Mesurer la similarité de graphes

Christine Solnon

LIRIS, UMR 5205 CNRS / Université Lyon 1

Avec la participation de :

- Pierre-Antoine Champin, LIRIS, Lyon
- Vianney le Clément, UCL, Louvain la neuve
- Guillaume Damiand, LIRIS, Lyon
- Yves Deville, UCL, Louvain la neuve
- Colin de la Higuera, LINA, Nantes
- Jean-Christophe Janodet, LHC, Saint Etienne
- Olfa Sammoud, LIRIS, Lyon
- Sébastien Sorlin, LIRIS, Lyon
- Stéphane Zampelli, UCL, Louvain la neuve

Graph matching problems

Why matching graphs ?

- Many applications require to measure object similarity
  $\leadsto$ Classification, Search by example, Case-based Reasoning, ...
- Graphs are often used to model objects
  $\leadsto$ Images, Molecules, Documents, Design objects, ...
- Graph similarity is measured by matching their vertices

What is a matching ?

A matching of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a relation $m \subseteq V_1 \times V_2$
$\leadsto$ $(u_1, u_2) \in m \Rightarrow$ vertex u_1 is matched to vertex u_2

Graph matching problems

Why matching graphs ?

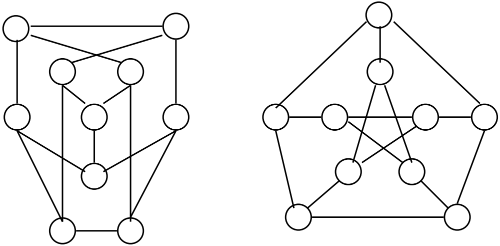
- Many applications require to measure object similarity
  $\rightsquigarrow$ Classification, Search by example, Case-based Reasoning, ...
- Graphs are often used to model objects
  $\rightsquigarrow$ Images, Molecules, Documents, Design objects, ...
- Graph similarity is measured by matching their vertices

What is a matching ?

A matching of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a relation $m \subseteq V_1 \times V_2$
$\rightsquigarrow$ $(u_1, u_2) \in m \Rightarrow$ vertex u_1 is matched to vertex u_2

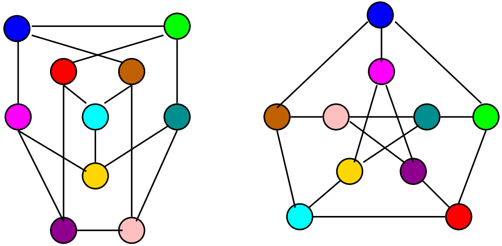
Well known examples of graph matching problems

- Graph Isomorphism $\rightsquigarrow$ Equivalence
- Subgraph Isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Graph Edit Distance $\rightsquigarrow$ Best univalent matching
- Extended Graph Edit Distance $\rightsquigarrow$ Best multivalent matching



Well known examples of graph matching problems

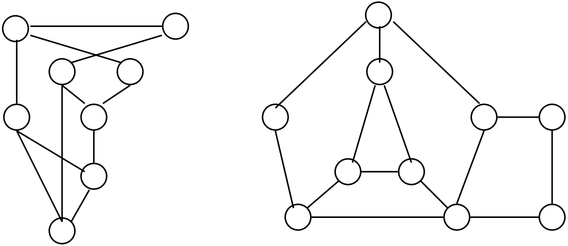
- Graph Isomorphism $\rightsquigarrow$ Equivalence
- Subgraph Isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Graph Edit Distance $\rightsquigarrow$ Best univalent matching
- Extended Graph Edit Distance $\rightsquigarrow$ Best multivalent matching



- Bijection $f : V_1 \rightarrow V_2$ that preserves all edges
- Isomorphic-complete problem... rather easy actually

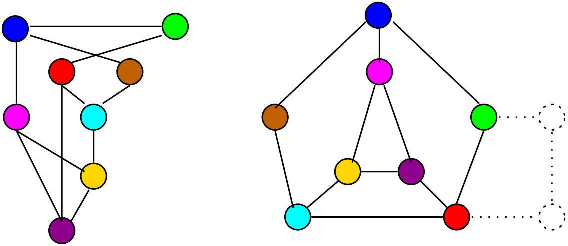
Well known examples of graph matching problems

- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching



Well known examples of graph matching problems

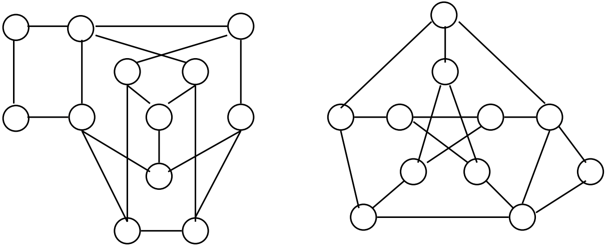
- Graph Isomorphism $\rightsquigarrow$ Equivalence
- Subgraph Isomorphism $\rightsquigarrow$ Inclusion
- Maximum common subgraph $\rightsquigarrow$ Intersection
- Graph Edit Distance $\rightsquigarrow$ Best univalent matching
- Extended Graph Edit Distance $\rightsquigarrow$ Best multivalent matching



- Injection $f : V_1 \rightarrow V_2$ that preserves all pattern edges
- NP-complete problem... still tractable for ”medium” size graphs

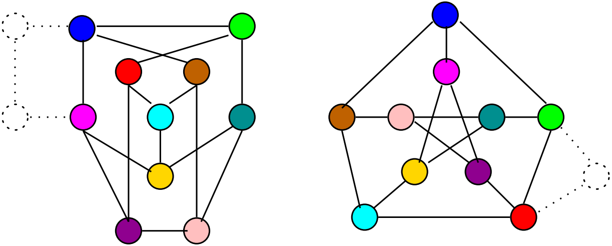
Well known examples of graph matching problems

- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching



Well known examples of graph matching problems

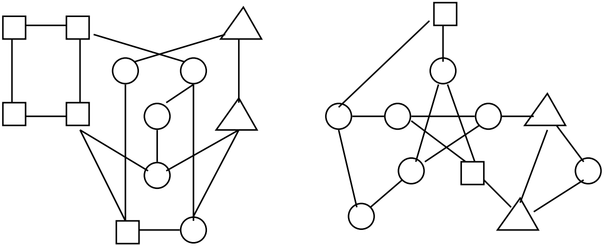
- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching



- Univalent matching that preserves as many edges as possible
- NP-hard problem... untractable for complete approaches

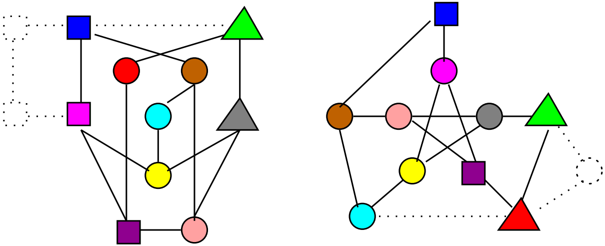
Well known examples of graph matching problems

- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching



Well known examples of graph matching problems

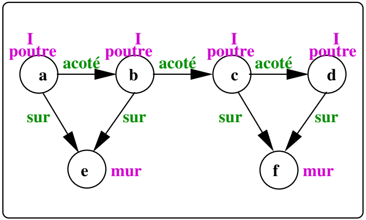
- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching

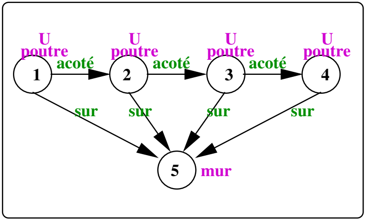


- Univalent matching that minimizes edition costs
- NP-hard problem... untractable for complete approaches

Well known examples of graph matching problems

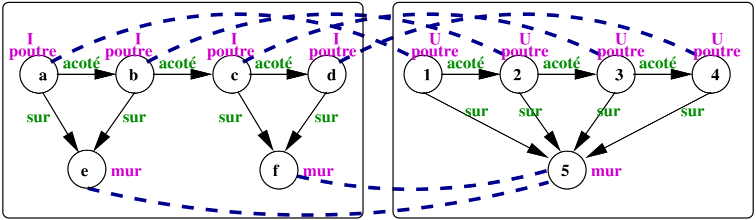
- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching





