CLOSET+:Searching for the Best Strategies for Mining Frequent Closed Itemsets

Jianyong Wang, Jiawei Han, Jian Pei

Presentation by:

Nasimeh Asgarian

Department of Computing Science
University of Alberta

Outline

- Introduction
- Strategies for frequent closed itemset mining
- Overview of CLOSET+
  - The hybrid tree projection
  - Item skipping technique
  - Efficient subset checking
- The algorithm
- Performance evaluation

Introduction

- There are several algorithms for finding frequent itemset, like Apriori.
  - They have good performance when the supported threshold is large.

Introduction

- There are several algorithms for finding frequent itemset, like Apriori.
  - ⋆ They have good performance when the supported threshold is large.

- Frequent closed itemset: frequent items that have no proper superset with the same support $\Rightarrow$ no redundancy.

Introduction

- There are several algorithms for finding frequent itemset, like Apriori.
  - ⋆ They have good performance when the supported threshold is large.

- Frequent closed itemset: frequent items that have no proper superset with the same support $\Rightarrow$ no redundancy.

- There are several algorithms for finding frequent closed itemsets, like CLOSET, CHARM, OP.

Introduction

- There are several algorithms for finding frequent itemset, like Apriori.
  - ⋆ They have good performance when the supported threshold is large.

- Frequent closed itemset: frequent items that have no proper superset with the same support $\Rightarrow$ no redundancy.
- There are several algorithms for finding frequent closed itemsets, like CLOSET, CHARM, OP.
- They find positive and negative aspects of the existing techniques.

Introduction

- There are several algorithms for finding frequent itemset, like Apriori.
  - ⋆ They have good performance when the supported threshold is large.

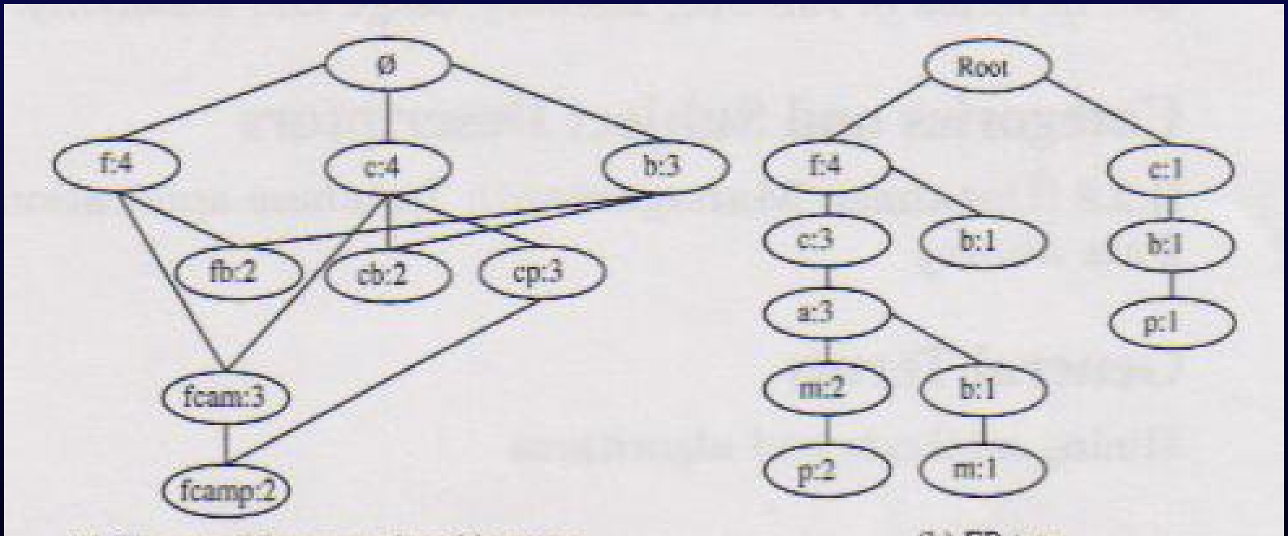
- Frequent closed itemset: frequent items that have no proper superset with the same support $\Rightarrow$ no redundancy.
- There are several algorithms for finding frequent closed itemsets, like CLOSET, CHARM, OP.
- They find positive and negative aspects of the existing techniques.
- Introduce new techniques and an algorithm, CLOSET+.

