Computing the S-Bandwidth of a Quantum Network's Graph Representation

Nathaniel Johnston, Sarah Plosker, and Luis M. B. Varona

MountAllison UNIVERSITY

Science Atlantic – MSCS Conference 2024
Wolfville, NS, Canada

October 5, 2024

Quantum Computers/Networks

While classical computers use **bits** (0 or 1), quantum computers use **qubits** (a "superposition" of both). Quantum computers can solve certain problems far faster but are very sensitive to external "noise."

- A **superposition** is a probability distribution of distinct states (made possible when **local realism** fails at the quantum scale)
- Recall the famous example of **Schrödinger's cat**:

Figure 1: My cat, **Ash**. (She may be both **awake** and **asleep**. . .)

Quantum Computers/Networks

While classical computers use **bits** (0 or 1), quantum computers use **qubits** (a "superposition" of both). Quantum computers can solve certain problems far faster but are very sensitive to external "noise."

- A **superposition** is a probability distribution of distinct states (made possible when **local realism** fails at the quantum scale)
- Recall the famous example of **Schrödinger's cat**:

Figure 1: My cat, **Ash**. (She may be both **awake** and **asleep**. . .)

Quantum Computers/Networks

While classical computers use **bits** (0 or 1), quantum computers use **qubits** (a "superposition" of both). Quantum computers can solve certain problems far faster but are very sensitive to external "noise."

- A **superposition** is a probability distribution of distinct states (made possible when **local realism** fails at the quantum scale)
- Recall the famous example of **Schrödinger's cat**:

Figure 1: My cat, **Ash**. (She may be both **awake** and **asleep**. . .)

Quantum Computers/Networks

When constructing quantum systems, we use undirected graphs to represent groups of linked processors, or **quantum networks**.

- **Vertices** represent separated quantum processors, while **edges** indicate the ability for qubits to move between these processors
- **Edge weights** (typically positive) represent voltages in cases where the processors are part of the same machine
- A system with **quantum channels** between all but two nodes:

Figure 2: The complete multipartite graph $K_{1,1,1,2}$ on 5 vertices.

Quantum Computers/Networks

When constructing quantum systems, we use undirected graphs to represent groups of linked processors, or **quantum networks**.

- **Vertices** represent separated quantum processors, while **edges** indicate the ability for qubits to move between these processors
- **Edge weights** (typically positive) represent voltages in cases where the processors are part of the same machine
- A system with **quantum channels** between all but two nodes:

Figure 2: The complete multipartite graph $K_{1,1,1,2}$ on 5 vertices.

Quantum Computers/Networks

When constructing quantum systems, we use undirected graphs to represent groups of linked processors, or **quantum networks**.

- **Vertices** represent separated quantum processors, while **edges** indicate the ability for qubits to move between these processors
- **Edge weights** (typically positive) represent voltages in cases where the processors are part of the same machine
- A system with **quantum channels** between all but two nodes:

Figure 2: The complete multipartite graph $K_{1,1,1,2}$ on 5 vertices.

Quantum Computers/Networks

When constructing quantum systems, we use undirected graphs to represent groups of linked processors, or **quantum networks**.

- **Vertices** represent separated quantum processors, while **edges** indicate the ability for qubits to move between these processors
- **Edge weights** (typically positive) represent voltages in cases where the processors are part of the same machine
- A system with **quantum channels** between all but two nodes:

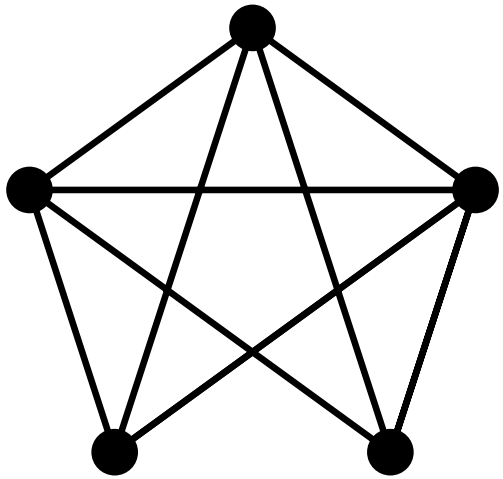


Figure 2: The complete multipartite graph $K_{1,1,1,2}$ on 5 vertices.

