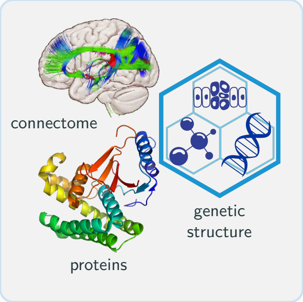
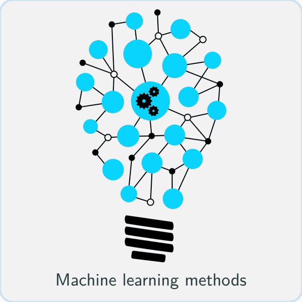
SUSAN: The Structural Similarity Random Walk Kernel

Janis Kalofolias, Pascal Welke, Jilles Vreeken

CISPA HELMHOLTZ-ZENTRUM i. G.
UNIVERSITÄT DES SAARLANDES
max planck institut informatik
UNIVERSITÄT BONN

Comparing graphs

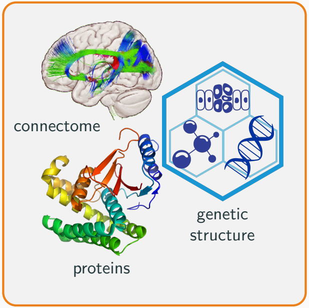


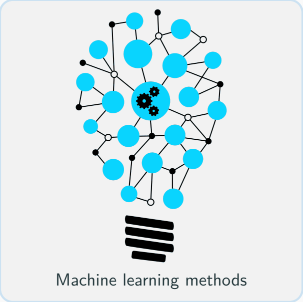


Applications

Standard Tools

Comparing graphs





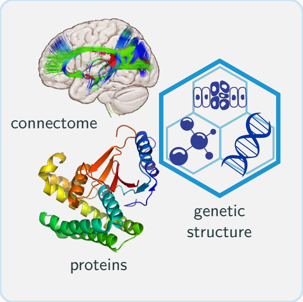
Applications

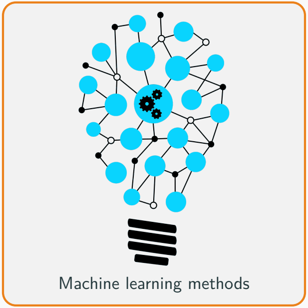
Classification, Regression, Clustering, Dim. Reduction

. . .

Standard Tools

Comparing graphs





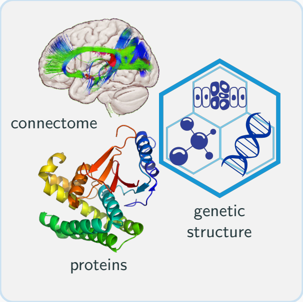
Applications

Classification, Regression,
Clustering, Dim. Reduction
...

Standard Tools

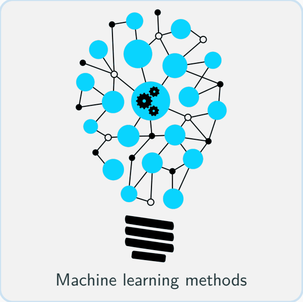
SVM, Logistic,
K-Means, PCR
...

Comparing graphs



Applications

Classification, Regression,
Clustering, Dim. Reduction
...

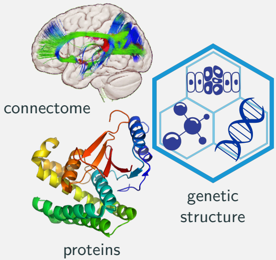


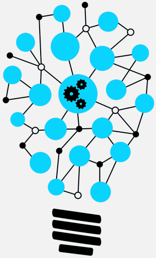
Standard Tools

SVM, Logistic,
K-Means, PCR
...

Can we apply standard tools on graphs?

Comparing graphs





Applications

Classification, Regression,
Clustering, Dim. Reduction
...

Non-vectorial data

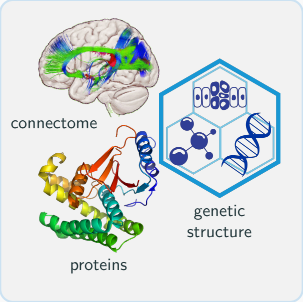
Standard Tools

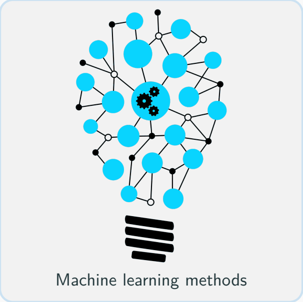
SVM, Logistic,
K-Means, PCR
...

Need vector data

Can we apply standard tools on graphs?

Comparing graphs





Applications

Classification, Regression,
Clustering, Dim. Reduction
...

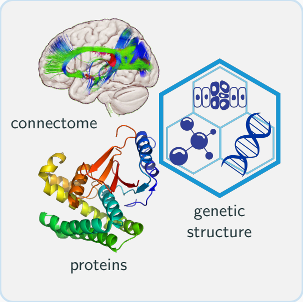
Standard Tools

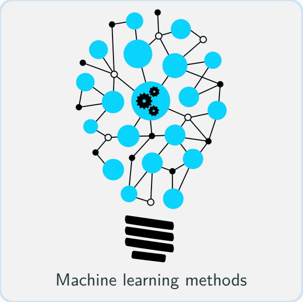
SVM, Logistic,
K-Means, PCR
...

Non-vectorial data ←- - - - - ✗ - - - - -→ Need vector data

Can we apply standard tools on graphs?

Comparing graphs





Applications

Classification, Regression,
Clustering, Dim. Reduction
...

Non-vectorial data

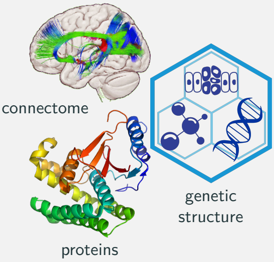
Standard Tools

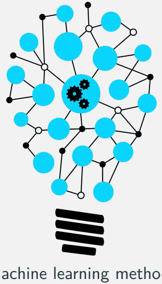
SVM, Logistic,
K-Means, PCR
...

Need vector data

Can we apply standard tools on graphs?

Comparing graphs





Applications

Classification, Regression,
Clustering, Dim. Reduction
...

Non-vectorial data

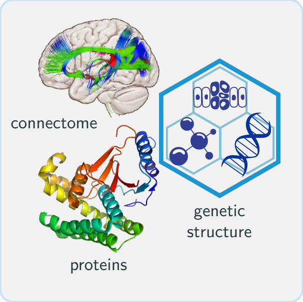
Standard Tools

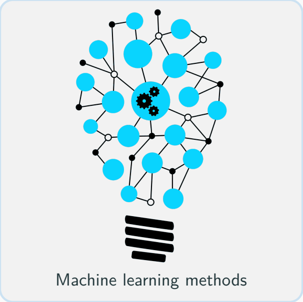
SVM, Logistic,
K-Means, PCR
...

Need vector data

Can we apply standard tools on graphs?

Comparing graphs





Applications

Classification, Regression,
Clustering, Dim. Reduction
...

Non-vectorial data

Standard Tools

SVM, Logistic,
K-Means, PCR
...

Need vector data

Can we apply standard tools on graphs?

