SUSAN: The Structural Similarity Random Walk Kernel

Janis Kalofolias, Pascal Welke, Jilles Vreeken
Comparing graphs

Applications

Standard Tools

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

Non-vectorial data

Can we apply standard tools on graphs?

⇒ Use a kernel on graphs

Machine learning methods

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

connectome

proteins

genetic structure
Comparing graphs

**Applications**
Classification, Regression, Clustering, Dim. Reduction...

**Standard Tools**
SVM, Logistic, K-Means, PCR...

Non-vectorial data

Can we apply standard tools on graphs?

⇒ Use a kernel on graphs
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

Can we apply standard tools on graphs?

Standard Tools
SVM, Logistic, K-Means, PCR

Need vector data
Comparing graphs

Applications
Classification, Regression, Clustering, Dim. Reduction …

Non-vectorial data ⇐urence of RAM series ⇒ Need vector data

Can we apply standard tools on graphs?

Standard Tools
SVM, Logistic, K-Means, PCR …
Comparing graphs

**Applications**
Classification, Regression, Clustering, Dim. Reduction

Non-vectorial data

Can we apply standard tools on graphs?

**Standard Tools**
SVM, Logistic, K-Means, PCR

Need vector data
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?

⇒ Use a kernel on graphs
How do kernels compare graphs?

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

Kernels define a space \( H \) with \( \langle \cdot, \cdot \rangle \) and mapping function \( \phi \). Use as graph similarity:

\[
K(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle
\]

We focus on Random Walk kernels.
How do kernels compare graphs?

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

Kernels define a space $H$ with $\langle \cdot, \cdot \rangle$ and mapping function $\phi \Rightarrow \text{Use as graph similarity}$ $k(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle_H$
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 \( \phi \).
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 $\phi$.
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 $\phi$. 

$\phi$ $\phi$ $\phi$
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 $\phi$.

$\implies$ 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 $\phi$

$\Rightarrow$ 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 $\phi$

$\implies$ 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 $\phi$

$$k(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$$
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 $\phi$

$\implies$ 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 $\phi$

$$k(G_1, G_2) := \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}}$$

We focus on Random Walk kernels.
Goal: Count graph walks

But:

• Assume vertex alignment
e.g.: a ≡ 1, b ≡ 2, c ≡ 3, d ≡ 4

• Create alignment graph

• Walk in alignment graph
e.g.: b ≡ 2, c ≡ 3, d ≡ 4, b ≡ 2

But:

Alignments are rarely available⇒ Use all possible alignments
But:

If vertices are not similar?⇒ Not all alignments equally good
**Goal:** Count graph walks

**example:** 3-step walk: (1, 2, 3, 4)

- **G1**

** ex:** 3-step walk: (1, 2, 3, 4)
Random Walk (Reproducing) Kernels

Goal: Count graph walks

Example:

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

Graph $G_1$
Random Walk (Reproducing) Kernels

\[ G \]

**Goal:** Count graph walks

\[ \text{# 1-step walks from 1, 3?} \]

\[
\begin{bmatrix}
0 \\
2 \\
0 \\
1
\end{bmatrix}
= \begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
1 \\
0
\end{bmatrix}
\]

\[ x_1 \quad A \quad x_0 \]

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

- Alignments are rarely available
- Use all possible alignments
- If vertices are not similar?
  - Not all alignments equally good
Goal: Count graph walks

G1

\# 1-step walks from 1, 3?

$$\begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} x_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} A \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} x_0$$
Goal: Count graph walks

# 1-step walks from 1, 3?

\[
\begin{bmatrix}
0 \\
2 \\
0 \\
1
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
1 \\
0
\end{bmatrix}
\]
Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

Goal: Count graph walks

# 1-step walks from 1, 3?

\[
\begin{bmatrix}
0 \\
2 \\
0 \\
1
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
1 \\
0
\end{bmatrix}
\]
Random Walk (Reproducing) Kernels [Gärtner et al., 2003]

Goal: Count graph walks

# 1-step walks from 1, 3?

$$\begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

#k-step walks from x₀?

$$x_k = A^k x_0$$
**Goal**: Count graph walks

**But**: in 2 graphs?
**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
**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
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
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
Goal: Count common walks

But: in 2 graphs?

