Graph Matching Methods

Presented by :
Romain Raveaux

Jean-Yves Ramel – Romain Raveaux
Laboratoire Informatique de Tours - FRANCE

Content

Part 1

- Forewords and preliminary tools

Aim of the talk

- Graph based methods in Pattern Recognition

- What we won't talk about :
  - Graph for image segmentation (Normalized Cut Graph, ...)
  - Graph for knowledge representation (Ontology, RDF, ...)

Introduction

- Terminology and notation

Definition and notation of a graph:

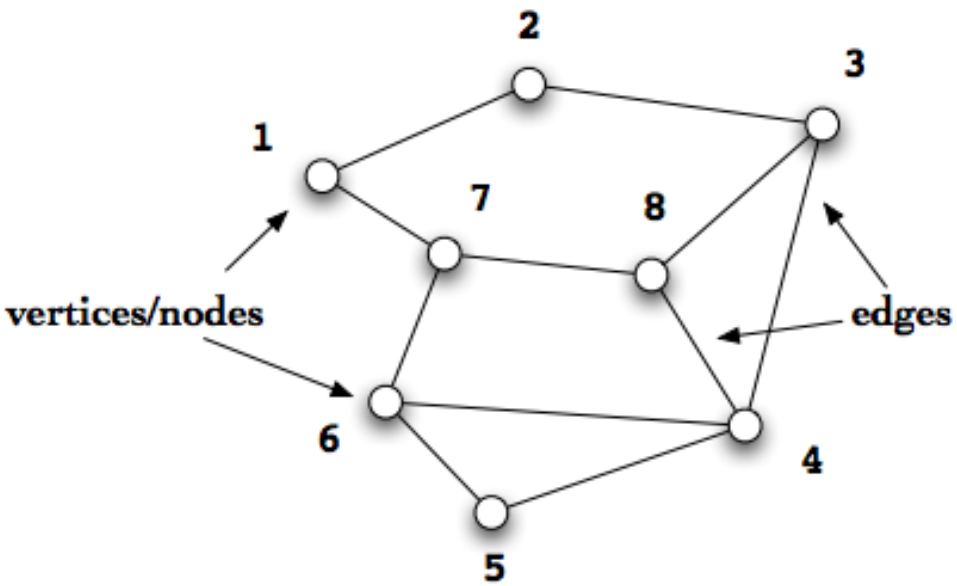
Definition

Let L_V and L_E denote the set of node and edge labels, respectively. A labeled graph G is a 4-tuple $G = (V, E, \mu, \xi)$, where

- V is the set of nodes,
- $E \subseteq V \times V$ is the set of edges
- $\mu : V \rightarrow L_V$ is a function assigning labels to the nodes, and
- $\xi : E \rightarrow L_E$ is a function assigning labels to the edges.

- Let $G_1 = (V_1, E_1, \mu_1, \xi_1)$ be the source graph
- And $G_2 = (V_2, E_2, \mu_2, \xi_2)$ the target graph
- With $V_1 = (u_1, ..., u_n)$ and $V_2 = (v_1, ..., v_m)$ respectively

Adjacency matrix

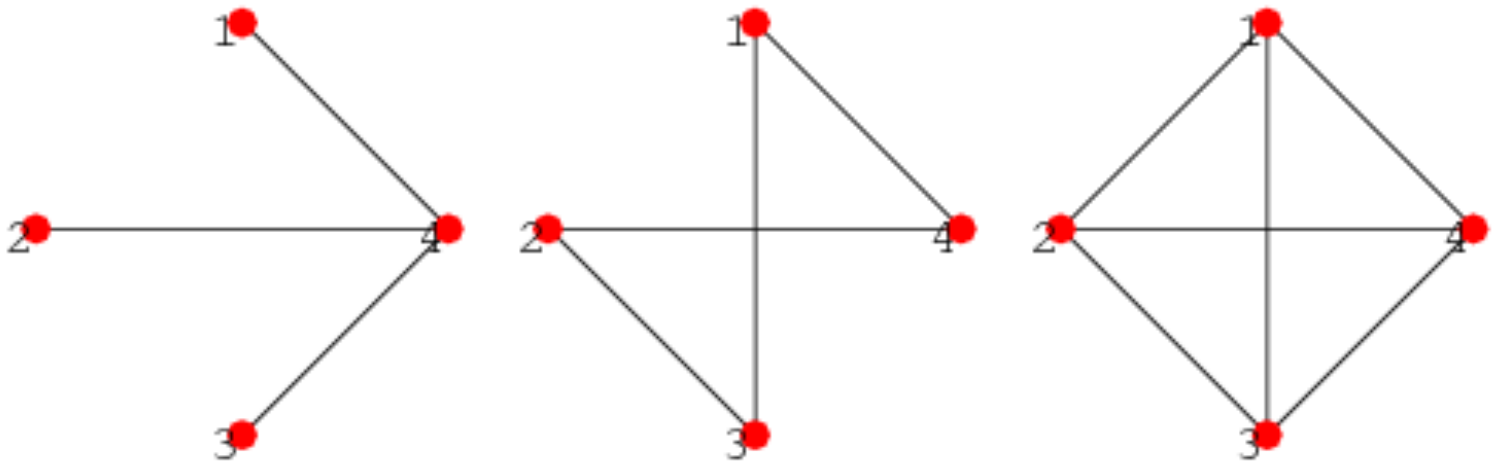


$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Undirected Graph G(V, E)

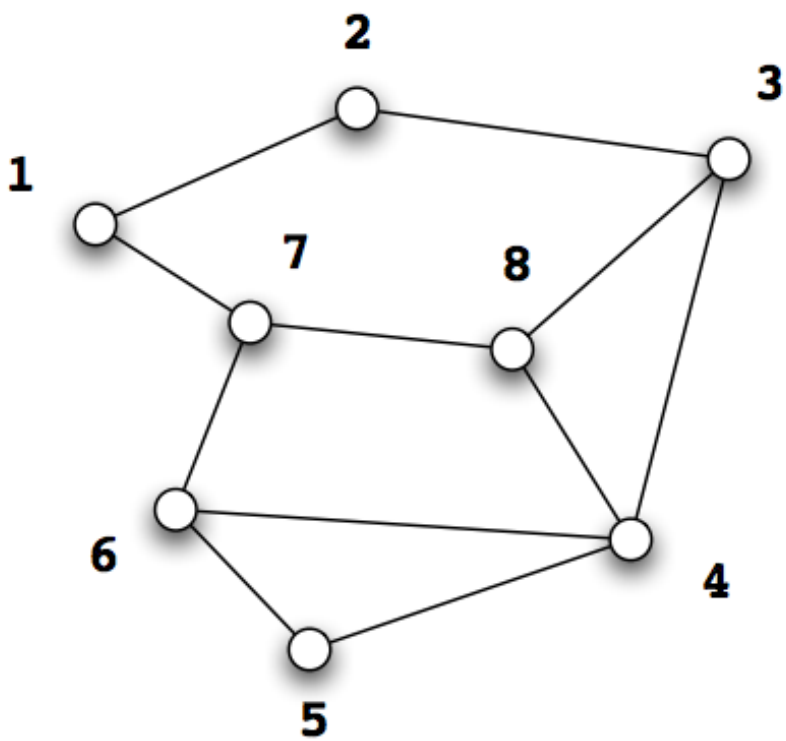
sub-matrix of A = a subgraph of G

Incidence Matrix



$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

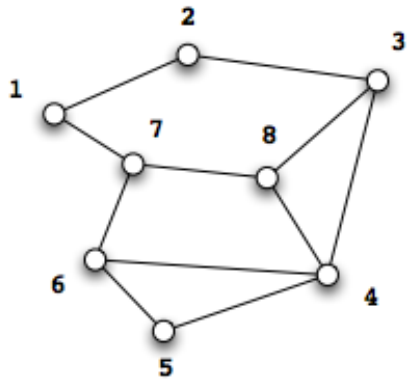
Degree matrix



$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Normalized Adjacency matrix $\tilde{A} = D^{-1}A$ is a stochastic matrix (each row sums to one)

Laplacian Matrix



$$L = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 4 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & -1 & 0 & 0 & -1 & 3 \end{bmatrix}$$

$$L = D - A$$

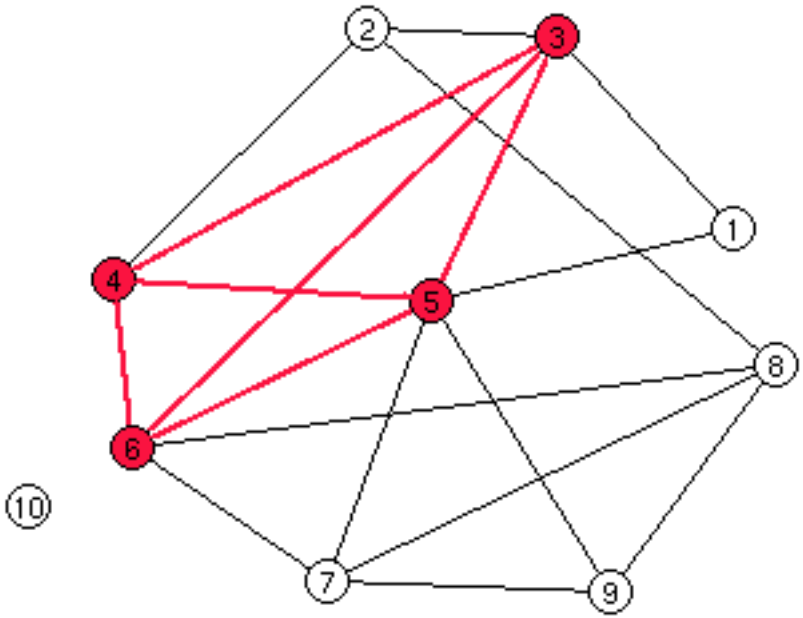
Normalized version

$$\tilde{L} = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$$

Spectrum bounded between 0 and 2

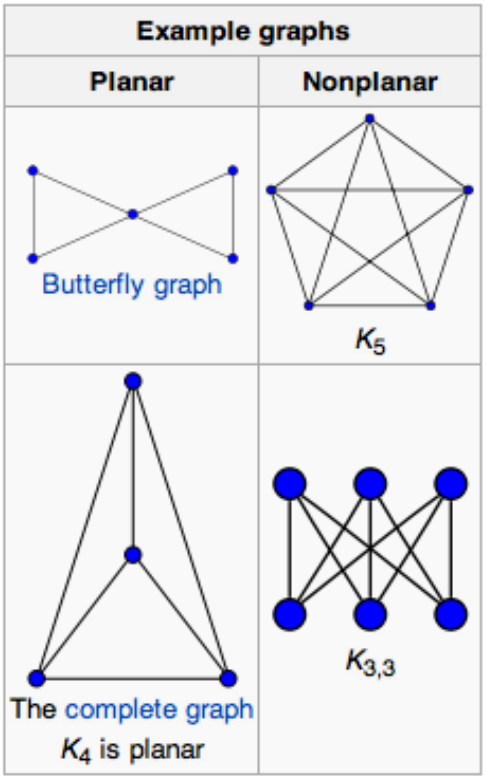
Clique

- Clique of size n.



Wrong beliefs

- « In computer vision, all graphs are planars. »
- Planar :
  - « In graph theory, a planar graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints.» Wikipedia

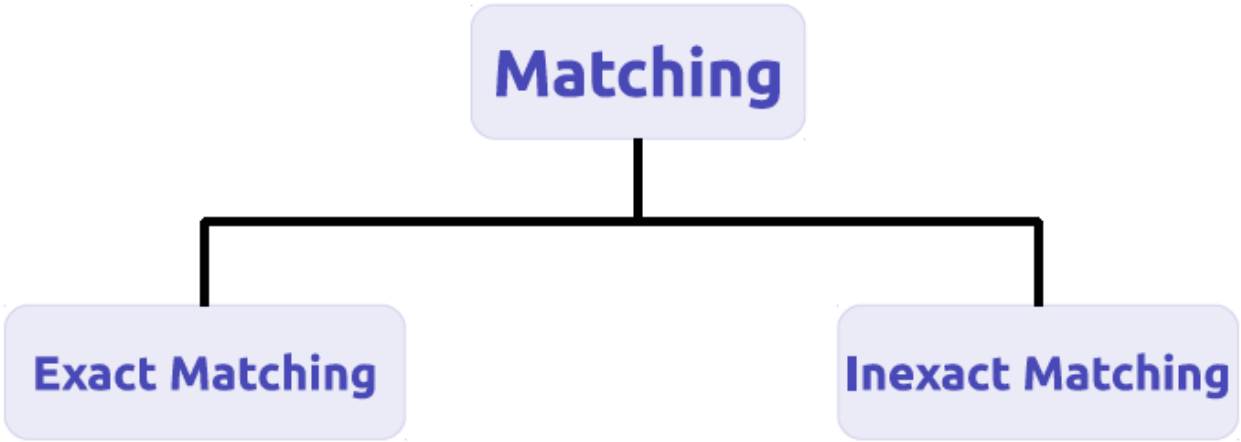


Part 2

- Graph Comparison
  - Section 1 : Graph Matching

Graph comparison (Structural Pattern Recognition)

- Overview

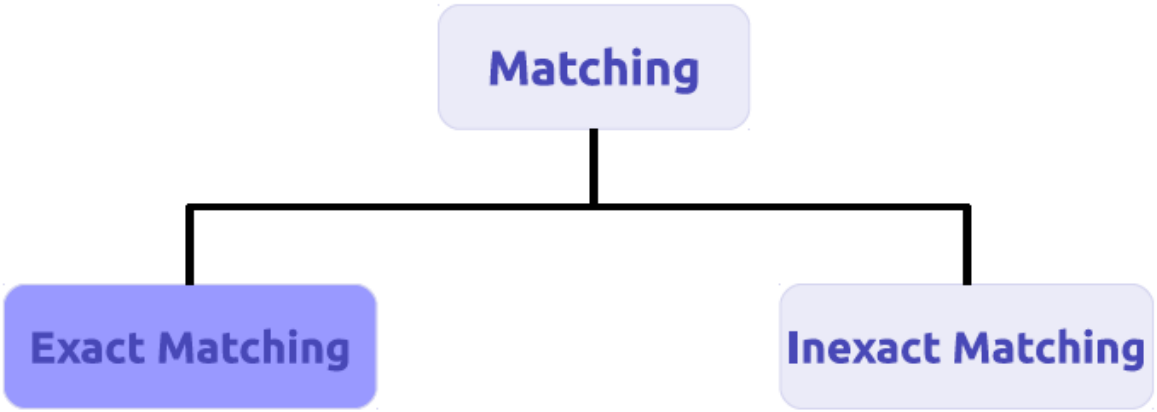


Goals

Exact Matching: Find an *edge preserving* mapping.

Inexact Matching: Find a mapping that minimizes matching costs.

Overview



Exact Matching Complexity

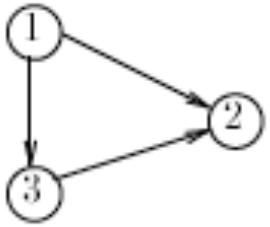
Graph Isomorphism : In NP, but unknown if P or NP complete.

Subgraph Isomorphism, Monomorphism, MCS... : NP complete.

There exists algorithms for special graphs with polynomial runtime.

