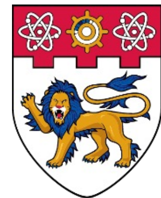
VLDB2023 vancouver

Towards Designing and Learning Piecewise Space-Filling Curves

Jiangneng Li[1], Zheng Wang[1], Gao Cong[1], Cheng Long[1], Han Mao Kiah[1], and Bin Cui[2]

[1]Nanyang Technological University [2]Peking University

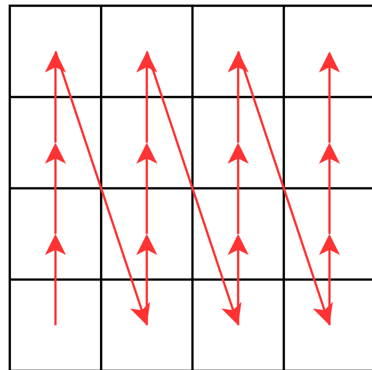


Introduction

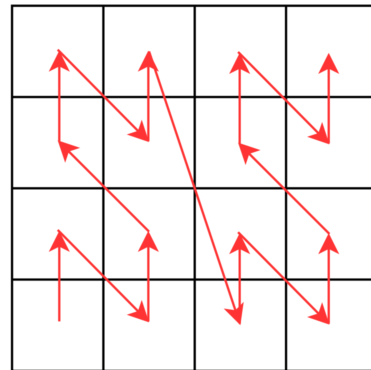
The motivation and our idea

Space-Filling Curve (SFC)

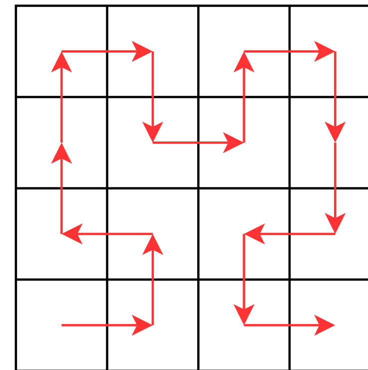
- A SFC is used to map a multi-dimensional data point to a value
- Then a one-dimensional index can be used to index the mapped values
  - B+tree index, supported by many DBMS, such as PostgreSQL, DynamoDB, HBase
  - Learned indexes



(a) C-curve



(b) Z-curve

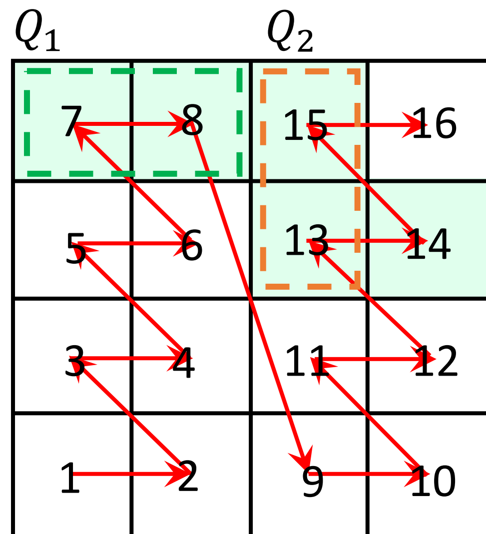


(c) Hilbert curve

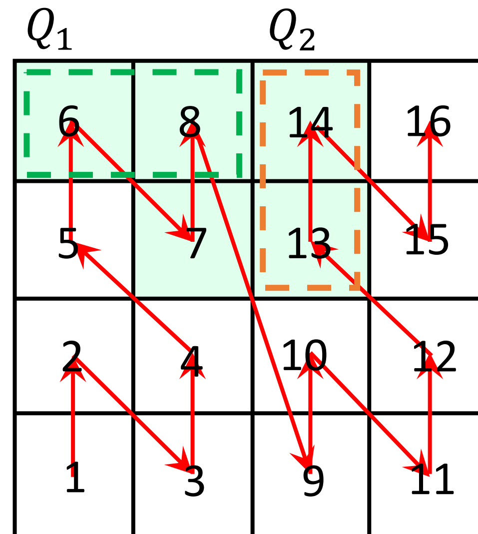
- Each type of SFC has its own fixed mapping function
- Cannot be adjusted to fit with different datasets.

Design instance-optimized SFCs

- No single SFC can dominate the performance on all datasets and query workloads



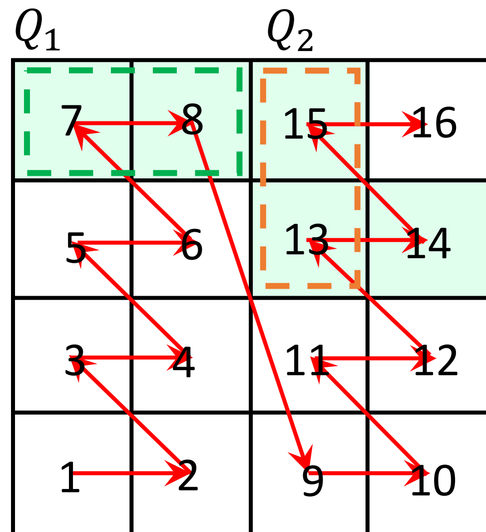
(a) SFC-1 works best for Q_1.



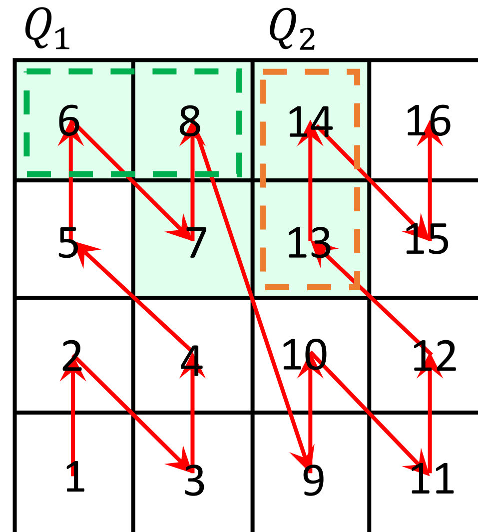
(b) SFC-2 works best for Q_2.

