SUSAN: The Structural Similarity Random Walk Kernel

Janis Kalofolias° Pascal Welke• Jilles Vreeken°

Abstract
Random walk kernels are a very flexible family of graph kernels, in which we can incorporate edge and vertex similarities through positive definite kernels. In this work we study the particular case within this family in which the vertex kernel has bounded support. We motivate this property as the configurable flexibility in terms of vertex alignment between the two graphs on which the walk is performed. We study several fast and intuitive ways to derive structurally aware labels and combine them with such a vertex kernel, which in turn is incorporated in the random walk kernel. We provide a fast algorithm to compute the resulting random walk kernel and we give precise bounds on its computational complexity. We show that this complexity always remains upper bounded by that of alternative methods in the literature and study conditions under which this advantage can be significantly higher. We evaluate the resulting configurations on their predictive performance on several families of graphs and show significant improvements against the vanilla random walk kernel and other competing algorithms.
Keywords: random walk kernel, band-limited kernel

1 Introduction

Through the choice of an appropriate graph kernel one can harness the power of generic machine learning methods to learn functions over graphs. Graph kernels are positive (semi-)definite functions on pairs of graphs, and are most often defined as R-convolutions [6]. These can be seen as an extension of the notion of convolution that operates on kernels defined on sub-structures of these graphs, and in a way that preserves the positive-definiteness of those kernels. Random walks have been among the first such formulations [5], and through their more recent generalisations [20] they still remain relevant for certain configurations [9].

Intuitively, given an alignment ϕ between the vertices of one graph to those of another, one can count the number of simultaneous random walks between the two graphs. These are random walks performed on two graphs, starting from one vertex in one graph and one vertex in the other; then they advance the vertex of each graph one edge at a time in lockstep, so that at each step the vertices of two graphs respect the given alignment ϕ. Of course, in general such an alignment is not available. The random walk kernels avoid the dependency on any one particular mapping between the vertices of two graphs by instead considering all simultaneous walks over all possible alignments between the vertices of the one graph to those of the other.

The number of all such walks can be elegantly formulated in terms of linear-algebraic operations, thus allowing the use of a variety of algorithms for its computation, while being very flexible: they allow to incorporate arbitrary distributions over the vertices of the graphs as starting and stopping points of the walks, different weights for different walk lengths, node and edge similarities encoded as kernels, as well as the possibility to incorporate edge labels [20]. This allows, for example, to down-weigh vertex alignments in the random walks between vertices that we know (or strongly suspect) to be dissimilar.

Recent work [14, 16, 18] suggests that (non-random-walk) kernels that align vertices based on structural properties improve predictive performance on several datasets. Examples of such properties are the coreness of vertices, vertex degree, or Weisfeiler-Lehman labels. In these cases, vertices with the same or *similar* value share a similar structure and should arguably be aligned in a random walk kernel using a suitable kernel on structural property values.

In our work we focus on the case that very dissimilar vertices are not just down–weighed, but are not allowed to be simultaneously visited by a random walk at all. Assuming that a good estimation of vertex similarity was available then one would expect that the count of walks that respect such vertex alignments would preserve or improve the accuracy of the walk, while the other walks might be considered noise. In many cases this results in more fine grained similarity functions that give high similarity to graphs that have many simultaneous random walks that respect structure, e.g., simultaneously travelling through similarly dense regions.

Consider the two small graphs G, H in Fig. 1. Assume we know that vertices with label 1 are dissimilar to label 3 vertices, but those with label 2 are somewhat similar to both. The vanilla random walk kernel would consider the full product graph in Fig. 1c, modelling vertex alignment using a kernel function on vertices. On the other hand, the actual information that is relevant in this scenario is represented by the much smaller product graph of Fig. 1b, while considering only identically labelled vertices as matches (Fig. 1a) would arguably lose too much information.

We thus capture this assumption as a restriction of the allowed alignment during the simultaneous walk, and encode this as a vertex kernel of bounded support on structure-aware ordinal labels, which can be extracted, for instance, from structural properties of the vertices. For the resulting ran-

°CISPA Helmholtz Center for Information Security, Germany {janis.kalofolias, jv}@cispa.de

•University of Bonn, welke@cs.uni-bonn.de

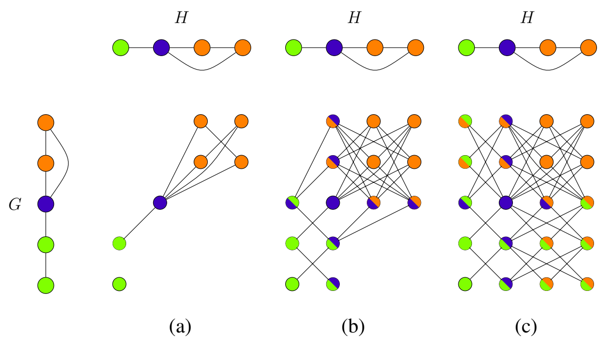

Figure 1: Two small graphs G, H with structural labels 1, 2, 3 indicated as colours. Considering only vertex pairs of identical labels as in (a) arguably [5] results in too sparse product graphs. The product graph (c) considered by the generic random walk kernel [20] includes all pairs of vertices. Our proposed kernel may disallow the mapping of label 1 to 3 and can then be computed on (b).

dom walk kernel, we present a computational method that is practically and asymptotically faster than alternative computational methods. This superiority increases as we enforce stricter assumptions and only allow alignments of closer labels. In practice, this restriction formally corresponds to using a vertex kernel with a smaller support. If (almost) complete alignment information is available, such as, for example in brain activity networks [15], our method can be computed in up to quadratic time, in contrast to the cubic time of the generalised framework. This further improves accordingly when using sparse graph representations.

Our contributions are as follows:

- We propose a fast algorithm to compute random walk kernels based on bounded support vertex kernels.
- We thus study the gap between the rather restrictive choice of Gärtner et al. [5] and the general framework of Vishwanathan et al. [20].
- We describe a class of bounded support vertex kernels for integer-valued structural attributes and
- study the performance of several such attributes when used within our framework.
- We empirically show significant improvements on some datasets, while generally being on par with the vanilla random walk kernel.

