Lecture Title: Beyond Itemsets – Mining Complex Patterns with Constraints

Duration: 1 Hour

Target Audience: Data Science/Computer Science students

Prerequisites: Understanding of basic association rule mining, Apriori/FP-Growth algorithms

Advanced Pattern Mining

Beyond the Market Basket

Mining Constraints, Sequences, and Graphs

Context: Part 1: Introduction & Learning Objectives

Opening Scenario

Real-World Data is Richer

The Basics

We have mastered finding simple itemsets like {Milk} → {Bread} in supermarket data.

The Reality

- **Sequences:** Customers don't just buy; they follow paths (e.g., Website Clickstreams: Home → Search → Checkout)
- **Structures:** Data often has shape (e.g., Molecular compounds in drug discovery)
- **Constraints:** We rarely want all patterns; we need focused mining (e.g., Only patterns where Profit > $100)

The Frontiers of Pattern Mining

Three Key Directions

Constraint-Based Mining

Mining with specific restrictions (Profit, Price, Category) to filter noise.

Sequential Pattern Mining

Discovering patterns in ordered data (Time-series, DNA, Logs).

Graph Pattern Mining

Discovering frequent substructures in connected data (Social networks, Chemical structures).

Learning Objectives

By the end of this module, you will be able to:

Understand Constraints

Apply constraint-based mining and pruning strategies to focus your search.

Discover Graphs

Identify frequent subgraphs from structured data (e.g., Apriori-based Graph Mining).

Mine Sequences

Discover sequential patterns from ordered data (e.g., GSP algorithm).

Apply Algorithms

Select and apply the appropriate algorithm for each specific pattern type.

Part 2: Constraint-Based Pattern Mining | Section 5.3

Pruning the Search Space

Focus: Using Data and Mining Constraints to find relevant patterns efficiently.

The Problem & Solution

Why Mine Everything?

The Problem:

Algorithms like Apriori and FP-Growth find ALL frequent patterns.

Result:

Thousands of patterns, most of which are irrelevant to the user.

The Solution:

Apply constraints during the mining process (Constraint Pushing), not after.

Benefit:

Drastically reduces the search space and improves performance.

Types of Constraints

Filtering at Different Levels

Data Constraints

Focus on specific data subsets.

Examples:

- Mine only transactions from 2024
- Mine only customers aged 25-34

Action:

Applied during data access; reduces dataset size upfront.

Pattern Constraints

Specify pattern characteristics.

Examples:

- Pattern must contain ‘organic’ items
- Length ≤ 5 items

Aggregate Constraints

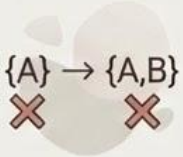
Based on aggregate properties of the items.

Examples:

- Average price > $50
- Total profit > $100

Pruning Power: Anti-monotonic Constraints

The “Downward Closure” Property



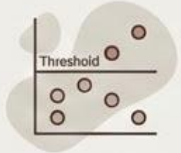
Definition

If a constraint fails for an itemset, it fails for ALL supersets.

Impact

Enables powerful pruning.

If {A, B} violates the constraint, we can prune {A, B, C}, {A, B, D}, etc. without checking them.

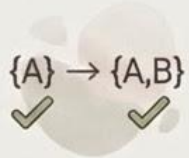


Examples

- min_sup (Frequency).
- max(price) < $100 (If max price of {A} is $150, any superset will also have a max ≥ $150).

Pruning Power: Monotonic Constraints

The “Upward Closure” Property



Definition

If a constraint holds for an itemset, it holds for ALL supersets.

Impact

Allows us to stop checking specific branches once the condition is met (or prune in the opposite direction).

Examples

- contains(‘laptop’). (If {A} contains a laptop, {A, B} also contains a laptop).
- sum(price) > $100. (If {A} costs $150, any superset will cost > $150).

Constraint Mining in Action

SQL-Like Example

Combining constraints for focused mining:

```
SELECT * FROM patterns
WHERE support >= 0.01
  AND contains('electronics')
  AND avg(price) BETWEEN 50 AND 200
```

-- Anti-monotonic

-- Monotonic

-- Aggregate / Convertible

Mining Space Pruning with Succinct Constraints

Section 5.3.3: Precise Generation

Succinct Constraints

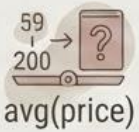
Can be satisfied by a small, pre-determined set of patterns.