Our Idea

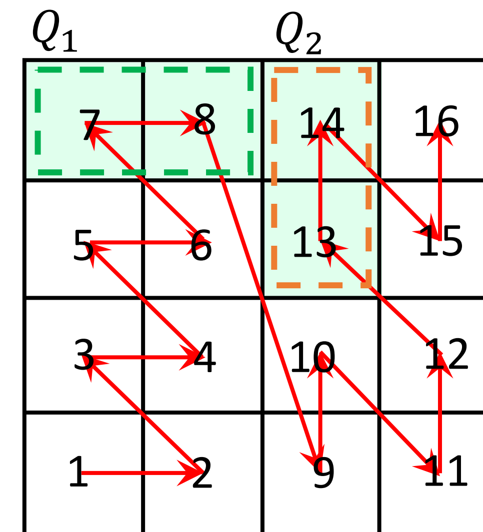
- Design a SFC that combining the advantage of multiple SFCs and thus reach to an optimized performance



(a) SFC-1 works best for Q_1.



(b) SFC-2 works best for Q_2.



(c) SFC-3 combines SFC-1 and SFC-2, works best for both queries.

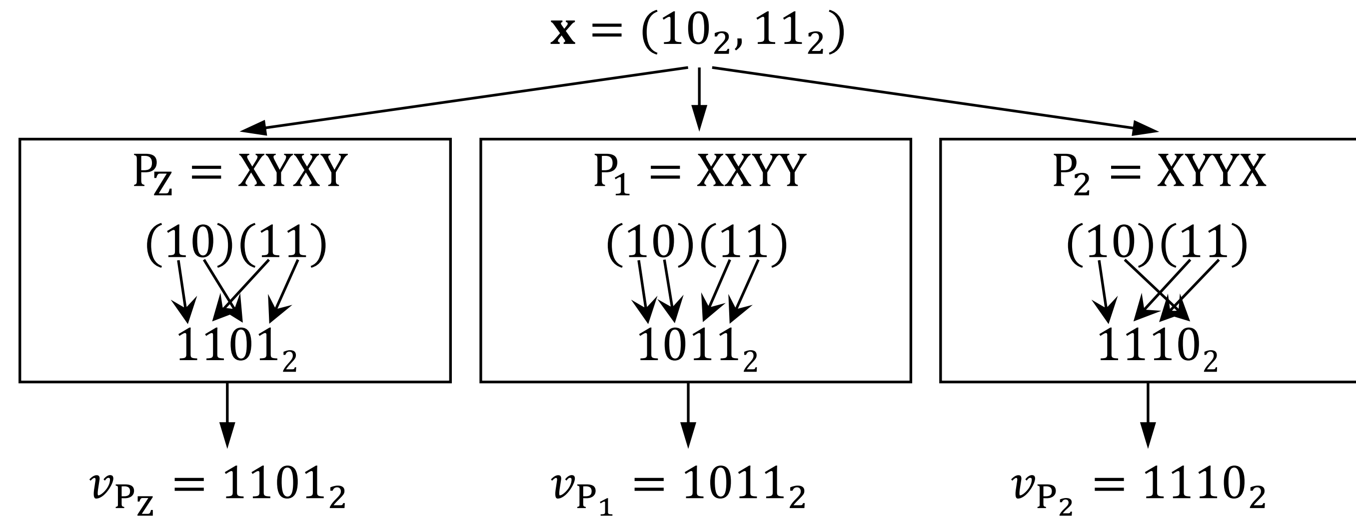
Problem Statement

- Database D
  - Each data point $\mathbf{x} \in D$, has n dimensions, denoted by $\mathbf{x} = (d_1, d_2, \dots, d_n)$
- Query Workload Q
  - Each query $q \in Q$, $q = (x_{\min}, y_{\min}, x_{\max}, y_{\max})$
- Space-Filling Curve Design for Query Processing
  - Given a database D and a query workload Q, we aim to develop a mapping function T, which maps each data point $\mathbf{x} \in D$ into an SFC value v, s.t. with an index structure (e.g., B+ Tree) built on the SFC values of data points in D, the query performance (e.g., I/O and query latency) on Q is optimized.

Our Method

Bit Merging Pattern (BMP) [1]

- The bit merging pattern describes a set of bit merging-based SFCs.
  - Idea: The input data is first written as the binary form, then merge the bit according to the pattern (e.g., XYXY)



[1] Shoji Nishimura and Haruo Yokota. 2017. QUILTS: Multidimensional Data Partitioning Framework Based on Query-Aware and Skew-Tolerant Space-Filling Curves. SIGMOD2017

Desired Properties

- Two preferred properties for an SFC mapping $T: \mathbf{x} \rightarrow v$
  - Injection property:

$$\forall \mathbf{x}_1 \neq \mathbf{x}_2, T(\mathbf{x}_1) \neq T(\mathbf{x}_2)$$

  - Monotonicity property:

$$\mathbf{x}' = \{b'_1, \dots, b'_n\}$$
$$\mathbf{x}'' = \{b''_1, \dots, b''_n\}$$

If $d'_i \geq d''_i$ is satisfied for $\forall i \in [1, n]$:

$$T(\mathbf{x}') \geq T(\mathbf{x}'')$$

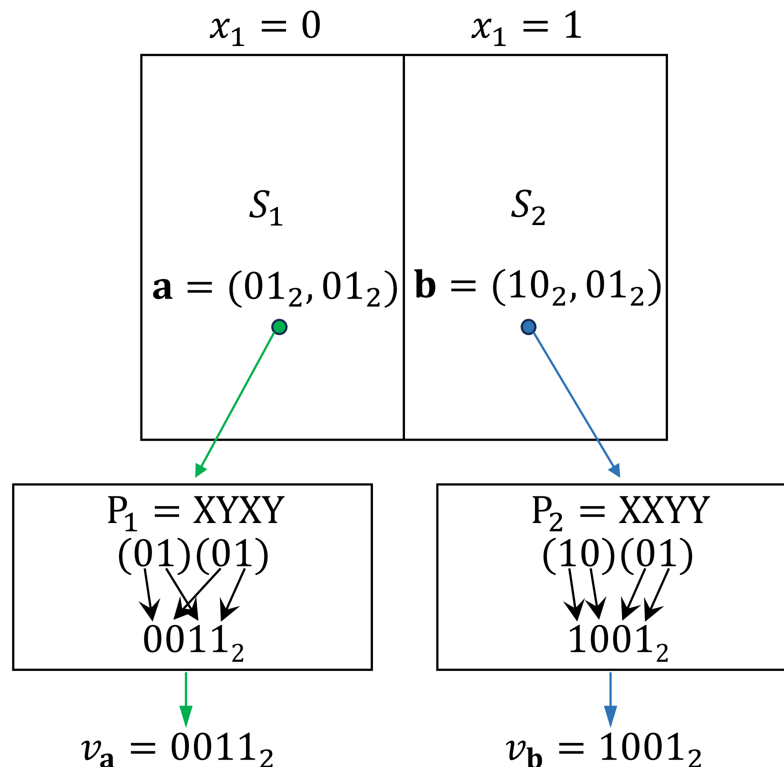
Monotonicity is desirable for designing window query algorithms:
It guarantees that the SFC values of data points in a query rectangle fall in the range of the SFC values formed by two boundary points of the query rectangle

