UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

MASTER IN COMPUTER SCIENCE

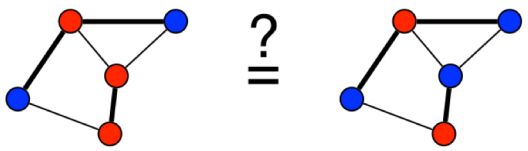
Pattern Recognition

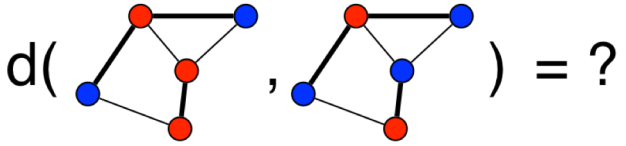
Lecture 9 : Graph Matching I

Dr. Andreas Fischer
andreas.fischer@unifr.ch

Graph Matching

- Strings vs graphs
  - Formally, strings are special cases of graphs.
  - Graph-based representation is more flexible and powerful but also computationally more challenging.
- Exact matching
  - Test if two graphs are equal (isomorphism).
  - Test if parts are equal (subgraph isomorphism).
- Inexact matching
  - Compute a dissimilarity between two graphs.

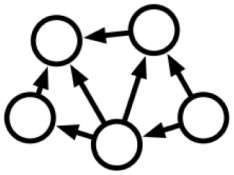




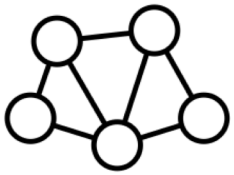
Definitions

Graph

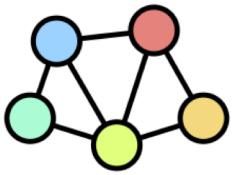
- A *graph* is a 4-tuple $g = (V, E, \alpha, \beta)$
  - V is a finite set of nodes
  - $E \subseteq V \times V$ is the set of edges
  - $\alpha : V \rightarrow L_V$ is the node labeling function
  - $\beta : E \rightarrow L_E$ is the edge labeling function
- L_V and L_E are label alphabets (symbols, numbers, vectors, …)
  - *unlabeled graphs* are a special case with $|L_V| = |L_E| = 1$
- Edges $e = (u,v) \in E$ are directed
  - *undirected graphs* are a special case where for each edge e = (u,v) there exists an edge e' = (v,u)



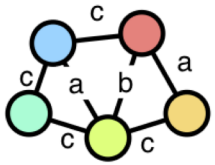
directed, unlabeled



undirected, unlabeled



undirected, labeled nodes

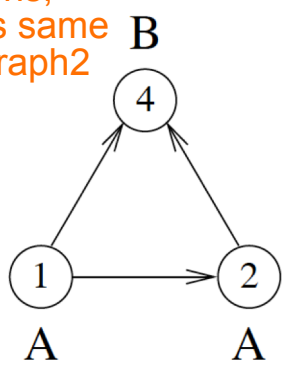


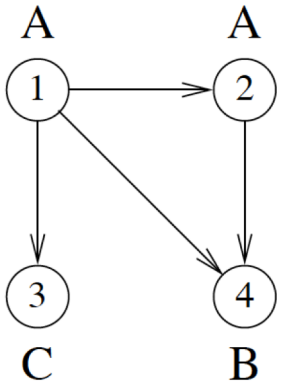
undirected, labeled

Subgraph

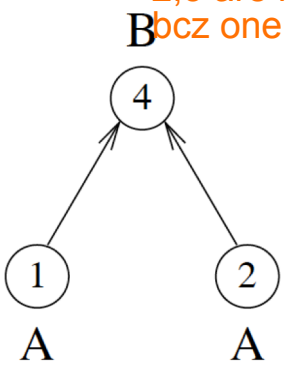
- Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$. g_1 is a *subgraph* of g_2, $g_1 \subseteq g_2$, if:
  - $V_1 \subseteq V_2$
  - $E_1 = E_2 \cap (V_1 \times V_1)$
  - $\alpha_1(v) = \alpha_2(v)$ if $v \in V_1$, undefined otherwise
  - $\beta_1(e) = \beta_2(e)$ if $e \in E_1$, undefined otherwise
- Sometimes the weaker condition $E_1 \subseteq E_2$ is considered for the edges.
  - In this case, a graph that satisfies the stronger condition above is called *induced subgraph*.
  - For the following algorithms both definitions are admissible but the stronger condition $E_1 = E_2 \cap (V_1 \times V_1)$ is often more useful.

1,2 are subgraphs, bcz graph 1 has same matching has graph2





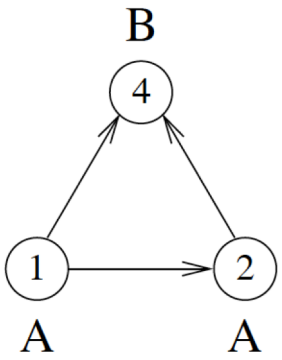
2,3 are not subgrapgs bcz one edge is missing

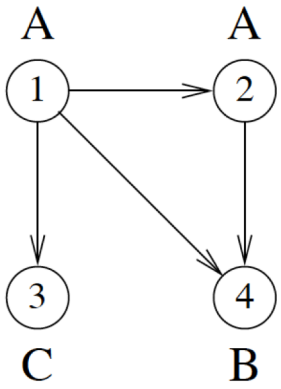


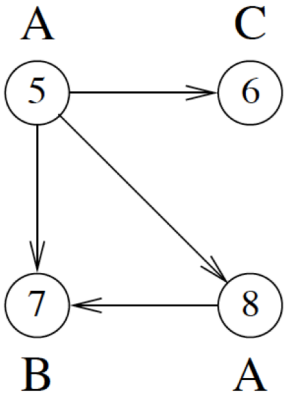
Graph Isomorphism

- Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$. A bijective function mapping $f : V_1 \rightarrow V_2$ is a *graph isomorphism* from g_1 to g_2 if:
  - $\alpha_1(u) = \alpha_2(f(u))$
  - for each edge $e_1 = (u,v) \in E_1$ there exists an edge $e_2 = (f(u),f(v))$ with $\beta_1(e_1) = \beta_2(e_2)$
  - for each edge $e_2 = (x,y) \in E_2$ there exists an edge $e_1 = (f^{-1}(x),f^{-1}(y))$ with $\beta_1(e_1) = \beta_2(e_2)$
- Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$. A injective function mapping $f : V_1 \rightarrow V_2$ is a *subgraph isomorphism* from g_1 to g_2 if:
  - there exists a subgraph $g \subseteq g_2$ such that f is a graph isomorphism from g_1 to g

