Vertex Ordering and Partitioning techniques in graphs

Reet Barik

School of Electrical Engineering and Computer Science
Washington State University

April 8, 2020

Summary

1. Motivation
2. Existing Works
   - MINLA: J. Petit. Journal of Experimental Algorithmics, 2003
   - MLOGA: Chierichetti et al. KDD, 2009
   - Gorder: Wei et al. International Conference on Management of Data, 2016
   - RCM: Cuthill et. al. ACM 1969
   - DegSort
   - Rabbit Order: Arai et. al. IEEE International Parallel and Distributed Processing Symposium, 2016
   - CHDFS: Banerjee et. al. IEEE Trans. Software Eng., 1988
   - Slashburn: Kang et. al. ICDM, 2011
   - LDG: Stanton et. al. KDD, 2012
   - METIS: Karypis et. al. J. Parallel Distrib. Comput. 1998
3. Proposed Idea
4. The End

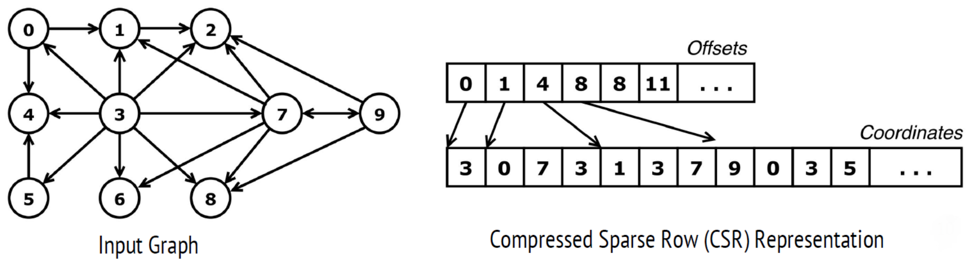
Motivation

Reordering Improves Spatial and Temporal Locality

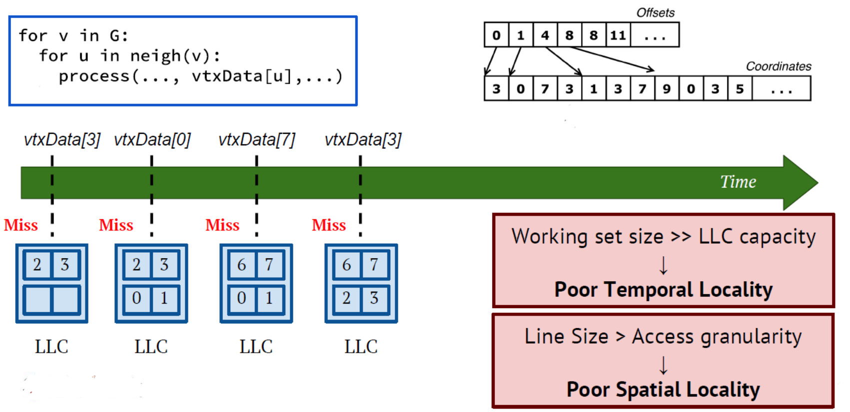
```
for v in G:
  for u in neigh(v):
    process(..., vtxData[u],...)
```

