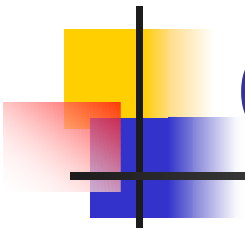
Frequent Subgraph Discovery
Mining Scientific & Relational Data Sets

Michihiro Kuramochi and George Karypis

Department of Computer Science & Engineering
University of Minnesota

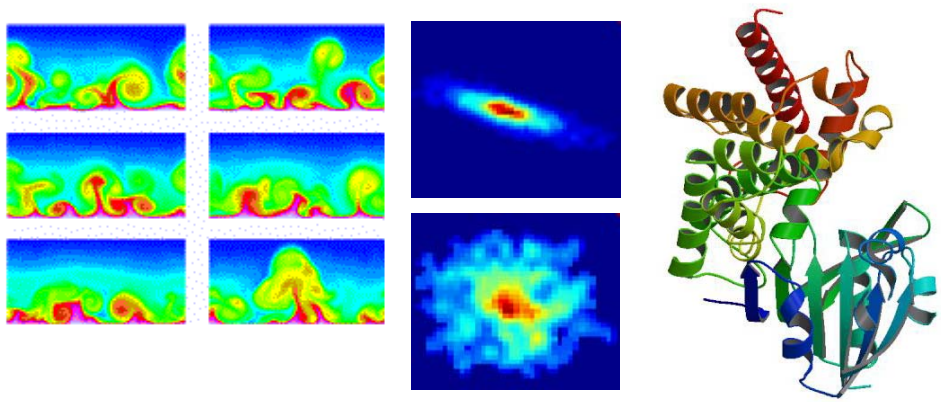


Outline

- Why do we need to use graphs?
- What is frequent subgraph discovery?
- Detail of the algorithm
- Empirical evaluation with synthetic and real datasets
- Conclusions

Data Mining And Scientific Data

- Data mining has emerged as a critical tool for knowledge discovery in large data sets.
  - It has been extensively used to analyze business, financial, and textual data sets.
- The success of these techniques has sparked interest in applying them to various scientific and engineering fields.
  - Astronomy
  - Biology
  - Ecosystem modeling
  - Fluid dynamics
  - Genomics
  - Structural mechanics



Pattern Discovery In Scientific Data

- Most of existing data mining algorithms are based on transaction representation, *i.e.*, **sets of items**.
- Scientific datasets with structures, layers, hierarchy and/or geometry often do not fit well in this transaction setting.
  - e.g., Numerical simulations, 3D protein structures, chemical compounds, etc.

Need algorithms that operate natively on datasets in their native/structured representation

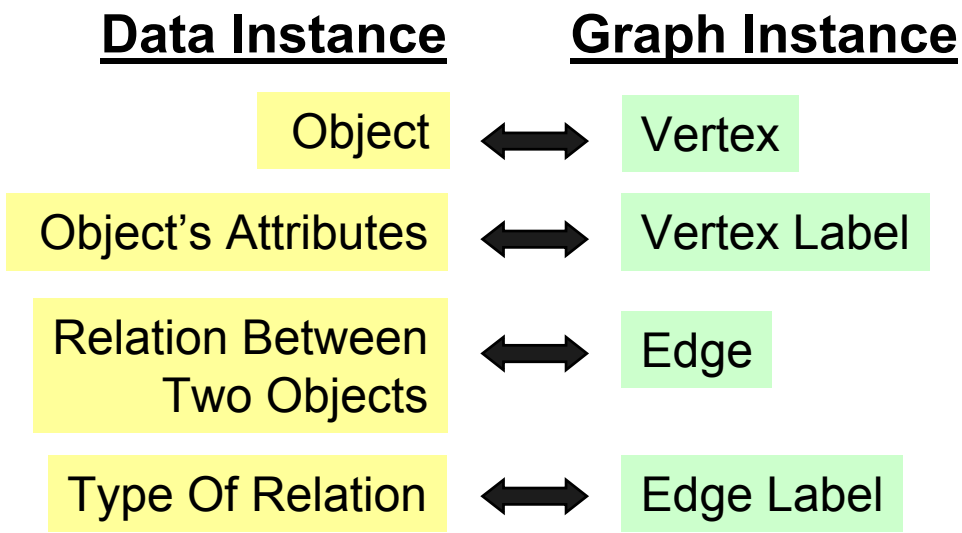
Representation For Scientific Datasets

- Which representation is good?
  - Abstract compared with the original raw (and typically flat) numerical data
  - Yet powerful enough to capture the important characteristics

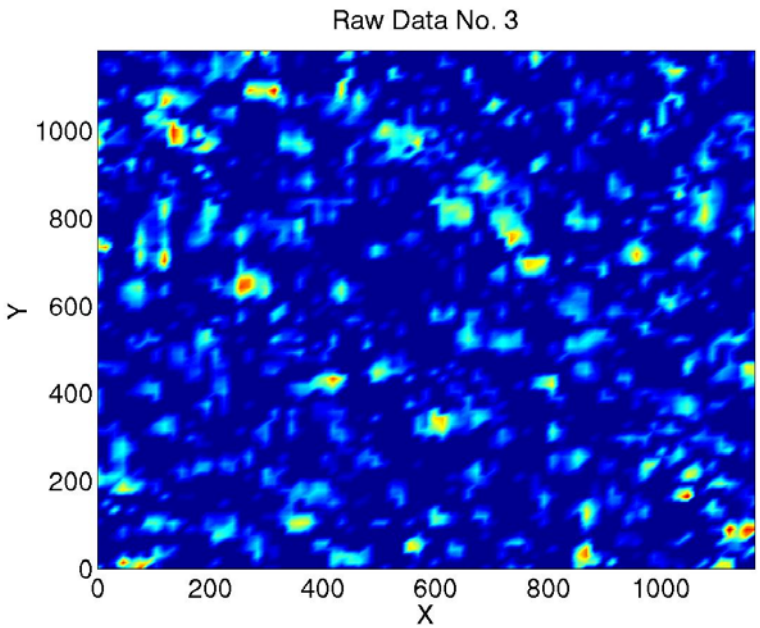
Labeled directed/undirected graphs

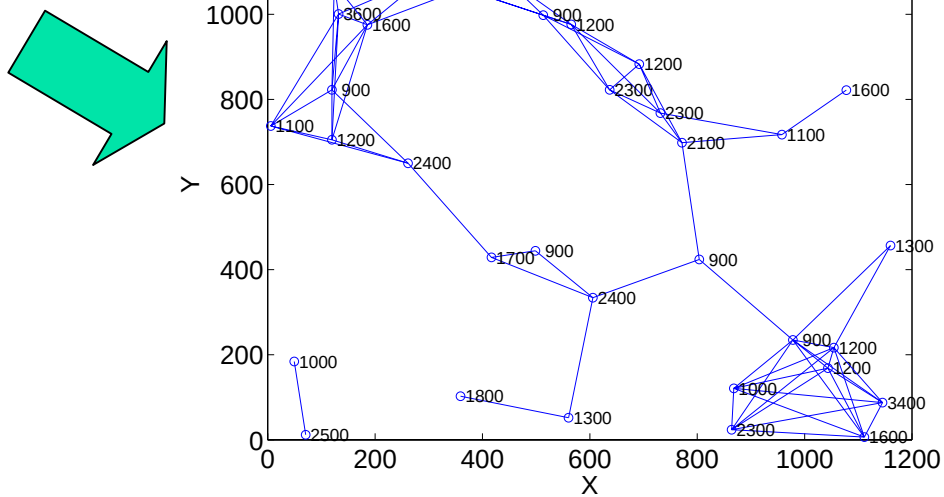
Modeling Data With Graphs…
Going Beyond Transactions

- Graphs can be used to accurately model and represent scientific data sets.
- Graphs are suitable for capturing arbitrary relations between the various objects.