Design Challenges

1. How to partition the space and design an effective BMP for each subspace?
2. How to design piecewise SFCs such that two desirable properties hold?
3. How to design a data-driven approach to build the piecewise SFC, given a database and query workload?

Piecewise SFC Design

- We propose a way of seamlessly integrating the subspace partitioning and BMP generation while ensuring the desired properties.

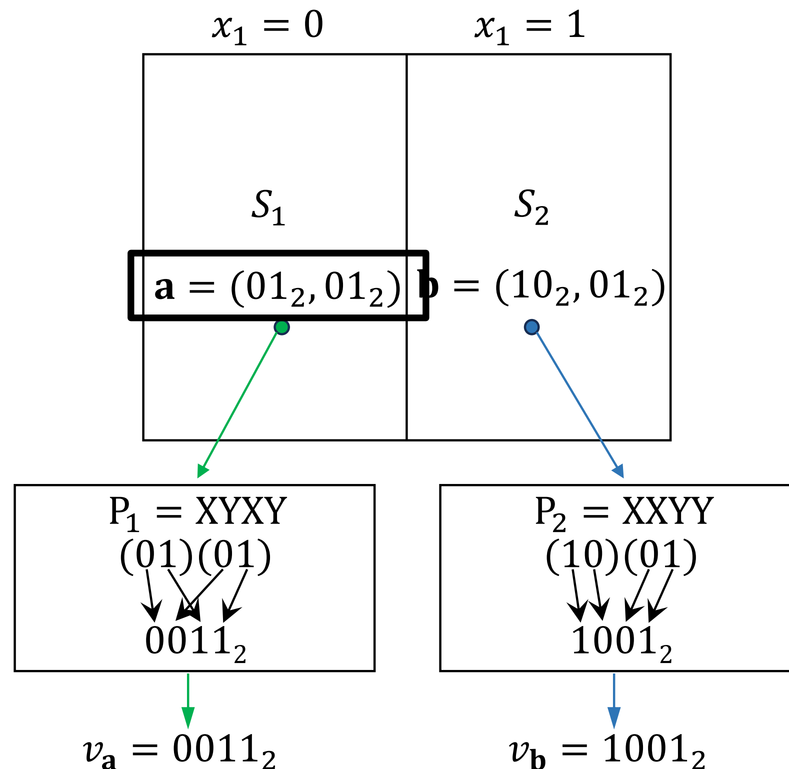


(a) Example of Piecewise SFC Design.

- We follow the left-to-right BMP design, and start with an empty string P, then we choose a bit X.
- Then the whole data space is partitioned into two subspaces w.r.t. the value of bit x_1, where one subspace corresponds to $x_1 = 0$ (resp. $x_1 = 1$).
- This partitioning enables us to separately design different BMPs for the two subspaces (S_1 and S_2).

Piecewise SFC Design

- We propose a way of seamlessly integrating the subspace partitioning and BMP generation while ensuring the desired properties.

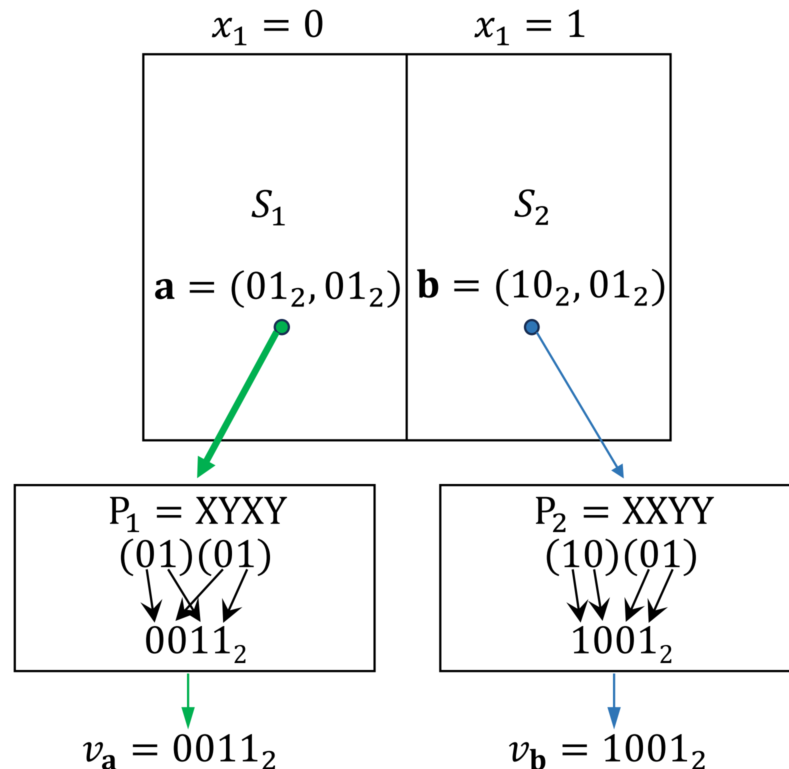


(a) Example of Piecewise SFC Design.

- We follow the left-to-right BMP design, and start with an empty string P, then we choose a bit X.
- Then the whole data space is partitioned into two subspaces w.r.t. the value of bit x_1, where one subspace corresponds to $x_1 = 0$ (resp. $x_1 = 1$).
- This partitioning enables us to separately design different BMPs for the two subspaces (S_1 and S_2).

Piecewise SFC Design

- We propose a way of seamlessly integrating the subspace partitioning and BMP generation while ensuring the desired properties.