Typical graph processing kernel

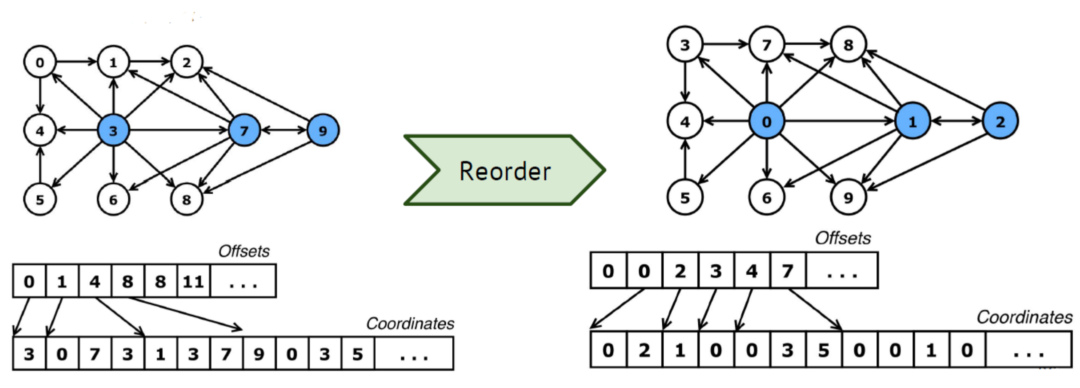
Reordering Improves Spatial and Temporal Locality



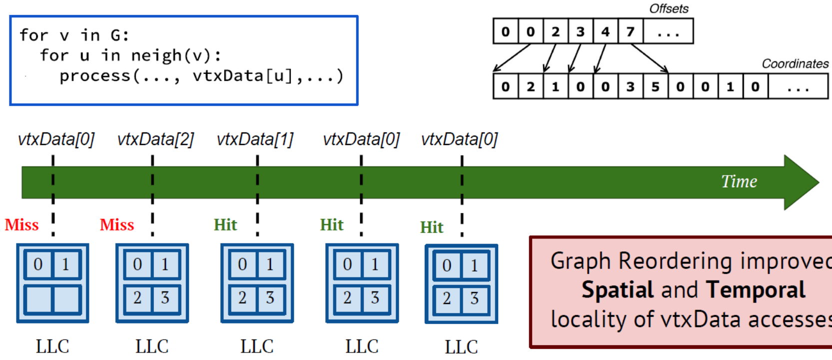
Reordering Improves Spatial and Temporal Locality



Reordering Improves Spatial and Temporal Locality



Reordering Improves Spatial and Temporal Locality

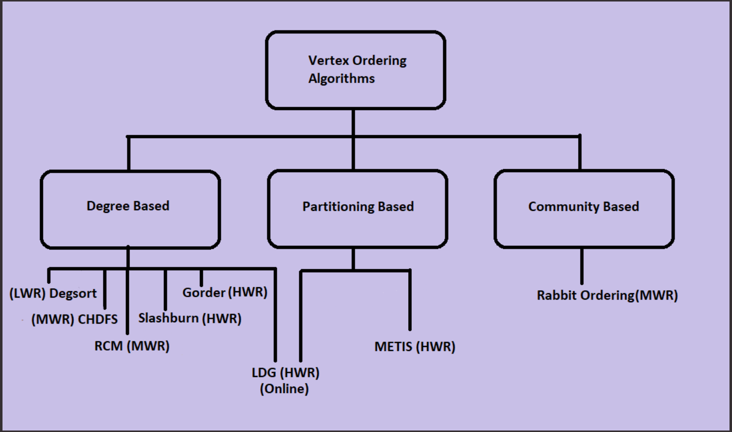


Graph Reordering improved **Spatial** and **Temporal** locality of vtxData accesses