Well known examples of graph matching problems

- Graph Isomorphism ⇝ Equivalence
- Subgraph Isomorphism ⇝ Inclusion
- Maximum common subgraph ⇝ Intersection
- Graph Edit Distance ⇝ Best univalent matching
- Extended Graph Edit Distance ⇝ Best multivalent matching



- Multivalent matching that minimizes edition costs
- NP-hard problem... untractable for complete approaches

Overview of the talk

- Filtering algorithms for (sub)graph isomorphism
  $\rightsquigarrow$ joint work with Y. Deville, S. Sorlin, and S. Zampelli
- Polynomial algorithm for plane subgraph isomorphism
  $\rightsquigarrow$ joint work with G. Damiand, C. de la Higuera, and J.-C. Janodet
- Heuristic approaches for multivalent matching problems
  $\rightsquigarrow$ joint work with P.-A. Champin, O. Sammoud, and S. Sorlin
- Constraint-based graph matching
  $\rightsquigarrow$ joint work with V. le Clément, and Y. Deville

Filtering algorithms for (sub)graph isomorphism

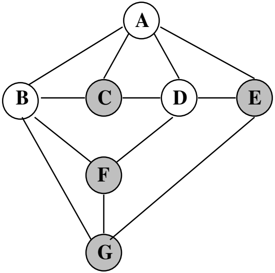
Basic principle of ”branch & filter” approaches

- Explore all possible matchings by structuring them in a tree
  $\rightsquigarrow$ Each node corresponds to a partial injective matching
- At each step: filter the set of candidate matchings
  $\rightsquigarrow$ Remove $(u, v) \in N_p \times N_t$ such that u cannot be matched to v

Filtering for (sub)graph isomorphism

- Propagation of all diff constraints [Régin 93] in $\mathcal{O}(n_p^2 n_t^2)$
- Propagation of edge constraints
  - Graph isomorphism $\rightsquigarrow$ degree-based labeling (Nauty, Saucy, IDL)
  - Subgraph isomorphism $\rightsquigarrow$ local all diff

Degree-based labeling for graph isomorphism: Example



α_G : Initial labeling $\rightsquigarrow$ degree

- $A, B, D \quad \rightarrow 4 \quad \Rightarrow$ ○
- $C, E, F, G \quad \rightarrow 3 \quad \Rightarrow$ ●

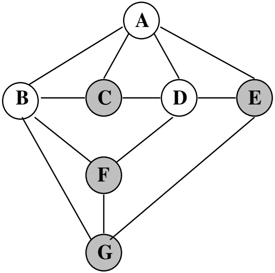
α'_G : First labeling extension

- $A \quad \rightarrow$ ○.{(2, ○), (2,)}
- $E, F \quad \rightarrow$.{(2, ○), (1,)}
- $B, D \quad \rightarrow$ ○.{(1, ○), (3,)}
- $C \quad \rightarrow$.{(3, ○)}
- $G \quad \rightarrow$.{(1, ○), (2,)}

α''_G : Second labeling extension

- $E \quad \rightarrow$.{(1,), (1,), (1,)}
- $F \quad \rightarrow$.{(2,), (1,)}
- $B \quad \rightarrow$.{(1,), (2,), (1,), (1,)}
- $D \quad \rightarrow$.{(1,), (1,), (2,)}

Degree-based labeling for graph isomorphism: Example



α_G : Initial labeling $\rightsquigarrow$ degree

- $A, B, D \rightarrow 4 \Rightarrow$ ○
- $C, E, F, G \rightarrow 3 \Rightarrow$ ●

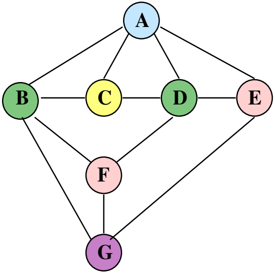
α'_G : First labeling extension

- $A \rightarrow$ ○.{(2, ○), (2, ●)}
- $E, F \rightarrow$ ●.{(2, ○), (1, ●)}
- $B, D \rightarrow$ ○.{(1, ○), (3, ●)}
- $C \rightarrow$ ●.{(3, ○)}
- $G \rightarrow$ ●.{(1, ○), (2, ●)}

α''_G : Second labeling extension

- $E \rightarrow$.{(1,), (1,), (1,)}
- $F \rightarrow$.{(2,), (1,)}
- $B \rightarrow$.{(1,), (2,), (1,), (1,)}
- $D \rightarrow$.{(1,), (1,), (2,)}

Degree-based labeling for graph isomorphism: Example



α_G : Initial labeling $\rightsquigarrow$ degree

- $A, B, D \rightarrow 4 \Rightarrow$ ○
- $C, E, F, G \rightarrow 3 \Rightarrow$ ●

α'_G : First labeling extension

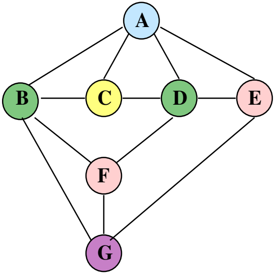
- $A \rightarrow$ ○.{(2, ○), (2, ●)} $\Rightarrow$ 🔵
- $E, F \rightarrow$ ●.{(2, ○), (1, ●)} $\Rightarrow$ 🔴
- $B, D \rightarrow$ ○.{(1, ○), (3, ●)} $\Rightarrow$ 🟢
- $C \rightarrow$ ●.{(3, ○)} $\Rightarrow$ 🟡
- $G \rightarrow$ ●.{(1, ○), (2, ●)} $\Rightarrow$ 🟣

$\rightsquigarrow$ relabel E, F, B, and D

α''_G : Second labeling extension

- $E \rightarrow$.{(1,), (1,), (1,)}
- $F \rightarrow$.{(2,), (1,)}
- $B \rightarrow$.{(1,), (2,), (1,), (1,)}
- $D \rightarrow$.{(1,), (1,), (2,)}

Degree-based labeling for graph isomorphism: Example



α_G : Initial labeling $\rightsquigarrow$ degree

$A, B, D \rightarrow 4 \Rightarrow$ [white]

$C, E, F, G \rightarrow 3 \Rightarrow$ [grey]

α'_G : First labeling extension

$A \rightarrow$ [white]$.\{(2,$ [white]$), (2,$ [grey]$)\} \Rightarrow$ [light blue]

$E, F \rightarrow$ [grey]$.\{(2,$ [white]$), (1,$ [grey]$)\} \Rightarrow$ [pink]

$B, D \rightarrow$ [white]$.\{(1,$ [white]$), (3,$ [grey]$)\} \Rightarrow$ [green]

$C \rightarrow$ [grey]$.\{(3,$ [white]$)\} \Rightarrow$ [yellow]

$G \rightarrow$ [grey]$.\{(1,$ [white]$), (2,$ [grey]$)\} \Rightarrow$ [purple]

$\rightsquigarrow$ relabel E, F, B, and D

α''_G : Second labeling extension

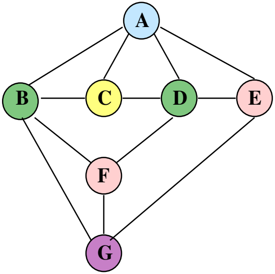
$E \rightarrow$ [pink]$.\{(1,$ [light blue]$), (1,$ [green]$), (1,$ [purple]$)\}$

$F \rightarrow$ [pink]$.\{(2,$ [green]$), (1,$ [purple]$)\}$

$B \rightarrow$ [green]$.\{(1,$ [light blue]$), (2,$ [yellow]$), (1,$ [pink]$), (1,$ [purple]$)\}$

$D \rightarrow$ [green]$.\{(1,$ [light blue]$), (1,$ [yellow]$), (2,$ [pink]$)\}$

Degree-based labeling for graph isomorphism: Example



α_G : Initial labeling $\rightsquigarrow$ degree

- $A, B, D \rightarrow 4 \Rightarrow$ ○
- $C, E, F, G \rightarrow 3 \Rightarrow$ ●

α'_G : First labeling extension

- $A \rightarrow$ ○.{(2, ○), (2, ●)} $\Rightarrow$ [light blue]
- $E, F \rightarrow$ ●.{(2, ○), (1, ●)} $\Rightarrow$ [pink]
- $B, D \rightarrow$ ○.{(1, ○), (3, ●)} $\Rightarrow$ [green]
- $C \rightarrow$ ●.{(3, ○)} $\Rightarrow$ [yellow]
- $G \rightarrow$ ●.{(1, ○), (2, ●)} $\Rightarrow$ [purple]