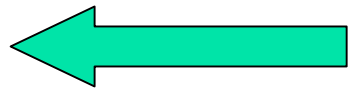


Example


Raw Data No. 3
Y
X
0
200
400
600
800
1000

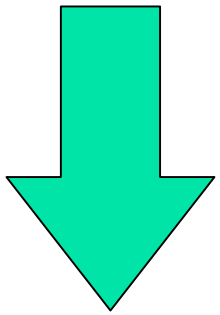

Corresponding Graph Transaction No. 3
Y
X
0
200
400
600
800
1000
1200
700
1100
2300
1100
3600
1600
900
1200
1200
900
2300
1600
2300
1100
1100
1200
2100
2400
1700
900
900
1300
2400
900
1200
1200
1000
1000
1800
1300
3400
2500
2300
1600

Graph Mining

- Finding patterns in these graph 
- Finding groups of similar graphs
- Building predictive models for the graphs

Interesting Patterns → Frequent Subgraphs

- Discovering interesting patterns



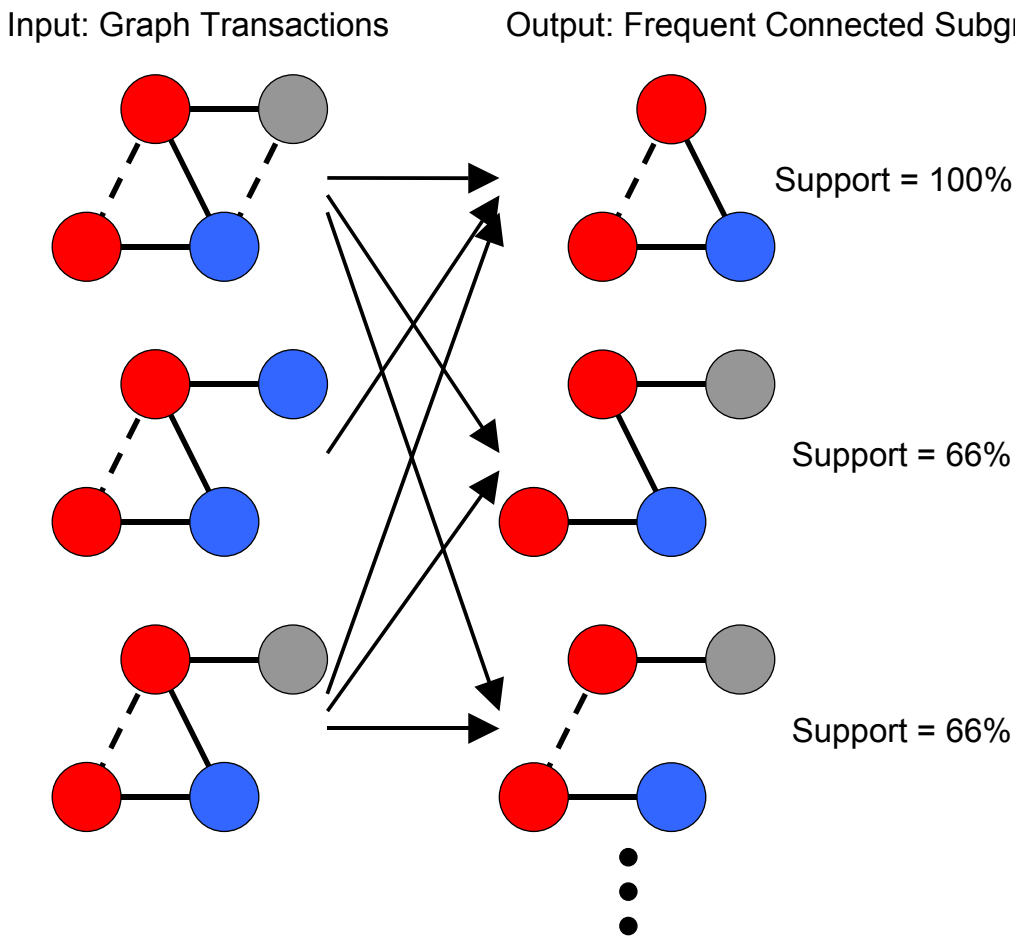
- Finding frequent, recurrent subgraphs

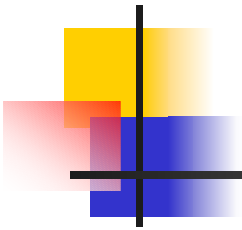
Efficient algorithms need to be developed that operate and take advantage of this representation

Finding Frequent Subgraphs: Input and Output

- Problem setting: similar to finding frequent itemsets for association rule discovery
- Input
  - Database of graph transactions
    - Undirected simple graph (no loops, no multiples edges)
    - Each graph transaction has labeled edges/vertices.
    - Transactions may not be connected
  - Minimum support threshold σ
- Output
  - Frequent subgraphs that satisfy the support threshold
  - Each frequent subgraph is connected.

Finding Frequent Subgraphs: Input and Output





FSG
Frequent Subgraph Discovery Algorithm

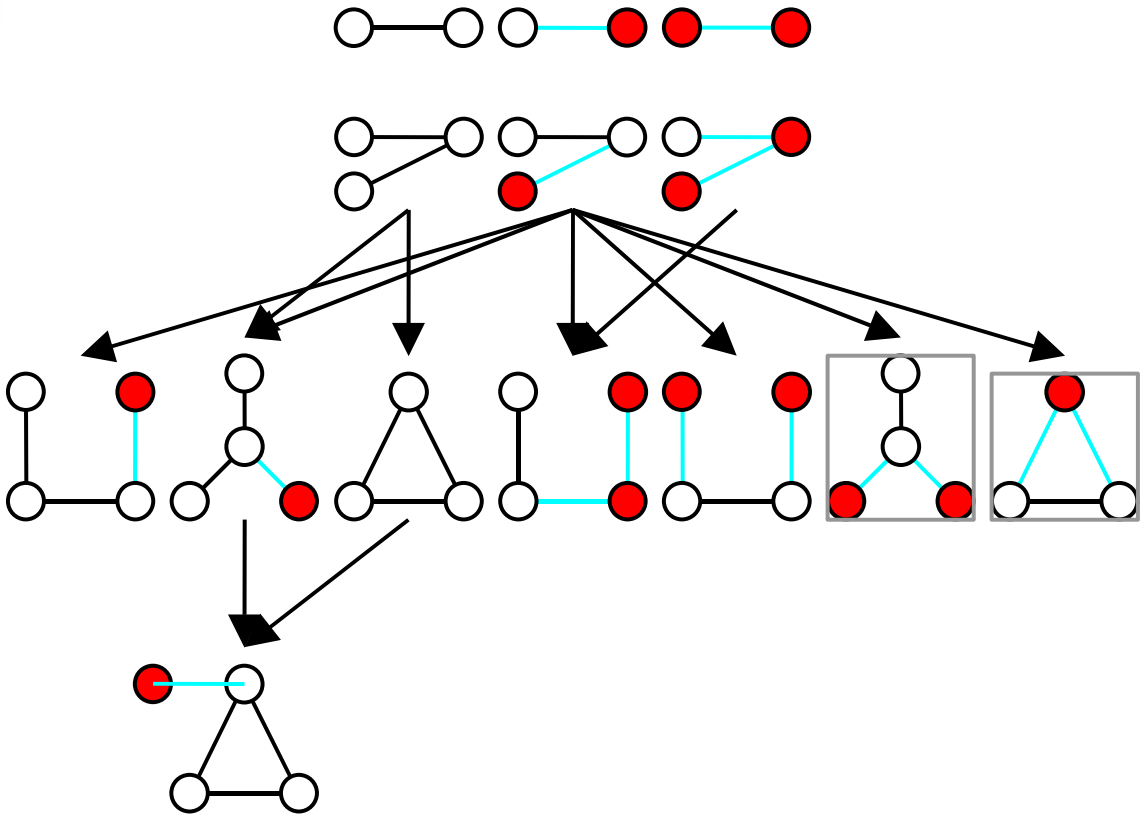
- Incremental and breadth-first fashion on the size of frequent subgraphs (like Apriori for frequent itemsets)
- Counting of frequent single and double edge subgraphs
- For finding frequent size k-subgraphs ($k \geq 3$),
  - Candidate generation

    Joining two size $(k-1)$-subgraphs similar to each other.
  - Candidate pruning by downward closure property
  - Frequency counting

    Check if a subgraph is contained in a transaction.
  - Repeat the steps for $k = k + 1$

    Increase the size of subgraphs by one edge.