Existing Works

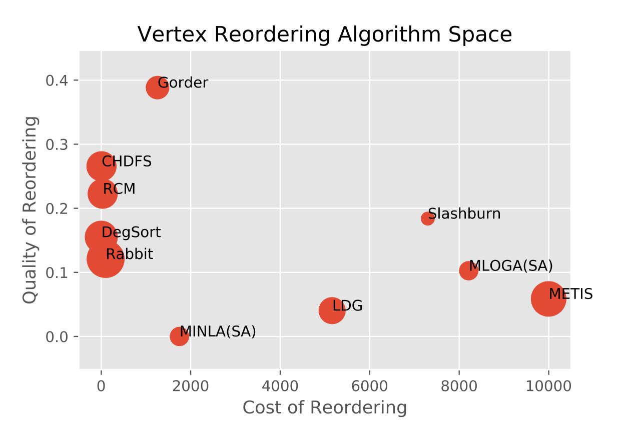
Algorithm Space

Classification of Vertex Reordering Algorithms



Algorithm Space

Cost vs. Quality vs. Parallelizability (Size of point) of Algorithm



Subsection 1

MINLA: J. Petit. Journal of Experimental Algorithmics, 2003

Minimum Linear Arrangement Problem

Problem

A *layout* or a *linear arrangement* of an undirected graph $G = (V, E)$ with $|V| = n$ is a one-to-one function $\phi : V \to 1...n$
The Minimum Linear Arrangement problem is a combinatorial optimization problem formulated as follows:
Given a graph $G = (V, E)$, find a layout ϕ that minimizes:

$$LA(G, \phi) = \sum_{uv \in E(G)} |\phi(u) - \phi(v)|$$

Solution

Simulated Annealing to optimize.

Subsection 2

MLOGA: Chierichetti et al. KDD, 2009

Minimum Logarithmic Gap Arrangement Problem

Problem

A *layout* or a *linear arrangement* of an undirected graph $G = (V, E)$ with $|V| = n$ is a one-to-one function $\phi : V \rightarrow 1...n$

The Minimum Logarithmic Gap Arrangement problem is a combinatorial optimization problem formulated as follows:

Given a graph $G = (V, E)$, find a layout ϕ that minimizes:

$$LA(G, \phi) = \sum_{uv \in E(G)} log_2(|\phi(u) - \phi(v)|)$$

Solution

Simulated Annealing to optimize and generate the ordering.

Subsection 3

Gorder: Wei et al. International Conference on Management of Data, 2016

Algorithm

```
for v in G:
  for u in neigh(v):
    process(..., vtxData[u],...)
```

Typical graph processing kernel

It can be observed that two types of relationships between nodes need to be taken into account: neighbors (if there exists and edge in between) and siblings (if there is a common in-neighbor).

Algorithm

The metric defined is aimed to capture the locality between two vertices. For two nodes u and v, the scoring function is given by:

$$S(u, v) = S_s(u, v) + S_n(u, v)$$

where,

- $S_s(u, v)$ is the number of the times that u and v co-exist in sibling relationships, which is the number of their common in-neighbors.
- $S_n(u, v)$ is the number of times that u and v are neighbors, which is either 0, 1, or 2.

Algorithm

- The solution offered takes the 'sliding window' approach.
- If there are two nodes u and v with ordering $\phi(u)$ and $\phi(v)$ respectively such that u comes before v in the ordering. For a fixed v and window size w, the algorithm takes a look at all the combination of u and v, for all nodes u that come before v in the sliding window of size w.
- The problem statement is as follows:
  Find the optimal graph ordering $\phi(\cdot)$, that maximizes $Gscore$ (the sum of locality score), F(·), based on a sliding window model with a window size w, where,

$$F(\phi) = \sum_{0<\phi(v)-\phi(u)\leq w} S(u, v)$$

- The above is then solved by reducing this to a parameterized (window size 'w') variant of the maximum traveling salesman problem.

Subsection 4

RCM: Cuthill et. al. ACM 1969

Reverse Cuthill-McKee

Objective

Reduce the bandwidth of the adjacency matrix for a given graph

Algorithm

1. Select a starting node which might be a node with minimum degree and relabel as 1
2. Neighboring nodes are relabeled in sequence beginning from 2 in order of increasing degree
3. This procedure is repeated starting from the node labeled 2, then 3 and so on.
4. This will terminate when all nodes of a component are labeled. Do this for all disconnected components (if any).

For matrices which can be transformed to band diagonal form with no zeroes in the band, this scheme will be optimal.