$\Longrightarrow$ Use a kernel on graphs

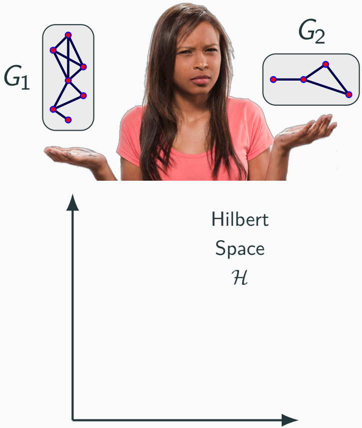
How do kernels compare graphs?

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?

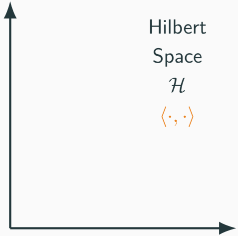


Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

How do kernels compare graphs?

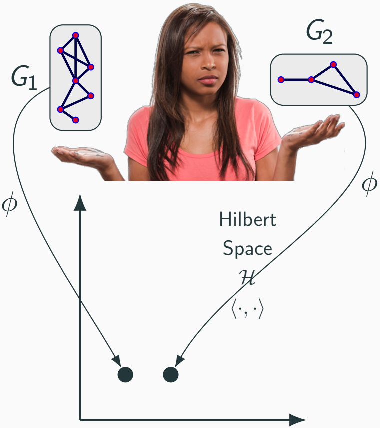
Goal: Can we define something like $\langle G_1, G_2 \rangle$?

Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ



How do kernels compare graphs?

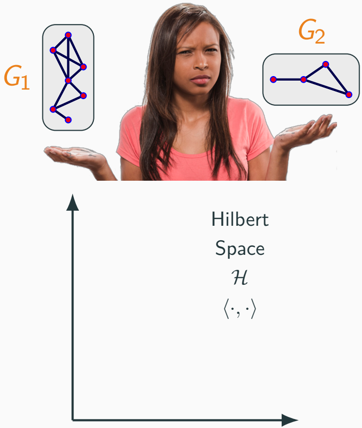
Goal: Can we define something like $\langle G_1, G_2 \rangle$?



Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?



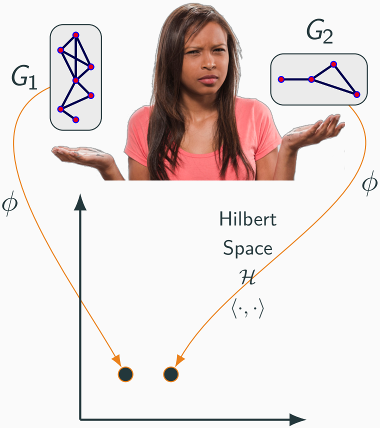
Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

$\Longrightarrow$ Use as graph similarity

G_1 , G_2

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?

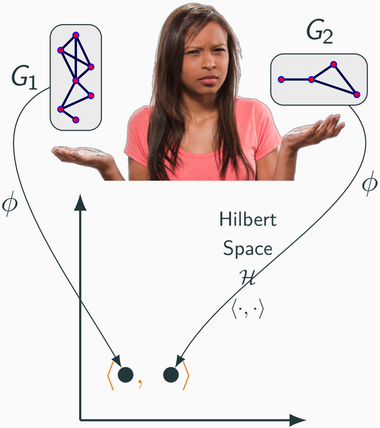


Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

$\Longrightarrow$ Use as graph similarity
$\phi(G_1), \phi(G_2)$

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?

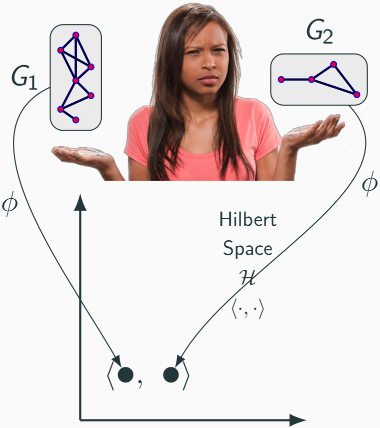


Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

$\Longrightarrow$ Use as graph similarity
$\langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?

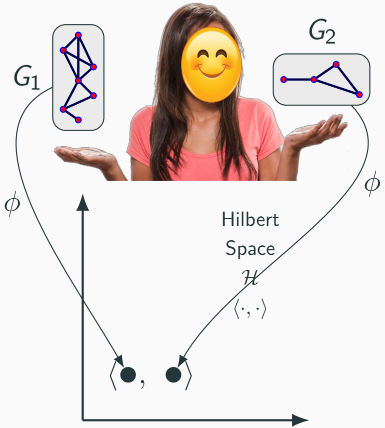


Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

$\Longrightarrow$ Use as graph similarity
$k(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?

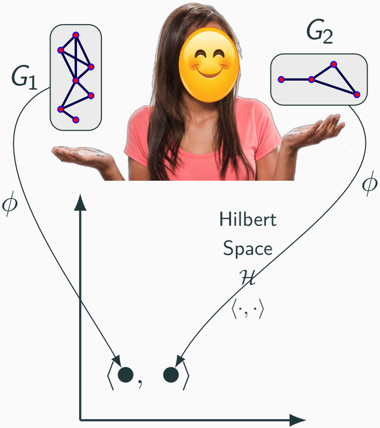


Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

$\Longrightarrow$ Use as graph similarity
$k(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$

How do kernels compare graphs?

Goal: Can we define something like $\langle G_1, G_2 \rangle$?



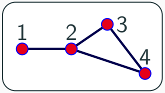
Kernels define a space $\mathcal{H}$
with $\langle \cdot, \cdot \rangle$ and mapping function ϕ

$\Longrightarrow$ Use as graph similarity
$k(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$

We focus on Random Walk kernels

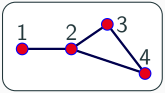
Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

G_1



Goal: Count graph walks

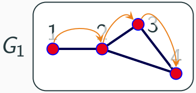
Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

G_1


ex: 3-step walk: $(1, 2, 3, 4)$

Goal: Count graph walks

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

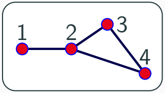


ex: 3-step walk: $(1, 2, 3, 4)$

Goal: Count graph walks

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

G_1



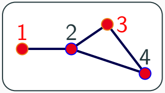
Goal: Count graph walks

1-step walks from 1, 3?

$$\underbrace{\begin{bmatrix}0\\2\\0\\1\end{bmatrix}}_{x_1} = \underbrace{\begin{bmatrix}0&1&0&0\\1&0&1&1\\0&1&0&1\\0&1&1&0\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1\\0\\1\\0\end{bmatrix}}_{x_0}$$

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

G_1

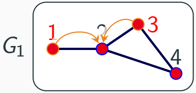


1-step walks from 1, 3?

$$\underbrace{\begin{bmatrix}0\\2\\0\\1\end{bmatrix}}_{x_1} = \underbrace{\begin{bmatrix}0&1&0&0\\1&0&1&1\\0&1&0&1\\0&1&1&0\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1\\0\\1\\0\end{bmatrix}}_{x_0}$$

Goal: Count graph walks

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

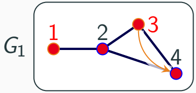


Goal: Count graph walks

\# 1-step walks from $1, 3$?

$$\underbrace{\begin{bmatrix}0\\2\\0\\1\end{bmatrix}}_{x_1} = \underbrace{\begin{bmatrix}0&1&0&0\\1&0&1&1\\0&1&0&1\\0&1&1&0\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1\\0\\1\\0\end{bmatrix}}_{x_0}$$

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

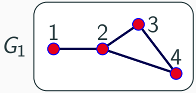


Goal: Count graph walks

1-step walks from 1, 3?

$$\underbrace{\begin{bmatrix}0\\2\\0\\1\end{bmatrix}}_{x_1} = \underbrace{\begin{bmatrix}0&1&0&0\\1&0&1&1\\0&1&0&1\\0&1&1&0\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1\\0\\1\\0\end{bmatrix}}_{x_0}$$

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



1-step walks from $1, 3$?

