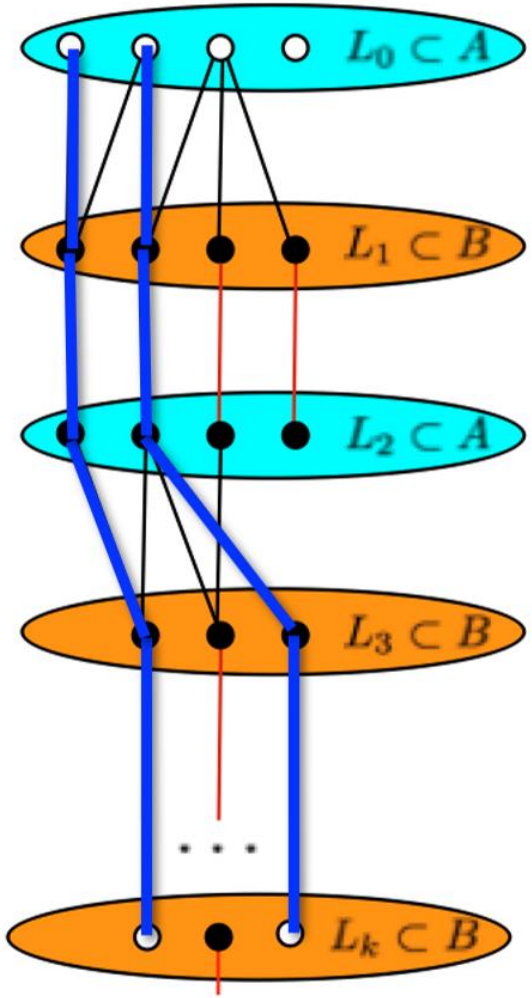
- Preview:

Satz 1.49. Der Algorithmus von Hopcroft und Karp durchläuft die while-Schleife nur $O(\sqrt{|V|})$ Mal. Er berechnet daher ein maximales Matching in einem bipartiten Graphen in Zeit $O(\sqrt{|V|} \cdot (|V| + |E|))$.

Matchings



k := Länge eines kürzesten augmentierenden Pfades
Finde eine inklusionsmaximale Menge S von paarweise disjunkten
gmentierenden Pfaden der Länge k.

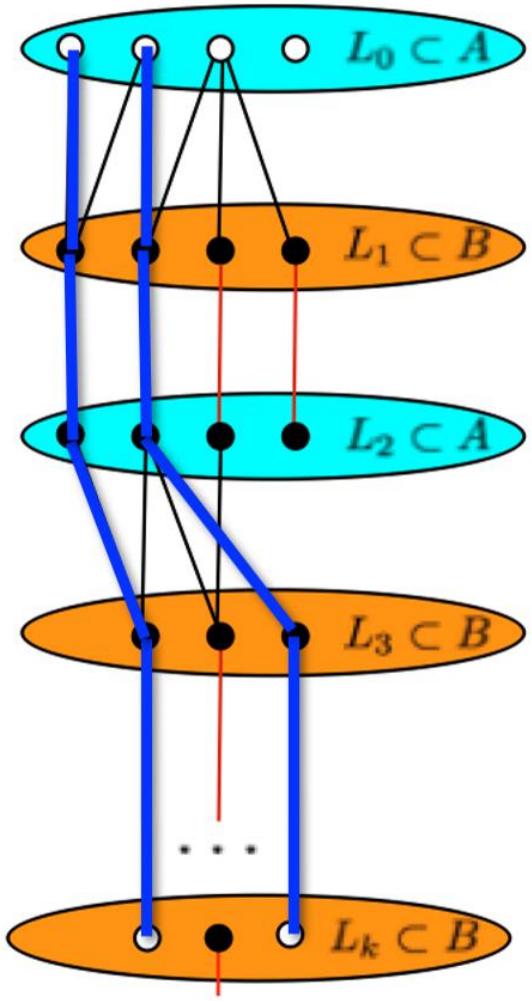


one augmenting path at a time

Matchings

MAXIMAL_MATCHING $(G = (A \oplus B, E))$ (Hopcroft und Karp)

1: $M := \{e\}$ für irgendeine Kante $e \in E$.
2: **while** es gibt noch augmentierende Pfade **do**
3: $k :=$ Länge eines kürzesten augmentierenden Pfades
4: Finde eine inklusionsmaximale Menge S von paarweise disjunkten augmentierenden Pfaden der Länge k.
5: **for all** P aus S **do**
6: $M := M \oplus P$. // *augmentiere entlang der Pfade aus S*
7: **return** M



Matchings

Satz 1.49. Der Algorithmus von Hopcroft und Karp durchläuft die while-Schleife nur $O(\sqrt{|V|})$ Mal. Er berechnet daher ein maximales Matching in einem bipartiten Graphen in Zeit $O(\sqrt{|V|} \cdot (|V| + |E|))$.

Exercise S4

Matchings

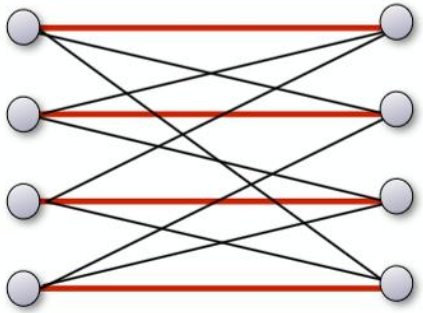
Theorem: (Hall, 1935)
Ein bipartiter graph $G=(A \dot\cup B, E)$ enthält ein Matching M der Kardinalität $|M|=|A|$ **gdw** $\forall A' \subseteq A : |A'| \leq |N(A')|$


Philip Hall
(1904-1982)

Korollar: (Frobenius, 1917)
Für alle k gilt: jeder k-reguläre bipartite graph enthält ein perfektes Matching.

Es gilt sogar: Graph ist Vereinigung von perfekten Matchings.


Ferdinand Georg Frobenius
(1849 – 1917)



Ein Graph G heisst k-*regulär* (engl. k-*regular*), falls für alle Knoten $v \in V$ gilt, dass $\deg(v) = k$.

Exercise S4.1 – *Perfect Bipartite Matching*

Let $G = (A \dot{\cup} B, E)$ be a bipartite k-regular graph (i.e. every vertex has degree exactly k).

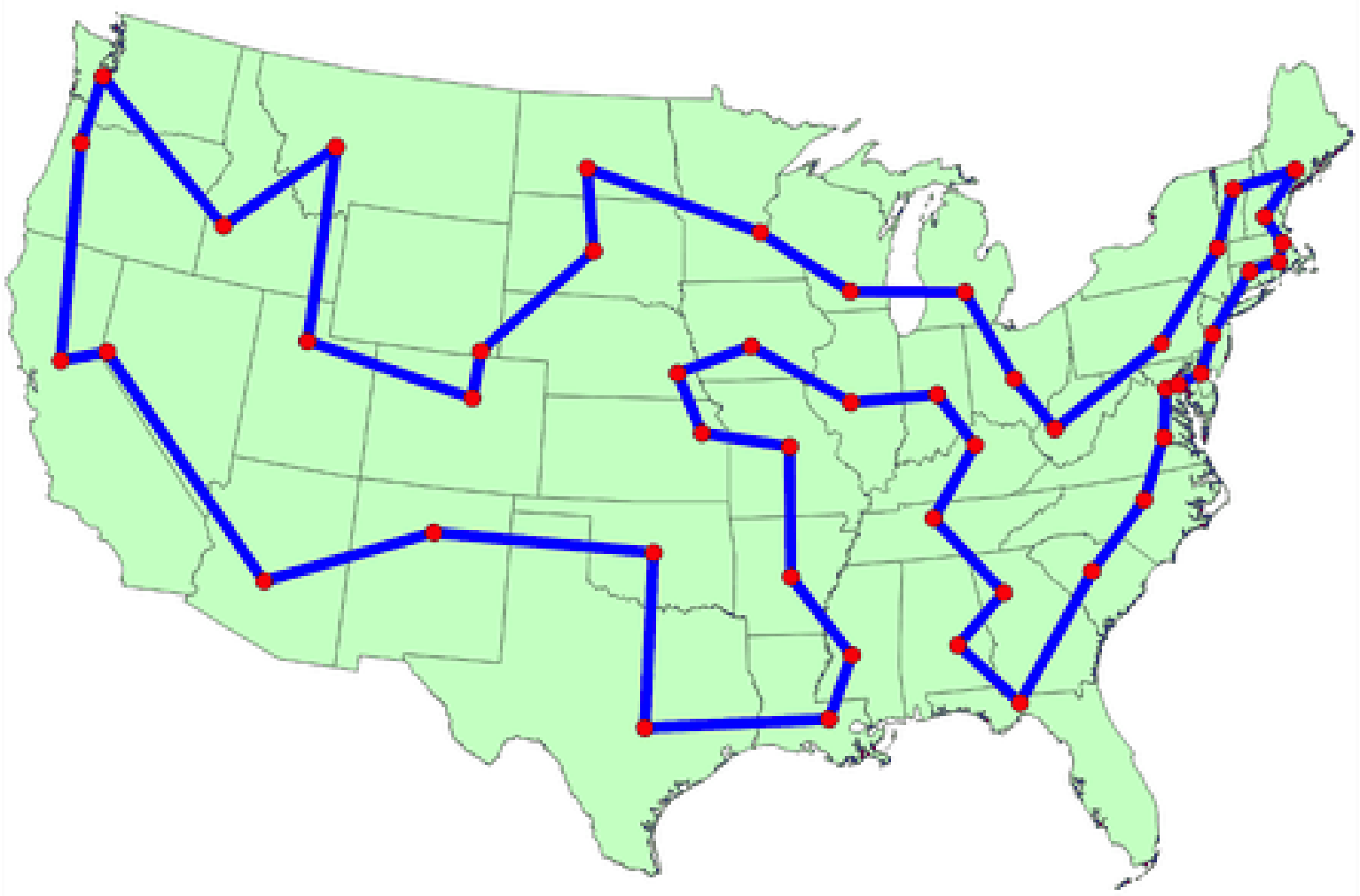
(a) Show that G contains a perfect matching.