Reverse Cuthill-McKee

Objective

Reduce the bandwidth of the adjacency matrix for a given graph

Algorithm

1. Select a starting node which might be a node with minimum degree and relabel as 1
2. Neighboring nodes are relabeled in sequence beginning from 2 in order of increasing degree
3. This procedure is repeated starting from the node labeled 2, then 3 and so on.
4. This will terminate when all nodes of a component are labeled. Do this for all disconnected components (if any).

For matrices which can be transformed to band diagonal form with no zeroes in the band, this scheme will be optimal.

Subsection 5

DegSort

HubSort or DegSort

Algorithm

Sort the vertices in decreasing order of their degree (as shown in the figure).

Subsection 6

Rabbit Order: Arai et. al. IEEE International Parallel and Distributed Processing Symposium, 2016

Overview

Intuition

This algorithm aims to achieve high locality by mapping the following:

- hierarchical community structures in real world graphs
- hierarchical structure of CPU caches.

Overview

Intuition

This algorithm aims to achieve high locality by mapping the following:

- hierarchical community structures in real world graphs
- hierarchical structure of CPU caches.

Overview

Intuition

This algorithm aims to achieve high locality by mapping the following:

- hierarchical community structures in real world graphs
- hierarchical structure of CPU caches.

Algorithm

Algorithm Overview of Rabbit Order

Input: Graph $G = (V, E)$
Output: Permutation $\pi : V \to N$ for new vertex ordering

▷ Perform hierarchical community-based ordering
1 *dendrogram* $\leftarrow$ CommunityDetection()
2 **return** OrderingGeneration(*dendrogram*)

3 **function** CommunityDetection()
  ▷ Perform incremental aggregation
4   **for each** $u \in V$ in increasing order of degree **do**
5     $v \leftarrow$ neighbor of u that maximizes $\Delta Q(u, v)$
6   **if** $\Delta Q(u, v) > 0$ **then**
7   Merge u into v and record this merge in *dendrogram*
8   **return** *dendrogram*

9 **function** OrderingGeneration(*dendrogram*)
10   $new\_id \leftarrow 0$
11   **for each** $v \in V$ in DFS visiting order on *dendrogram* **do**
12     $\pi[v] \leftarrow new\_id$; $new\_id \leftarrow new\_id + 1$
13   **return** π

The modularity gain in Step 6 is defined as follows:

$$\triangle Q(u, v) = 2\left(\frac{w_{uv}}{2m} - \frac{d(u)d(v)}{(2m)^2}\right)$$

Subsection 7

CHDFS: Banerjee et. al. IEEE Trans. Software Eng., 1988

Children Depth First Search

Algorithm

This is a mixture of the traditional Breadth First Search and Depth First Search traversal methods. The pseudocode is as follows:

```
PROCEDURE Children-Depth-First Traversal (P):
  IF node P was not previously visited THEN
  DO
    Visit node P;
        Visit ALL previously unvisited children of P;
      FOR EACH child C of P
        CALL Children-Depth-First (C);
    END;
    END PROCEDURE.
```

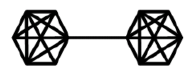
Subsection 8

Slashburn: Kang et. al. ICDM, 2011

Slashburn Algorithm

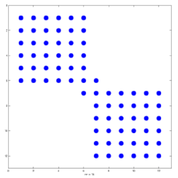
Intuition

- Search for 'Caveman Communities' as shown in the figure



Caveman graph C

- Find an ordering of nodes to get block-diagonal Adj matrix



Adjacency Matrix of C

Slashburn Algorithm

Problem Statement

Given a graph with the adjacency matrix A, find a permutation $\pi : V \longrightarrow [n]$ such that the storage cost function $cost(A)$ is minimized.