2,3 are iso.. bcz it had same number of nodes with exactly matching of edges

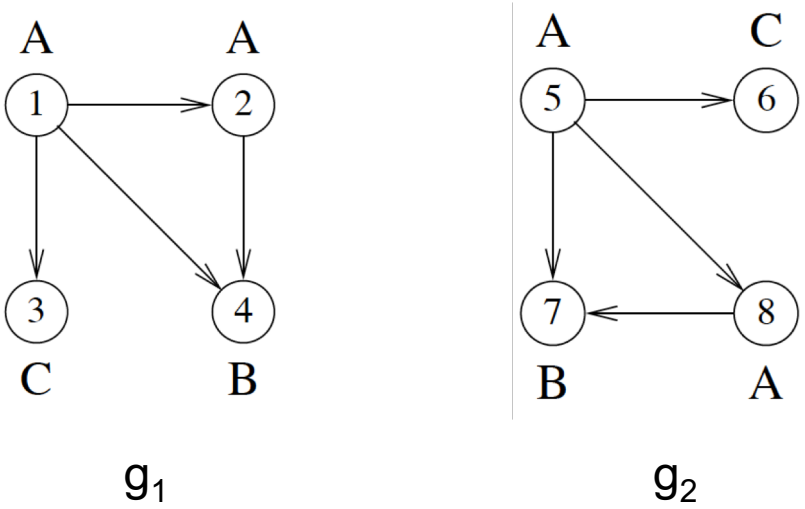






Example: Graph Isomorphism

- Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ as follows:
  - $V_1 = \{1, 2, 3, 4\}$ and $V_2 = \{5, 6, 7, 8\}$
  - $E_1 = \{(1,2), (1,3), (1,4), (2,4)\}$ and $E_2 = \{(5,6), (5,7), (5,8), (8,7)\}$
  - $\alpha_1 : 1 \mapsto A, 2 \mapsto A, 3 \mapsto C, 4 \mapsto B$
    $\alpha_2 : 5 \mapsto A, 6 \mapsto C, 7 \mapsto B, 8 \mapsto A$
  - β_1, β_2 : no edge labels are used
- Clearly there exist a graph isomorphism from g_1 to g_2.



g_1 g_2

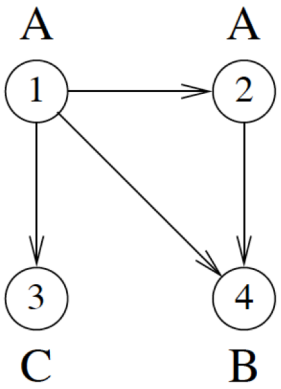
Common Subgraph and Supergraph

- Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$. A graph $g = (V, E, \alpha, \beta)$ is a *common subgraph* of g_1 and g_2 if:
  - there exist subgraph isomorphisms $f_1 : V \rightarrow V_1$ and $f_2 : V \rightarrow V_2$
  - A common subgraph g of g_1 and g_2 is a *maximum common subgraph* if there is no other common subgraph of g_1 and g_2 with more nodes.
- Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$. A graph $g = (V, E, \alpha, \beta)$ is a *common supergraph* of g_1 and g_2 if:
  - there exist subgraph isomorphisms $f_1 : V_1 \rightarrow V$ and $f_2 : V_2 \rightarrow V$
  - A common supergraph g of g_1 and g_2 is a *minimum common supergraph* if there is no other common supergraph of g_1 and g_2 with less nodes.
- Note that the maximum common subgraph and the minimum common supergraph are not necessarily unique.

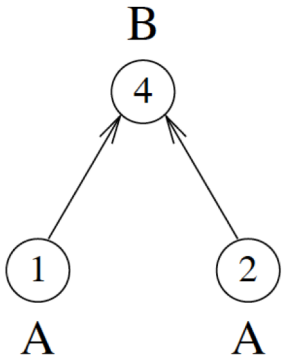
Example: Subgraph and Supergraph

- g_3 is a maximum common subgraph of g_1 and g_2
- g_4 is a minimum common supergraph of g_1 and g_2

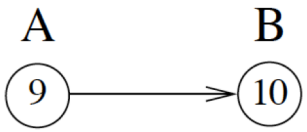
g3 has presented in g1,g2 and g4 too so it is common graph