Expressed directly in terms of items.

Example: "Pattern must contain at least one item from {laptop, phone, tablet}."

Succinct vs. Non-Succinct

Candidates can be generated directly without counting.



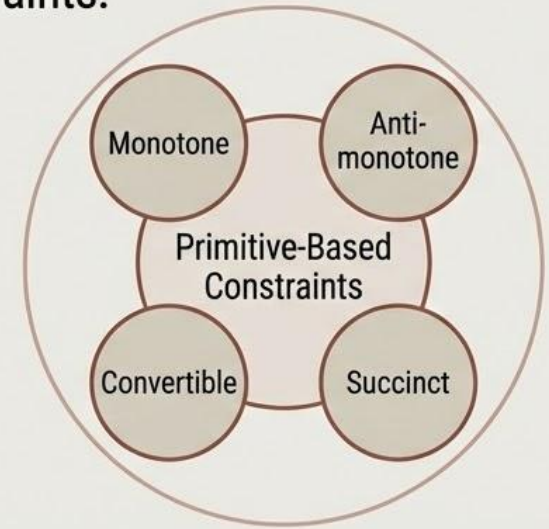
Requires support counting to verify (e.g., avg(price)).

Primitive & Convertible Constraints

Advanced Pruning Strategies

Primitive-Based Constraints

A superclass covering monotone, anti-monotone, convertible, and succinct constraints.



Convertible Constraints

Constraints that are NOT naturally monotonic/anti-monotonic but become so if items are ordered.

Example: avg(price) < $100.

Strategy: If we sort items by price in descending order, we can treat this as an anti-monotonic constraint and prune efficiently!

Part 3: Mining Sequential Patterns

Section 5.4

From Itemsets to Sequences

Focus: Discovering patterns in ordered data data(e.g., Time-Series, Clickstreams).

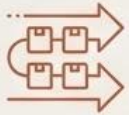
Concepts and Primitives

Section 5.4.1

Defining the Problem

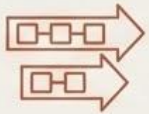
Finding frequent subsequences in a sequence database.

Basic Concepts



Sequence:
An ordered list of itemsets (e.g., \<(Laptop) (Printer)>).

Itemset:
A set of items occurring together at the same time (e.g., {Laptop, Mouse}).



Subsequence:
Sequence A is a subsequence of B if A's itemsets appear in the same relative order within B.

Example Database

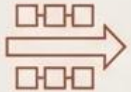
SID	Sequence
1	\<{a, b} {c} {d, e}\>
2	\<{a} {c} {d}\>
3	\<{b} {c, e}\>

Scalable Methods: Candidate Generation

Section 5.4.2: Approach 1

GSP (Generalized Sequential Patterns):

Based on the Apriori property.



Horizontal Format: Scans the database multiple times.

Process: Generate length-k candidates → Test against DB → Repeat.

SPADE:

Vertical Format: Uses ID-lists (intersection) instead of scanning.

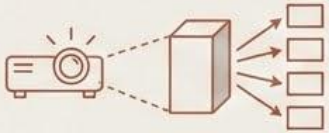
Faster than GSP but still relies on candidate generation.

Scalable Methods: Pattern-Growth

Section 5.4.2: Approach 2 (The Winner)

PrefixSpan (Prefix-Projected Sequential Pattern Mining):

The most efficient method.



Core Idea: Project the database into smaller “projected databases” based on frequent prefixes.

Algorithm:

1. {a} Find length-1 sequential patterns.
2. For each pattern, project the DB (keep only suffixes).
3. Mine each projected DB recursively.

PrefixSpan Example & Advantages

No Candidate Generation! ✔

Example Projection

Original DB:

<a(ab)c>,
<abac>,
<(ab)cb>

Projecting for prefix <a>:

Suffixes:
<(ab)c>,
<bac>,
<(ab)cb>

Advantages:

No Candidate Generation: Avoids the massive cost of generating/testing false candidates.