Two cost functions can be considered:

- $cost(A, b) =$ number of non-empty blocks
- $cost(A, b) = |T|.2log\frac{n}{b} + \sum_{\tau \in T} b^2.H(\frac{z(\tau)}{b^2})$

Slashburn Algorithm

Problem Statement

Given a graph with the adjacency matrix A, find a permutation $\pi : V \longrightarrow [n]$ such that the storage cost function $cost(A)$ is minimized.
Two cost functions can be considered:

- $cost(A, b) =$ number of non-empty blocks
- $cost(A, b) = |T|.2log\frac{n}{b} + \sum_{\tau \in T} b^2.H(\frac{z(\tau)}{b^2})$

Slashburn Algorithm

Problem Statement

Given a graph with the adjacency matrix A, find a permutation $\pi : V \longrightarrow [n]$ such that the storage cost function $cost(A)$ is minimized.
Two cost functions can be considered:

- $cost(A, b) =$ number of non-empty blocks
- $cost(A, b) = |T|.2log\frac{n}{b} + \sum_{\tau \in T} b^2.H(\frac{z(\tau)}{b^2})$

Slashburn Algorithm

Problem Statement

Given a graph with the adjacency matrix A, find a permutation $\pi : V \longrightarrow [n]$ such that the storage cost function $cost(A)$ is minimized.
Two cost functions can be considered:

- $cost(A, b) =$ number of non-empty blocks
- $cost(A, b) = |T|.2log\frac{n}{b} + \sum_{\tau \in T} b^2.H(\frac{z(\tau)}{b^2})$

Slashburn Algorithm

Algorithm

Algorithm : SlashBurn

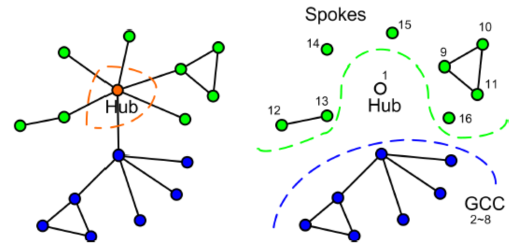
Input: Edge set E of a graph $G = (V, E)$,
a constant k(default = 1).

Output: Array Γ containing the ordering $V \rightarrow [n]$.

1: Remove k-hubset from G to make the new graph G'. Add the removed k-hubset to the front of Γ.

2: Find connected components in G'. Add nodes in non-giant connected components to the back of Γ, in the decreasing order of sizes of connected components they belong to.

3: Set G to be the giant connected component(GCC) of G'. Go to step 1 and continue, until the number of nodes in the GCC is smaller than k.



(a) Before SlashBurn (b) After SlashBurn

Subsection 9

LDG: Stanton et. al. KDD, 2012

Linear Deterministic Greedy

Problem Setup

- A simple streaming graph model is considered here.
- A cluster of k machines with memory capacity C each (such that kC is large enough to hold the whole graph).
- The vertices arrive in a stream with the set of edges where it is a member and as they do, a partitioner decides to place the vertex on one of the k machines.
- A vertex is never moved after it has been placed.

Linear Deterministic Greedy

Problem Setup

- A simple streaming graph model is considered here.
- A cluster of k machines with memory capacity C each (such that kC is large enough to hold the whole graph).
- The vertices arrive in a stream with the set of edges where it is a member and as they do, a partitioner decides to place the vertex on one of the k machines.
- A vertex is never moved after it has been placed.

Linear Deterministic Greedy

Problem Setup

- A simple streaming graph model is considered here.
- A cluster of k machines with memory capacity C each (such that kC is large enough to hold the whole graph).
- The vertices arrive in a stream with the set of edges where it is a member and as they do, a partitioner decides to place the vertex on one of the k machines.
- A vertex is never moved after it has been placed.

Linear Deterministic Greedy

Problem Setup

- A simple streaming graph model is considered here.
- A cluster of k machines with memory capacity C each (such that kC is large enough to hold the whole graph).
- The vertices arrive in a stream with the set of edges where it is a member and as they do, a partitioner decides to place the vertex on one of the k machines.
- A vertex is never moved after it has been placed.