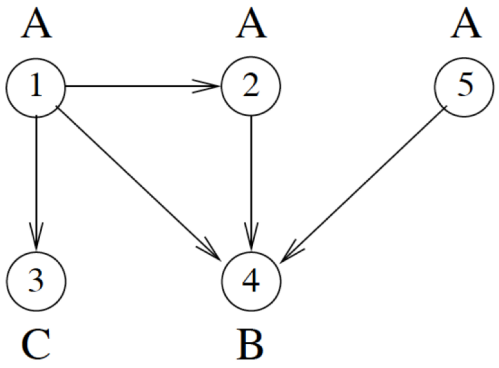
g_1



g_2



g_3

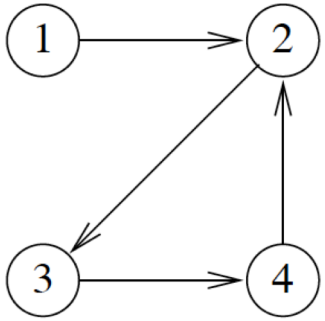


g_4

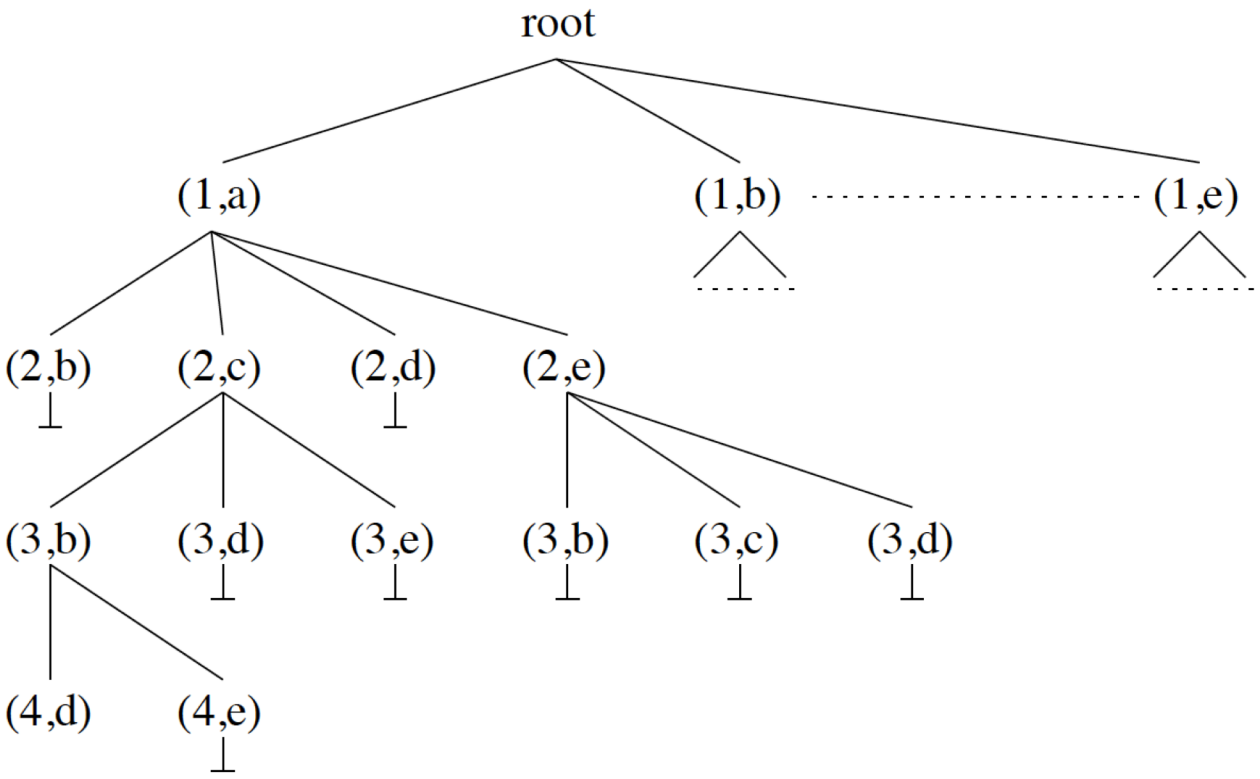
Exact Matching

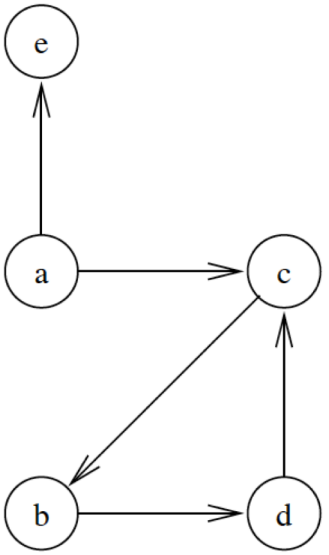
Graph and Subgraph Isomorphisms

- Input: graphs g_1 and g_2
- Output: graph or subgraph isomorphisms from g_1 to g_2
- The following procedure is often used:



g_1

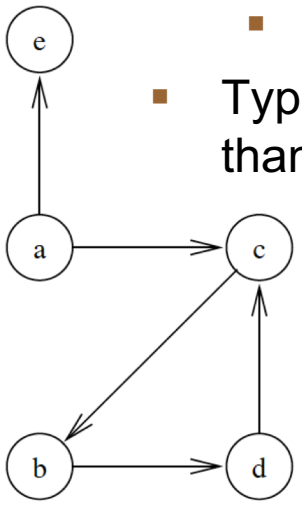




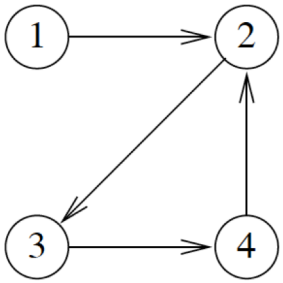
g_2

Structure Conservation Criterion

- Dead ends are reached if the *structure conservation criterion* is violated: if $f(u_1) = u_2$ then
  - each node $v_1 \in V_1$ with $(u_1,v_1) \in E_1$ can only be mapped on a node v_2 in V_2 with $(u_2,v_2) \in E_2$ and *vice versa*
  - furthermore $\alpha_1(v_1) = \alpha_2(v_2)$ and $\beta_1((u_1,v_1)) = \beta_2((u_2,v_2))$
  - analog for $(u_1,v_1) \notin E_1$, $(v_1,u_1) \in E_1$, and $(v_1,u_1) \notin E_1$
- If all nodes from g_1 are mapped to nodes from g_2 without violating the structure conservation criterion then
  - if $|V_1| < |V_2|$: $g_1 \subseteq g_2$ (g_1 is isomorph to subgraph of g_2)
  - if $|V_1| = |V_2|$: $g_1 = g_2$ (g_1 and g_2 are isomorph)
- Typically, there are several graph and subgraph isomorphisms. If more than one has to be found, the whole search tree has to be expanded.



g_2



g_1

Time Complexity

- Time complexity of this greedy procedure is $O(|V_1|^{|V2|})$.

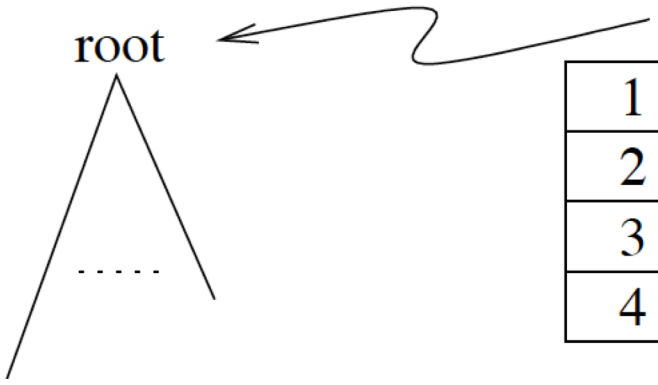
We cant have efficient way of isomorphism

- In general, subgraph isomorphism is known to be NP-complete.
- For graph isomorphism, there is a recent proposal by László Babai who claims that it can be solved in quasi-polynomial time:

  L. Babai. Graph isomorphism in quasipolynomial time. CoRR, arXiv:1512.03547, 2015.

Future Match Table

- The greedy procedure can be speeded up with a *future match table*, a look-ahead technique.
  - a row for each node $u \in V_1$
  - a column for each node $v \in V_2$
- Initialization:
  - $F(u,v) = 1$ if $\alpha_1(u) = \alpha_2(v)$ and in-/outdegree(u) ≤ in-/outdegree(v)
  - $F(u,v) = 0$ otherwise
- $F(u,v) = 1$ means that $u \in V_1$ can potentially be mapped to $v \in V_2$. Only those nodes have to be considered during the search procedure.



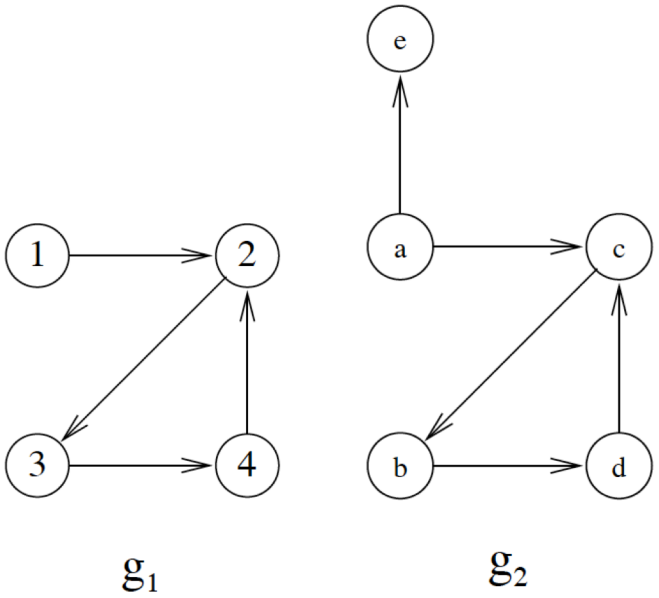
	a	b	c	d	e
1	1	1	1	1	
2			1		
3		1	1	1	
4		1	1	1	

Updating the Future Match Table

- If u has been mapped to v, that is f(u) = v, the following updates are performed:
  1. Copy F(u,v) from the predecessor in the search tree
  2. Set all entries in the row of u to 0 except F(u,v) = 1
  3. Set all entries in the column of v to 0 except F(u,v) = 1
  4. **for** all unmapped nodes x in V_1
     for all nodes y in V_2 with F(x,y) = 1
     if structure conservation criterion is violated
     then F(x,y) = 0
     if there is a row in F(u,v) that contains only zeroes
     then backtrack
     endfor
     endfor

Example 1: Future Match Table

- - : removed in steps 2/3
- x : removed in step 4
- Linear search after (1,a)!



