Abstract

Transformers were originally proposed as a sequence to sequence model for text but have become vital for a wide range of modalities, including images, audio, video, and undirected graphs. However, transformers for directed graphs are a surprisingly underexplored topic, despite their applicability to fundamental domains including source code and logic circuits. While spectral encodings based on the combinatorial Laplacian relate to the sinusoidal positional encodings for ordered sequences, they cannot capture directedness in graphs. To overcome this limitation, we propose to use the Magnetic Laplacian – a direction-aware generalization of the combinatorial Laplacian. Empirically, we show that the extra directionality information is useful in various downstream tasks, including correctness testing of sorting networks and source code understanding. Together with a data-flow-centric graph construction, our model outperforms the prior state of the art on the Open Graph Benchmark Code2 relatively by 14.7%.1

1 Introduction

Transformers have become the central component in many state-of-the-art approaches spanning a wide range of modalities. For example, transformers are used to generate solutions for competitive programming tasks from textual descriptions (Li et al. 2022) or find approximate solutions to combinatorial optimization problems like the Traveling Salesman Problem (Kool, Hoof, and Welling 2019). Transformers also have had first success on graph learning tasks (Min et al. 2022). The biggest challenge with graphs as input modality is arguably how the attention mechanism can become structure-aware. Prevalent strategies include (1) modifying the attention mechanism itself to incorporate structural information (Ying et al. 2021), (2) hybrid architectures that also contain Graph Neural Networks (GNNs) (Mialon et al. 2021; Chen, O’Bray, and Borgwardt 2022), and (3) approaches relying on positional encodings for graphs (Dwivedi and Bresson 2021).

We find that transformers for directed graphs have received surprisingly little attention so far, despite their applicability to important tasks, including graph representations for source code. Specifically, most of the literature for structure-aware positional encodings (3) either uses basic measures like pairwise shortest path distances (Guo et al. 2020) or symmetrizes the graph for theoretically motivated positional encodings, e.g., based on graph spectral theory (Dwivedi and Bresson 2021). Importantly, symmetrizing the graph neglects any information about the edges’ direction and therefore comes with severe limitations. For this reason, we propose to use the eigenvectors of the Magnetic Laplacian (§ 3), a natural direction-aware generalization of the well-known combinatorial Laplacian for undirected graphs (see Fig. 1). We argue that such a positional encoding is an important step towards general-purpose transformers that universally handle both undirected and directed graphs (see Fig. 4 for architecture).

We show that directional information can be beneficial for downstream tasks. We study two semantically rich high-level reasoning tasks. Namely, function name prediction on the Open Graph Benchmark (OGB) Code2 dataset (Hu et al. 2020) and correctness prediction of sorting networks (see § 4). Sorting networks are an old problem in theoretical computer science (Knuth 1973) and here the goal is to predict whether a given network is correct, i.e., sorts all sequences of a fixed length. Studying sorting networks also reveals important intuitions about the importance of directed graphs: (a) symmetrization maps both correct and incorrect sorting networks to the same input graph. (b) a sequence of program instructions can be understood as a single topological sort of the underlying directed graph. This allows quantifying how many equivalent programs map to the same graph. In other words, directed graphs can drastically reduce the effective input dimensionality. The topological sort analogy is an important insight since treating source code as a sequence is still the de facto standard. These insights motivate us to rethink the graph construction for source code (see § 5).

Figure 1: First eigenvector $\gamma_0$ of Magnetic Laplacian (b also comb. Lap.). Node size shows real and color imaginary value.
Contributions. (I) We propose spectral positional encodings that also generalize to directed graphs. (II) We introduce the task of predicting the correctness of sorting networks, a canonical ambiguity-free application where directionality is of the essence. (III) We set a new state on the art for the OGB Code2 dataset (2.85% higher F1 score, 14.7% relatively).