Overview

model B



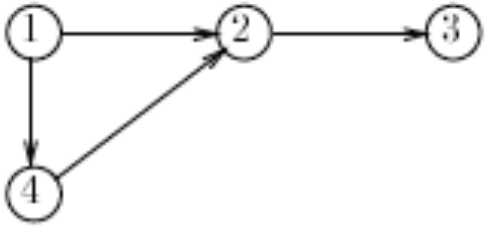
subgraph isomorphism:

$1 \to 1$

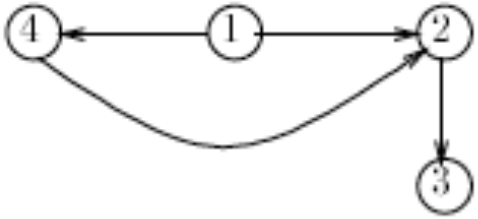
$2 \to 2$

$3 \to 4$

input graph



model C



isomorphism:

$1 \to 1$

$2 \to 2$

$3 \to 3$

$4 \to 4$

Overview

- Exact graph isomorphism (GI)

- <u>Formalism</u> :
  Graphs $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$
  Iff it exists a bijection $f : V_1 \rightarrow V_2$
  $\forall a,b \in V_1$,
  $$(a,b) \in E_1 \Leftrightarrow (f(a),f(b)) \in E_2$$

- <u>Complexity :</u>
  - NP-hard problem in general
    (GI $\in$ NP, GI $\in$ P, GI $\in$ NP-complet)
  - Certain classes of graphs GI $\in$ P (Polynomial algorithm)

Complexity

Type of graph	Complexity	Reference
Planar	$O(n)$	[Hopcroft, Tarjan, 1972] [Hopcroft, Wong, 1974]
General	$Exp(n^{1/2+O(1)})$	[Babai, Luks, 1983]

Graph Isomorphism

- Node invariant
- Canonical adjacency matrix
  - Automorphism

Graph Isomorphism

- Node invariant
  - Used in most approaches
  - The number i(v) associated to a given node such as if an isomorphism mapps v and v' then :

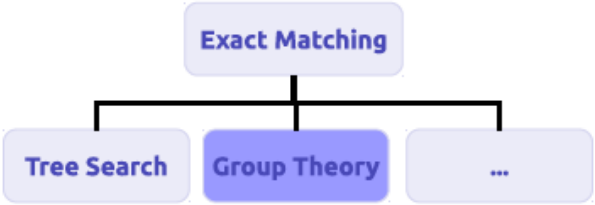
$$i(v) = i(v')$$

    - *degree*
    - *twopaths* : # Nodes at distance 2 from v
    - *k-cliques* : # Cliques of size k containing v
    - *independant k-sets* : # Independent set of size k containing v
    - *distances* : # Nodes at every distance 1,…,n from v

  - **Heavy cost** (excepted degree) : To solve the isomorphism is often faster than to compute every invariant

Graph Isomorphism

Group theory approach

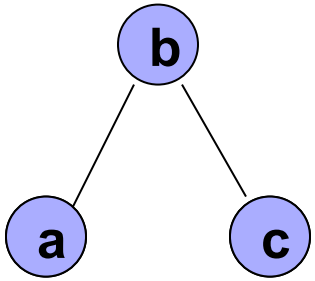


McKay's Nauty

- Nauty - No automorphisms, yes?
- Application for isomorphism only
- It uses the property that the canonical labeling for isomorph graphs is identical.
- It constructs the automorphism group of each of the input graphs and derives a canonical labeling

Graph Isomorphism

- Canonical label (C(G)):
  - *Example 1* :
    - Compute all automorphisms of G
    - Automorphism search equal isomorphism search



	a	b	c
a	0	1	0
b	1	0	1
c	0	1	0

Automorphism c→a, a →b, b →c

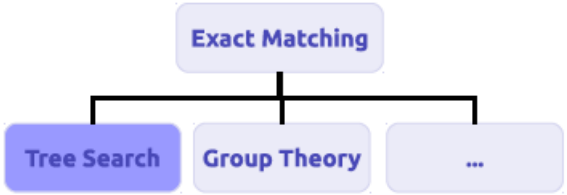
	c	a	b
c	0	0	1
a	0	0	1
b	1	1	0

	a	c	b
a	0	0	1
c	0	0	1
b	1	1	0

C(G) = 001001110

SubGraph Isomorphism

Tree-Search approach

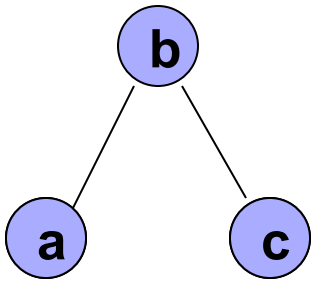


SubGraph Isomorphism

- Approach: Exhaustive Tree search [Corneil, Gotlieb,70]

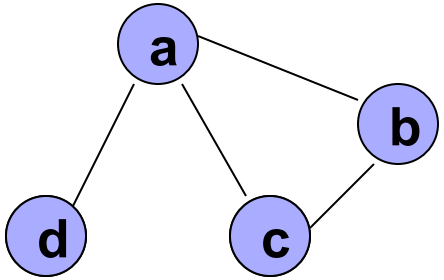
SubGraph Isomorphism

- G1



	a	b	c
a	0	1	0
b	1	0	1
c	0	1	0

G2



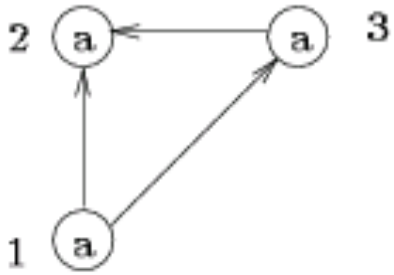
	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	0
d	1	0	0	0

	a	b	c
a	0	1	0
b	1	0	1
c	0	1	0

	?	?	?	?
?	0	1	0	?
?	1	0	1	?
?	0	1	0	?
?	?	?	?	?

SubGraph Isomorphism

- Approach: Tree search with backtracking [Corneil, Gotlieb,70]
  - Row-Column representation of the adjacency matrix



	1	2	3
1	a	1	1
2	0	a	0
3	0	1	a

$\Rightarrow$

a_1

a

a_2

	1
0	a

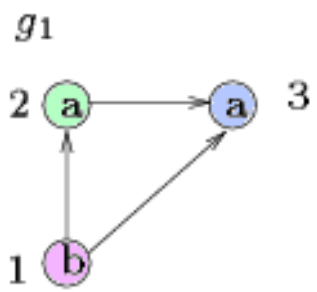
a_3

		1
		0
0	1	a

$$a_1 = (a)$$
$$a_2 = (1, a, 0)$$
$$a_3 = (1, 0, a, 1, 0)$$

SubGraph Isomorphism

- <u>Method</u> :
  - Build all matrix permutations of G.



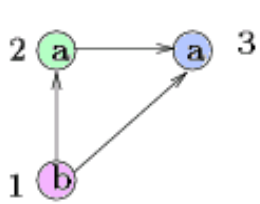
$$A = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline & b & 1 & 1 \\ & 0 & a & 1 \\ & 0 & 0 & a \end{array} \qquad B = \begin{array}{c|ccc} & 1 & 3 & 2 \\ \hline & b & 1 & 1 \\ & 0 & a & 0 \\ & 0 & 1 & a \end{array} \qquad C = \begin{array}{c|ccc} & 2 & 1 & 3 \\ \hline & a & 0 & 1 \\ & 1 & b & 1 \\ & 0 & 0 & a \end{array}$$

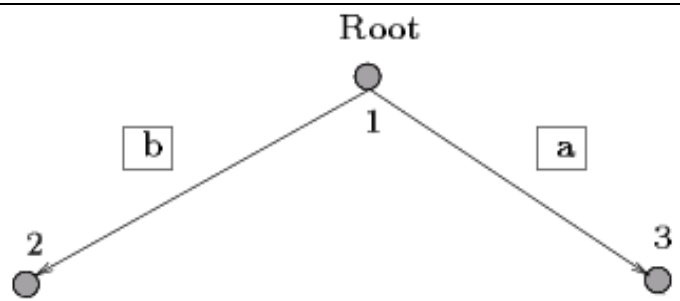
$$D = \begin{array}{c|ccc} & 2 & 3 & 1 \\ \hline & a & 1 & 0 \\ & 0 & a & 0 \\ & 1 & 1 & b \end{array} \qquad E = \begin{array}{c|ccc} & 3 & 1 & 2 \\ \hline & a & 0 & 0 \\ & 1 & b & 1 \\ & 1 & 0 & a \end{array} \qquad F = \begin{array}{c|ccc} & 3 & 2 & 1 \\ \hline & a & 0 & 0 \\ & 1 & a & 0 \\ & 1 & 1 & b \end{array}$$

SubGraph Isomorphism

- Method :
  - Build all matrix permutations of G_1
  - Build the decision tree of all matrix permutations