FSG: Algorithm



Single edges

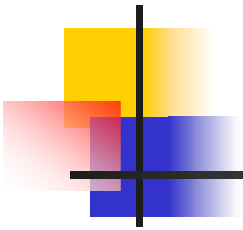
Double edges

3-candidates

3-frequent subgraphs

4-candidates

4-frequent subgraphs



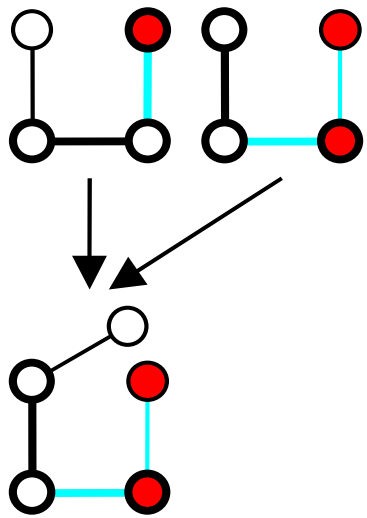
FSG: Methodology

- Edge-based
  - Transaction ↔ Labeled graph
  - Item ↔ Vertex
  - Itemset ↔ Connected subgraph
  - Size of an itemset ↔ Number of edges (size of a subgraph)
- FSG finds frequent connected subgraphs in the bottom-up and breadth-first fashion.

Trivial Operations Become Complicated With Graphs…

- Candidate generation
  - To determine two candidates for joining, we need to perform subgraph isomorphism.
  - Isomorphism for redundancy check
- Candidate pruning
  - To check downward closure property, we need subgraph isomorphism.
- Frequency counting
  - Subgraph isomorphism for checking containment of a frequent subgraph
- How to reduce the number of graph/subgraph isomorphism operations?