Search Space Reduction: Database projection shrinks the problem size at every step.

Performance: Typically 2-3 orders of magnitude faster than GSP for large databases.

Constraint-Based Mining of Sequences

Section 5.4.3: Focusing the Search

Why Constraints?

Sequence data is larger and more complex; we need to filter noise.

Constraint Types:

Length Constraints: Min/Max pattern length (e.g., 3-5 clicks).

Item Constraints: Must contain/not contain specific items (e.g., product_page).

Gap Constraints: Max time allowed between events (e.g., < 5 mins).

Duration Constraints: Max time for the entire sequence (e.g., < 30 mins).

Pruning Strategies

Optimization

Pushing Constraints:

Apply constraints deep inside the PrefixSpan projection loop.

Anti-monotonic Constraints:

Prune early (e.g., if a sequence is too long, stop projecting).

Convertible Constraints:

Use item ordering to enforce constraints efficiently.

Part 4: Mining Subgraph Patterns
Section 5.5

Mining Complex Structures

Discovering Frequent Subsructures in Graphs

Focus: Discovering Frequent Substructures in Graphs

Context: From sequential data to interconnected data.

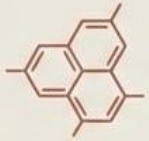
Why Mine Graphs?

Modeling Complexity

Beyond Itemsets: Graphs are more expressive than simple itemsets or sequences.

Goal: Find recurring substructures (e.g., a specific functional group in a chemical database).

Applications:



Chemical Compounds: Atoms (vertices) connected by bonds (edges).

Social Networks: People (vertices) connected by relationships (edges).

Bioinformatics: Protein structures and interactions.

Basic Definitions & The Problem

Section 5.5.1: The Language of Graphs

Definitions

Graph (G): (V, E) with vertices and edges.

Subgraph: G' is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$.

Graph Isomorphism: Two graphs are topologically identical (mapping vertices 1:1 preserves edges).

Problem Statement

Given a graph dataset $D = \{G_1, G_2, \ldots, G_n\}$. Find all subgraphs that appear in at least min_sup graphs.

Challenges

Subgraph Isomorphism is NP-Complete: Checking if graph A is inside graph B is computationally expensive.

Exponential Search Space: The number of possible subgraphs grows explosively.

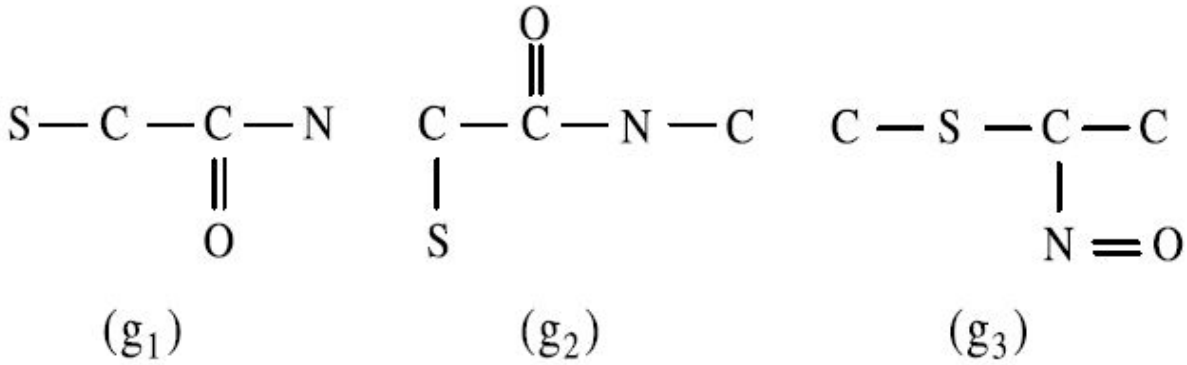


FIGURE 5.12

A sample graph data set.