SubGraph Isomorphism





A

1	2	3
b	1	1
0	a	1
0	0	a

B

1	3	2
b	1	1
0	a	0
0	1	a

C

2	1	3
a	0	1
1	b	1
0	0	a

D

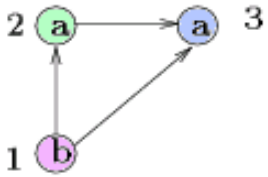
2	3	1
a	1	0
0	a	0
1	1	b

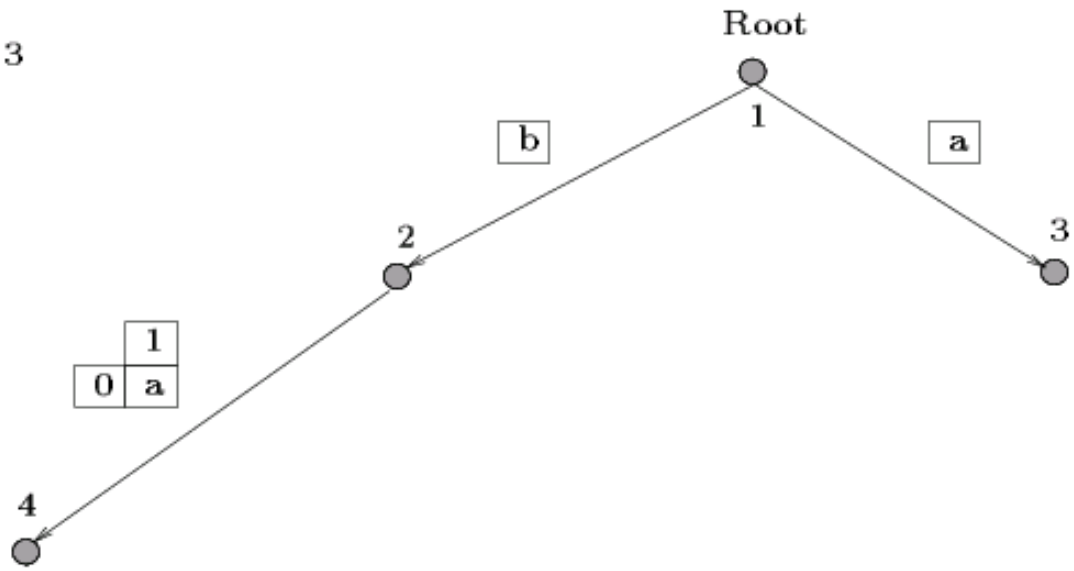
E

3	1	2
a	0	0
1	b	1
1	0	a

F

3	2	1
a	0	0
1	a	0
1	1	b





A

	1	2	3
	b	1	1
	0	a	1
	0	0	a

B

	1	3	2
	b	1	1
	0	a	0
	0	1	a

C

	2	1	3
	a	0	1
	1	b	1
	0	0	a

D

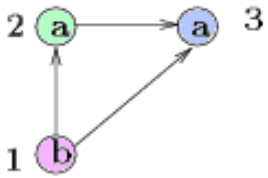
	2	3	1
	a	1	0
	0	a	0
	1	1	b

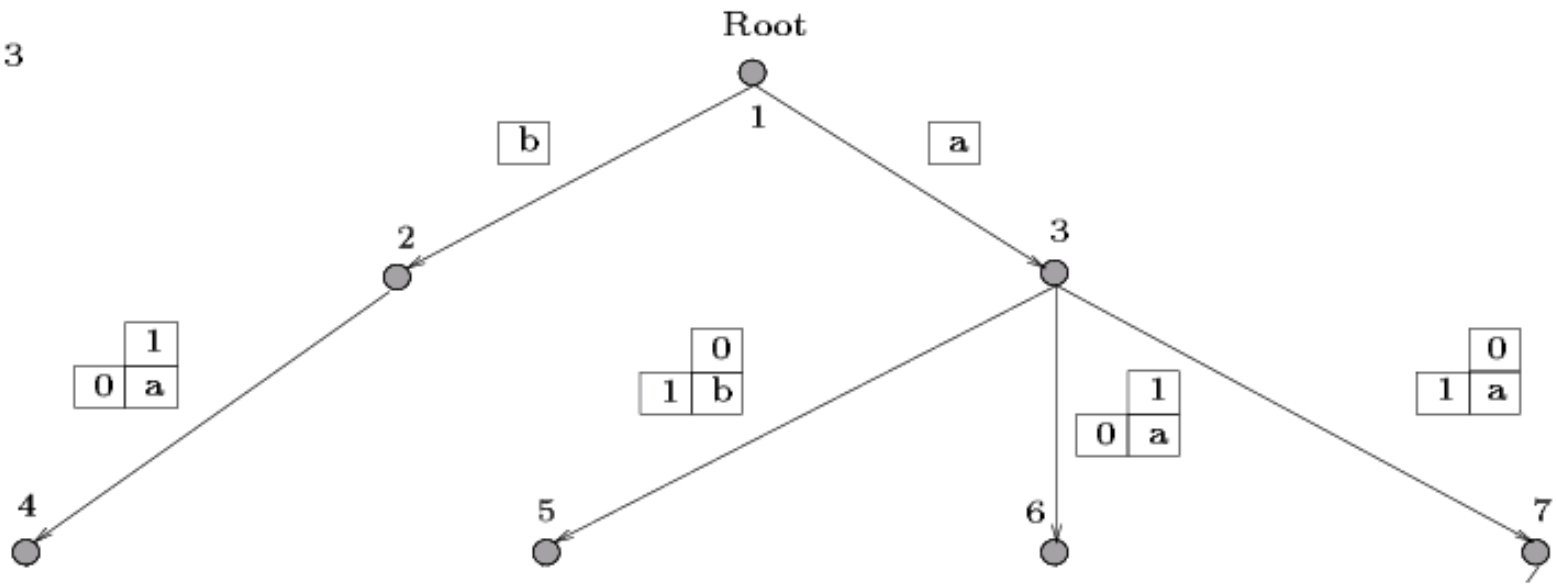
E

	3	1	2
	a	0	0
	1	b	1
	1	0	a

F

	3	2	1
	a	0	0
	1	a	0
	1	1	b





	1	2	3
	b	1	1
A	0	a	1
	0	0	a

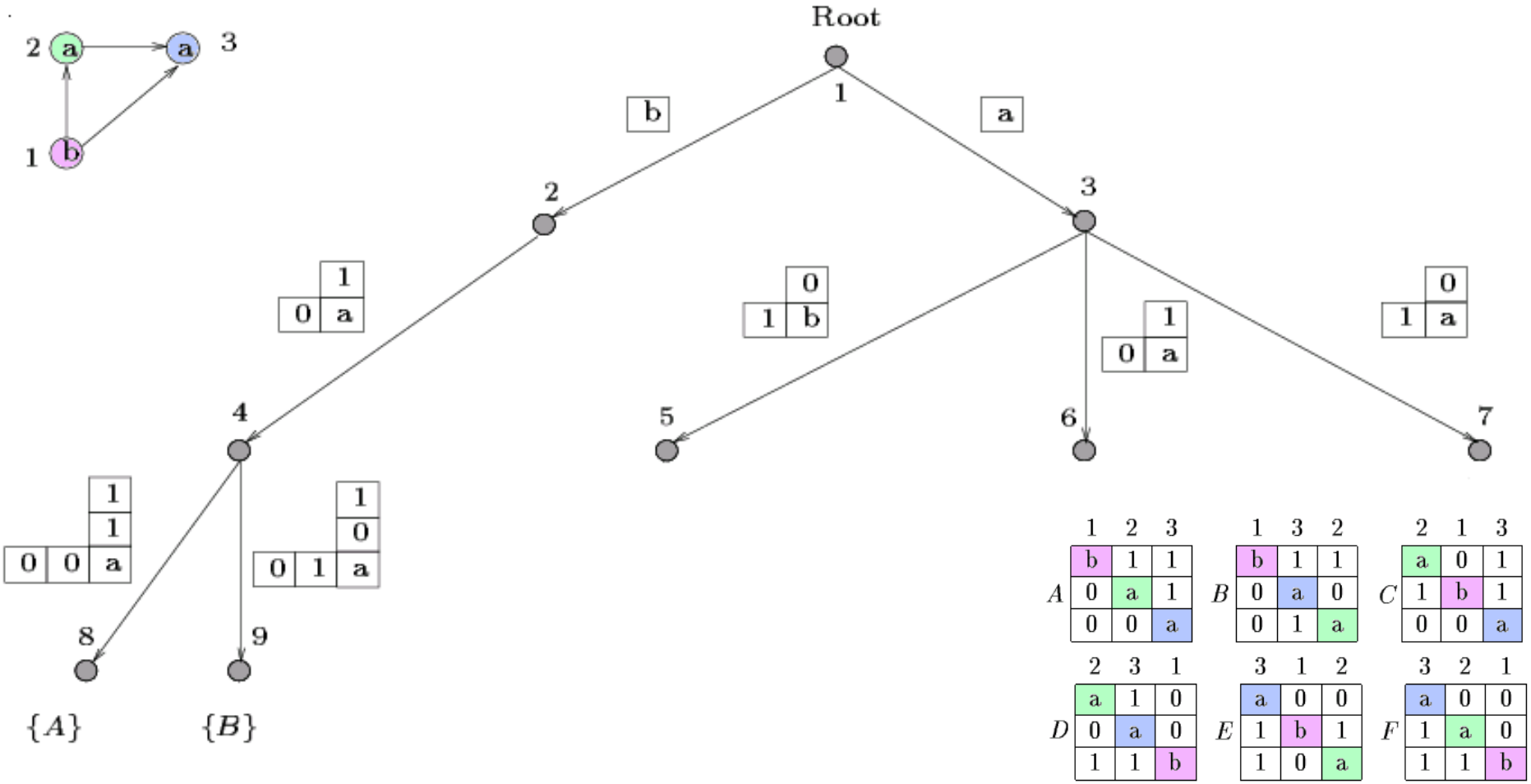
	1	3	2
	b	1	1
B	0	a	0
	0	1	a

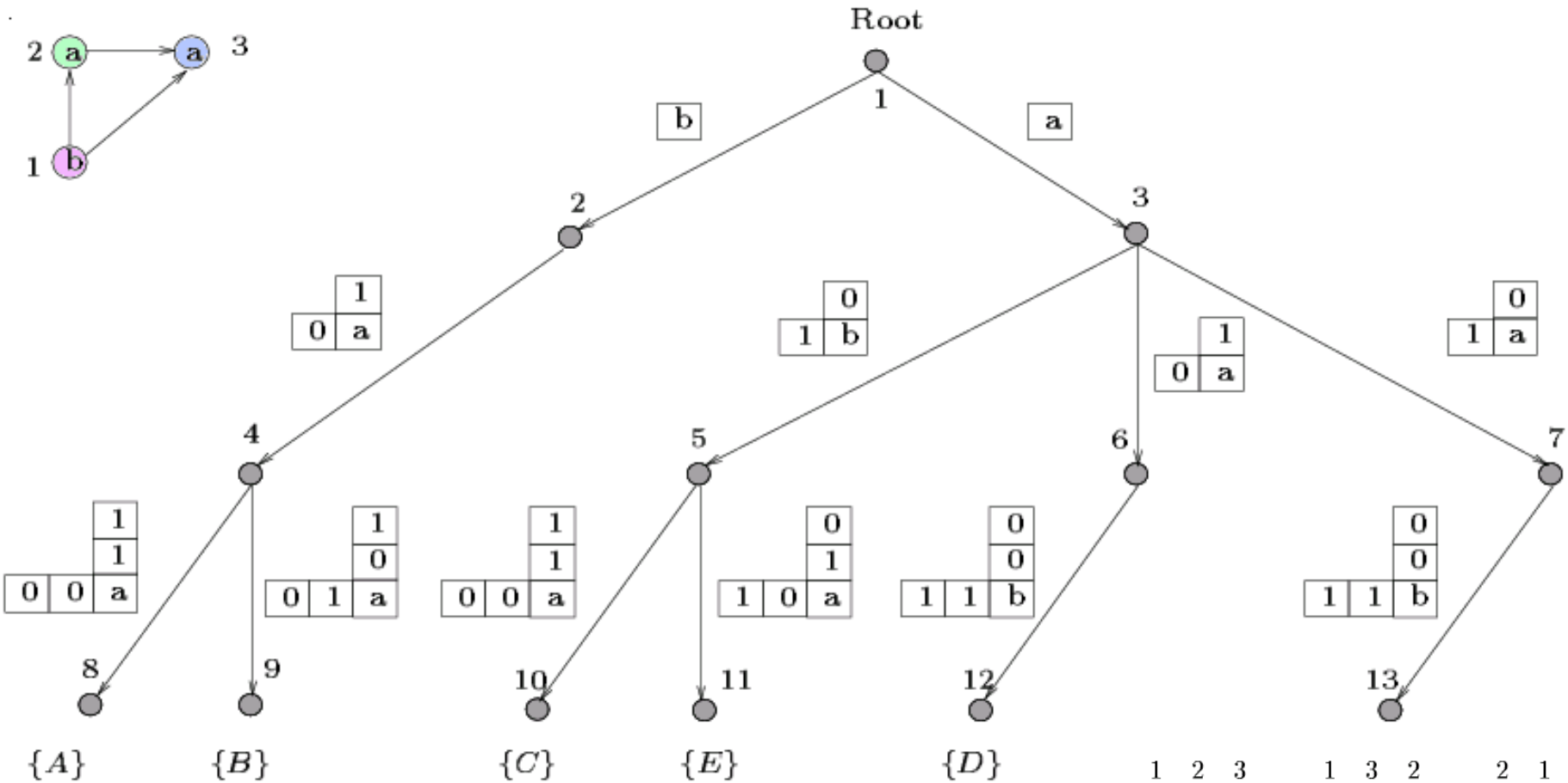
	2	1	3
	a	0	1
C	1	b	1
	0	0	a

	2	3	1
	a	1	0
D	0	a	0
	1	1	b

	3	1	2
	a	0	0
E	1	b	1
	1	0	a

	3	2	1
	a	0	0
F	1	a	0
	1	1	b


Root
1
b
a
2
3
4
5
6
7
8
9
{A}
{B}



	1	2	3
	b	1	1
A	0	a	1
	0	0	a

	1	3	2
	b	1	1
B	0	a	0
	0	1	a

	2	1	3
	a	0	1
C	1	b	1
	0	0	a

	2	3	1
	a	1	0
D	0	a	0
	1	1	b

	3	1	2
	a	0	0
E	1	b	1
	1	0	a

	3	2	1
	a	0	0
F	1	a	0
	1	1	b

Tree construction

- <u>Method</u> :
  - Build all matrix permutations of G_1
  - Build the decision tree of all matrix permutations

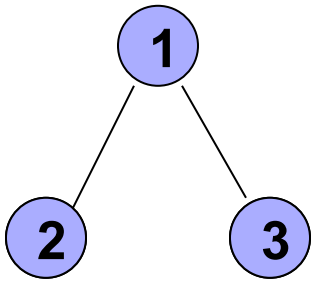
Non-directed and unlabel graph : $O(3^n)$

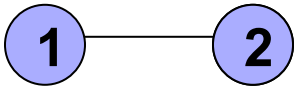
n : #nodes G_1

Isomoprhism search into the tree

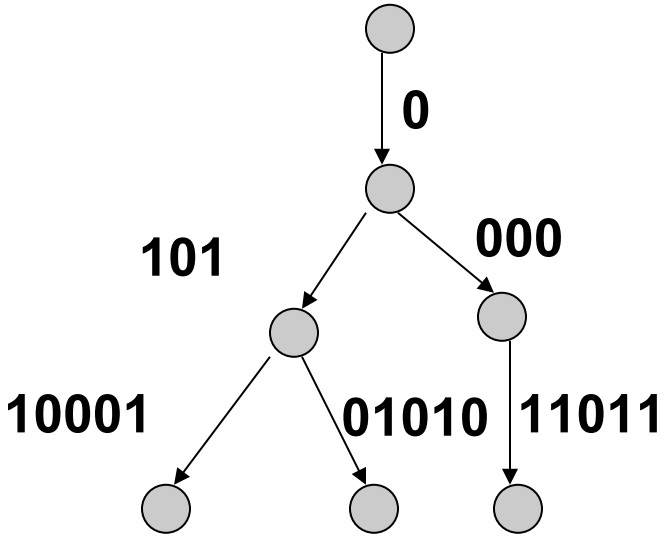
- Isomoprhism search of G_2 with a subgraph of G_1
  → Decision automate