$\rightsquigarrow$ relabel E, F, B, and D

α''_G : Second labeling extension

- $E \rightarrow$ [pink].{(1, [light blue]), (1, [green]), (1, [purple])}
- $F \rightarrow$ [pink].{(2, [green]), (1, [purple])}
- $B \rightarrow$ [green].{(1, [light blue]), (2, [yellow]), (1, [pink]), (1, [purple])}
- $D \rightarrow$ [green].{(1, [light blue]), (1, [yellow]), (2, [pink])}

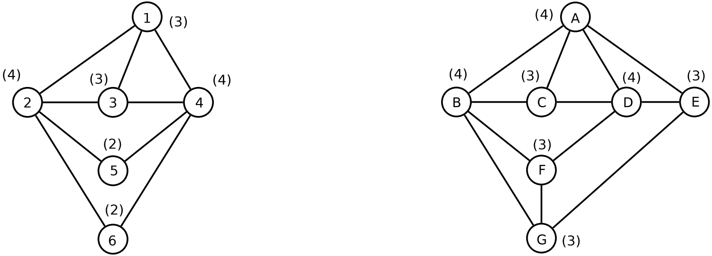
All different labels $\Rightarrow$ stop

Properties

- Correction: 2 nodes with different labels cannot be matched by an isomorphism function
- Time complexity of filtering (worst case): $\mathcal{O}(|V|^3 \log |V|)$

$\leadsto$ Solve instances with a few thousands of nodes in a second or so

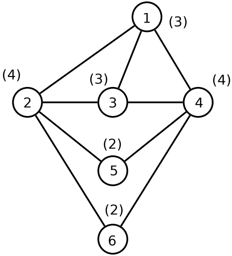
Filtering for subgraph isomorphism: example

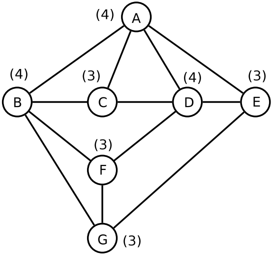


- Degree-based filtering:

⇝ 2 and 4 cannot be matched to C, E, F and G

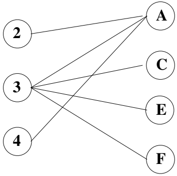
Filtering for subgraph isomorphism: example





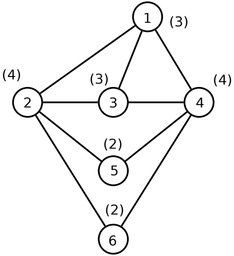
- Neighborhood all-diff filtering: look-ahead step

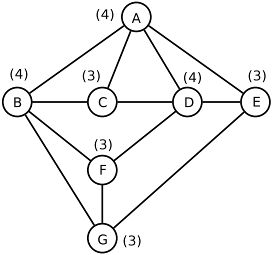
1 may be matched to *D* only if its neighbors may be matched to different neighbors of *D*



Both 2 and 4 can only be matched to *A*
⇝ 1 cannot be matched to *D*

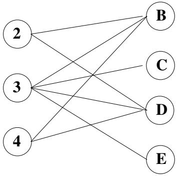
Filtering for subgraph isomorphism: example





- Neighborhood all-diff filtering: forward-checking step

Once 1 is matched to A, remove couples that can't be matched



2 and 4 can only be matched to B and D
$\rightsquigarrow$ 3 cannot be matched to B and D

Properties

- Correction: does not remove solutions
- Time complexity of filtering (worst case): $\mathcal{O}(|V_p| \cdot |V_t| \cdot d^{9/2})$ (Algorithm of Hopcroft and Karp)

$\rightsquigarrow$ Solve instances with a few hundreds of nodes in a minute or so

Overview of the talk

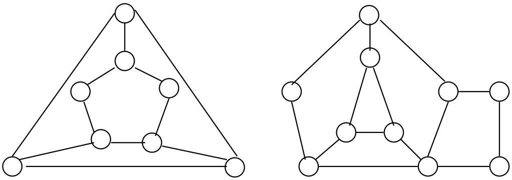
- Filtering algorithms for (sub)graph isomorphism
  ⇝ joint work with Y. Deville, S. Sorlin, and S. Zampelli
- **Polynomial algorithm for plane subgraph isomorphism**
  ⇝ joint work with G. Damiand, C. de la Higuera, and J.-C. Janodet
- Heuristic approaches for Multivalent matching problems
  ⇝ joint work with P.-A. Champin, O. Sammoud, and S. Sorlin
- Constraint-based graph matching
  ⇝ joint work with V. le Clément, and Y. Deville

Motivations

Search patterns in images

- Model images ⇝ graphs (RAGs, Delaunay triangulation, ...)
- Search patterns ⇝ subgraph isomorphism

  NP-complete in the general case... but do we consider the right problem when graphs model images ?



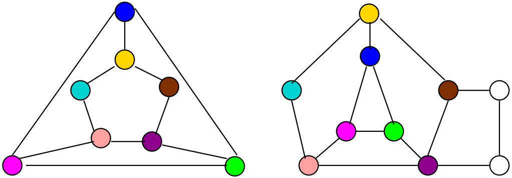
Is there a subgraph isomorphism ???

Motivations

Search patterns in images

- Model images $\rightsquigarrow$ graphs (RAGs, Delaunay triangulation, ...)
- Search patterns $\rightsquigarrow$ subgraph isomorphism

  NP-complete in the general case... but do we consider the right problem when graphs model images ?



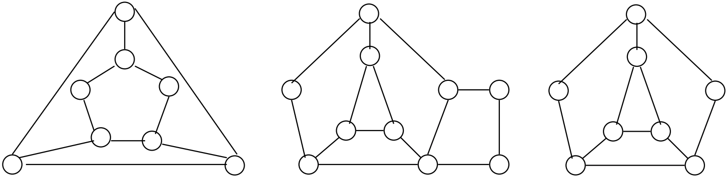
Yes but... the two graphs look rather different !
Graphs modeling images are planar and are embedded in planes.
$\rightsquigarrow$ Let us compare planar embeddings of graphs !

Motivations

Search patterns in images

- Model images ⇝ graphs (RAGs, Delaunay triangulation, ...)
- Search patterns ⇝ subgraph isomorphism

  NP-complete in the general case... but do we consider the right problem when graphs model images ?



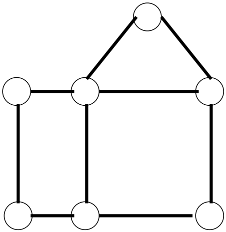
Yes but... the two graphs look rather different !
Graphs modeling images are planar and are embedded in planes.
⇝ Let us compare planar embeddings of graphs !

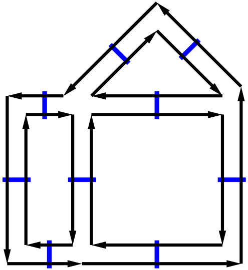
2D Combinatorial maps

From plane graphs to 2D combinatorial maps

- Each edge is decomposed into 2 linked darts
- Faces are defined by dart successions

Plane graph





Combinatorial map

Algorithm for submap isomorphism

function testSubIsomorphism(M, M')

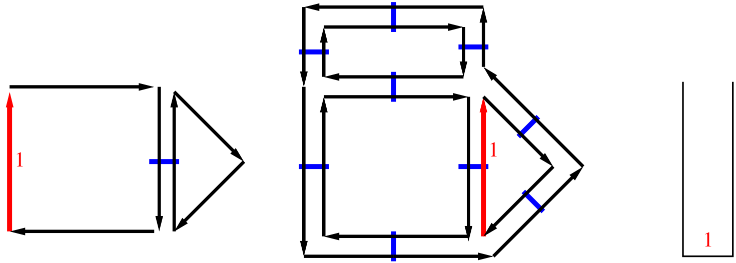
Input: 2 open connected maps M and M'
Output: returns true iff M is isomorphic to a submap of M'

- Choose $d_0 \in D$
- **For** every dart $d'_0 \in D'$ **do** :
  - **If** traverseAndMatch(M, M', d_0, d'_0)
  - **then return** true
- **return** false