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

\[\rightarrow\] Use all possible alignments
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
⇒ Use all possible alignments
**Goal:** Count common walks

**But:** in 2 graphs?

- Assume vertex alignment
  
  \[ a \equiv 1, \ b \equiv 2, \ c \equiv 3, \ d \equiv 4 \]

- Create alignment graph

- Walk in alignment graph
  
  \[ b \equiv 2, \ c \equiv 3, \ d \equiv 4, \ b \equiv 2 \]

**But:** Alignments are rarely available

\[ \implies \text{Use all possible alignments} \]
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

$\implies$ Use all possible alignments

But: If vertices are not similar?

Direct product graph:

$A_x = A \otimes A'$

$A_x x_x = (Ax) \otimes (A'x')$
**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

\[ \implies \text{Use all possible alignments} \]

**But**: If vertices are not similar?

\[
A_x = A \otimes A' \\
A_x x_x = (Ax) \otimes (A'x')
\]
**Goal:** Count common walks

**But:** in 2 graphs?

- Assume vertex alignment
  
  \[a \equiv 1, \ b \equiv 2, \ c \equiv 3, \ d \equiv 4\]

- Create alignment graph

- Walk in alignment graph
  
  \[b \equiv 2, \ c \equiv 3, \ d \equiv 4, \ b \equiv 2\]

**But:** Alignments are rarely available

\[\Rightarrow \text{Use all possible alignments}\]

**But:** If vertices are not similar?

\[\Rightarrow \text{Not all alignments equally good}\]
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

⇒ Only match similar vertices
Are all vertex alignments equally good?

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

\[ \Rightarrow \text{Only match similar vertices} \]

Labeled vertices

\[ \text{HO} \quad \text{N} \quad \text{O} \]

\[ \text{versus} \]

\[ \text{HO} \quad \text{H} \quad \text{N} \quad \text{O} \]

\[ \checkmark \text{same label} \Rightarrow \text{similar vertices} \]
Are all vertex alignments equally good?

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

⇒ Only match similar vertices

Labeled vertices

\[
\begin{align*}
\text{HO} & \quad \text{H} \\
\text{N} &
\end{align*}
\]

\[
\begin{align*}
\text{HO} & \quad \text{H} \\
\text{N} &
\end{align*}
\]

vs

\[
\begin{align*}
\text{H} & \quad \text{N} \\
\text{CH}_3 &
\end{align*}
\]

✓ same label ⇒ 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

⇒ Only match similar vertices

Labeled vertices

\[ \text{HO} \quad \text{H} \quad \text{N} \quad \text{O} \]

- Same label ⇒ 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

⇒ Only match similar vertices

Labeled vertices

H\text{O} \quad \text{N}

vs

\text{HO} \quad \text{H}\text{N} \text{O}

- same label ⇒ 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

\[ \Rightarrow \text{Only match similar vertices} \]

**Labeled vertices**

\[
\begin{array}{c}
\text{HO} \\
\text{H} \\
\text{N} \\
\text{O}
\end{array}
\quad \text{vs} \quad
\begin{array}{c}
\text{HO} \\
\text{H} \\
\text{N} \\
\text{CH}_3
\end{array}
\]

- \( \checkmark \) same label \( \Rightarrow \) similar vertices
- \( \times \) \( G_2 \) has no \( O \). What now?
- \( \times \) How close is \( C \) to \( H \)?

**Unlabeled graphs**

- \( \checkmark \) many similarity measures
- \( \times \) not always clear or easy

We seek a vertex partitioning

- structurally aware
- efficient to compute
- defines partition similarity

We propose to use

\[ \Rightarrow \text{core decomposition} \]
Definition \((k\text{-core of graph } G)\)
A maximal subgraph with vertices of degree at least \(k\).
Core decomposition

Definition \((k\text{-core of graph } G)\)
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**

![Diagram of a graph with a highlighted $k$-core subgraph]

$H(4)$
Core decomposition

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

**Example**

![Diagram showing $H(0)$, $H(3)$, and $H(4)$ as examples of k-cores.](image)
Core decomposition

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

Example

\begin{tikzpicture}
\draw[fill=red!30!white,opacity=0.3] (0,0) circle (2cm);
\draw[fill=green!30!white,opacity=0.3] (0,0) circle (4cm);
\draw[fill=blue!30!white,opacity=0.3] (0,0) circle (6cm);
\end{tikzpicture}
Definition ($k$-core of graph $G$)
A maximal subgraph with vertices of degree at least $k$.