Introduction

- There are several algorithms for finding frequent itemset, like Apriori.
  - ⋆ They have good performance when the supported threshold is large.
- Frequent closed itemset: frequent items that have no proper superset with the same support $\Rightarrow$ no redundancy.
- There are several algorithms for finding frequent closed itemsets, like CLOSET, CHARM, OP.
- They find positive and negative aspects of the existing techniques.
- Introduce new techniques and an algorithm, CLOSET+.
- Compare their algorithm with other algorithms in terms of runtime, memory usage, and scalability.

Running example

Tid	set of Items	ordered frequent item list
100	a,c,f,m,p	f,c,a,m,p
200	a,c,d,f,m,p	f,c,a,m,p
300	a,b,c,f,g,m	f,c,a,b,m
400	b,f,t	f,b
500	b,c,n,p	c,b,p



(a) The set of frequent closed itemsets (b) FP-tree

Strategies for Frequent Itemset Mining

Breath-first search vs. Depth-first search

Strategies for Frequent Itemset Mining

Breath-first search vs. Depth-first search

- BFS methods use the frequent itemsets at level $k-1$ to generate candidates at level k. They have to scan the database to find the support for candidates at level k.

Strategies for Frequent Itemset Mining

Breath-first search vs. Depth-first search

- BFS methods use the frequent itemsets at level $k-1$ to generate candidates at level k. They have to scan the database to find the support for candidates at level k.
- DFS methods search the subtree of an itemset only if the itemset is frequent. When the itemsets becomes longer, DFS shrinks the search space quickly.

Strategies for Frequent Itemset Mining

Breath-first search vs. Depth-first search

- BFS methods use the frequent itemsets at level $k-1$ to generate candidates at level k. They have to scan the database to find the support for candidates at level k.
- DFS methods search the subtree of an itemset only if the itemset is frequent. When the itemsets becomes longer, DFS shrinks the search space quickly.

DFS is the winner for databases with long patterns.

Horizontal vs. Vertical formats

Horizontal vs. Vertical formats

- Vertical format: a *tid-list* is kept for each item, which can be large for dense datasets. To find the frequent itemsets, they have to find the intersection of tid-lists (which is costly), and with each intersection they find only one frequent itemset.

Horizontal vs. Vertical formats

- Vertical format: a *tid-list* is kept for each item, which can be large for dense datasets. To find the frequent itemsets, they have to find the intersection of tid-lists (which is costly), and with each intersection they find only one frequent itemset.

- Horizontal format: each transaction recorded as a list of items. They require less space, and with each scan of the database, they find many frequent itemsets which can be used to grow the prefix itemsets to generate frequent itemsets.

Data Compression Techniques

Data Compression Techniques

- diffset is data compression technique for vertical format recorded transactions.
  It only keeps track of the differences in tids of a candidate from its parent.

Data Compression Techniques

- diffset is data compression technique for vertical format recorded transactions.
  It only keeps track of the differences in tids of a candidate from its parent.

- `FP-tree` of a transaction database is a prefix tree of the list of frequent items in transaction. It is data compression technique for horizontal format recorded transactions. It has several advantages in finding frequent itemsets:
  - ⋆ infrequent items found in the first database scan won't be used in tree construction.
  - ⋆ a set of transactions sharing the same subset of items may share common prefix path from the root in an `FP-tree`.
  - ⋆ Its compression ratio can reach several thousand even for sparse datasets.

Pruning Techniques for closed itemset mining

Pruning Techniques for closed itemset mining

- Lemma 3.1. Item merging: Let X be a frequent itemset. If every transaction containing itemset X also contains itemset Y but not any proper superset of Y, they $X \cup Y$ forms a frequent closed itemset and there is no need to search any itemset containing X but not Y.

Pruning Techniques for closed itemset mining

- Lemma 3.1. Item merging: Let X be a frequent itemset. If every transaction containing itemset X also contains itemset Y but not any proper superset of Y, they $X \cup Y$ forms a frequent closed itemset and there is no need to search any itemset containing X but not Y.

- Lemma 3.2. Sub-itemset pruning: Let X be a frequent itemset currently under consideration. If X is a proper subset of an already found frequent closed itemset Y and support(X) $=$ support(Y), then X and all of X's descendants can not be frequent closed itemsets and thus can be pruned.