Although we present our work as a stand-alone kernel, perhaps a more important goal is to show that even for datasets on which we do not outperform the vanilla kernel in terms of accuracy, we still attain shorter running time. Additionally, our work can also improve state-of-the-art kernels that use random walks as a component [13]. For conciseness we postpone the proofs to the online appendix.[1]

[1] https://eda.mmci.uni-saarland.de/susan

2 Preliminaries

We begin with an overview of the necessary concepts that we make use of in our analysis and introduce the relevant notation. For conciseness we first define $[n] := \{1, \ldots, n\}$ to be the set of the first n positive naturals. We additionally define $[n]_0 := \{0, 1, \ldots, n\}$.

In this work we consider undirected graphs $G = (V, E)$. The set of edges can also be represented as the **adjacency matrix** $\mathbf{A} \in \mathbb{R}^{n\times n}$, which has entries $[\mathbf{A}]_{i,j}$ equal to 1 if $(v_i, v_j) \in E$, or 0 otherwise, where we assume a fixed ordering $v_1, \ldots, v_n$ of V. To describe edge-weighted graphs we straightforwardly replace the adjacency matrix with its weighted version. Since G is undirected, the adjacency matrix is symmetric: $\mathbf{A} = \mathbf{A}^\top$. For two matrices $\mathbf{A}' \in \mathbb{R}^{n'\times m'}$ and $\mathbf{A}'' \in \mathbb{R}^{n''\times m''}$, we write their Kronecker product as $\mathbf{A}' \otimes \mathbf{A}'' \in \mathbb{R}^{n'n''\times m'm''}$; similarly, for $\mathbf{A}', \mathbf{A}'' \in \mathbb{R}^{n\times m}$ we denote their Hadamard product as $\mathbf{A}' \circ \mathbf{A}'' \in \mathbb{R}^{n\times m}$. In the following analysis we generally consider a pair of graphs, G' and G'', in which case we implicitly refer to property p of the first graph as p' and of the second as p''.

2.1 The Random Walk kernel

The basic random walk kernel applied on two graphs $G' = (V', E')$ and $G'' = (V'', E'')$ is equal to the number of simultaneous walks between them. More precisely, assume a mapping $\phi : V' \to V''$ between the nodes of the two graphs; then a simultaneous walk would only be allowed to traverse an edge $e = (u', v') \in E'$ if an edge also exists in G'' between the mapped vertices of it $(\phi_{u'}, \phi_{v'}) \in E''$. Equivalently, we could visualise such a walk as one over the graph G_ϕ with vertices $(u', \phi_{u'})$ for all $u' \in V'$ and edges $E_\phi := \big\{\big((u,v),(\phi_{u'},\phi_{v'})\big) \mid (u',v') \in E' \wedge (\phi_{u'},\phi_{v'}) \in E''\big\}$.

The random walk kernel avoids the dependency on any one particular such mapping ϕ. Instead, it considers all possible mappings by performing a simultaneous walk on the graph with vertices the Cartesian product $V_\times := V' \times V''$ and edges the union $E_\times := \bigcup_\phi E_\phi := \{((u', u''), (v', v'')) \mid (u', v') \in E' \wedge (u'', v'') \in E''\}$ over all possible vertex mappings. This results in the *direct product graph* $G_\times := (V_\times, E_\times)$. A walk on $G_\times$ is equivalent to a simultaneous walk on graphs G' and G''. Indeed, consider advancing a walk on $G_\times$ from node $(u', u'') \in V_\times$: we can interpret this as first randomly selecting a mapping ϕ which i) respects the current node $\phi_{u'} = u''$, and for which ii) E_ϕ contains at least one edge with an endpoint (u', u''); then traversing one of these edges from E_ϕ at random. In this work, we down-weight or outright exclude certain such alignments ϕ.

The adjacency matrix $\mathbf{A}_\times$ of $G_\times$ is equal to the Kronecker product of the adjacency matrices of G' and G'' [21]. That is: $\mathbf{A}_\times = \mathbf{A}' \otimes \mathbf{A}''$. Additionally, the ν-th power of the adjacency matrix $\mathbf{A}^\nu$ of a graph contains in its element $[\mathbf{A}^\nu]_{i,j}$ the number of walks with exactly ν steps from the

i-th to the j-th vertex of the graph. Hence, the number of all such walks can be written as $\mathbf{e}^\top \mathbf{A}_\times^\nu \mathbf{e}$, where $\mathbf{e} = (1, \ldots, 1)^\top$ is the vector of all ones. The random walk kernel that counts all simultaneous random walks is thus defined as

$$k(G', G'') = \sum_{\nu=0}^{\infty} \mu_\nu \mathbf{e}^\top \mathbf{A}_\times^\nu \mathbf{e} \,, \tag{2.1}$$

where the constants μ_ν ensure the series converges [5].

The choice of the sequence μ gives rise to two special cases that allow the analytic computation of the series in Eq. (2.1). The geometric sequence $\mu_{geom}(\nu) := \lambda^\nu$ defines the **geometric** random walk kernel, while the power series coefficients of the exponential function $\mu_{exp}(\nu) := {}^1/_{\nu!}$ define the **exponential** one. Using symbolic forms these are

$$\begin{aligned} k_{geom}(G', G'') &:= \mathbf{e}^\top (I - \lambda \mathbf{A}_\times)^{-1} \mathbf{e} \text{ and} \\ k_{exp}(G', G'') &:= \mathbf{e}^\top \exp(\lambda \mathbf{A}_\times) \, \mathbf{e} \,, \end{aligned} \tag{2.2}$$

where in the former λ takes any value small enough so that $\|\lambda \mathbf{A}_\times\| < 1$, as necessary for the geometric series to converge, while in the latter it is a positive parameter.

The random walk kernel of Eq. (2.1) as proposed by Gärtner et al. [5] has been extended or adapted in several ways. Borgwardt et al. [3] use a kernel on vertices and edges to define the edge similarities of the direct product graph. Additionally, the random walk can be equipped with starting/ending probabilities and weighted matrices. These all fit in the framework [20]

$$k(G', G'') = \sum_{\nu=0}^{\infty} \mu(\nu) \mathbf{q}^\top \mathbf{W}_\times^\nu \mathbf{p} \,, \tag{2.3}$$

where $\mathbf{p} = \mathbf{p}' \otimes \mathbf{p}''$ and $\mathbf{q} = \mathbf{q}' \otimes \mathbf{q}''$ are start and stop probabilities on vertices and $\mathbf{W}_\times$ contains generalised edge similarities, as computed by a kernel between the edges of the two graphs. These generalisations can now be incorporated in the analytic expressions of Eq. (2.2)

$$\begin{aligned} k_{geom}(G', G'') &:= \mathbf{q}^\top (\mathbf{I} - \lambda \mathbf{W}_\times)^{-1} \mathbf{p} \text{ and} \\ k_{exp}(G', G'') &:= \mathbf{q}^\top \exp(\lambda \mathbf{W}_\times) \, \mathbf{p} \,, \end{aligned} \tag{2.4}$$

where the parameter λ plays a similar role as in Eq. (2.2). Now, the formulation of Eq. (2.3) & (2.4) allows for a vertex kernel that encodes the similarity of vertex combinations in the generalised edge similarities.