Linear Deterministic Greedy

Stream Order and Heuristic

Stream order:

- Random: Vertices arrive in an order given by the random permutationo of the vertices.
- BFS: Select a starting node from each connected component and traverse using BFS. Do that for all connected components (component ordering is random).
- DFS: Replace BFS by DFS in the previous.

Linear Deterministic Greedy

Stream Order and Heuristic

Heuristic:

1. Assign v to the partition where it has the most edges.
2. Weighted by a penalty function based on partition capacity (larger partitions are penalized more).
3. Ties are broken by assigning v the partition of minimal size. Further ties are broken randomly.

The ordering is calculated as follows:

$$ind = argmax_{i \in [k]}(|P^t(i) \cap \tau(v)| w(t, i))$$

where, $\tau(v)$ is the set of neighboring vertices of v and $w(t, i) = 1 - \frac{|P^t(i)|}{C}$

Linear Deterministic Greedy

Stream Order and Heuristic

Heuristic:

1. Assign v to the partition where it has the most edges.
2. Weighted by a penalty function based on partition capacity (larger partitions are penalized more).
3. Ties are broken by assigning v the partition of minimal size. Further ties are broken randomly.

The ordering is calculated as follows:

$$ind = argmax_{i \in [k]}(|P^t(i) \cap \tau(v)| w(t,i))$$

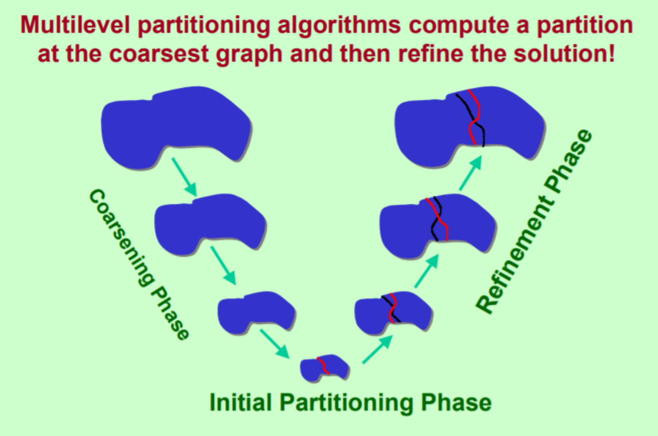
where, $\tau(v)$ is the set of neighboring vertices of v and $w(t,i) = 1 - \frac{|P^t(i)|}{C}$

Subsection 10

METIS: Karypis et. al. J. Parallel Distrib. Comput. 1998

METIS: Multilevel k-way partitioning

Intuition



METIS: Multilevel k-way partitioning

Step 1: Coarsening

Done by using Maximal Matching in one of the following 4 ways:

- Random Matching (RM)
- Heavy Edge Matching (HEM)
- Light Edge Matching (LEM)
- Heavy Clique Matching (HCM)

Note: A 'matching' of a graph is a set of edges no two of which are incident on the same vertex. A 'maximal matching' is a matching such that, if any edge in the graph is not in the matching, then it has at least one of its endpoints matched.

METIS: Multilevel k-way partitioning

Step 1: Coarsening

Done by using Maximal Matching in one of the following 4 ways:

- Random Matching (RM)
- Heavy Edge Matching (HEM)
- Light Edge Matching (LEM)
- Heavy Clique Matching (HCM)

Note: A 'matching' of a graph is a set of edges no two of which are incident on the same vertex. A 'maximal matching' is a matching such that, if any edge in the graph is not in the matching, then it has at least one of its endpoints matched.

METIS: Multilevel k-way partitioning

Step 2: Partitioning

Done by using any of the following algorithms:

- Spectral bisection (SB)
- KL Algorithm
- Graph growing partitioning algorithm (GGP)
- Greedy graph growing partitioning algorithm (GGGP)

METIS: Multilevel k-way partitioning

Step 3: Uncoarsening

KL algorithm results in good partitions in the partitioning phase. Hence the following two algorithms are used for the uncoarsening phase (refines in the least number of iterations).