Quantum Computers/Networks

Definition (Perfect State Transfer)

*A quantum network with localities $u \perp v$ exhibits **perfect state transfer** over a period T of unitary evolution if and only if a qubit p's **wave function** satisfies*
$|\Psi(p \in u, t_0)| = |\Psi(p \in v, t_0 + T)| = 1.$

- i.e., when a particle holding quantum information completely "redistributes" from node u at time t_0 to node v at time $t_0 + T$
- Perfect state transfer (PST) thus facilitates **high-fidelity** information transmission in quantum communication systems
- A qubit's position is a probability distribution ("wave function") rather than a precise location, so PST is not easy to achieve. . .
- . . . but a certain property of network graph representations called *S*-**bandwidth** serves as an indicator in some cases

Quantum Computers/Networks

Definition (Perfect State Transfer)

A quantum network with localities $u \perp v$ exhibits ***perfect state transfer*** *over a period T of unitary evolution if and only if a qubit p's* ***wave function*** *satisfies*
$|\Psi(p \in u, t_0)| = |\Psi(p \in v, t_0 + T)| = 1.$

- **i.e.,** when a particle holding quantum information completely "redistributes" from node u at time t_0 to node v at time $t_0 + T$
- Perfect state transfer (PST) thus facilitates **high-fidelity** information transmission in quantum communication systems
- A qubit's position is a probability distribution ("wave function") rather than a precise location, so PST is not easy to achieve. . .
- . . . but a certain property of network graph representations called ***S*-bandwidth** serves as an indicator in some cases

Quantum Computers/Networks

Definition (Perfect State Transfer)

A quantum network with localities $u \perp v$ exhibits ***perfect state transfer*** *over a period T of unitary evolution if and only if a qubit p's* ***wave function*** *satisfies*
$|\Psi(p \in u, t_0)| = |\Psi(p \in v, t_0 + T)| = 1.$

- **i.e.,** when a particle holding quantum information completely "redistributes" from node u at time t_0 to node v at time $t_0 + T$
- Perfect state transfer (PST) thus facilitates **high-fidelity** information transmission in quantum communication systems
- A qubit's position is a probability distribution ("wave function") rather than a precise location, so PST is not easy to achieve. . .
- . . . but a certain property of network graph representations called *S*-**bandwidth** serves as an indicator in some cases

Quantum Computers/Networks

Definition (Perfect State Transfer)

*A quantum network with localities $u \perp v$ exhibits **perfect state transfer** over a period T of unitary evolution if and only if a qubit p's **wave function** satisfies*
$|\Psi(p \in u, t_0)| = |\Psi(p \in v, t_0 + T)| = 1.$

- **i.e.,** when a particle holding quantum information completely "redistributes" from node u at time t_0 to node v at time $t_0 + T$
- Perfect state transfer (PST) thus facilitates **high-fidelity** information transmission in quantum communication systems
- A qubit's position is a probability distribution ("wave function") rather than a precise location, so PST is not easy to achieve...
- ...but a certain property of network graph representations called *S*-**bandwidth** serves as an indicator in some cases

Quantum Computers/Networks

Definition (Perfect State Transfer)

A quantum network with localities $u \perp v$ exhibits ***perfect state transfer*** *over a period T of unitary evolution if and only if a qubit p's* ***wave function*** *satisfies*
$|\Psi(p \in u, t_0)| = |\Psi(p \in v, t_0 + T)| = 1.$

- **i.e.,** when a particle holding quantum information completely "redistributes" from node u at time t_0 to node v at time $t_0 + T$
- Perfect state transfer (PST) thus facilitates **high-fidelity** information transmission in quantum communication systems
- A qubit's position is a probability distribution ("wave function") rather than a precise location, so PST is not easy to achieve...
- ...but a certain property of network graph representations called ***S*-bandwidth** serves as an indicator in some cases

S-Bandwidth and PST

A small number of **Laplacian integral** graphs are $\{-1, 1\}$- and $\{-1, 0, 1\}$-**diagonalizable**. This can be an indicator of PST in quantum networks.

Definition (Matrix Bandwidth)

*A matrix X has a **matrix bandwidth** of $\beta(X) = k$ if and only if $x_{ij} = 0$ whenever $|i - j| \geq k$.*

- For instance, the following matrix has a bandwidth of 2:

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 & 0 & 0 \\ 0 & 4 & 6 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 & 3 & 0 \\ 0 & 0 & 0 & 3 & 3 & 7 \\ 0 & 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

S-Bandwidth and PST

A small number of **Laplacian integral** graphs are $\{-1, 1\}$- and $\{-1, 0, 1\}$-**diagonalizable**. This can be an indicator of PST in quantum networks.