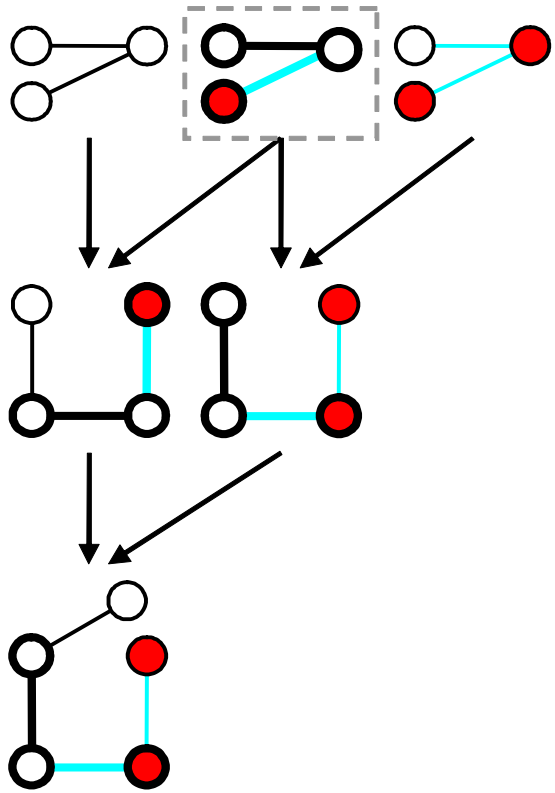
FSG Approach: Candidate Generation



3-frequent

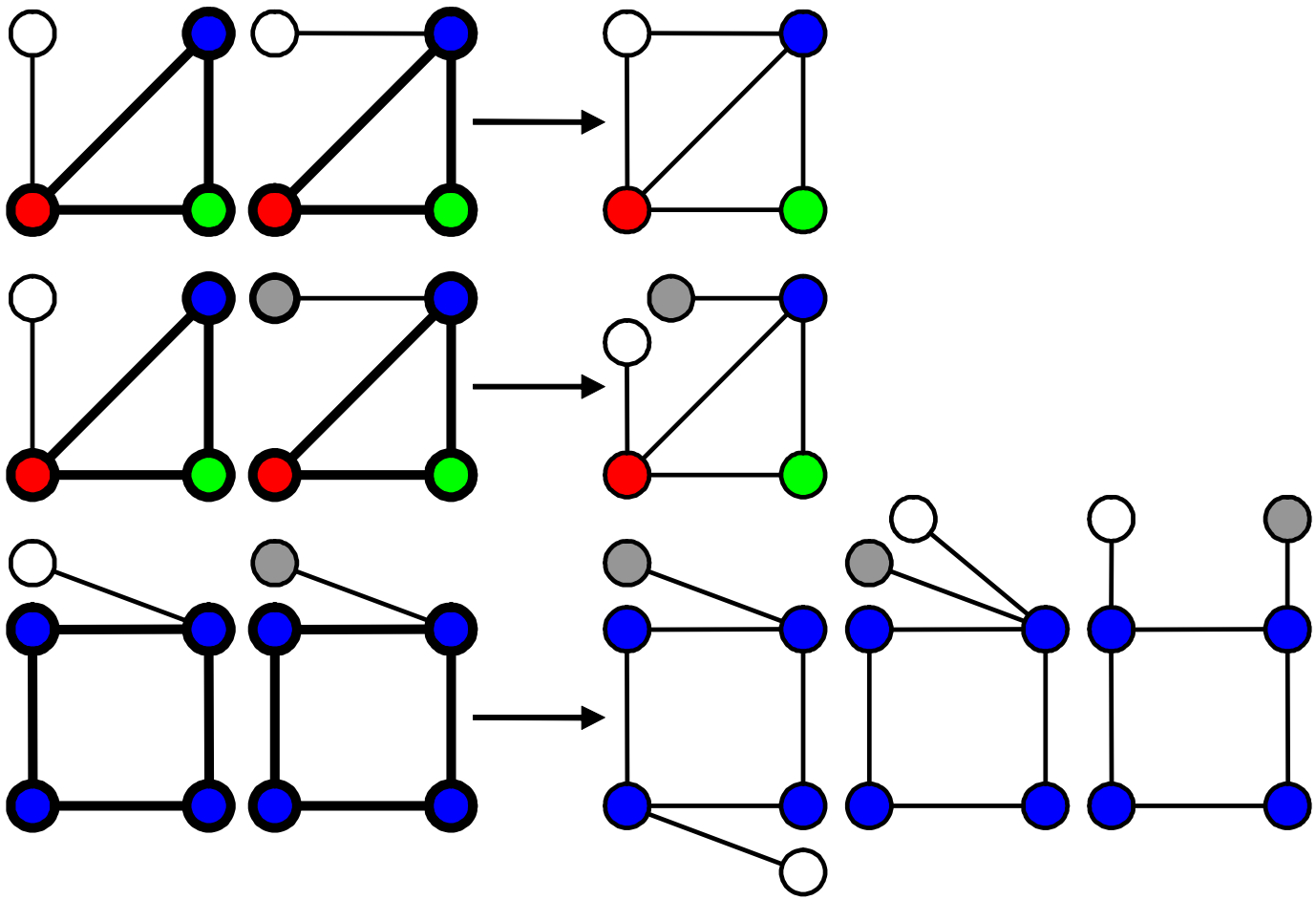
4-candidates

FSG Approach: Candidate Generation

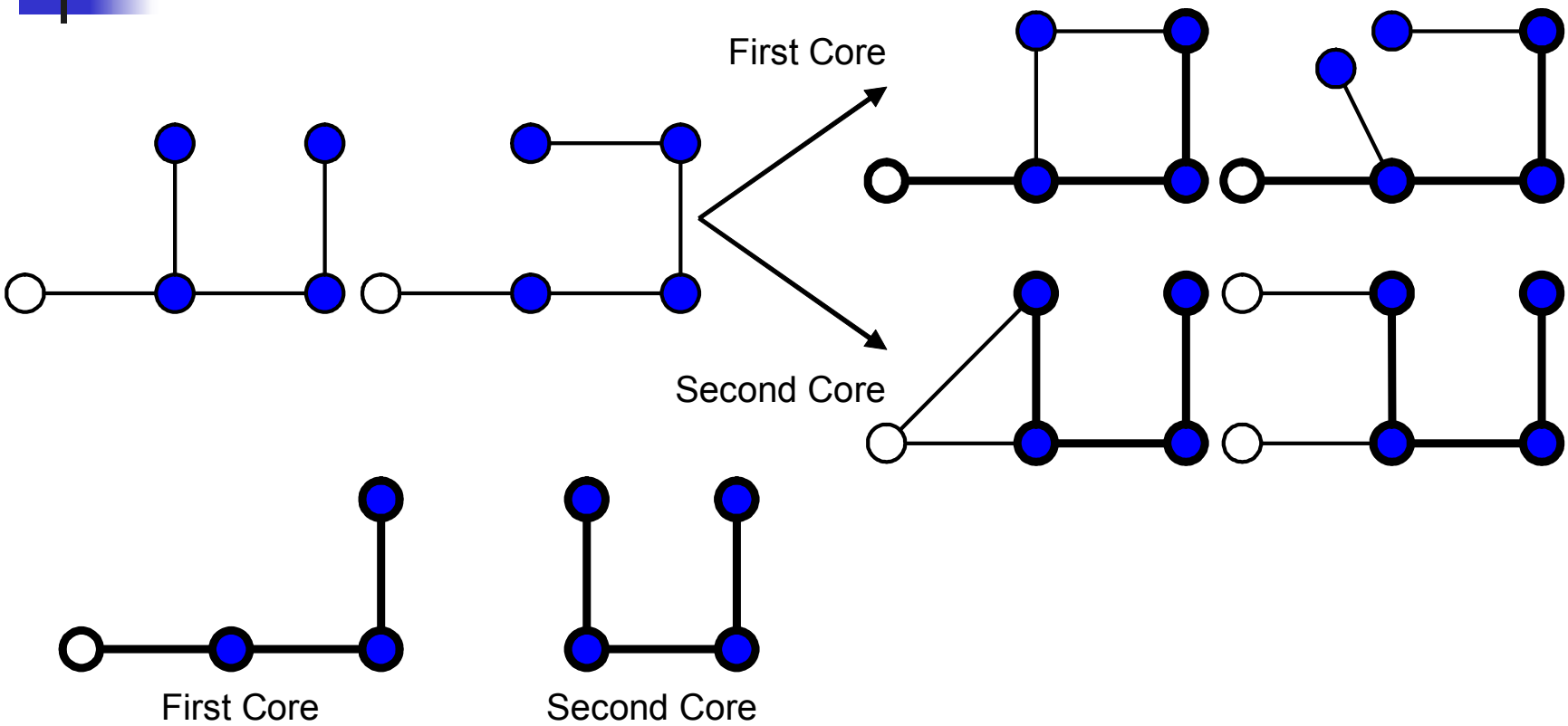


- Intersection of the parent lists of two 3-frequent subgraphs
- Without subgraph isomorphism, we can detect the core of the two 3-frequent subgraphs.
- Redundancy check by canonical labeling

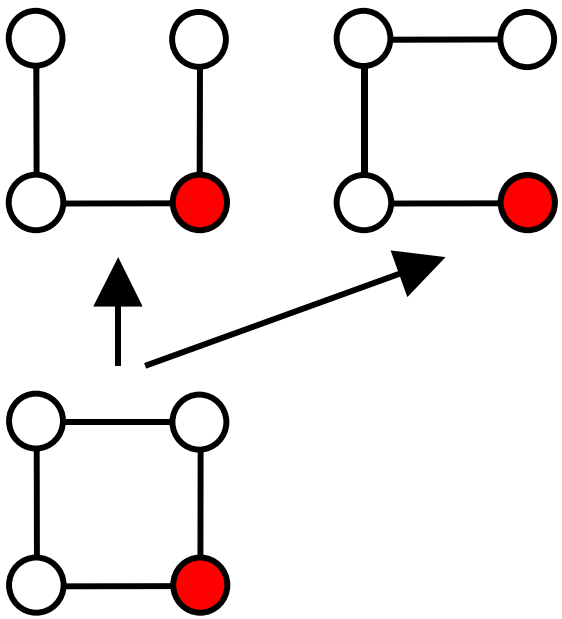
Candidate Generation Based On Core Detection



Candidate Generation Based On Core Detection



FSG Approach: Candidate Pruning



3-subgraphs

4-candidate

- Downward closure property
- Every ($k-1$)-subgraph must be frequent
- Keep the list of those ($k-1$)-subgraphs

FSG Approach: Candidate Pruning

- Pruning of size k-candidates
  - For all the $(k-1)$-subgraphs of a size k-candidate, check if downward closure property holds.
    - Canonical labeling is used to speedup the computation.
  - Build the parent list of $(k-1)$-frequent subgraphs for the k-candidate.
    - Used later in the candidate generation, if this candidate survives the frequency counting check.

FSG Approach: Frequency Counting

Transactions

T1 = {f1, f2, f3}

T2 = {f1}

T3 = {f2}

Frequent Subgraphs

TID(f1) = {T1, T2}

TID(f2) = {T1, T3}

Candidate

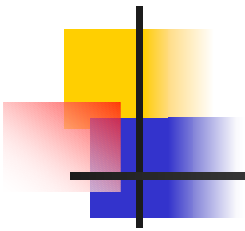
c = join(f1, f2)

TID(c) = subset(TID(f1) AND TID(f2))

- Perform only subg_isomorph(c, T1))
- TID lists require a lot of memory!

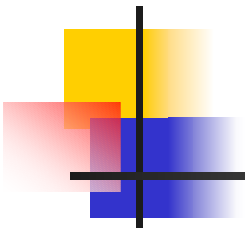
FSG Approach: Frequency Counting

- Frequency counting
  - Keep track of the TID lists.
  - If a size k-candidate is contained in a transaction, all the size $(k-1)$-parents must be contained in the same transaction.
  - Perform subgraph isomorphism only on the intersection of the TID lists of the parent frequent subgraphs of size $k-1$.
    - Significantly reduces the number of subgraph isomorphism checks.
    - Trade-off between running time and memory



