Spanning Subgraph Isomorphism

Drew Meier, Dr. Austin Mohr, Thomas Schuler

Department of Mathematics
Nebraska Wesleyan University

August 4, 2016

NEBRASKA
WESLEYAN
UNIVERSITY

Outline

1. What is Subgraph Isomorphism
2. Cut Vertex Refinement
3. Perfect Matching Refinements
4. Skeleton
5. Data

Outline

1. What is Subgraph Isomorphism
2. Cut Vertex Refinement
3. Perfect Matching Refinements
4. Skeleton
5. Data

Outline

1. What is Subgraph Isomorphism
2. Cut Vertex Refinement
3. Perfect Matching Refinements
4. Skeleton
5. Data

Outline

1. What is Subgraph Isomorphism
2. Cut Vertex Refinement
3. Perfect Matching Refinements
4. Skeleton
5. Data

Outline

1. What is Subgraph Isomorphism
2. Cut Vertex Refinement
3. Perfect Matching Refinements
4. Skeleton
5. Data

Subgraph Isomorphism

- Given a pattern graph and a host graph, *Subgraph Isomorphism* is the problem of determining whether the host graph contains a subgraph that is isomorphic to the pattern graph

Subgraph Isomorphism

- Given a pattern graph and a host graph, *Subgraph Isomorphism* is the problem of determining whether the host graph contains a subgraph that is isomorphic to the pattern graph
- We restrict ourselves to patterns that span the host

Subgraph Isomorphism

- Given a pattern graph and a host graph, *Subgraph Isomorphism* is the problem of determining whether the host graph contains a subgraph that is isomorphic to the pattern graph
- We restrict ourselves to patterns that span the host
- Ullman (1976)

Candidate Matrix

We use a matrix, C, to hold information about candidate vertices, whose rows correspond to vertices of the pattern, P, and columns correspond the vertices of the host, H.

Candidate Matrix

We use a matrix, C, to hold information about candidate vertices, whose rows correspond to vertices of the pattern, P, and columns correspond the vertices of the host, H.

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

Candidate Matrix

We use a matrix, C, to hold information about candidate vertices, whose rows correspond to vertices of the pattern, P, and columns correspond the vertices of the host, H.

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

$C_{i,j} = 1$ indicates vertex i of P is a candidate for vertex j of H.

Candidate Matrix

We use a matrix, C, to hold information about candidate vertices, whose rows correspond to vertices of the pattern, P, and columns correspond the vertices of the host, H.

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

$C_{i,j} = 1$ indicates vertex i of P is a candidate for vertex j of H. Reductions eliminate portions of the search space by eliminating candidates