2 Sinusoidal and Laplacian Encodings

It is commonly argued that the eigenvectors of the (combinatorial) Laplacian generalize the sinusoidal positional encodings, due to their relationship via the Graph Fourier Transformation (GFT) and Discrete Fourier Transformation (DFT). Even though sinusoidal positional encodings capture the direction, eigenvectors of the Laplacian do not. We next discuss their relationship and differences for a sequence (Fig. 1a,b).

Sinusoidal encodings \( PE_{v,2k} := \cos(v/10,000 2k/d_{\text{model}}) \) and \( PE_{v,2k+1} := \sin(v/10,000 2k/d_{\text{model}}) \) encode the position of the first token/node are non-negative. However, for general same sign per eigenvectors of the Laplacian and (2) the basis. We order eigenvalues and eigenvectors s.t. guaranteed to be diagonalizable and that \( A \in \{0, 1\}^{n \times n} \). Symmetrization is required s.t. \( \Gamma \) form an orthogonal basis. We next discuss their relationship and differences for a sequence (Fig. 1a,b).

Eigenvectors of Laplacian. Similarly, we can define a “Graph Fourier Transformation” via the eigendecomposition of the combinatorial Laplacian \( L = \Gamma \Lambda \Gamma^{-1} \), with diagonal matrix \( \Lambda \) of eigenvalues and orthogonal matrix \( \Gamma \) of eigenvectors. \( \Gamma \) can be used as pos. encodings. The real, symmetric, and positive semi-definite unnormalized Laplacian \( L_U \) as well as degree-normalized Laplacian \( L_N \) are defined as:

\[
\begin{align*}
L_U &= D_S - A_S \quad (2) \\
L_N &= I - (D_S^{-1/2} A_S D_S^{-1/2})^{\top} \quad (3)
\end{align*}
\]

are defined based on the diagonal degree matrix \( D_S \) of the symmetrized adjacency matrix \( A_S = A \vee A^\top \) with \( A \in \{0, 1\}^{n \times n} \). Symmetrization is required s.t. \( L \) is guaranteed to be diagonalizable and that \( \Gamma \) form an orthogonal basis. We order eigenvalues and eigenvectors s.t. \( 0 \leq \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1} \). We call \( \lambda_0 \) or \( \lambda_{0,0} \) the first eigenvalue and \( \gamma_0 \) or \( \Gamma_{0,0} \) the first eigenvector.

Two notable differences to the DFT are (1) the real-valued eigenvectors of the Laplacian and (2) the sign invariance. Specifically, if we consider a sequence (Eq. 6), the eigenvectors are given by the Cosine Transformation Type II: \( \Gamma_{j,k} = \pm \cos((j + 1/2)\pi/k) \), where we must choose the same sign per \( k \) or, equivalently, an eigenvector. The Cosine Transformation typically fixes \( \pm 1 \) to +. Thus, the encodings of the first token/node are non-negative. However, for general graphs it is not that simple to resolve this ambiguity (e.g. multiple sink and source nodes). Thus, we typically use an arbitrary sign for each \( \gamma \) (Dwivedi and Bresson 2021).

Due to this sign invariance, the eigenvectors \( \gamma \) of the combinatorial Laplacian cannot distinguish the direction. Alternatively, from symmetrization, we see that distinguishing direction is not possible. That is, we obtain the same symmetrized graph regardless of how we reverse edges.

3 Directional Spectral Positional Encodings

For the ability to distinguish direction, we propose to use the Magnetic Laplacian, a generalization of the combinatorial Laplacian. The Magnetic Laplacian is a Hermitian matrix that encodes the direction of edges with complex numbers. This way we can have the best of both worlds: (1) the eigenvectors form an orthogonal basis and (2) we capture the directionality.