Overview of CLOSET+

- Divide-and conquer paradigm
- Depth-first search strategy
- Horizontal format-based
- `FP-tree` as compression technique
- Hybrid tree-projection method to improve the space efficiency
- Both pruning techniques plus a new technique: item skipping
- Efficient subset checking method to save memory usage and speed up closure checking. (Previous algorithms need to maintain all frequent closed itemset found so far in order to check if newly found frequent closed itemset is really closed).

The Hybrid Tree Projection Method

The Hybrid Tree Projection Method

- Bottom-up physical tree-projection
  - ⋆ For dense datasets.
  - ⋆ CLOSET+ builds projected FP-tree in support ascending order
  - ⋆ There is a header table for each FP-tree, which holds each item’s ID, count, and a side-link pointer that links all the nodes with the same itemID as the labels.

The Hybrid Tree Projection Method

- Bottom-up physical tree-projection
  - ⋆ For dense datasets.
  - ⋆ CLOSET+ builds projected FP-tree in support ascending order
  - ⋆ There is a header table for each FP-tree, which holds each item's ID, count, and a side-link pointer that links all the nodes with the same itemID as the labels.

- Top-down pseudo tree-projection
  - ⋆ For sparse datasets.
  - ⋆ CLOSET+ builds projected FP-tree in support descending order
  - ⋆ There is a header table for each FP-tree, which holds local frequent items, their counts, and a side-link pointer to FP-tree nodes in order to locate the subtrees for a certain prefix itemset.