- KL refinement
- Boundary KL refinement

Proposed Idea

Proposed Idea

Approach I: Baseline

Proposed Idea

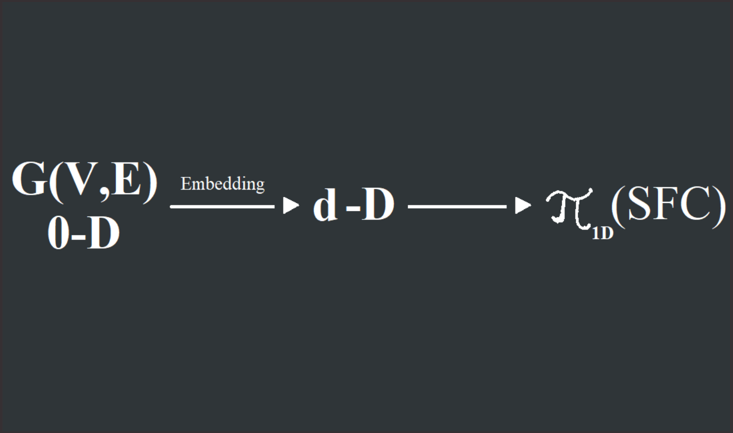
Approach I: Baseline

Proposed Idea

Approach I: Baseline

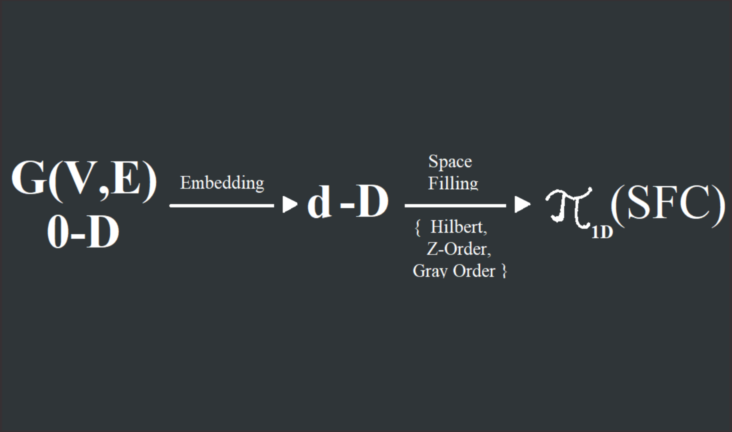
Proposed Idea

Approach I: Baseline



Proposed Idea

Approach I: Baseline



Proposed Idea

Approach II

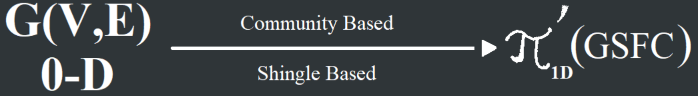
Proposed Idea

Approach II

$$\underset{\text{0-D}}{G(V,E)} \longrightarrow \pi'_{1D}(GSFC)$$

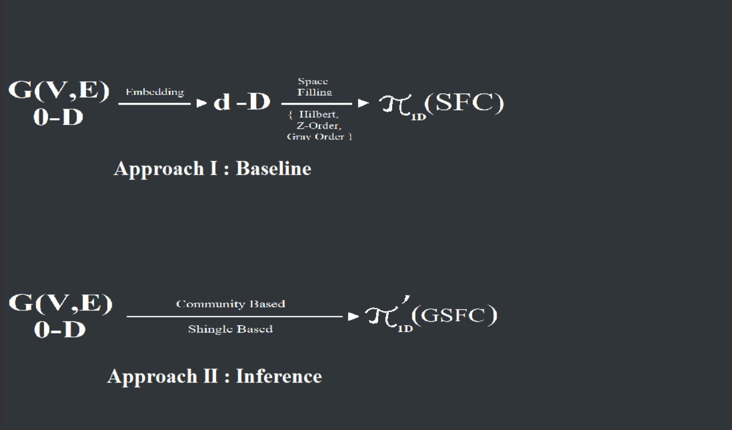
Proposed Idea

Approach II



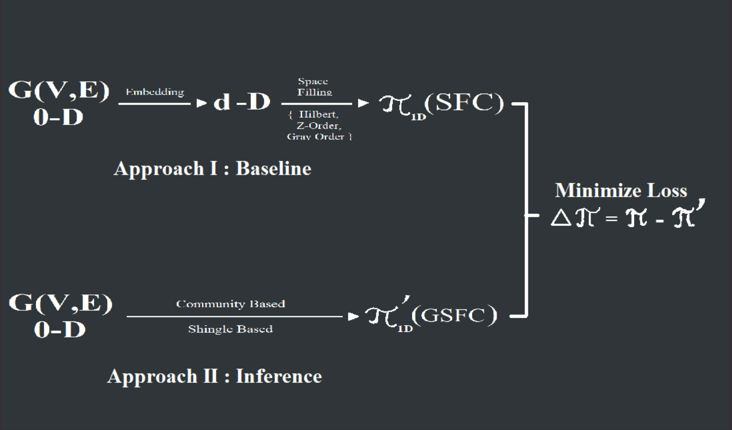