Trivial Reductions

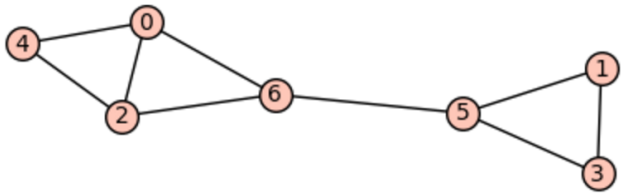
- Reduction by Degree

Trivial Reductions

- Reduction by Degree
- Reduction by Adjacency

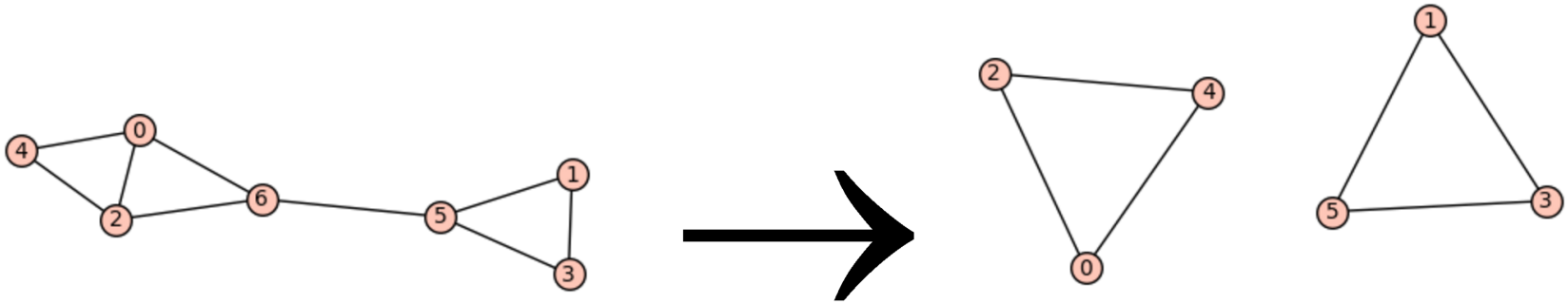
Cut Vertex Refinement

A cut vertex, u, is a cut vertex on a graph, G, if the graph $G - u$ has more components than G.



Cut Vertex Refinement

A cut vertex, u, is a cut vertex on a graph, G, if the graph $G - u$ has more components than G.



Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Cut Vertex Refinement

Theorem

A cut vertex, u, of the host, H, must be mapped to a cut vertex, v, of the pattern P

Proof.

Suppose u is a cut vertex of the host graph, H. Now suppose we assign a non-cut vertex v of the pattern, P, to u. Consider only the connected components of H and P containing u and v, H_u and P_v. Since u is a cut vertex, $H_u - u$ has at least 2 components. Since v is not a cut vertex, $P_v - v$ is connected. Clearly, no mapping of the connected pattern, $P_v - v$, into the disconnected host $H_u - u$ can exist. Thus, a cut vertex of the host graph must map to a cut vertex of the pattern graph. □

Component Partition Refinement

Definition

Let $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_k)$ be two partitions of a common integer. We say the partition a **refines** the partition b provided there is a partition of the indexing set $\{1, ..., n\}$ into blocks $J_1, J_2, ..., J_\ell$ such that $b_i = \sum_{j \in J_i} a_j$ for all $1 \leq i \leq k$

Component Partition Refinement

Definition

Let $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_k)$ be two partitions of a common integer. We say the partition a **refines** the partition b provided there is a partition of the indexing set $\{1, ..., n\}$ into blocks $J_1, J_2, ..., J_\ell$ such that $b_i = \sum_{j \in J_i} a_j$ for all $1 \leq i \leq k$

Example:

Component Partition Refinement

Definition

Let $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_k)$ be two partitions of a common integer. We say the partition a **refines** the partition b provided there is a partition of the indexing set $\{1, ..., n\}$ into blocks $J_1, J_2, ..., J_\ell$ such that $b_i = \sum_{j \in J_i} a_j$ for all $1 \leq i \leq k$

Example:

$$a = [1, 1, 3, 4] \quad b = [2, 7]$$

Component Partition Refinement

Definition

Let $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_k)$ be two partitions of a common integer. We say the partition a **refines** the partition b provided there is a partition of the indexing set $\{1, ..., n\}$ into blocks $J_1, J_2, ..., J_\ell$ such that $b_i = \sum_{j \in J_i} a_j$ for all $1 \leq i \leq k$

Example:

$$a = [1, 1, 3, 4] \quad b = [2, 7]$$

Let $J_1 = \{1, 2\}$ and $J_2 = \{3, 4\}$. Then,

$$a_1 + a_2 = 2 = b_1 \text{ and } a_3 + a_4 = 7 = b_2$$

Component Partition Refinement

Definition

Let $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_k)$ be two partitions of a common integer. We say the partition a **refines** the partition b provided there is a partition of the indexing set $\{1, ..., n\}$ into blocks $J_1, J_2, ..., J_\ell$ such that $b_i = \sum_{j \in J_i} a_j$ for all $1 \leq i \leq k$

Example:

$$a = [1, 1, 3, 4] \quad b = [2, 7]$$

Let $J_1 = \{1, 2\}$ and $J_2 = \{3, 4\}$. Then,

$$a_1 + a_2 = 2 = b_1 \text{ and } a_3 + a_4 = 7 = b_2$$

Thus, a refines b.