(b) Show that G contains k pairwise disjoint perfect matchings.
Hint: use (a)

Metric TSP with Matching

Travelling-Sales-Person Problem

Metric TSP with Matching



Metric TSP with Matching

<u>Metrisches Travelling Salesman Problem</u>

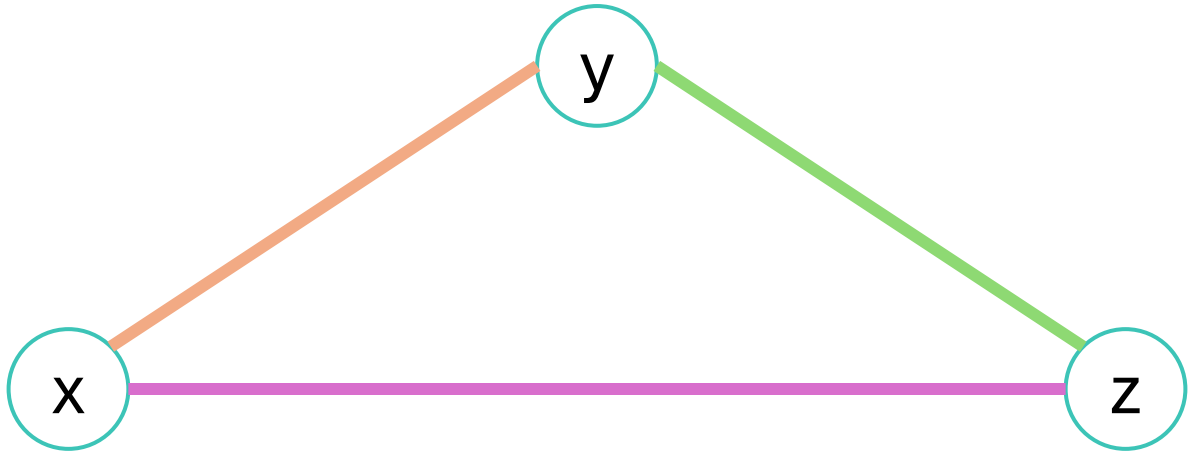
Gegeben: ein vollständiger Graph K_n und eine Funktion $\ell : \binom{[n]}{2} \to \mathbb{N}_0$
mit $\ell(\{x,z\}) \leq \ell(\{x,y\}) + \ell(\{y,z\})$ für alle $x,y,z \in [n]$

Gesucht: ein Hamiltonkreis C in K_n mit

$$\sum_{e \in C} \ell(e) = \min\{\sum_{e \in C'} \ell(e) \mid C' \text{ ist ein Hamiltonkreis in } K_n\}.$$

Metric TSP with Matching

Intuition:



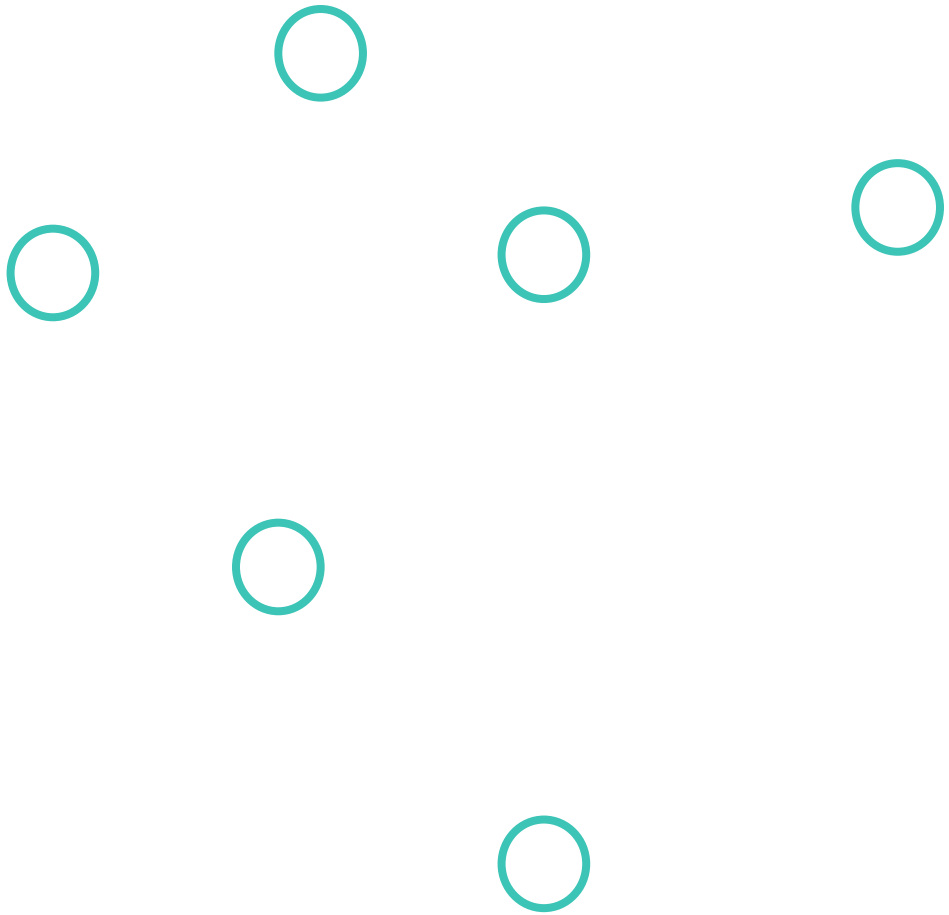
For all x, y, z in [n] with e = {x, z}, f = {x, y}, g = {y, z} the inequality holds:

$$l(\mathbf{e}) \leq l(\mathbf{f}) + l(\mathbf{g})$$

In other words: A “shortcut” can never be longer than an indirect path

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



Goal:

- Find a 1.5-approx. algorithm for the metric TSP using matchings

This means:

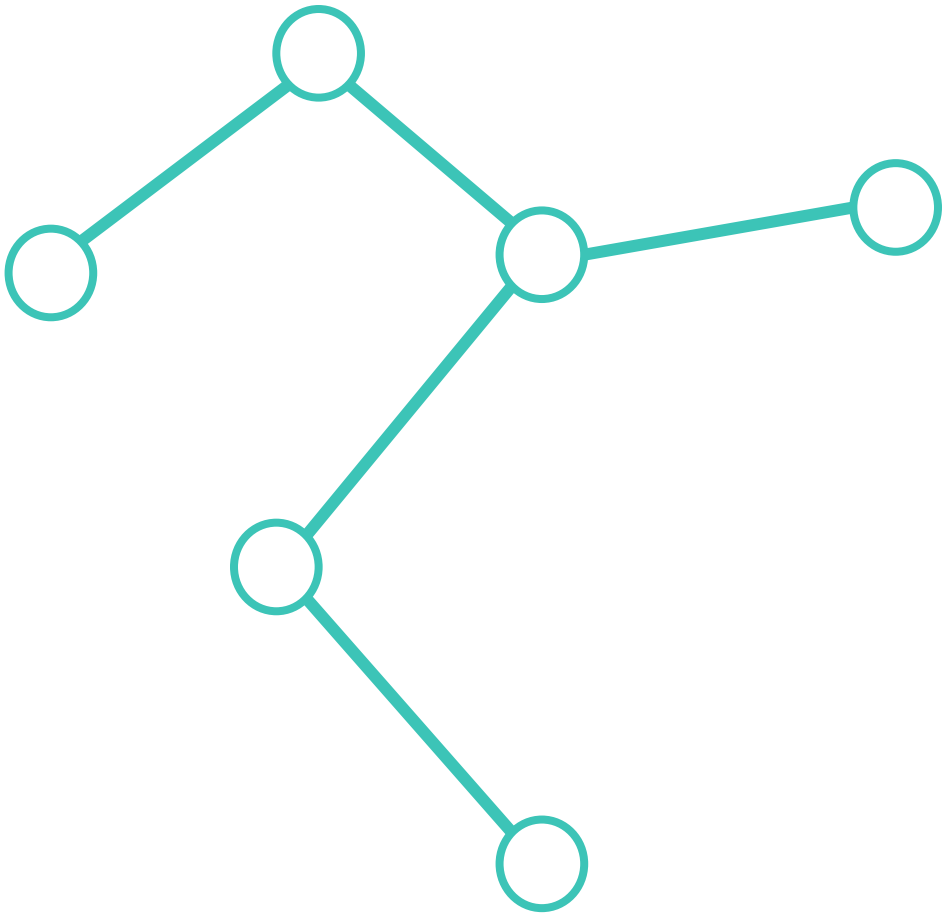
Algorithm has to find Hamiltonian cycle C s.t.

$$l(C) \leq 1.5 * \text{opt}(K_n, l)$$

Better than last times' 2-approx. algorithm!