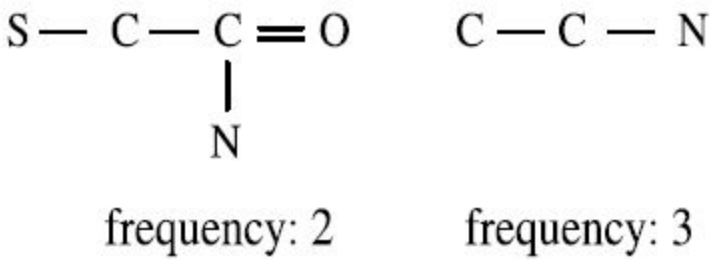


FIGURE 5.13

Frequent graphs.

Major Approaches: Apriori-Based AGM & FSG

Iterative candidate generation (Level-wise search).

Approaches

AGM (Inokuchi):
Vertex-based candidate generation.

FSG (Kuramochi & Karypis):
Edge-based candidate generation.

Process

Join

Size-k frequent subgraphs

Size-(k+1) candidates

Limitation

Requires Isomorphism Checking at every step and generates huge numbers of candidates.

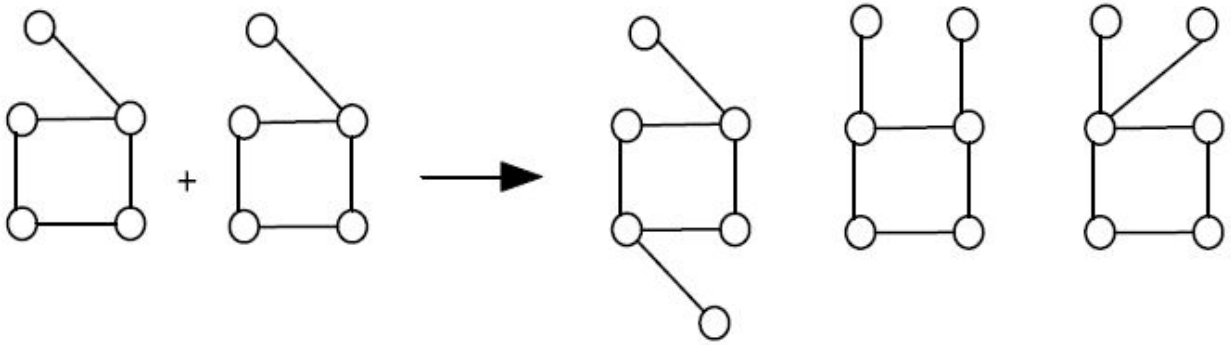


FIGURE 5.15

FSG: Two substructure patterns and their potential candidates.

The FSG algorithm adopts an *edge-based candidate generation* strategy that increases the substructure size by one edge in each call of AprioriGraph. Two size-k patterns are merged if and only if they share the same subgraph having $k-1$ edges, which is called the **core**. Here, *graph size* is taken to be the number of edges in the graph. The newly formed candidate includes the core and the additional two edges from the size-k patterns. Fig. 5.15 shows potential candidates formed by two structure patterns. Each candidate has one more edge than these two patterns, but this additional edge can be associated with different vertices. This example illustrates the complexity of joining two structures to form a large pattern candidate.

Major Approaches: Pattern-Growth

gSpan & Friends

Concept

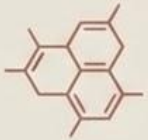
Grow patterns directly without generating false candidates.

Key Algorithms

gSpan (Yan & Han, 2002): The first Depth-First Search (DFS) algorithm.

CloseGraph: Mines closed frequent patterns (reduces redundancy).

SPIN: Mines maximal frequent subgraphs.



Gaston: highly efficient for specific graph types (molecular).

The gSpan Algorithm

Depth-First Search & Canonical Labels

Innovation

Uses a canonical representation called DFS-Code. Based on DFS traversal order. Uniquely represents a graph (no need to check isomorphism repeatedly).

Process

Represent graph via DFS-Code.

Perform Depth-First Search extension.

Prune using DFS lexicographic order.

Result

Systematic enumeration without candidate generation.

DFS-Code Example

Translating Structure to Text

Graph & Traversal

Graph: Vertices $\{v_0, v_1, v_2\}$ with edges $(v_0\text{-}v_1)$, $(v_0\text{-}v_2)$.

DFS Traversal: $v_0 \rightarrow v_1 \rightarrow v_2$.