The Magnetic Laplacian is defined as

\[
L_U^{(q)} := D_S - A_S \oslash \exp \left( i \Theta^{(q)} \right) \quad (4)
\]

with Hadamard product \( \oslash \), element-wise \( \exp \), potential \( q \) and \( i = \sqrt{-1} \). Recall, \( D_S \) is the symmetrized degree matrix and \( A_S \) the symmetrized adjacency matrix. Except for the exp of

\[
\Theta^{(q)}_{u,v} := 2\pi q (A_{u,v} - A_{v,u}), \quad q \geq 0 \quad (5)
\]

Eq. 4 is equivalent to the Laplacian \( L_U := D_S - A_S \). The Magnetic Laplacian for a sequence with \( q = 0 \) and \( q = 1/4 \) are given next as well as their first eigenvc. in Fig. 1b and a.

\[
L_U^{(q)} := \begin{bmatrix}
1 & -1 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix} \quad (6) \quad L_U^{(1/4)} := \begin{bmatrix}
1 & -1 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix} \quad (7)
\]

In our experiments, we use the degree-normalized counterpart \( L_N^{(q)} := I - (D_S^{-1/2} A_S D_S^{-1/2})^{\top} \oslash \exp \left( i \Theta^{(q)} \right) \). The exp \( (i \Theta^{(q)}) \) term encodes the direction of edges. It resolves either to \( 1 \) if an edge exists in both directions \( (A_{u,v} = A_{v,u} = 0) \) or to \( (\pm 2\pi q) \), with the sign encoding the edge direction. The potential \( q \) determines the ratio of real and imaginary part (recall \( \exp(i\alpha) = \cos(\alpha) + i \sin(\alpha) \)). \( A_{u,v} = A_{v,u} = 0 \) is not of interest due to the multiplication with the zero entries in \( A_S \) or \( D_S^{-1/2} A_S D_S^{-1/2} \). The Magnetic Laplacian contains the comb. Laplacian as a special case \( (q = 0) \). If the graph is undirected (i.e., \( A = A^\top \)), we obtain the comb. Laplacian for \( q \leq 0.25 \). However, directed edges are purely imaginary if \( q = 1/4 \) (see Eq. 7).

Although the Magnetic Laplacian has been used for a Spectral GNN (Zhang et al. 2021), Community Detection (Fanuel et al. 2016) or visualization of directed graphs (Fanuel et al. 2018), there is not much guidance on how to choose \( q \) and prior work in the computer science literature treats \( q \) as a (hyper-) parameter. Fortunately, the first eigenvector \( \gamma_0 \) of Eq. 4 solves a relaxation of the so-called “angular synchronization problem” (Bandeira, Singer, and Spielman 2013):

\[
\angle(\gamma_0) \in \arg \min_{\alpha \in [0,2\pi)} \eta(\alpha) \quad \text{with} \quad \eta(\alpha) = \sum_{u,v \in E} \left| \exp(i \alpha_u) - \exp(i \Theta_{u,v}) \right|^2 \quad (8)
\]

In angular synchronization, we seek the \( L^2 \) optimal estimate of \( n \) angles \( \alpha \) given \( m \) (noisy) measurements of phase offsets \( \alpha_i - \alpha_j \mod 2\pi \) where \( i, j \in \{0, 1, \ldots, n-1\} \). Hence, each directed edge \((u,v)\) encourages a rotation of the (otherwise constant) first eigenvector between nodes \( u \) and \( v \). If there are no contradicting edges (i.e., in a tree) and node \( u \) is an ancestor of \( v \), then the total phase shift amounts to \( 2\pi q h \) where \( h \) is the number of steps to reach \( v \) from \( u \). Additionally, if \( A_{u,v} = A_{v,u} = 1 \), a synchronization of the “angle difference” between nodes \( u \) and \( v \) is encouraged.
We choose $q = q'/\min(m,n,1)$, with relative potential $q' = 1/4$ and number of purely directed edges $\bar{m} = \{(u,v) \in E \mid (v,u) \notin E\}$. Then, the maximal angle difference between any two nodes is bounded $\angle(\Gamma_{v,0}, \Gamma_{u,0}) \leq \frac{\pi}{2}$. In other words, the longest possible simple path is of length $\min(m,n)$ (called $h$ above). We empirically verify the choice of $q'$ in Fig. 6. $q' = 1/4$ is among the best and for high values of $q$ the performance drops severely (absolute $q > 0.05$).

