Logic programming and object reconstruction

A. Dal Palù  G. De Gasperis  A. Dovier  A. Formisano
A. Fusiello  M. Gavanelli  R. Gentilini  E. Omodeo

Convegno 2018 del Gruppo Nazionale Calcolo Scientifico
Montecatini Terme, 14-16 February, 2018
Object reconstruction

A general term for a large number of variations, problems, and applications

- solving edge-matching puzzles, jigsaw puzzles, packing puzzles, ...
- assembling torn documents
- restoring fragmented wall paintings
- rebuild broken pottery
- 3D scene reconstruction from images
- ...etc...
Object reconstruction

The emphasis might be placed on different aspects/features

- shape of fragments (regular or irregular, homogeneous or not)
- knowledge about specific features of the fragments/object, e.g. presence of pictorial information, chromatic info, texture, orientation, topological info,...
- possible overlapping
- connectedness (missing fragments?)
- extraneous fragments
- gaps between pieces’ edges (e.g. damaged border, erosion, ...)
- known/unknown map
- ...
Object reconstruction

Similar problems: tiling problems; nesting/packing problems; spatial reasoning; cutting problems; ...

Large literature and many approaches

- dynamic programming
- greedy algorithms
- integer (quadratic) programming
- genetic algorithms
- curve/surface/shape matching
- hierarchical search strategies and local search
- ...

mainly exploiting numerical and analytic techniques ...
A declarative approach?

- higher-level knowledge representation
- focus on modeling (not on algorithms)
- elaboration tolerance and incremental modeling
- admits incompleteness and uncertainty in modeling (errors, approximations, imprecise measures or extracted features, ...)
- quantitative and qualitative descriptions
- constraints and preferences
- ...

All in one: Answer set programming
ASP in a nutshell

**Syntax**: in its simplest form an ASP program is a (finite) set of rules:

\[ B_0 \leftarrow B_1, \ldots, B_n, \text{not } B_{n+1}, \ldots, \text{not } B_{n+m}. \]

Each \( B_i \) is a propositional atom, \( p(x_1, \ldots, x_h) \) asserting \( x_1, \ldots, x_h \) satisfy the property \( p \) (Rmk.: Variables used as shorthand for constants)

Some useful extensions and special cases

- **fact** (if \( n = m = 0 \)) and **constraint** (if \( B_0 \) is missing)
- **weight literals** may occur in rules. E.g., \( \ell \{ w_1:L_1, \ldots, w_k:L_k \} \) \( u \)
- **aggregates** may occur in rules. E.g., \( \#max \{ w_1:L_1, \ldots, w_k:L_k \} \)
- **preferences**, **weak constraints**, **optimization directives**, ...

**Semantics**: minimal, closed, and supported propositional models (set of atoms)

**Common approach in modeling**: represent a problem as a program whose models encode the solutions of the problem

A **solver** computes the solutions/models of the ASP-specification
ASP-encoding of reconstruction problems

Let’s focus on a specific variant of the problem and make some assumptions:

- assembling torn documents (hence, 2D)
- fragments are polygons
- each edge of a fragment may be characterized by some features (pictorial info, texture, ...)
  W.l.o.g., we can represent all these features by associating a color
- the edges composing the external boundary are known

Moreover, for the time being,

- no missing or extraneous fragments
- no gaps or eroded edges
Representation of fragments

- A polygon is represented by the ordered sequence of its vertices (in clockwise order)
- Vertices are identified with their position in such sequence (edges are ordered/identified accordingly)
- Spatial positions of vertices is described as relative displacements w.r.t. the first vertex (i.e., vertex 1 in the ordered sequence)

Moreover,

- Each edge may have a “color” specified
- Admissible rotations may be specified for each polygon
- No overlapping is admitted
Representation of fragments: example

% poly(PId, Vnum, Xoff, Yoff)
  poly(p1, 1, 0, 0).
  color(p1, 2, blue).
  poly(p1, 2, 120, 100).
  color(p1, 4, red).
  poly(p1, 3, 254, 14).
  external_side(p1, 3).
  poly(p1, 4, 228, -110).
  angle(p1, 0..90).
  poly(p1, 5, -120, -126).
  poly(p1, 6, -176, 68).
Plainly, in the reconstructed object

- each first vertex will be placed at certain coordinates w.r.t. a pair of orthogonal axis
- to break symmetries, fix vertex 1 of one polygon (say $p_1$) in $(0, 0)$
- each fragment may be rotated w.r.t. its input description
- fragments’ edges must match (also taking features/colors into account)
- fragments are rigid and cannot overlap
ASP specification: rotations

Selection of one rotation:

\[ \{ \text{rotation(Pid, Angle)} : \text{angle(Pid, Angle)} \} \] 1 :- polyId(Pid).
polyId(Pid) :- poly(Pid, _, _, _, _).