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$

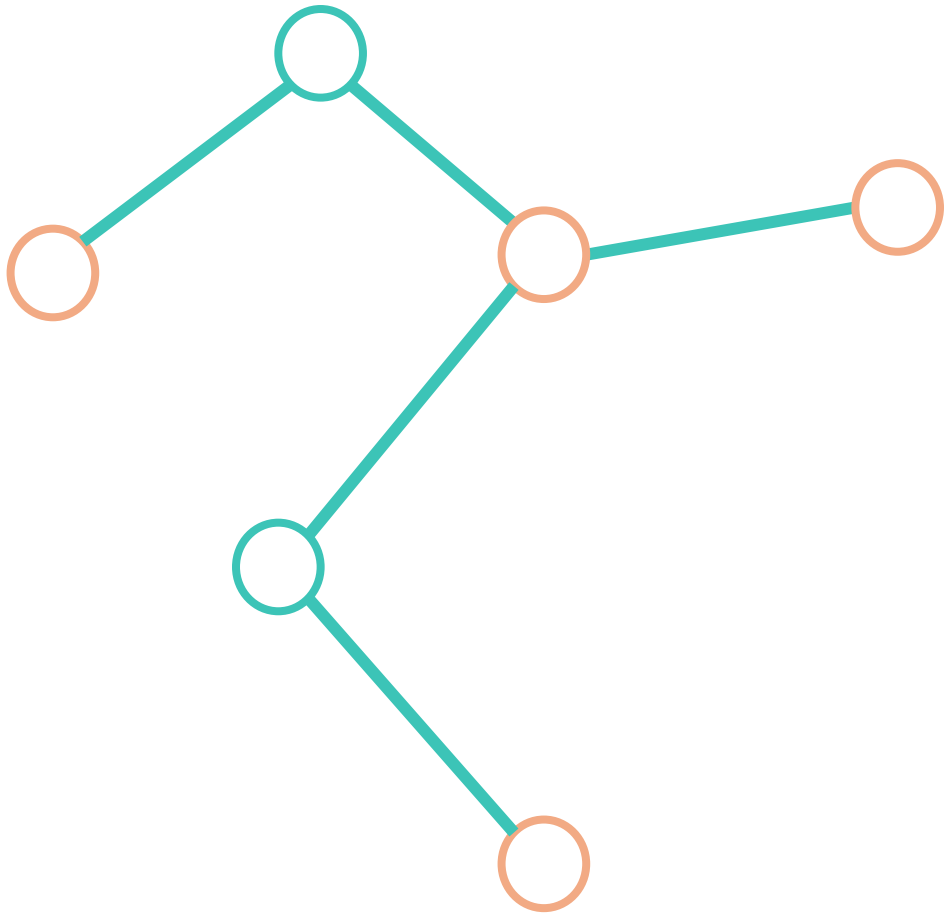


1. Find MST T (Minimal Spanning Tree)

$$l(T) \leq opt(K_n, l)$$

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



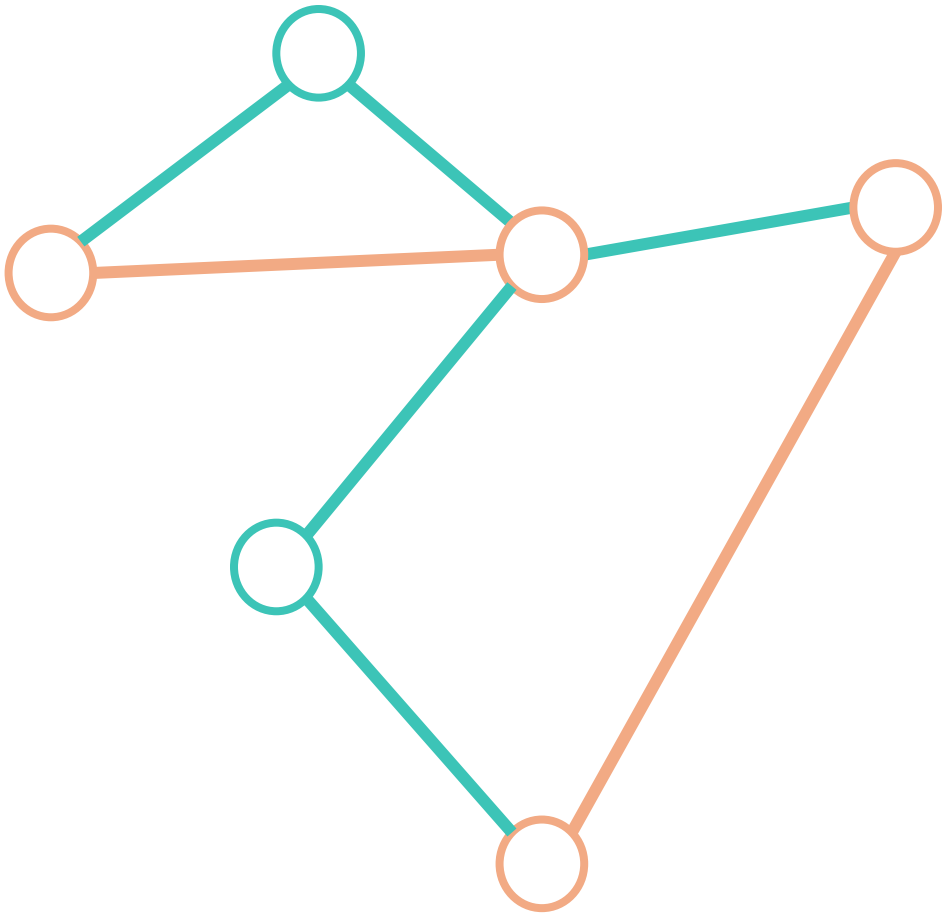
1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