2.1.1 Computation In theory, the geometric and exponential kernels of Eq. (2.1) can be computed by solving a linear system and computing a matrix exponential, respectively, each of which operate on a matrix that is easily derived from $\mathbf{A}_\times = \mathbf{A}' \otimes \mathbf{A}''$. In practice, however, both these operations are cubic in general, and since $\mathbf{W}_\times \in \mathbb{R}^{n'n'' \times n'n''}$, they would require an excessive computation of $O(n^6)$.

To this end Vishwanathan et al. [19] note that the spectral decomposition of $\mathbf{A}_\times$ can be efficiently computed as $S_\times D_\times S_\times^\top = (S' \otimes S'')(D' \otimes D'')(S' \otimes S'')^\top$. This allows one to compute Eq. (2.2) as $\mathbf{q}^\top S_\times \left(\sum_{\nu=0}^{\nu_{\max}} \mu(\nu) D_\times^\nu\right) S_\times^\top \mathbf{p}$, where the—otherwise cubic—operation need only operate on the diagonal matrix $D_\times$. As an additional benefit, the spectral decomposition of each adjacency matrix may only be computed once for all graph pairs; then the entire Gram matrix needs only $O(n^2 m^2 \nu_{\max} + mn^3)$ time for m graphs.

Alternatively, the same work used the Conjugate Gradient iterative solver to compute k_{geom} of Eq. (2.2). Each iteration performs a matrix-vector operation with a Kronecker matrix by employing the identity

$$(\mathbf{A}' \otimes \mathbf{A}'')\mathbf{x} = \operatorname{vec}(\mathbf{A}'' \operatorname{mat}(\mathbf{x}) \mathbf{A}'^\top) \,, \tag{2.5}$$

where the operation $\operatorname{vec} : \mathbb{R}^{m \times n} \to \mathbb{R}^{mn}$ creates a vector by concatenating all columns of a matrix and $\operatorname{mat} : \mathbb{R}^{mn} \to \mathbb{R}^{m \times n}$ is the inverse operation. Using this identity within iterative methods requires $O\left(n^3 k\right)$ computations per kernel entry, assuming k iterations.

However, when a vertex or an edge kernel is used, as in the general framework of Eq. (2.3), the matrix $\mathbf{W}_\times$ does not have a Kronecker decomposition. Then, these and other proposed methods are not applicable any more, and iterative methods must be used, which are based on repeated matrix-vector operations. These methods also allow using an edge kernel as long as it has a known feature representation on a finite d-dimensional Hilbert space [20]; the matrix-vector operation is then performed on each of the d dimensions of feature vectors, which needs $O\left(n^3 d\right)$ computations.

As a workaround, the resulting similarity matrix can be approximated by its nearest Kronecker product, although not without downsides: the additional complexity of a rank-1 singular value decomposition on the full matrix [10], as well as the potential of arbitrary quality degradation due to the involved approximation.

2.2 Graph Concepts We now introduce the graph coreness, a structural property that will be used in our kernel. For a vertex v we denote its **neighbours** as $N(v) := \{u \in V \mid (u, v) \in E\}$ and its **degree**, i.e., the number of its neighbours, by $\delta(v) := |N(v)|$.

The **induced subgraph** $G[U] := (U, \{(u, v) \in E \mid u, v \in U\})$ of a vertex subset $U \subseteq V$ is the subgraph of G with vertices U and those edges of E with both endpoints in U. We also denote $\delta_U(u)$ the degree of vertex u in $G[U]$.

A k**-core component** of a graph G is an (inclusion-wise) maximal connected subgraph of G whose vertices U have all a degree of at least $\delta_U(u) \geq k$. The subgraph comprising all k-core components of this graph is its k**-core** $H(k)$, with vertices the set $V(k)$. Hence, $H(k) := G[V(k)]$.

The annotated k-cores of the example graph in Fig. 2 show that the k-cores are nested to form a hierarchy over the

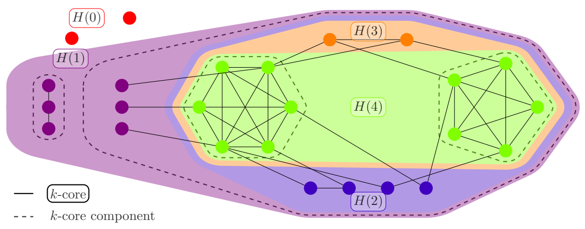

Figure 2: The core decomposition of a graph defines a partitioning of its vertices into named groups, its k-shells, that contain vertices with similar connectedness.

vertices. We also define the k-**shell** of G as the set of vertices that lie in the k-core but not in the $k+1$-core (same-coloured vertices in the figure). In this way, the k-shells define a partitioning over the vertices: the **core decomposition** of G, $\kappa_G : V \to [\kappa]_0$, which assigns to each vector v its **core number** (or **coreness**) $\kappa_G(v) := \{k \mid v \in V(k)\}$. The maximum core value of all vertices in the graph is called its **degeneracy** $\kappa := \max_{v\in V} \kappa_G(v)$.

3 Structure-Aware Vertex Similarities

We now introduce the main object of our study: a random walk kernel that uses a vertex kernel based on structural attributes of the vertices, namely their degree or coreness. This vertex kernel can be combined with any other vertex kernel and/or an edge kernel, thus remaining fully flexibile.