Isomorphism search

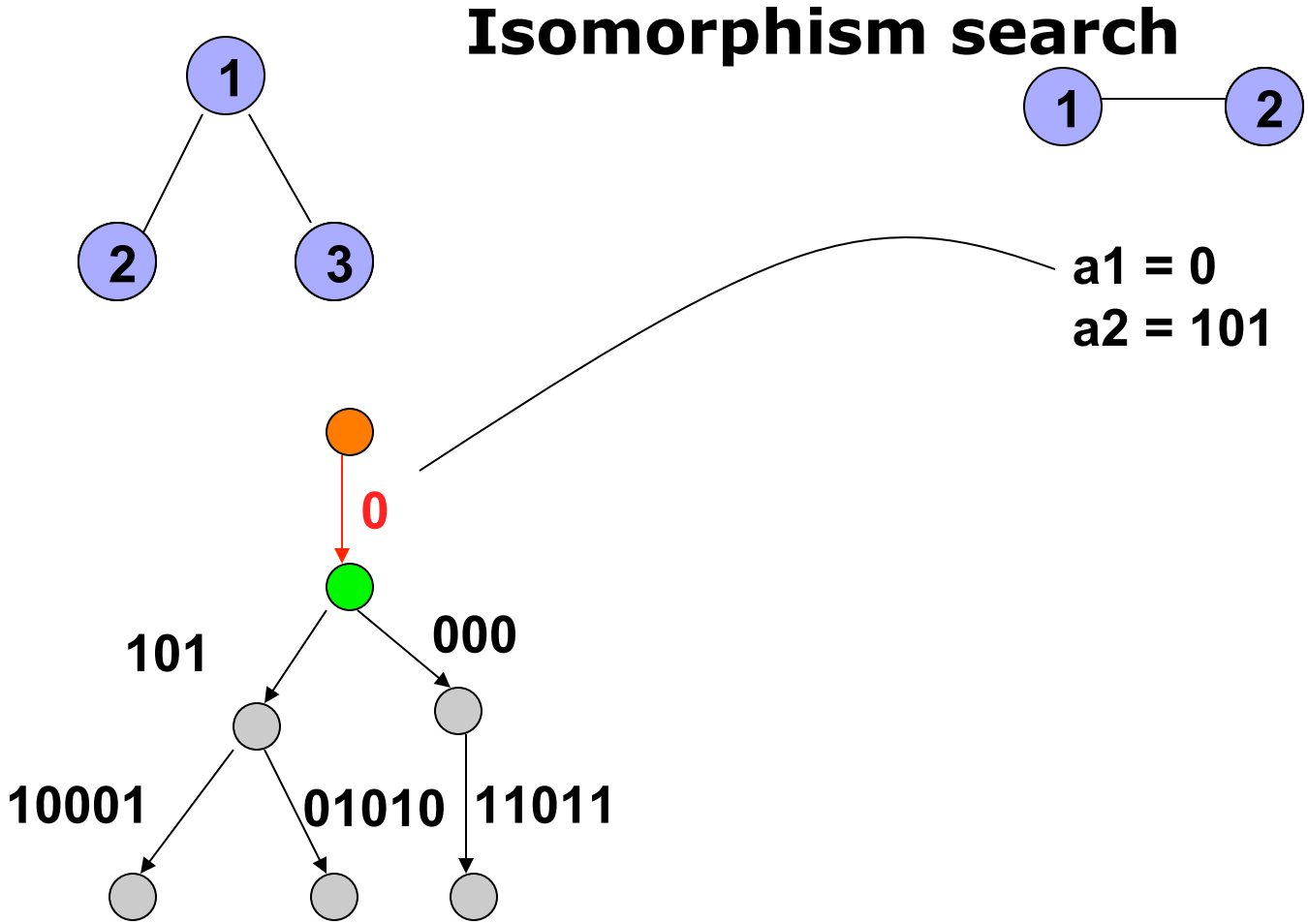




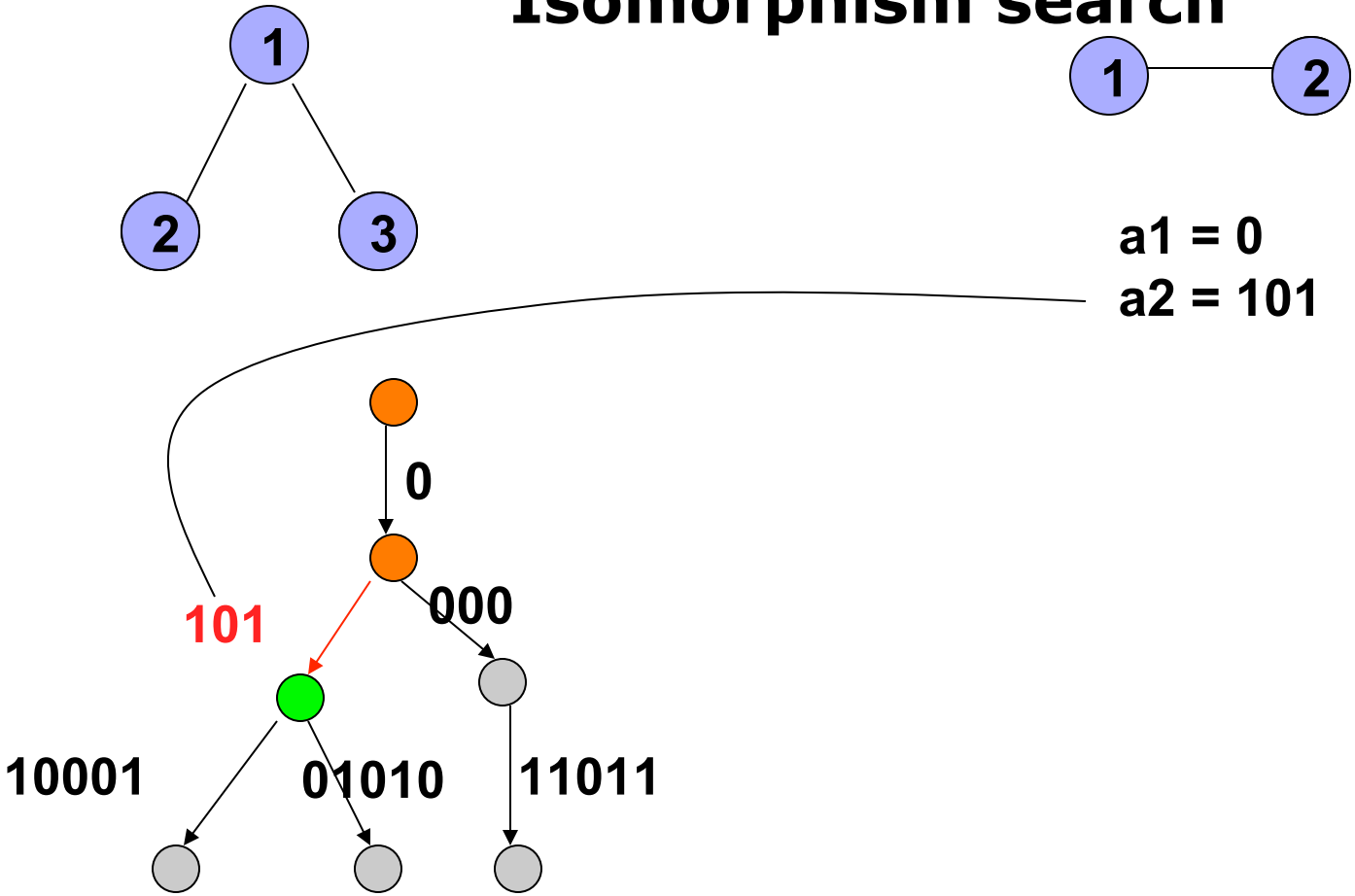
a1 = 0
a2 =
101



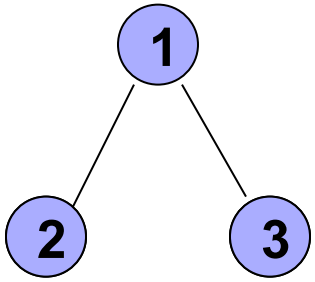
Isomorphism search

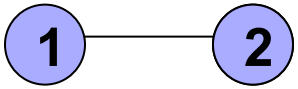


Isomorphism search

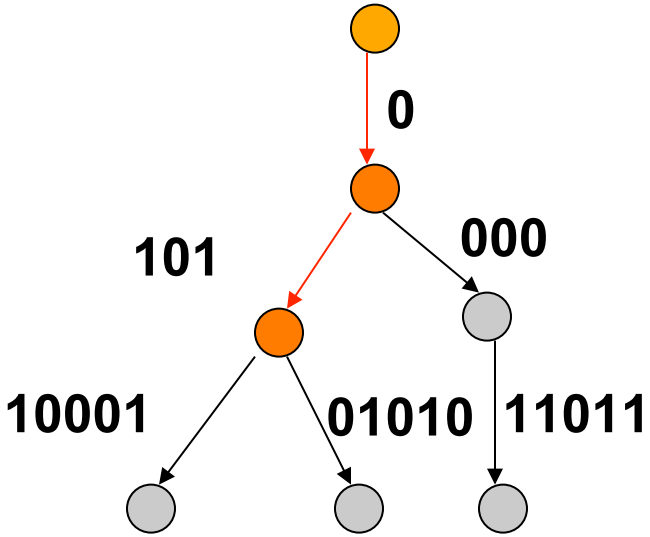


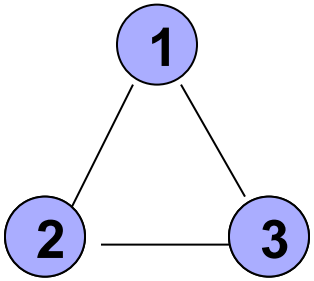
Isomorphism search





a1 = 0
a2 = 101





a1 = 0
a2 = 101
a3 = 11011

Isomorphism search

- Isomorphism search of G_2 with a subgraph from G_1

  Decision automate :

  Un-directed, unlabel graphs : $O(m^2)$

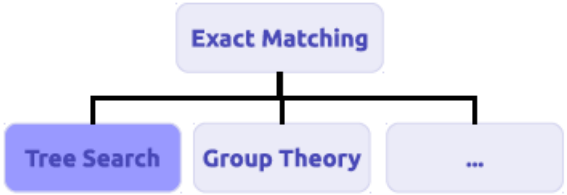
  m : # nodes G_2

SubGraph Isomorphism

- Exact Matching
  - Ullmann's Algorithm [J.R. Ullmann 1976]
  - Tree-Search algorithm (Depth-Search-First)
  - Uses adjacency matrices and additional constraints for matching and pruning.
  - Application for graph isomorphism, subgraph isomorphism and monomorphism, also for MCS problem

SubGraph Isomorphism

Tree-Search approach



Basic Idea

- Iteratively expand partial match by adding new pairs of matched nodes.
- The pair is chosen using some necessary conditions.
- Prune unfruitful search paths.
- If no further vertex pairs may be added due to constraint, undo last additions (backtracking)
- Algorithm stops if match has been found or all matchings that satisfy the constraints has been tried.

Ullmann's Algorithm

Given: Two graphs $G_A(V_A, E_A)$ and $G_B(V_B, E_B)$ and their adjacency matrices: A and B

Idea: $n = |V_a|$, $m = |V_b|$, $n \times m$ permutation matrix M with following form:

- M contains only '0' and '1'
- Exact one '1' in each row
- Not more than one '1' in each column

Permutate adjacency matrix B by multiplying it with M, and compare adjacency.

Ullmann's Algorithm

$M \times B$: Move row j to row i $\forall M_{ij} = 1$

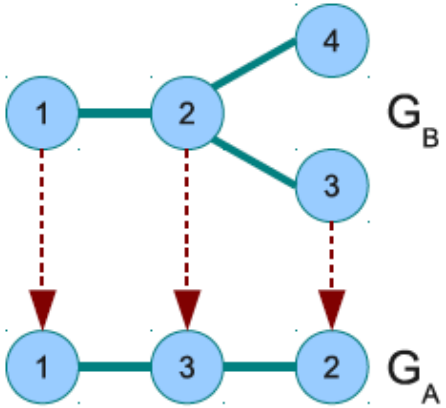
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$M = M^T$ $\quad$ $B = B^T$

$(MB)^T$: Move column j to column i

$M(MB)^T$: Move column j to column i and row j to row i

Ullmann's Algorithm



0	1	0	0
1	0	1	1
0	1	0	0
0	1	0	0

$B=B^T$

1	0	0	0
0	0	1	0
0	1	0	0

M

$$M(MB)^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = C$$

Ullmann's Algorithm

Creating pairs of nodes by exchanging rows and columns (renaming).

Adjacency condition

Let $C = M(MB)^T$,
A is a (subgraph-) isomorphism iff

$$A_{ij} = 1 \Rightarrow C_{ij} = 1 \forall i, j$$

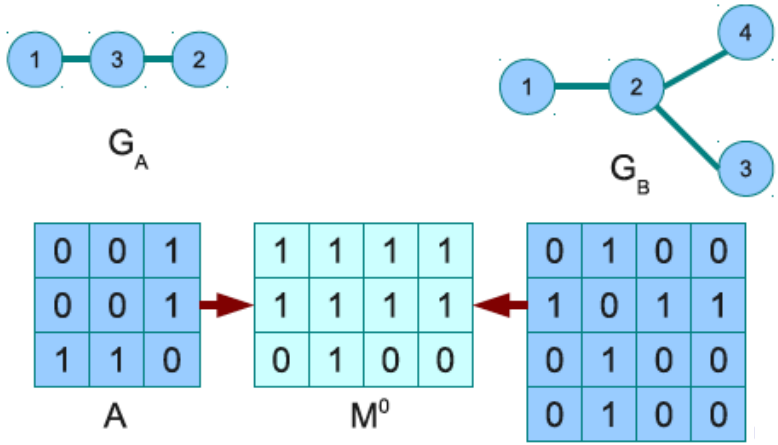
How do we get M?

Ullmann's Algorithm

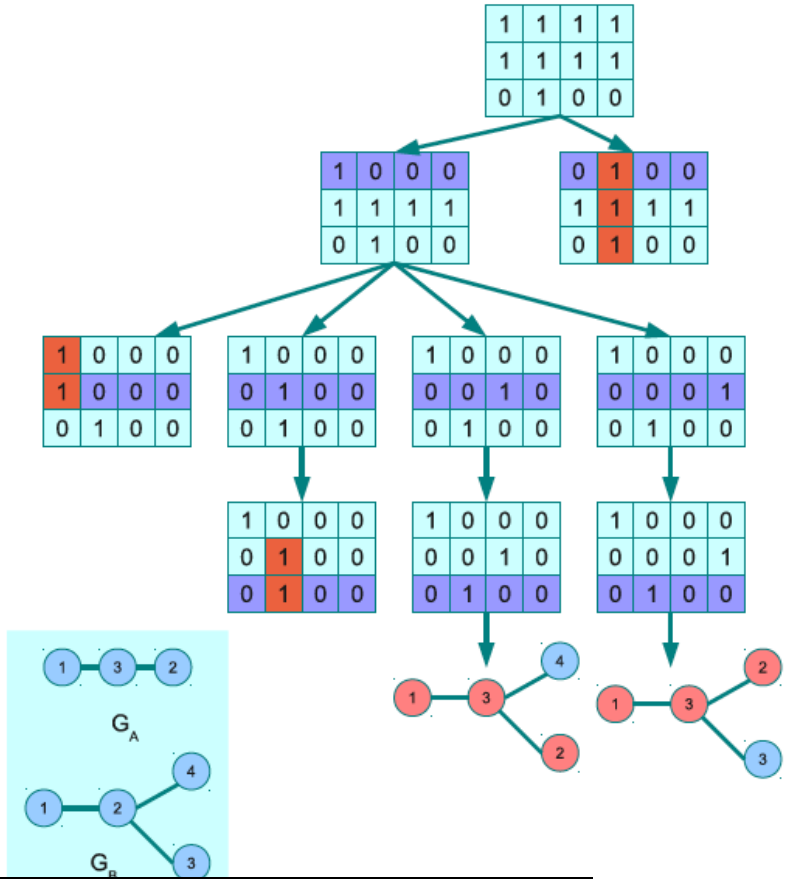
- Build Startmatrix M^0 by setting all values to 1 (allow all permutations)
- Set values to 0 for all M^0_{ij} where $deg(B_j) < deg(A_i)$ (remove impossible permutations)

$$M^0_{ij} = \begin{cases} 1 & if \quad deg(B_j) \geq deg(A_i) \\ 0 & \text{otherwise} \end{cases} \quad , \forall i, j$$

- Generate systematically permutation matrices M^d.



Ullmann's Algorithm



Ullmann's Algorithm

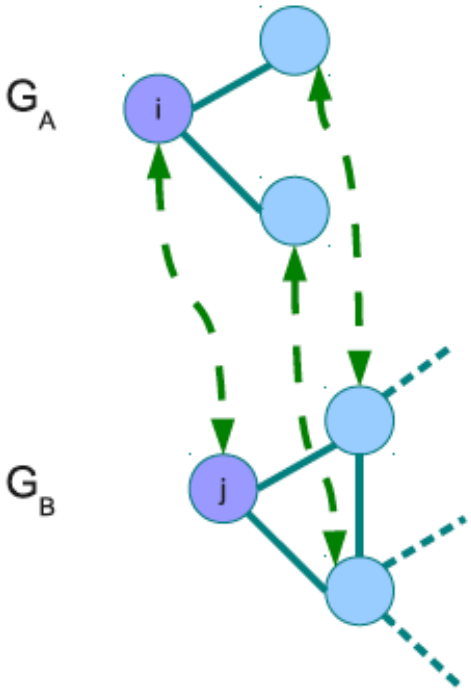
Refinement Procedure:

For all neighbours in A there must be proper neighbours in B.

Formally:

$$\forall k(A_{ik} = 1 \Rightarrow \exists p(M_{kp}B_{pj} = 1))$$

Set $M^d_{ij} = 0$ where conditions are not complied.



SubGraph Isomorphism

- Improvements :
  - Ullman $O(m^2n^2)$ [Ullman, 76]
  - Forward-checking and looking-ahead [Haralick, Elliot, 80]
  - Discrete Relaxation [Kim, Kack, 91]
  - Maximal clique detection [Blake, 94]

The Planar Graph Case (1)

A Polynomial Algorithm for Subisomorphism of Open Plane Graphs [Damiand et al., 2009].

- Searching for a pattern in a plane graph
- To model plane graphs with 2-dimensional combinatorial maps
  - data structures for modelling the topology of a subdivision of a plane into nodes, edges and faces.
- A polynomial algorithm for this problem

Refs:

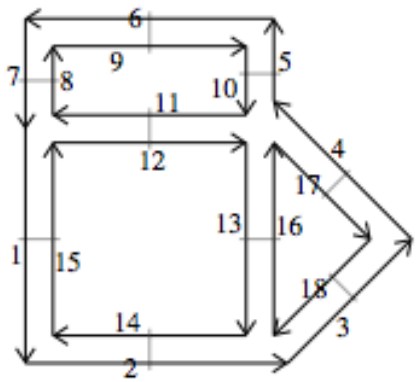
- http://dtai.cs.kuleuven.be/ilp-mlg-srl/papers/MLG09-11.pdf

-http://hal-ujm.ccsd.cnrs.fr/docs/00/38/97/63/PDF/map-isomorphism.pdf

The Planar Graph Case (2)

Planar Graphs may be drawn in the plane, but even more specically just one of the possible planar embeddings is

relevant as it actually models the image topology, that is, the order in which faces are encountered when turning around a node.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
β_1	2	3	4	5	6	7	1	9	10	11	8	13	14	15	12	17	18	16
β_2	15	14	18	17	10	9	8	7	6	5	12	11	16	2	1	13	4	3

Fig. 2. Combinatorial map example. Darts are represented by numbered black segments. Two darts 1-sewn are drawn consecutively, and two darts 2-sewn are concurrently drawn and in reverse orientation, with little grey segment between the two darts.