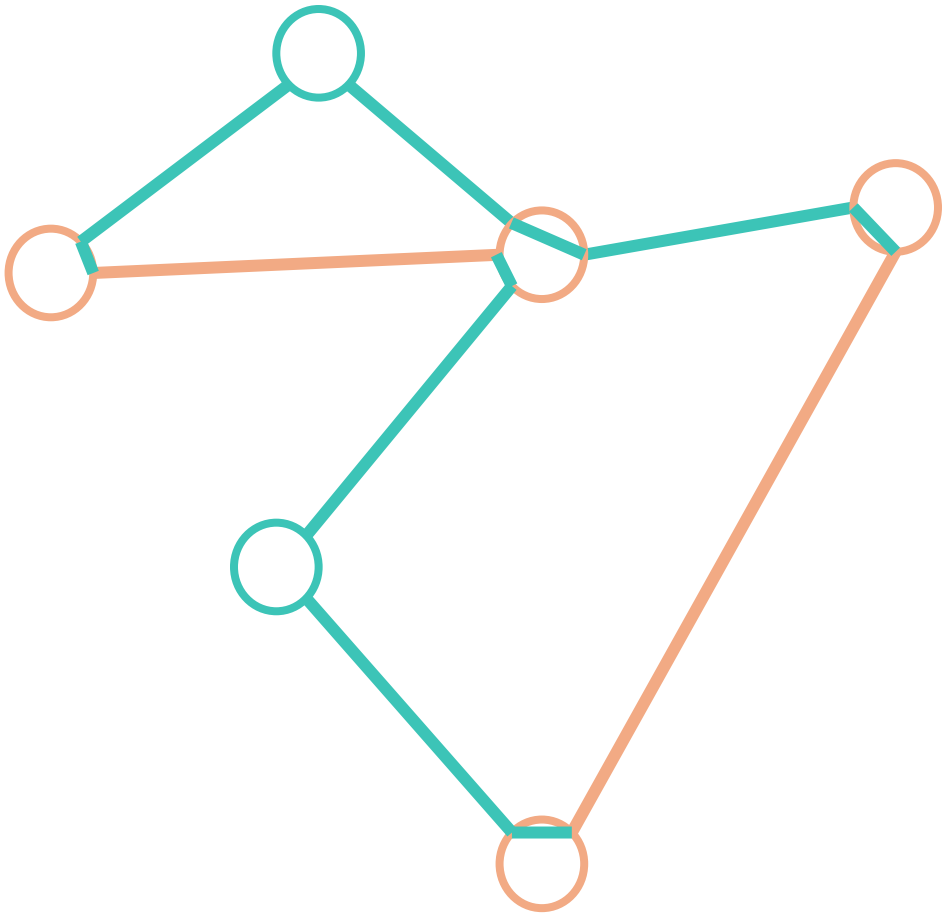
$$l(T) \leq opt(K_n, l)$$

2. X := Vertices with uneven deg. in T
find minimal matching M for X

$$l(M) \leq 0.5 * opt(K_n, l)$$

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

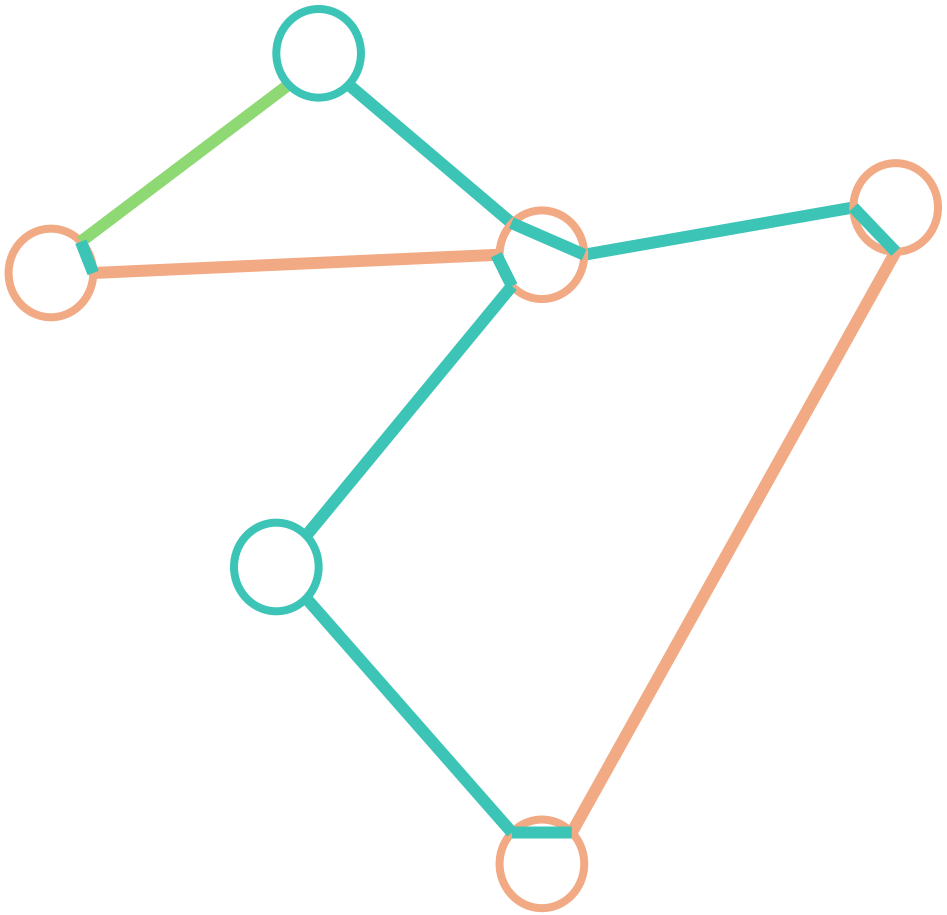
   $l(M) \leq 0.5 * opt(K_n, l)$

3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

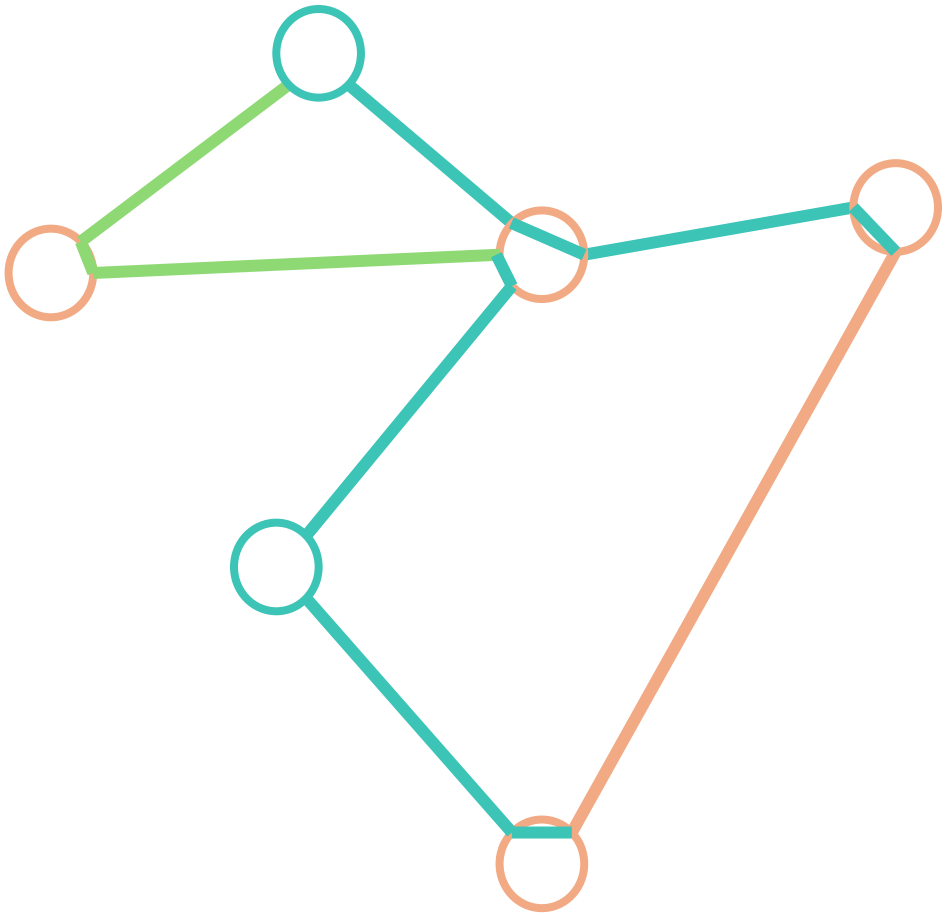
3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

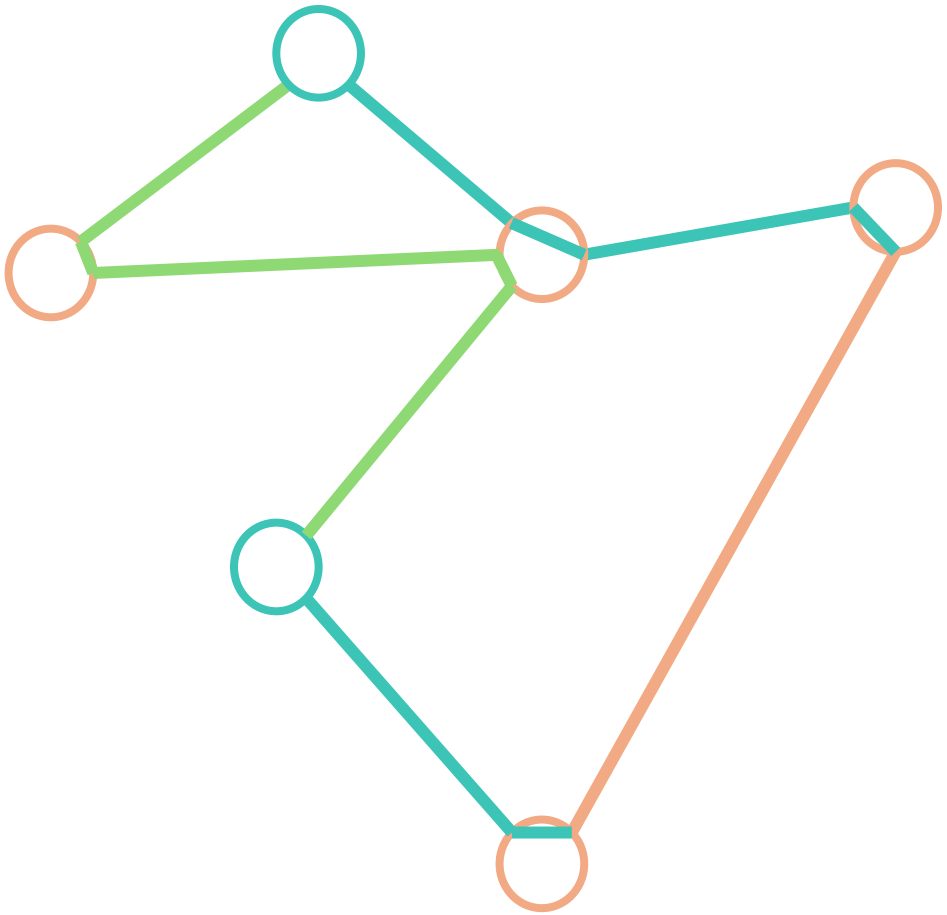
3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

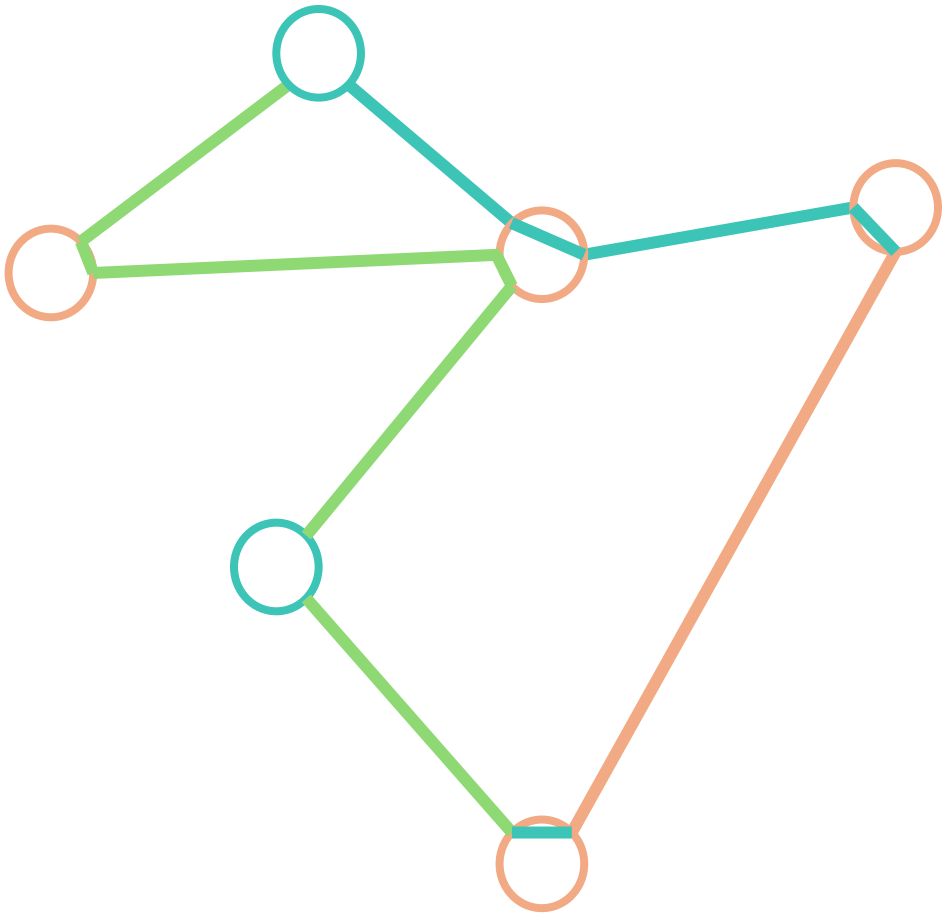
3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