Example

Decomposition: $\kappa : V \rightarrow \mathbb{N}$
Core decomposition

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

Example

\[ H(0) \]
\[ H(1) \]
\[ H(2) \]
\[ H(3) \]
\[ H(4) \]

Definition (**vertex coreness**)  
\[ \kappa(u) := \max_{u \in H(k)} k \]

Decomposition:  
\[ \kappa : V \rightarrow \mathbb{N} \]
Core decomposition

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

**Example**

\[ H(0) \]

\[ H(1) \]

\[ H(2) \]

\[ H(3) \]

\[ H(4) \]

**Definition (vertex coreness)**

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

**Decomposition:** \( \kappa : V \rightarrow \mathbb{N} \)

- \( k \)-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:** $\kappa : V \rightarrow \mathbb{N}$

- $k$-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**

![Example Diagram]

**Definition (vertex coreness)**

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

**Decomposition:** $\kappa : V \rightarrow \mathbb{N}$

- $k$-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

**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 $\delta$:

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

Use core values as integer labels and/or existing labels.
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

Goal: Count similar walks

Use kernel over $\mathbb{Z}$

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

where $\delta$: bounded support
Goal: Count similar walks

Use core values as integer labels and/or existing labels close integers ⟷ similar structure alignment similarity from label kernel

Depending on $\delta$:

- $\delta = \infty$: 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 $\delta$: bounded support
Use kernel over $\mathbb{Z}$

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

where $\delta$: 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 $\delta$:

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

- \(\delta = \infty\): vanilla RW too loose

Use kernel over \(\mathbb{Z}\)

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

where \(\delta\): bounded support
**Random Walk (Reproducing) Kernels**

**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 $\delta$:
- $\delta = \infty$ 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 $\delta$: bounded support
Random Walk (Reproducing) Kernels

[Gaertner 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

Depending on $\delta$:
- $\delta = \infty$ vanilla RW too loose
- $\delta = 0$ [Gaertner 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 $\delta$: bounded support

direct product graph

$G_1$

$G_2$

$1 \ 2 \ 3 \ 4$

$a \ b \ c \ d \ e \ f \ g$
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

Depending on $\delta$:

- $\delta = \infty$ 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 $\delta$: bounded support direct product graph
Random Walk (Reproducing) Kernels

Use kernel over \( \mathbb{Z} \)

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

where \( \delta \): bounded support

Goal: Count similar walks

Use core values as integer labels and/or existing labels close integers ⇔ similar structure alignment similarity from label kernel

Depending on \( \delta \):

- \( \delta = \infty \) 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

\[ G_1 \times G_2 \]

direct product graph

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array} \]

\[ \begin{array}{cccc}
a & b & c & d \\
\end{array} \]

\[ \begin{array}{cccc}
e & f & g & \end{array} \]

direct product graph

**Goal:** Count similar walks

Use kernel over \( \mathbb{Z} \)

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

where \( \delta \): bounded support

Use core values as integer labels and/or existing labels close integers \( \iff \) similar structure alignment similarity from label kernel

Depending on \( \delta \):

- \( \delta = \infty \) vanilla RW too loose
- \( \delta = 0 \) [Gärtner et al., 2003] too strict
- \( \delta \in \mathbb{R}_+ \) SUSAN adaptive!

\[ \text{e.g.: } 0, 0.5, 1, 1.5, 2 \]
Use kernel over \( \mathbb{Z} \)

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

where \( \delta \): bounded support

\( \mathbf{Goal} \): 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 \( \delta \):

- \( \delta = \infty \) : vanilla RW
  - too loose
- \( \delta = 0 \) : [Gärtner et al., 2003]
  - too strict
- \( \delta \in \mathbb{R}_+ \)
  - SUSAN
  - adaptive!