Error-tolerant isomorphism

- Exact graph matching is useless in many computer vision applications
- Concerning graph matching under noise and distortion
- The matching incorporates an error model to identify the distortions which make one graph a distorted version of the other

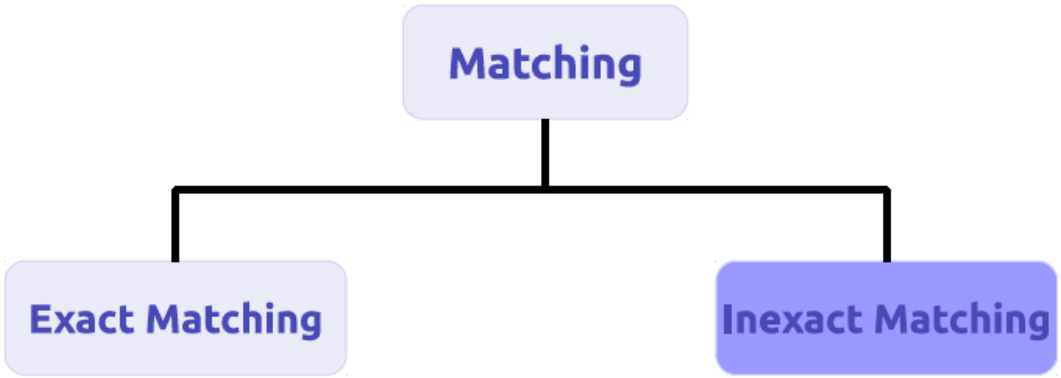
Problems for real world applications

- Error-tolerant
- To measure the similarity of two graphs.
- Runtime may grow exponentially with number of nodes
- This is an enormous problem for large datasets of graphs

Wanted: Polynomial-time similarity measure for graphs

Inexact graph matching

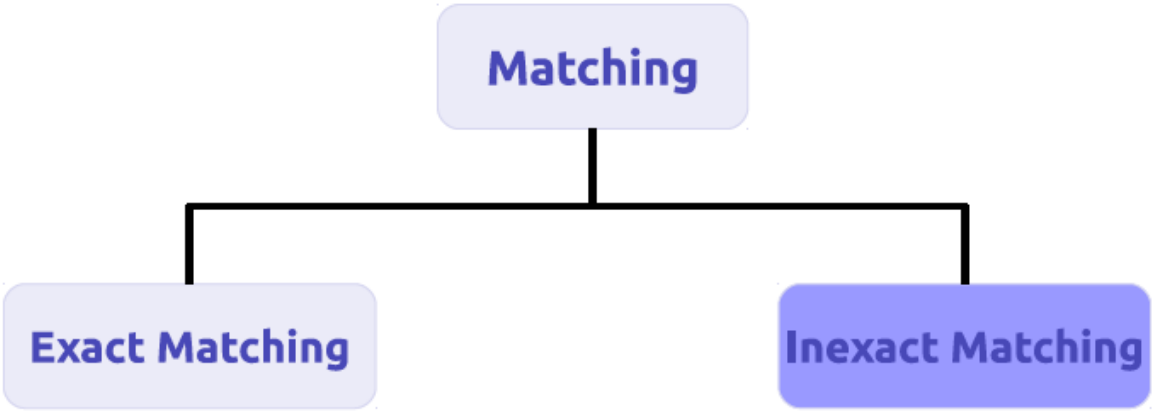
Inexact Matching



Reasons for using inexact matching

- Deformations in graphs (eg. noise, variability of patterns, nondeterministic elements...)
- Exact matching too expensive

Inexact graph matching

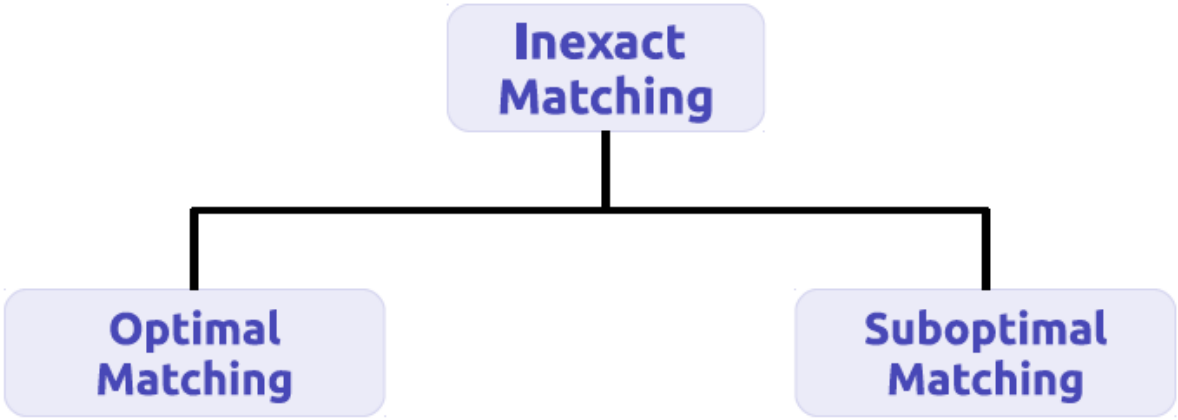


How

- Relax constraints - no edge preserving.
- Penalize graph differences.
- Find a mapping that minimizes matching costs.

Inexact graph matching

Optimal and suboptimal inexact matching algorithms



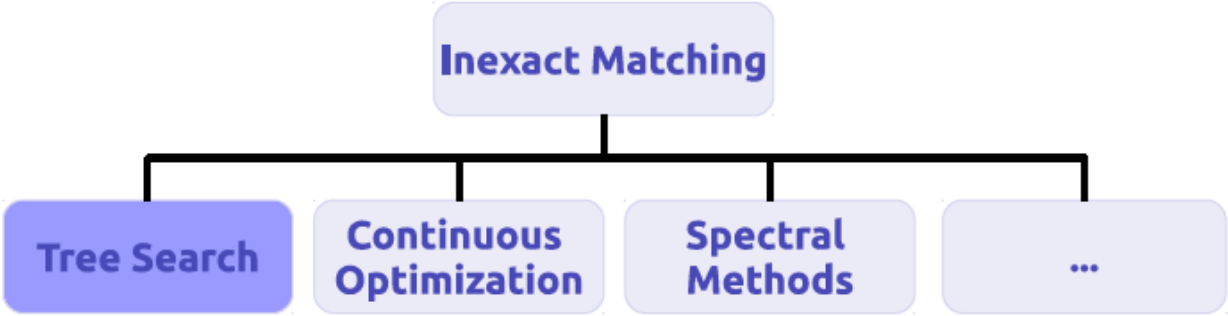
Runtime

Optimal Inexact Matching : More expensive than exact algorithms.

Suboptimal Inexact Matching : Usually polynomial matching time.

Inexact graph matching

Tree-Search approaches

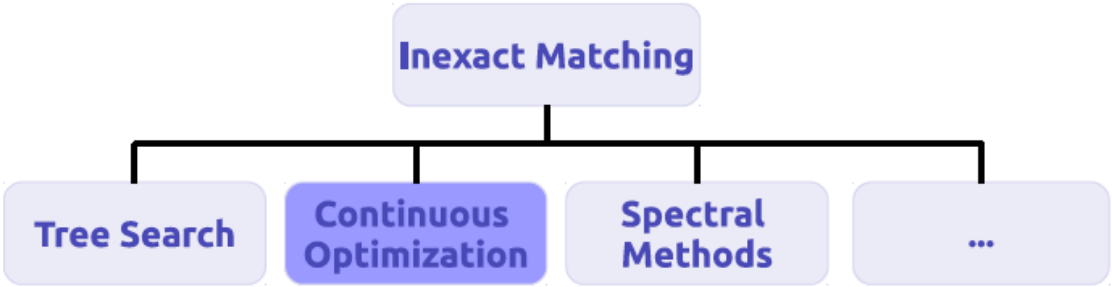


Tree Search with Backtracking

- Search guided by cost function.
- Heuristic estimate matching cost for remaining nodes.
- Prune unfruitful paths.

Inexact graph matching

Continuous optimization approaches

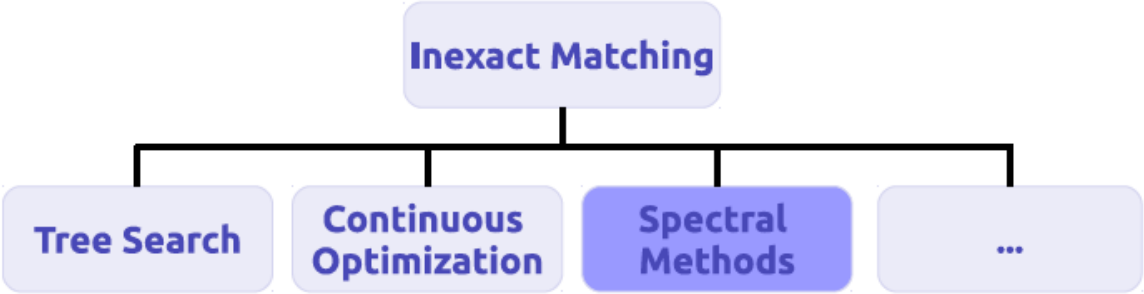


Continuous Optimization

- Idea:
  - Convert discrete optimization problem to a continuous, nonlinear optimization problem.
  - Use a nonlinear optimization algorithm.
  - Convert back to graph matching domain.
- Suboptimal Matching
- Polynomial Runtime

Inexact graph matching

Spectral methods



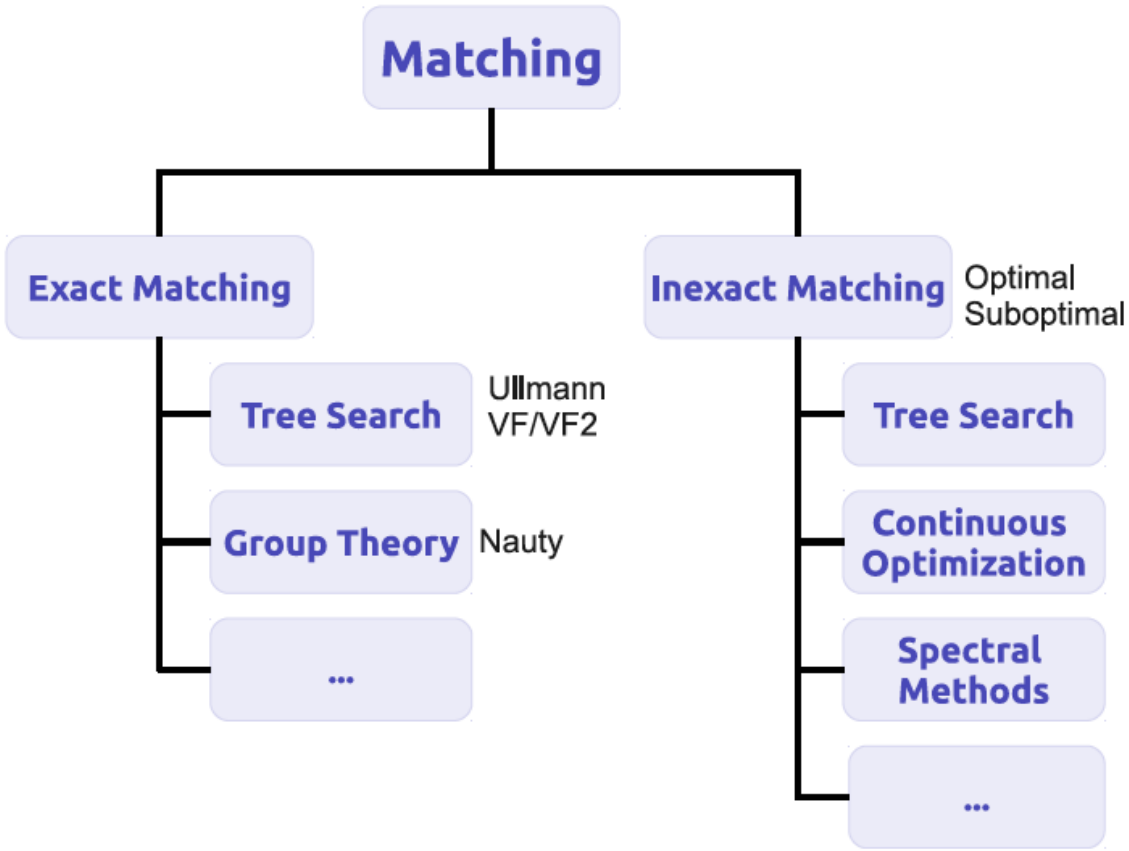
Spectral Methods

Uses property, that

- Eigenvalues and eigenvectors of adjacency matrix of a graph are invariant to node permutations.
- $G_1, G_2 isomorph \Rightarrow$
  $EW(Adj(G_1)) = EW(Adj(G_2)) \wedge EV(Adj(G_1)) = EV(Adj(G_2))$

Graph matching

Recall

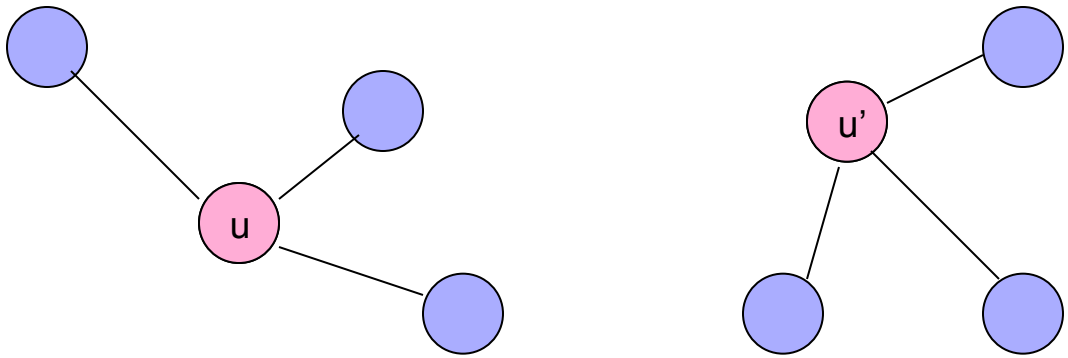


Inexact isomorphism (1)

- Univalent matching
  - Probabilistic propagation
  - Network based approach
- Multivalent matching
  - Solnon and Champin
    - Similarity measure based on the notion of mapping between graph vertices.
    - A complete approach, and then describe an efficient greedy algorithm.

Probabilist Relaxation

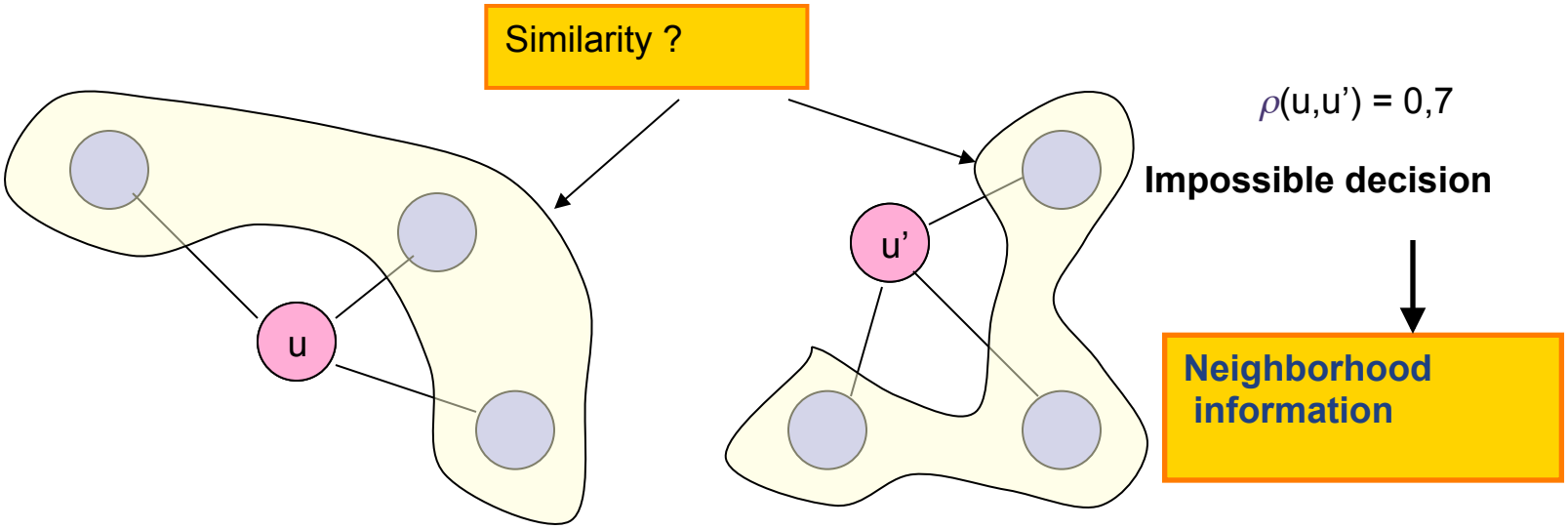
- Probabilist Relaxation [Hummel, Zucker, 1983] :
  - *Principle :*
    - Similarity measure ρ between nodes (attributes comparison)



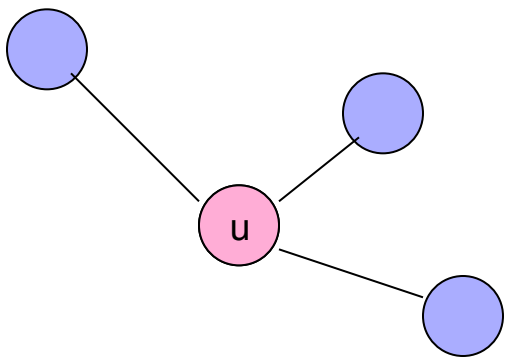
$\rho(u,u') = 0,7$

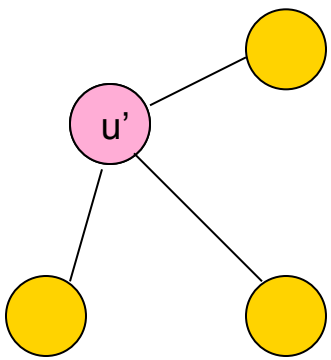
Impossible decision

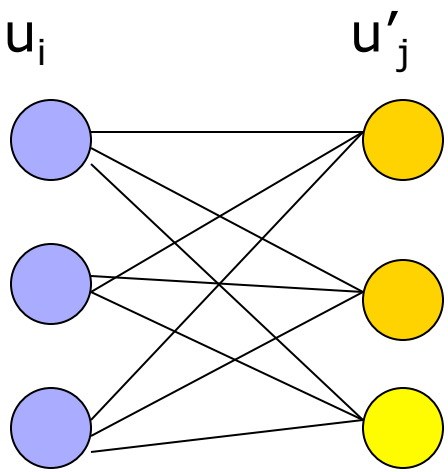
- Probabilist Relaxation [Hummel, Zucker, 1983] :
  - *Principle :*
    - Similarity measure ρ between nodes.
    - Contextual information is used to improve node similarity based on attributes



Complete bipartite graph of neighbours nodes.