root

	a	b	c	d	e
1	1	1	1	1	
2			1		
3		1	1	1	
4		1	1	1	

(1,a)

	a	b	c	d	e
1	1	-	-	-	
2			1		
3		1	x	1	
4		1	x	1	

(2,c)

	a	b	c	d	e
1	1				
2			1		
3		1		x	
4		x		1	

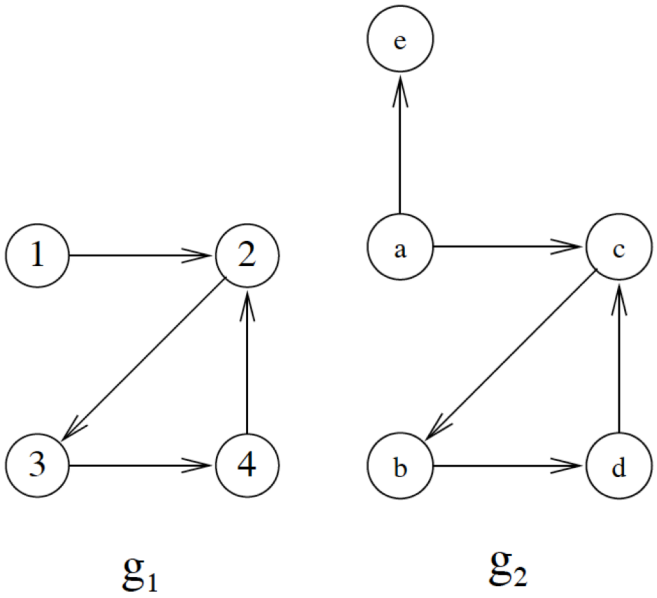
(3,b)

	a	b	c	d	e
1	1				
2			1		
3		1			
4				1	

(4,d)

Example 2: Future Match Table

- Another speedup is to select the node that has the fewest ones.
- The whole search is linear!



root

	a	b	c	d	e
1	1	1	1	1	
2			1		
3		1	1	1	
4		1	1	1	

(2,c)

	a	b	c	d	e
1	1	x	-	1	
2			1		
3		1	-	x	
4		x	-	1	

(3,b)

	a	b	c	d	e
1	1			x	
2			1		
3		1			
4				1	

(1,a)

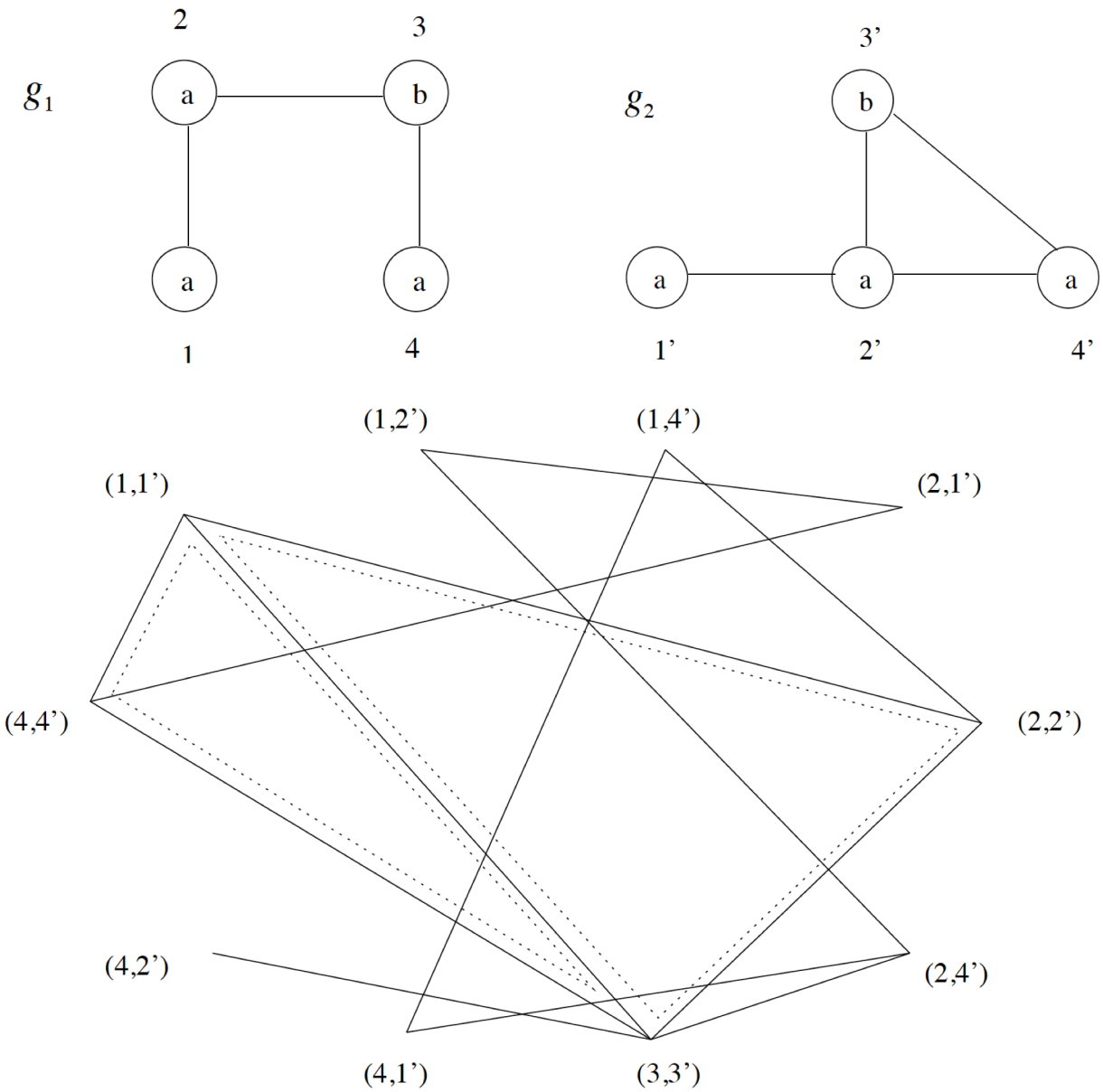
	a	b	c	d	e
1	1				
2			1		
3		1			
4				1	

(4,d)

Maximum Common Subgraph

- Input: graphs g_1 and g_2
- Output: maximum common subgraph of g_1 and g_2
- Can also be computed with a search procedure. Another possibility is to solve it by finding a maximum clique in the so-called association graph.
- The *association graph* of g_1 and g_2 is an undirected, unlabeled graph G=(V,E) defined as follows:
  - $V = \{ (u_1,u_2) \mid u_1 \text{ in } V_1 \wedge u_2 \text{ in } V_2 \wedge \alpha(u_1) = \alpha(u_2) \}$
    (that is, nodes in G are the Cartesian product of nodes in g_1 and nodes in g_2 that have the same label)
  - E : there is an edge between (u_1,u_2) and (v_1,v_2) iff the mapping $f(u_1)=v_1$ and $f(u_2)=v_2$ fulfills the structure conservation criterion.
    (that is, if there is an edge from u_1 to v_1 in g_1, an edge with the same label has to exist from u_2 to v_2 in g_2; if and only if this condition is fulfilled G contains an edge from (u_1,u_2) to (v_1,v_2))