Definition (Matrix Bandwidth)

*A matrix X has a **matrix bandwidth** of $\beta(X) = k$ if and only if $x_{ij} = 0$ whenever $|i - j| \geq k$.*

- For instance, the following matrix has a bandwidth of 2:

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 & 0 & 0 \\ 0 & 4 & 6 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 & 3 & 0 \\ 0 & 0 & 0 & 3 & 3 & 7 \\ 0 & 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

S-Bandwidth and PST

A small number of **Laplacian integral** graphs are $\{-1, 1\}$- and $\{-1, 0, 1\}$-**diagonalizable**. This can be an indicator of PST in quantum networks.

Definition (Matrix Bandwidth)

*A matrix X has a **matrix bandwidth** of $\beta(X) = k$ if and only if $x_{ij} = 0$ whenever $|i - j| \geq k$.*

- For instance, the following matrix has a bandwidth of 2:

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 & 0 & 0 \\ 0 & 4 & 6 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 & 3 & 0 \\ 0 & 0 & 0 & 3 & 3 & 7 \\ 0 & 0 & 0 & 0 & 5 & 0 \end{bmatrix}$$

S-Bandwidth and PST

Definition (Graph S-Bandwidth)

A graph G has an ***S-bandwidth*** *of $\beta_S(G) = k$ if and only if $L(G) = PDP^{-1}$ for some $P \in S^{n\times n} \subseteq \mathbb{R}^{n\times n}$ with $\beta(P^T P) = k$.*

- i.e., when G's Laplacian matrix has a full set of eigenvectors $v_1, v_2, \ldots, v_n \in S^n$ such that $v_i \cdot v_j = 0$ whenever $|i - j| \geq k$
- We say that a graph is *S*-diagonalizable if its S-bandwidth is finite—when $S = \{-1, 1\}$ or $\{-1, 0, 1\}$, this may indicate PST
- To test a graph for $\{-1, 1\}$- and $\{-1, 0, 1\}$-diagonalizability and determine its exact bandwidths, we first need to find the eigenvectors of its Laplacian matrix...

S-Bandwidth and PST

Definition (Graph S-Bandwidth)

*A graph G has an **S-bandwidth** of $\beta_S(G) = k$ if and only if $L(G) = PDP^{-1}$ for some $P \in S^{n\times n} \subseteq \mathbb{R}^{n\times n}$ with $\beta(P^T P) = k$.*

- **i.e.,** when G's **Laplacian matrix** has a full set of eigenvectors $v_1, v_2, \ldots, v_n \in S^n$ such that $v_i \cdot v_j = 0$ whenever $|i - j| \geq k$
- We say that a graph is **S-diagonalizable** if its S-bandwidth is finite—when $S = \{-1, 1\}$ or $\{-1, 0, 1\}$, this may indicate PST
- To test a graph for $\{-1, 1\}$- and $\{-1, 0, 1\}$-diagonalizability and determine its exact bandwidths, we first need to find the eigenvectors of its Laplacian matrix...

S-Bandwidth and PST

Definition (Graph S-Bandwidth)

*A graph G has an **S-bandwidth** of $\beta_S(G) = k$ if and only if $L(G) = PDP^{-1}$ for some $P \in S^{n\times n} \subseteq \mathbb{R}^{n\times n}$ with $\beta(P^T P) = k$.*

- **i.e.,** when G's **Laplacian matrix** has a full set of eigenvectors $v_1, v_2, \ldots, v_n \in S^n$ such that $v_i \cdot v_j = 0$ whenever $|i - j| \geq k$
- We say that a graph is **S-diagonalizable** if its S-bandwidth is finite—when $S = \{-1, 1\}$ or $\{-1, 0, 1\}$, this may indicate PST
- To test a graph for $\{-1, 1\}$- and $\{-1, 0, 1\}$-diagonalizability and determine its exact bandwidths, we first need to find the eigenvectors of its Laplacian matrix...

S-Bandwidth and PST

Definition (Graph S-Bandwidth)

*A graph G has an **S-bandwidth** of $\beta_S(G) = k$ if and only if $L(G) = PDP^{-1}$ for some $P \in S^{n\times n} \subseteq \mathbb{R}^{n\times n}$ with $\beta(P^T P) = k$.*

- **i.e.,** when G's **Laplacian matrix** has a full set of eigenvectors $v_1, v_2, \ldots, v_n \in S^n$ such that $v_i \cdot v_j = 0$ whenever $|i - j| \geq k$
- We say that a graph is **S-diagonalizable** if its S-bandwidth is finite—when $S = \{-1, 1\}$ or $\{-1, 0, 1\}$, this may indicate PST
- To test a graph for $\{-1, 1\}$- and $\{-1, 0, 1\}$-diagonalizability and determine its exact bandwidths, we first need to find the eigenvectors of its Laplacian matrix...