Component Partition Refinement

Definition

Given a graph G of order n and a vertex u, let $G_1, ..., G_k$ denote the connected components of $G - u$ labeled so that $|G_1| \geq |G_2| \geq \cdots \geq |G_k|$. The **component partition** $C(v)$ is the partition of $n - 1$ given by $(|G_1|, |G_2|, ..., |G_k|)$.

Component Partition Refinement

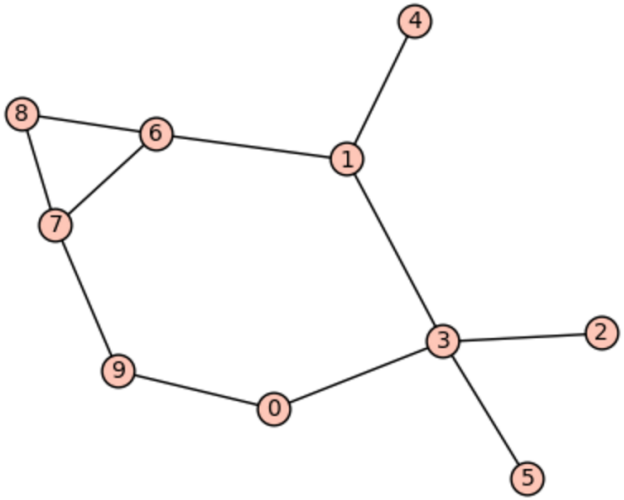
Definition

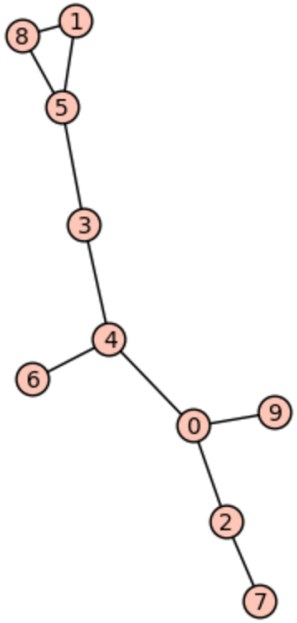
Given a graph G of order n and a vertex u, let $G_1, ..., G_k$ denote the connected components of $G - u$ labeled so that $|G_1| \geq |G_2| \geq \cdots \geq |G_k|$. The **component partition** $C(v)$ is the partition of $n-1$ given by $(|G_1|, |G_2|, ..., |G_k|)$.

Theorem

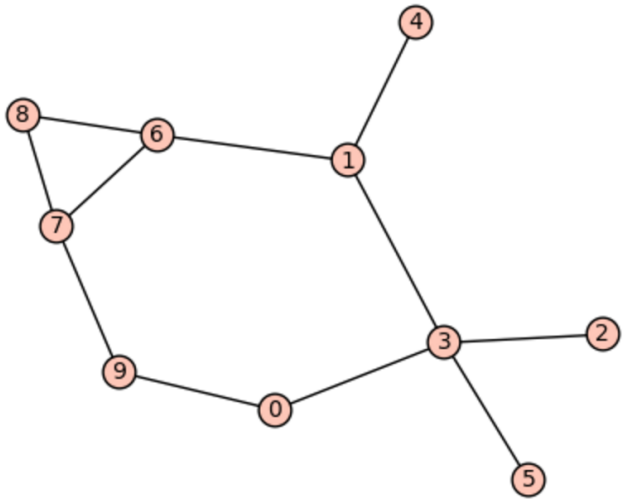
Let P and H be graphs of equal order and $f : V(P) \to V(H)$ be a subgraph isomorphism. For any vertex $u \in V(H)$, the component partition $C(f^{-1}(v))$ refines the component partition $C(u)$.