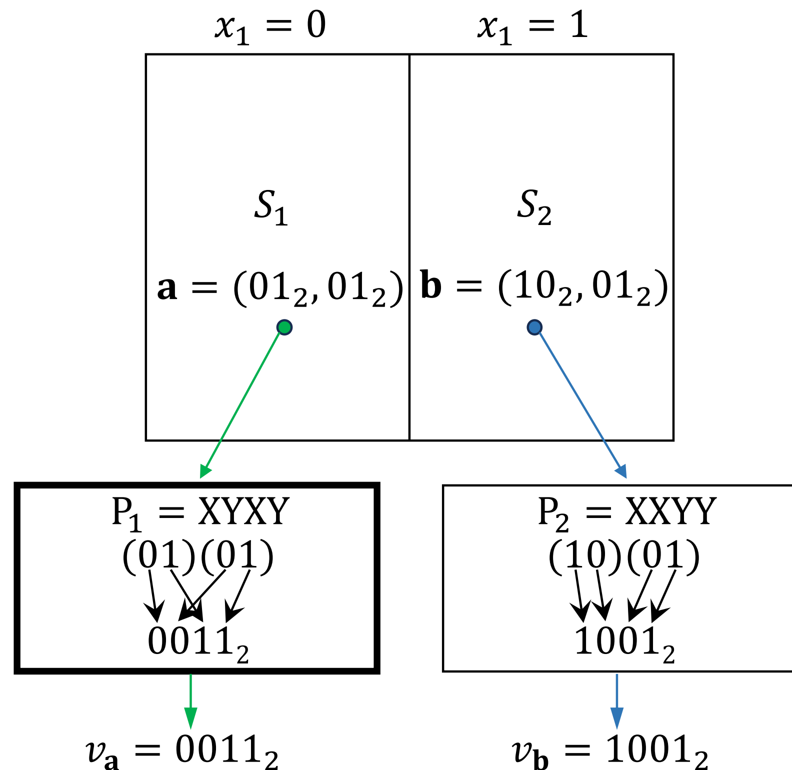


(a) Example of Piecewise SFC Design.

- We follow the left-to-right BMP design, and start with an empty string P, then we choose a bit X.
- Then the whole data space is partitioned into two subspaces w.r.t. the value of bit x_1, where one subspace corresponds to $x_1 = 0$ (resp. $x_1 = 1$).
- This partitioning enables us to separately design different BMPs for the two subspaces (S_1 and S_2).

Piecewise SFC Design

- We propose a way of seamlessly integrating the subspace partitioning and BMP generation while ensuring the desired properties.

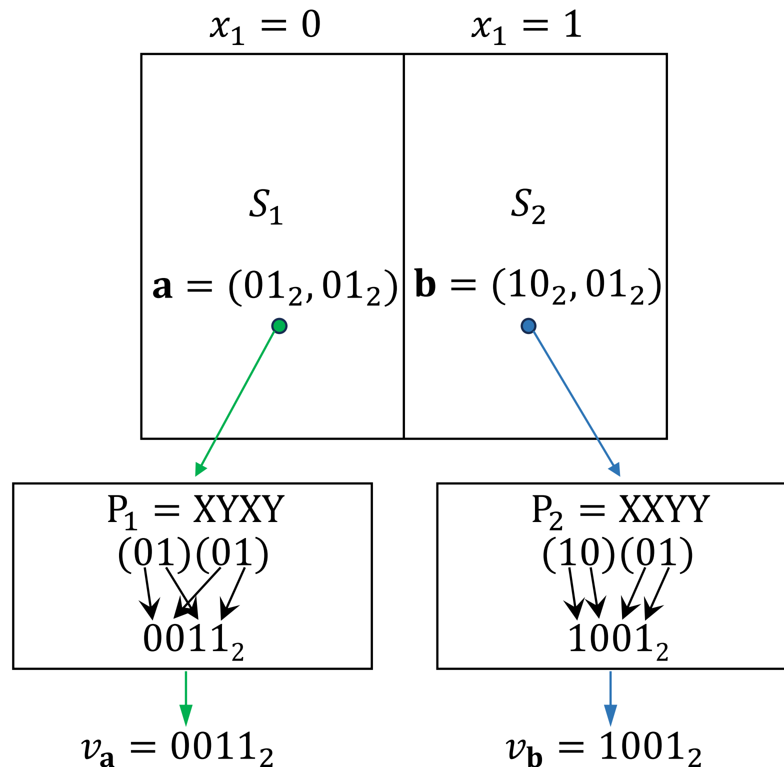


(a) Example of Piecewise SFC Design.

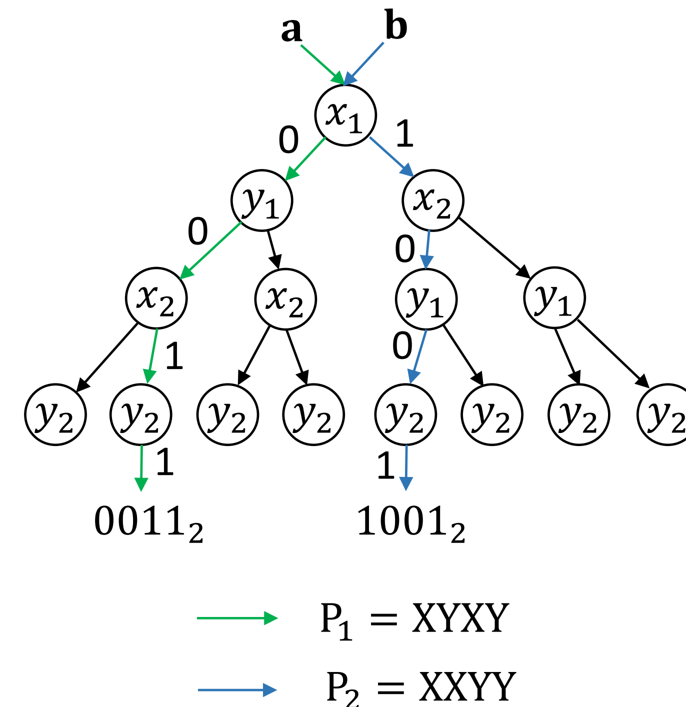
- We follow the left-to-right BMP design, and start with an empty string P, then we choose a bit X.
- Then the whole data space is partitioned into two subspaces w.r.t. the value of bit x_1, where one subspace corresponds to $x_1 = 0$ (resp. $x_1 = 1$).
- This partitioning enables us to separately design different BMPs for the two subspaces (S_1 and S_2).