DFS-Code Representation

DFS-Code Tuple: (from, to, from-label, edge-label, to-label)

```
(0, 1, A, a, B)
(1, 2, B, b, C)
```

This code uniquely identifies the structure, allowing efficient storage and comparison.

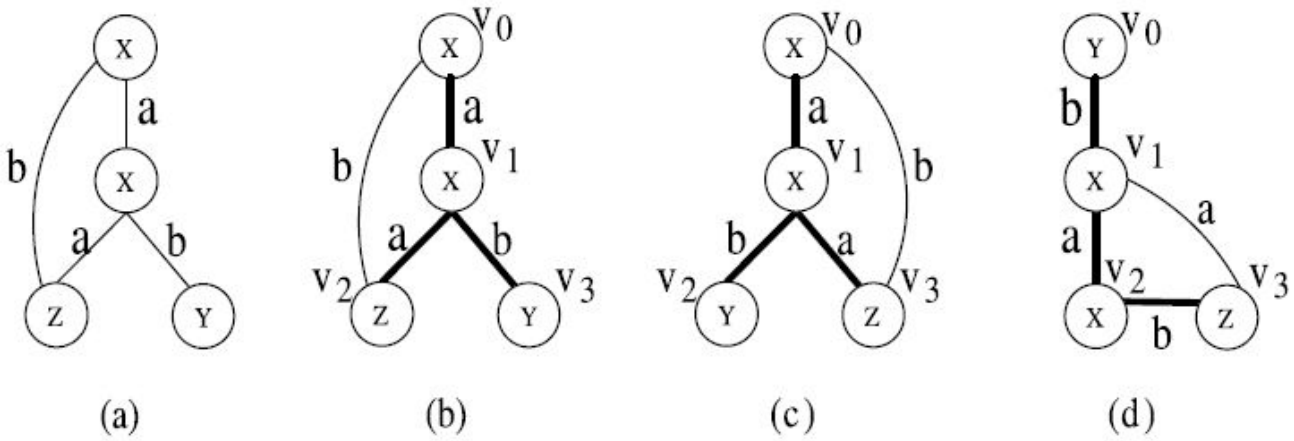


FIGURE 5.17

DFS subscripting.

performed, that is, the vertex visiting order. The darkened edges in Figs. 5.17(b) to 5.17(d) show three DFS trees for the same graph of Fig. 5.17(a). The vertex labels are x, y, and z; the edge labels are a and b. Alphabetic order is taken as the default order in the labels. When building a DFS tree, the visiting sequence of vertices forms a linear order. We use subscripts to record this order, where $i < j$ means v_i is visited before v_j when the depth-first search is performed. A graph G subscripted with a DFS tree T is written as G_T. T is called a **DFS subscripting** of G. Given a DFS tree T, we call the starting vertex in T, v_0, the *root*, and the last visited vertex, v_n, the *right-most vertex*. The straight path from v_0 to v_n is called the *right-most path*. In Figs. 5.17(b) to 5.17(d), three different subscriptings are generated based on the corresponding DFS trees. The right-most path is (v_0, v_1, v_3) in Figs. 5.17(b) and 5.17(c), and (v_0, v_1, v_2, v_3) in Fig. 5.17(d).

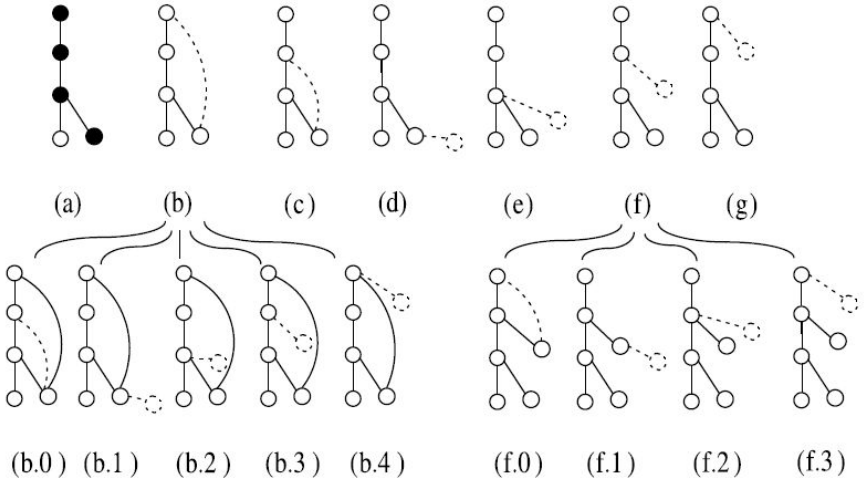


FIGURE 5.18

Right-most extension.