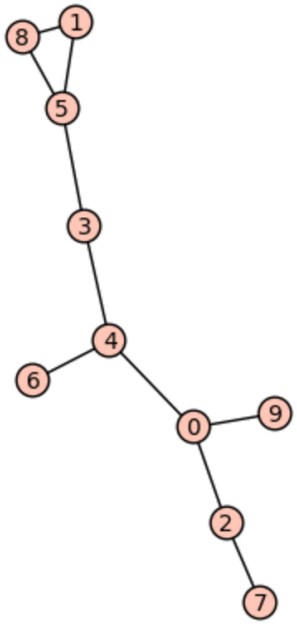
Component Partition Refinement (Example)





Component Partition Refinement (Example)

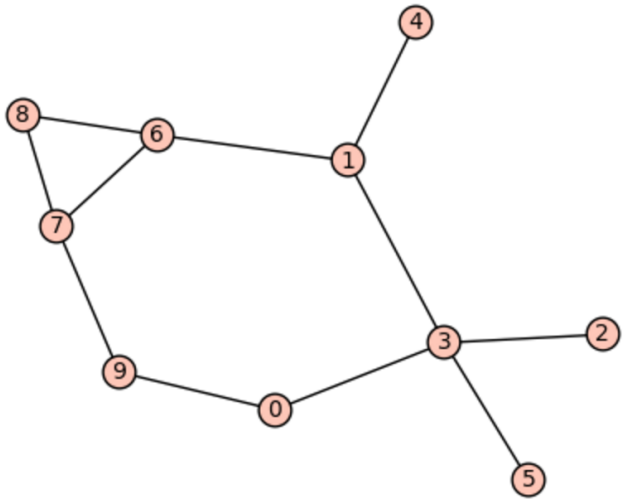




$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

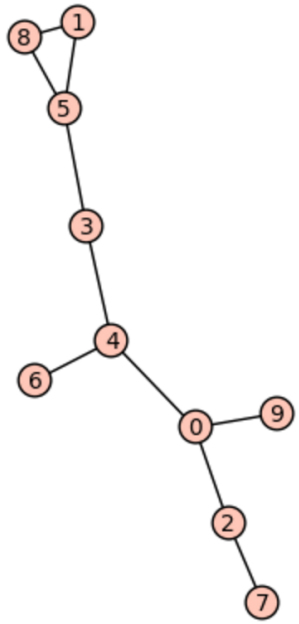
Degree Reduction

Component Partition Refinement (Example)



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Degree Reduction



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Cut Vertex Reduction

Matching Reductions

Definition

A *matching* on a graph, G, is a subset of edges in G such that no edges share common vertices.

Matching Reductions

Definition

A *matching* on a graph, G, is a subset of edges in G such that no edges share common vertices.

Definition

A *perfect matching* on a graph, G, is a matching on G such that all vertices of G are visited by an edge in the set.

Matching Reductions

Upon assignment of vertex v of the pattern to vertex u of the host, we set up a bipartite graph as follows:

Matching Reductions

Upon assignment of vertex v of the pattern to vertex u of the host, we set up a bipartite graph as follows:

- One partition is the set of neighbors of u

Matching Reductions

Upon assignment of vertex v of the pattern to vertex u of the host, we set up a bipartite graph as follows:

- One partition is the set of neighbors of u
- The other neighbors of v

Matching Reductions

Upon assignment of vertex v of the pattern to vertex u of the host, we set up a bipartite graph as follows:

- One partition is the set of neighbors of u
- The other neighbors of v
- Vertices i, j in the bipartition have an edge if vertex i is a candidate for vertex j.

Matching Reductions

Upon assignment of vertex v of the pattern to vertex u of the host, we set up a bipartite graph as follows:

- One partition is the set of neighbors of u
- The other neighbors of v
- Vertices i, j in the bipartition have an edge if vertex i is a candidate for vertex j.

We call this the neighborhood bipartition, $K_{u,v}$.

Matching Reductions

Upon assignment of vertex v of the pattern to vertex u of the host, we set up a bipartite graph as follows:

- One partition is the set of neighbors of u
- The other neighbors of v
- Vertices i, j in the bipartition have an edge if vertex i is a candidate for vertex j.