Example: Association Graph

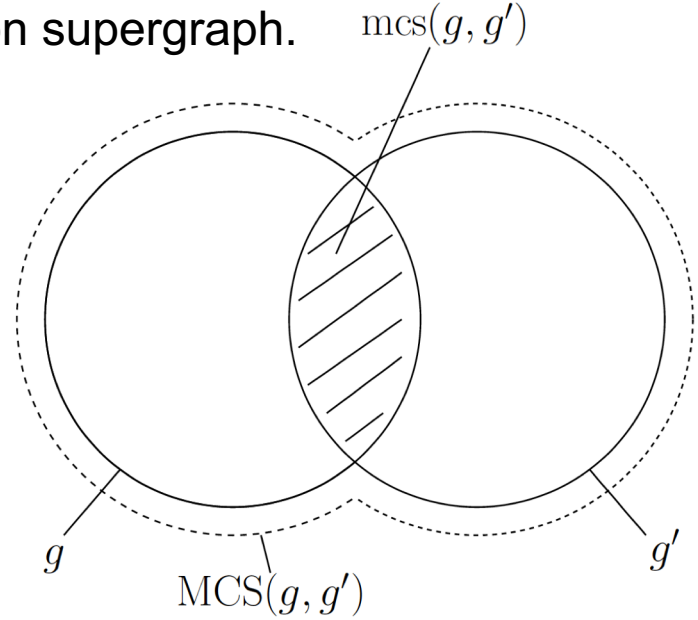


Maximum Clique

- Each node (u,v) in G corresponds to a mapping f(u)=v.
- If two nodes (u_1,v_1) and (u_2,v_2) are connected in G, both corresponding mappings fulfill the structure conservation criterion.
- Therefore, a completely connected subset of nodes in G, called *clique*, corresponds to an isomorphism between a subgraph $g_1' \subseteq g_1$ and a subgraph $g_2' \subseteq g_2$.
- Therefore, a maximum clique in G corresponds to a maximum common subgraph of g_1 and g_2.
  - To find a maximum clique, all subsets of nodes in G have to be tested for complete connectedness.

Other Maximum Clique Applications

- This procedure can also be used to find graph and subgraph isomorphisms:
  - If all nodes of g_1 (g_2) are represented in the maximum clique, a subgraph isomorphism from g_1 to g_2 (from g_2 to g_1) is found.
  - If all nodes of both graphs are represented in the maximum clique, a graph isomorphism from g_1 to g_2 is found.
- It can be shown that finding a minimum common supergraph corresponds to finding a maximum common subgraph.
  - Therefore, the maximum clique procedure can also be used to compute a minimum common supergraph.



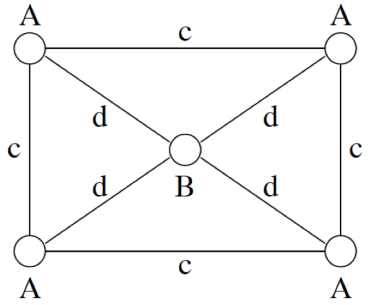
Inexact Matching

Graph Edit Distance

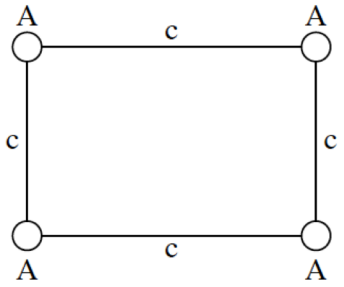
- One of the most general dissimilarity measures for graphs. Can handle arbitrary graphs with any kind of label alphabets for nodes and edges.
  - Flexible and powerful but known to be NP-complete.
  - Other methods include spectral methods (eigenvalue decomposition of the adjacency matrix) and kernel methods.
- Idea: apply edit operations to g_1 to transform it into g_2, similar to the string edit distance.
- Standard set of edit operations:
  - node deletion $v \rightarrow \epsilon$, $v \in V$, with cost $c(v \rightarrow \epsilon)$
  - node insertion $\epsilon \rightarrow v$, $v \in V$, with cost $c(\epsilon \rightarrow v)$
  - node label substitution $a \rightarrow b$; $a,b \in L_V$, with cost $c(a \rightarrow b)$
  - edge deletion $e \rightarrow \epsilon$, $e=(u,v) \in E$, with cost $c(e \rightarrow \epsilon)$
  - edge insertion $\epsilon \rightarrow e$, $e=(u,v) \in E$, with cost $c(\epsilon \rightarrow e)$
  - edge label substitution $c \rightarrow d$; $c,d \in L_E$, with cost $c(c \rightarrow d)$

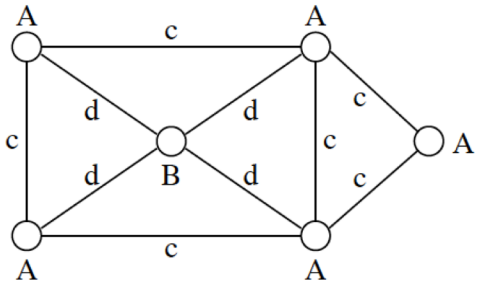
Example: Edit Operations

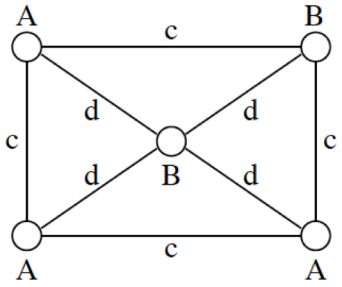
- Original graph:



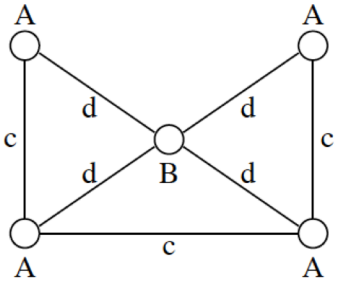
- Node deletion/insertion/substitution (plus implied operations):

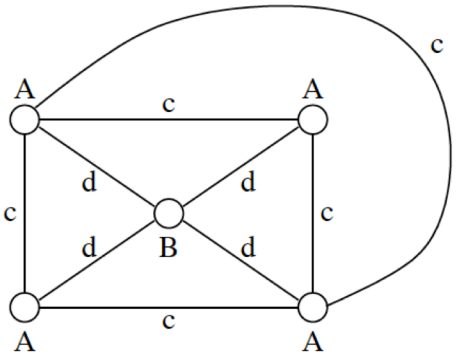


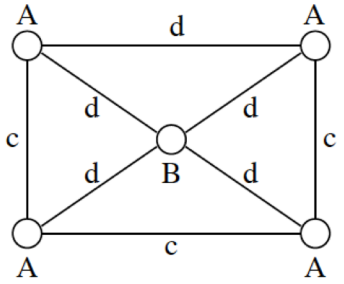




- Edge deletion/insertion/substitution







Definition

- An *edit path* S = $s_1,\ldots,s_n$ is a sequence of node and edge edit operations that completely transform g_1 into g_2.
- The cost of an edit path is the sum of its edit operation costs:

$$c(S) = \sum_{i=1}^{n} c(s_i)$$

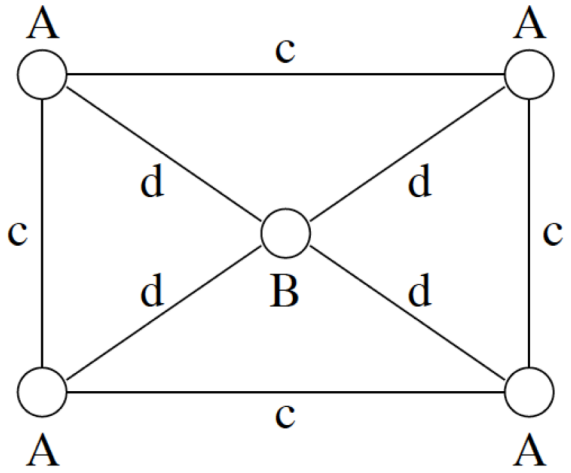
- The *graph edit distance* $d(g_1,g_2)$ between graphs g_1 and g_2 is:

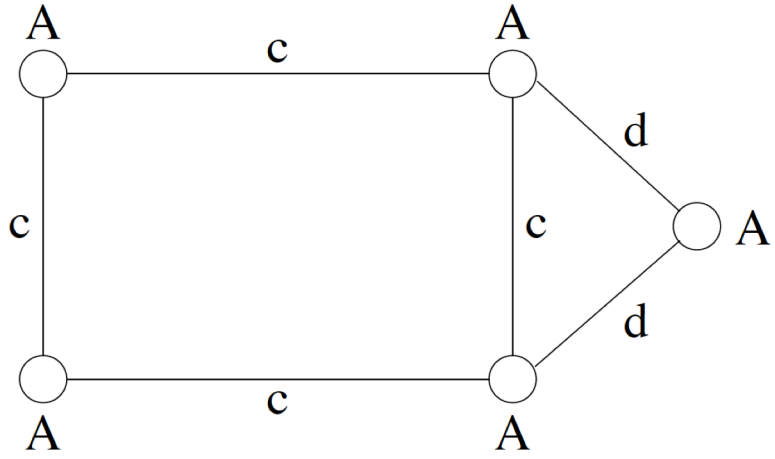
$$d(g_1, g_2) = \min\{c(S) \mid S \text{ is an edit path that transforms } g_1 \text{ into } g_2\}$$

- Note that:
  - The costs for node deletion, insertion, and substitution typically do not depend on the node v but on its label α(v). Same for edges.
  - The deletion of a node implies the deletion of its adjacent edges. Sometimes implicit deletions are distinguished from real ones. Same for node insertion.

Example: Graph Edit Distance

- Standard cost model:
  - cost 1 for deletion / insertion
  - cost 1 for substitution if $\alpha(u) \neq \alpha(v)$, same for edges
  - cost 0 for substitution if $\alpha(u) = \alpha(v)$, same for edges
- $d(g_1,g_2) = 3$ (2 edge deletions, 1 node substitution)



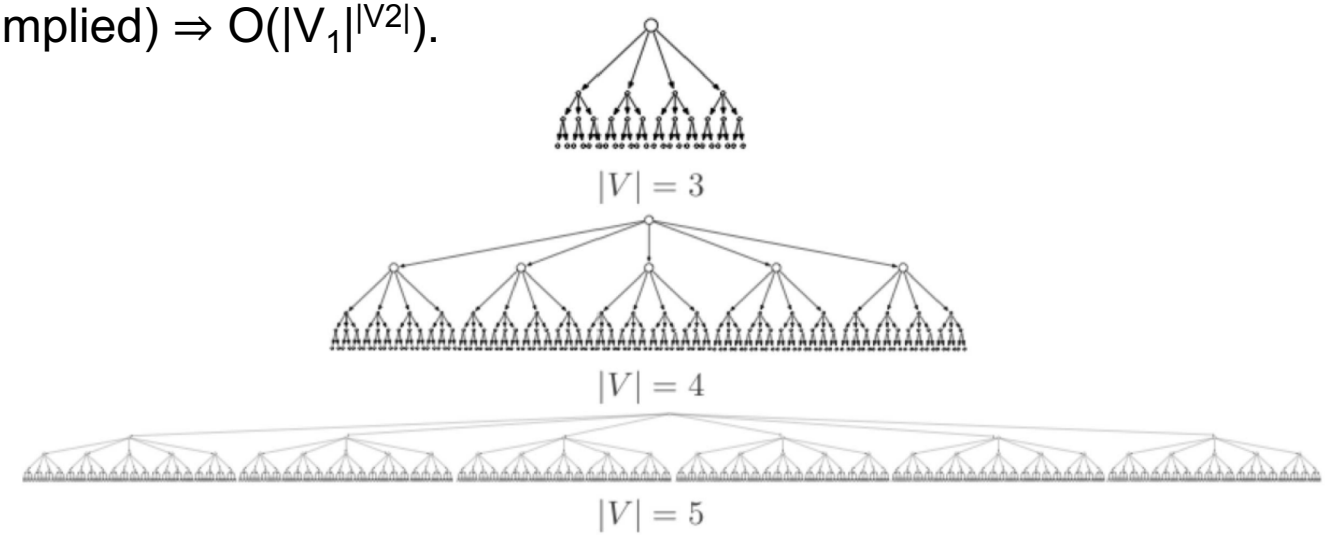


Edit Distance Metric

- The graph edit distance is a metric if:
  - $c(s) \geq 0$ for all edit operations s
  - $c(s) = 0$ iff s is an identical node or edge substitution
  - $c(s) = c(s^{-1})$ where s^{-1} is the inverse edit operation to s
  - $c(s) \leq c(s') + c(s'')$ if $s = s' \circ s''$ where $s' \circ s''$ is the consecutive execution of s' and s''