The reason underlying our use of structural graph properties is twofold. On the one hand, vertex labels on graphs might be unavailable, or unsuitable as a basis for vertex alignment. For example, the number of joint walks between the graphs can deviate detrimentally for noisy labels, especially when "hub"-vertices of one graph are mislabelled. Additionally, it can be difficult to evaluate the similarity of vertices based on labels beyond testing for label equality, e.g., when these are categorical, often retaining too little information [5]. In contrast, many structural vertex properties, such as coreness, allow for a natural way to compare vertices with unequal but similar values, since a similar value indicates a similarly robust connection or "centrality" in a graph [7, 17]. Using structural properties as vertex labels allows the use of a multitude of available kernels on scalars for the evaluation of vertex similarity. Choosing a smoother amongst them allows for greater resilience against edge or label noise,[2] while retaining the discriminating potential of these labels.

This extension can be incorporated in the random walk kernel of Eq. (2.3) by using as edge weight matrix $\mathbf{W}_\times$ the values of the kernel $k_E : V'^2 \times V''^2 \to \mathbb{R}$ on any pair of edges (u', v') and (u'', v'') with $u', v' \in V'$ and $u'', v'' \in V''$. Let $n' = |V'|$. We can now formally write

$$[\mathbf{W}_\times]_{(i-1)n'+r,(j-1)n'+s} = k_E\big((v'_i, v'_j), (v''_r, v''_s)\big) ,$$

where the edge similarity kernel k_E can be defined as

$$k_E\big((u', v'), (u'', v'')\big) := \\ k_{adj}\big((u', v'), (u'', v'')\big) k_{struc}(u', u'') k_{struc}(v', v'') .$$

Here, k_{adj} is a kernel on graph edges and k_{struc} the structurally-aware vertex similarity kernel. In place of k_{adj} we can use any kernel subject to the constraints of Sec. 2.1.1: one with a known and low-dimensional representation. Nevertheless, for the sake of simplicity, we use the linear kernel on the graph adjacency entries; then the Gram matrix of this kernel becomes equal to the Kronecker product of the two adjacency matrices

$$k_{adj}\big((v'_i, v'_j), (v''_r, v''_s)\big) := [\mathbf{A}']_{i,j} [\mathbf{A}'']_{r,s} ,$$
$$K_{adj} = \mathbf{A}' \otimes \mathbf{A}''$$

Without loss of generality, we can consider the structural vertex kernel k_{struc} as a kernel k_{att} over the image of a feature map $l_G : V \to \mathcal{X}$ that extracts structure-aware properties from the vertices of each graph, i.e.,

$$k_{struc}(v', v'') := k_{att}\big(l_{G'}(v'), l_{G''}(v'')\big) .$$

As its Gramian goes over all pairs $V' \times V''$, it is rank 1:

$$K_{struc} = \mathbf{k}\mathbf{k}^\top , \text{ with}$$
$$[\mathbf{k}]_{(i-1)n'-r} := k_{struc}(v'_i, v''_r) . \tag{3.6}$$

Finally, the edge similarity matrix can be written as

$$\mathbf{W}_\times = (\mathbf{A}' \otimes \mathbf{A}'') \circ K_{struc} . \tag{3.7}$$

Importantly, the above formulations allow us to adapt the identity of Eq. (2.5) for our kernel.

Lemma 3.1. *The matrix-vector operation of the edge similarity matrix of Eq.* (3.7) *can be computed in* $O(n'^2 n'' + n' n''^2)$ *as*

$$\mathbf{W}_\times \mathbf{x} = \mathrm{vec}\big[\mathbf{T} \circ \big(\mathbf{A}''\big(\mathbf{T} \circ \mathrm{mat}[\mathbf{x}]\big)\mathbf{A}'^\top\big)\big] , \tag{3.8}$$

where $\mathbf{T} := \mathrm{mat}[\mathbf{k}]$ *is the matricisation of Eq.* (3.6).

For greater insight, we note that the matrix $\mathbf{T}$ can be regarded as the lower off-diagonal block of the vertex kernel Gramian, as applied on the concatenation of the vertices of the two graphs $[v'_1, \ldots, v'_{n'}, v''_1, \ldots, v''_{n''}]$.

$$[\mathbf{T}]_{i,j} = k_{struc}(v''_i, v'_j) = k_{att}\big(l_{G''}(v''_i), l_{G'}(v'_j)\big) .$$

When the involved vertices are not considered similar according to k_{struc} this matrix becomes sparse; then we can study the zeros of this matrix to avoid parts of the matrix-vector computations $\widetilde{K}_I \cdot \mathbf{x}$ that do not affect the final result.

[2]There are some graph classes and perturbation models for which this seems necessary if the structural similarity is defined by coreness [1].

COROLLARY 3.1. *Let at most τ pairs (v', v'') have a non-zero similarity according to k_{struc}, i.e., $|\{(v', v'') \in V' \times V'' \mid k_{struc}(v', v'') \neq 0\}| = \tau \leq n'n''$. Then the matrix-vector operation of Eq.* (3.8) *can be computed in $O\left((n' + n'')\,\tau\right)$.*

A special instance of the above arises when the structural attributes extracted from the graph vertices lie on a set of discrete scalars, and since l_G can be an arbitrary function, we can assume without any loss of generality that $l_G \in \{1, \ldots, L\} \equiv [L]$. Particular interest lies in the case when we use an integer kernel k_{att} that is of **bounded support**, i.e., when there exists a $\delta \geq 0$ such that $k_{att}(i, j) = 0$ for all $|i - j| > \delta$. This threshold δ is the kernel **bandwidth**.

We hence call **Structural Similarity random walk (SUSAN)** the random walk kernel that uses as vertex kernel k_{struc} a bounded support kernel k_{att} over integer-valued structural attributes derived from the vertices of each graph

$$k_{struc}(v', v'') = k_{\text{att}}\big(l_{G'}(v')\,, l_{G''}(v'')\big) k_{\text{v}}(v', v'')\ , \tag{3.9}$$

where k_{v} is either a pre-existing vertex kernel to be incorporated or the identity function, if so preferred. The resulting kernel remains both positive-definite and of bounded support. Due to its last property the kernel considers only potentially beneficial relations, while gaining special structure that enables a significantly more efficient computation.