Identifying $\{-1, 0, 1\}$-Eigenvectors

- The **linear conditions** we have identified give the following bound on $\{-1, 0, 1\}$-eigenvectors for an order n Laplacian:

$$\frac{1}{2}\sum_{k=1}^{n}\binom{n}{k}\binom{n-k}{k} \leq 3^{n-2} \quad \forall n \in \mathbb{N}$$

- Therefore, eigenvector identification has **complexity** $O(3^{n-2})$
- This **exponential increase** in the number of eigenvectors to check is one of our two main bottlenecks. . . unfortunately, this seems to already be the theoretical lower bound

Identifying $\{-1, 0, 1\}$-Eigenvectors

- The **linear conditions** we have identified give the following bound on $\{-1, 0, 1\}$-eigenvectors for an order n Laplacian:

$$\frac{1}{2}\sum_{k=1}^{n}\binom{n}{k}\binom{n-k}{k} \leq 3^{n-2} \quad \forall n \in \mathbb{N}$$

- Therefore, eigenvector identification has **complexity** $O(3^{n-2})$
- This **exponential increase** in the number of eigenvectors to check is one of our two main bottlenecks. . . unfortunately, this seems to already be the theoretical lower bound

Identifying $\{-1, 0, 1\}$-Eigenvectors

- The **linear conditions** we have identified give the following bound on $\{-1, 0, 1\}$-eigenvectors for an order n Laplacian:

$$\frac{1}{2}\sum_{k=1}^{n}\binom{n}{k}\binom{n-k}{k} \leq 3^{n-2} \quad \forall n \in \mathbb{N}$$

- Therefore, eigenvector identification has **complexity** $O(3^{n-2})$
- This **exponential increase** in the number of eigenvectors to check is one of our two main bottlenecks. . . unfortunately, this seems to already be the theoretical lower bound

Finding k-Orthogonal Bases

Now, we have all $\{-1, 0, 1\}$-eigenvectors **stored as columns in matrices** (one for each eigenspace):

$$\begin{aligned}
&\textbf{Eigenspace 1:} && \left[v_{\lambda_1,1} \mid v_{\lambda_1,2} \mid \ldots \mid v_{\lambda_1,r_1} \right] \\
&\textbf{Eigenspace 2:} && \left[v_{\lambda_2,1} \mid v_{\lambda_2,2} \mid \ldots \mid v_{\lambda_2,r_2} \right] \\
& && \vdots \\
&\textbf{Eigenspace } \boldsymbol{k}\textbf{:} && \left[v_{\lambda_k,1} \mid v_{\lambda_k,2} \mid \ldots \mid v_{\lambda_k,r_k} \right]
\end{aligned}$$

- If any eigenspace contains less $\{-1, 0, 1\}$-eigenvectors than its multiplicity, the graph is not $\{-1, 0, 1\}$-diagonalizable
- Otherwise, convert each matrix to RREF (saving the pivots) to verify that we have enough **linearly independent** eigenvectors
- Similarly, test for $\{-1, 1\}$-diagonalizability

Finding k-Orthogonal Bases

Now, we have all $\{-1, 0, 1\}$-eigenvectors **stored as columns in matrices** (one for each eigenspace):

$$\begin{aligned}
&\textbf{Eigenspace 1:} && \left[v_{\lambda_1,1} \mid v_{\lambda_1,2} \mid \ldots \mid v_{\lambda_1,r_1} \right] \\
&\textbf{Eigenspace 2:} && \left[v_{\lambda_2,1} \mid v_{\lambda_2,2} \mid \ldots \mid v_{\lambda_2,r_2} \right] \\
& && \vdots \\
&\textbf{Eigenspace } \boldsymbol{k}\textbf{:} && \left[v_{\lambda_k,1} \mid v_{\lambda_k,2} \mid \ldots \mid v_{\lambda_k,r_k} \right]
\end{aligned}$$

- If any eigenspace contains less $\{-1, 0, 1\}$-eigenvectors than its multiplicity, the graph is not $\{-1, 0, 1\}$-diagonalizable
- Otherwise, convert each matrix to RREF (saving the pivots) to verify that we have enough **linearly independent** eigenvectors
- Similarly, test for $\{-1, 1\}$-diagonalizability