Bit Merging Tree (BMTree)

- The BMTree is to model the partition and BMP design information of a piecewise SFC.



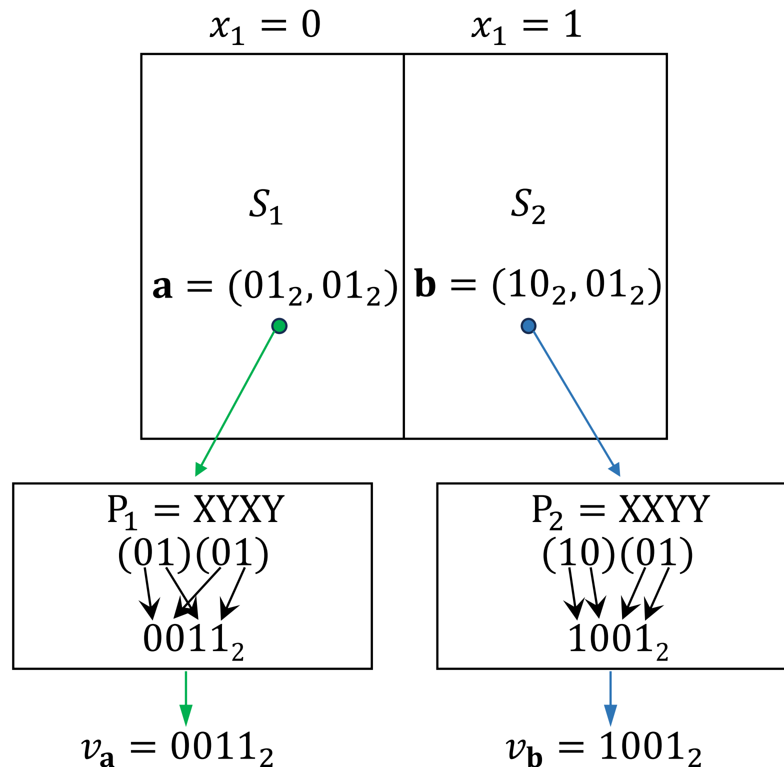
(a) Example of Piecewise SFC Design.



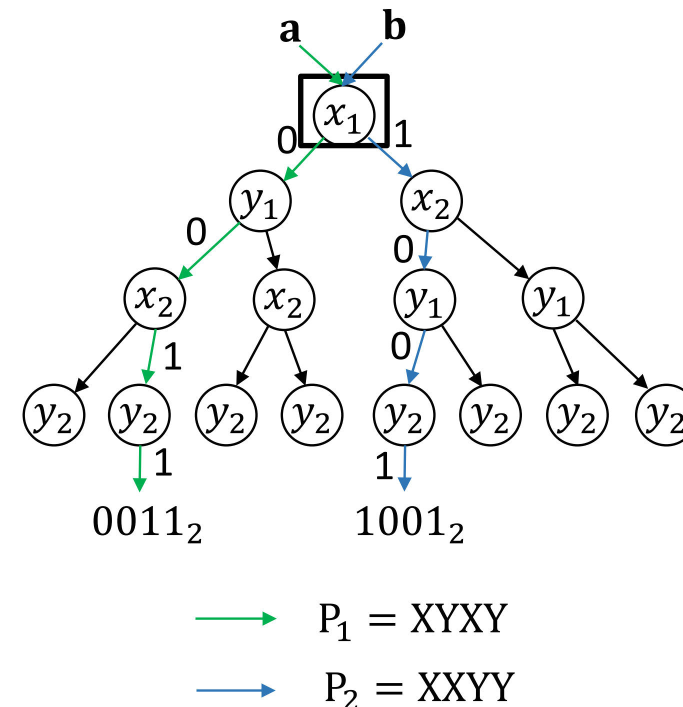
(b) Example of BMTree Structure.

Bit Merging Tree (BMTree)

- The BMTree is to model the partition and BMP design information of a piecewise SFC.



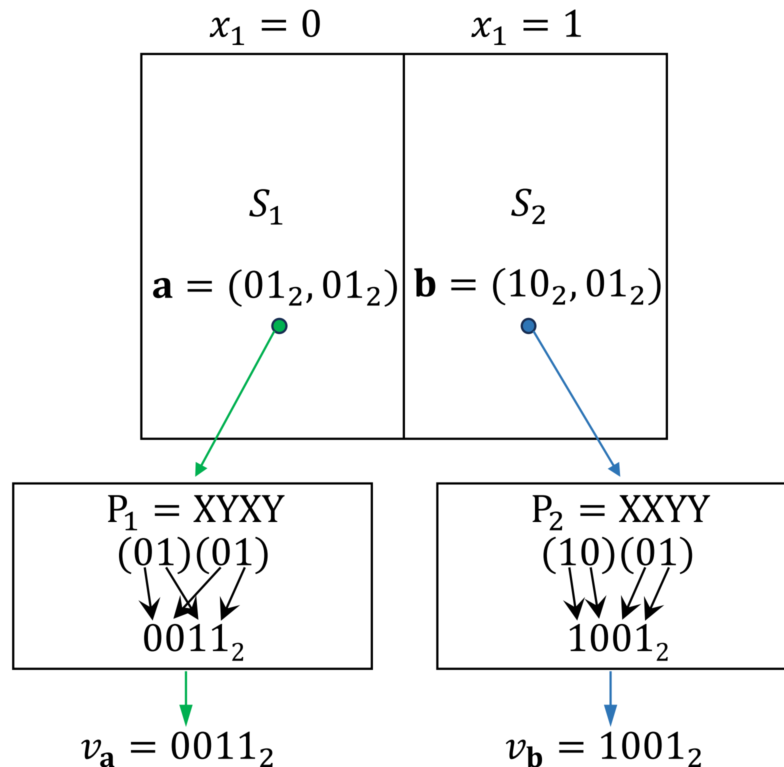
(a) Example of Piecewise SFC Design.