We now study the useful structure of SUSAN and analyse its computational complexity. First, and without loss of generality, we assume the rows and columns of each adjacency matrix to be arranged so that they correspond to vertices in increasing order of $l_G(v)$. This gives rise to the block representation of the adjacency matrix $\mathbf{A}$ of graph G as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,L} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{L,1} & \cdots & \mathbf{A}_{L,L} \end{bmatrix}\ ,$$

where each block $\mathbf{A}_{i,j}$ contains the (possibly empty) block of all edges (u, v) from vertices with $l_G(u) = i$ to those with $l_G(v) = j$, and L is the maximal value of l_G. It will be of help to define the size of each block as $\mathbf{A}_{i,j} \in \mathbb{R}^{b_i \times b_j}$, where $b_i := |\{u \in V \mid l_G(v) = i\}|$. We can now compute the exact number of non-zero elements τ of $\mathbf{T}$, by observing that this matrix becomes a banded block matrix with block-bandwidth δ, and whose each block has equal elements.

LEMMA 3.2. *The matrix-vector operation $\mathbf{W}_\times \mathbf{x}$ for the SUSAN kernel whose vertex kernel has a bounded support with bandwidth δ can be computed in exactly*

$$c = (2n' + 2n'' + 1) \sum_{i=0}^{\kappa'} b'_i \sum_{j=\max(0,\lambda-\delta)}^{\min(\kappa'', i+\delta)} b''_j - n'n'' \tag{3.10}$$

floating point operations, which is $O\big((\delta + 1)(n' + n'')B^2\big)$, for B the maximal size among all b'_k, b''_k. In contrast, a naïve computation requires $2n'n''(n' + n'')$ such operations.

3.1 Selection of the Structural Attribute Although within the context of the SUSAN kernel we can plug in any integral structural attribute, the greatest efficiency comes with small bandwidths. For this reason, two natural such properties arise as particularly helpful: the vertex degree and its coreness, which can both be computed at virtually no cost.

As a side-note, the choice of the vertex kernel may also affect the matrix-vector complexity c. By using the vertex degree, a bound on the non-zero entries of each adjacency matrix row also arises, given the degree that each block belongs to. Combined with the bound for the non-zeros of the matrix $\mathbf{T}$, we can replace the innermost sum in Eq. (3.10) with $\sum_{j=\max(0,i-\delta)}^{\min(L'', i+\delta)} \min(i, b_i)$. In the case of coreness the quantity c can be bounded from below as $c \in o\big((n' + n'')\sqrt{n'}\sqrt{n''}\big)$, even with a bandwidth of $\delta = 0$, since there always exists a k-shell with at least $\sqrt{n'}$ vertices in any simple graph G' (see online appendix).

3.2 Selection of the Attribute Kernel Note that, in order to ensure positive definiteness for the SUSAN kernel, all its constituents must also be positive definite, and therefore also k_{att}. We hence define a class of bounded support positive definite kernels that can be used in the role of k_{att}, to assess the similarity of integer-valued structural vertex properties.

To fully specify k_{att} as a positive definite kernel it is sufficient to define its feature mapping ϕ on some Hilbert space. For a given bandwidth δ and an arbitrary **shape vector** $s \in \mathbb{R}^{\lfloor \delta/2 \rfloor + 1}$ we can define such a feature map $\phi_s^\delta : \mathbb{Z} \to \mathbb{R}^\omega$, whose image contains (countably) infinite-dimensional elements with indices $\lambda = \ldots, -3/2, -1, -1/2, 0, 1/2, 1, 3/2, \ldots$:

$$[\phi_s^\delta(i)]_\lambda := \begin{cases} s_{\lceil |i-\lambda| \rceil} & 2|i - \lambda| \leq \delta, \delta \text{ odd}, 2\lambda \text{ odd} \\ s_{|i-\lambda|+1} & 2|i - \lambda| \leq \delta, \delta \text{ even}, \lambda \in \mathbb{Z} \\ 0 & \text{otherwise}\ . \end{cases}$$

The image of ϕ_s^δ is a (countably) infinite-dimensional vector space that becomes Hilbert through the natural inner product. Thus, a positive definite kernel k_{att} over $\mathbb{Z}$ can be defined as

$$k_{\text{att}}(i, j) := \langle \phi_s^\delta(i)\,, \phi_s^\delta(j) \rangle_{\mathbb{R}^\omega} := \sum_{\lambda=-\infty}^{\infty} [\phi_s^\delta(i)]_\lambda \cdot [\phi_s^\delta(j)]_\lambda\ , \tag{3.11}$$

and can be easily verified that it has bounded support with bandwidth δ. Additionally, it also belongs to the class of **shift invariant kernels**; i.e., its value only depends on the difference of its entries: $k_{\text{att}}(i, j) = k_{\text{att}}(|i - j|)$, where we slightly abused notation. Since we assume that the structural properties are captured as an integer by the structural labels

l_G, shift-invariance is a natural property that avoids making any further assumptions on these structural properties.

Note that not all shift-invariant bounded support functions are positive definite, but any function in the form of Eq. (3.11) is. Out of these, arguably the simplest one arises when s is the constant vector with elements $s_i = \frac{1}{\sqrt{\delta+1}}$. This choice yields the kernel whose graph resembles a triangle

$$k_{\text{att}}(i,j) = \max\left(0, 1 - \frac{|i-j|}{\delta+1}\right),$$

and is also the one we adopt to complete Eq. (3.9).

3.3 Computation of SuSAN Among the contributions of this work is a proof-of-concept implementation of the main BLAS3 components of the matrix-vector operation $\mathbf{W}_\times \mathbf{x}$ of the SuSAN. This implementation uses the key observation formalised in Lemma 3.2, and therefore has a complexity that is upper bounded by that of its naïve computation.

We recall from Sec. 2.1.1 that fast algorithms are available for the vanilla random walk kernel. However, when using a vertex and/or edge kernels, like in the case of SuSAN, these methods cannot be applied, since the similarity matrix $\mathbf{W}_\times$ does not have a Kronecker decomposition. Then iterative methods have to be used, all of which rely on an efficient computation of the matrix-vector operation $\mathbf{W}_\times \mathbf{x}$.

We complete the work of [20] by applying the iterative method of Al Mohy et al. [2] to compute the exponential version of SuSAN. This method involves a truncated Taylor expansion, in which the order is computed for the required accuracy based on a bound on the norms of the matrix. We empirically study the convergence of this method in Sec. 4.1.

With this we establish that both practical algorithms for the computation of the SuSAN kernel benefit from a more efficient implementation of this matrix-vector operation.

Note that an additional advantage of our algorithm is that it only needs to store the τ elements of the $\mathbf{x}$ vector. This not only improves the cache locality of the data during computation, especially in the case of small δ, but can also improve the convergence of the used solver (c.f., Sec. 4.1).