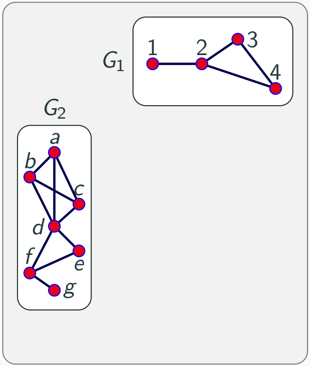
$$\underbrace{\begin{bmatrix}0\\2\\0\\1\end{bmatrix}}_{x_1} = \underbrace{\begin{bmatrix}0&1&0&0\\1&0&1&1\\0&1&0&1\\0&1&1&0\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1\\0\\1\\0\end{bmatrix}}_{x_0}$$

#k-step walks from x_0?

$$x_k = A^k x_0$$

Goal: Count graph walks

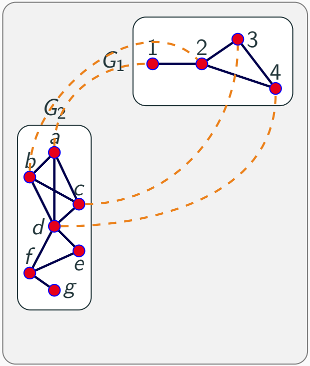
Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Goal: Count graph walks

But: in 2 graphs?

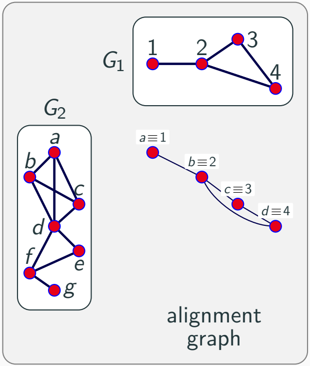
Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$

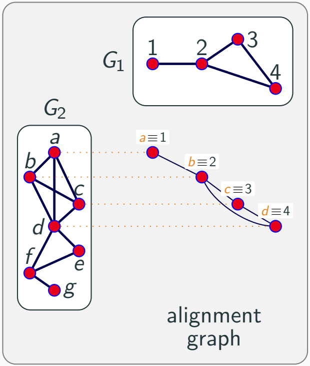


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

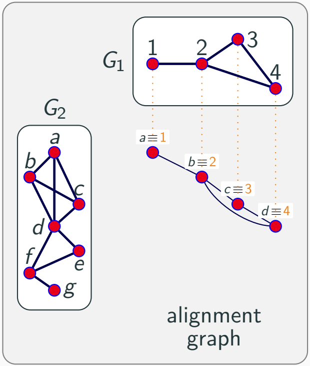


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

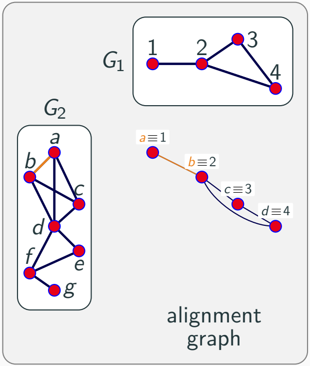


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

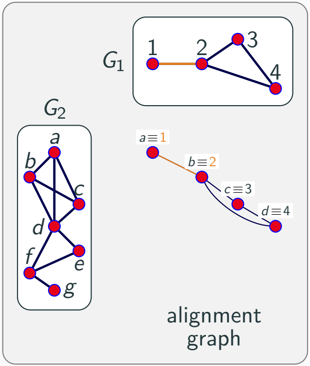


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

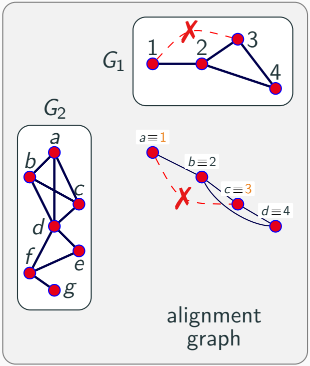


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

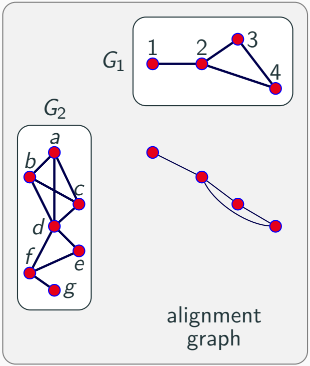


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

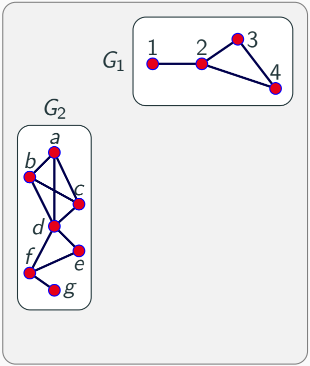


Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2$, $c \equiv 3$, $d \equiv 4$, $b \equiv 2$

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



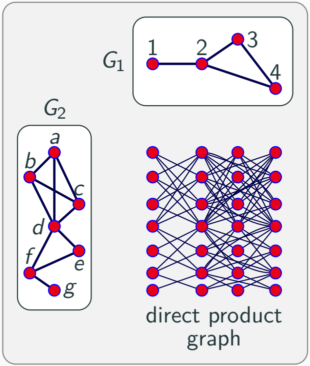
 **Goal**: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2, c \equiv 3, d \equiv 4, b \equiv 2$

But: Alignments are rarely available

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



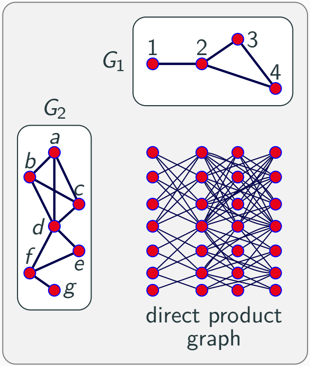
Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1$, $b \equiv 2$, $c \equiv 3$, $d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2$, $c \equiv 3$, $d \equiv 4$, $b \equiv 2$

But: Alignments are rarely available

$\Longrightarrow$ Use all possible alignments



Direct product graph:

$$A_\times = A \otimes A'$$

Goal: Count common walks

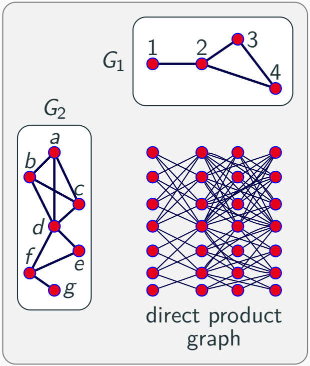
But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2, c \equiv 3, d \equiv 4, b \equiv 2$

But: Alignments are rarely available

$\Longrightarrow$ Use all possible alignments

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Direct product graph:

$$A_\times = A \otimes A'$$

$$A_\times x_\times = (Ax) \otimes (A'x')$$

Goal: Count common walks

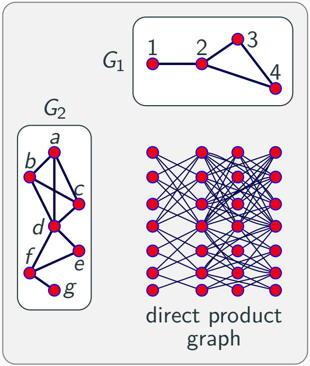
But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2, c \equiv 3, d \equiv 4, b \equiv 2$

But: Alignments are rarely available

$\Longrightarrow$ Use all possible alignments

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Direct product graph:

$$A_\times = A \otimes A'$$

$$A_\times x_\times = (Ax) \otimes (A'x')$$

Goal: Count common walks

But: in 2 graphs?