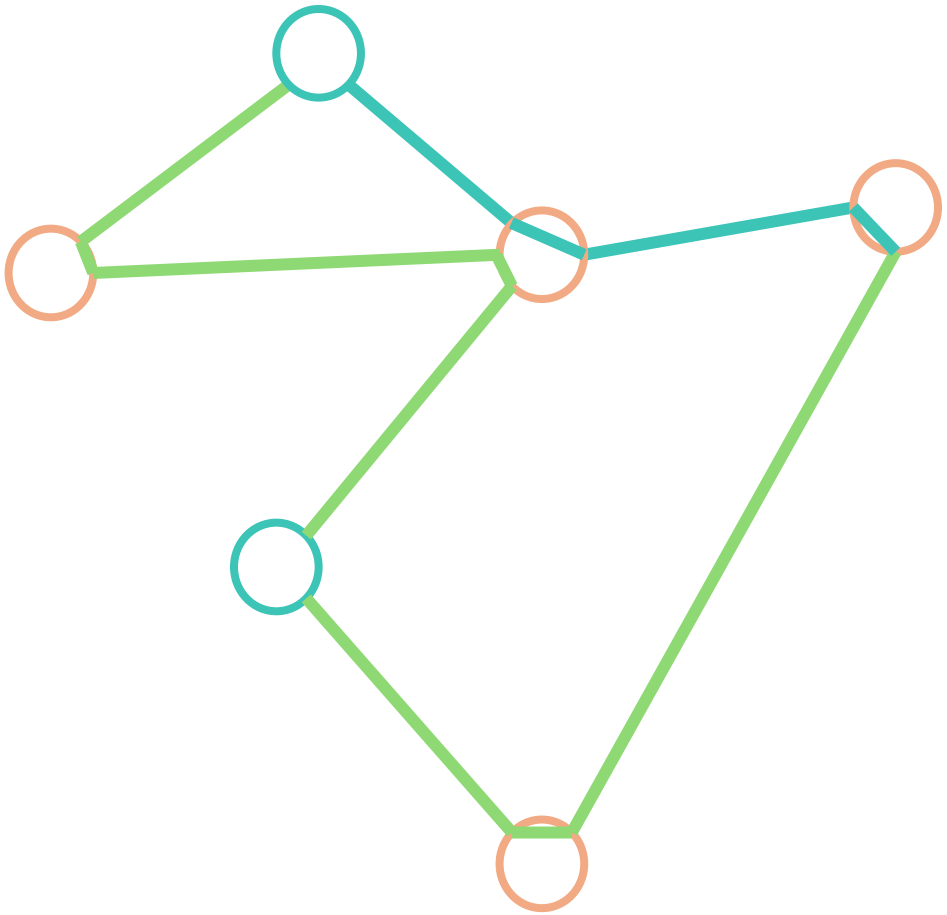
3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

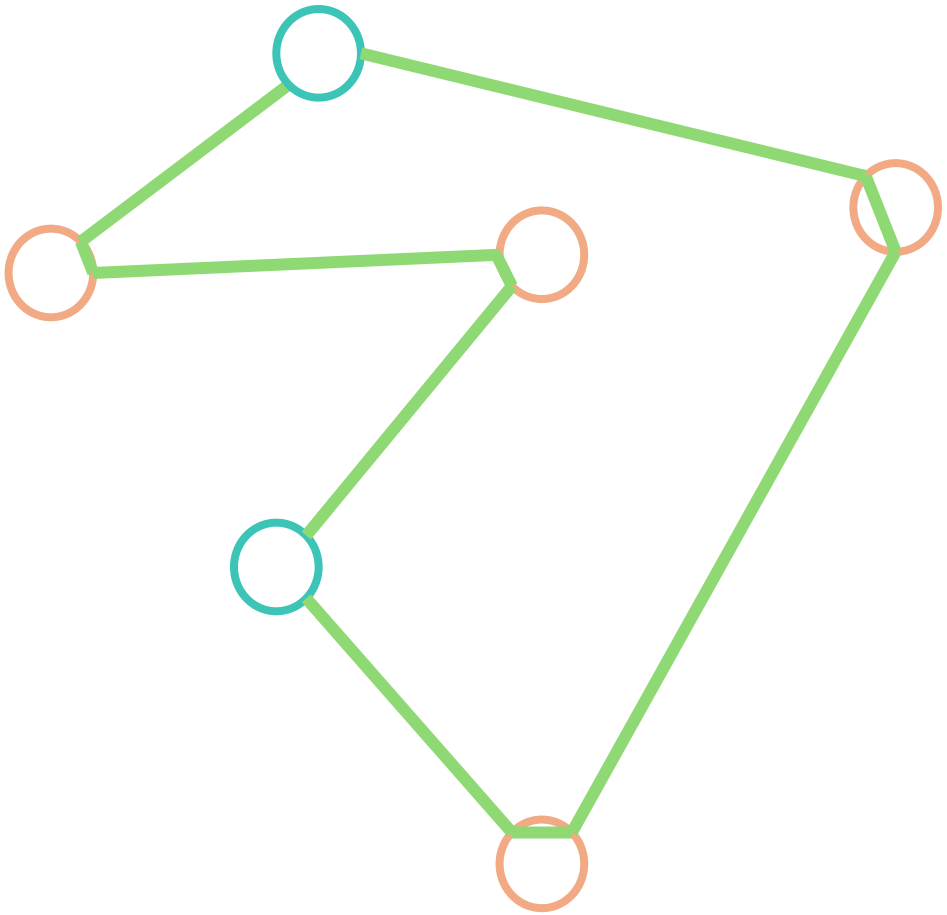
3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

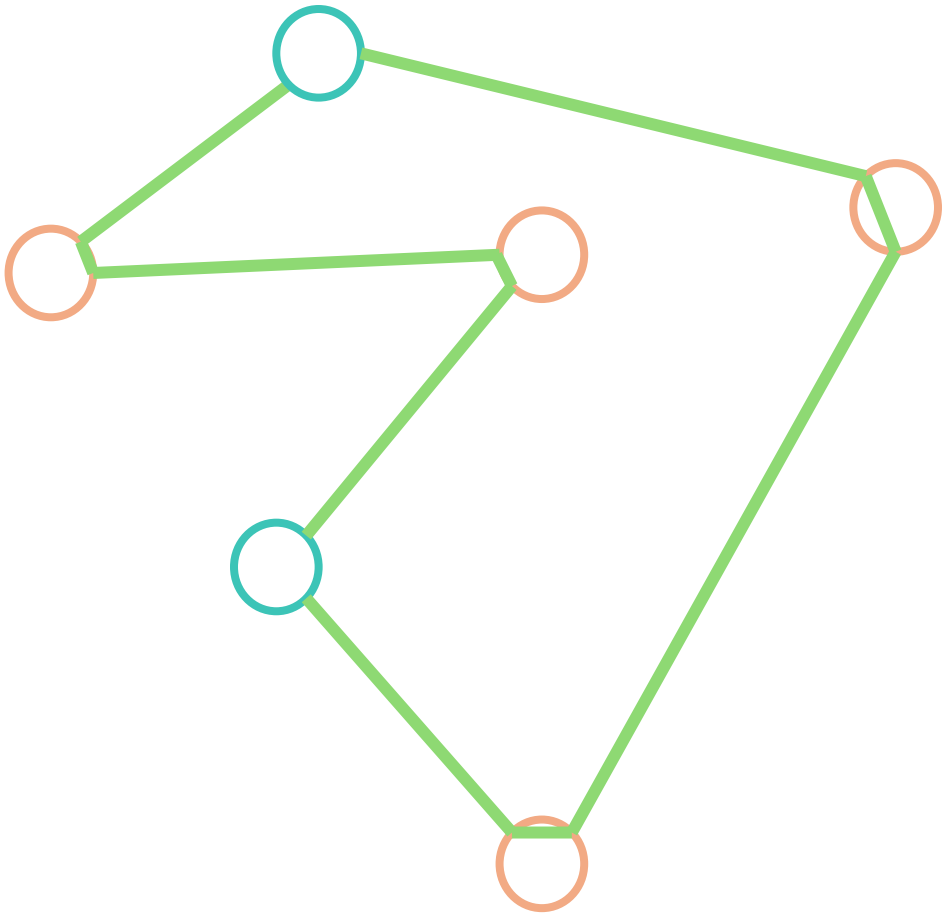
3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

$$l(T) \leq opt(K_n, l)$$

2. X := Vertices with uneven deg. in T
find minimal matching M for X

$$l(M) \leq 0.5 * opt(K_n, l)$$

3. Find Eulerian cycle W using T and M

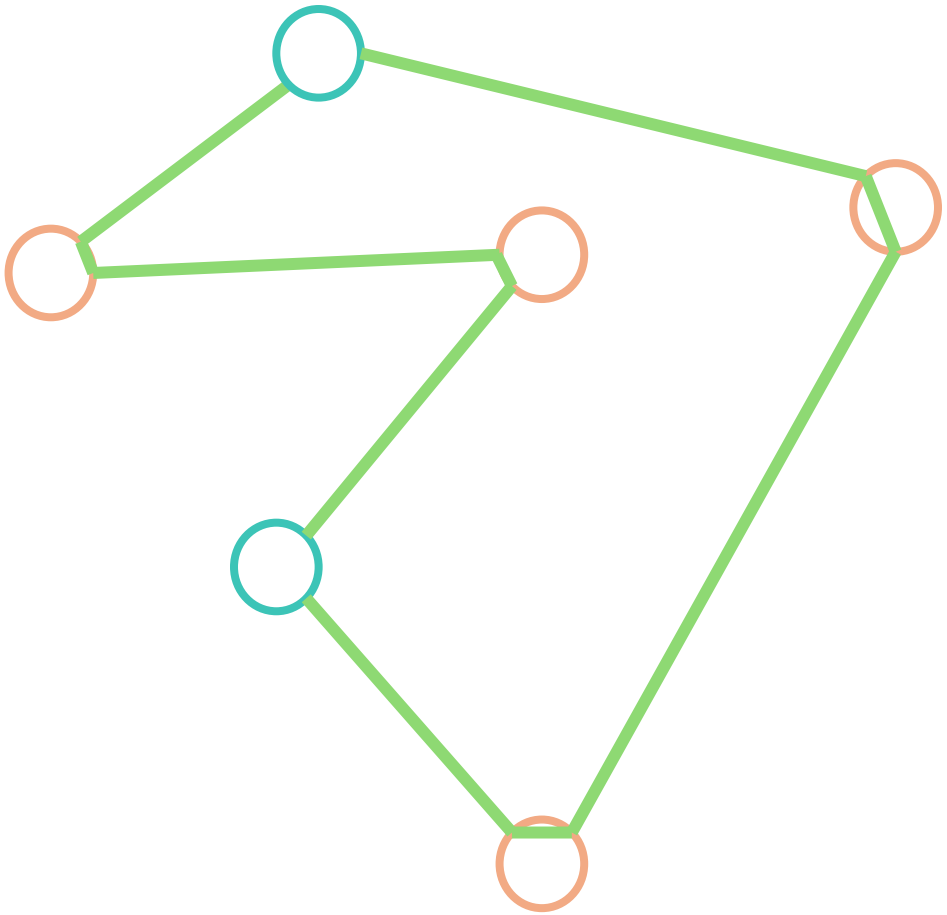
$$l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

$$l(C) \leq l(W) \leq 1.5 * opt(K_n, l)$$

Metric TSP

For all x, y, z in [n] inequality holds: $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$