4 Experiments

We now evaluate our proposed algorithm to compute the random walk kernel for bounded-support vertex kernels, as well as the utility of several structural similarity measures within this framework. In particular, we consider coreness, vertex degrees, and application specific structural vertex similarities. SuSAN is implemented[3] in C++ and Python.

As an example of data with application specific vertex similarities, we use three brain connectome datasets [15]: these represent connections between brain regions (as vertices) that are consistently labelled by integers across patients. Furthermore, close-by labels indicate functional proximity[4]. These, as well as the other datasets considered below are publicly available at Morris et al. [12].

4.1 Efficiency To evaluate the theoretical advantage of our implementation on real-world datasets, we first compare our algorithm for computing SuSAN against a random walk kernel with a vertex kernel of unbounded support. To this end, we compute the vanilla random walk kernel with an appropriate iterative method for each kernel kind (see Sec. 3.3), as the faster methods of [20] are then inapplicable. We hence attain an implementation-independent measurement by comparing the time required to compute SuSAN using the same implementation for both our algorithm and the baseline matrix-vector operation. For simplicity, and since the matrices of these datasets are of small dimension, we assume full matrix storage. We do note, however, that highly optimised implementations of the baseline matrix–vector operation (MVO) could outperform our proof-of-concept algorithm in certain hardware, despite its theoretical superiority.

On each brain connectome dataset, and for an increasing bandwidth parameter δ, we use the iterative schemes of Sec. 3.3 to compute both the exponential and the geometric SuSAN kernels. In the top row of Fig. 3 we compare the elapsed wall-clock time when using our proof-of-concept implementation of the matrix-vector product against the baseline computation to compute the same kernel. We see that for both kernels and for small bandwidths a speedup of up to an order of magnitude is attained; with increasing δ the runtime slows down to that of the naïve algorithm.

Although the key contributing factor in the more efficient computation is a faster MVO computation, we also study the required number of iterations as a potential secondary factor of efficiency. Therefore, in the bottom row of Fig. 3 we compare the average number of MVOs for SuSAN against those for the vanilla random walk kernel, using the same iterative method as above. We observe that the conjugate gradient solver (geometric variant) converges faster for the lower bandwidth vertex kernels, primarily due to the lower dimension of the problem in the case of SuSAN. Since, however, the difference in required MVOs is smoother than the running time, there seems to be an additional factor in play. We conjecture this to be the low-pass property of the vertex kernel of SuSAN, which seems to impose more smoothness in the Krylov Space that this solver uses.

4.2 Accuracy Next we investigate the predictive performance of bounded bandwidth structural vertex similarities

[3] Available at https://eda.mmci.uni-saarland.de/susan

[4] Vertices with similar labels have a much higher than expected probability of being connected by an edge in these datasets, which in turn implies a high correlation between the EEG time series of the two regions. We observe (cf. Sect. 4.2) that a nontrivial bandwidth on the labels increases the kernel performance.

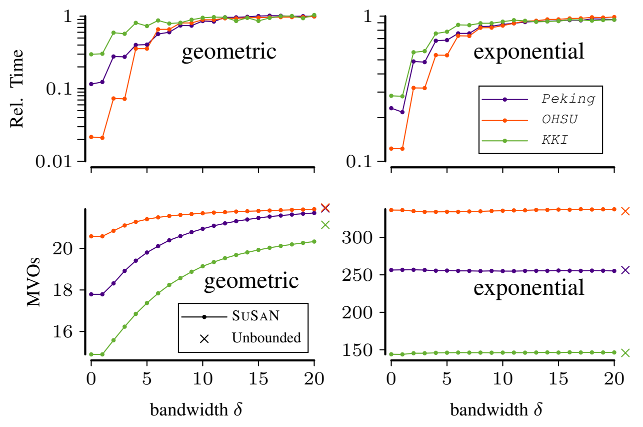

Figure 3: [Top] Relative time performance of SuSAN vs. random walk with unbounded vertex kernel (100%). [Bottom] Absolute number of matrix-vector operations (MVOs) required for the iterative computation of SuSAN.

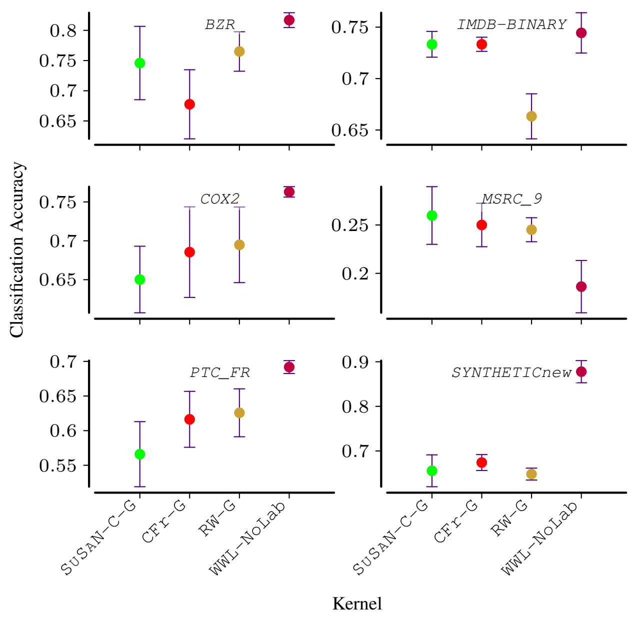

Figure 4: Classification accuracy and standard deviations of SVM classifiers on benchmark datasets for the geometric variants. Provided for the sake of completion, as these datasets do not yield rich enough structural vertex properties.

combined with random walk kernels. We show that using a bounded bandwidth kernel does not deteriorate performance compared to the vanilla random walk kernel and other related state-of-the-art graph kernels. We instantiate the (exponential resp. geometric) random walk kernel with integer kernel on vertex labels as described in Sect. 3.2 using coreness (SuSAN-C-E, resp. SuSAN-C-G), vertex degree (SuSAN-Deg-E, SuSAN-Deg-G), and Weisfeiler-Lehman labels (SuSAN-WL-E, SuSAN-WL-G). We compare against vanilla random walk kernels (RW-E, RW-G), the Core Framework kernel (CFr-E, CFr-G) [14], as an alternative to combine random walks with degeneracy, and against the Wasserstein-Weisfeiler-Lehman kernel [18] as a representative state-of-the-art alternative, without using vertex labels (WWL-NoLab).