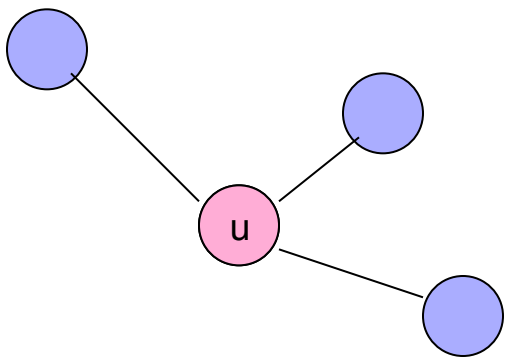


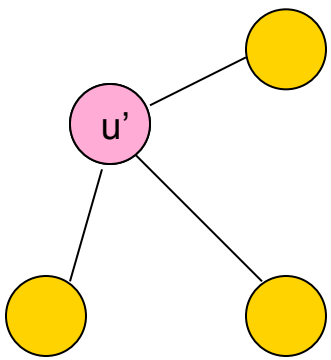


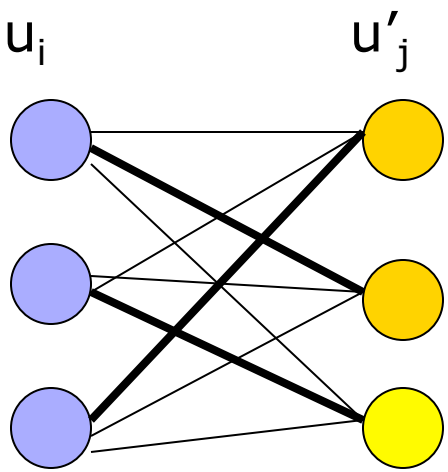
Weights on edges

$w(u_i,u'_j) = \rho(u_i,u'_j)$

Complete bipartite graph of neighbours nodes.







Maximal linkage

↓

Neighbours similarity

- Re-enforcing similarity between node pair in function of the neighbourhood :

$$\gamma(u,u') = \rho(u,u') * sim(N(u), N(u'))$$

- Recursive procedure

$$\gamma_0(u,u') = \rho(u,u')$$
$$\gamma_{i+1}(u,u') = \gamma_i(u,u') * sim_i(N(u), N(u'))$$

- **Heuristic** : Assignations
  - If $\gamma(u,u') >$ threshold
  - If no more potential candidate

N(node) = neighbourhood of a node.

- Problems :
  - Convergence
  - Sub-optimal
  - Does not take into account the global structure of the graph
- Advantages :
  - Provide a one to one matching. Univalent.
  - An efficient method.

Inexact graph matching (2)

- A network based approach to exact and inexact graph matching
  - B.T. Messmer and H. Bunke
  - 1993

A network based approach to exact and inexact graph matching

Models = Graph data set

Input graph = query graph

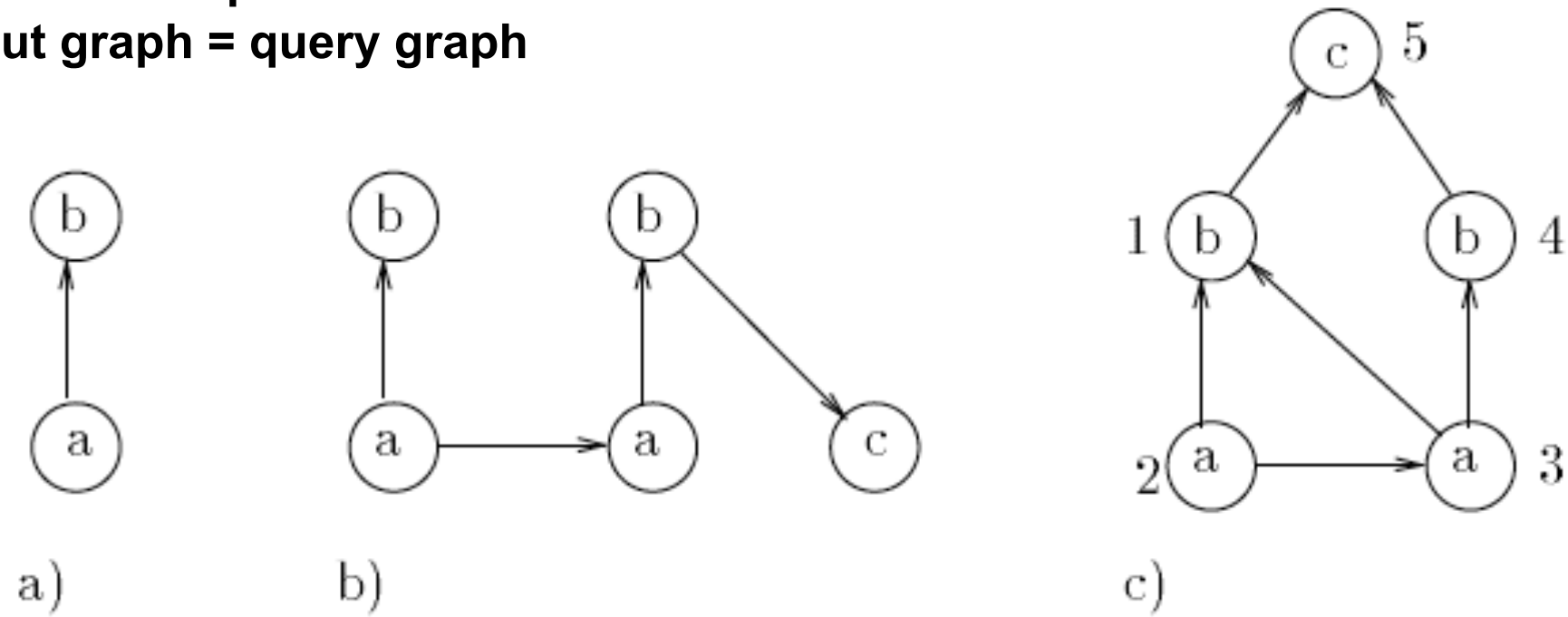
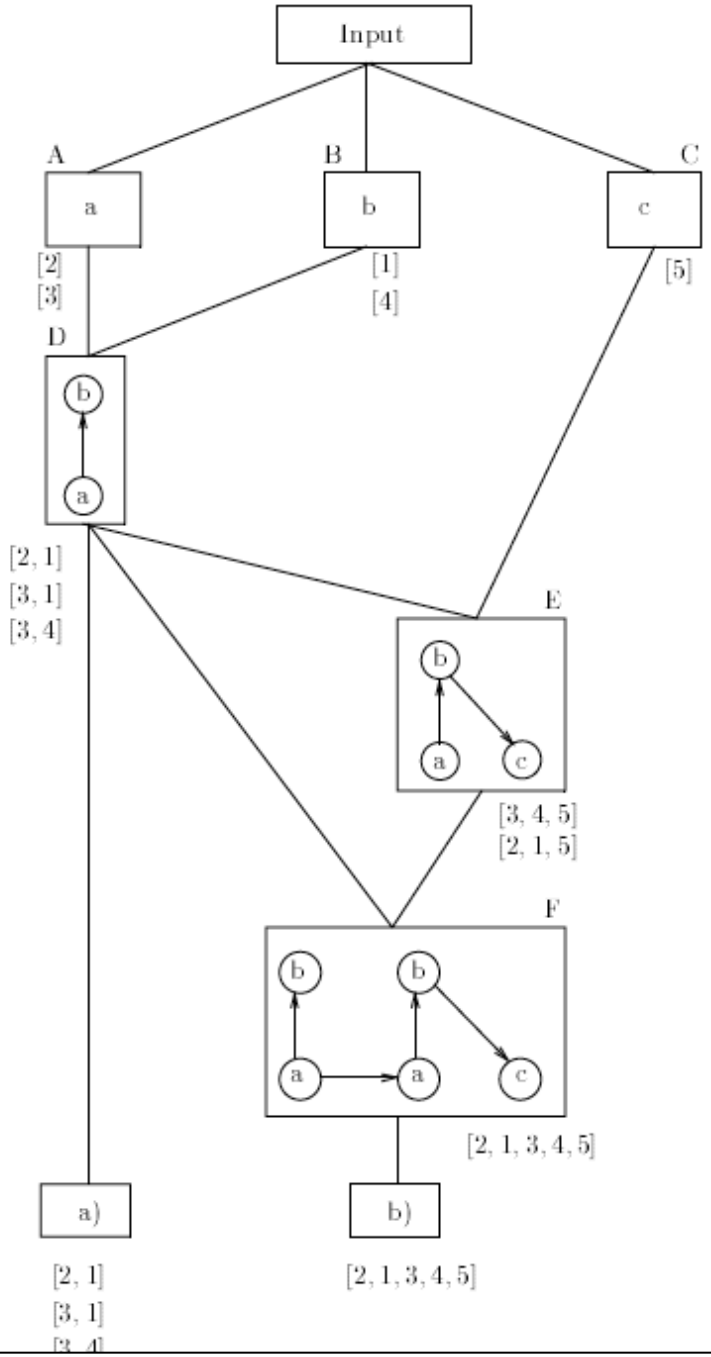
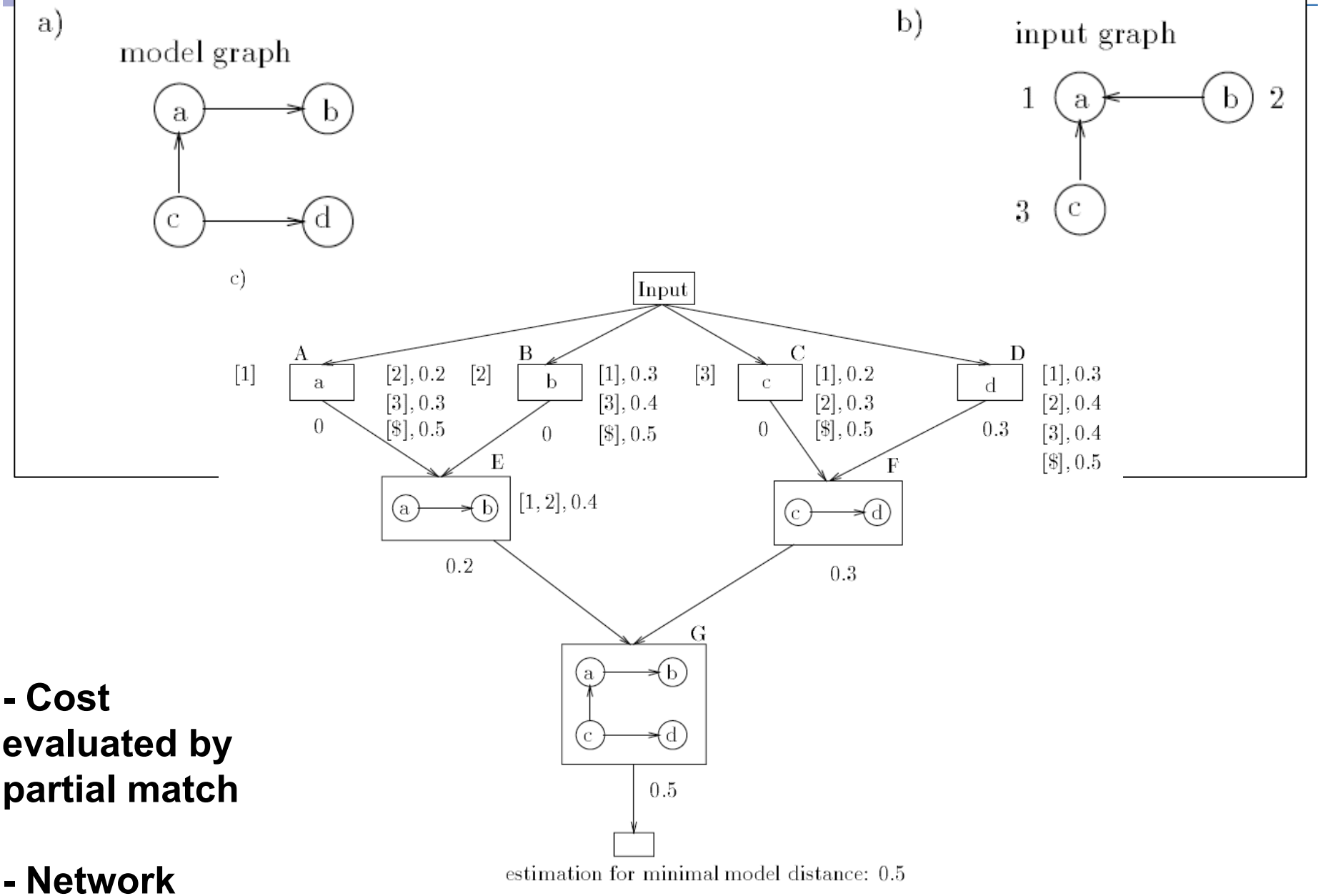


Figure 4: Two model graphs and an input graph


Input
A
a
[2]
[3]
B
b
[1]
[4]
C
c
[5]
D
b
a
[2, 1]
[3, 1]
[3, 4]
E
b
a
c
[3, 4, 5]
[2, 1, 5]
F
b
b
a
a
c
[2, 1, 3, 4, 5]
a)
[2, 1]
[3, 1]
b)
[2, 1, 3, 4, 5]

