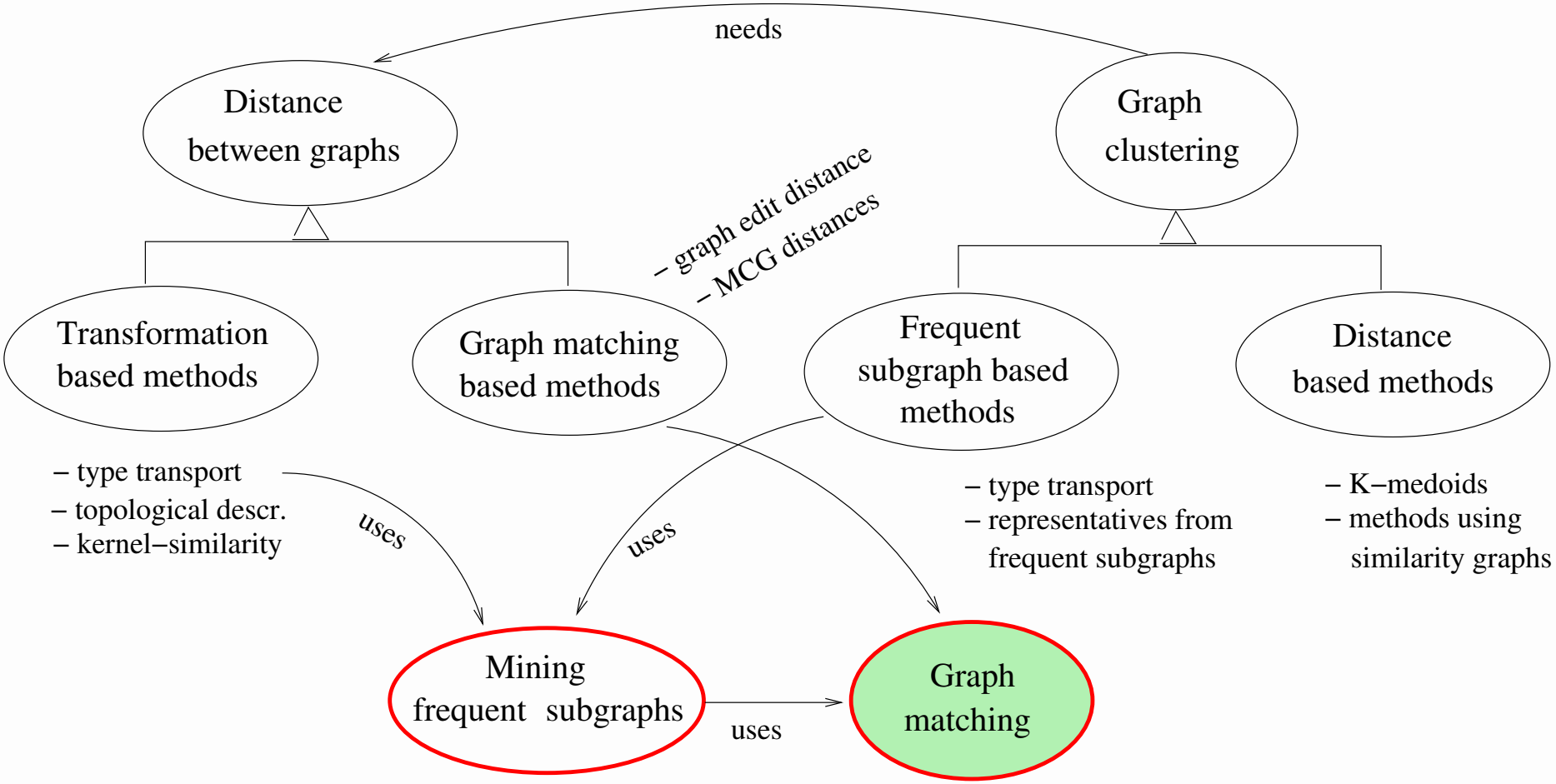
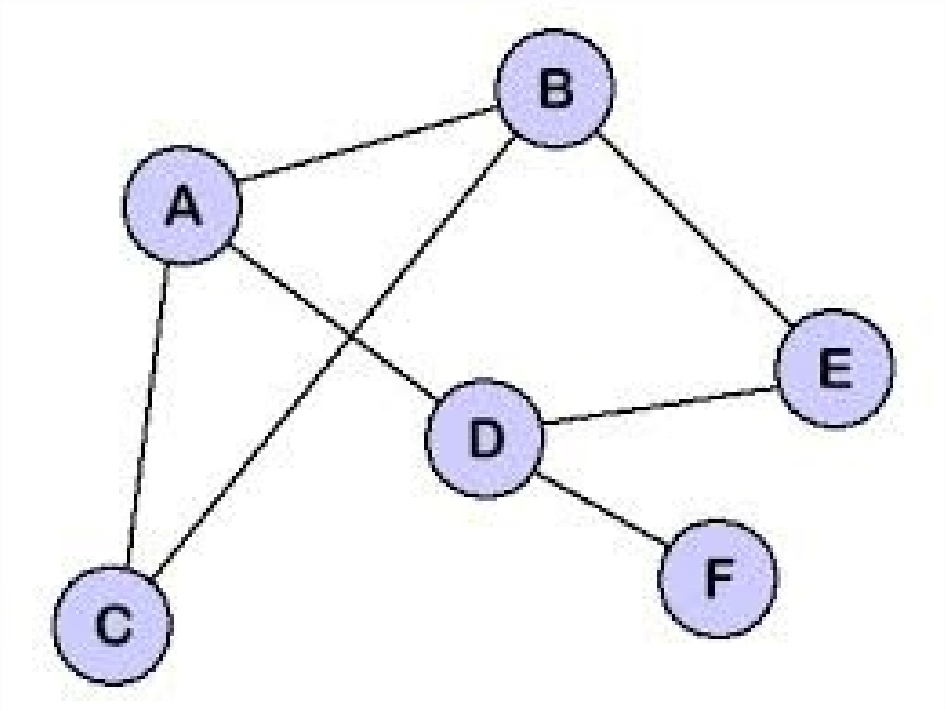
Mining database of multiple graphs



Graph notations

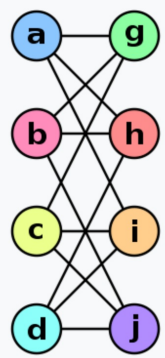
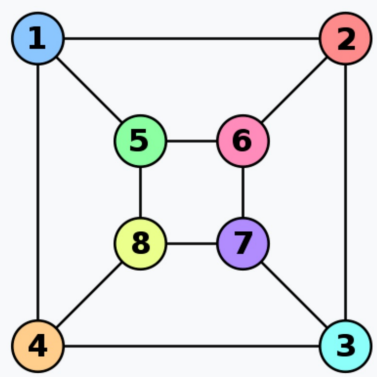


- $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ graph
- $\mathbf{V} = \{v_1, \ldots, v_n\}$ = set of vertices or nodes
- $|\mathbf{V}|$ = number of nodes
- node label $l(v_i)$
- $\mathbf{E} = \{e_1, \ldots, e_m\}$ = set of edges, $e_i = (v, u)$, $v, u \in V$
- $|\mathbf{E}|$ = number of edges

Now we assume that edges undirected and don't have labels

Graph isomorphism of unlabelled graphs

Two unlabelled graphs $\mathbf{G}_1 = (\mathbf{V}, \mathbf{E})$ and $\mathbf{G}_2 = (\mathbf{U}, \mathbf{F})$ are **isomorphic** or **matching** if there is an edge-preserving bijection $f : \mathbf{V} \to \mathbf{U}$ such that for any $v_1, v_2 \in \mathbf{V}$:
$(v_1, v_2) \in \mathbf{E} \Leftrightarrow (f(v_1), f(v_2)) \in \mathbf{F}$.