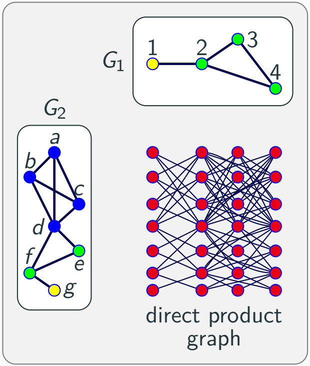
- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2, c \equiv 3, d \equiv 4, b \equiv 2$

But: Alignments are rarely available

$\Longrightarrow$ Use all possible alignments

But: If vertices are not similar?

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Direct product graph:

$$A_\times = A \otimes A'$$

$$A_\times x_\times = (Ax) \otimes (A'x')$$

Goal: Count common walks

But: in 2 graphs?

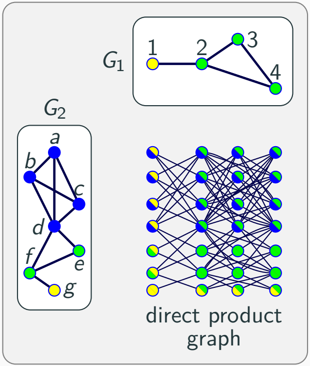
- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2, c \equiv 3, d \equiv 4, b \equiv 2$

But: Alignments are rarely available

$\Longrightarrow$ Use all possible alignments

But: If vertices are not similar?

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Direct product graph:

$$A_\times = A \otimes A'$$

$$A_\times x_\times = (Ax) \otimes (A'x')$$

Goal: Count common walks

But: in 2 graphs?

- Assume vertex alignment
  e.g.: $a \equiv 1, b \equiv 2, c \equiv 3, d \equiv 4$
- Create alignment graph
- Walk in alignment graph
  e.g.: $b \equiv 2, c \equiv 3, d \equiv 4, b \equiv 2$

But: Alignments are rarely available

$\Longrightarrow$ Use all possible alignments

But: If vertices are not similar?

$\Longrightarrow$ Not all alignments equally good

Are all vertex alignments equally good?

- Dissimilar vertices can be noisy
- Do not contribute to similarity

Are all vertex alignments equally good?

- Dissimilar vertices can be noisy
- Do not contribute to similarity

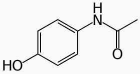
$\implies$ Only match similar vertices

Are all vertex alignments equally good?

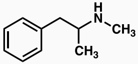
- Dissimilar vertices can be noisy
- Do not contribute to similarity

$\Longrightarrow$ Only match similar vertices

Labeled vertices



vs



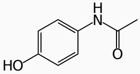
✓ same label $\Rightarrow$ similar vertices

Are all vertex alignments equally good?

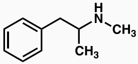
- Dissimilar vertices can be noisy
- Do not contribute to similarity

$\Longrightarrow$ Only match similar vertices

Labeled vertices



vs



✓ same label $\Rightarrow$ similar vertices

✗ G_2 has no O. What now?

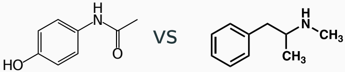
✗ How close is C to H?

Are all vertex alignments equally good?

- Dissimilar vertices can be noisy
- Do not contribute to similarity

$\Longrightarrow$ Only match similar vertices

Labeled vertices



✓ same label $\Rightarrow$ similar vertices

✗ G_2 has no O. What now?

✗ How close is C to H?

Unlabeled graphs

✓ many similarity measures

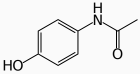
✗ not always clear or easy

Are all vertex alignments equally good?

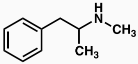
- Dissimilar vertices can be noisy
- Do not contribute to similarity

$\Longrightarrow$ Only match similar vertices

Labeled vertices



vs



✓ same label $\Rightarrow$ similar vertices

✗ G_2 has no O. What now?

✗ How close is C to H?

Unlabeled graphs

✓ many similarity measures

✗ not always clear or easy

We seek a vertex partitioning

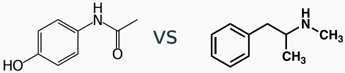
- structurally aware
- efficient to compute
- defines partition similarity

Are all vertex alignments equally good?

- Dissimilar vertices can be noisy
- Do not contribute to similarity

$\Longrightarrow$ Only match similar vertices

Labeled vertices



✓ same label $\Rightarrow$ similar vertices

✗ G_2 has no O. What now?

✗ How close is C to H?

Unlabeled graphs

✓ many similarity measures

✗ not always clear or easy

We seek a vertex partitioning

- structurally aware
- efficient to compute
- defines partition similarity

We propose to use

$\Longrightarrow$ core decomposition

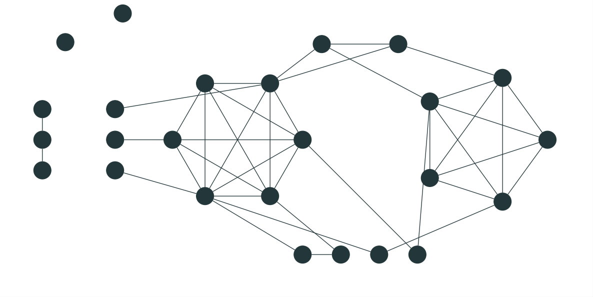
Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

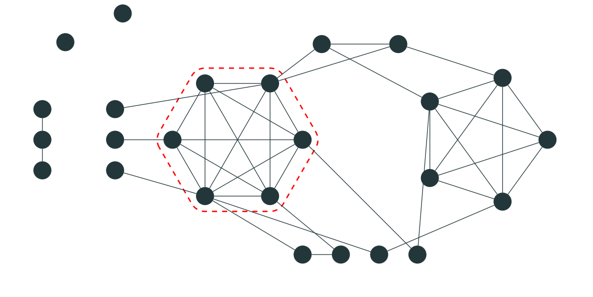
Example



Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

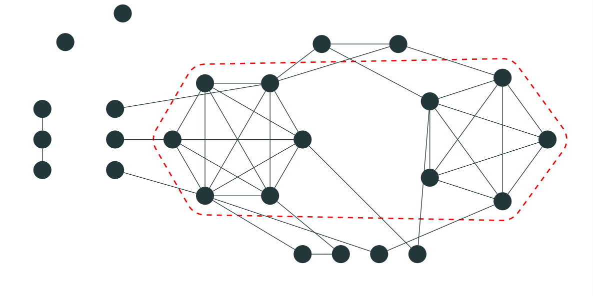
Example



Core decomposition

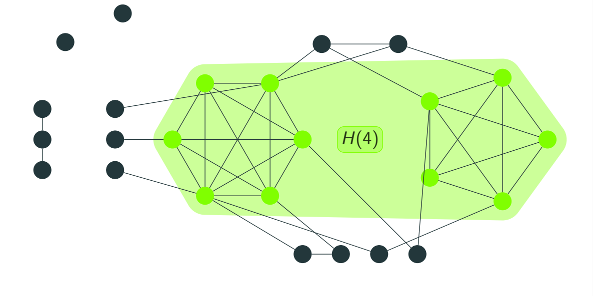
Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Example



Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

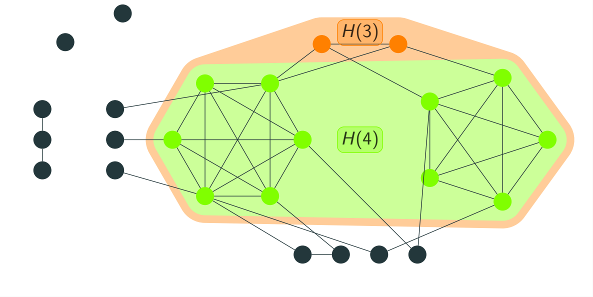
Example



Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

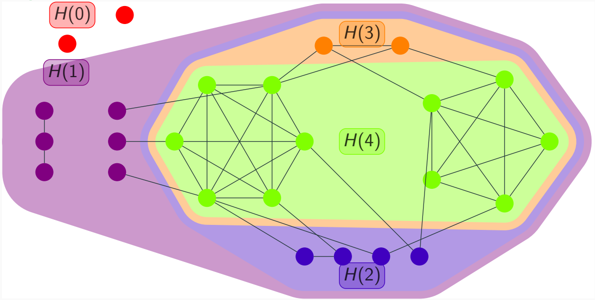
Example



Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

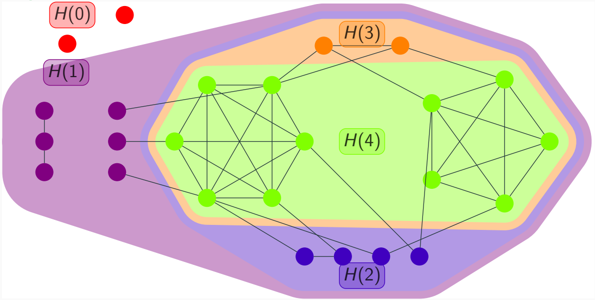
Example



Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Example

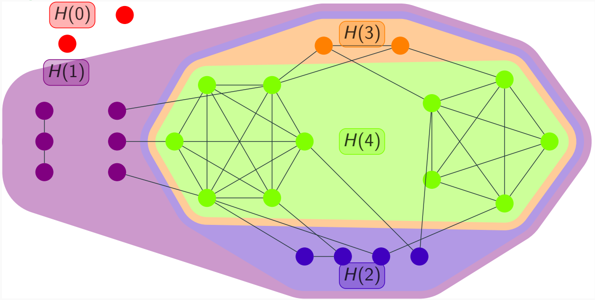


Decomposition: $\kappa : V \to \mathbb{N}$

Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Example



Decomposition: $\kappa : V \to \mathbb{N}$

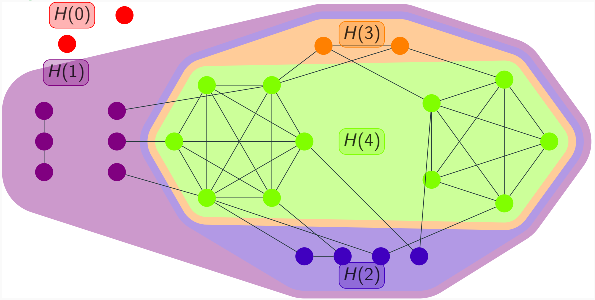
Definition (vertex coreness)

$$\kappa(u) := \max_{u \in H(k)} k$$

Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Example



Definition (vertex coreness)

$$\kappa(u) := \max_{u \in H(k)} k$$

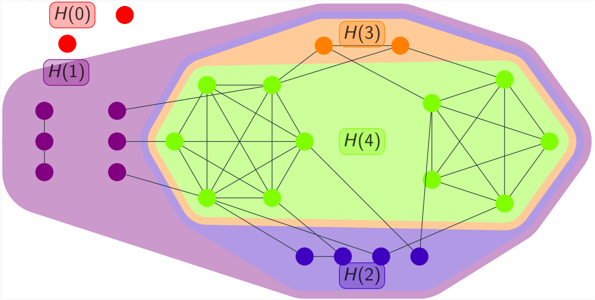
Decomposition: $\kappa : V \to \mathbb{N}$

- k-core vertices have similar structure [Shin et al., 2016]

Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Example



Definition (vertex coreness)

$$\kappa(u) := \max_{u \in H(k)} k$$

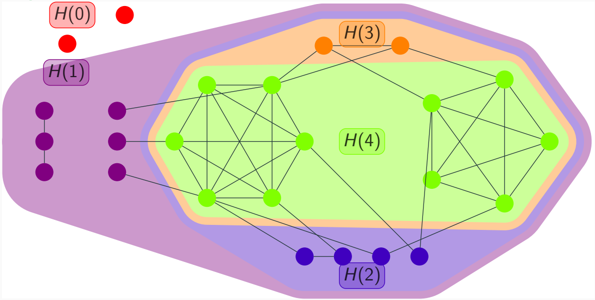
Decomposition: $\kappa : V \to \mathbb{N}$

- k-core vertices have similar structure [Shin et al., 2016]
- Needs only $O(n)$. [Batagelj and Zaversnik, 2003]

Core decomposition

Definition (k-core of graph G)
A maximal subgraph with vertices of degree at least k.

Example



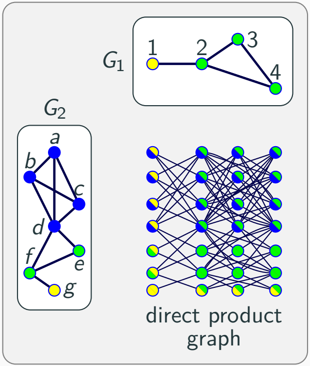
Definition (vertex coreness)

$$\kappa(u) := \max_{u \in H(k)} k$$

Decomposition: $\kappa : V \to \mathbb{N}$

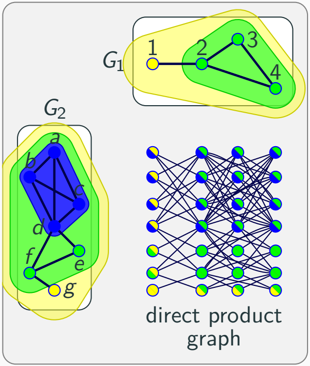
- k-core vertices have similar structure [Shin et al., 2016]
- Needs only $O(n)$. [Batagelj and Zaversnik, 2003]
- Intuitive comparison between labels