Finding k-Orthogonal Bases

Now, we have all $\{-1, 0, 1\}$-eigenvectors **stored as columns in matrices** (one for each eigenspace):

$$\begin{aligned}
&\textbf{Eigenspace 1:} && \left[v_{\lambda_1,1} \mid v_{\lambda_1,2} \mid \ldots \mid v_{\lambda_1,r_1} \right] \\
&\textbf{Eigenspace 2:} && \left[v_{\lambda_2,1} \mid v_{\lambda_2,2} \mid \ldots \mid v_{\lambda_2,r_2} \right] \\
& && \vdots \\
&\textbf{Eigenspace } \boldsymbol{k}\textbf{:} && \left[v_{\lambda_k,1} \mid v_{\lambda_k,2} \mid \ldots \mid v_{\lambda_k,r_k} \right]
\end{aligned}$$

- If any eigenspace contains less $\{-1, 0, 1\}$-eigenvectors than its multiplicity, the graph is not $\{-1, 0, 1\}$-diagonalizable
- Otherwise, convert each matrix to RREF (saving the pivots) to verify that we have enough **linearly independent** eigenvectors
- Similarly, test for $\{-1, 1\}$-diagonalizability

Finding k-Orthogonal Bases

Now, we have all $\{-1, 0, 1\}$-eigenvectors **stored as columns in matrices** (one for each eigenspace):

$$\begin{aligned}
&\textbf{Eigenspace 1:} && \left[\mathrm{v}_{\lambda_1,1} \mid \mathrm{v}_{\lambda_1,2} \mid \ldots \mid \mathrm{v}_{\lambda_1,r_1}\right] \\
&\textbf{Eigenspace 2:} && \left[\mathrm{v}_{\lambda_2,1} \mid \mathrm{v}_{\lambda_2,2} \mid \ldots \mid \mathrm{v}_{\lambda_2,r_2}\right] \\
& && \vdots \\
&\textbf{Eigenspace } \boldsymbol{k}\textbf{:} && \left[\mathrm{v}_{\lambda_k,1} \mid \mathrm{v}_{\lambda_k,2} \mid \ldots \mid \mathrm{v}_{\lambda_k,r_k}\right]
\end{aligned}$$

- If any eigenspace contains less $\{-1, 0, 1\}$-eigenvectors than its multiplicity, the graph is not $\{-1, 0, 1\}$-diagonalizable
- Otherwise, convert each matrix to RREF (saving the pivots) to verify that we have enough **linearly independent** eigenvectors
- Similarly, test for $\{-1, 1\}$-diagonalizability

Finding k-Orthogonal Bases

We must find not only linearly independent eigenbases, but those with minimum **k-orthogonality** (i.e., with a Gram matrix $P^T P$ of bandwidth k). Since matrix bandwidth is **permutation-variant**:

- For $k = 1$ (pairwise orthogonality): Check whether the Gram matrix of the basis set is **diagonal**
- For $k = 2$ (quasi-orthogonality): Check whether the Gram matrix of the basis is the adjacency matrix of a **path subgraph**
- For $2 < k < n$: Here we must resort to some more complex **subgraph isomorphism** tests, which can still be improved...

Finding k-Orthogonal Bases

We must find not only linearly independent eigenbases, but those with minimum **k-orthogonality** (i.e., with a Gram matrix $P^T P$ of bandwidth k). Since matrix bandwidth is **permutation-variant**:

- For $k = 1$ (pairwise orthogonality): Check whether the Gram matrix of the basis set is **diagonal**
- For $k = 2$ (quasi-orthogonality): Check whether the Gram matrix of the basis is the adjacency matrix of a **path subgraph**
- For $2 < k < n$: Here we must resort to some more complex **subgraph isomorphism** tests, which can still be improved...

Finding k-Orthogonal Bases

We must find not only linearly independent eigenbases, but those with minimum **k-orthogonality** (i.e., with a Gram matrix $P^T P$ of bandwidth k). Since matrix bandwidth is **permutation-variant**:

- For $k = 1$ (pairwise orthogonality): Check whether the Gram matrix of the basis set is **diagonal**
- For $k = 2$ (quasi-orthogonality): Check whether the Gram matrix of the basis is the adjacency matrix of a **path subgraph**
- For $2 < k < n$: Here we must resort to some more complex **subgraph isomorphism** tests, which can still be improved...

Finding k-Orthogonal Bases

We must find not only linearly independent eigenbases, but those with minimum **k-orthogonality** (i.e., with a Gram matrix $P^T P$ of bandwidth k). Since matrix bandwidth is **permutation-variant**:

- For $k = 1$ (pairwise orthogonality): Check whether the Gram matrix of the basis set is **diagonal**
- For $k = 2$ (quasi-orthogonality): Check whether the Gram matrix of the basis is the adjacency matrix of a **path subgraph**
- For $2 < k < n$: Here we must resort to some more complex **subgraph isomorphism** tests, which can still be improved...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Finding k-Orthogonal Bases

Now to actually find a basis...