\text{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 $\delta$: 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 $\delta$:

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

**Goal:** Count similar walks

Use kernel over $\mathbb{Z}$

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

where $\delta$: bounded support

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

Use core values as integer labels and/or existing labels close integers $\iff$ similar structure alignment similarity from label kernel

Depending on $\delta$:

- $\delta \in \mathbb{R}_+$ SUSAN adaptive!
Finally: sum # common walks:
- of any # steps (with weight $\mu_n$)
- from each vertex to every other

$$k(G_1, G_2) = e^\top \sum_{n=0}^{\infty} \mu_n A^*_n e$$
Finally: sum # common walks:
- of any # steps (with weight $\mu_n$)
- from each vertex to every other

$$k(G_1, G_2) = e^\top \sum_{n=0}^{\infty} \mu_n A^n_x e$$
Finally: sum # common walks:

- of any # steps (with weight $\mu_n$)
- from each vertex to every other

$$k(G_1, G_2) = e^\top \sum_{n=0}^{\infty} \mu_n A^n e$$
Finally: sum # common walks:
- of any # steps (with weight $\mu_n$)
- from each vertex to every other

$$k(G_1, G_2) = e^T \sum_{n=0}^{\infty} \mu_n A^n e$$
Finally: sum # common walks:
- of any # steps (with weight $\mu_n$)
- from each vertex to every other

$$k(G_1, G_2) = e^T \sum_{n=0}^{\infty} \mu_n A^n e$$
Finally: sum \# common walks:
- of any \# steps (with weight $\mu_n$)
- from each vertex to every other

Practical weights $\mu$ give:
- Geometric: $B_g = (I - \lambda A_x)^{-1} e$
- Exponential: $B_e = \exp(A_x)e$
Finally: sum # common walks:

- of any # steps (with weight $\mu_n$)
- from each vertex to every other

Practical weights $\mu$ give:

- Geometric: $B_g = (I - \lambda A_x)^{-1}e$
- Exponential: $B_e = \exp(A_x)e$

$\Rightarrow$ computable as matrix vector (MV) operations with $A_x$
Finally: sum \# common walks:

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

Practical weights \( \mu \) give:

- Geometric: \( B_g = (I - \lambda A_x)^{-1}e \)
- Exponential: \( B_e = \exp(A_x)e \)  
  \[ \text{[Al-Mohy and Higham, 2011]} \]

\( k(G_1, G_2) = e^\top \sum_{n=0}^{\infty} \mu_n A_x^n e \)

\( \Rightarrow \) computable as matrix vector (MV) operations with \( A_x \)

**But**: How do we compute the MV operations efficiently?
To compute SUSAN efficiently
To compute SUSAN efficiently

**Lemma**
The MV operator for SUSAN with bandwidth $\delta$ is computable as

$$A \times x = T \odot (A''(T \odot X)A^\top)$$

for $T$ block banded with constant blocks and bandwidth $\delta$, 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 $\delta$ is computable as

$$A_x x = T \odot (A''(T \odot X)A^{T_T})$$

for $T$ block banded with constant blocks and bandwidth $\delta$, 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

Lemma

The MV operator for SUSAN with bandwidth $\delta$ is computable as

$$A_x x = T \odot (A''(T \odot X)A^\top)$$

for $T$ block banded with constant blocks and bandwidth $\delta$, 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
- grouping the vertices of equal coreness

**Lemma**

The MV operator for SUSAN with bandwidth $\delta$ is computable as

$$A \times x = T \odot (A''(T \odot X)A^T)$$

for $T$ block banded with **constant blocks** and bandwidth $\delta$, 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
- grouping the vertices of equal coreness
- exploit the bounded support

**Lemma**

The MV operator for SUSAN with bandwidth $\delta$ is computable as

$$A \times x = T \odot (A'' (T \odot X) A^T)$$

for $T$ block banded with constant blocks and bandwidth $\delta$, 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
- grouping the vertices of equal coreness
- exploit the bounded support
- and reduce computational complexity.

**Lemma**

The MV operator for SUSAN with bandwidth $\delta$ is computable as

$$A \times x = T \odot (A''(T \odot X)A^T)$$

for $T$ block banded with constant blocks and bandwidth $\delta$, 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)

- **SUSAN** outperforms naive computation, especially for small \( \delta \).
- (geometric) converges faster for smaller \( \delta \).

![Graph showing time comparison for geometric and exponential bandwidths.](image)
SUSAN

- outperforms naive computation, especially for small $\delta$. 
Time comparison

SUSAN

- outperforms naive computation, especially for small $\delta$. 

Number of iterations until convergence
SUSAN

- outperforms naive computation, especially for small $\delta$.
- (geometric) converges faster for smaller $\delta$. 
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

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
We study
- random walk graph kernels
- weighted vertex alignments

We propose
- coreness as structurally-aware vertex labels
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
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
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
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
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