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Goal: Count similar walks

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

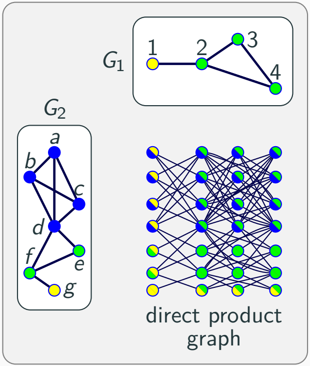


Goal: Count similar walks

Use core values as integer labels

and/or existing labels

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



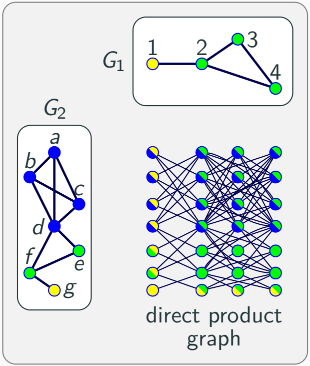
Goal: Count similar walks

Use core values as integer labels

and/or existing labels

close integers $\iff$ similar structure

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Goal: Count similar walks

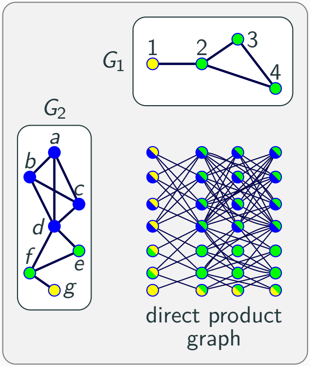
Use core values as integer labels

and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

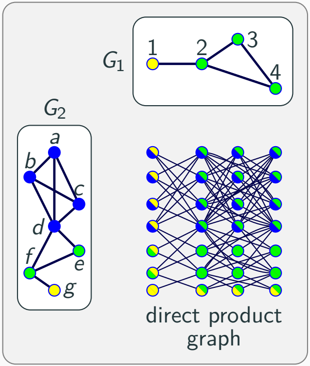
Use core values as integer labels

and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

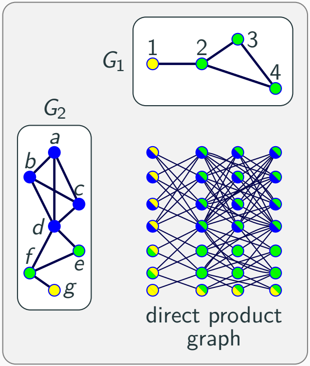
and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

close integers $\iff$ similar structure

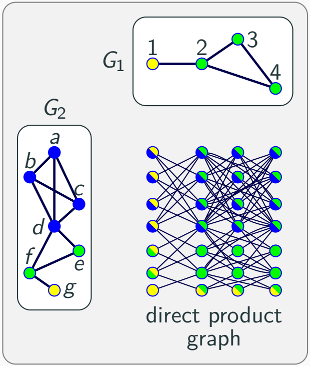
alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW

  too loose

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

close integers $\iff$ similar structure

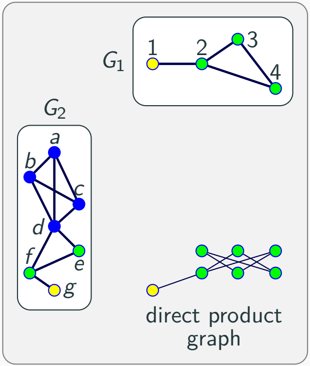
alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW

  too loose

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

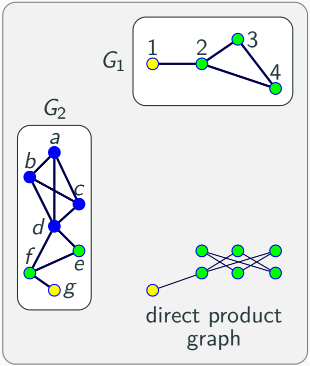
and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

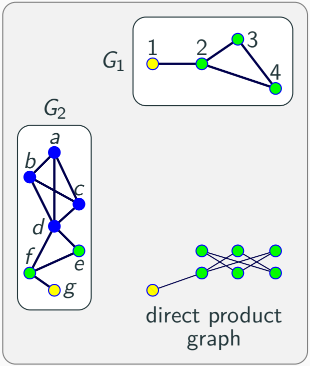
and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l - l'|}{\delta + 1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

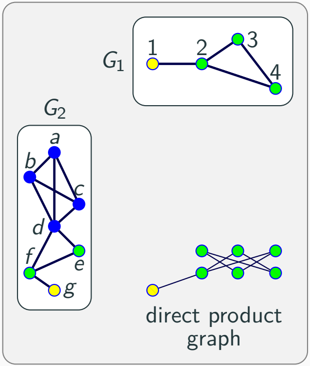
close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict
- $\delta \in \mathbb{R}_+$ SUSAN
  adaptive! **e.g.:** 0,0.5,1,1.5,2

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

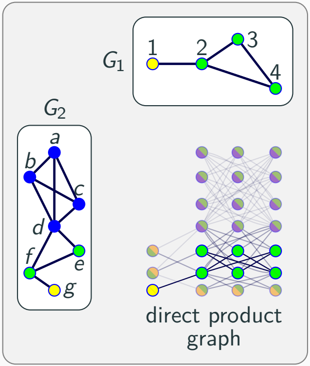
and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict
- $\delta \in \mathbb{R}_+$ SUSAN
  adaptive! **e.g.:** 0,0.5,1,1.5,2



Use kernel over $\mathbb{Z}$

$$k_\delta(l, l') := \max\left(0, 1 - \frac{|l - l'|}{\delta + 1}\right)$$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

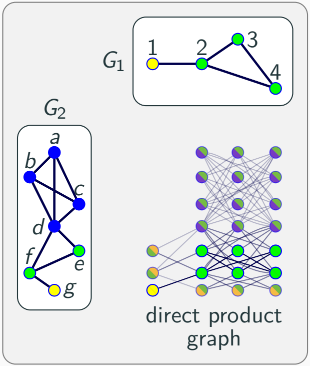
close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict
- $\delta \in \mathbb{R}_+$ SUSAN
  adaptive! **e.g.:** 0,0.5,1,1.5,2

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$$k_\delta(l, l') := \max\left(0, 1 - \frac{|l - l'|}{\delta + 1}\right)$$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

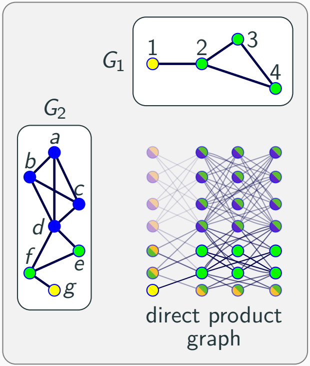
close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict
- $\delta \in \mathbb{R}_+$ SUSAN
  adaptive! **e.g.:** 0,0.5,1,1.5,2

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

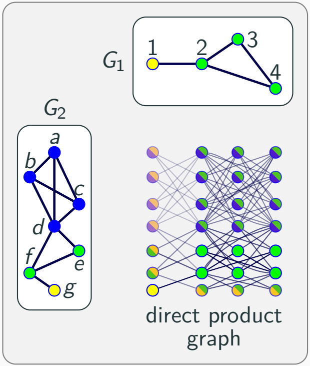
close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict
- $\delta \in \mathbb{R}_+$ SUSAN
  adaptive! **e.g.:** 0,0.5,1,1.5,2

Random Walk (Reproducing) Kernels [Gärtner et al., 2003]