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Matching can be presented as $\mathcal{M} = \{(v, u) \mid v \in V, u \in U, u = f(v)\}$

Image source `https://en.wikipedia.org/wiki/Graph_isomorphism`

Graph isomorphism of labelled graphs

Two labelled graphs $\mathbf{G}_1 = (\mathbf{V}, \mathbf{E})$ and $\mathbf{G}_2 = (\mathbf{U}, \mathbf{F})$ are **isomorphic**, if there is an edge- and label-preserving bijection $f : \mathbf{V} \to \mathbf{U}$ such that

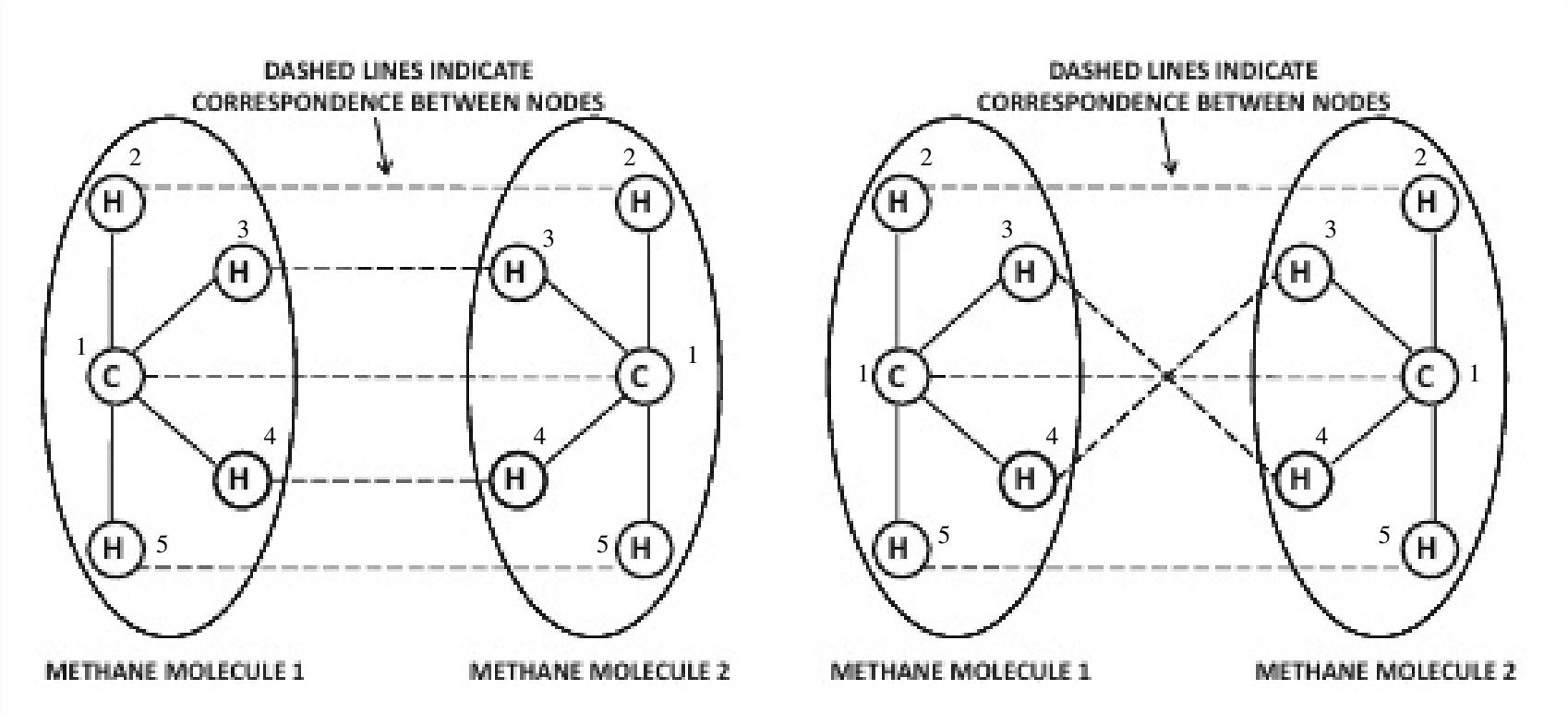
(i) Corresponding nodes have same labels: $\forall v \in \mathbf{V}$ and $f(v) \in \mathbf{U}$ $l(v) = l(f(v))$.

(ii) An edge between matched nodes exists in G_1 iff the corresponding edge exists in G_2: $\forall v_1, v_2 \in \mathbf{V}$: $(v_1, v_2) \in \mathbf{E} \Leftrightarrow (f(v_1), f(v_2)) \in \mathbf{F}$.

Note: no polynomial time algorithms are known (except special cases)

There can be many matchings!

Two matchings for molecules 1 and 2. Totally 4!=24 matchings!



$\mathcal{M} = \{(1,1),(2,2),(3,3),(4,4),(5,5)\}$ $\mathcal{M} = \{(1,1),(2,2),(3,4),(4,3),(5,5)\}$

Image source: Aggarwal Fig. 17.2

Subgraph isomorphism

Does a certain **query graph** $\mathbf{G}_q$ match a part of another graph $\mathbf{G}$?

Query graph $\mathbf{G}_q = (\mathbf{V}, \mathbf{E})$ is a **subgraph isomporphism** of $\mathbf{G} = (\mathbf{U}, \mathbf{F})$, if there is an injection $f : \mathbf{V} \to \mathbf{U}$ such that

(i) For all $v \in \mathbf{V}$ there is $f(v) \in \mathbf{U}$ such that $l(v) = l(f(v))$; and

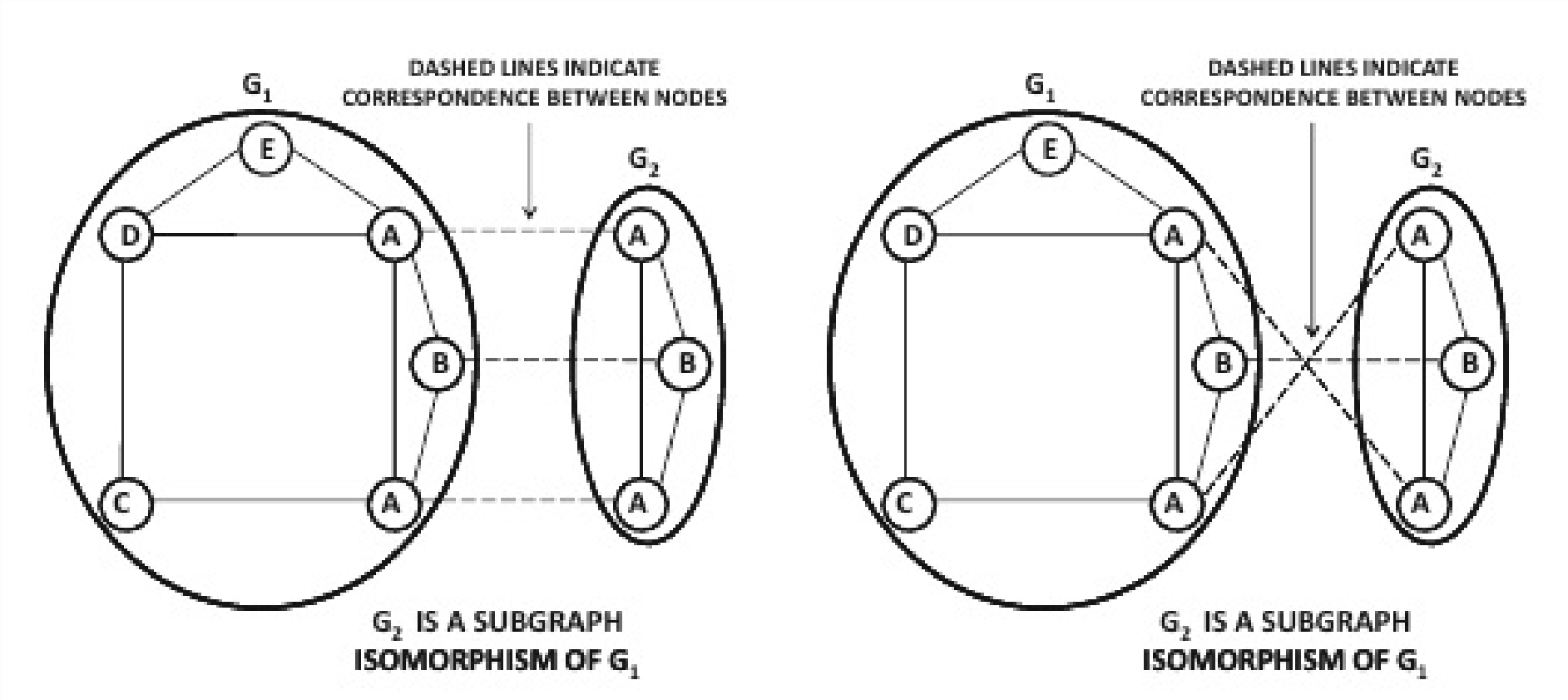
(ii) For any $v_1, v_2 \in \mathbf{V}$: $(v_1, v_2) \in \mathbf{E} \Leftrightarrow (f(v_1), f(v_2)) \in \mathbf{F}$.