We evaluate each candidate kernel by the accuracy of a C-SVM classifier [4] equipped with it. We compute random 80%/20% train/test splits, using stratified random sampling. On each training set we use a 3-fold cross validation to identify optimal hyper-parameters of each kernel using a grid search with 15 samples. For every random walk kernel we search the λ parameter in the range $\lambda \in [10^{-6}, 5]$ for the exponential and in $\lambda \in [10^{-5}, 1]$ for the geometric variants. For the WWL kernels we use the values $\lambda \in [10^{-5}, 10]$, as suggested by the authors. The SVM regularisation parameter is selected from $C \in [10^{-3}, 10^{5}]$. Both λ and C are sampled using a logarithmic scale. For the truncated versions of our kernels we use grid search over the set $\delta \in \{0, 1, 2, 3, 4, 5, 10, 15\}$.

For the sake of completeness, we first report results on standard graph kernel benchmark datasets in Fig. 4. Our kernel is never significantly worse than the Core Framework or the vanilla random walk kernel on these datasets. However, it is significantly better than the latter on the *IMDB-Binary* dataset ($p = 0.012$). This indicates that the computation can be sped up without hurting predictive performance.

Fig. 5 reports results of our kernel variants and competitors on brain connectome graphs. It shows that our bounded bandwidth kernel with Weisfeiler-Lehman labels achieves highest predictive performance on average on *OHSU*, our kernel with coreness highest performance on *Peking_1* and its geometric variant on *KKI*. To estimate the statistical significance of these results, we repeated each process from the beginning on a fresh split of each dataset, for 30 iterations and use Welch's two sample t-test with significance level $p = 0.05$. Table 1 shows the p-values of two tests, comparing SuSAN-C-G and SuSAN-C-E to the competitors. There is no statistically significant performance difference between the SuSAN variants and the vanilla random walks, except for the Weisfeiler-Lehman based kernels that perform (comparatively) well on *OHSU*, and poorly on *Peking_1*. When compared to the Core framework kernel, SuSAN seems mostly on par with it, except on *Peking_1* where CFr-E performs poorly in comparison, and *KKI*, where SuSAN-C-G is significantly better than its core-framework equivalent ($p = 0.0027$). Due to the increased computational effort of the core framework (cf. Sect. 5) using our kernel and coreness is hence beneficial.

Finally, to assess the usefulness of degeneracy as a vertex label extractor in the case of no label information, we

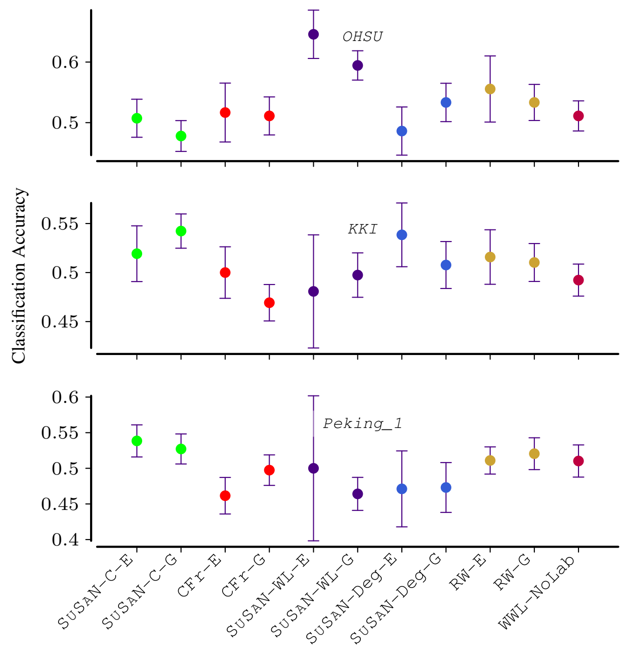

Figure 5: Classification accuracy and standard deviations of SVM classifiers on brain connectome datasets.

also compare it against the WWL kernel without label information (`WWL-NoLab`). We find that SuSAN performs well when compared with other methods without vertex information, and even significantly outperforms the state-of-the-art for the *KKI* (p-value=0.02). To the same end, we also report the results for the WWL kernel, in which we replace the default use of degrees with coreness (`WWL-Core`), with occasionally better performance.

For the sake of completeness we also compare against the full WWL (`WWL`). Unsurprisingly, `WWL` outperforms SuSAN, since the latter is not given access to label information.

5 Related Work

Gärtner et al. [5] introduced Random Walk kernels, initially with a runtime of $O\left(n^6\right)$. Notably, their kernel already allows to incorporate integral vertex labels in a way that corresponds to the trivial bandwidth $\delta = 0$ in our setting. Vishwanathan et al. [19, 20] proposed several iterative methods which reduced the computational complexity to $O\left(n^3\right)$. They generalized this to a framework for random walk kernels which can incorporate nonuniform starting and stopping probabilities on vertices, different weights for different walk lengths, node and edge similarities encoded as kernels, as well as the possibility to incorporate edge labels. However, their framework cannot benefit from bounded support vertex kernels. Kang et al. [8] propose to speed up random walk graph kernel computation using a low rank approximation of the adjacency matrix $\mathbf{W}_\times$ of the product graph, which in turn can be computed implicitly by low rank approximations of $\mathbf{A}'$ and $\mathbf{A}''$. They obtain a runtime of $O\left(\kappa n^2 r^4 + r^6 + mr\right)$ for κ different labels, where r is the rank parameter. In contrast, our method is exact (up to numeric precision) and runs in $O\left(\delta n B^2\right)$ for bandwidth $\delta \leq \kappa$ and $B \leq n$.

Structural vertex labels have been used before to define graph kernels. For a recent broad-scoped survey, see Kriege et al. [9]; we focus here on the combination of random walk kernels and structural properties. Mahé et al. [11] first propose to use *Morgan indices* as vertex labels in random walk kernels. Nikolentzos et al. [14] introduce a graph kernel framework based on core decompositions. They suggest to compute kernels of the form $k(G', G'') = \sum_{k=0}^{\infty} k_{\text{base}}(H(k)', H(k)'')$ explicitly, which increases the complexity of the kernel computations by a factor κ. In this work, however, we use the structural information given by the core decomposition to speed up the kernel computations. We thus don't fall into their framework; notably, our kernel tends to get faster when many core values are present and allows more fine-grained control over vertex alignments.