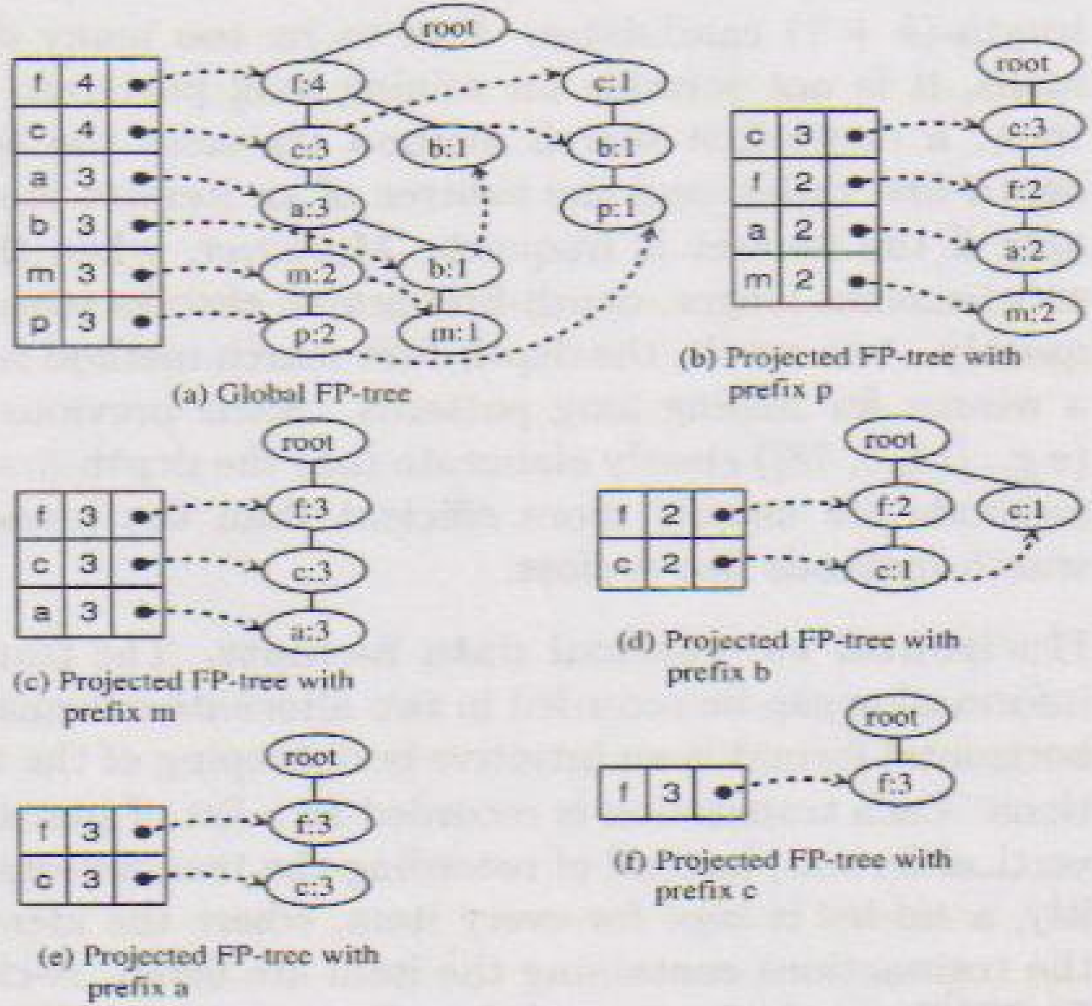
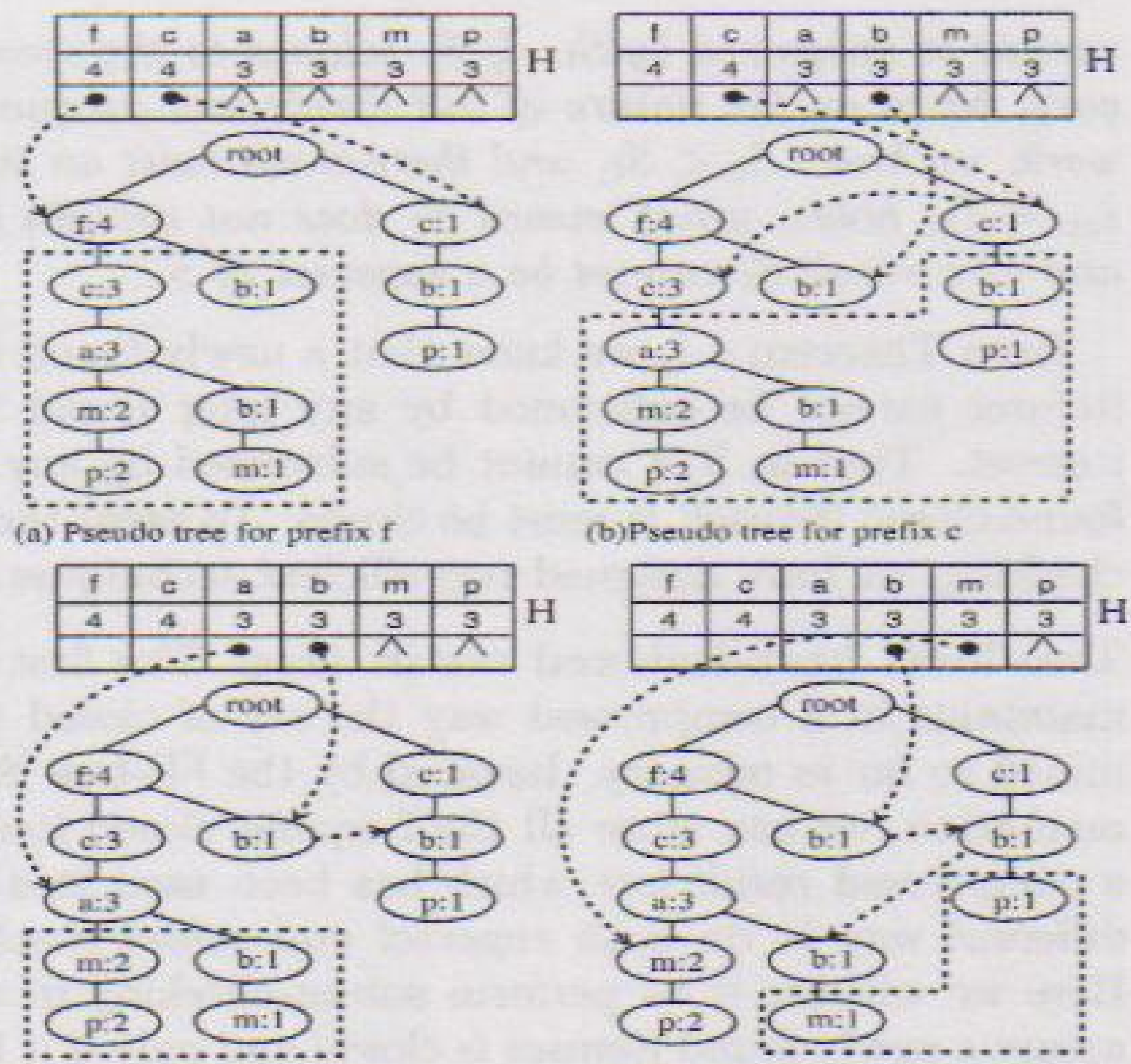


Figure 2: Bottom-up physical tree-projection.