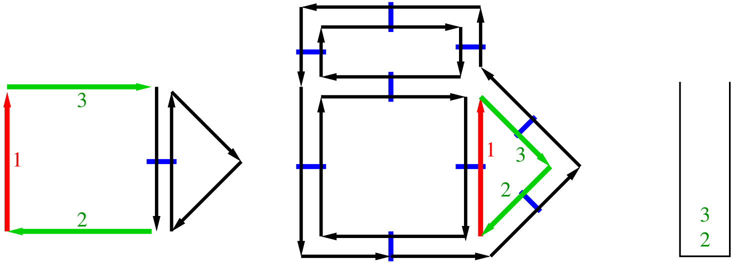
Complexity in $\mathcal{O}(|D| \cdot |D'|)$

- There are at most $|D'|$ map traversals
- Each traversal is in $\mathcal{O}(|D|)$

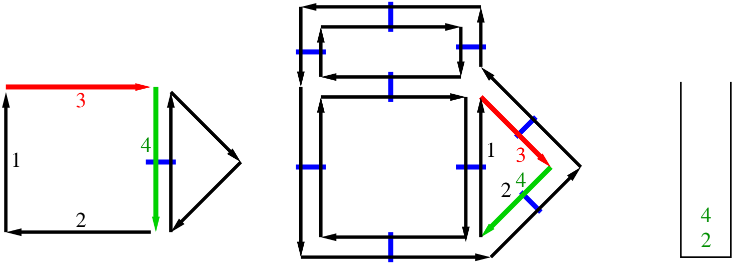
Example with « wrong » initial darts



Example with « wrong » initial darts

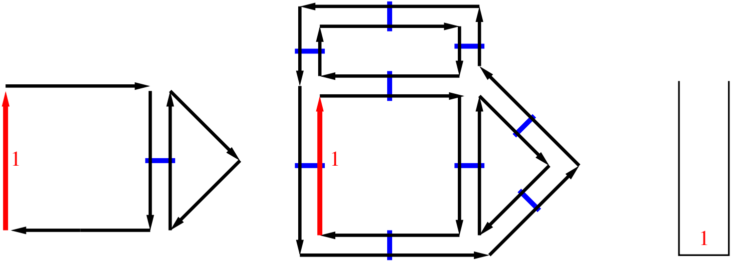


Example with « wrong » initial darts

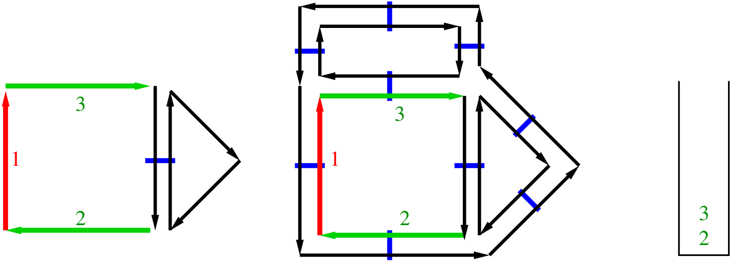


Two different pattern darts are matched to a same target dart
⇝ stop and try another dart

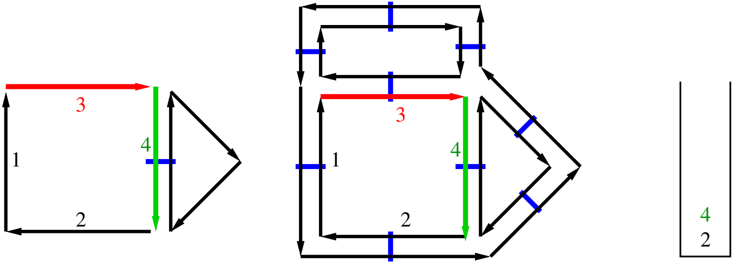
Example with ≪ good ≫ initial darts



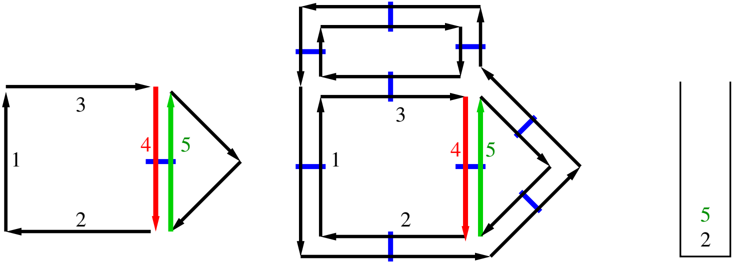
Example with « good » initial darts



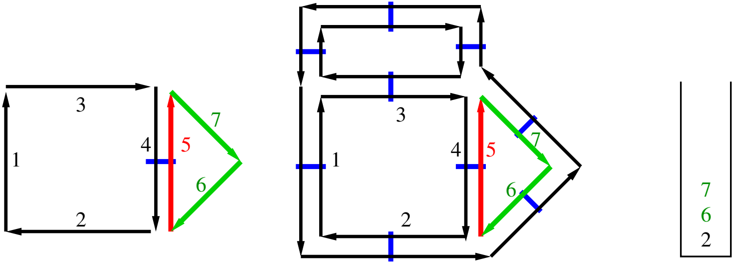
Example with « good » initial darts



Example with ≪ good ≫ initial darts



Example with « good » initial darts



All darts of the pattern are discovered and the matching is a subisomorphism
⇝ testSubIso returns true

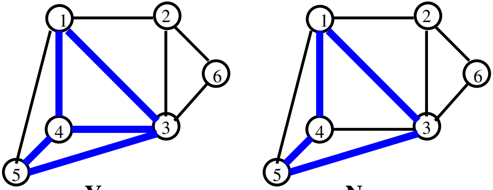
From plane graphs to combinatorial maps (1/2)

...or how to use submap isomorphism to solve some subgraph isomorphism problems...

Compact plane subgraph isomorphism

- Plane graph $\rightsquigarrow$ embedding of a planar graph in a plane
- G_1 and G_2 are plane-isomorphic if there exists a bijection $f : N_1 \rightarrow N_2$ which preserves edges and topology
- G_1 is a compact plane subgraph of G_2 if G_1 is plane isomorphic to a compact subgraph

  $\rightsquigarrow$ remove nodes and edges adjacent to the unbounded face

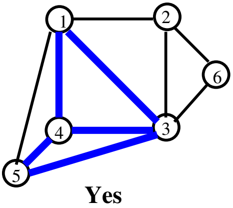


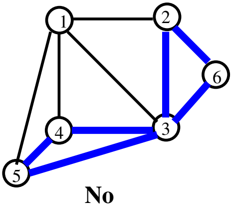
Yes **No**

From plane graphs to combinatorial maps (2/2)

Precondition for using test(Sub)Isomorphism(M, M')

M and M' must be connected
$\rightsquigarrow$ plane graphs must be connected...
...and their unbounded face must be bounded by an elementary cycle



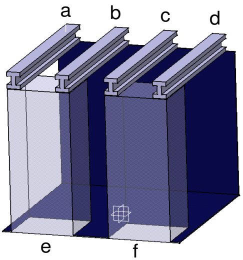


$\rightsquigarrow$ a polynomial algorithm to solve compact plane subgraph isomorphism when unbounded faces are bounded by elementary cycles...