Notes: 1) Usually it is required that the graphs are connected.

2) Sometimes a weaker condition suffices for (ii):

if $(v_1, v_2) \in \mathbf{E} \Rightarrow (f(v_1), f(v_2)) \in \mathbf{F}$

Subgraph isomorphism: example



- Algorithm: see Aggarwal Ch 17.2.1

Image source: Aggarwal Fig. 17.3

Maximum common subgraph (MCG)

Problem: Given $\mathbf{G}_1$ and $\mathbf{G}_2$, find $\mathbf{G}_0 = (\mathbf{V}_0, \mathbf{E}_0)$ such that

(i) $\mathbf{G}_0$ is a subgraph isomorphism of both $\mathbf{G}_1$ and $\mathbf{G}_2$ and

(ii) $|\mathbf{V}_0|$ is as large as possible.

+ useful for comparing graphs
  - distances between graphs
  - frequent subgraph discovery

− NP-hard problem (like subgraph isomorphism)

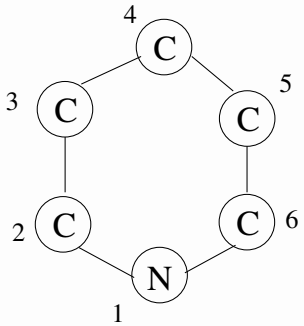
Algorithm for $MCG(G_1, G_2)$

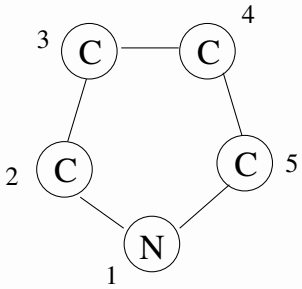
function $MCG(\mathbf{G}_1, \mathbf{G}_2, \mathcal{M}, \mathcal{M}_{best})$
/* *Create candidates for matching node pairs* */
$\mathcal{C} = \{(v, u) \mid v \in \mathbf{V}, u \in \mathbf{U}, l(v) = l(u), (v, u) \notin \mathcal{M}\}$
Prune $\mathcal{C}$
/* *Recursion:* */
for all $(v, u) \in \mathcal{C}$
  if valid$(\mathcal{M}, (v, u))$ // *is* (u, v) *a valid extension?*
    $\mathcal{M}_{best} = MCG(\mathbf{G}1, \mathbf{G}2, \mathcal{M} \cup (v, u), \mathcal{M}_{best})$
if $(|\mathcal{M}| > |\mathcal{M}_{best}|)$
  return $\mathcal{M}$
else return $\mathcal{M}_{best}$

$\mathbf{G}_1 = (\mathbf{V}, \mathbf{E})$, $\mathbf{G}_2 = (\mathbf{U}, \mathbf{F})$
Call: $MCG(\mathbf{G}_1, \mathbf{G}_2, \emptyset, \emptyset)$

Algorithm: valid extensions

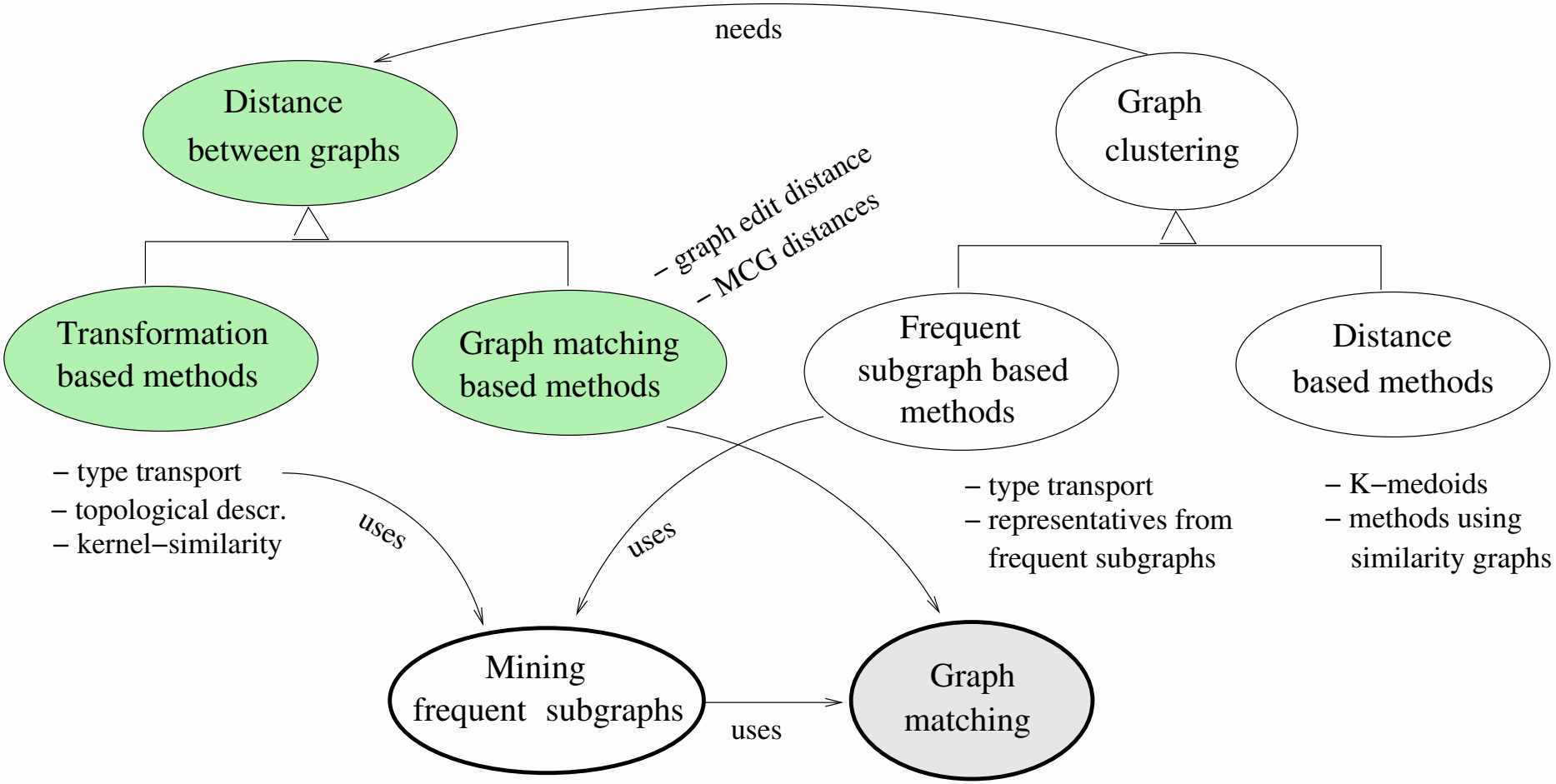
valid$(\mathcal{M}, (v, u))$
if $\big(\exists u_2 \in \mathbf{U} : ((u, u_2) \in F)\&\&((v_2, u_2) \in \mathcal{M})\&\&((v, v_2) \notin E)\big)$ **or**
$\big(\exists v_2 \in \mathbf{V} : ((v, v_2) \in E)\&\&((v_2, u_2) \in \mathcal{M})\&\&((u, u_2) \notin F)\big)$
return 0
else return 1





E.g., $\mathcal{M} = \{(1,1),(2,2),(3,3),(6,5)\}$.
$(4,4)$ is invalid extension – why?
Is there any valid extension?