Empirical Evaluation

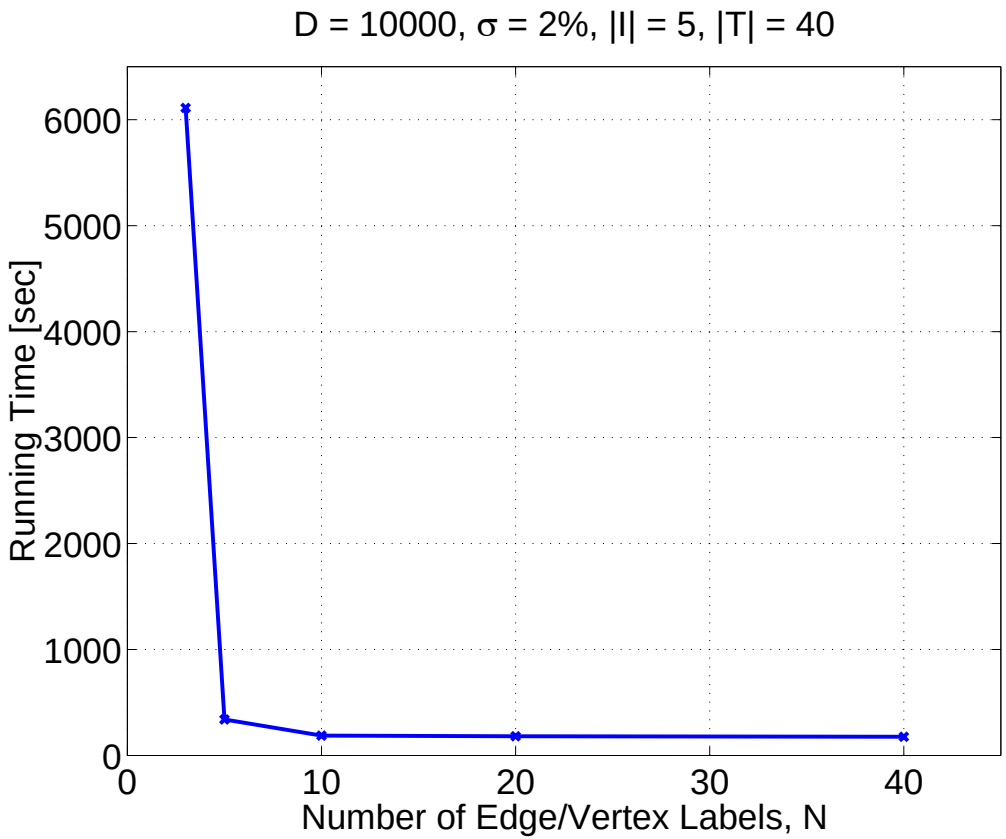
- Synthetic datasets
  - Sensitivity study
    - Number of labels used in input graphs
    - Transaction size
    - Number of transactions
- Real dataset
  - Chemical compounds from PTE challenge
- Pentium III 650 MHz, 2GB RAM



Evaluation: Synthetic Datasets

- Try to mimic the idea of the data generator for frequent itemset discovery, used in the Apriori paper (Agrawal and Srikant, *VLDB*, 1994).
- Generate a pool of potential frequent subgraphs ("seeds").
- Embed randomly selected seeds into each transaction until the transaction reaches the specified size.

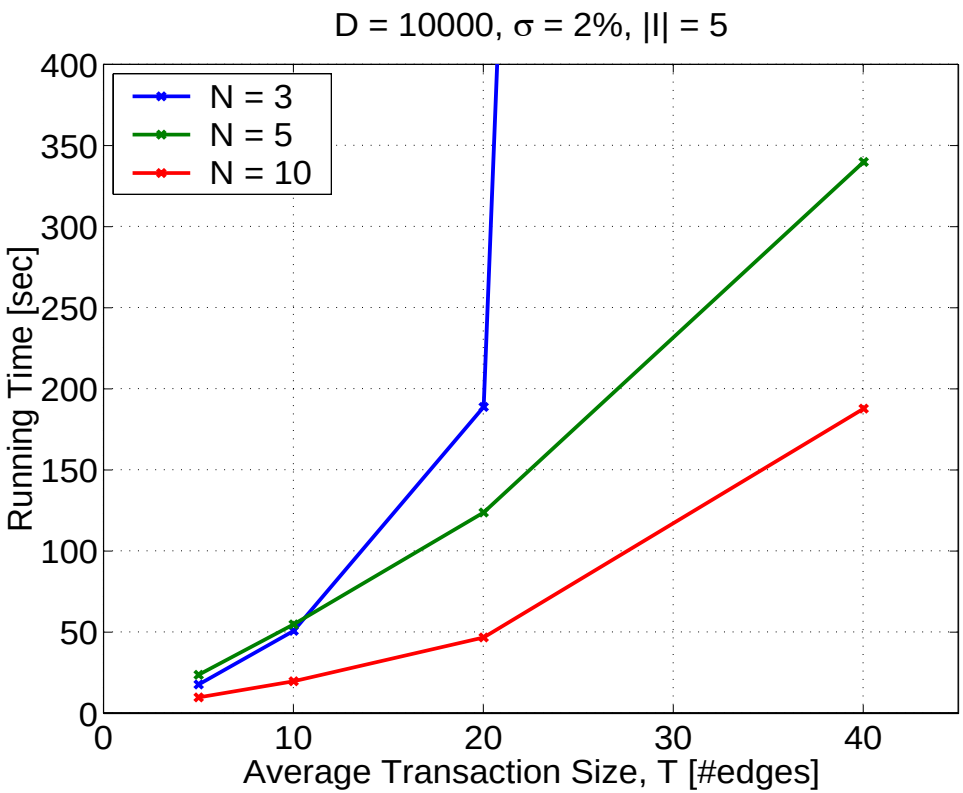
Sensitivity: The Number Of Labels



- 10000 transactions
- 2% support
- Average seed size |I| = 5
- Average transaction size |T| = 40

- More labels → Faster execution

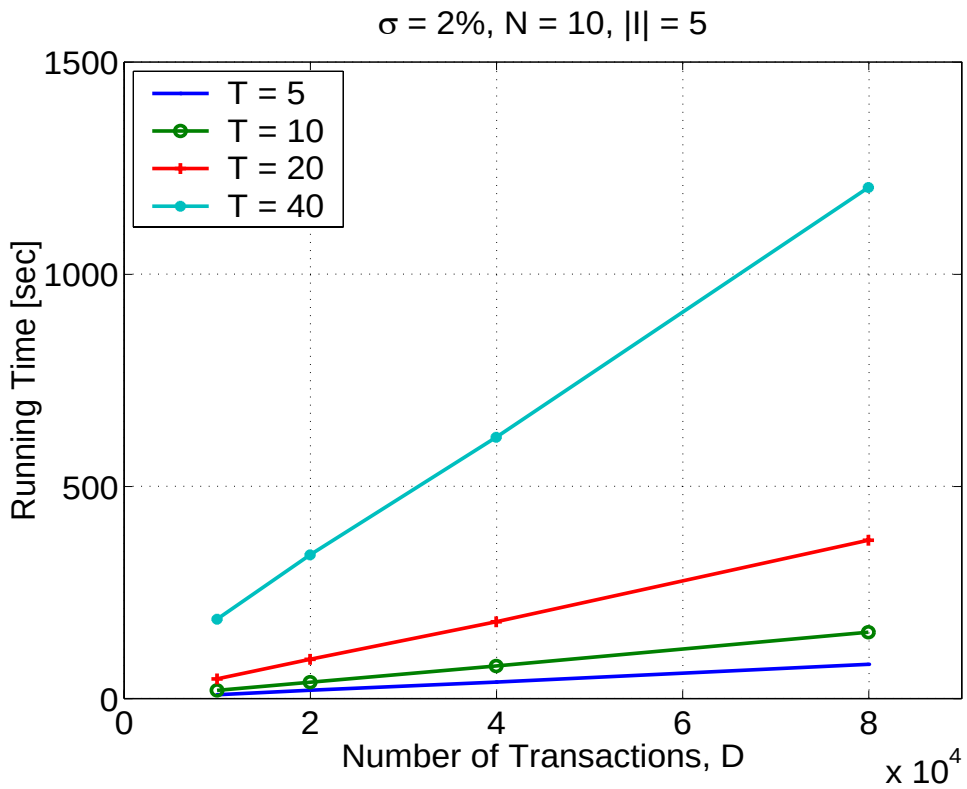
Sensitivity: Transaction Size



- 10000 transactions
- 2% support
- Average seed size
  |I| = 5

- Average transaction size, *T*, significantly affects the execution time, especially with fewer labels.