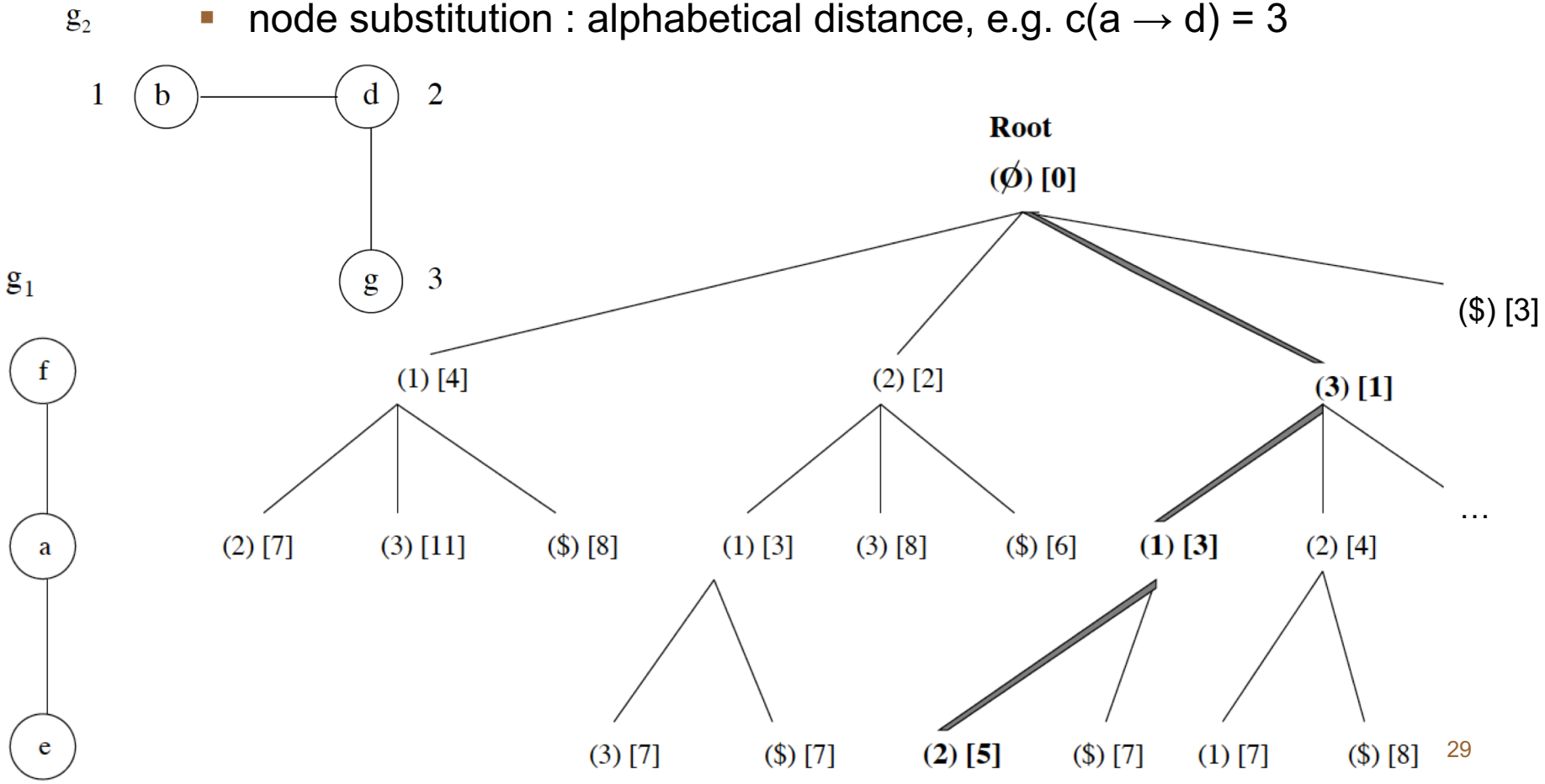
Edit Distance Computation

- With the special cost model c(s) = 0 if s is a node deletion, and c(s) = ∞ otherwise, we get:
  - $d(g_1,g_2) = 0 \Leftrightarrow g_2 \subseteq g_1$
  - Therefore we cannot expect that an algorithm in polynomial time exists for graph edit distance, because subgraph isomorphism is NP-complete.
- Typically, an exhaustive search is performed over all possible node mappings of nodes in g_1 to nodes in g_2 (edge edit operations are implied) ⇒ $O(|V_1|^{|V2|})$.



Example: Edit Distance Computation

- Cost model:
  - node deletion and insertion: 3
  - edge deletion and insertion: 1
  - node substitution : alphabetical distance, e.g. c(a → d) = 3

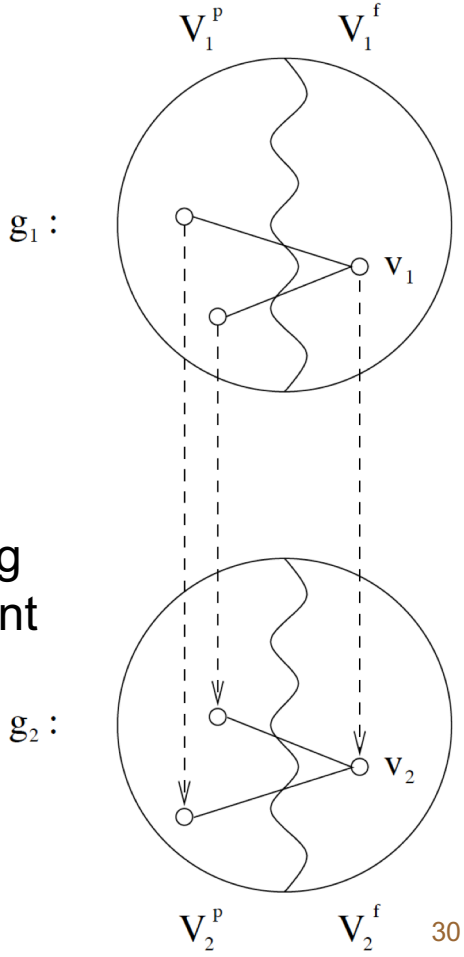


Faster Computation with A* Search

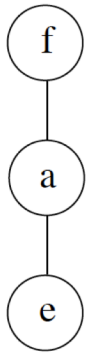
- To speedup the computation, a lower bound of the future cost can be used, known as *A* search*:
  - cost = past cost + h(future cost)
  - h(future cost) ≤ future cost
  - always expand the node with the lowest cost
  - first complete edit path is an optimal solution
- Consider

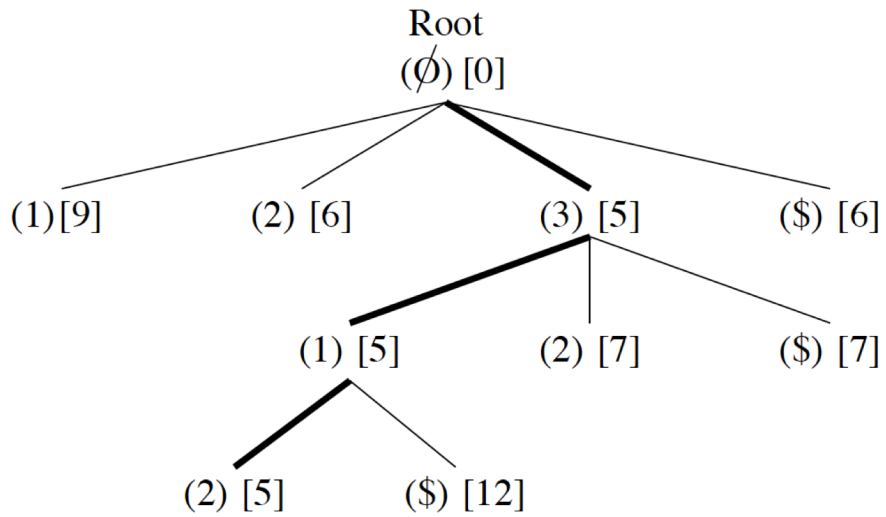
$$V_1 = V_1^p \cup V_1^f, V_2 = V_2^p \cup V_2^f$$

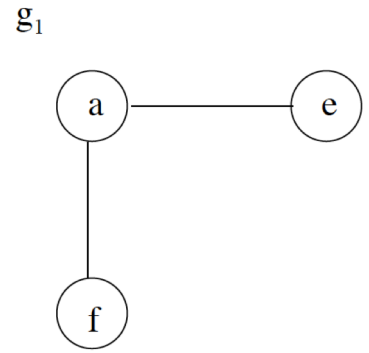
where p refers to the already mapped nodes (past) and f refers to the unmapped nodes (future).
To obtain a lower bound, the best possible mapping is computed for each future node taking into account past mappings.