Next to distances



Distances based on maximum common subgraphs

- Let's assume graph size = number of nodes, i.e., for $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ notate $|\mathbf{G}| = |\mathbf{V}|$
- Let $MCS(\mathbf{G}_1, \mathbf{G}_2)$=maximum common subgraph of $\mathbf{G}_1$ and $\mathbf{G}_2$ and $|MCS(\mathbf{G}_1, \mathbf{G}_2)|$=its size

1. **Unnormalized non-matching measure**:

$$U(\mathbf{G}_1, \mathbf{G}_2) = |\mathbf{G}_1| + |\mathbf{G}_2| - 2 \cdot |MCS(\mathbf{G}_1, \mathbf{G}_2)|$$

- = number on non-matching nodes
- Problem: what if graphs have very different sizes?

Normalized MCS distances

2. **Union-normalized distance** $Udist \in [0, 1]$

$$Udist(\mathbf{G}_1, \mathbf{G}_2) = 1 - \frac{|MCS(\mathbf{G}_1, \mathbf{G}_2)|}{|\mathbf{G}_1| + |\mathbf{G}_2| - |MCS(\mathbf{G}_1, \mathbf{G}_2)|}$$

= number of non-matching nodes normalized by union size

3. **Max-normalized distance** $Mdist \in [0, 1]$

$$Mdist(\mathbf{G}_1, \mathbf{G}_2) = 1 - \frac{|MCS(\mathbf{G}_1, \mathbf{G}_2)|}{\max\{|\mathbf{G}_1|, |\mathbf{G}_2|\}}$$

- metric

MCS distances can be computed efficiently **only for small graphs!**

Graph edit distance

What is the minimum cost of edit operations to transform $\mathbf{G}_1$ to $\mathbf{G}_2$?

(i) node insertion

(ii) node deletion (deletes also incident edges)

(iii) edge insertion

(iv) edge deletion

(v) label substitution of nodes

- application-specific costs
- may be exponentially many possible edit paths!
- NP-hard

Graph edit distance: example

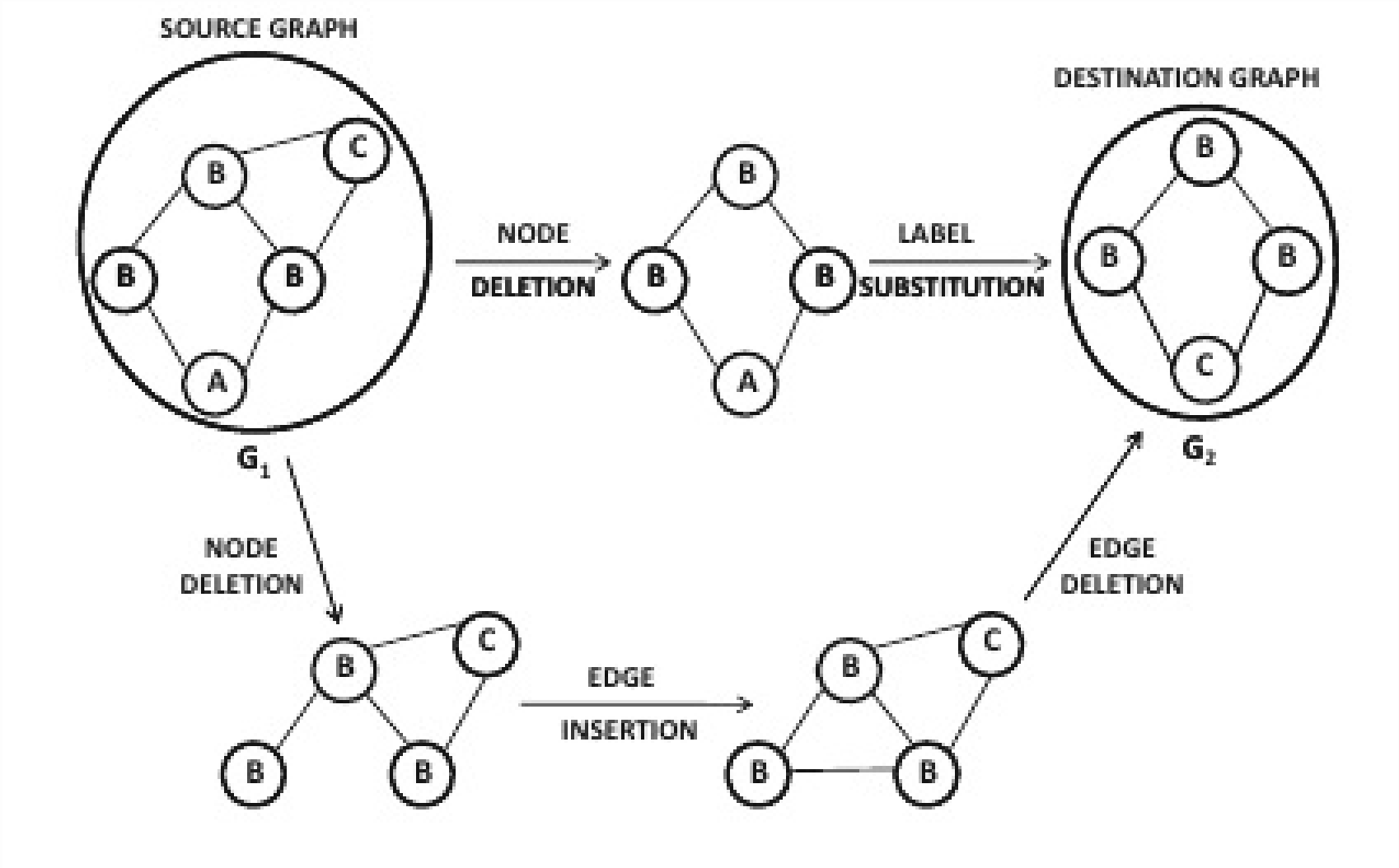


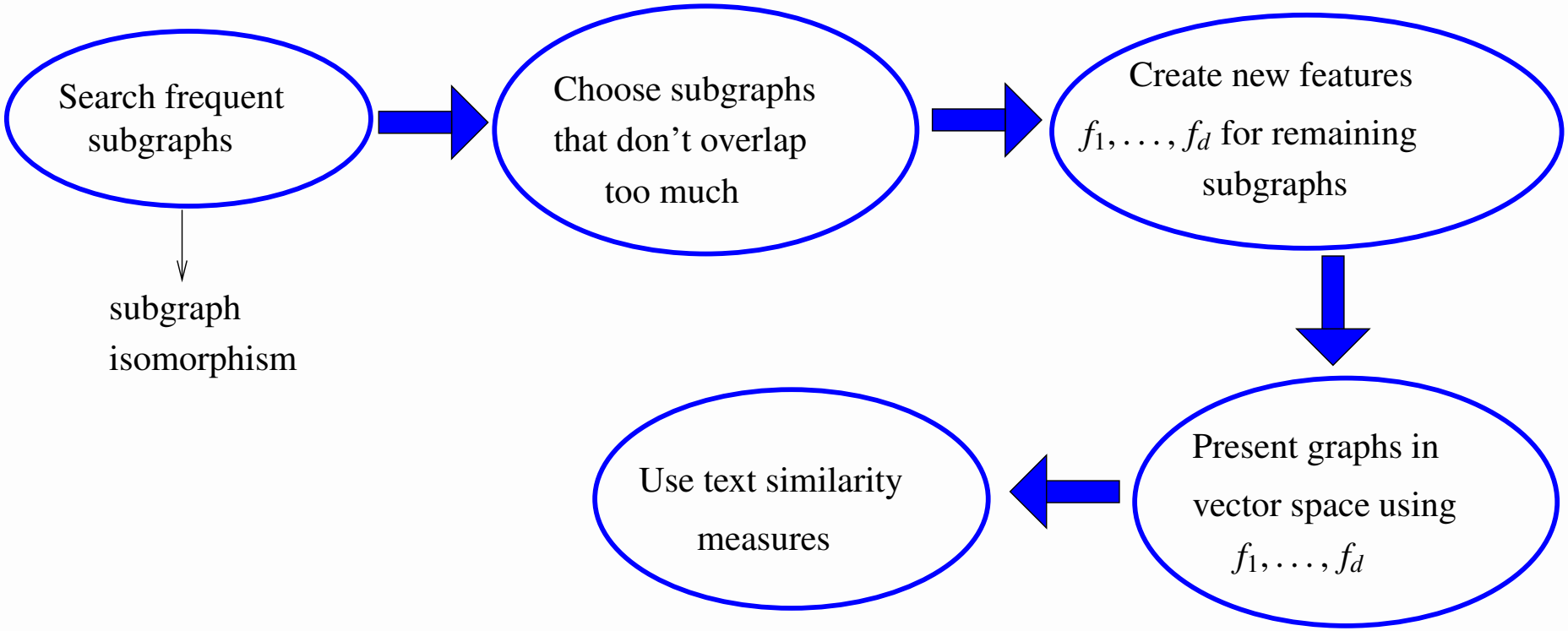
Image source: Aggarwal Fig. 17.6

Transformation-based distances

Idea: Transform graphs into a new space where distances are easier to calculate

a) Type transport using frequent subgraphs
b) Topological descriptors
c) Kernel similarity