- We construct a search tree of **eigenvector indices**, starting from roots with one index each
- Create children by adding column indices corresponding to eigenvectors, only keeping options that preserve linear independence and maintain k-orthogonality
- Conduct a **depth-first search** on this **lazily constructed tree** until a k-orthogonal basis is found
- If no basis is found, repeat for $(k+1)$-orthogonality ...
- Perform a search for $\{-1, 1\}$-bandwidth as well if appropriate
- Unfortunately, this search tree grows in **factorial time**, making it another bottleneck...

Minimizing S-Bandwidth

We can now use these foundations to construct the first algorithm to compute a graph's S-bandwidth!

- First, we check each eigenspace for a **k-orthogonal basis** up to $k = \mu - 1$ (where μ is the dimension of the eigenspace)
- If no $(\mu - 1)$-orthogonal basis is found, we use the pivots from our original row reduction to get a μ-orthogonal basis
- With every test, save the value of k and only start checking subsequent eigenspaces against that **orthogonality parameter**
- Complete this process for all eigenspaces to minimize the overall S-bandwidth of the graph, and we are done!

Minimizing S-Bandwidth

We can now use these foundations to construct the first algorithm to compute a graph's S-bandwidth!

- First, we check each eigenspace for a **k-orthogonal basis** up to $k = \mu - 1$ (where μ is the dimension of the eigenspace)
- If no $(\mu - 1)$-orthogonal basis is found, we use the pivots from our original row reduction to get a μ-orthogonal basis
- With every test, save the value of k and only start checking subsequent eigenspaces against that orthogonality parameter
- Complete this process for all eigenspaces to minimize the overall S-bandwidth of the graph, and we are done!

Minimizing S-Bandwidth

We can now use these foundations to construct the first algorithm to compute a graph's S-bandwidth!

- First, we check each eigenspace for a **k-orthogonal basis** up to $k = \mu - 1$ (where μ is the dimension of the eigenspace)
- If no $(\mu - 1)$-orthogonal basis is found, we use the pivots from our original row reduction to get a μ-orthogonal basis
- With every test, save the value of k and only start checking subsequent eigenspaces against that **orthogonality parameter**
- Complete this process for all eigenspaces to minimize the overall S-bandwidth of the graph, and we are done!

Minimizing S-Bandwidth

We can now use these foundations to construct the first algorithm to compute a graph's S-bandwidth!

- First, we check each eigenspace for a **k-orthogonal basis** up to $k = \mu - 1$ (where μ is the dimension of the eigenspace)
- If no $(\mu - 1)$-orthogonal basis is found, we use the pivots from our original row reduction to get a μ-orthogonal basis
- With every test, save the value of k and only start checking subsequent eigenspaces against that **orthogonality parameter**
- Complete this process for all eigenspaces to minimize the overall S-bandwidth of the graph, and we are done!

Results / Findings / Future Work

We have tabulated, with exact $\{-1, 1\}$- and $\{-1, 0, 1\}$-bandwidths,

- all $\{-1, 0, 1\}$-diagonalizable simple connected graphs on $n \leq 11$ vertices, and
- all $\{-1, 0, 1\}$-diagonalizable simple connected regular graphs (a superset of the bipartite ones) on $n \leq 14$ vertices

We have also proven several results about the Laplacian spectra of S-diagonalizable graphs, the effects of different edge weights on S-bandwidths, and S-diagonalizable graph composition.

Next, we hope to improve methods of matrix bandwidth reduction, reduce the complexity of our DFS search tree, investigate remaining conjectures on S-diagonalizable graphs.

Results / Findings / Future Work

We have tabulated, with exact $\{-1, 1\}$- and $\{-1, 0, 1\}$-bandwidths,

- all $\{-1, 0, 1\}$-diagonalizable **simple connected graphs** on $n \leq 11$ vertices, and
- all $\{-1, 0, 1\}$-diagonalizable simple connected regular graphs (a superset of the bipartite ones) on $n \leq 14$ vertices

We have also proven several results about the Laplacian spectra of S-diagonalizable graphs, the effects of different edge weights on S-bandwidths, and S-diagonalizable graph composition.

Next, we hope to improve methods of matrix bandwidth reduction, reduce the complexity of our DFS search tree, investigate remaining conjectures on S-diagonalizable graphs.