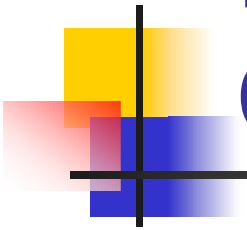
Sensitivity: Number of Transactions



- 2% support
- Transaction size
  T = 5, 10, 20, 40
- Average seed size
  |I| = 5
- 10 edge/vertex labels

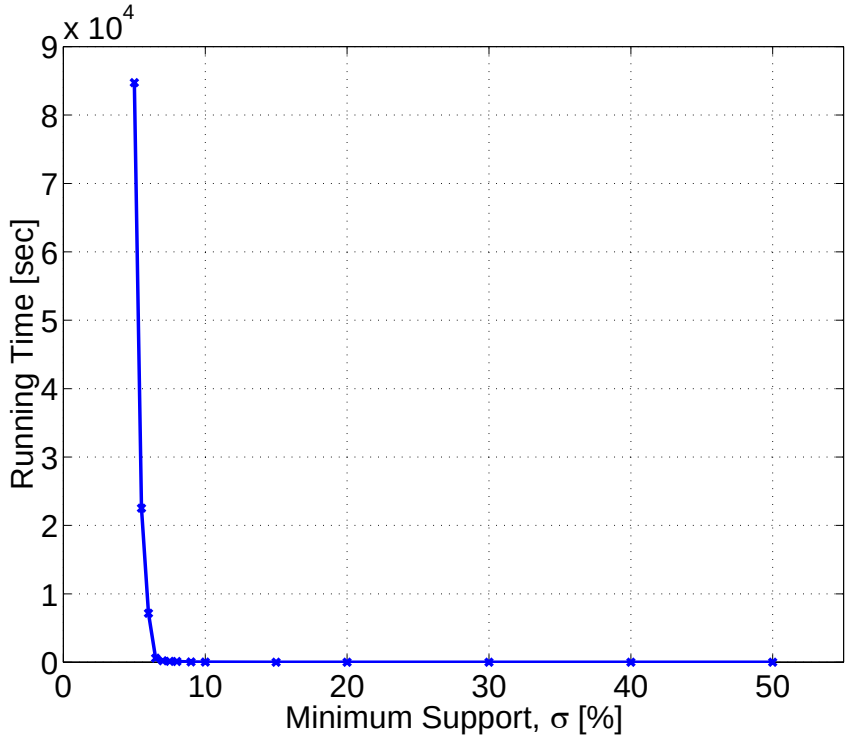
- Linear scalability

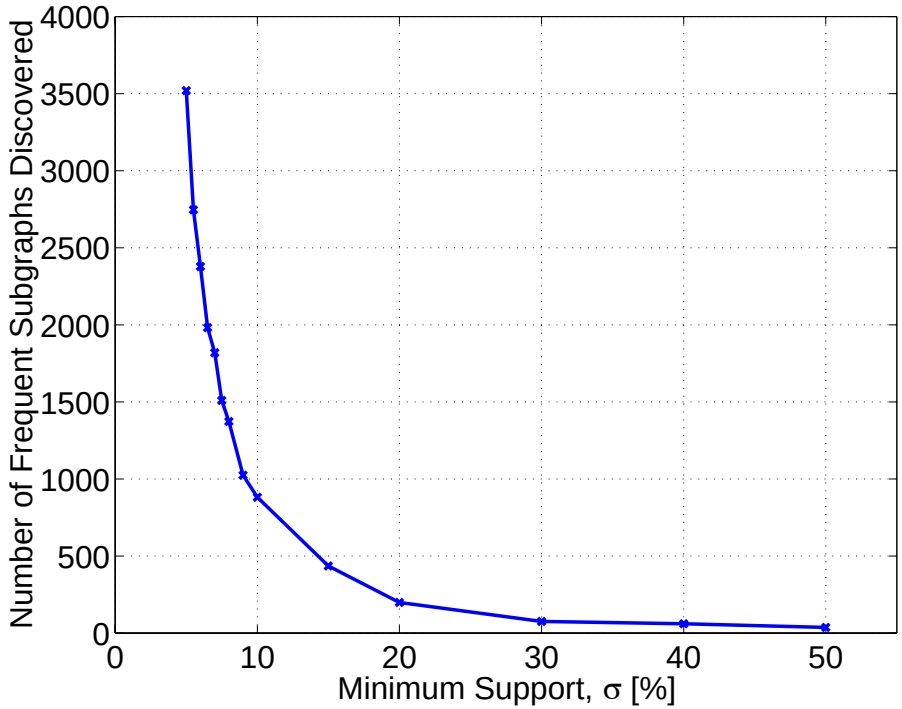
Evaluation: Chemical Compound Dataset



- Predictive Toxicology Evaluation (PTE) Challenge (Srinivasan et al., *IJCAI*, 1997)
- 340 chemical compounds
- Sparse
  - Average transaction size 27.4 edges, 27.0 vertices
  - Maximum transaction has 214 edges.
- 4 edge labels, 66 vertex labels

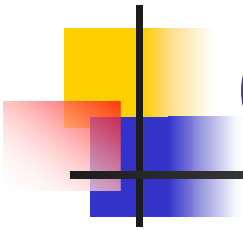
Evaluation: Chemical Compound Dataset





Evaluation: Chemical Compound Dataset

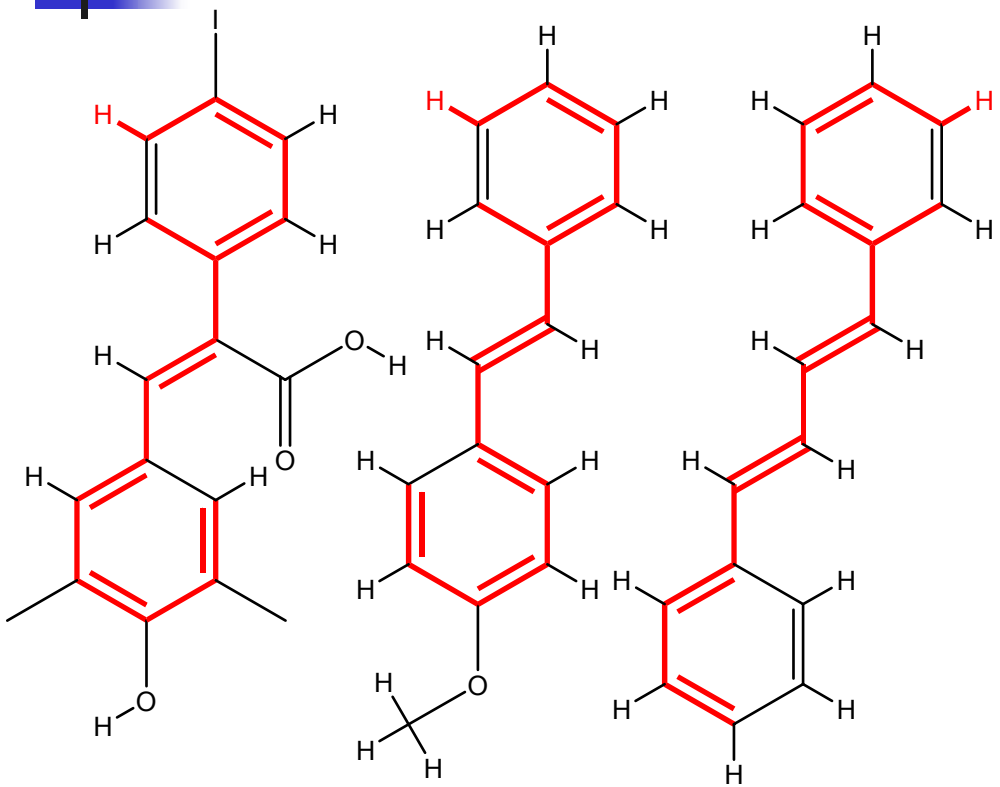
- AGM spent 8 days for support 10% on 400 MHz PC.
  - Inokuchi et al., An apriori-based algorithm for mining frequent substructures from graph data, *PKDD*, 2000
  - Vertex-based
  - AGM finds frequent induced subgraphs that may not be connected
- FSG took 28 seconds for support 10%.