Type transport using frequent subgraphs



f_i = number of times ith subgraph occurs in G or binary or tf-idf presentation

involves an NP-hard subproblem

Topological descriptors

Idea: calculate different kinds of indices from graphs $\Rightarrow$ new numerical features $\Rightarrow$ Use distances for numerical data

- structural information lost
- utility domain-specific (e.g., good in chemical domain)
- e.g., Wiener index:

$$W(\mathbf{G}) = \sum_{v,u \in \mathbf{V}} d(v,u)$$

$d(v,u)$=length of shortest path from v to u

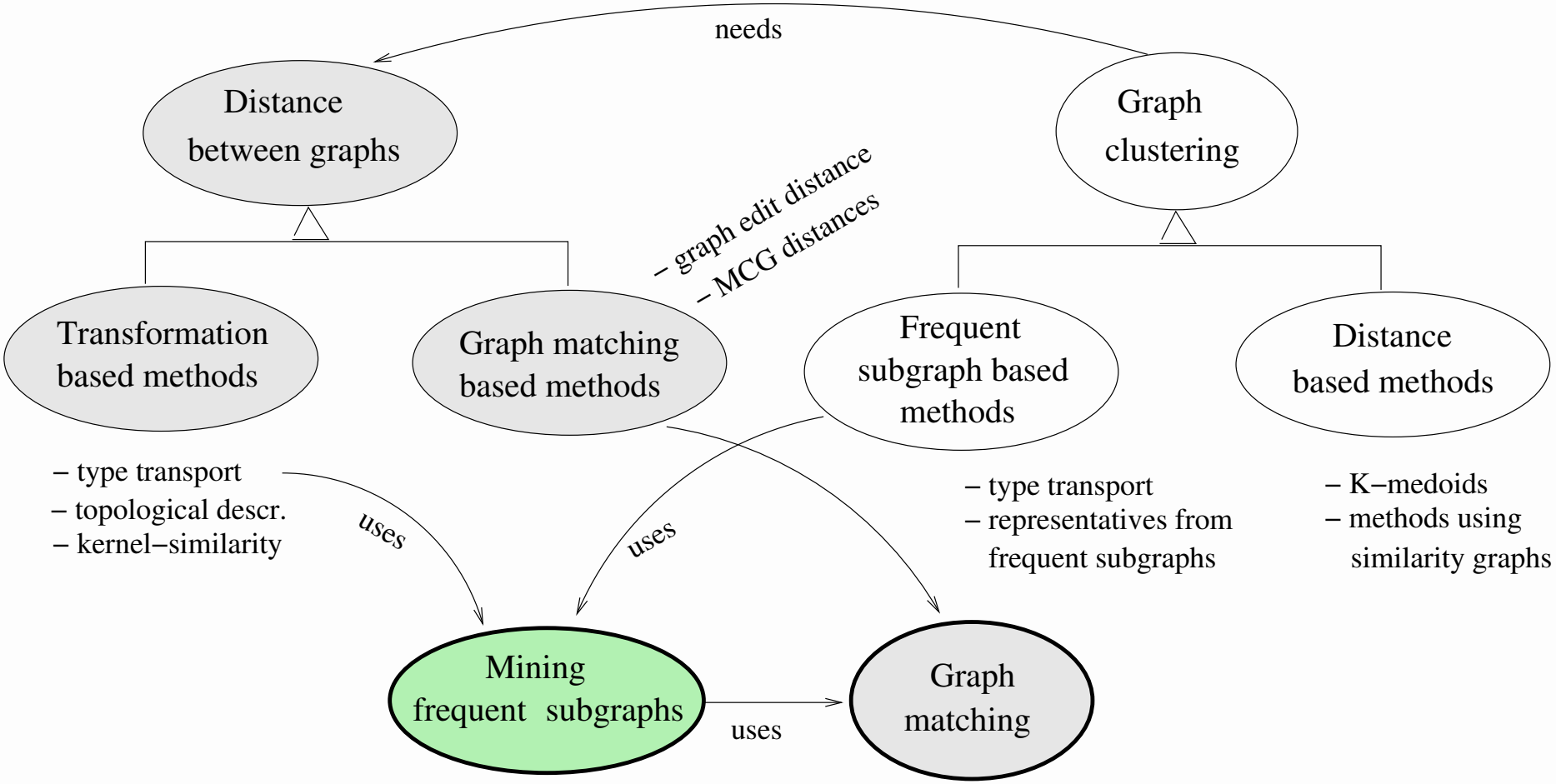
- more in Aggarwal Ch 17.3.2

Kernel similarity

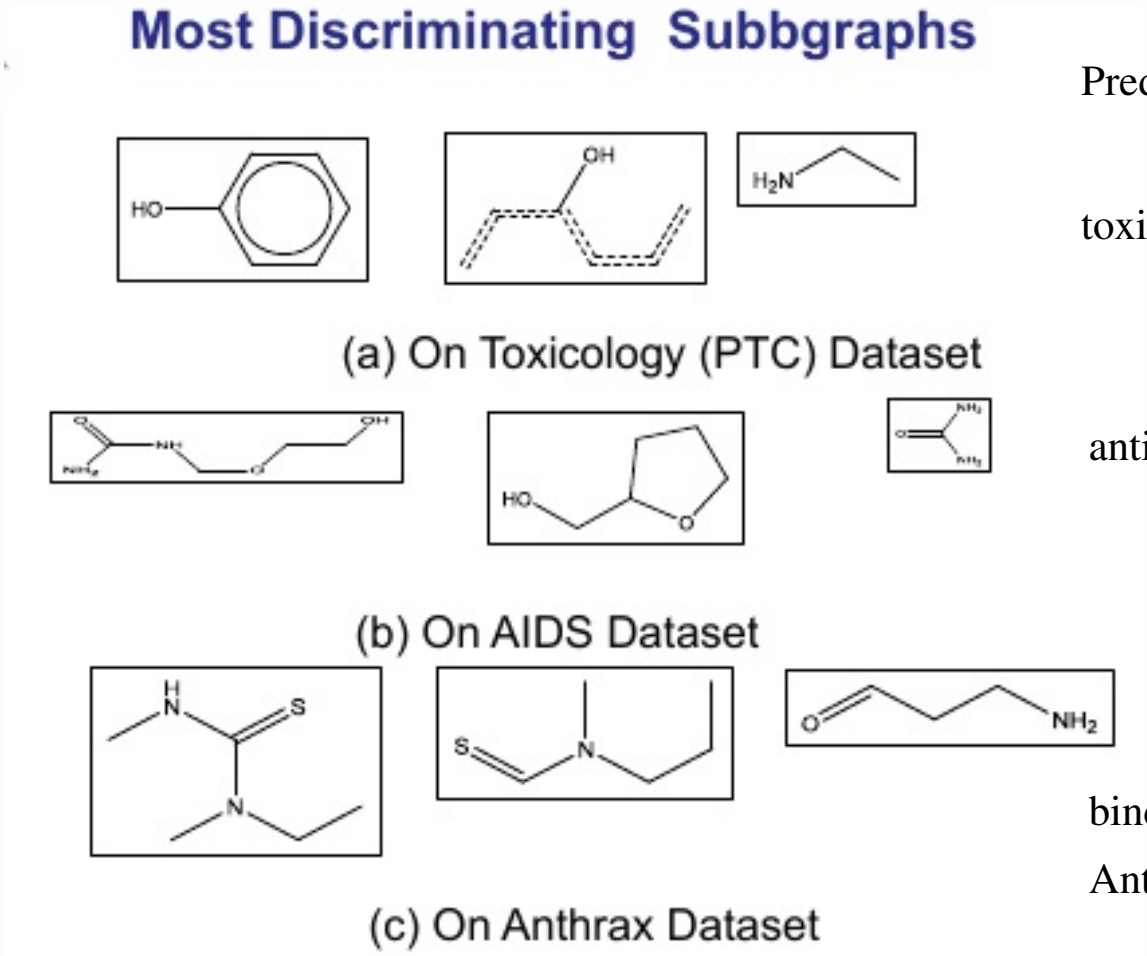
Idea:

- Assume transformation Φ such that similarity of $\mathbf{G}_1$ and $\mathbf{G}_2$ can be measured by $\Phi(\mathbf{G}_1) \cdot \Phi(\mathbf{G}_2)$
- Design **kernel function** K such that $K(\mathbf{G}_1, \mathbf{G}_2) = \Phi(\mathbf{G}_1) \cdot \Phi(\mathbf{G}_2)$ and use it as a similarity measure (without transformation)
- e.g. shortest path kernel ($\mathcal{O}(n^4)$) and random walk kernel ($\mathcal{O}(n^6)$)
- practical for small graphs
- more in Aggarwal Ch 17.3.3

Next to frequent subgraph discovery



Frequent subgraph discovery: Motivation



Predict:

toxicity of compounds

anti−HIV activity

binding ability with Anthrax toxin

Image source: `https://slideplayer.com/slide/5894097/`

Frequent subgraph discovery

Task: Given graph database, search frequent subgraphs given threshold $\min_{fr}$.

- Search idea: utilize **monotonicity of frequency**!
- If $\mathbf{G}_1$ is a subgraph of $\mathbf{G}_2$, then $fr(\mathbf{G}_1) \geq fr(\mathbf{G}_2)$
- similar algorithms than for frequent itemsets, but more complex
- two variants: size of graph may refer to a) number of nodes b) number of edges
  $\Rightarrow$ how new candidates are generated

GraphApriori algorithm

$\mathcal{F}_i$ = frequent subgraphs of size i, $\mathcal{C}_i$ = candidates

- $\mathcal{F}_1 = \{\mathbf{G} \mid \text{where } |\mathbf{G}| = 1, P(\mathbf{G}) \geq \min_{fr}\}$; $i = 1$
- while $\mathcal{F}_i \neq \emptyset$
  - generate candidates $\mathcal{C}_{i+1}$ from $\mathcal{F}_i$
  - prune $\mathbf{G} \in \mathcal{C}_{i+1}$ if $\mathbf{G}$ has a subgraph $\mathbf{G}'$ such that $|\mathbf{G}'| = i$ and $\mathbf{G}' \notin \mathcal{F}_i$ (=**monotonicity criterion**)
  - count frequencies $fr(\mathbf{G})$, $\mathbf{G} \in \mathcal{C}_{i+1}$
  - set $\mathcal{F}_{i+1} = \{\mathbf{G} \in \mathcal{C}_{i+1} \mid P(\mathbf{G}) \geq \min_{fr}\}$
  - $i = i + 1$
- return $\cup_i \mathcal{F}_i$

GraphApriori: Candidate generation

For all $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{F}_i$, $|\mathbf{G}_1| = |\mathbf{G}_2| = i$

1. determine if $\mathbf{G}_1$ and $\mathbf{G}_2$ have a common subgraph $\mathbf{G}_0$ of size $i - 1$
   - may be many isomorphic matchings $\Rightarrow$ **many alternative $\mathbf{G}_0$s!**
2. for each $\mathbf{G}_0$ create candidate graphs of size $i + 1$
   - **node-based**: include all common + 2 non-matching nodes (with extra edge or not)
   - **edge-based**: include all $i - 1$ common edges and 2 unique edges (with extra node or not)

- same subgraphs may be generated multiple times $\Rightarrow$ redundancy checking