6 Discussion

We provide a fast way to exactly compute the random walk kernel for bounded support integer kernels on vertex labels, allowing to align structurally similar vertices, while improving the runtime of the method. Our experiments show that our method is faster than a similarly optimized version of the generic random walk kernel and that it achieves a significant improvement on some benchmark datasets.

As an application, we have evaluated the suitability of coreness (or degeneracy) in random walk kernels when vertex label information is not available or noisy. Another suitable application concerns node aligned graphs, i.e., graphs that share a common set of vertices but have different edge sets. If a total order on vertices is known, e.g., a space filling curve for vertices residing in Euclidean space, it is easy to include some smoothing via a bounded bandwidth integer kernel. This would allow to additionally consider alignments of non-identical but very similar vertices. In this case, choosing a mapping from vertices to integers and constant bandwidth allows to seamlessly use this alignment information in the random walk kernel, resulting in linear runtime (in the input graph size) for the matrix vector multiplication in Lemma 3.2. A further structurally aware alignment option are Morgan indices [cf. 11] which measure the sizes of (generalized) vertex neighbourhoods.

Our proposed structurally aware kernel framework allows to incorporate as much alignment information as is available and is faster, the more information there is. This ranges from quadratic (if all alignment information is available) to cubic runtime (if no alignment is available, corresponding to the general random walk kernel.

dataset	SuSAN type	CFr-E	CFr-G	SuSAN-Deg-E	SuSAN-Deg-G	RW-E	RW-G	SuSAN-WL-E	SuSAN-WL-G	WWL	WWL-Core	WWL-NoLab
KKI	SuSAN-C-E	0.3115	0.0738	0.6694	0.3788	0.4781	0.3976	0.0525	0.2757	0.7940	0.3664	0.2080
	SuSAN-C-G	0.0983	**0.0027**	0.4594	0.1248	0.2148	0.1114	**0.0064**	0.0609	0.6112	0.0854	**0.0199**
OHSU	SuSAN-C-E	0.8820	0.5345	0.3420	0.7192	0.6919	0.7253	0.9045	0.9834	0.9930	0.8986	0.5382
	SuSAN-C-G	0.9598	0.7924	0.5669	0.9101	0.8603	0.9187	0.9745	0.9992	0.9998	0.9901	0.8226
Peking_1	SuSAN-C-E	**0.0111**	0.0690	0.1306	0.0517	0.1131	0.2623	0.3184	**0.0060**	0.9457	**0.0153**	0.1609
	SuSAN-C-G	**0.0302**	0.1633	0.1766	0.0973	0.2605	0.4148	0.4074	**0.0243**	0.9659	0.0592	0.2932
KKI	SuSAN-C-E	0.6885	0.9262	0.3306	0.6212	0.5219	0.6024	0.9475	0.7243	0.2060	0.6336	0.7920
	SuSAN-C-G	0.9017	0.9973	0.5406	0.8752	0.7852	0.8886	0.9936	0.9391	0.3888	0.9146	0.9801
OHSU	SuSAN-C-E	0.1180	0.4655	0.6580	0.2808	0.3081	0.2747	0.0955	**0.0166**	**0.0070**	0.1014	0.4618
	SuSAN-C-G	**0.0402**	0.2076	0.4331	0.0899	0.1397	0.0813	**0.0255**	**0.0008**	**0.0002**	**0.0099**	0.1774
Peking_1	SuSAN-C-E	0.9889	0.9310	0.8694	0.9483	0.8869	0.7377	0.6816	0.9940	0.0543	0.9847	0.8391
	SuSAN-C-G	0.9698	0.8367	0.8234	0.9027	0.7395	0.5852	0.5926	0.9757	**0.0341**	0.9408	0.7068

Table 1: Listing of p-values for the Welch t-test: comparing the performance of SuSAN (using coreness) against all other kernels on the brain connectome datasets. Null hypotheses: [*top*] SuSAN is not better than X, [*bottom*] SuSAN is not worse that X. That is, a **blue** value [top] indicates that SuSAN significantly outperforms X, whereas a **red** value [bottom] means SuSAN is significantly outperformed by X. We note that WWL has access to vertex labels, and thus gets an unfair advantage.

Acknowledgements PW was partially funded by the Federal Ministry of Education and Research of Germany under grant 01|S18038C.

References

[1] A. Adiga and A. K. S. Vullikanti. How Robust Is the Core of a Network? In *ECMLPKDD*, 2013.

[2] A. H. Al-Mohy and N. J. Higham. Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators. *SIAM J. Sci. Comp.*, 2011.

[3] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics'05*.

[4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *JMLR*, 2008.

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

[6] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California - Santa Cruz, 1999.

[7] J. Kalofolias, M. Boley, and J. Vreeken. Discovering robustly connected subgraphs with simple descriptions. In *ICDM*, 2019.

[8] U. Kang, H. Tong, and J. Sun. Fast random walk graph kernel. In *SDM*, 2012.

[9] N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 2020.

[10] C. F. V. Loan. The ubiquitous Kronecker product. *J. Comput. Appl. Math.*, 2000.

[11] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of Marginalized Graph Kernels. In *ICML'04*.

[12] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *GRL+@ICML*, 2020. arXiv:2007.08663.

[13] G. Nikolentzos and M. Vazirgiannis. Random Walk Graph Neural Networks. In *NeurIPS*, 2020.

[14] G. Nikolentzos, P. Meladianos, S. Limnios, and M. Vazirgiannis. A Degeneracy Framework for Graph Similarity. In *IJCAI*, 2018.

[15] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang. Task Sensitive Feature Exploration and Learning for Multitask Graph Classification. *IEEE Trans. Cybern.*, 2017.

[16] T. H. Schulz, T. Horváth, P. Welke, and S. Wrobel. A generalized weisfeiler-lehman graph kernel. 2021. arXiv:2101.08104.

[17] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 1983.

[18] M. Togninalli, M. E. Ghisu, F. Llinares-López, B. Rieck, and K. M. Borgwardt. Wasserstein Weisfeiler-Lehman graph kernels. In *NeurIPS*, 2019.

[19] S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In *NeurIPS*, 2006.

[20] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph Kernels. *JMLR*, 2010.

[21] P. M. Weichsel. The Kronecker product of graphs. *AMS Proc.*, 1962.