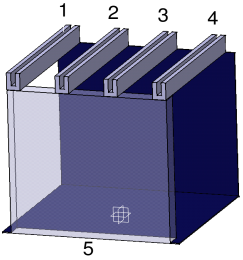
Overview of the talk

- Filtering algorithms for (sub)graph isomorphism
  $\rightsquigarrow$ joint work with Y. Deville, S. Sorlin, and S. Zampelli
- Polynomial algorithm for plane subgraph isomorphism
  $\rightsquigarrow$ joint work with G. Damiand, C. de la Higuera, and J.-C. Janodet
- **Heuristic approaches for multivalent matching problems**
  $\rightsquigarrow$ joint work with P.-A. Champin, O. Sammoud, and S. Sorlin
- Constraint-based graph matching
  $\rightsquigarrow$ joint work with V. le Clément, and Y. Deville

Motivations for multivalent matchings



Object 1



Object 2

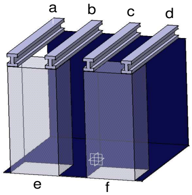
- Allow multivalent matchings
  $\rightsquigarrow$ 'e' and 'f' should be matched to '5'
- Similarity wrt [Tversky 77] : $sim(a, b) = \frac{f(car(a) \cap car(b))}{f(car(a) \cup car(b))}$
  $\rightsquigarrow$ Identify common features

Describing objects by labeled graphs

Let L_V and L_E be sets of node and edge labels
Labeled graph = $\langle V, r_V, r_E \rangle$ such that

- $V \rightsquigarrow$ nodes
- $r_V \subseteq V \times L_V \rightsquigarrow$ nodes labeling
- $r_E \subseteq V \times V \times L_E \rightsquigarrow$ edge labeling

$r_V \cup r_E \rightsquigarrow$ graph features

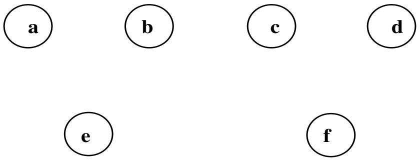


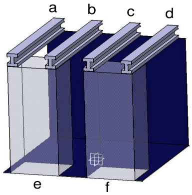
Describing objects by labeled graphs

Let L_V and L_E be sets of node and edge labels
Labeled graph = $\langle V, r_V, r_E \rangle$ such that

- $V \rightsquigarrow$ nodes
- $r_V \subseteq V \times L_V \rightsquigarrow$ nodes labeling
- $r_E \subseteq V \times V \times L_E \rightsquigarrow$ edge labeling

$r_V \cup r_E \rightsquigarrow$ graph features





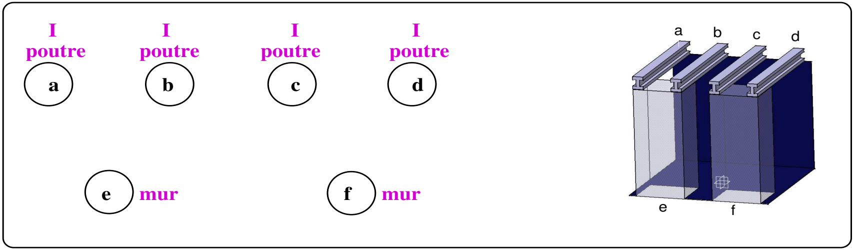
Nodes: $V = \{a, b, c, d, e, f\}$

Describing objects by labeled graphs

Let L_V and L_E be sets of node and edge labels
Labeled graph = $\langle V, r_V, r_E \rangle$ such that

- $V \rightsquigarrow$ nodes
- $r_V \subseteq V \times L_V \rightsquigarrow$ nodes labeling
- $r_E \subseteq V \times V \times L_E \rightsquigarrow$ edge labeling

$r_V \cup r_E \rightsquigarrow$ graph features



Node labeling: $L_V = \{beam, I, wall\}$
$r_V = \{(a, beam), (b, beam), (c, beam), (d, beam),$
$(a, I), (b, I), (c, I), (d, I), (e, wall), (f, wall)\}$

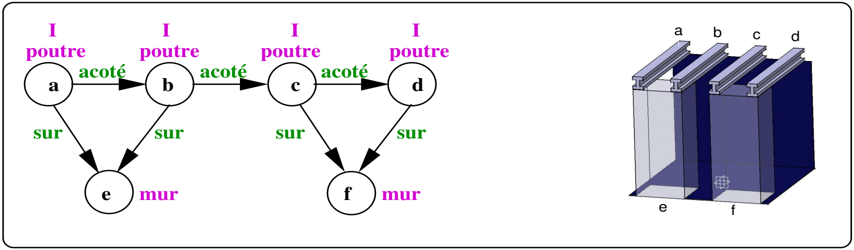
Describing objects by labeled graphs

Let L_V and L_E be sets of node and edge labels

Labeled graph = $\langle V, r_V, r_E \rangle$ such that

- $V \rightsquigarrow$ nodes
- $r_V \subseteq V \times L_V \rightsquigarrow$ nodes labeling
- $r_E \subseteq V \times V \times L_E \rightsquigarrow$ edge labeling

$r_V \cup r_E \rightsquigarrow$ graph features

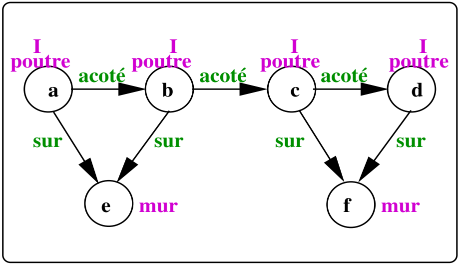


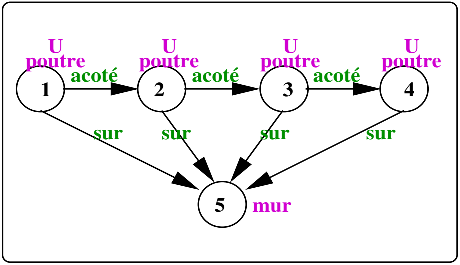
Edge labeling: $L_E = \{next, on\}$

$r_E = \{(a, b, next), (b, c, next), (c, d, next),$
$(a, e, on), (b, e, on), (c, f, on), (d, f, on)\}$

Common features wrt a matching

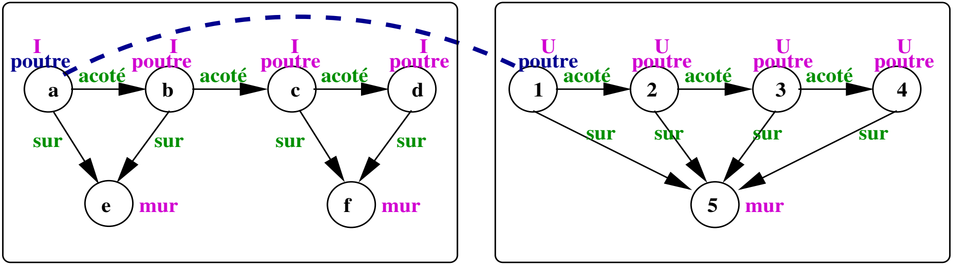
$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$





Common features wrt a matching

$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$

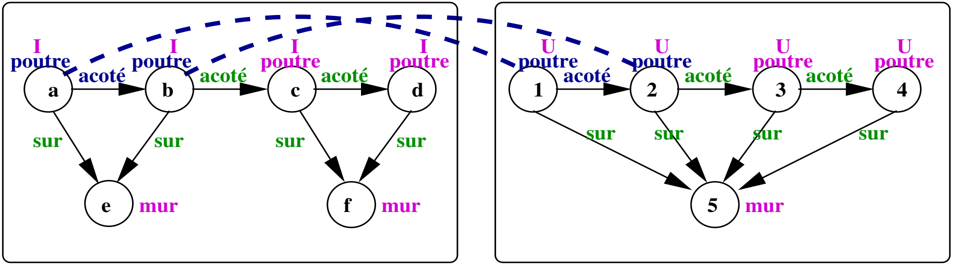


m = {(a,1),

$G_1 \sqcap_m G_2$ = { (a,beam), (1,beam),

Common features wrt a matching

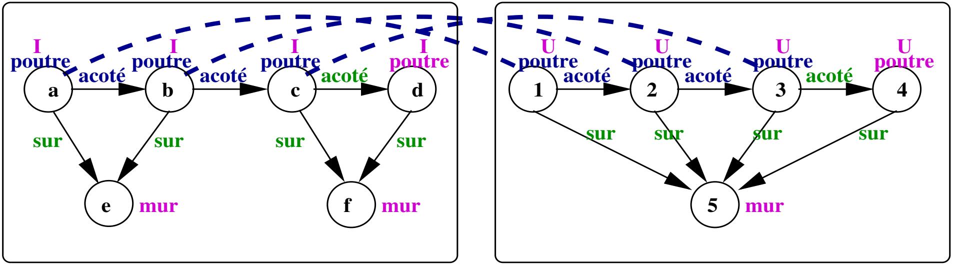
$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$



m = {(a,1), (b,2),
$G_1 \sqcap_m G_2$ = { (a,beam), (1,beam), (b,beam), (2,beam), (a,b,*next*), (1,2,*next*),