We call this the neighborhood bipartition, $K_{u,v}$.

Theorem

For vertex u to be a candidate for vertex v upon assignment of v to u, a matching on $K_{u,v}$ the size of the partition v must exist.

Perfect Matching Requirement

We set up a bipartite graph with the same procedure as with the matching reduction; however, the partitions are the entire set of vertices for H and P. We call this graph, $K_{H,P}$

Perfect Matching Requirement

We set up a bipartite graph with the same procedure as with the matching reduction; however, the partitions are the entire set of vertices for H and P. We call this graph, $K_{H,P}$

Theorem

Given $K_{H,P}$, for there to exist a subgraph isomorphism from P to H, a perfect matching must exist on $K_{H,P}$.

Perfect Matching Requirement

We set up a bipartite graph with the same procedure as with the matching reduction; however, the partitions are the entire set of vertices for H and P. We call this graph, $K_{H,P}$

Theorem

Given $K_{H,P}$, for there to exist a subgraph isomorphism from P to H, a perfect matching must exist on $K_{H,P}$.

Proof.

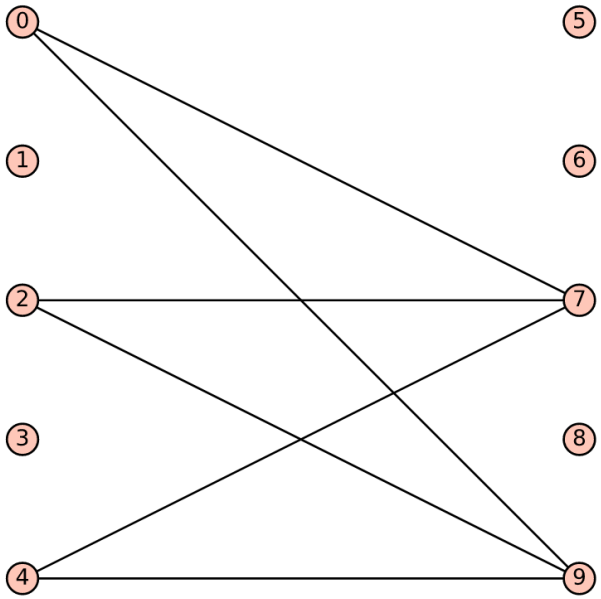
By way of contradiction suppose there is no perfect matching on $K_{H,P}$. Since the partitions of $K_{H,P}$ are of equal order, this implies at least one vertex of each partition has no edge connecting them. This means at least one vertex from P cannot be mapped to any vertex from H. Since at least one vertex from the pattern is left out regardless of which edge configuration we choose for the matching, at least one vertex cannot be mapped. □

Perfect Matching Requirement (Example)

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Perfect Matching Requirement (Example)

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$



Skeleton

- Reduce the candidate matrix as much as possible

Skeleton

- Reduce the candidate matrix as much as possible
- Once all reductions are made, assign a vertex