1. Find MST T (Minimal Spanning Tree)

   $l(T) \leq opt(K_n, l)$

2. X := Vertices with uneven deg. in T
   find minimal matching M for X

   $l(M) \leq 0.5 * opt(K_n, l)$

3. Find Eulerian cycle W using T and M

   $l(W) = l(T) + l(M) \leq 1.5 * opt(K_n, l)$

4. Take shortcuts in W s.t. every node is visited only once → Hamiltonian cycle C

   $l(C) \leq l(W) \leq 1.5 * opt(K_n, l)$

Metric TSP with Matching

Satz 1.43. Für das METRISCHE TRAVELLING SALESMAN PROBLEM gibt es einen 2-Approximationsalgorithmus mit Laufzeit $O(n^2)$.

Satz 1.51. Für das METRISCHE TRAVELLING SALESMAN PROBLEM gibt es einen $3/2$-Approximationsalgorithmus mit Laufzeit $O(n^3)$.

- Finding MST dominates the runtime

Metric TSP with Matching

Satz 1.43. Für das METRISCHE TRAVELLING SALESMAN PROBLEM gibt es einen 2-Approximationsalgorithmus mit Laufzeit $O(n^2)$.

Satz 1.51. Für das METRISCHE TRAVELLING SALESMAN PROBLEM gibt es einen 3/2-Approximationsalgorithmus mit Laufzeit $O(n^3)$.

- Finding ~~MST~~ dominates the runtime

  matching in $O(n^3)$

Colouring

Colouring

Definition 1.56. Eine *(Knoten-)Färbung* (engl. *(vertex) colouring*) eines Graphen $G = (V, E)$ mit k Farben ist eine Abbildung $c\colon V \to [k]$, sodass gilt

$$c(u) \neq c(v) \quad \text{für alle Kanten } \{u, v\} \in E.$$

Die *chromatische Zahl* (engl. *chromatic number*) $\chi(G)$ ist die minimale Anzahl Farben, die für eine Knotenfärbung von G benötigt wird.

Colouring

Definition 1.56. Eine *(Knoten-)Färbung* (engl. *(vertex) colouring*) eines Graphen $G = (V, E)$ mit k Farben ist eine Abbildung $c\colon V \to [k]$, sodass gilt

$$c(u) \neq c(v) \quad \text{für alle Kanten } \{u, v\} \in E.$$

Die *chromatische Zahl* (engl. *chromatic number*) $\chi(G)$ ist die minimale Anzahl Farben, die für eine Knotenfärbung von G benötigt wird.

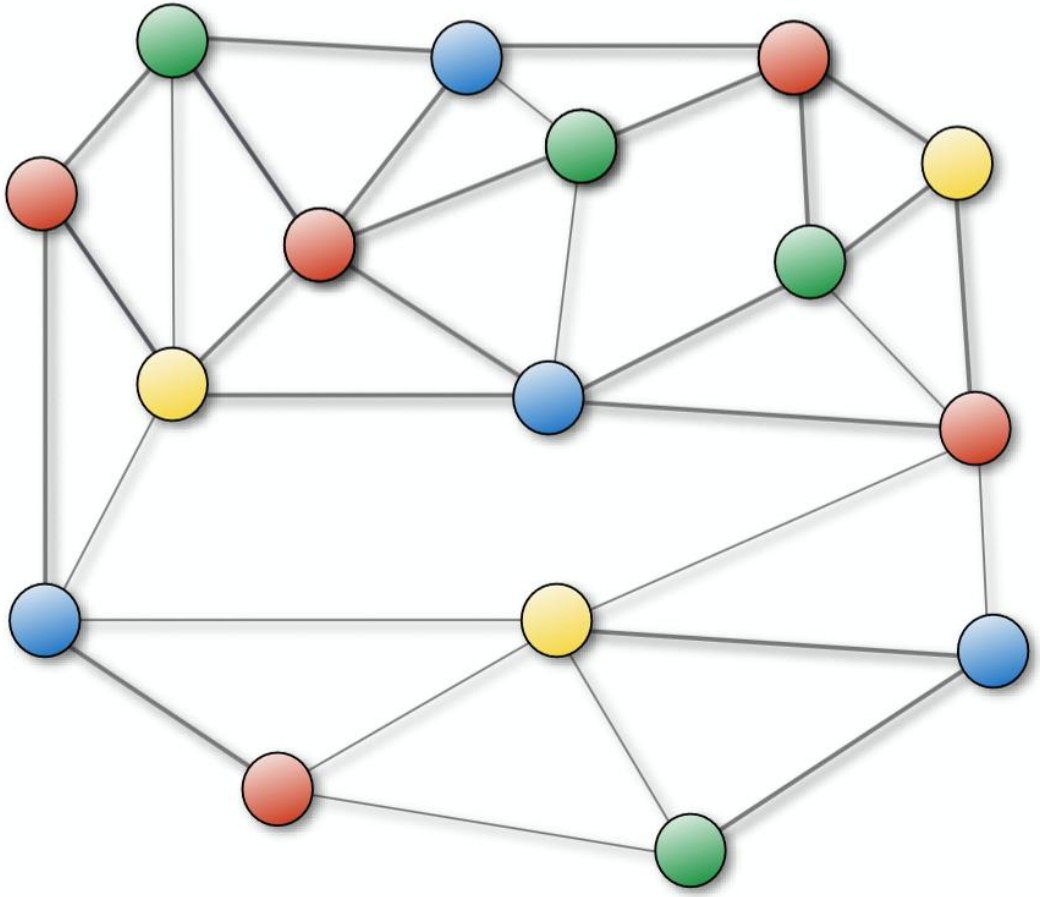
more edges → more problems

contrary to matchings and connectivity

Colouring

more edges → more problems

contrary to matchings and connectivity



Colouring

A k+1-edge-connected graph is also k-edge-connected

- A 4-edge-connected graph is also 3-edge-connected

A k+1-colourable graph is not necessarily colourable in k colours

A k-colourable graph is also k+1-colourable

- A graph that can be coloured using 3 colours can also be coloured using 4 colours

Colouring

Let $G = (V, E)$ be a graph. The *chromatic number* of G, denoted $\chi(G)$, is the smallest integer k such that G admits a proper k-coloring; that is, there exists a function

$$c : V \to \{1, 2, \ldots, k\}$$

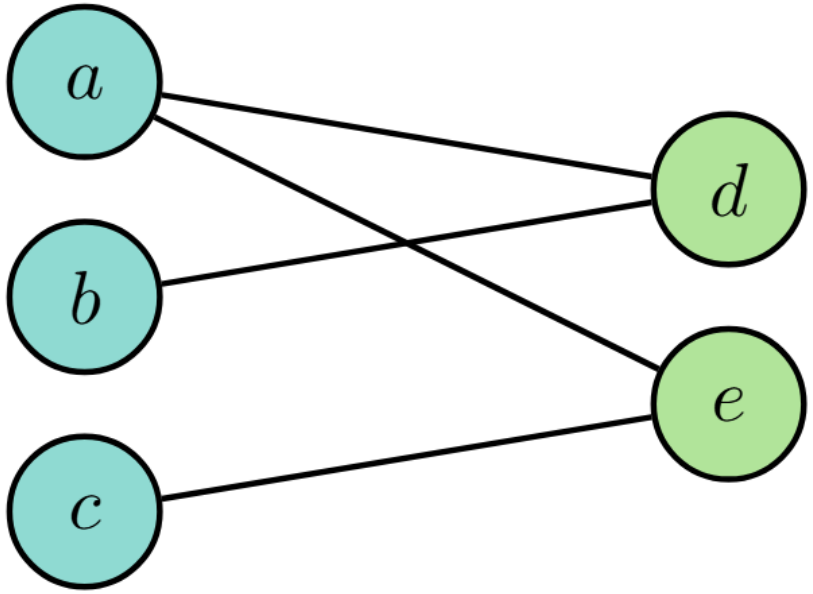
such that for every edge $uv \in E$ we have $c(u) \neq c(v)$.

For a graph G, the following statements are equivalent:

1. $\chi(G) \leq k$.
2. G is k-partite.

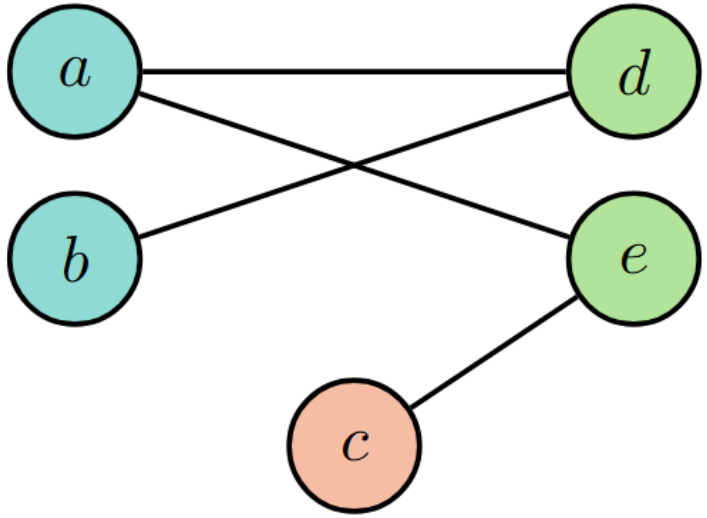
Colouring

Example: A graph is bipartite if and only if it is 2-colourable.