Use kernel over $\mathbb{Z}$

$k_\delta(l, l') := \max\left(0, 1 - \frac{|l-l'|}{\delta+1}\right)$

where δ: bounded support

Goal: Count similar walks

Use core values as integer labels

and/or existing labels

close integers $\iff$ similar structure

alignment similarity from label kernel

Depending on δ:

- $\delta = \infty$ vanilla RW
  too loose
- $\delta = 0$ [Gärtner et al., 2003]
  too strict
- $\delta \in \mathbb{R}_+$ SUSAN
  adaptive! **e.g.:** 0,0.5,1,1.5,2

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Practical weights μ give:

- Geometric: $\mathbf{B}_g = (I - \lambda \mathbf{A}_\times)^{-1}\mathbf{e}$
- Exponential: $\mathbf{B}_e = \exp(\mathbf{A}_\times)\mathbf{e}$

Computing the Kernel II

Finally: sum # common walks:

- of any # steps (with weight μ_n)
- from each vertex to every other

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}$$

Practical weights μ give:

- Geometric: $\mathbf{B}_g = (I - \lambda \mathbf{A}_\times)^{-1}\mathbf{e}$
- Exponential: $\mathbf{B}_e = \exp(\mathbf{A}_\times)\mathbf{e}$

$\Longrightarrow$ computable as matrix vector (MV) operations with $\mathbf{A}_\times$

Computing the Kernel II

Finally: sum # common walks:

$$k(G_1, G_2) = \mathbf{e}^\top \underbrace{\sum_{n=0}^{\infty} \mu_n \mathbf{A}_\times^n \mathbf{e}}_{\mathbf{B}}$$

- of any # steps (with weight μ_n)
- from each vertex to every other

Practical weights μ give:

- Geometric: $\mathbf{B}_g = (I - \lambda \mathbf{A}_\times)^{-1}\mathbf{e}$ — Conjugate Gradient
- Exponential: $\mathbf{B}_e = \exp(\mathbf{A}_\times)\mathbf{e}$ — [Al-Mohy and Higham, 2011]

$\Longrightarrow$ computable as matrix vector (MV) operations with $\mathbf{A}_\times$

But: How do we compute the MV operations efficiently?

Computing the kernel II: Efficiently

To compute SUSAN efficiently

To compute SUSAN efficiently

Lemma

The MV operator for SUSAN with bandwidth δ is computable as

$$\mathbf{A}_\times x = \mathbf{T} \odot (\mathbf{A}''(\mathbf{T} \odot \mathbf{X})\mathbf{A}'^\top)$$

for $\mathbf{T}$ *block banded with constant blocks and bandwidth δ, time*

$$O((\delta + 1)(n' + n'')b^2)$$

for b the largest core size and n', n'' the vertex numbers of G', G''.

To compute SUSAN efficiently

- we decompose the contribution of each graph

Lemma
The MV operator for SUSAN *with bandwidth* δ *is computable as*

$$\mathbf{A}_{\times}x = \mathbf{T} \odot (\mathbf{A}''(\mathbf{T} \odot \mathbf{X})\mathbf{A}'^{\top})$$

for $\mathbf{T}$ *block banded with constant blocks and bandwidth* δ*, time*

$$O((\delta + 1)(n' + n'')b^2)$$

for b *the largest core size and* n'*,* n'' *the vertex numbers of* G', G''*.*

To compute SUSAN efficiently

- we decompose the contribution of each graph
- this reveals a block structure

Lemma
The MV operator for SUSAN *with bandwidth* δ *is computable as*

$$\mathbf{A}_\times x = \mathbf{T} \odot (\mathbf{A}''(\mathbf{T} \odot \mathbf{X})\mathbf{A}'^\top)$$

for $\mathbf{T}$ *block banded with constant blocks and bandwidth* δ*, time*

$$O((\delta + 1)(n' + n'')b^2)$$

for b *the largest core size and* n'*,* n'' *the vertex numbers of* G', G''*.*

To compute SUSAN efficiently

- we decompose the contribution of each graph
- this reveals a block structure
- groupping the vertices of equal coreness

Lemma

The MV operator for SUSAN with bandwidth δ is computable as

$$\mathbf{A}_{\times}x = \mathbf{T} \odot (\mathbf{A}''(\mathbf{T} \odot \mathbf{X})\mathbf{A}'^{\top})$$

for $\mathbf{T}$ block banded with constant blocks and bandwidth δ, time

$$O((\delta + 1)(n' + n'')b^2)$$

for b the largest core size and n', n'' the vertex numbers of G', G''.

Computing the kernel II: Efficiently

To compute SUSAN efficiently

- we decompose the contribution of each graph
- this reveals a block structure
- groupping the vertices of equal coreness
- exploit the bounded support

Lemma
The MV operator for SUSAN with bandwidth δ is computable as

$$\mathbf{A}_\times x = \mathbf{T} \odot (\mathbf{A}''(\mathbf{T} \odot \mathbf{X})\mathbf{A}'^\top)$$

for $\mathbf{T}$ block banded with constant blocks and bandwidth δ, time

$$O((\delta + 1)(n' + n'')b^2)$$

for b the largest core size and n', n'' the vertex numbers of G', G''.

Computing the kernel II: Efficiently

To compute SUSAN efficiently

- we decompose the contribution of each graph
- this reveals a block structure
- groupping the vertices of equal coreness
- exploit the bounded support
- and reduce computational complexity.

Lemma
The MV operator for SUSAN with bandwidth δ is computable as

$$\mathbf{A}_\times x = \mathbf{T} \odot (\mathbf{A}''(\mathbf{T} \odot \mathbf{X})\mathbf{A}'^\top)$$

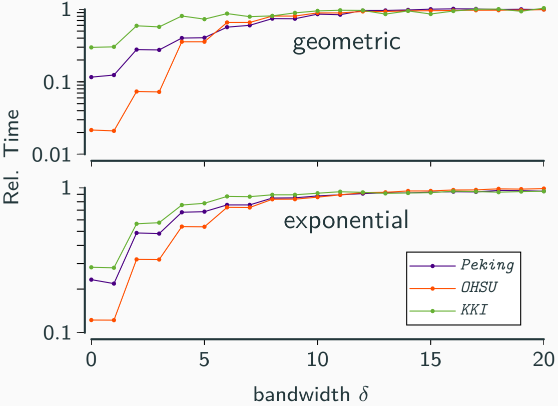
for $\mathbf{T}$ block banded with constant blocks and bandwidth δ, time

$$O((\delta + 1)(n' + n'')b^2)$$

for b the largest core size and n', n'' the vertex numbers of G', G''.

Results

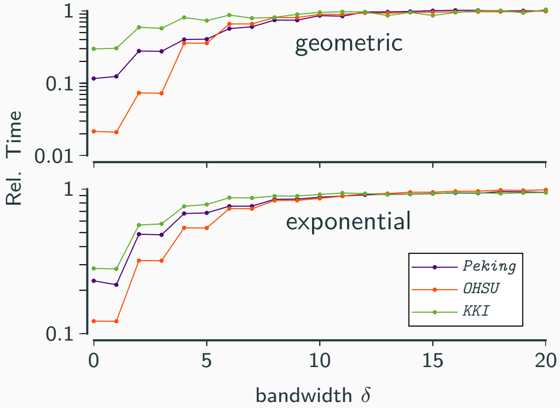
Time comparison



Relative wall-clock time

(SUSAN vs. naïve)

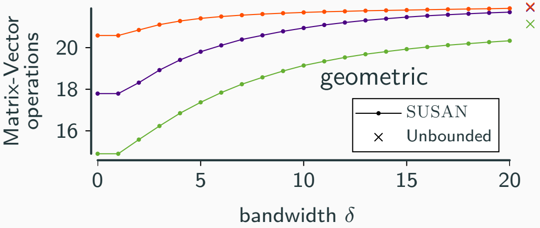
Time comparison



SUSAN

- outperforms naive computation, especially for small δ.