Common features wrt a matching

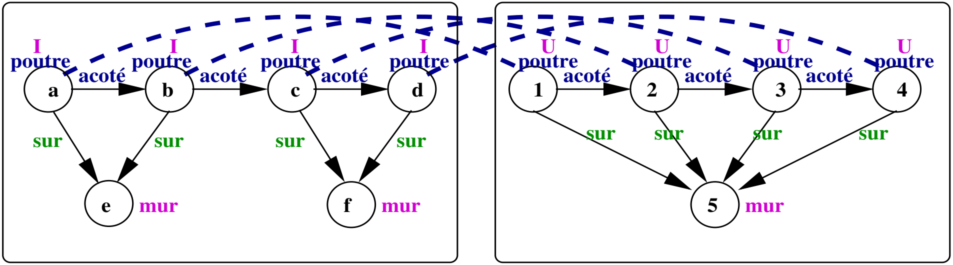
$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$



m = {(a,1), (b,2), (c,3),
$G_1 \sqcap_m G_2$ = { (a,beam), (1,beam), (b,beam), (2,beam), (a,b,*next*), (1,2,*next*), (c,beam), (3,beam), (b,c,*next*), (2,3,*next*),

Common features wrt a matching

$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$

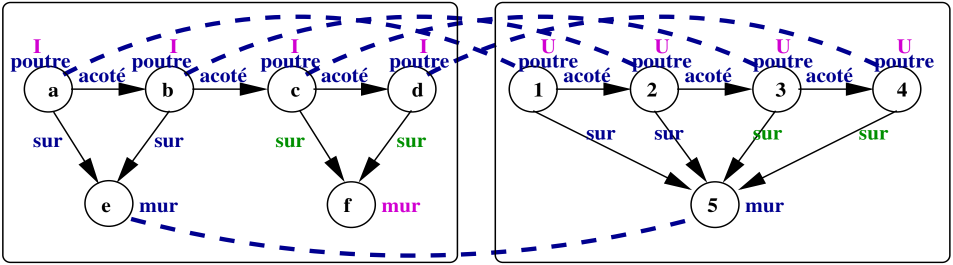


m = {(a,1), (b,2), (c,3), (d,4),

$G_1 \sqcap_m G_2$ = { (a,beam), (1,beam), (b,beam), (2,beam), (a,b,*next*), (1,2,*next*), (c,beam), (3,beam), (b,c,*next*), (2,3,*next*), (d,beam), (4,beam), (c,d,*next*), (3,4,*next*),

Common features wrt a matching

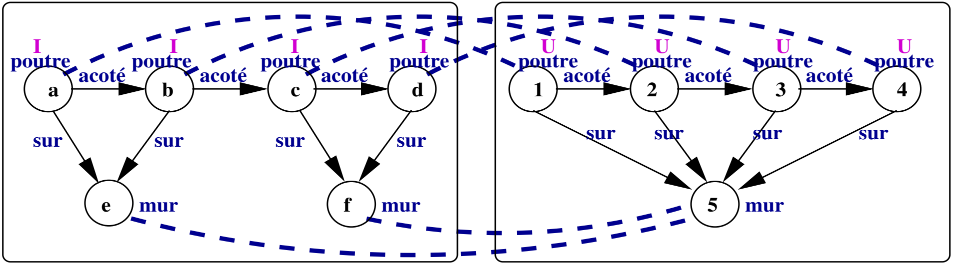
$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$



m = {(a,1), (b,2), (c,3), (d,4), (e,5),
$G_1 \sqcap_m G_2$ = { (a,beam), (1,beam), (b,beam), (2,beam), (a,b,*next*), (1,2,*next*), (c,beam), (3,beam), (b,c,*next*), (2,3,*next*), (d,beam), (4,beam), (c,d,*next*), (3,4,*next*), (e,wall), (5,wall), (a,e,on), (b,e,on), (1,5,on), (2,5,on),

Common features wrt a matching

$$G_1 \sqcap_m G_2 = \{c \in r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2} / c \text{ common to } G_1 \text{ and } G_2 \text{ via } m\}$$



m = {(a,1), (b,2), (c,3), (d,4), (e,5), (f,5)}

$G_1 \sqcap_m G_2$ = { (a,beam), (1,beam), (b,beam), (2,beam), (a,b,*next*), (1,2,*next*), (c,beam), (3,beam), (b,c,*next*), (2,3,*next*), (d,beam), (4,beam), (c,d,*next*), (3,4,*next*), (e,wall), (5,wall), (a,e,on), (b,e,on), (1,5,on), (2,5,on), (f,wall), (c,f,on), (d,f,on), (3,5,on), (4,5,on)}

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

$G_1 \sqcap_m G_2$ = features common to G_1 and G_2 via m

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

$G_1 \sqcap_m G_2$ = features common to G_1 and G_2 via m

$r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2}$ = set of all features of G_1 and G_2

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

$G_1 \sqcap_m G_2$ = features common to G_1 and G_2 via m

$r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2}$ = set of all features of G_1 and G_2

f = function that quantifies features

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

$G_1 \sqcap_m G_2$ = features common to G_1 and G_2 via m
$r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2}$ = set of all features of G_1 and G_2
f = function that quantifies features
$splits(m)$ = set of nodes that are matched to more than one node

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

$G_1 \sqcap_m G_2$ = features common to G_1 and G_2 via m
$r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2}$ = set of all features of G_1 and G_2
f = function that quantifies features
$splits(m)$ = set of nodes that are matched to more than one node
g = function that quantifies splits

Similarity of 2 graphs

Similarity of G_1 and G_2 induced by a matching m

$$sim_m(G_1, G_2) = \frac{f(G_1 \sqcap_m G_2) - g(splits(m))}{f(r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2})}$$

$G_1 \sqcap_m G_2$ = features common to G_1 and G_2 via m
$r_{V_1} \cup r_{E_1} \cup r_{V_2} \cup r_{E_2}$ = set of all features of G_1 and G_2
f = function that quantifies features
$splits(m)$ = set of nodes that are matched to more than one node
g = function that quantifies splits

Similarity of G_1 and G_2

$$sim(G_1, G_2) = max_{m \subseteq V_1 \times V_2} sim_m(G_1, G_2)$$

Measuring the similarity of G_1 and G_2 $\leadsto$ find $m \subseteq V_1 \times V_2$ that maximizes $score(m) = f(G_1 \sqcap_m G_2) - g(splits(m))$

Computing the similarity of two graphs

A very hard problem...

- Goal = find $m \subseteq V_1 \times V_2$ that maximizes $score(m)$
- $\mathcal{NP}$-hard problem $\rightsquigarrow 2^{|V_1| \cdot |V_2|}$ combinations

Heuristic approaches

- Greedy: quickly build a rather good matching
- Tabu: iteratively improves a matching by local perturbations
- ACO: use pheromone to guide greedy constructions

Computing the similarity of two graphs

A very hard problem...

- Goal = find $m \subseteq V_1 \times V_2$ that maximizes $score(m)$
- $\mathcal{NP}$-hard problem $\rightsquigarrow 2^{|V_1| \cdot |V_2|}$ combinations

Heuristic approaches

- Greedy: quickly build a rather good matching
- Tabu: iteratively improves a matching by local perturbations
- ACO: use pheromone to guide greedy constructions

Greedy algorithm

Greedy construction of a matching m

- $m \leftarrow \emptyset$
- **Iterate**
  - $Cand \leftarrow V_1 \times V_2 - m$
  - Choose $(u_1, u_2) \in Cand$ that maximizes *score*
    $\leadsto$ break ties with a look-ahead function
- **Exit when** $score(m \cup \{(u_1, u_2)\}) < score(m)$
  - $m \leftarrow m \cup \{(u_1, u_2)\}$
- **End iterate**

Properties

- Polynomial complexity $\mathcal{O}((|V_1| \cdot |V_2|)^2)$
- Non optimal
- Non deterministic $\leadsto$ may be iterated

Reactive tabu search

Exploration of the neighborhood of a matching m

- $m \leftarrow Greedy(G_1, G_2)$
- **While** termination condition not reached
  - Choose $m' \in Neighborhood(m)$ such that
    - Moving from m to m' isn't "Tabu"
    - m' maximizes the score function
  - $m \leftarrow m'$
  - Make the move from m' to m "Tabu"
- **End while**