For each polygon Pid exactly one among its possible rotations is selected. Then, the rotation is performed (using integer arithmetic):

\[
\text{poly_rotR}(P, 1, Xs, Ys) :- \text{rotation}(P, Angle), \text{poly}(P, 1, Xs, Ys).
\]
\[
\text{poly_rotR}(P, V, Xv, Yv) :- \text{rotation}(P, Angle), V > 1, \text{poly}(P, V, Xs, Ys), \text{cosSinOfAngle}(Angle, Cos, Sin), Xv = (Xs \times \text{Cos}) - (Ys \times \text{Sin}), Yv = (Xs \times \text{Sin}) + (Ys \times \text{Cos}).
\]

\[
\text{cosSinOfAngle}(Angle, Cos, Sin) :- \ldots
\]
ASP specification: edges matchings

Two edges can match if their features/color are compatible and they are parallel (in the opposite direction w.r.t. the order of vertices):

 compatible_sides(Pid1, L1, Pid2, L2) :-
       compatibleFeatures(Pid1,L1,Pid2,L2),
       inner_side(Pid1,L1), inner_side(Pid2,L2),
       opposite(Pid1, L1, Pid2, L2).

 compatibleFeatures(P1,L1,P2,L2) :- ...

 segment(P,N, X1,Y1, X2, Y2) :- succSide(P,N,M),
       poly_rotR(P,N,X1,Y1), poly_rotR(P,M,X2,Y2).

 opposite(Pid1, S1, Pid2, S2) :- Pid1 < Pid2,
       segment(Pid1, S1, P1x, P1y, Q1x, Q1y),
       segment(Pid2, S2, P2x, P2y, Q2x, Q2y),
       (Q1x-P1x)*(P2y-Q2y) = (P2x-Q2x)*(Q1y-P1y).

(simplified)
As concerns the selection among possible matchings, we exploit cardinality constraints. For instance, to impose a 1-to-1 correspondence:

\[
1 \{ \text{selected\_matching}(P1,L1,P2,L2) \\
    : \text{compatible\_sides}(P1,L1,P2,L2) \} 1 : - \\
    \text{internal\_side}(P1,L1).
\]

\%L1 of P1 cannot match with both
\% L2 and L3 from the same polygon P2:
\:- \text{selected\_matching}(P1,L1,P2,L2), \\
    \text{selected\_matching}(P1,L1,P2,L3), L2!=L3.
\:- ...(simplified)...

...and similarly to express a 1-to-n correspondence (plus constraints on directions/length of matched edges)
ASP specification: placement

Placement of polygons is specified as follows:

First, the vertex 1 of each polygon is placed:

poly_cooS1(1, 0, 0). % first polygon in (0,0)
poly_cooS1(P2, X, Y) :- P1<P2, succSide(P2, L2, L2suc),
selected_matching(P1, L1, P2, L2),
poly_rotR(P2, L2suc, XL2, YL2),
poly_rotR(P1, L1, XL1, YL1),
poly_cooS1(P1, X0, Y0),
X=X0+XL1-XL2, Y=Y0+YL1-YL2.