Colouring

A bipartite graph is technically also k-partite for any $k \geq 2$. By moving vertex c to a third set, we illustrate it as a tripartite graph.



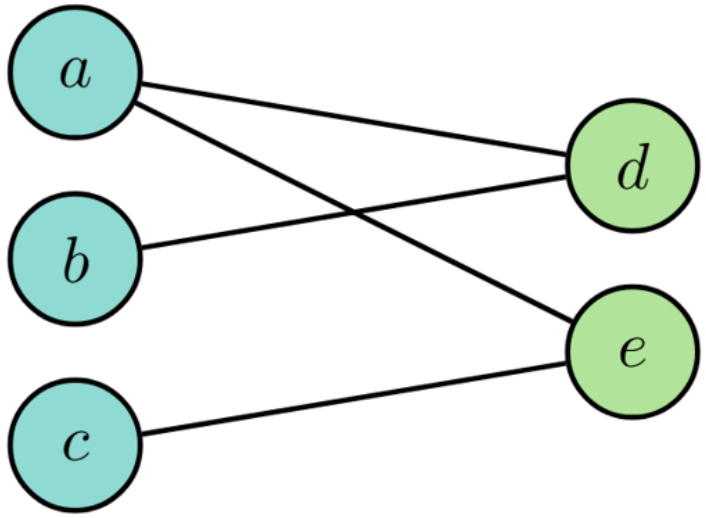
Colouring

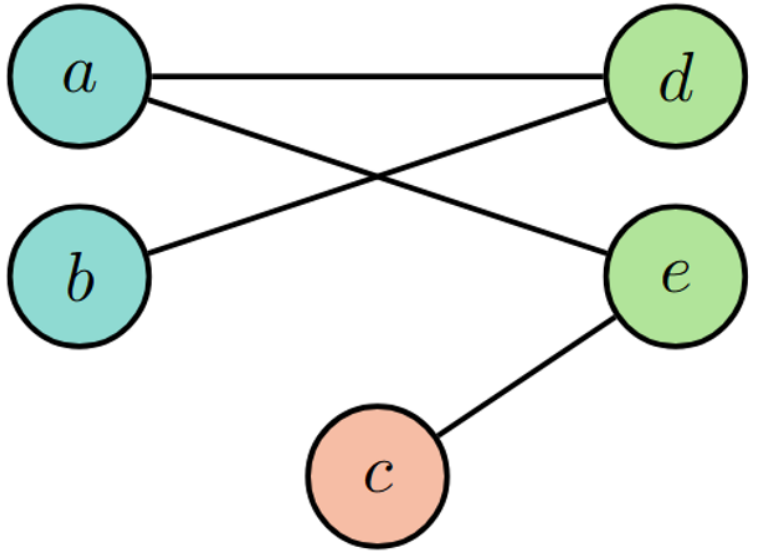
A k-colourable graph is also k+1-colourable

- A graph that can be coloured using 3 colours can also be coloured using 4 colours

$\Leftrightarrow$

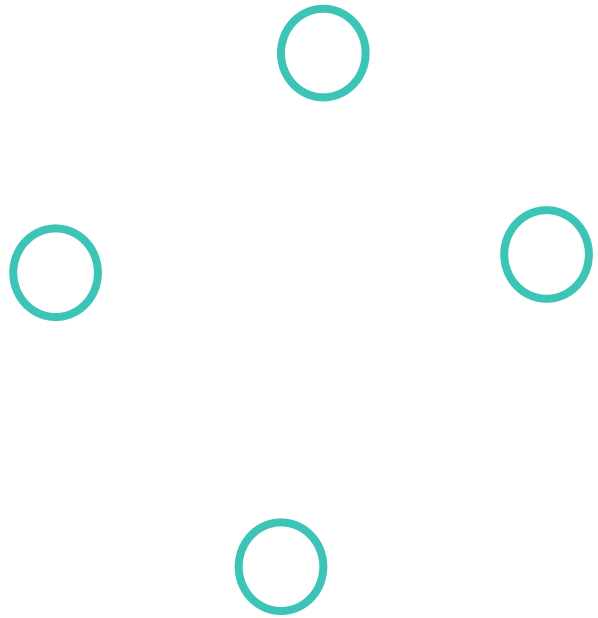
A k-partite graph is also k+1-partite





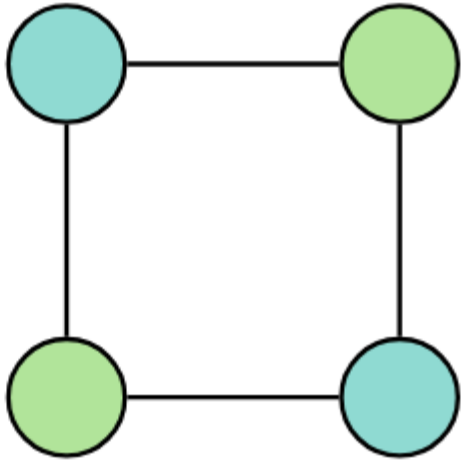
Colouring

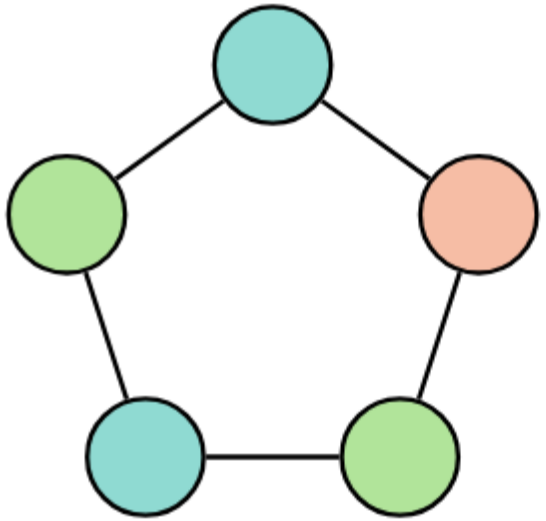
- What is the chromatic number of this graph?
- Is it bipartite?
- Is it tripartite?
- Is it k-partite for $k \geq 1$



Colouring

All **even cycles** are bipartite and therefore 2-colourable. All **odd cycles** (uneven) require 3 colours, since the last vertex will always be adjacent to vertices with different colours.





Satz 1.58. Ein Graph $G = (V, E)$ ist genau dann bipartit, wenn er keinen Kreis ungerader Länge als Teilgraphen enthält.

Colouring

GREEDY-FÄRBUNG (G)

1: wähle eine beliebige Reihenfolge der Knoten: $V = \{v_1, \ldots, v_n\}$
2: $c[v_1] \leftarrow 1$
3: **for** $i = 2$ **to** n **do**
4: $\quad c[v_i] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ für alle } u \in N(v_i) \cap \{v_1, \ldots, v_{i-1}\}\}$

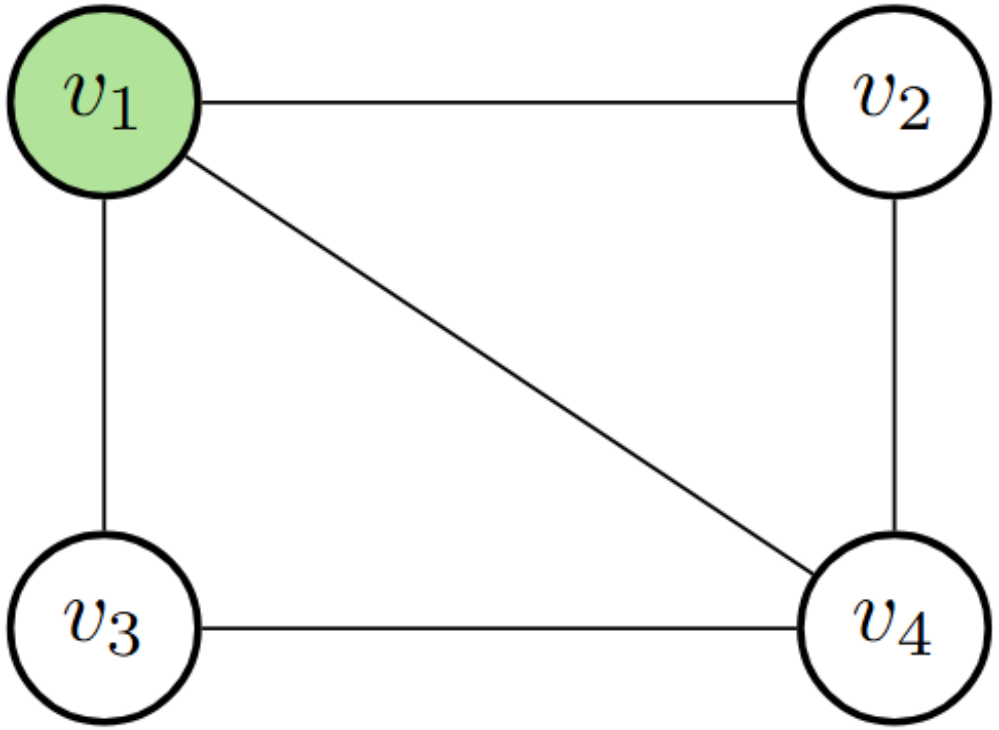
Satz 1.60. Sei G ein zusammenhängender Graph. Für die Anzahl Farben $C(G)$, die der Algorithmus GREEDY-FÄRBUNG benötigt, um die Knoten des Graphen G zu färben, gilt

$$\chi(G) \leq C(G) \leq \Delta(G) + 1.$$

Ist der Graph als Adjazenzliste gespeichert, findet der Algorithmus die Färbung in Zeit $O(|E|)$.