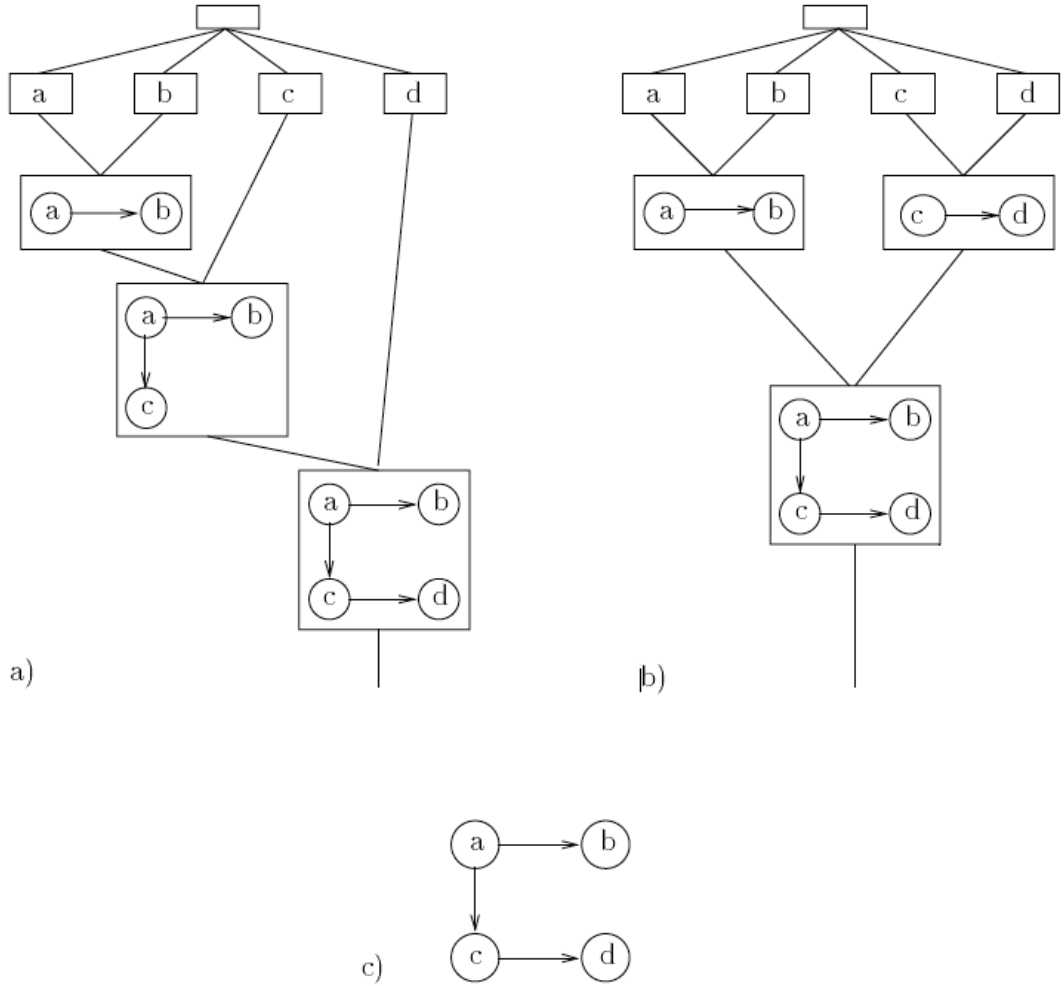


- Cost evaluated by partial match

- Network search by A*

Messmer 93

- Depends on the sub-graph insertion into the network. (Network compilation is the problem)



Inexact graph matching

- Integer Linear Programming
  - [Lebodic 2009] ICDAR

Integer Linear Programming Variables

$$x_{i,k} \in \{0,1\} \quad \forall i \in V_{\mathcal{S}}, \forall k \in V_{\mathcal{G}}$$

$$y_{ij,kl} \in \{0,1\} \quad \forall ij \in E_{\mathcal{S}}, \forall kl \in E_{\mathcal{G}}$$

Integer Linear Programming Constraints

Every vertex of **V**S must be matched to a unique vertex of **V**G

$$\sum_{k \in V_{\mathcal{G}}} x_{i,k} = 1 \quad \forall i \in V_{\mathcal{S}}$$

Every edge of **E**S must be matched to a unique edge of **E**G

$$\sum_{kl \in E_{\mathcal{G}}} y_{ij,kl} = 1 \quad \forall ij \in E_{\mathcal{S}}$$

Every vertex of **V**G must be matched to at most an edge of **E**S

$$\sum_{i \in V_{\mathcal{S}}} x_{i,k} \leq 1 \quad \forall k \in V_{\mathcal{G}}$$

If two vertices are matched together, an edge originating one of these two vertices must be matched with an edge originating the other vertex :

$$\sum_{kl \in E_{\mathcal{G}}} y_{ij,kl} = x_{i,k} \quad \forall k \in V_{\mathcal{G}}, \forall ij \in E_{\mathcal{S}}$$

If two vertices are matched together, an edge targeting one of these two vertices must be matched with an edge targeting the other vertex :

$$\sum_{kl \in E_{\mathcal{G}}} y_{ij,kl} = x_{j,l} \quad \forall l \in V_{\mathcal{G}}, \forall ij \in E_{\mathcal{S}}$$

Integer Linear Programming Constraints

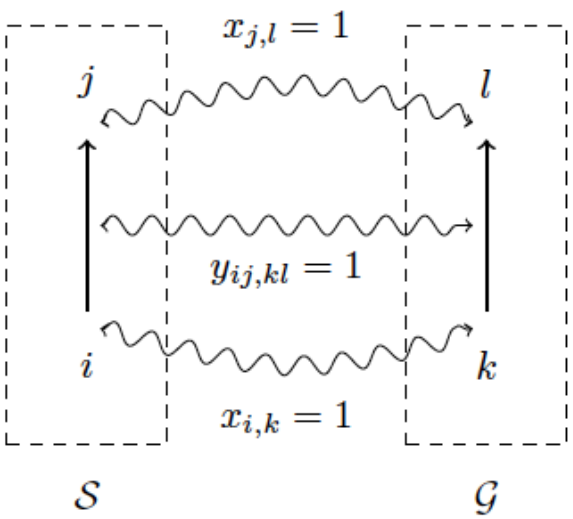


Figure 2. an example of matching. $\mathcal{S}$ and $\mathcal{G}$ both contain a single edge, respectively ij and kl. The following solution is represented on this figure : $x_{i,k} = 1$ (resp. $x_{j,l} = 1$, $y_{ij,kl} = 1$), i.e. i (resp. j, ij) is matched with k (resp. l, kl). Reversely, since i (resp. j) is not matched with l (resp. k), $x_{i,l} = 0$ (resp. $x_{j,k} = 0$).

Integer Linear Programming
Function to minimize

$$\min_{x,y} \sum_{i \in V_S} \sum_{k \in V_G} d(i,k) * x_{i,k} + \sum_{ij \in E_S} \sum_{kl \in E_G} d(ij,kl) * y_{ij,kl}$$

Graph Edit Distance (ED) [Bunke 99]

The minimum amount of distortion that is needed to transform G_1 into G_2

- Distortions s_i: deletions, insertions, substitutions of nodes and edges.
- Edit path $S = s_1, ..., s_n$: A sequence of edit operations that transforms G_1 into G_2.
- Cost functions: Measuring the strength of a given distortion.
- Edit distance $d(G_1, G_2)$: Minimum cost edit path between two graphs.

Problem of Edit Distance: NP complete

- Explore the space of all possible mappings of the nodes and edges of G_1 to the nodes and edges of G_2.
- Edit Distance computation also has a worst case exponential complexity which prevents its use in large datasets.

Graph Edit Distance (ED)

Figure: A possible edit path between graph g1 and g2 (node labels are represented by different shades of grey)

Maximum Common Subgraph (MCS)

$$d(G_1, G_2) = 1 - \frac{mcs(G_1, G_2)}{|G_1| + |G_2| - mcs(G_1, G_2)}$$

Relation between MCS and Edit Distance :

ED = Edit Distance

$$ED(G_1, G_2) = |G_1| + |G_2| - 2|mcs(G_1, G_2)|.$$

Graph edit distance

Algorithm 1 Graph Edit Distance Algorithm

Require: Non-empty attributed graphs $g_1 = (V_1, E_1, \mu_1, v_1)$ and $g_2 = (V_2, E_2, \mu_2, v_2)$ where $V_1 = \{u_1, ..., u_{|v_1|}\}$ and $V_2 = \{u_2, ..., u_{|v_2|}\}$

Ensure: A minimum cost edit path from g_1 to g_2 e.g. p_{min}=$\{u_1 \rightarrow v_3,\ u_2 \rightarrow \epsilon$,... ,$\epsilon \rightarrow v_2\}$

```
initialize OPEN to the empty set
For each node w ∈ V_2, insert the substitution {u1 → w} into OPEN
Insert the deletion {u_1 → ε} into OPEN
loop
    Remove p_min = argmin_{p∈OPEN}{g(p) + h(p)} from OPEN
    if p_min is a complete edit path then
        Return p_min as the solution
    else
        Let p_min = {u_1 → v_i1, ..., u_k → v_ik}
        if k < |V_1| then
            For each w ∈ V_2 \ {v_i1, ..., v_ik}, insert p_min ∪ {u_{k+1} → w} into OPEN
            Insert p_min ∪ u_{k+1} → ε into OPEN
        else
            Insert p_min ∪ {⋃_{wεV2\{v_i1,...,v_ik}} {ε → w}} into OPEN
        end if
    end if
end loop
```

Graph edit distance

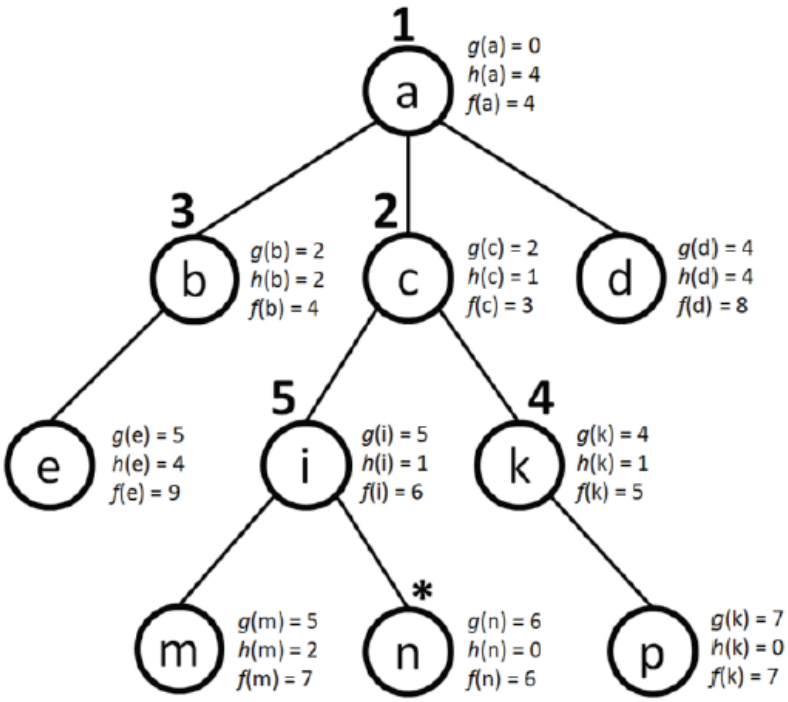


FIGURE 2.6: The numbers indicate in which order the nodes were expanded: the node with the lowest estimated cost f among all currently known nodes is always picked first. Eventually node n is reached and recognized as the optimal solution because the estimated remaining costs are 0 and no unexpanded node has a lower estimated total cost.

Inexact Graph Matching
Subgraph assignement

- Inexact Graph Matching can be reformulated into a simpler problem :
  - An assignement problem
  - Node assignement problem
- But it is an approximation of the problem

SubGraph decomposition and matching

Basic idea:

- Methods are based on an optimization procedure **mapping local substructures**
- Any node(u_n) from G_1 can be assigned to any node(v_m) of G_2,
- Incurring some **cost** that depends on the u_n-v_m assignment.
- It is required to map all nodes in such a way that the total cost of the assignment is **minimized**.

Cost matrix representation (C):

- C_{ij} correspond to the costs of assigning the ith node of G_1 to the jth node of G_2.

$$C = \overbrace{\begin{vmatrix} c_{1,1} & \cdots & \cdots & c_{1,m} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ c_{n,1} & \cdots & \cdots & c_{n,m} \end{vmatrix}}^{V1} \Big\} V2$$

SubGraph decomposition and matching

- The cost matrix contains the distances between every pair of subgraphs from G_1 and G_2.
  - What's the best (minimum-cost) way to assign the subgraphs?
- Assignment problem solved by the Hungarian method [Kuhn 1955]
- The cost of the minimum-weight subgraph matching :
  - SubGraph Matching Distance $SGMD(G_1, G_2)$

Example of possible variations of $SGMD$:

- $SGMD_{ED}$: Based on edit distance.
- $SGMD_{GP}$: Based on graph probing.

$$C = \begin{vmatrix} c_{1,1} & \cdots & \cdots & c_{1,m} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ c_{n,1} & \cdots & \cdots & c_{n,m} \end{vmatrix}$$

SubGraph decomposition and matching

[Raveaux10]

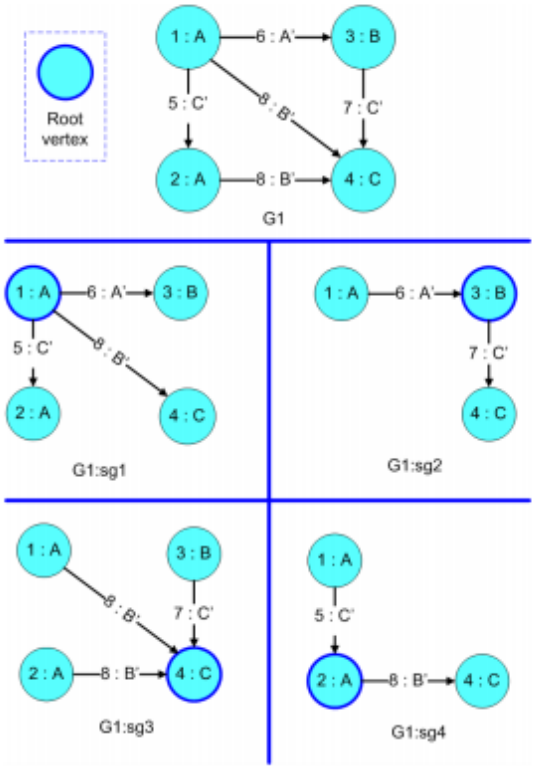
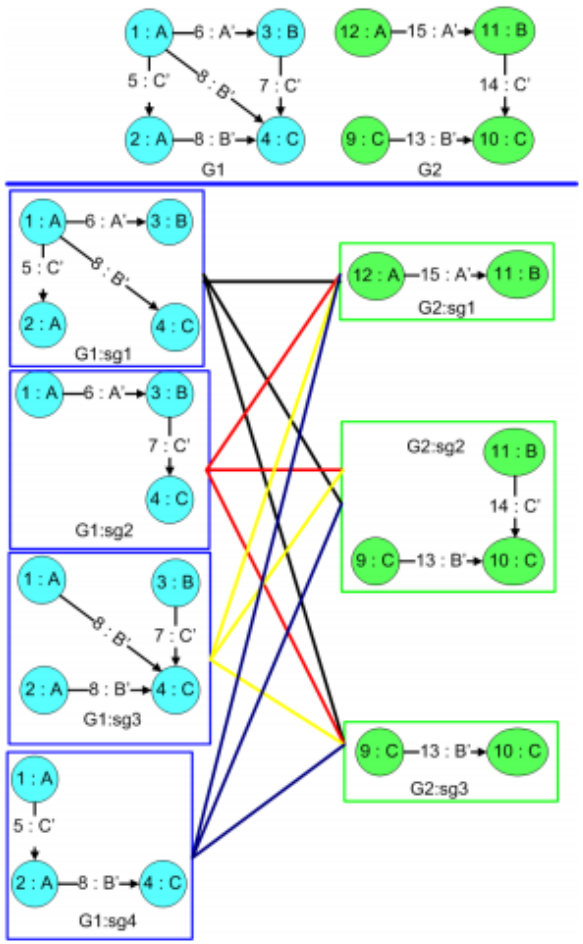


Figure: Decomposition into subgraph world



- Subgraph distance
- Bi-partite graph matching

Integer Projected Fixed Point (IPFP)

- Leordeanu et al. (2009)

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\{ S(\mathbf{x}) \stackrel{\text{def.}}{=} \frac{1}{2}\mathbf{x}^T \Delta \mathbf{x} + \mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{1},\ \mathbf{x} \in \{0,1\}^{(n+m)^2} \right\} \tag{17}$$

where $\mathbf{A}\mathbf{x} = \mathbf{1}_{2(n+m)}$, with $\mathbf{A} \in \{0,1\}^{2(n+m)\times(n+m)^2}$, is the matrix version of the bijectivity constraints given by Eq. 7.