Conclusions

- Linear scalability w. r. t. # of transactions
- FSG runs faster as the number of distinct edge/vertex labels increases.
- Average size of transactions |T|
  - Significant impact on the running time
  - Subgraph isomorphism for frequency counting
  - Edge density
    - Increases the search space of graph/subgraph isomorphism exponentially.
- Suitable for sparse graph transactions

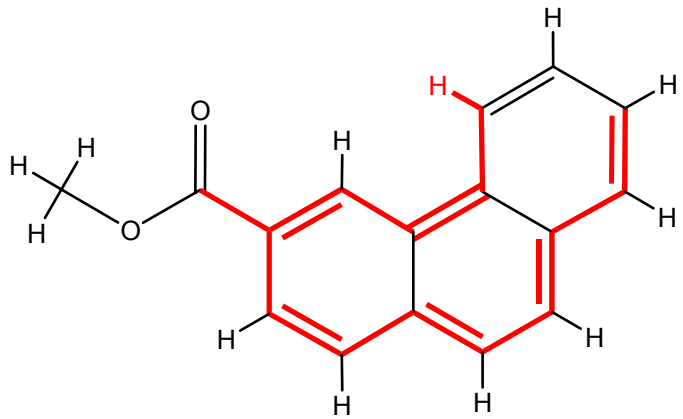
Topology Is Not Enough (Sometimes)



3-(3,5-dibromo-4-hydroxyphenyl)-2-(4-iodophenyl)acrylic acid

1-methoxy-4-(2-phenylvinyl)benzene

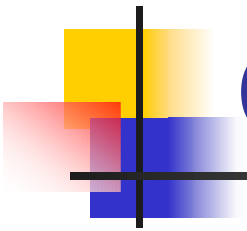
(4-phenyl-1,3-butadienyl)benzene



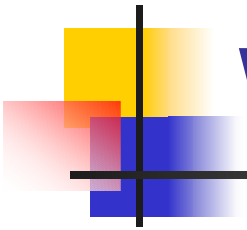
methyl 3-phenanthrenecarboxylate

- 100 chemical compounds with 30 atoms
- Support = 10%
- 3 patterns of 14 edges found

Extension To Discovering Geometric Patterns



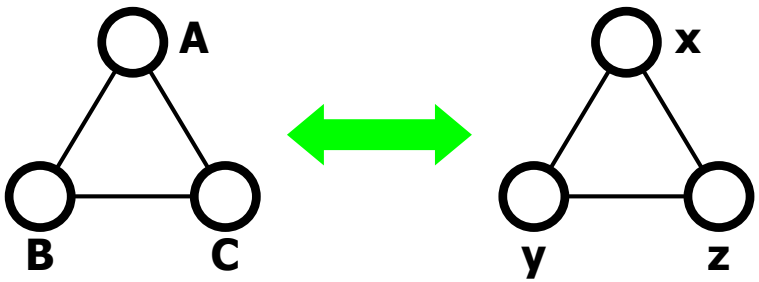
- Ongoing work, no preliminary results. Sorry!
- Geometric graphs
  - Most of scientific datasets naturally contain 2D/3D geometric information.
  - Each vertex has 2D/3D coordinates associated.
  - Geometric graphs are the same as the purely topological graphs except the coordinates (i.e., edges and vertices have labels assigned).



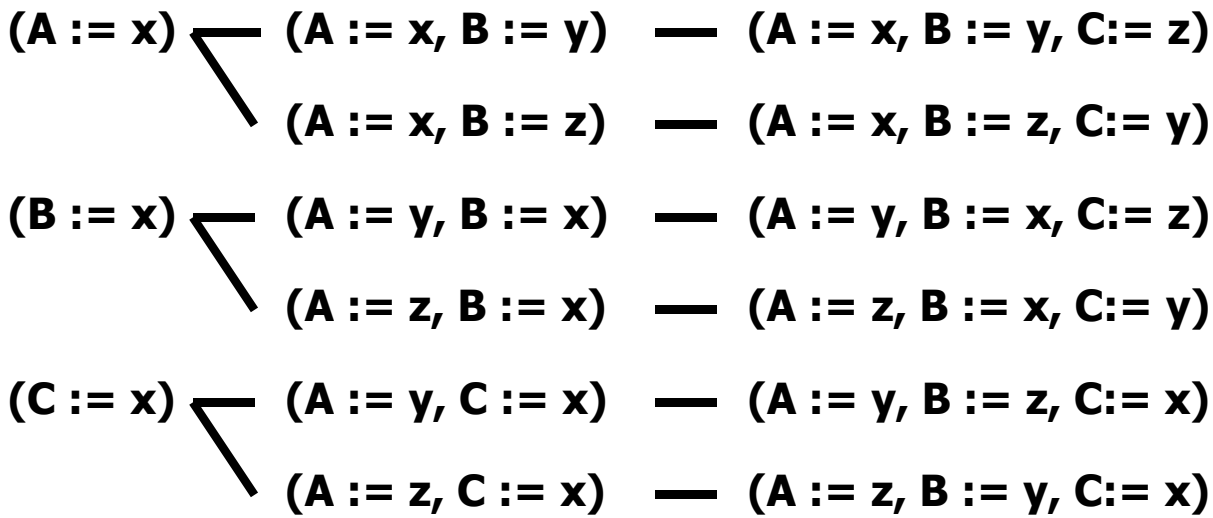
What Is Good With Geometry?

- Coordinates of vertices are helpful
- In topological graph finding, isomorphism which is known to be expensive operation, is inevitable.
  - Candidate generation
  - Frequency counting
- By using coordinates, we can narrow down the search space of (sub)graph isomorphism drastically.
  - Geometric hashing (pre-computing geometric configurations)
    - Rotation
    - Scaling
    - Translation

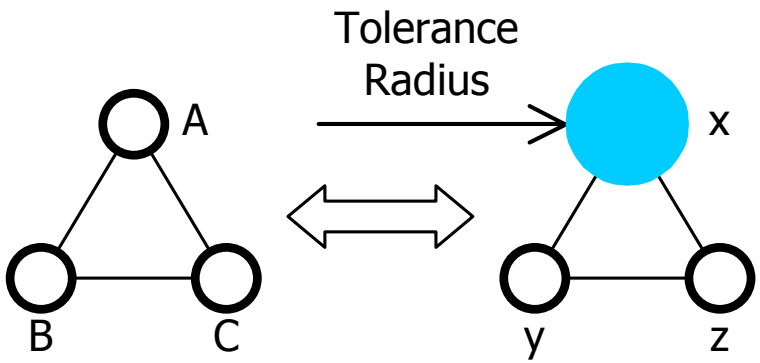
Search Space Of Purely Topological Isomorphism



- Complexity = $O(n!)$
  - n: # of vertices

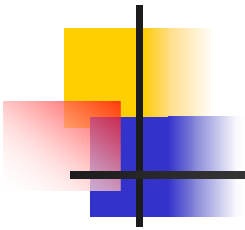


Use Of Coordinates In Geometric Isomorphism



- The radius of the shaded area determines the tolerance of the matching.