Proposed Idea

Overview



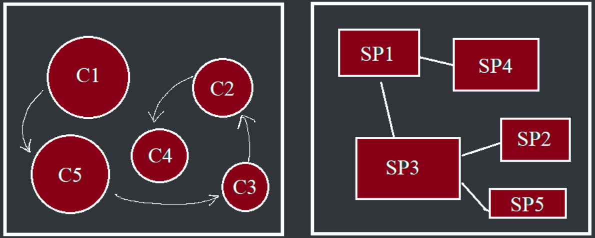
Proposed Idea

Overview



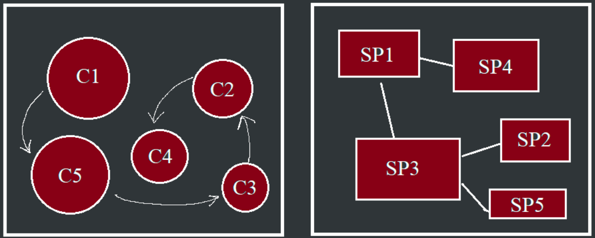
Proposed Idea

GSFC: Graph Space Filling Curve



Proposed Idea

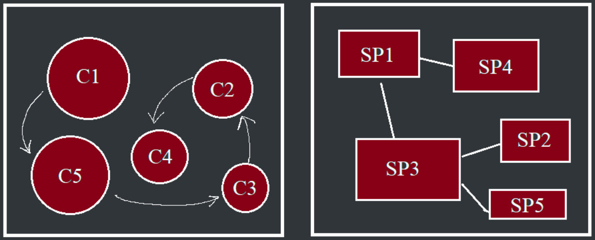
GSFC: Graph Space Filling Curve



Community based

Proposed Idea

GSFC: Graph Space Filling Curve

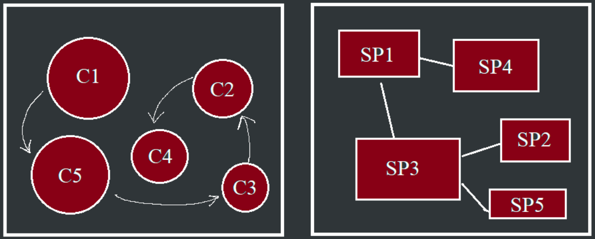


Community based

Spanning Tree based

Proposed Idea

GSFC: Graph Space Filling Curve



Community based

Spanning Tree based

Ordering: Greedy, B-way Matching etc. etc.

Thank You