Time comparison

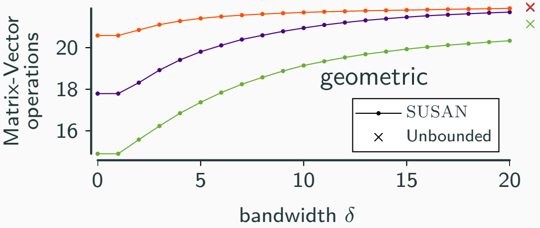


Number of iterations until convergence

SUSAN

- outperforms naive computation, especially for small δ.

Time comparison



Number of iterations until convergence

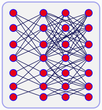
SUSAN

- outperforms naive computation, especially for small δ.
- (geometric) converges faster for smaller δ.

Conclusion

We study

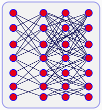
- random walk graph kernels
- weighted vertex alignments



Conclusion

We study

- random walk graph kernels
- weighted vertex alignments

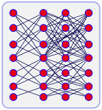


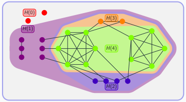
We propose

Conclusion

We study

- random walk graph kernels
- weighted vertex alignments





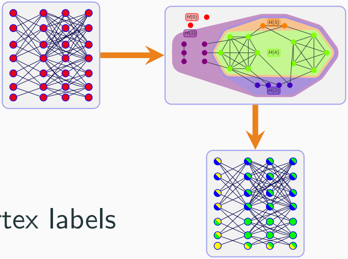
We propose

- coreness as structurally-aware vertex labels

Conclusion

We study

- random walk graph kernels
- weighted vertex alignments



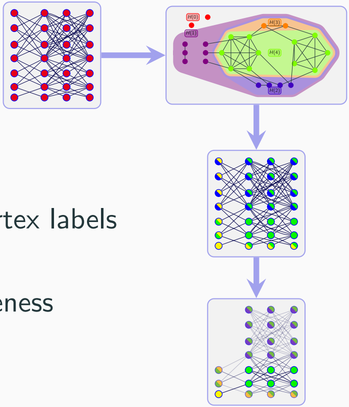
We propose

- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity

Conclusion

We study

- random walk graph kernels
- weighted vertex alignments



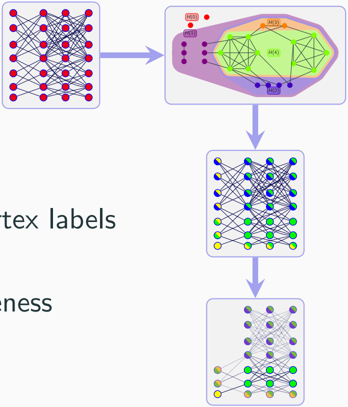
We propose

- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity
- bounded support kernel over coreness

Conclusion

We study

- random walk graph kernels
- weighted vertex alignments



We propose

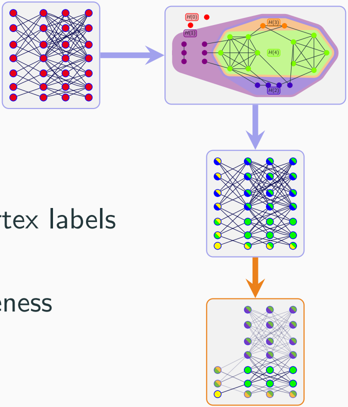
- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity
- bounded support kernel over coreness

With our work

Conclusion

We study

- random walk graph kernels
- weighted vertex alignments



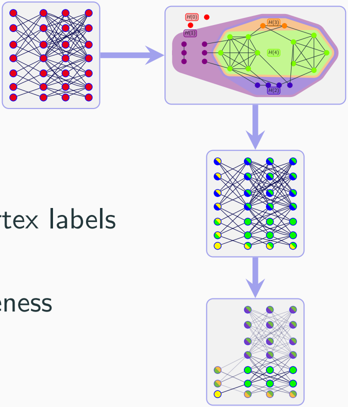
We propose

- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity
- bounded support kernel over coreness

With our work

- close the gap between loose and strict alignment constraints

Conclusion



We study

- random walk graph kernels
- weighted vertex alignments

We propose

- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity
- bounded support kernel over coreness

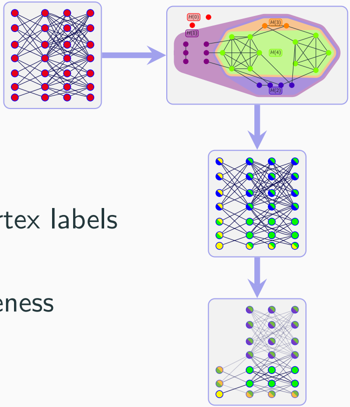
With our work

- close the gap between loose and strict alignment constraints
- competitive classification accuracy for certain datasets

Conclusion

We study

- random walk graph kernels
- weighted vertex alignments



We propose

- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity
- bounded support kernel over coreness

With our work

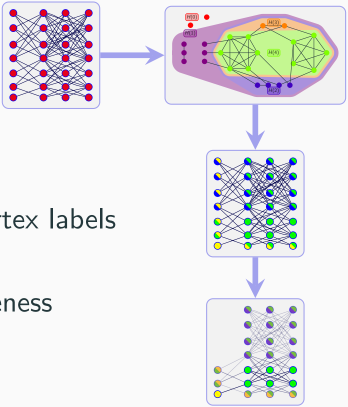
- close the gap between loose and strict alignment constraints
- competitive classification accuracy for certain datasets
- efficient iterative scheme for practical variants

Conclusion

Thank you!

We study

- random walk graph kernels
- weighted vertex alignments



We propose

- coreness as structurally-aware vertex labels
- induce intuitive vertex similarity
- bounded support kernel over coreness

With our work

- close the gap between loose and strict alignment constraints
- competitive classification accuracy for certain datasets
- efficient iterative scheme for practical variants

References i

- Al-Mohy, A. H. and Higham, N. J. (2011).
  Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators.
  SIAM J. Sci. Comp.
- Batagelj, V. and Zaversnik, M. (2003).
  An O(m) Algorithm for Cores Decomposition of Networks.
  arXiv:cs/0310049.

Gärtner, T., Flach, P., and Wrobel, S. (2003).
On Graph Kernels: Hardness Results and Efficient Alternatives.
In *Learning Theory and Kernel Machines*.

Shin, K., Eliassi-Rad, T., and Faloutsos, C. (2016).
CoreScope: Graph Mining Using k-Core Analysis—Patterns, Anomalies and Algorithms.
In *Data Mining (ICDM), 2016 IEEE 16th International Conference On*, pages 469–478. IEEE.