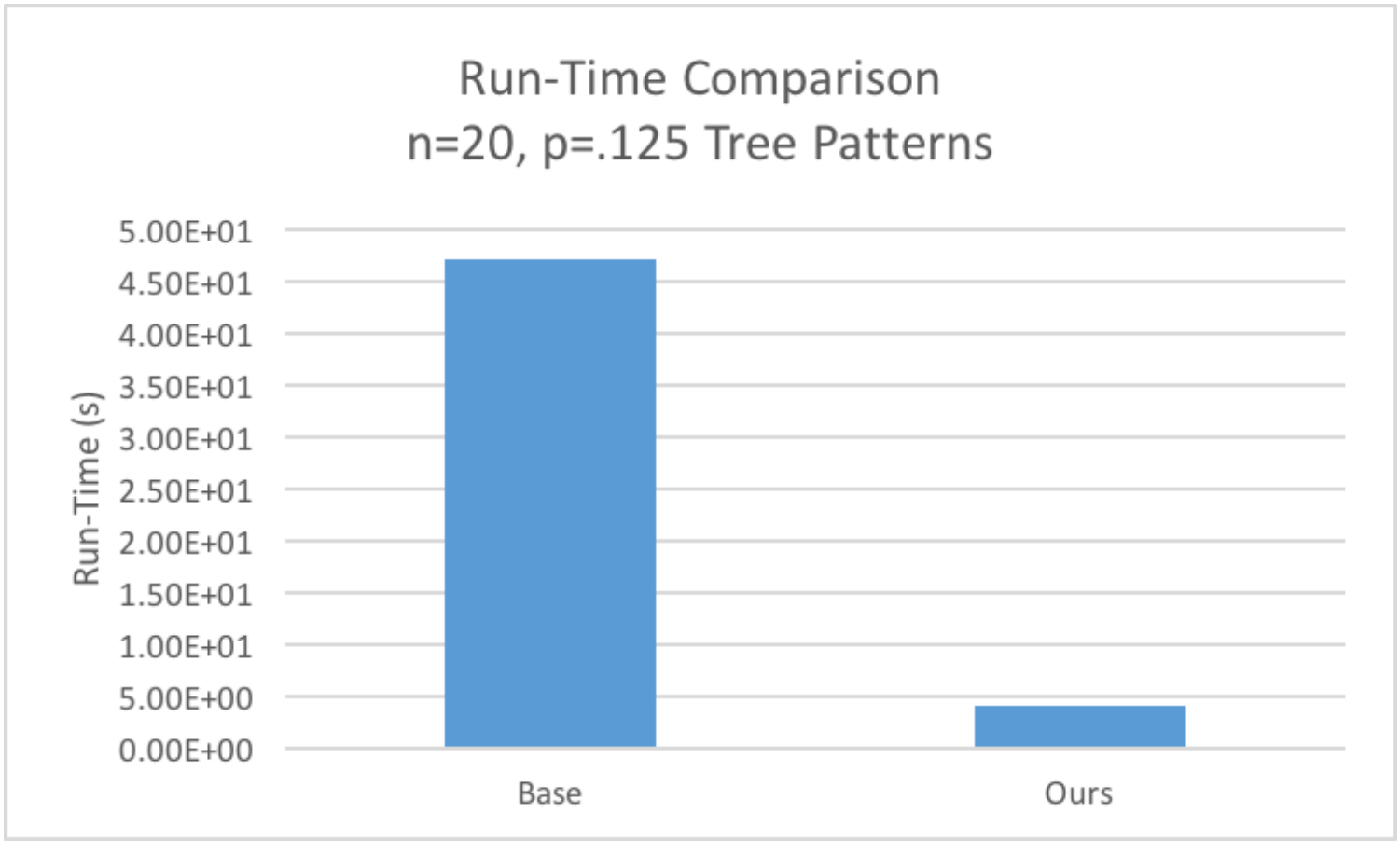
Skeleton

- Reduce the candidate matrix as much as possible
- Once all reductions are made, assign a vertex
- Reduce the new candidate matrix as much as possible, check for fail conditions