Reactive tabu search

Exploration of the neighborhood of a matching m

- $m \leftarrow Greedy(G_1, G_2)$
- **While** termination condition not reached
  - Choose $m' \in$ *Neighborhood*(m) such that
    - Moving from m to m' isn't "Tabu"
    - m' maximizes the score function
  - $m \leftarrow m'$
  - Make the move from m' to m "Tabu"
- **End while**

Neighborhood(m) = matchings obtained by adding or removing a couple of nodes to m

Reactive tabu search

Exploration of the neighborhood of a matching m

- $m \leftarrow Greedy(G_1, G_2)$
- **While** termination condition not reached
  - Choose $m' \in Neighborhood(m)$ such that
    - Moving from m to m' isn't "Tabu"
    - m' maximizes the score function
  - $m \leftarrow m'$
  - Make the move from m' to m "Tabu"
- **End while**

Tabu principle $\rightsquigarrow$ Prevent the search from cycling

- Memorize the k last moves in a tabu list
- k determines the intensification/diversification balance
  - Decrease $k \rightsquigarrow$ Intensify
  - Increase $k \rightsquigarrow$ Diversify
- Reactive search $\rightsquigarrow$ Dynamically adjust k

ACO algorithm

Use pheromone to learn for good matchings

$\tau(u_1, u_2)$ = past experience wrt matching u_1 with u_2

Greedy construction of a matching m

- $m \leftarrow \emptyset$
- **While** m can be improved
  - $Cand \leftarrow \{(u_1, u_2)$ that improve $m\}$
  - Choose $(u_1, u_2) \in Cand$ / proba. depending on
    - Pheromone factor $\leadsto$ past experience of the colony
    - Heuristic factor $\leadsto$ *score* function
  - $m \leftarrow m \cup \{(u_1, u_2)\}$

Pheromone updating step

Every *nbAnts* constructions:

- Evaporate (multiply by $\rho \in]0; 1[$)
- Reward the best matching found

Experimental comparison

Benchmarks

- Test suite : randomly generated instances
- Test suite 2: Instances of [Boeres et al. 2004]

Conclusion

- For short CPU time limits: Tabu is better
- For longer CPU time limits: ACO is (slightly) better
- Both approaches are rather ”robust”

Overview of the talk

- Filtering algorithms for (sub)graph isomorphism
  $\rightsquigarrow$ joint work with Y. Deville, S. Sorlin, and S. Zampelli
- Polynomial algorithm for plane subgraph isomorphism
  $\rightsquigarrow$ joint work with G. Damiand, C. de la Higuera, and J.-C. Janodet
- Heuristic approaches for multivalent matching problems
  $\rightsquigarrow$ joint work with P.-A. Champin, O. Sammoud, and S. Sorlin
- **Constraint-based graph matching**
  $\rightsquigarrow$ joint work with V. le Clément, and Y. Deville

Motivation

Dedicated matching algorithms

Customized algorithm to solve a specific problem: efficient...
but cannot be used to solve a slightly different matching problem

Generic matching algorithms

May be used to solve any matching problem... but not always as efficient as dedicated approaches for specific matching problems

Motivation

Dedicated matching algorithms

Customized algorithm to solve a specific problem: efficient...
but cannot be used to solve a slightly different matching problem

Constraint-based graph matching

- a high level modeling language for graph matching
- a synthesizer that generates an efficient algorithm from the model

$\rightsquigarrow$ reuse state-of-the-art approaches, combine them, ...

Generic matching algorithms

May be used to solve any matching problem... but not always as efficient as dedicated approaches for specific matching problems

Characteristics of our approach

Written in Comet

- Supports both CP, CBLS, and MIP
- Object-Oriented

Easy to use as a black-box

- Easy modeling of classical problems
- May be used to model new problems
  $\leadsto$ Handling specificities through additional constraints

The box may be opened and is easily extensible

- Add new constraints
- Add new solving algorithms, heuristics

$\leadsto$ Extend the synthesizer

Modeling language for graph matching

Constraints on the cardinality of the matching

bijective (1,1), injective (1,0..1), univalent (0..1,0..1), or multivalent (0..n,0..n)

- hard constraints: must be satisfied
- soft constraints: should be satisfied as much as possible

Constraints on edges

- hard constraints: edges must be matched
- soft constraints: maximize the number of matched edges

Constraints on labels (in case of labeled graphs))

- hard: matched components must have identical labels
- soft: maximize the similarity of matched component labels

Example 1: Graph isomorphism

- Declare 2 graph objects `g1` and `g2` and a matching `m`

```
bool[,] adj1 = ...
bool[,] adj2 = ...
SimpleGraph<Mod> g1(adj1);
SimpleGraph<Mod> g2(adj2);
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on `m` $\rightsquigarrow$ bijective matching $(1, 1)$

```
m.post(cardMatch(g1.getAllNodes(), 1, 1));
m.post(cardMatch(g2.getAllNodes(), 1, 1));
```

- Post constraints to ensure edge matching

```
m.post(matchedToSomeEdges(g1.getAllEdges()));
m.post(matchedToSomeEdges(g2.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```
m.close();
DefaultGMSynthesizer synth();
GMSolution<Mod> sol = synth.solveMatching(m);
```

Example 1: Graph isomorphism

- Declare 2 graph objects `g1` and `g2` and a matching `m`

```
bool[,] adj1 = ...
bool[,] adj2 = ...
SimpleGraph<Mod> g1(adj1);
SimpleGraph<Mod> g2(adj2);
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on `m` $\leadsto$ bijective matching $(1, 1)$

```
m.post(cardMatch(g1.getAllNodes(), 1, 1));
m.post(cardMatch(g2.getAllNodes(), 1, 1));
```

- Post constraints to ensure edge matching

```
m.post(matchedToSomeEdges(g1.getAllEdges()));
m.post(matchedToSomeEdges(g2.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```
m.close();
DefaultGMSynthesizer synth();
GMSolution<Mod> sol = synth.solveMatching(m);
```

Example 1: Graph isomorphism

- Declare 2 graph objects `g1` and `g2` and a matching `m`

```
bool[,] adj1 = ...
bool[,] adj2 = ...
SimpleGraph<Mod> g1(adj1);
SimpleGraph<Mod> g2(adj2);
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on `m` $\rightsquigarrow$ bijective matching $(1, 1)$

```
m.post(cardMatch(g1.getAllNodes(), 1, 1));
m.post(cardMatch(g2.getAllNodes(), 1, 1));
```

- Post constraints to ensure edge matching

```
m.post(matchedToSomeEdges(g1.getAllEdges()));
m.post(matchedToSomeEdges(g2.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```
m.close();
DefaultGMSynthesizer synth();
GMSolution<Mod> sol = synth.solveMatching(m);
```

Example 1: Graph isomorphism

- Declare 2 graph objects `g1` and `g2` and a matching `m`

```
bool[,] adj1 = ...
bool[,] adj2 = ...
SimpleGraph<Mod> g1(adj1);
SimpleGraph<Mod> g2(adj2);
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on `m` $\rightsquigarrow$ bijective matching $(1, 1)$

```
m.post(cardMatch(g1.getAllNodes(), 1, 1));
m.post(cardMatch(g2.getAllNodes(), 1, 1));
```

- Post constraints to ensure edge matching

```
m.post(matchedToSomeEdges(g1.getAllEdges()));
m.post(matchedToSomeEdges(g2.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```
m.close();
DefaultGMSynthesizer synth();
GMSolution<Mod> sol = synth.solveMatching(m);
```

Example 2: Induced Subgraph Isomorphism

- Declare 2 graph objects `g1` and `g2` and a matching `m`

```
bool[,] adj1 = ...
bool[,] adj2 = ...
SimpleGraph<Mod> g1(adj1);
SimpleGraph<Mod> g2(adj2);
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on `m` $\rightsquigarrow$ injective matching $(1, 0..1)$

```
m.post(cardMatch(g1.getAllNodes(), 1, 1));
m.post(cardMatch(g2.getAllNodes(), 0, 1));
```

- Post constraints to ensure edges of G_1 to be matched

```
m.post(matchedToSomeEdges(g1.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```
m.close();
DefaultGMSynthesizer synth();
GMSolution<Mod> sol = synth.solveMatching(m);
```

Example 3: Largest Common Induced Subgraph

- Declare 2 graph objects `g1` and `g2` and a matching `m`