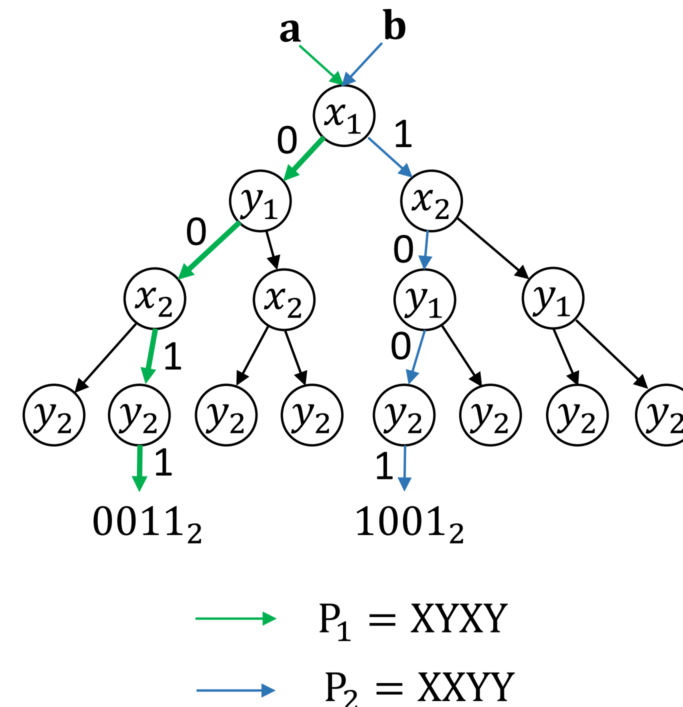
(b) Example of BMTree Structure.

Bit Merging Tree (BMTree)

- The BMTree is to model the partition and BMP design information of a piecewise SFC.



(a) Example of Piecewise SFC Design.

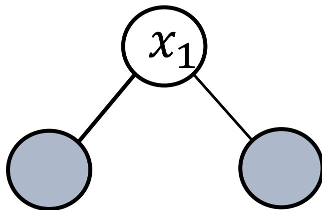


(b) Example of BMTree Structure.

BMTree Construction

- We model the SFC design procedure as the BMTree construction procedure.
  - During the BMTree construction, each time we fill one level of BMTree with the selected bits, which also partition more subspaces and generate the next level of leaf nodes.

(1) BMTree whose root node is filled with x_1



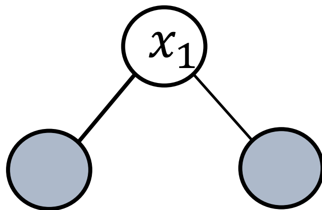
(2) Possible bit choices to fill the two leaf nodes

1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

BMTree Construction

- We model the SFC design procedure as the BMTree construction procedure.
  - During the BMTree construction, each time we fill one level of BMTree with the selected bits, which also partition more subspaces and generate the next level of leaf nodes.

(1) BMTree whose root node is filled with x_1



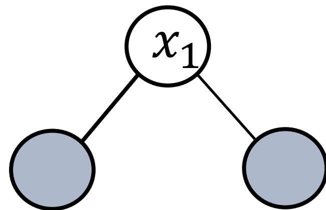
(2) Possible bit choices to fill the two leaf nodes

1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

BMTree Construction

- We model the SFC design procedure as the BMTree construction procedure.
  - During the BMTree construction, each time we fill one level of BMTree with the selected bits, which also partition more subspaces and generate the next level of leaf nodes.

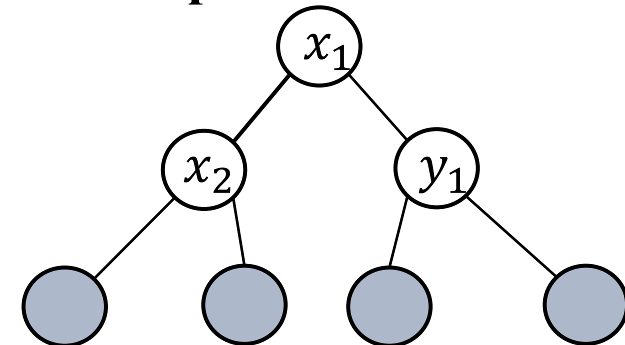
(1) BMTree whose root node is filled with x_1



(2) Possible bit choices to fill the two leaf nodes

1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

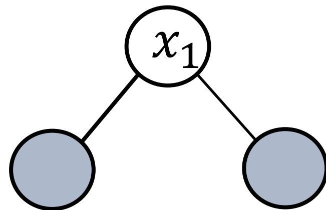
(3) BMTree constructed one level deeper



BMTree Construction

- We model the SFC design procedure as the BMTree construction procedure.
  - During the BMTree construction, each time we fill one level of BMTree with the selected bits, which also partition more subspaces and generate the next level of leaf nodes.

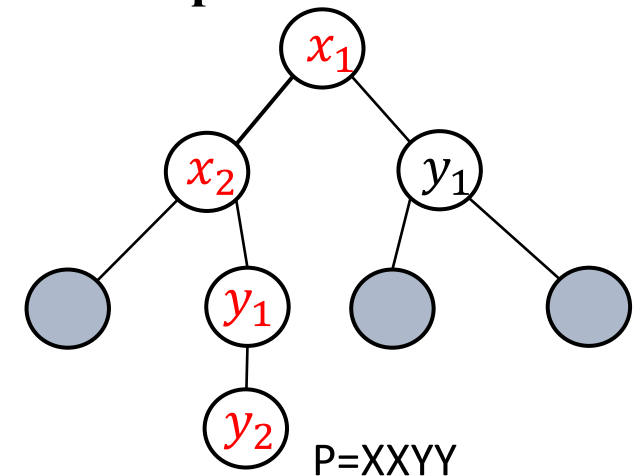
(1) BMTree whose root node is filled with x_1