Table 5.6 DFS code for Fig. 5.17(b), 5.17(c), and 5.17(d).

edge	γ_0	γ_1	γ_2
e_0	$(0, 1, X, a, X)$	$(0, 1, X, a, X)$	$(0, 1, Y, b, X)$
e_1	$(1, 2, X, a, Z)$	$(1, 2, X, b, Y)$	$(1, 2, X, a, X)$
e_2	$(2, 0, Z, b, X)$	$(1, 3, X, a, Z)$	$(2, 3, X, b, Z)$
e_3	$(1, 3, X, b, Y)$	$(3, 0, Z, b, X)$	$(3, 1, Z, a, X)$

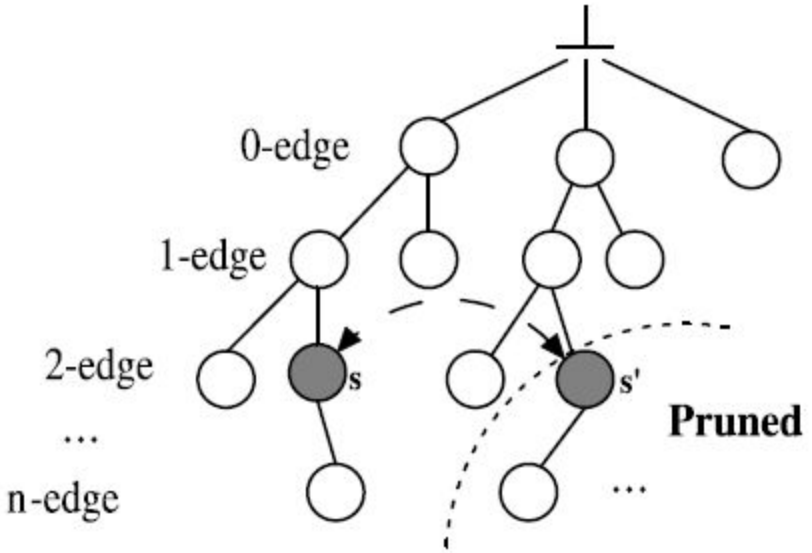


FIGURE 5.19

Lexicographic search tree.

Fig. 5.19 shows how to arrange all DFS codes in a search tree through right-most extensions. The root is an empty code. Each node is a DFS code encoding a graph. Each edge represents a right-most extension from a $(k-1)$-length DFS code to a k-length DFS code. The tree itself is ordered: left siblings are smaller than right siblings in the sense of DFS lexicographic order. Since any graph has at least one DFS code, the search tree can enumerate all possible subgraphs in a graph data set. However, one graph may have several DFS codes, minimum and nonminimum. The search of nonminimum DFS codes does not produce a useful result. *"Is it necessary to perform right-most extension on nonminimum DFS codes?"* The answer is "*no*." If codes s and s' in Fig. 5.19 encode the same graph, the search space under s' can be safely pruned.

Performance Comparison

Choosing the Right Tool

Algorithm	Approach	Best For
AGM	Apriori (Vertex)	Small graphs
FSG	Apriori (Edge)	Sparse graphs
gSpan	Pattern-Growth	General Purpose (Standard)
CloseGraph	Closed Patterns	Reducing Redundancy
Gaston	Pattern-Growth	Chemical Structures

State-of-the-Art: FCSG-Miner (2024)

Cutting Edge Research

Focus

Mines frequent closed subgraphs in transactional graph datasets.

Innovation

Introduces MMNI (Multiple Minimum Node Image) measure.

More informative than simple support or occurrence counts.

Holds the Downward Closure Property (essential for pruning).