```
bool[,] adj1 = ...
bool[,] adj2 = ...
SimpleGraph<Mod> g1(adj1);
SimpleGraph<Mod> g2(adj2);
Matching<Mod> m(g1,g2);
```

- Post cardinality constraints on `m` $\rightsquigarrow (0..1, 0..1)$

```
m.post(cardMatch(g1.getAllNodes(), 0, 1));
m.post(cardMatch(g2.getAllNodes(), 0, 1));
```

- Post a soft constraint to maximize the nb of matched vertices

```
m.softpost(minMatch(g1.getAllNodes(), 1), 1)
```

- Post constraints to ensure edge matching

```
m.post(matchedToAllEdges(g1.getAllEdges()));
m.post(matchedToAllEdges(g2.getAllEdges()));
```

- Ask the synthesizer to create the solver... and search a solution

```
m.close(); DefaultGMSynthesizer synth();
```

Synthesizing a solver for graph matching problems (1/3)

Warning: Ongoing research with a very first prototype
$\rightsquigarrow$ many improvements are still to be done !

Canonical form of modeling constraints

Aggregate all modeling constraints of a same type

- Cardinality (MinMatch, MaxMatch, CardMatch, ...)
- Edge matching (MatchedToSomeEdges, MatchedToAllEdges, ...)
- Label matching (MatchAllNodeLabels, MatchAllEdgeLabels, ...)

$\rightsquigarrow$ Derive characteristics from the canonical model

Choose a search approach

- CP if no soft constraints and MaxCard $\leq$ 1 for all nodes of a graph
- CBLS otherwise

Synthesizing a solver for graph matching problems (2/3)

Creation of low level variables

Associate a variable with every vertex of both graphs

- Domains are defined wrt cardinality constraints

MinMatch	MaxMatch	Type	Domain
1	1	int	N
0	1	int	$N \cup \{\bot\}$
Otherwise		set	2^N

- Ensure symmetry (X_u matched to $v \Rightarrow X_v$ matched to u):
    - CP $\rightsquigarrow$ Channeling constraints
    - CBLS $\rightsquigarrow$ invariants

Synthesizing a solver for graph matching problems (3/3)

Post the canonical constraints

- CP (hard constraints only)
  - Cardinality constraints
    $\rightsquigarrow$ Partly handled by variable domains
    $\rightsquigarrow$ Global allDiff for injective and bijective matchings
  - Edge constraints $\rightsquigarrow$ binary constraints
  - Label constraints on nodes $\rightsquigarrow$ variable domains
  - Label constraints on edges $\rightsquigarrow$ binary constraints
- CBLS (hard and soft constraints)
  - Cardinality $\rightsquigarrow$ neighborhood if hard; invariants if soft
  - Edge $\rightsquigarrow$ invariants
  - Node labels $\rightsquigarrow$ neighborhood if hard; invariants if soft
  - Edge labels $\rightsquigarrow$ invariants

(Preliminary) Experimental Results (1/2)

$\mathcal{SI} \rightsquigarrow$ Subgraph Isomorphism

#N	Synthesizer/CP					vf2 [Cordella et al. 99]				
	5%	10%	20%	33%	50%	5%	10%	20%	33%	50%
100	0.8	0.5	0.7	0.1	0.2	0.0	0.0	0.0	2.0	0.0
500	19.3	4.7	10.5	15.8	30.7	0.1	0.1	246.7	192.3	–
1000	30.6	595.8	119.0	152.3	–	86.7	–	–	–	–

- Vf2 better for small instances
- Synthesizer outperforms vf2 for larger instances
- Additional constraint improves the search process

(Preliminary) Experimental Results (1/2)

$\mathcal{SI}$ ⇝ Subgraph Isomorphism
$\mathcal{SI}+$ ⇝ Subgraph Isomorphism + additional distance constraint

#N	Synthesizer/CP					vf2 [Cordella et al. 99]				
	5%	10%	20%	33%	50%	5%	10%	20%	33%	50%
100	0.8	0.5	0.7	0.1	0.2	0.0	0.0	0.0	2.0	0.0
500	19.3	4.7	10.5	15.8	30.7	0.1	0.1	246.7	192.3	–
1000	30.6	595.8	119.0	152.3	–	86.7	–	–	–	–
100	0.3	0.1	0.1	0.1	0.2					
500	3.0	4.4	9.5	16.9	28.9					
1000	16.1	47.8	82.5	148.0	–					

- Vf2 better for small instances
- Synthesizer outperforms vf2 for larger instances
- Additional constraint improves the search process

(Preliminary) Experimental Results (2/2)

Maximum common subgraph $\rightsquigarrow$ CBLS

#nodes	time		iterations		edges%	
25	8.5	2.5	7768.1	2301.3	48.3	1.1
50	33.9	10.7	8023.8	2543.3	40.2	0.5
100	141.5	46.4	8398.4	2755.0	34.5	0.2

- First results to assess feasibility
- Complete approaches cannot handle these instances
- We haven't (yet) compared these results with other approaches

Further works on modeling for graph matching

- Improve the analysis of the matching characteristics
  $\rightsquigarrow$ identify sub-problems that are "easy" to solve
- Integrate dedicated filtering algorithms $\rightsquigarrow$ CP
  - Iterative partitionning for graph isomorphism (Nauty)
  - Iterative labeling for subgraph iso. (Zampelli et al 2009)
- Integrate reactive search and other meta-heuristics for CBLS
  $\rightsquigarrow$ Parameter tuning... !
- Combine CP and CBLS

Graph similarity measures

- H. Bunke and X. Jiang: Graph matching and similarity, chapitre de “Intelligent systems and interfaces”, Kluwer, 2000
- R. Ambauen, S. Fishe, and H. Bunke : Graph edit distance with node splitting and merging and its application to diatom identification, Workshop on Graph-based representation in PR, LNCS 2726:95-106, Springer, 2003
- P.-A. Champin and C. Solnon : Measuring the similarity of labeled graphs, Int. Conf. on Case-Based Reasoning, LNCS 2689:80-95, Springer, 2003
- M. Boeres, C. Ribeiro, and I. Bloch : A randomized heuristic for scene recognition by graph matching, Workshop on Experimental Algorithms, 100-113, 2004
- S. Sorlin, C. Solnon, and J.-M. Jolion : A Generic Graph Distance Measure Based on Multivalent Matchings, chapitre de “Applied Graph Theory in Pattern Recog.”, Vol 52:151-182, Springer, 2007

(Sub)graph isomorphism

- B. D. McKay : Practical Graph Isomorphism, Congressus Numerantium, 30:45-87, 1981
- S. Sorlin and C. Solnon: A parametric filtering algorithm for the graph isomorphism problem, Constraints 13(4):518-537, 2008
- L.P. Cordella, P. Foggia, C. Sansone, and M. Vento : An Improved Algorithm for Matching Large Graphs, in GbR, pages 149-159, 2001
- L. Larrosa et G. Valiente : Constraint satisfaction algorithms for graph pattern matching, Math Structures in Computer Science 12(4):403-422, 2002
- S. Zampelli, Y. Deville, C. Solnon: Solving subgraph isomorphism problems with constraint programming, Constraints (to appear)
- G. Damiand, C. de la Higuera, J.C. Janodet, E. Samuel, and C. Solnon: A Polynomial Algorithm for Submap Isomorphism : Application to searching patterns in images, in GbR, LNCS 5534, pp102-112, Springer, 2009

Heuristic algorithms and Constraint-based graph matching

- M. Boeres, C. Ribeiro and I. Bloch: A randomized heuristic for scene recognition by graph matching, Workshop on Experimental Algorithms, 100-113, 2004
- P.A. Champin and C. Solnon: Measuring the similarity of labeled graphs, ICCBR, LNAI 2689:80-95, 2003
- S. Sorlin and C. Solnon: Reactive Tabu Search for Measuring Graph Similarity, GbR, LNCS 3434:172-182, 2005
- O. Sammoud, C. Solnon and K. Ghédira : Ant Algorithm for the Graph Matching Problem, EvoCOP, LNCS 3448:213-223, 2005
- V. le Clément, Y. Deville, and C. Solnon: Constraint-based Graph Matching, CP, LNCS 5732: 274:288, 2009