(2) Possible bit choices to fill the two leaf nodes

1. Left: x_2, Right: x_2
2. Left: x_2, Right: y_1
3. Left: y_1, Right x_2
4. Left: y_1, Right y_1

(3) BMTree constructed one level deeper

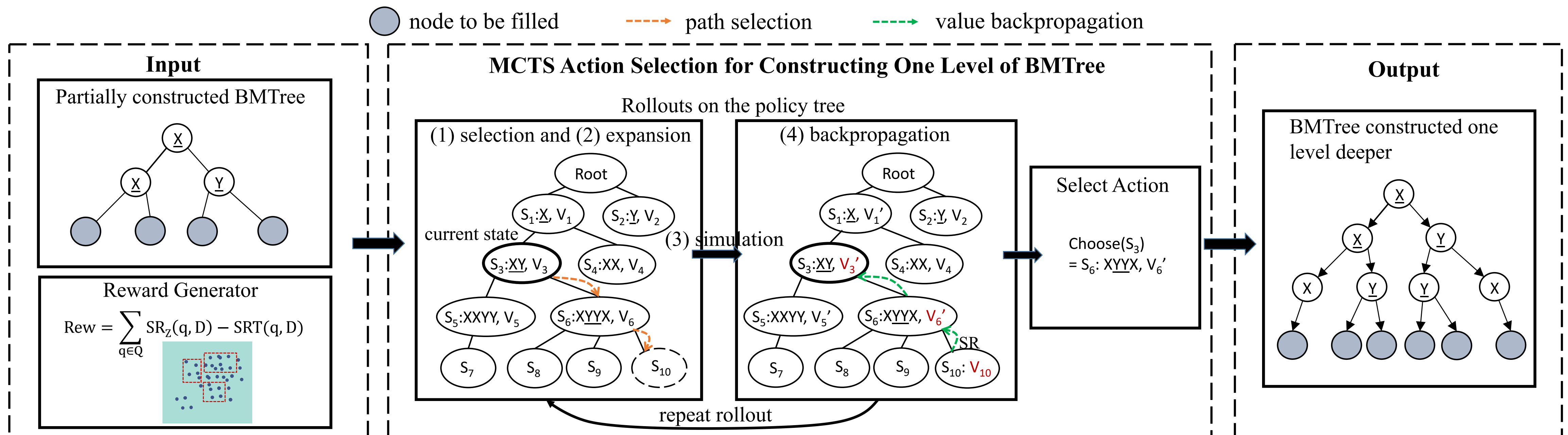


Use Reinforcement Learning to construct BMTree

- The reason why use reinforcement learning:
  - Heuristic methods are difficult to be designed to construct BMTree to optimize the query performance for a workload on a database instance.
  - Utilizing reinforcement learning could directly optimize the BMTree based on the reward.

MCTS based BMTree Construction

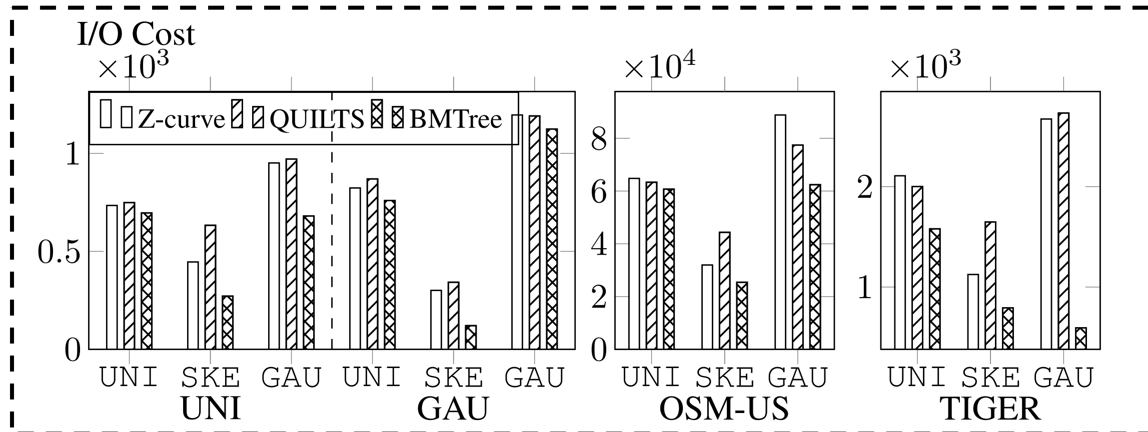
- We leverage Monte Carlo Tree Search method to help constructing BMTree.



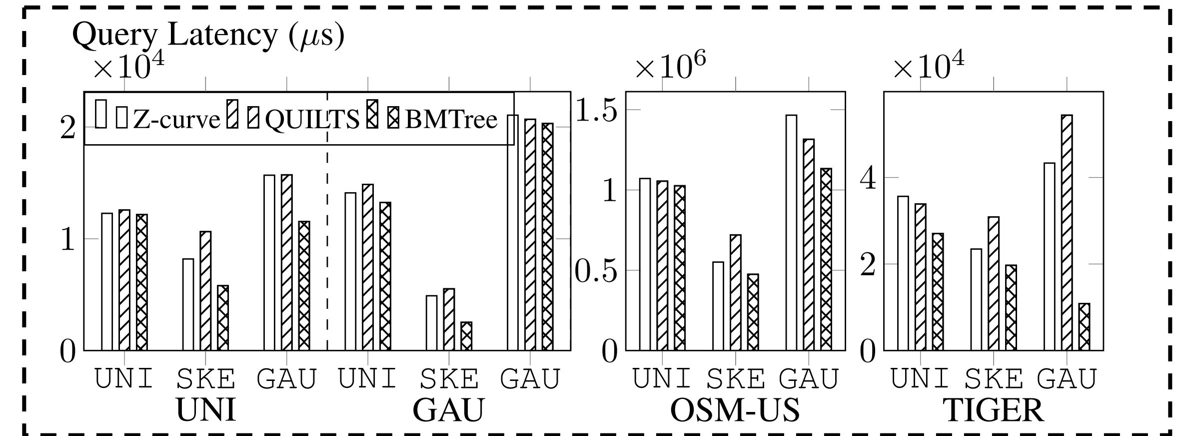
Experiment

Comparing between SFCs

- Experiment on PostgreSQL.