- Delta : is edge dissimilarity matrix
- c : node dissimilarity cost (vector form)
- x : permutation matrix (vector form)

Integer Projected Fixed Point (IPFP)

- Relaxed Problem

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\{ S(\mathbf{x}) \mid \mathbf{A}\mathbf{x} = \mathbf{1},\ \mathbf{x} \geq \mathbf{0}_{(n+m)^2} \right\}. \qquad (18)$$

Integer Projected Fixed Point (IPFP)

Algorithm 1 IPFP($\mathbf{x}, \mathbf{c}, \Delta$)

while a fixed point is not reached **do**
  $\mathbf{b}^* \leftarrow \operatorname{argmin}\{(\mathbf{x}^T\Delta + \mathbf{c}^T)\mathbf{b} \mid \mathbf{A}\mathbf{b} = \mathbf{1},\ \mathbf{b} \in \{0,1\}^{(n+m)^2}\}$
  $t^* \leftarrow \operatorname{argmin}\{S(\mathbf{x} + t(\mathbf{b}^* - \mathbf{x})) \mid t \in [0,1]\}$
  $\mathbf{x} \leftarrow \mathbf{x} + t^*(\mathbf{b}^* - \mathbf{x})$
end while

- S is linearly approximated by its 1st-order expansion around the current solution
- Keeping x fixed, the minimization of S (b) is approximatively equivalent to a relaxed LSAP (b >=0).

Graduated NonConvexity and Concativity Procedure (GNCCP)

- **(Liu and Qiao, 2014)**

$$S_\zeta(\mathbf{x}) = (1 - |\zeta|)S(\mathbf{x}) + \zeta \mathbf{x}^T \mathbf{x} \tag{19}$$

where $\zeta \in [-1, 1]$. When $\zeta = 1$, $S_\zeta(\mathbf{x}) = \mathbf{x}^T \mathbf{x}$ is fully convex, and when $\zeta = -1$, $S_\zeta(\mathbf{x}) = -\mathbf{x}^T \mathbf{x}$ is concave.

Graduated NonConvexity and Concativity Procedure (GNCCP)

- a convex-concave relaxation through the modified quadratic function

Algorithm 2 GNCCP($\Delta, k_{\max}$)

$\zeta = 1, d = 0.1, \mathbf{x} = \mathbf{0}$
while $(\zeta > -1) \wedge (\mathbf{x} \notin \mathcal{A}_{n,m})$ **do**
  $\Delta_\zeta \leftarrow \frac{1}{2}(1 - |\zeta|)\Delta + \zeta \mathbf{I}$
  $\mathbf{x} \leftarrow \mathrm{IPFP}(\mathbf{x}, (1 - |\zeta|)\mathbf{c}, \Delta_\zeta)$
  $\zeta \leftarrow \zeta - d$
end while

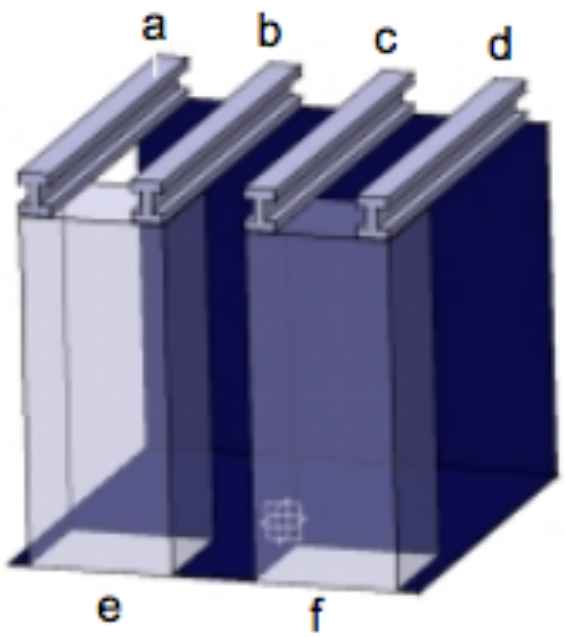
- no initial mapping is required
- algorithm smoothly interpolates convex and concave relaxations by iteratively decreasing \zeta from 1 to -1
- Convergence is reached when x is a permutation
- Note that the minimum of the concave relaxation is a permutation

Inexact graph matching

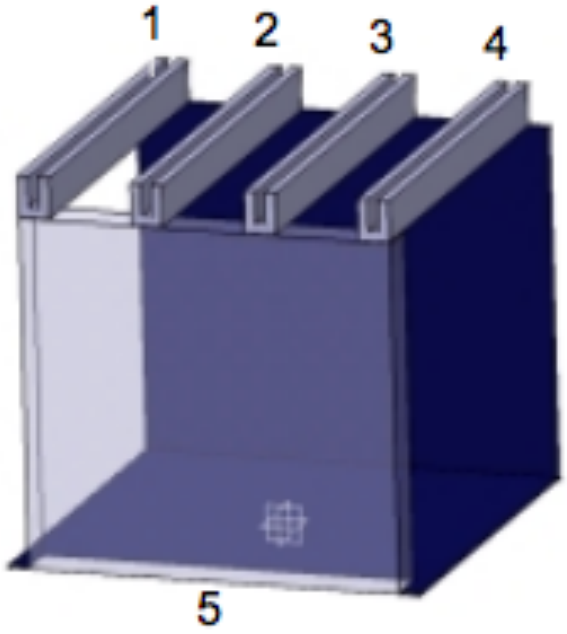
- Multivalent matching :
  - Many to many
  - Champin and Solnon **[Champin / Solnon] (2003-2005)**

Multivalent matching

[Champin / Solnon] (2003-2005)



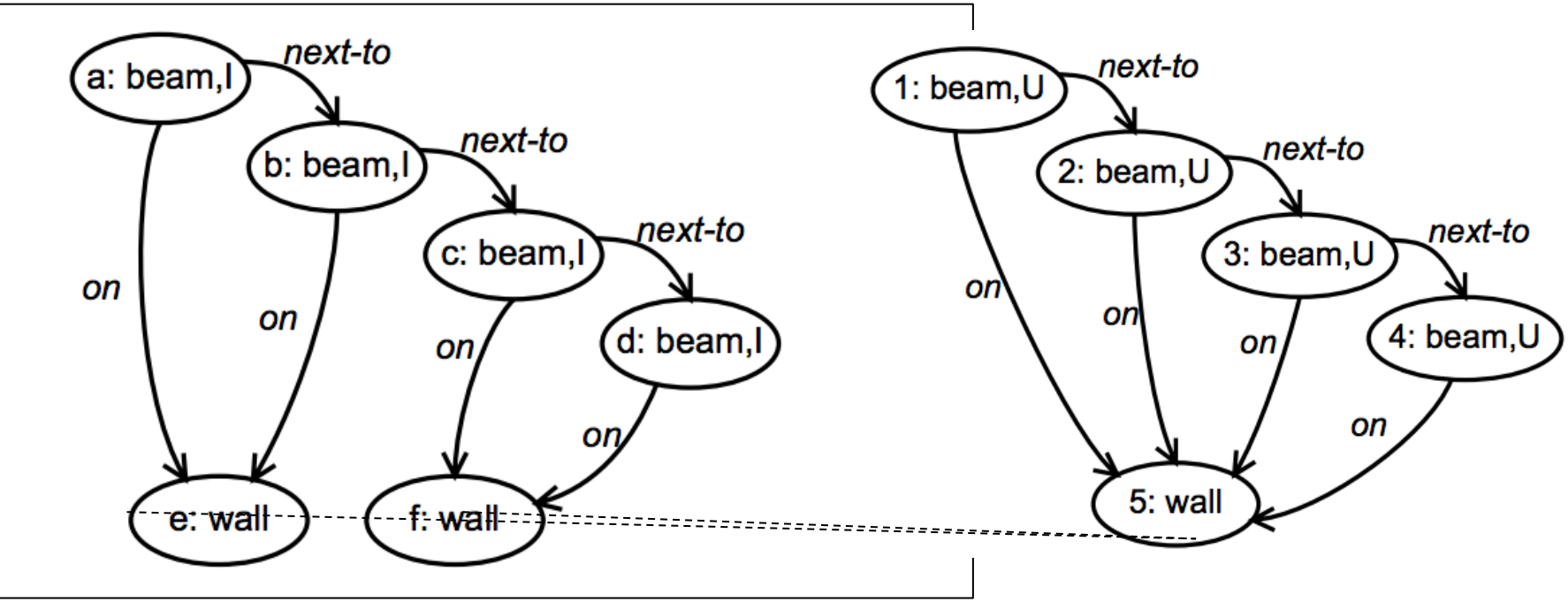
Object 1



Object 2

[Champin / Solnon] (2003-2005)

Similarity with Respect to a Mapping



Score function with split operation

$$score(m) = f(descr(G_1) \sqcap_m descr(G_2)) - g(splits(m))$$

Find the mapping *m* that maximizes the score function

- $m \subseteq V_1 \times V_2$

How to choose the similarity function ?

[Champin / Solnon] (2003-2005)

Solution Space

This problem is highly combinatorial

Search space = all different mappings —all subsets of V1 × V2— and it contains $2^{|V1|*|V2|}$ states.

This search space can be structured in a lattice by the set inclusion relation

It can be explored in an exhaustive way with a “branch and bound” approach.

How to explore the solution space :

- Complete search
  - Lattice
- Heuristic
  - Greedy algorithm
  - Taboo search
  - …

Literature

- D.Conte, Thirty Years Of Graph Matching in Pattern Recognition, International Journal of Pattern Recognition and Artificial Intelligence, Vol. 18, No. 3, p.265-298, (2004)

Conclusion

- 5 problems
  1. Exact Subgraph Isomorphism (ESGI)
  2. Substitution-Tolerant Subgraph Isomorphism (STSI)
  3. Inexact Subgraph Isomorphism (ISGI)
  4. Inexact Graph Isomorphism (IGI)
  5. Multivalent Inexact Graph Isomorphism (MIGI)

Conclusion

- Exact problems : problems 1 and 2
  - Many methods to solve this problem :
    - Ullman, VF, LAD
    - Large graphs can be tackeled (>200 nodes)

Conclusion

- Exact problems

Table 1
Results obtained on T_1 for the proposed approach, VF2 and LAD. This table gives the sum of the processing time in seconds for the five solved instances.

p	$n_{\mathcal{G}}$	$n_{\mathcal{S}} = \lvert V_{\mathcal{S}} \rvert$								
		10			25			50		
		ILPiso	VF2	LAD	ILPiso	VF2	LAD	ILPiso	VF2	LAD
0.01	50	0.02	0.03	0.00	0.04	0.02	0.00	0.08	0.02	0.00
	100	0.02	0.02	0.00	0.06	0.01	0.00	0.10	0.02	0.00
	250	0.03	0.02	0.02	0.10	0.02	0.02	0.28	0.02	0.02
	500	0.08	0.06	0.01	0.26	0.05	0.04	0.72	0.06	0.01
0.05	50	0.03	0.02	0.00	0.06	0.02	0.00	0.30	0.02	0.02
	100	0.06	0.01	0.02	0.21	0.02	0.01	0.76	0.02	0.02
	250	0.15	0.03	0.04	0.85	0.04	0.04	4.24	0.05	0.04
	500	0.68	0.14	0.34	2.87	0.14	0.40	14.82	0.13	0.40
0.1	50	0.05	0.02	0.01	0.22	0.02	0.01	1.28	0.02	0.02
	100	0.10	0.02	0.01	0.63	0.02	0.01	3.10	0.01	0.01
	250	0.42	0.06	0.10	3.64	0.01	0.14	15.52	0.06	0.08
	500	1.90	0.24	1.26	15.33	0.22	1.33	42.33	0.24	1.25

Conclusion

- Substitution tolerant problem :
  - Recently re-expressed as an ILP formulation

Table 2
Results obtained on T_2 for the proposed approach. This table gives the sum of the processing time in seconds for the five solved instances.

p	$n_{\mathcal{G}} = \lvert V_{\mathcal{G}} \rvert$	$n_{\mathcal{S}} = \lvert V_{\mathcal{S}} \rvert$		
		10	25	50
0.01	50	0.02	0.04	0.08
	100	0.04	0.05	0.12
	250	0.07	0.14	0.46
	500	0.08	0.39	1.76
0.05	50	0.02	0.10	0.52
	100	0.04	0.27	1.82
	250	0.19	1.58	12.68
	500	1.17	9.51	49.87
0.1	50	0.07	0.48	9.60
	100	0.19	1.60	26.34
	250	1.55	16.18	108.76
	500	2.50	98.75	370.80

Conclusion

- Inexact graph matching
  - Exact graph edit distance computation algorithm Exact based on the well-known A* algorithm.
  - Exact is only able to compute the edit distances of graphs typically containing 12 vertices at most in practice.

Conclusion

- Inexact graph matching

Table 2
Accuracy (Acc), correlation (ρ), time (t) on all datasets

Database	Heuristic-A*			Beam(10)			BP		
	Acc	ρ	t	Acc	ρ	t	Acc	ρ	t
Letter (L)	91.0	1.00	649′05″	91.1	0.95	107′52″	91.1	0.98	7′52″
Letter (M)	77.9	1.00	2061′29″	78.5	0.93	120′04″	77.6	0.93	6′11″
Letter (H)	63.0	1.00	4914′45″	63.9	0.93	149′47″	61.6	0.97	8′48″
COIL	93.3	1.00	199′19″	93.3	0.99	156′46″	93.3	0.99	24′27″
GREC	–	–	–	76.7	–	1826′46″	86.3	–	12′11″
Fingerprint	–	–	–	84.6	–	166′05″	78.7	–	2′09″
Molecules	–	–	–	96.2	–	1047′55″	97.0	–	4′17′
Proteins	–	–	–	–	–	–	89.3	–	322′21″

Conclusion

- The definition of an error model is strongly application-dependent.
- Inexact Graph matching provides :
  - Node mapping
  - Similarity or dissimilarity measure
- Graph matching cannot be directly applied to vectorial machine learning techniques :
  - Classification : MLP, SVM, Bayesian classifiers …
  - Clustering : K-means, …
- Graph matching can be directly applied to dissimilarity-based machine learning techniques :
  - Classification: K-NN
  - Clustering : PAM, hierarchical clustering, …