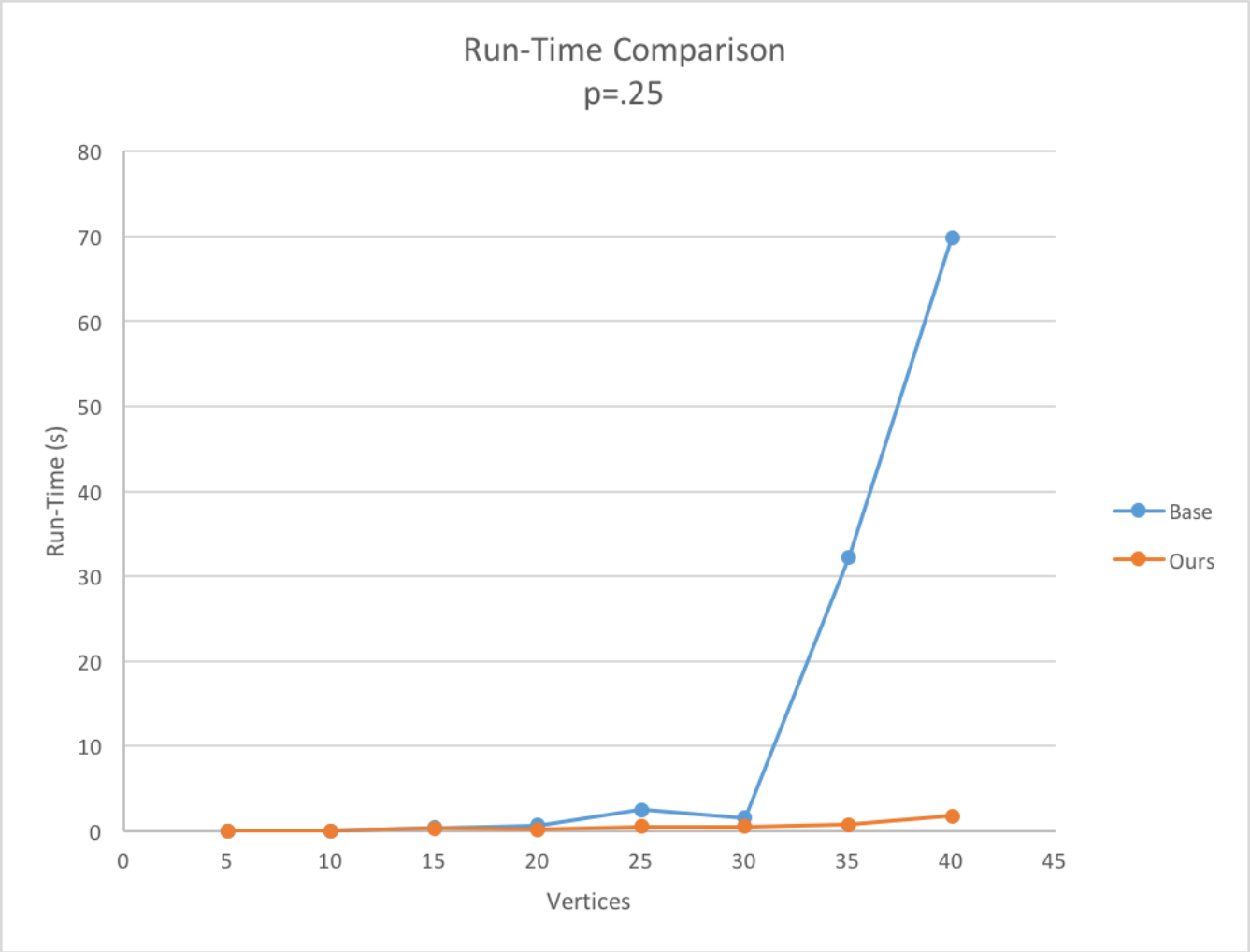
Skeleton

- Reduce the candidate matrix as much as possible
- Once all reductions are made, assign a vertex
- Reduce the new candidate matrix as much as possible, check for fail conditions
- Repeat until search tree is exhausted

Data


Run-Time Comparison
n=20, p=.125 Tree Patterns
Run-Time (s)
5.00E+01
4.50E+01
4.00E+01
3.50E+01
3.00E+01
2.50E+01
2.00E+01
1.50E+01
1.00E+01
5.00E+00
0.00E+00
Base
Ours

Data


Run-Time Comparison
p=.25
Run-Time (s)
80
70
60
50
40
30
20
10
0
0
5
10
15
20
25
30
35
40
45
Vertices
Base
Ours

References

- Douglas West, *Introduction to Graph Theory*, Prentice-Hall, 2nd edition, 2001
- Julian R. Ullman, *An algorithm for subgraph isomorphism*, J. of the ACM **23** (1976), 31-42.

Thanks

Thanks for listening!