**Scale and rotation.** Eigenvectors are neither guaranteed to be unique nor does Eq. 8 imply an absolute angle for any node (we only have relative "measurements"). In fact, if $\gamma$ is an eigenvector of $L$ then so is $c\gamma$, even if $c \in \mathbb{C}$ with $|c| > 0$ (proof: $cL\gamma = c\lambda\gamma \Rightarrow L(c\gamma) = \lambda(c\gamma)$). For real eigenvectors, there is the convention to choose $c \in \mathbb{R} \setminus \{0\}$ (possibly $c < 0$). Similarly, we choose the sign and rotate the eigenvectors according to a convention, that we find to work sufficiently well in our experiments. Our convention fixes the rotation for one of the source nodes — much like for sinusoidal encodings. First, we choose the maximum real magnitude of each eigenvector to be positive. This resolves the sign-ambiguity for the lowest eigenvector $\gamma_0$. For the other eigenvectors, this convention fails to resolve the sign ambiguity if minimum and maximum have identical magnitude. Then, we rotate all eigenvectors s.t. $\gamma_{v,u} = 0$ where we index a source node $v$ based on the first eigenvector $u = \arg\max_u \gamma_{v,u}$ (with imaginary component 3). Here we leverage our choice $q' \leq 1/4$, s.t., no neighboring node overflows the range $[-\pi, \pi]$ before determining $u$.

**MagLapNet.** Similarly as prior approaches (Lim et al. 2022; Kreuzer et al. 2021), we also preprocess the eigenvectors before using them as positional encodings. We consider the eigenvectors associated with the $k$ lowest eigenvalues $\Gamma_{\leq k}$ and treat $k$ as hyperparameter. To tackle the sign ambiguity of eigenvectors $\Gamma$, we follow a similar approach as SignNet (Lim et al. 2022) (excluding first eigenvector $\gamma_0$). SignNet is invariant to sign changes since each eigenvector is processed like $f_{\text{elem}}(-\gamma_j) + f_{\text{elem}}(\gamma_j)$, where $f_{\text{elem}}$ is permutation invariant over the nodes (e.g. MLP on each node or GNN; we use an MLP). The crucial piece is that $\gamma_j$ is a complex number and that real and imaginary parts are processed jointly via $f_{\text{elem}}$. We stress that we use a lightweight network using $\approx 0.5\%$ of the total # parameters. After this first block, we stack the different components associated to each node and apply LayerNorm, Self-Attention, and Dropout. Similarly, to Kreuzer et al. (2021) we apply the self-attention independently for each node over the $k$ eigenvectors. The last reshape stacks the positional encodings associated with each node and the MLP $f_{\text{re}}$ matches the transformer dimensions.

**4 Application: Sorting Networks’ Correctness**

Sorting networks (Knuth 1973) are directed graphs for a certain class of comparison based algorithms that sort any input sequence of fixed size with a sequence of comparators. Sorting networks are a particularly interesting application since they mark the middle ground between logical statements and source code. In fact, asserting their correctness is related to satisfiability. Interestingly, we use the sorting network task to make the implications of symmetrization (Laplacian encodings) and sequentialization (sinusoidal encodings) explicit.

We consider sorting networks that consist of a sequence of conditional exchange instructions. In Fig. 2a, each horizontal line represents an element of the sequence of variables and the vertical lines are the comparators. Thus, a sorting network can also be expressed as a sequence of length $n$ consisting of $v_i \leftarrow v_j = \text{sorted}(v_i, v_j)$ statements, where $v_i$ and $v_j$ are two of the $p$ variables, i.e. $i,j \in \{0,1,\ldots,p-1\}$. In our graph construction (Fig. 2b), we treat every instruction as a node with $i$ and $j$ as features (sinusoidal encoding). If a node operates on indices $i$ and $j$, we add an edge from the last occurrences of $i$ and $j$ (if there are any). Thus, all nodes have exactly two incoming and two outgoing edges, except for the source and sink nodes. Due to the in-place operations or vice versa lack of buffers, the edges are typically connected to close ancestor nodes.