Performance

Achieves up to 50% time reduction vs. traditional algorithms like cgSpan.

Applications

Analyzing Bike-Sharing Systems, Social Networks.

Slide 10: Real-World Application: Chemical Informatics – Discovery in Drugs

TASK: Discover **common substructures** in molecules.

INSIGHT: Frequent subgraphs often correspond to **Functional Groups**

Frequent Subgraph

(specific clusters of atoms responsible for chemical reactions)

Functional Group (e.g., Benzene Ring, Hydroxyl)

IMPACT: Helps in **Drug Discovery** by **identifying active structural motifs.**

Identifying active motifs for new medicines.

TOOL: Algorithms like gSpan are standard tools for finding these recurring patterns in massive chemical databases.

The Right Tool for the Job

Selecting Algorithms for Advanced Pattern Mining

Final synthesis of Constraint-Based, Sequential, and Graph Mining.

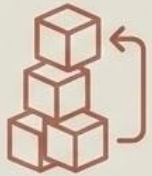
Algorithm Selection Guide (Constraint-Based)

Optimizing the Search

Anti-monotonic Constraints

Strategy:
Prune early in Apriori/FP-Growth.

Example:
min_sup, max(price) < $100.



Convertible Constraints

Strategy:
Order items appropriately to force anti-monotonicity.

Example:
avg(price) (Sort by price desc).

Succinct Constraints

Strategy:
Pre-compute the candidate space directly.

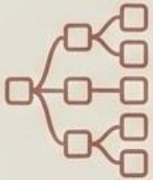
Example:
contains(item_X).

Algorithm Selection Guide (Sequential)

Handling Order

Small Datasets

GSP (Generalized Sequential Patterns): Works fine; conceptually simple (Apriori-based).



Large Datasets

PrefixSpan: The clear winner. Uses pattern-growth to avoid expensive candidate generation.

With Constraints

Push Constraints: Integrate constraints deep into PrefixSpan's projection phase for maximum pruning.

Algorithm Selection Guide (Graph)

Handling Structure

Small, Sparse Graphs

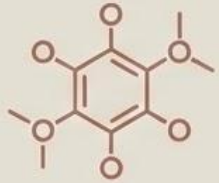
FSG: Acceptable (Edge-based Apriori).

General Purpose

gSpan: State-of-the-art for 20+ years. No candidate generation; uses DFS-Code.

Closed Patterns

CloseGraph or FCSG-Miner: Essential for reducing redundancy in dense results.



Large Molecular Datasets

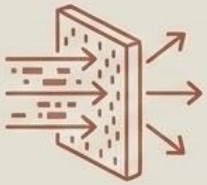
Gaston: Highly optimized for chemical structures.

Common Themes Across All Methods

The “Laws” of Advanced Mining

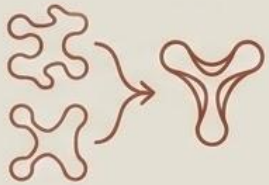
Pattern-Growth > Candidate Generation

Growing patterns is almost always more efficient than generating and testing candidates.



Constraints are Critical

They dramatically improve efficiency by pruning the search space early.



Canonical Forms are Essential

Whether it’s a sequence or a graph, unique representations (like DFS-Code) prevent duplicate work.

DFS > BFS

Depth-First Search (finding one long pattern quickly) is often better than Breadth-First Search for complex structures.

Real-World Applications

Mapping Theory to Practice

Pattern Type	Application	Recommended Algorithm
Sequential	Clickstream Analysis	PrefixSpan
Sequential	Purchase History	GSP (if small)
Graph	Chemical Compounds	gSpan / Gaston
Graph	Protein Structure	Gaston
Graph	Social Communities	FCSG-Miner
Constrained	Targeted Marketing	Various (with constraints)

Future Directions

Where the Field is Heading

Dynamic Graphs

Mining patterns in graphs that evolve over time (e.g., Social Network interactions).

Distributed Graph Mining

Scaling algorithms to run across clusters for massive datasets.

Closed Patterns

Focusing on concise representations to reduce “pattern explosion.”

High-Utility Patterns

Moving beyond “frequency” to “value” (e.g., High Profit > High Frequency).