% to impose uniqueness:
:- poly_cooS1(Pid, X1, _), poly_cooS1(Pid, X2, _), X1<X2.
:- poly_cooS1(Pid, _, Y1), poly_cooS1(Pid, _, Y2), Y1<Y2.
The candidate solutions is obtained by propagating the location to all other vertices:

\[
\text{solution}(\text{Pid}, 1, X, Y) :\leftarrow \text{poly_cooS1}(\text{Pid}, X, Y).
\]

\[
\text{solution}(\text{Pid}, L, X, Y) :\leftarrow \text{polyId}(\text{Pid}), \ L>1,
\text{poly_cooS1}(\text{Pid}, X1, Y1), \\
\text{poly_rotR}(\text{Pid}, L, Xrot, Yrot), \\
X = Xrot+X1, \\
Y = Yrot+Y1.
\]
Avoiding crossing edges.
direction/4 establishes whether \((X, Y)\) is on the left/right side of the line containing segment \(S\):

\[
direction(Pid, S, X, Y, D) :- \text{segment}(Pid, S, P_x, P_y, Q_x, Q_y), D=(((X-P_x)*(Q_y-P_y)) - ((Q_x-P_x)*(Y-P_y))).
\]

Impose that any two edges (\(S_1--S_{1b}\) of \(Pid_1\) and \(S_2--S_{2b}\) of \(Pid_2\)) cannot cross each other:

\[
:- Pid_1 < Pid_2, \ %\text{to break symmetry}
\text{solution}(Pid_1, S_1, P_{1x}, P_{1y}), \text{solution}(Pid_1, S_{1b}, Q_{1x}, Q_{1y}),
\text{solution}(Pid_2, S_2, P_{2x}, P_{2y}), \text{solution}(Pid_2, S_{2b}, Q_{2x}, Q_{2y}),
\text{succSide}(Pid_1, S_1, S_{1b}), \text{succSide}(Pid_2, S_2, S_{2b}),
direction(Pid_2, S_2, P_{1x}, P_{1y}, D_1),
direction(Pid_2, S_2, Q_{1x}, Q_{1y}, D_2), D_1*D_2<0,
direction(Pid_1, S_1, P_{2x}, P_{2y}, D_3),
direction(Pid_1, S_1, Q_{2x}, Q_{2y}, D_4), D_3*D_4<0.
\]
Dealing with incompleteness and uncertainty

To admit matching between “almost-parallel” edges, one may introduce qualitative/quantitative conditions (e.g., thresholds,...). For instance:

\[
\text{opposite}(\text{Pid}_1, S_1, \text{Pid}_2, S_2) :- \ Pid_1 < \text{Pid}_2, \\
\quad \text{segment}(\text{Pid}_1, S_1, P_1x, P_1y, Q_1x, Q_1y), \\
\quad \text{segment}(\text{Pid}_2, S_2, P_2x, P_2y, Q_2x, Q_2y), \\
\quad T = (Q_1x-P_1x)*(P_2y-Q_2y) - (P_2x-Q_2x)*(Q_1y-P_1y), \\
\quad \text{acceptCondition}(S_1,S_2,T),
\]

\[
\text{acceptCondition}(\text{Pid}_1, S_1, \text{Pid}_2, S_2, T) :- \ldots \\
\quad \% \text{specify acceptability conditions,}... \\
\]

Tolerances in dealing with features? It suffices to modify the definition of

\[
\text{compatibleFeatures}(P_1,L_1,P_2,L_2) :- \ldots 
\]
Dealing with incompleteness and uncertainty

To deal with missing fragments, one admits that some edge may not match any other edge. For instance:

\[
0 \{ \text{selected\_matching}(P1,L1,P2,L2) \\
    : \text{compatible\_sides}(P1,L1,P2,L2) \} 1 :- \\
    \text{internal\_side}(P1,L1).
\]

and then search for the solution that minimizes the number of “holes”:

\[
\text{matched}(P,L) :- \text{selected\_matching}(P,L,_,_).
\]
\[
\text{matched}(P,L) :- \text{selected\_matching}(_,_,P,L).
\]
\[
\text{unmatched}(P,L) :- \text{not matched}(P,L).
\]

\[
\#\text{minimize } \{ 1,P,L: \text{unmatched}(P,L) \}.
\]

(or similarly to minimize the overall area of “holes”, or ...)

Note that extraneous fragments can be dealt with in an analogous manner.
Ongoing work...

- Web-oriented tool for specification and automated object reconstruction (a prototype GUI with minimal functionalities exists)
- Integration with external computational services or with CSP-solvers
- Feature extraction (currently, done by hand...)
- ASP-code optimization
- Integrate heuristics and techniques from ADL
- Integration with numerical approaches!
- ...
Acknowledgement to GNCS-INdAM


Acknowledgement to GNCS-INdAM