Example of node-based join

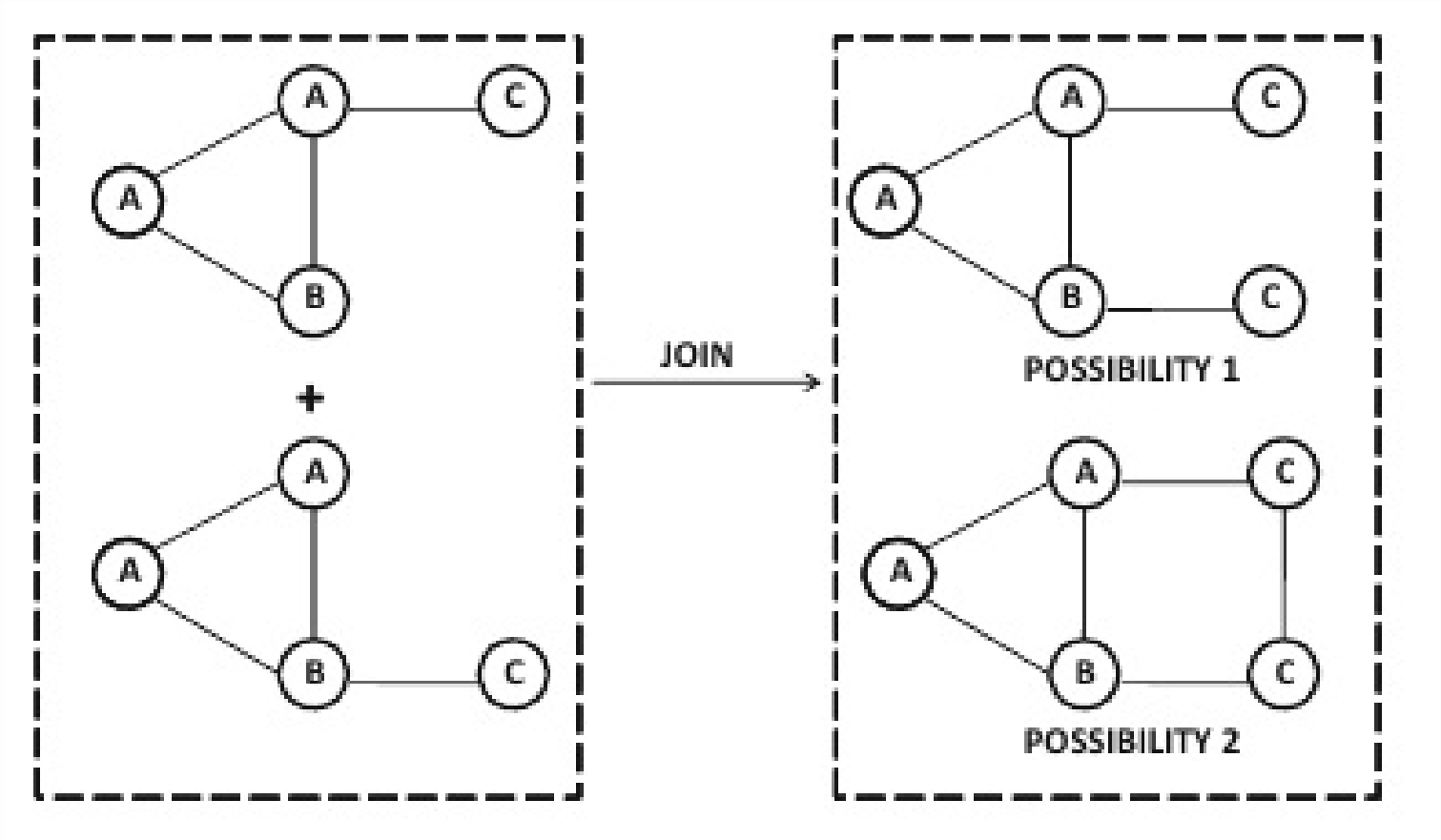


Image source: Aggarwal Fig. 17.12

Example of edge-based join

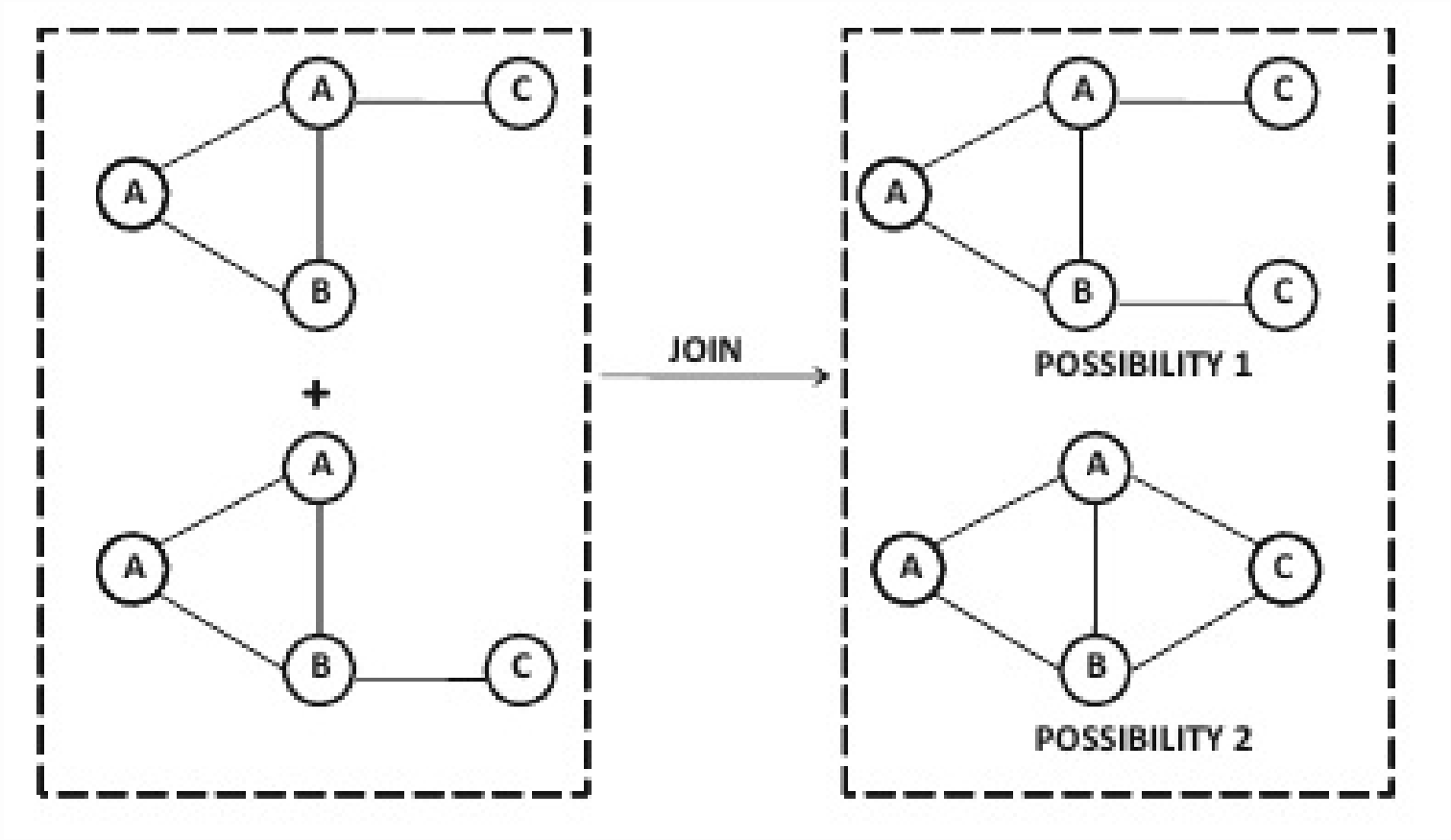


Image source: Aggarwal Fig. 17.13

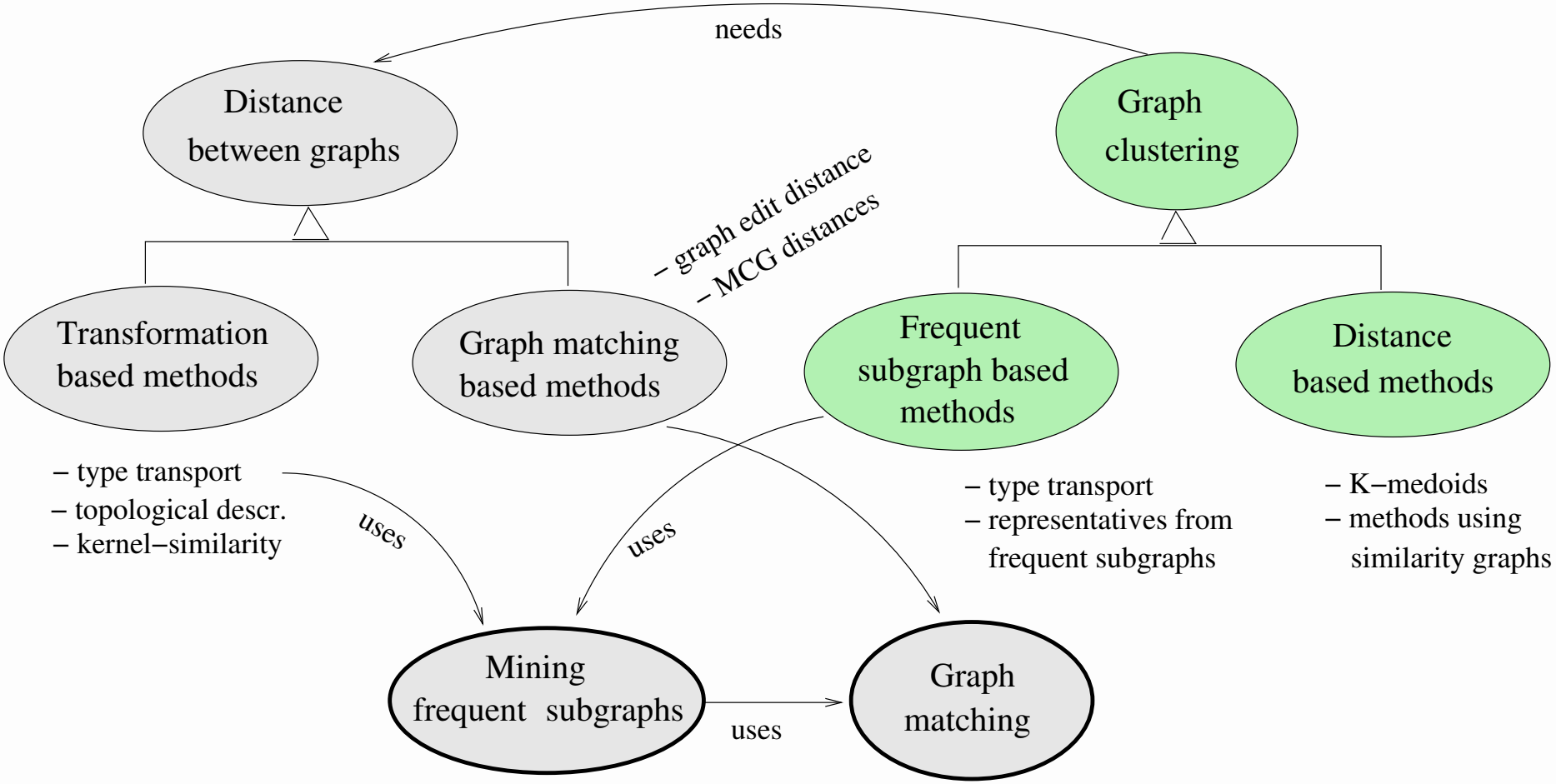
Why this is heavy?