**Directed graph vs. sequence.** An important intuition is that each topological sort of the directed graph is an equivalent "program", which corresponds to a particular ordering of the statements, and there are many such semantically equivalent “programs” for the same graph. In Fig. 3, we show the number of topological sorts for a type of compact and deterministically constructed sorting networks for different input sequence lengths. For such a network and sequences length of just 8, the number of equivalent sequentializations already exceeds 1 million. Representing directed graphs as sequences, therefore, introduces a huge amount of arbitrary undesirable orderedness. Hence, using directed graph representations can significantly reduce the size of the input space compared to using a sequence representation as many equivalent sequences are collapsed into just one directed graph.

**Symmetrization hurts.** There exist correct and incorrect sorting networks that map to the same undirected graph. A model using such an undirected graph cannot distinguish these cases. For example, the correct sorting network for length three with comparators $\{(0,2), (0,1), (1,2)\}$ and its reversed version (incorrect) map to the same undirected graph.

**Dataset.** We construct a dataset consisting of 800,000 training instances for equally probable sequence lengths $7 \leq p_{\text{train}} \leq 11$, generate the validation data with $p_{\text{val}} = 12$, and assess performance on sequence lengths $13 \leq p_{\text{test}} \leq 16$. We construct the sorting networks greedily until we have a correct sorting network. For this, we draw a random pair of comparators, excluding immediate duplicates and compara-
We compare our Magnetic Laplacian (ML) positional encoding with just one exception. Thus, without bells and whistles, the Magnetic Laplacian outperforms all other encodings. At first glance, it appears surprising that including sinusoidal positional encodings helps (recall a directed graph reflects many equivalent sequences). We hypothesize that there are complementary challenges in understanding the directed graph in comparison to a single manifestation of the program. Having both information appears to bring certain synergies that the model can leverage for better generalization. The biggest drawback of adding sinusoidal encodings is that we sacrifice the robustness w.r.t. “meaningless reorderings”. In other words, one has to trade-off accuracy and robustness.

### 5 Application: Function Name Prediction

We consider the task of function name prediction since it is an established task in the graph learning community (Hu et al. 2020) where the direction of edges is important. Similar to sorting networks, each program is one particular ordering of the program statements, and there could be many equivalent programs with different orderings. Thus, it is surprising that graphs for source code used for machine learning are inherently sequential. Specifically, most (if not all) graphs in prior work connect nodes sequentially to reflect the order of statements in the source. For example, the Open Graph Benchmark Code2 dataset represents the 450,000 functions with its Abstract Syntax Tree (AST) and sequential connections. Since the input space of sequences can be much larger than the input space of directed graphs (see § 4), for some tasks such a graph construction is an unfortunate choice.

#### Our Graph Construction

Inspired by Bieber et al. (2022). They also connect instructions sequentially, albeit, they handle control flow. Notable differences are: (a) We instead construct a Directed Acyclic Graph (DAG) for each “block” (e.g. body of if statement) that reflects the dependencies between instructions. We then connect the statements between blocks considering the control flow. (b) we address the (non-) commutative properties for basic python operations via edge features; (c) we do not reference the tokenized source code; (d) we omit the “last read” edges since these edges introduce unnecessary sequentialism; (e) otherwise we construct the graph similarly to OGB Code2 for comparability. For example, we aggregate child nodes containing only
W/o GNN, we find that random walks and the AST depth positional encodings based on the Magnetic Laplacian help to improve the predictive performance. For the Code2 graphs, the GNN for query and key appears to be of great importance. We hypothesize that this is due to the sparsely populated node features. Only a few nodes are attributed and, additionally, the permitted vocabulary is restrictive. The local message passing might spread the information to neighboring nodes to adjust for this sparseness. Moreover, w/o GNN we do not make use of edge features.

6 Conclusion

Directed graphs can lower the effective input dimensionality, while symmetrization can hurt performance. Our positional encodings based on the Magnetic Laplacian help to address both effects and can improve the understanding of directed graphs. We see this as an important step towards true multi-purpose transformers for graphs universally handling undirected and directed graphs.

References