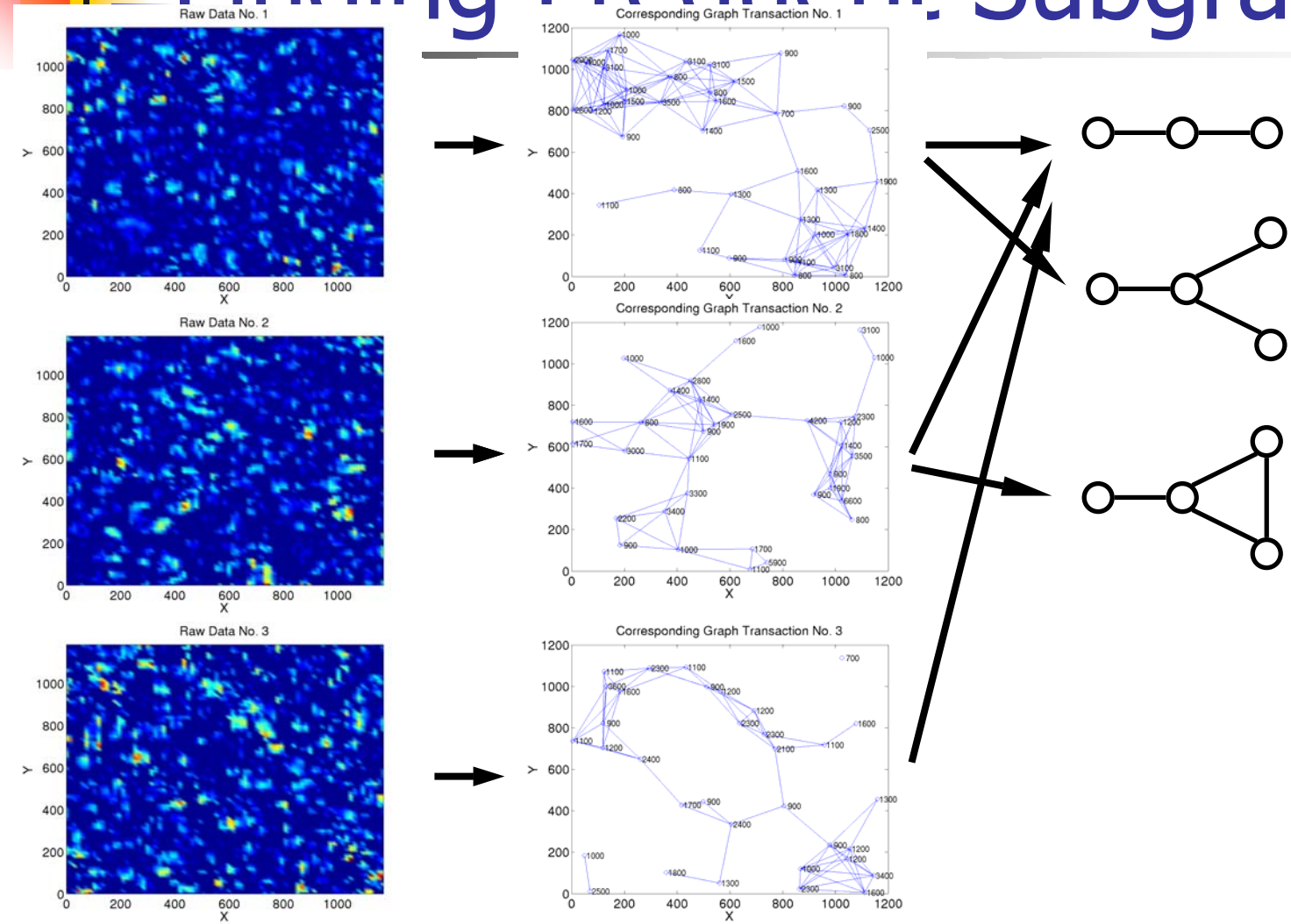
- To seek for a map of "A", we can focus the inside of the shaded area.
  - That eliminates "y" and "z" for a mapping of "A".
- Complexity depends on the # of configurations
  = $O(n^2)$, instead of $O(n!)$ for 2D
  n: # of vertices



Thank you!

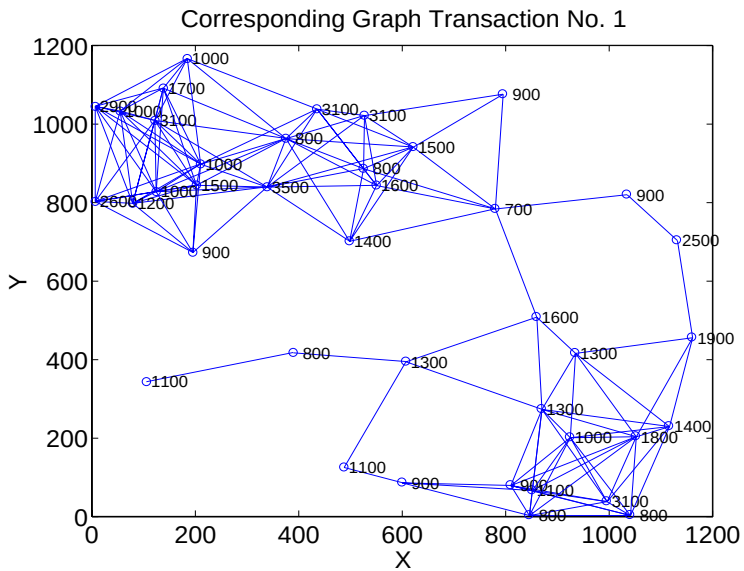
http://www.cs.umn.edu/~karypis

Finding Frequent Subgraphs

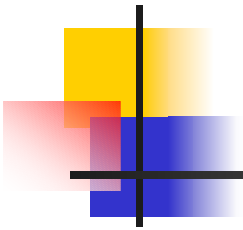


FSG: Methodology

- AGM (Inokuchi et al., An apriori-based algorithm for mining frequent substructures from graph data, *PKDD 2000*)
  - Vertex-based
    - Transaction ↔ Labeled graph
    - Item ↔ Vertex
    - Itemset ↔ Induced subgraph
    - Size of itemset ↔ Number of vertices (order of subgraph)
  - Discovered frequent subgraphs may not be connected


Corresponding Graph Transaction No. 1
X
Y
0
200
400
600
800
1000
1200
1000
1700
2900
1000
3100
3100
3100
900
800
1500
1000
800
1600
1500
1500
3500
2600
1200
1600
700
900
900
1400
2500
1600
1900
1300
800
1300
1100
1300
1400
1000
1800
1100
900
900
1100
3100
800
800

Graph/Subgraph Isomorphism: Hard Problems



- Graph isomorphism
  - Determine if two graphs are equivalent.
  - Suspected to be neither in P nor in NP-complete.
- Subgraph isomorphism
  - Determine if a graph is a part of another.
  - NP-complete
- Canonical labeling is equivalent to graph isomorphism.
- They are expensive operations, but doable especially when graphs are relatively small.
  - Frequent subgraphs are smaller than the input transactions.
- How to reduce the number of graph/subgraph isomorphism operations?

FSG Approach: Candidate Generation

- To generate a size k-candidate (k edges)
  - Take the intersection of the parent lists of two ($k-1$)-frequent subgraphs
    - To see if two ($k-1$)-frequent subgraphs share the same size ($k-2$)-parent.
    - Parent lists are obtained at the pruning phase.
    - Subgraph isomorphism free!
  - Example
    - parent(c^5) = { g^4, h^4, i^4}, parent(d^5) = { f^4, g^4, h^4}
    - Generate size 6-candidates from the cores g^4 and h^4.
  - Canonical labeling for redundancy check