(a) Pseudo tree for prefix f

(b)Pseudo tree for prefix c

(c)Pseudo tree for prefix a

(d)Pseudo tree for prefix b

The item Skipping Technique

- Lemma 4.1. (Item skipping) If a local frequent item has the same support in several header tables at different levels, one can safely prune it from the header tables at the higher levels.

The item Skipping Technique

- Lemma 4.1. (Item skipping) If a local frequent item has the same support in several header tables at different levels, one can safely prune it from the header tables at the higher levels.

- Example:

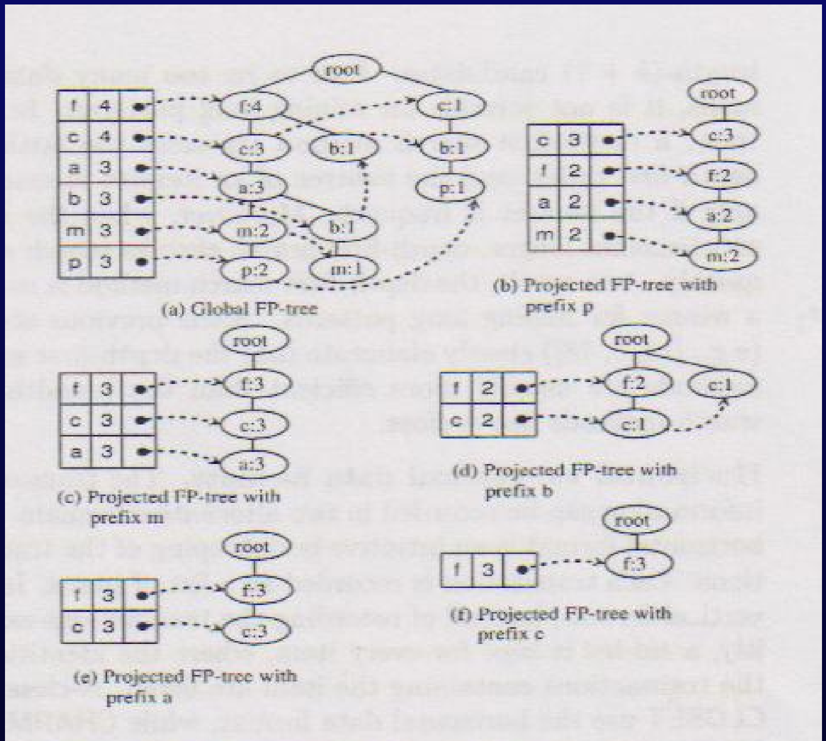


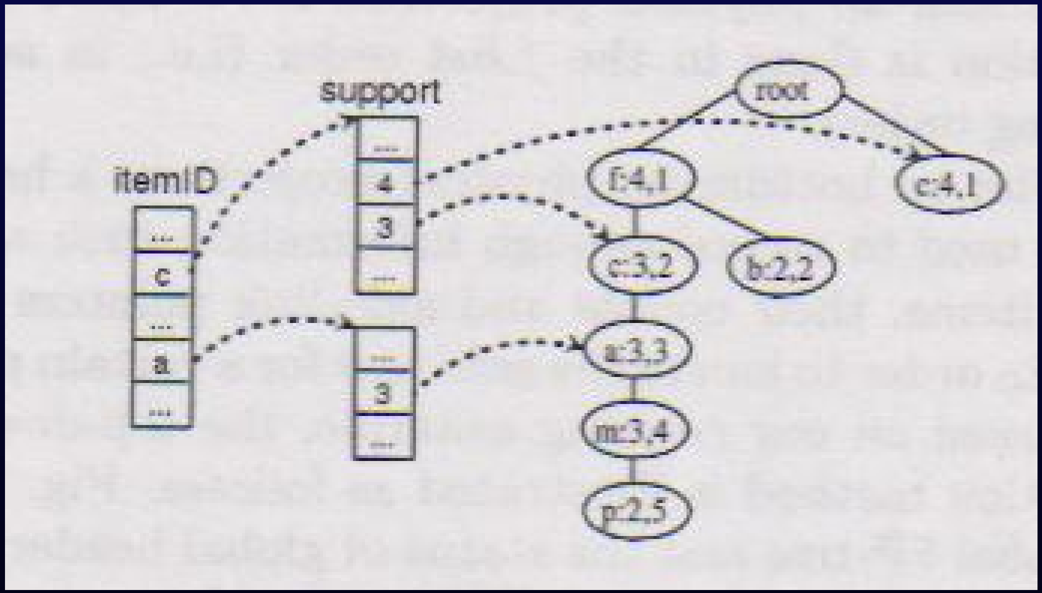
Figure 2: Bottom-up physical tree-projection.