- number of candidate patterns may be huge!
- subgraph isomorphism to identify pairs of subgraphs for joining
- graph isomorphism for redundancy checking
- subgraph isomorphism for monotonicity pruning
- subgraph isomorphism for frequency counting

Easier if

- many unique node labels
- only small subgraphs are searched
- edge-based join is used (usually less candidates)

Next to graph clustering



Distance-based clustering methods

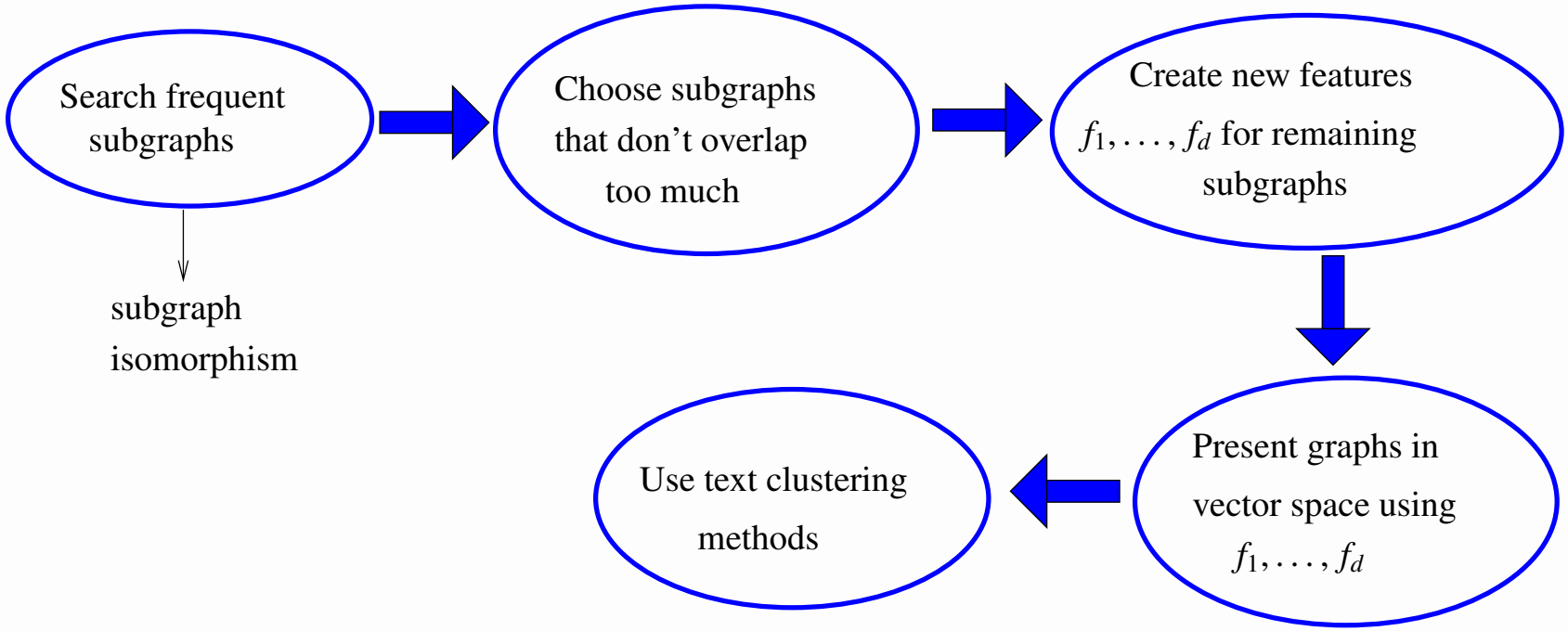
Common approaches:

1. K-medoids (needs just a distance function)
2. Spectral and other graph-based methods
   - construct a nearest neighbour/similarity graph of graph objects
   - cluster nodes of the new graph

Remember: graph distance measures very expensive to compute! $\rightarrow$ suitable for smaller graphs

Methods based on frequent subgraphs

Approach 1. Type transport: graphs → multidimensional



f_i = number of times
ith subgraph occurs in G
or binary or tf-idf representation

involves an NP-hard subproblem

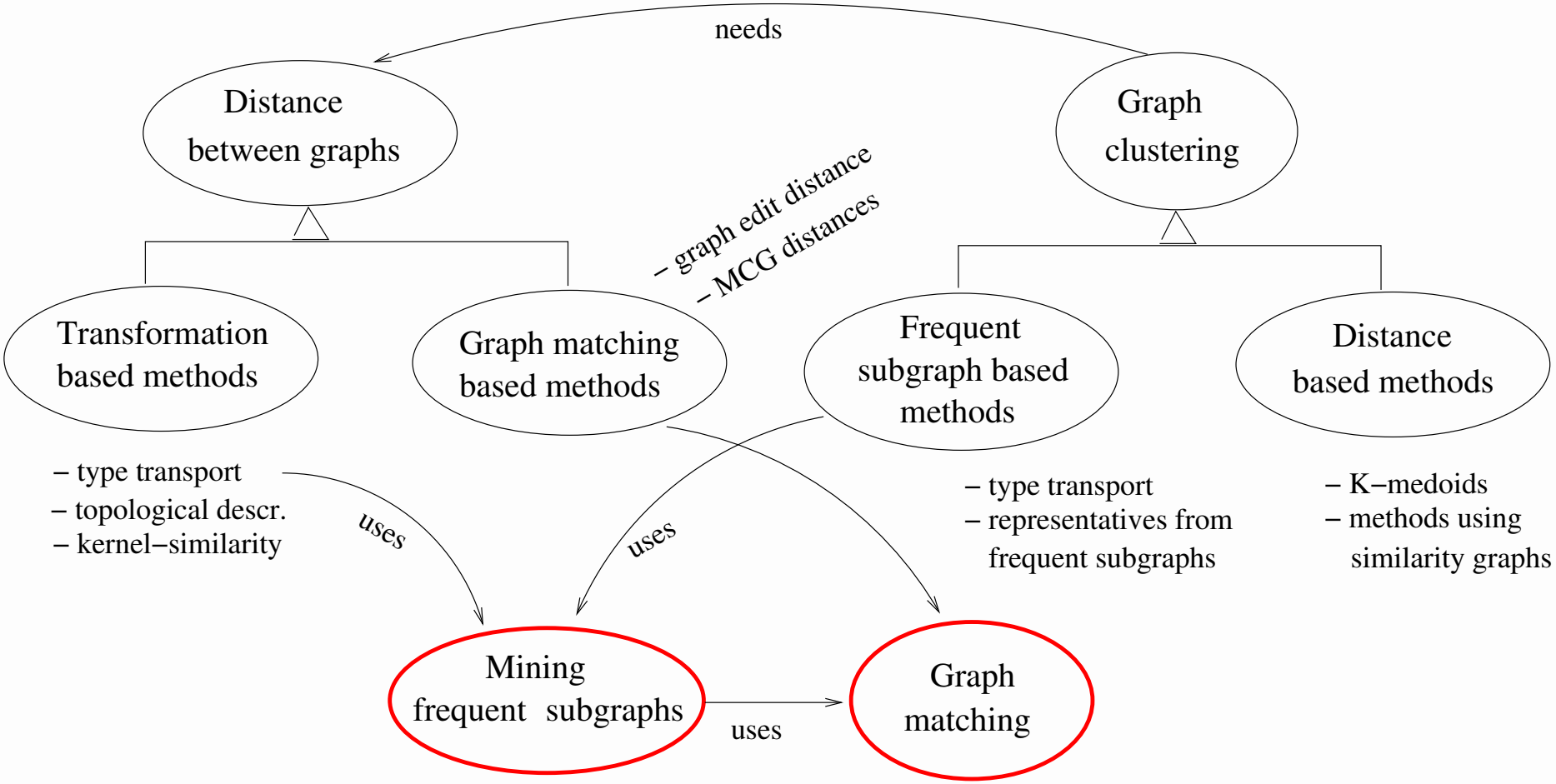
Methods based on frequent subgraphs

Approach 2. XProj: cluster representatives = **sets** of frequent subgraphs

- Initialization: Create K random clusters $C_1, \ldots, C_K$
- for all C_i: $\mathcal{F}_i$ = set of frequent subgraphs (of a given size) from C_i
- repeat until convergence:
  - assign each $\mathbf{G}_j$ to C_i where $sim(G_j, \mathcal{F}_i)$ largest
  - for all C_i determine new $\mathcal{F}_i$

$sim(G_j, \mathcal{F}_i)$ = fraction of frequent graphs in $\mathcal{F}_i$ that occur in $\mathbf{G}_j$

Summary


needs
Distance between graphs
Graph clustering
– graph edit distance
– MCG distances
Transformation based methods
Graph matching based methods
Frequent subgraph based methods
Distance based methods
– type transport
– topological descr.
– kernel–similarity
uses
uses
– type transport
– representatives from frequent subgraphs
– K–medoids
– methods using similarity graphs
Mining frequent subgraphs
uses
Graph matching