Results / Findings / Future Work

We have tabulated, with exact $\{-1, 1\}$- and $\{-1, 0, 1\}$-bandwidths,

- all $\{-1, 0, 1\}$-diagonalizable **simple connected graphs** on $n \leq 11$ vertices, and
- all $\{-1, 0, 1\}$-diagonalizable **simple connected regular graphs** (a superset of the **bipartite** ones) on $n \leq 14$ vertices

We have also proven several results about the **Laplacian spectra** of S-diagonalizable graphs, the effects of different **edge weights** on S-bandwidths, and S-diagonalizable **graph composition**.

Next, we hope to improve methods of **matrix bandwidth reduction**, reduce the complexity of our **DFS search tree**, investigate remaining conjectures on S-diagonalizable graphs.

Results / Findings / Future Work

We have tabulated, with exact $\{-1, 1\}$- and $\{-1, 0, 1\}$-bandwidths,

- all $\{-1, 0, 1\}$-diagonalizable **simple connected graphs** on $n \leq 11$ vertices, and
- all $\{-1, 0, 1\}$-diagonalizable **simple connected regular graphs** (a superset of the **bipartite** ones) on $n \leq 14$ vertices

We have also proven several results about the **Laplacian spectra** of S-diagonalizable graphs, the effects of different **edge weights** on S-bandwidths, and S-diagonalizable **graph composition**.

Next, we hope to improve methods of **matrix bandwidth reduction**, reduce the complexity of our **DFS search tree**, investigate remaining conjectures on S-diagonalizable graphs.

Results / Findings / Future Work

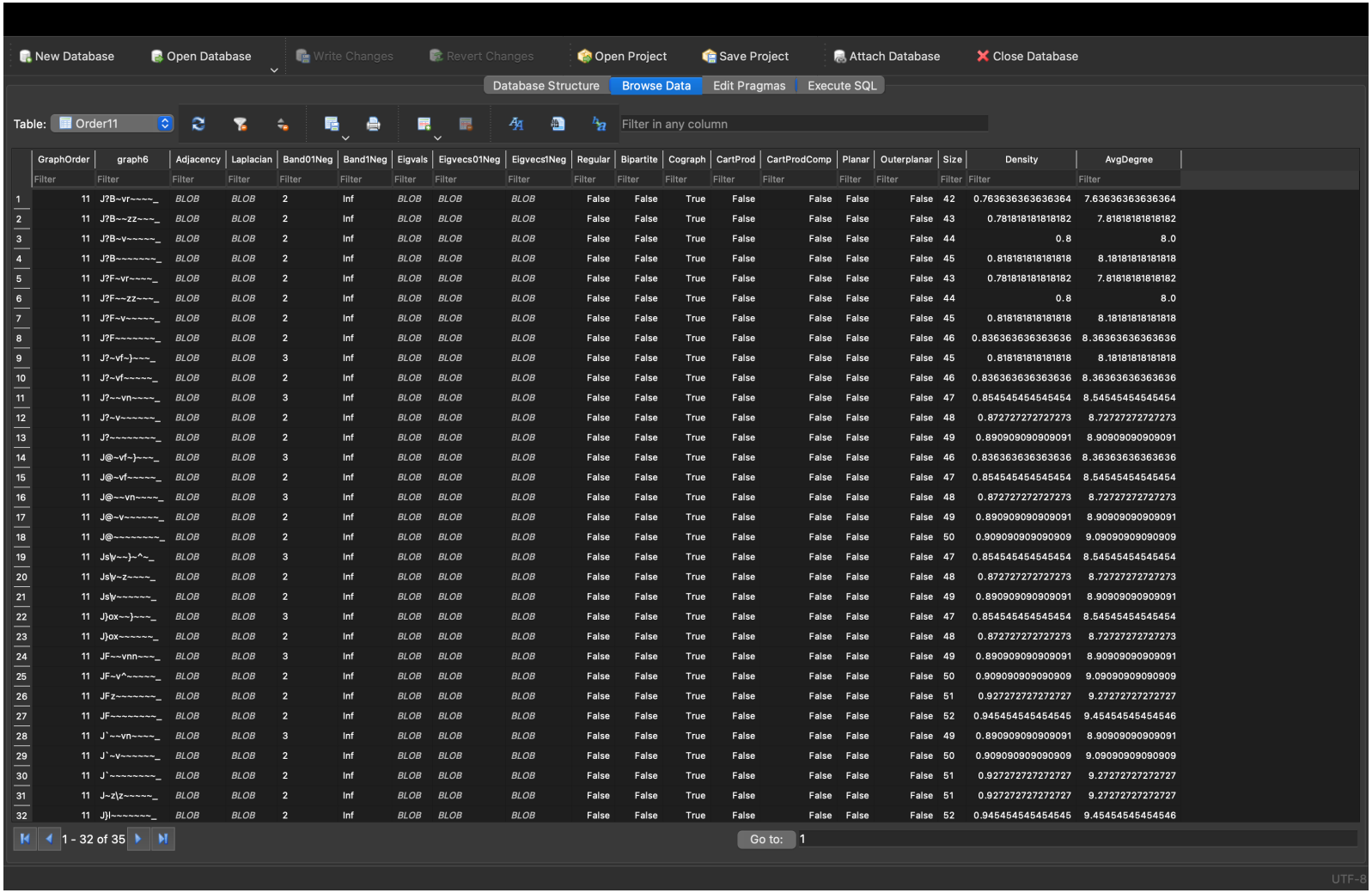
We have tabulated, with exact $\{-1, 1\}$- and $\{-1, 0, 1\}$-bandwidths,