Efficient Subset Checking

- Superset checking: Checks if the new frequent itemset is a superset of some already found closed itemset candidate with the same support.
- Subset checking: Checks if the new frequent itemset is a superset of some already found closed itemset candidate with the same support.
- CLOSET+: Only needs to do subset checking (Theorem 4.1.)

- Two-level hash indexed result tree
  - For dense datasets.
  - Keeps the set of closed itemsets in a compressed way.
  - One level uses ID of the last item in current itemset, S_c as hash key.
  - The other uses support of S_c as hash key.
  - Insert each closed itemset into result tree according to *f-list*, at each node record its length of the path from this node to the root.

- Two-level hash indexed result tree
  - For dense datasets.
  - Keeps the set of closed itemsets in a compressed way.
  - One level uses ID of the last item in current itemset, S_c as hash key.
  - The other uses support of S_c as hash key.
  - Insert each closed itemset into result tree according to *f-list*, at each node record its length of the path from this node to the root.



- Pseudo-projection based upward checking
  - ⋆ For sparse datasets.
  - ⋆ Global `FP-tree` has the complete information for the database $\Rightarrow$ no need to store closed itemsets in memory.
  - ⋆ How to do subset checking?
  - ⋆ Lemma 4.2. For a certain prefix itemset, X, as long as we can find any item which (1) appears in each prefix path w.r.t. prefix itemset X, and (2) does not belong to X, any itemset with prefix X will be non-closed. Otherwise, the union of X and the complete set of its local frequent items which have the same support as X will form a closed itemset.

The Algorithm

input: a transaction database TDB and the support threshold.
output: the complete set of frequent closed itemsets.

1. Scan TDB to find the frequent itemsets, sort them in support descending order.

2. Scan TDB and build the `FP-tree`, find the average count of an `FP-tree` node to judge if the data set is dense of sparse.

3. With divide-and-conquer and depth-first search mine the `FP-tree` for frequent closed itemsets in a *top-down* manner for sparse datasets and *bottom-up* manner for dense datasets. Use the efficient subset checking techniques to do closure checking.

4. Stop when all items in global header table have been mined.

Performance Evaluation

They have tested their algorithm on both sparse and dense datasets and compared it with `OP`, `CHARM`, and `CLOSET`.

Performance Evaluation

They have tested their algorithm on both sparse and dense datasets and compared it with `OP`, `CHARM`, and `CLOSET`.

Sparse datasets: `OP` is faster than `CLOSET+` when threshold is high, but it reverses for low thresholds. `CHARM` is sometimes faster, but `CLOSET+` uses less memory when threshold is low. `CLOSET+` always performs better than `CLOSET`.

Performance Evaluation

They have tested their algorithm on both sparse and dense datasets and compared it with `OP`, `CHARM`, and `CLOSET`.

Sparse datasets: `OP` is faster than `CLOSET+` when threshold is high, but it reverses for low thresholds. `CHARM` is sometimes faster, but `CLOSET+` uses less memory when threshold is low. `CLOSET+` always performs better than `CLOSET`.

Dense datasets: `CLOSET+` is faster than `OP` and `CLOSET`, specially when supported threshold is low. `CHARM` performs similarly in terms of runtime, but `CLOSET+` uses less memory.

Performance Evaluation

They have tested their algorithm on both sparse and dense datasets and compared it with `OP`, `CHARM`, and `CLOSET`.

Sparse datasets: `OP` is faster than `CLOSET+` when threshold is high, but it reverses for low thresholds. `CHARM` is sometimes faster, but `CLOSET+` uses less memory when threshold is low. `CLOSET+` always performs better than `CLOSET`.

Dense datasets: `CLOSET+` is faster than `OP` and `CLOSET`, specially when supported threshold is low. `CHARM` performs similarly in terms of runtime, but `CLOSET+` uses less memory.

Scalability: `CLOSET+` has better performance than `CHARM` and `CLOSET` in terms of scalability in both database size and number of distinct items.

Thanks!