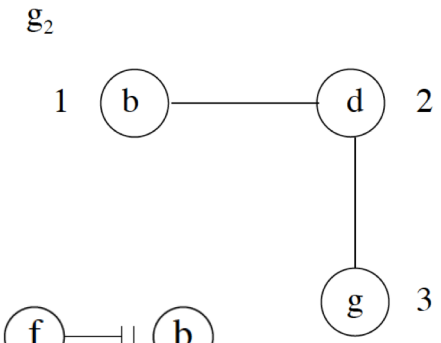


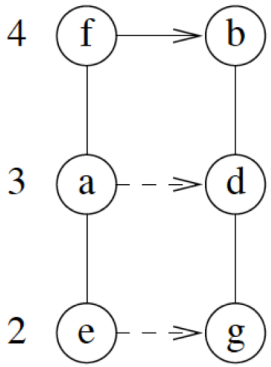
Example: A* Search



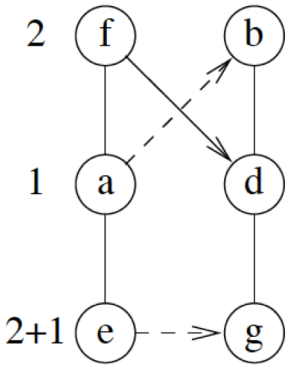




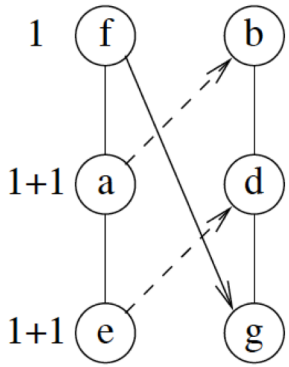




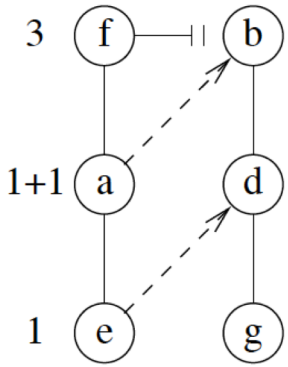
c=9



c=6



c=5



c=6

there is no link betwen e and g so we have to insert g, then it takes 1

Examples

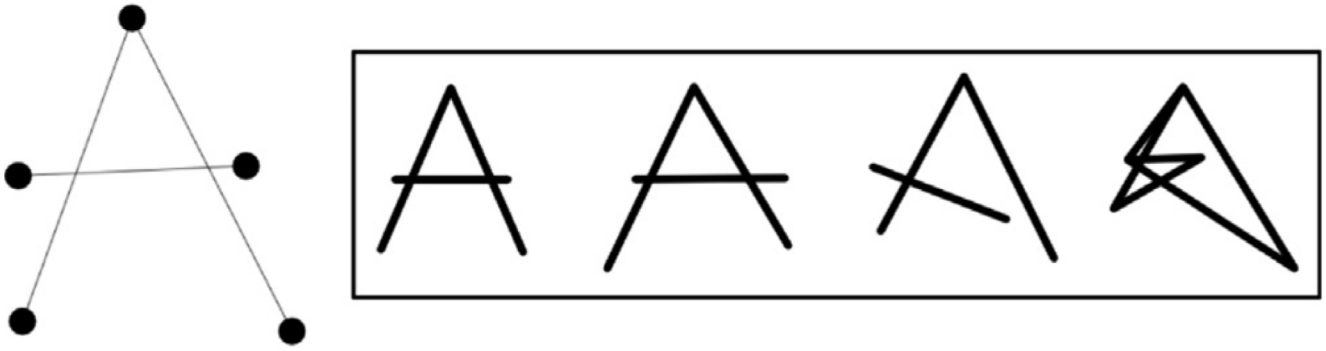
Nearest Neighbor Classification

- NN classification based on graph edit distance:
  - X = graph domain
  - θ = { learning samples }
  - f_θ : assign the class label of the most similar known sample

$$f_\theta(x) = C_i \Leftrightarrow \underset{p \in \theta}{\operatorname{argmin}}\, d(x,p) \in C_i$$

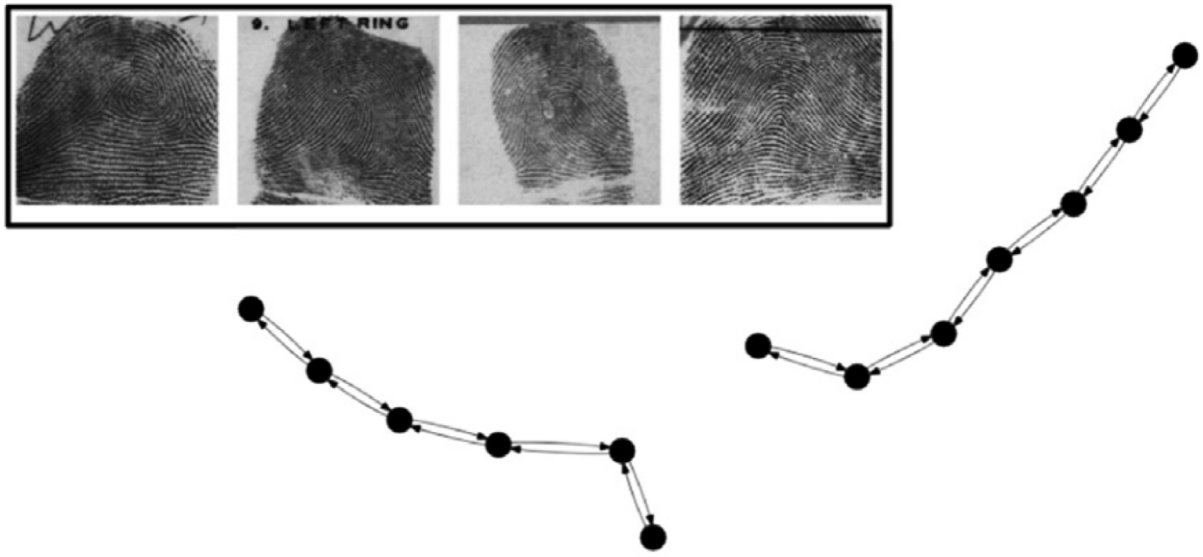
Letter Graphs

- Nodes: line endpoints, labeled with Cartesian coordinates (x,y)
- Edges: undirected, unlabeled lines
- Euclidean cost function:
  - node substitution: $||(x_1,y_1) - (x_2,y_2)||$
  - node deletion/insertion: C_n
  - edge deletion/insertion: C_e
  - C_n and C_e are cost function parameters that are optimized during cross-validation.



Fingerprint Graphs

- Nodes: unlabeled keypoints
- Edges: directed edges labeled with their angle
- Cost function:
  - edge substitution: $\min(|\phi_1-\phi_2|, 2\pi - |\phi_1-\phi_2|)$
  - node deletion/insertion: C_n
  - edge deletion/insertion: C_e
  - This cost function is invariant to translation.



Molecular Compounds

- Nodes: atoms labeled with their symbol
- Edges: unlabeled, undirected links
- Dirac cost function:
  - node substitution: $2 \cdot C_n$ if symbols are not equal, 0 otherwise
  - node deletion/insertion: C_n
  - edge deletion/insertion: C_e