- all $\{-1, 0, 1\}$-diagonalizable **simple connected graphs** on $n \leq 11$ vertices, and
- all $\{-1, 0, 1\}$-diagonalizable **simple connected regular graphs** (a superset of the **bipartite** ones) on $n \leq 14$ vertices

We have also proven several results about the **Laplacian spectra** of S-diagonalizable graphs, the effects of different **edge weights** on S-bandwidths, and S-diagonalizable **graph composition**.

Next, we hope to improve methods of **matrix bandwidth reduction**, reduce the complexity of our **DFS search tree**, investigate remaining conjectures on S-diagonalizable graphs.

Results / Findings / Future Work



	GraphOrder	graph6	Adjacency	Laplacian	Band01Neg	Band1Neg	Eigvals	Eigvecs01Neg	Eigvecs1Neg	Regular	Bipartite	Cograph	CartProd	CartProdComp	Planar	Outerplanar	Size	Density	AvgDegree
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	11	J?B~vf~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	42	0.763636363636364	7.63636363636364
2	11	J?B~~zz~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	43	0.781818181818182	7.81818181818182
3	11	J?B~v~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	44	0.8	8.0
4	11	J?B~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	45	0.818181818181818	8.18181818181818
5	11	J?F~vf~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	43	0.781818181818182	7.81818181818182
6	11	J?F~~zz~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	44	0.8	8.0
7	11	J?F~v~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	45	0.818181818181818	8.18181818181818
8	11	J?F~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	46	0.836363636363636	8.36363636363636
9	11	J?~vf~}~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	45	0.818181818181818	8.18181818181818
10	11	J?~vf~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	46	0.836363636363636	8.36363636363636
11	11	J?~~vn~~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	47	0.854545454545454	8.54545454545454
12	11	J?~v~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	48	0.872727272727273	8.72727272727273
13	11	J?~~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	49	0.890909090909091	8.90909090909091
14	11	J@~vf~}~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	46	0.836363636363636	8.36363636363636
15	11	J@~vf~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	47	0.854545454545454	8.54545454545454
16	11	J@~~vn~~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	48	0.872727272727273	8.72727272727273
17	11	J@~v~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	49	0.890909090909091	8.90909090909091
18	11	J@~~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	50	0.909090909090909	9.09090909090909
19	11	Js\v~~}~^~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	47	0.854545454545454	8.54545454545454
20	11	Js\v~z~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	48	0.872727272727273	8.72727272727273
21	11	Js\v~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	49	0.890909090909091	8.90909090909091
22	11	J}ox~~}~~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	47	0.854545454545454	8.54545454545454
23	11	J}ox~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	48	0.872727272727273	8.72727272727273
24	11	JF~~vnn~~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	49	0.890909090909091	8.90909090909091
25	11	JF~v^~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	50	0.909090909090909	9.09090909090909
26	11	JFz~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	51	0.927272727272727	9.27272727272727
27	11	JF~~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	52	0.945454545454545	9.45454545454546
28	11	J`~~vn~~~_	BLOB	BLOB	3	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	49	0.890909090909091	8.90909090909091
29	11	J`~v~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	50	0.909090909090909	9.09090909090909
30	11	J`~~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	51	0.927272727272727	9.27272727272727
31	11	J~z\z~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	51	0.927272727272727	9.27272727272727
32	11	J}l~~~~~~_	BLOB	BLOB	2	Inf	BLOB	BLOB	BLOB	False	False	True	False	False	False	False	52	0.945454545454545	9.45454545454546

Figure 3: Some tabulated data on $\{-1, 0, 1\}$-diagonalizable graphs.

Thank you!

Figure 4: Pusheen the Cat <3