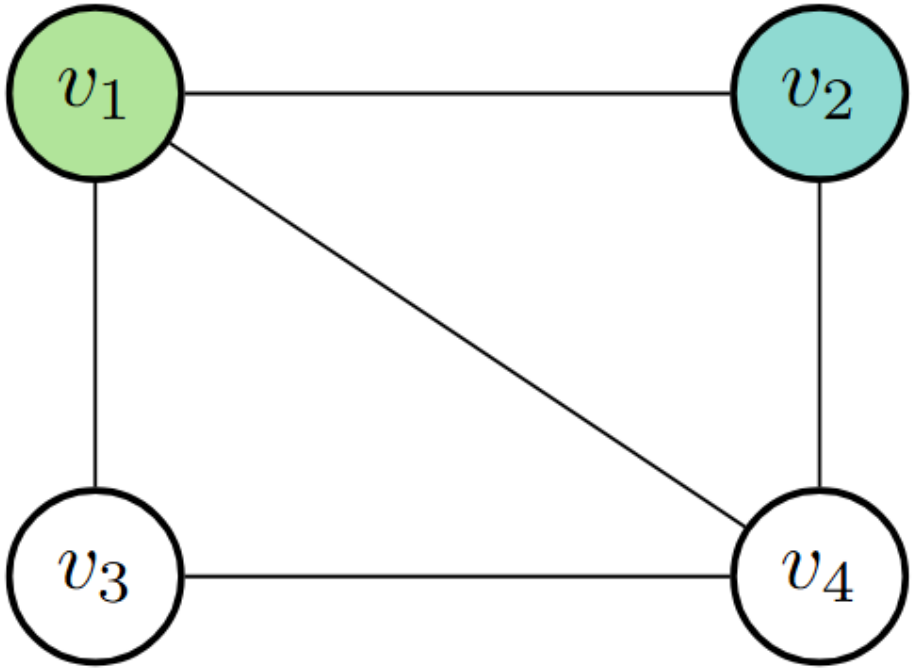
Colouring

Step 1: $c(v_1) \leftarrow 1$ (the first available color, Green)



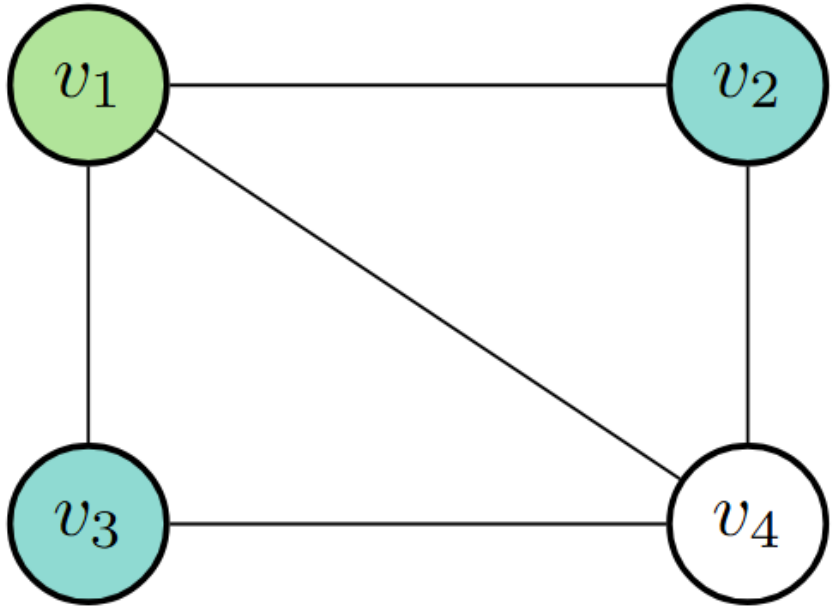
Colouring

Step 2: Color v_2. It has neighbor v_1 $(c(v_1) = 1)$.
Minimal available color for v_2 is $k = 2$ (Blue). $c(v_2) \leftarrow 2$.



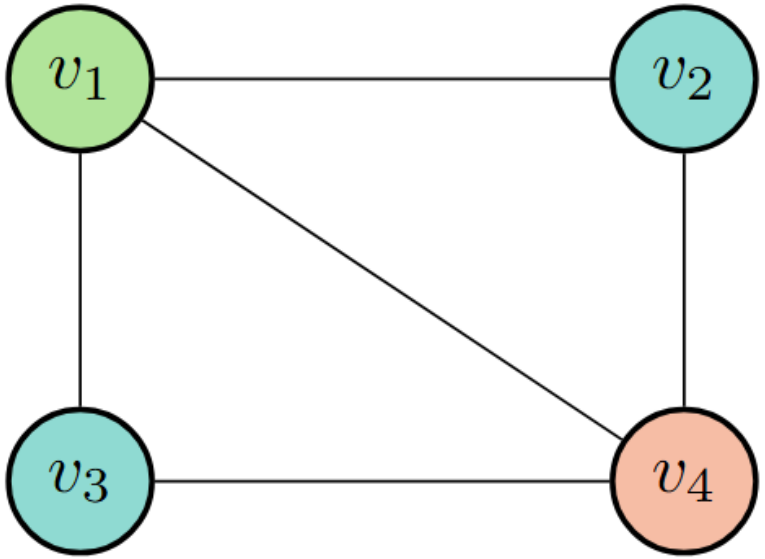
Colouring

Step 3: Color v_3. Neighbors: $\{v_1\}$.
Colors already used by neighbors: $\{c(v_1) = 1\}$.
Minimal available color is $k = 2$ (Blue). $c(v_3) \leftarrow 2$.



Colouring

Step 4: Color v_4. Neighbors: $\{v_1, v_2, v_3\}$.
Colors already used: $\{c(v_1) = 1, c(v_2) = 2, c(v_3) = 2\}$.
Colors $\{1, 2\}$ are blocked. Minimal available color is $k = 3$ (Orange).
$c(v_4) \leftarrow 3$.



Colouring

GREEDY-FÄRBUNG (G)

1: wähle eine beliebige Reihenfolge der Knoten: $V = \{v_1, \ldots, v_n\}$
2: $c[v_1] \leftarrow 1$
3: **for** $i = 2$ **to** n **do**
4: $\quad c[v_i] \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \text{ für alle } u \in N(v_i) \cap \{v_1, \ldots, v_{i-1}\}\}$

Satz 1.60. Sei G ein zusammenhängender Graph. Für die Anzahl Farben $C(G)$, die der Algorithmus GREEDY-FÄRBUNG benötigt, um die Knoten des Graphen G zu färben, gilt

$$\chi(G) \leq C(G) \leq \Delta(G) + 1.$$

Ist der Graph als Adjazenzliste gespeichert, findet der Algorithmus die Färbung in Zeit $O(|E|)$.

Colouring

Satz 1.64 (Satz von Brooks). Ist $G = (V, E)$ ein zusammenhängender Graph, der weder vollständig ist noch ein ungerader Kreis ist, also $G \neq K_n$ und $G \neq C_{2n+1}$, so gilt

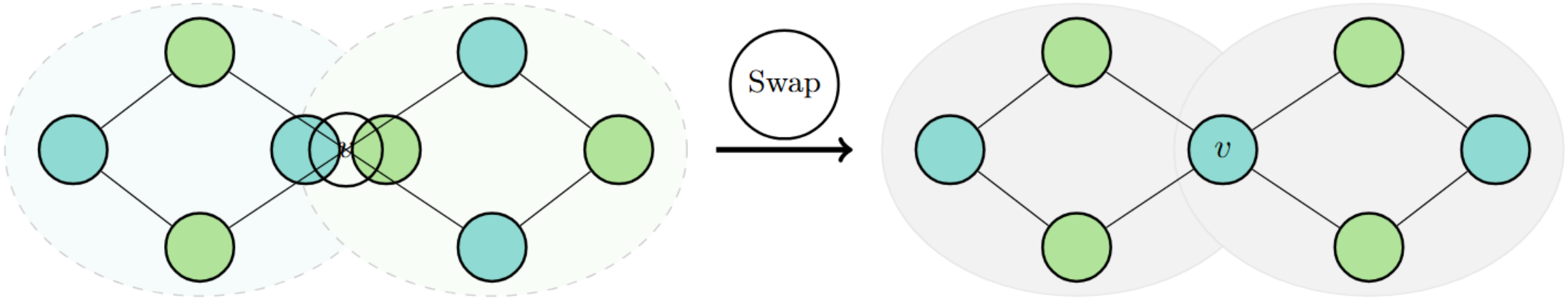
$$\chi(G) \leq \Delta(G)$$

und es gibt einen Algorithmus, der die Knoten des Graphen in Zeit $O(|E|)$ mit $\Delta(G)$ Farben färbt.

Colouring

Application of block-graph:

Every block in G is k-colourable → G is k-colourable



Exercise: Colouring

Exercise 1 – *Graph coloring*

(a) Let $G = (V, E)$ be a graph, and $w \in V$ be a vertex, such that for all $v \in W \setminus \{w\}$ we have $\deg(v) \leq k - 1$. Show that G has a proper coloring into at most k colors.

(b) Let $G = (V, E)$ be a graph, and let $S \subset V$ be a set of all vertices with degree larger than $k - 1$. Show that if $|S| \leq k$, there is a proper coloring of G into at most k colors. Describe an algorithm finding such a coloring in time $O(|V| + |E|)$.

(c) Let $G = (V, E)$ be a graph with $|E| < \binom{k}{2}$. Show that there exist a proper coloring of G into k colors.

Umfrage

A&W G-13