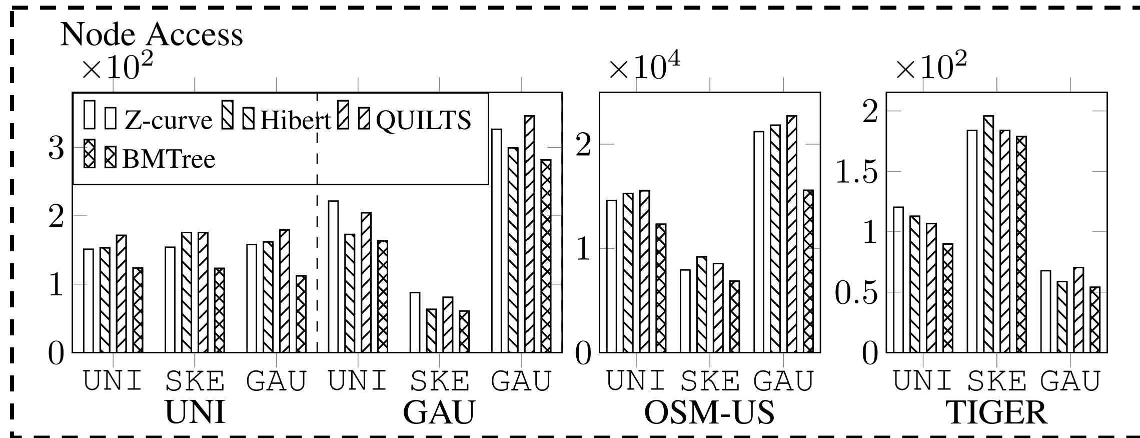
(a) I/O Cost



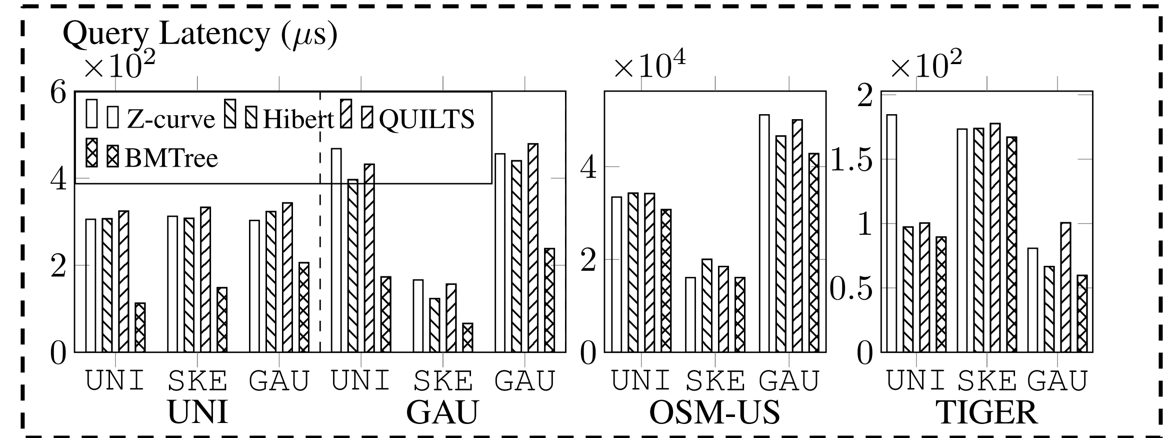
(b) Query Latency

Comparing between SFCs

- Experiment on RSMI [2] (a learned index).



(a) Node Access

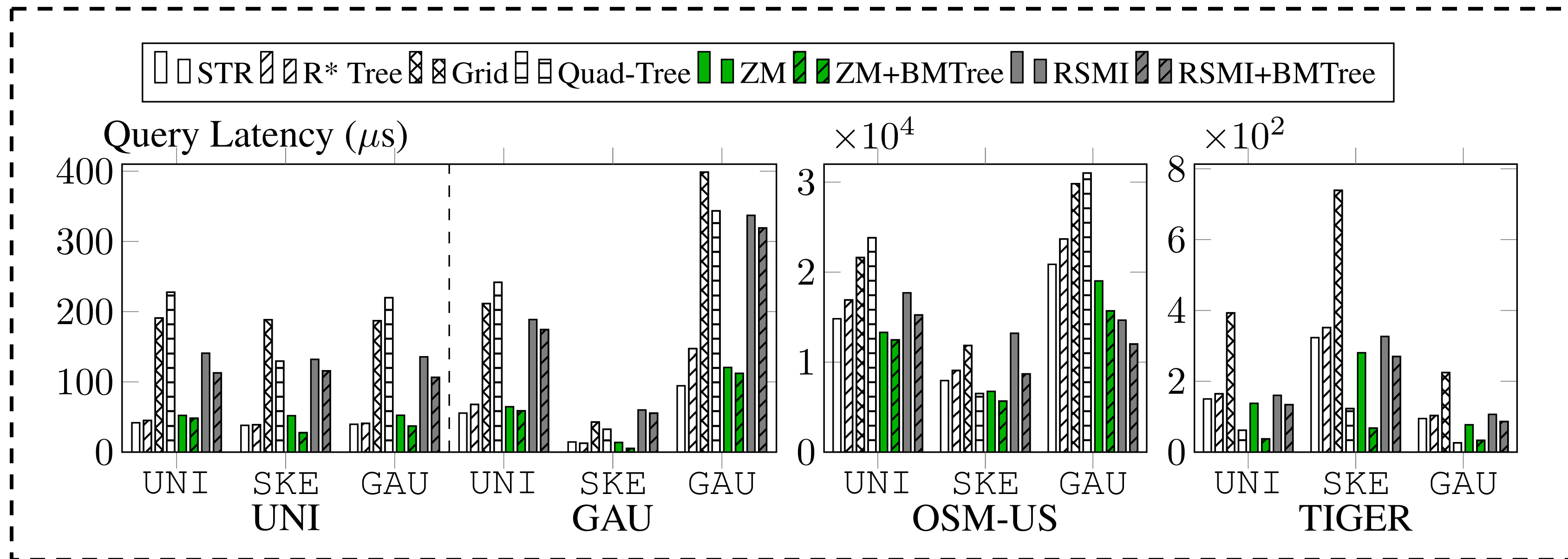


(b) Query Latency

[2] Jianzhong Qi, Guanli Liu, Christian S. Jensen, and Lars Kulik. 2020. Effectively Learning Spatial Indices.

Comparing between indexes

- Comparing traditional spatial indexes with BMTree-enhanced one-dimensional indexes



Conclusion and Takeaways

- Why the idea of piecewise SFC would work
  - The design of the BMTree considered a SFC set with a large size, which inherently contains a better SFC.
  - The idea of piecewise enables the policy to adapt the mapping schemes of subspaces depending on the specific database instance situation.

Thank you

Questions?