Global Learning of Typed Entailment Rules
(Berant et al., 2011)

Stefan Grünewald

Overview

1) Entailment Rules
2) Graph-theoretic Approach
3) Learning Entailment Rules
4) Optimization Techniques
5) Experimental Evaluation

Entailment Rules

Q: “Which country controls Okinawa?”

C: “... Japan annexed Okinawa ...”

→ A: “Japan controls Okinawa”

Entailment Rules

annex(X,Y)

annex(X,Y) → control(X,Y)

∴ control(X,Y)

Entailment Rules

annex(X,Y)

annex(X,Y) → control(X,Y)

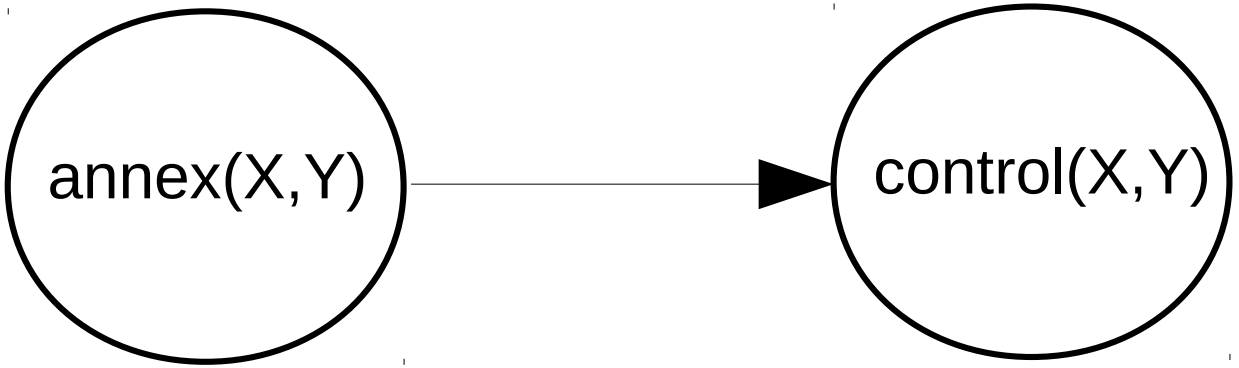
∴ control(X,Y)

Find lots of these!

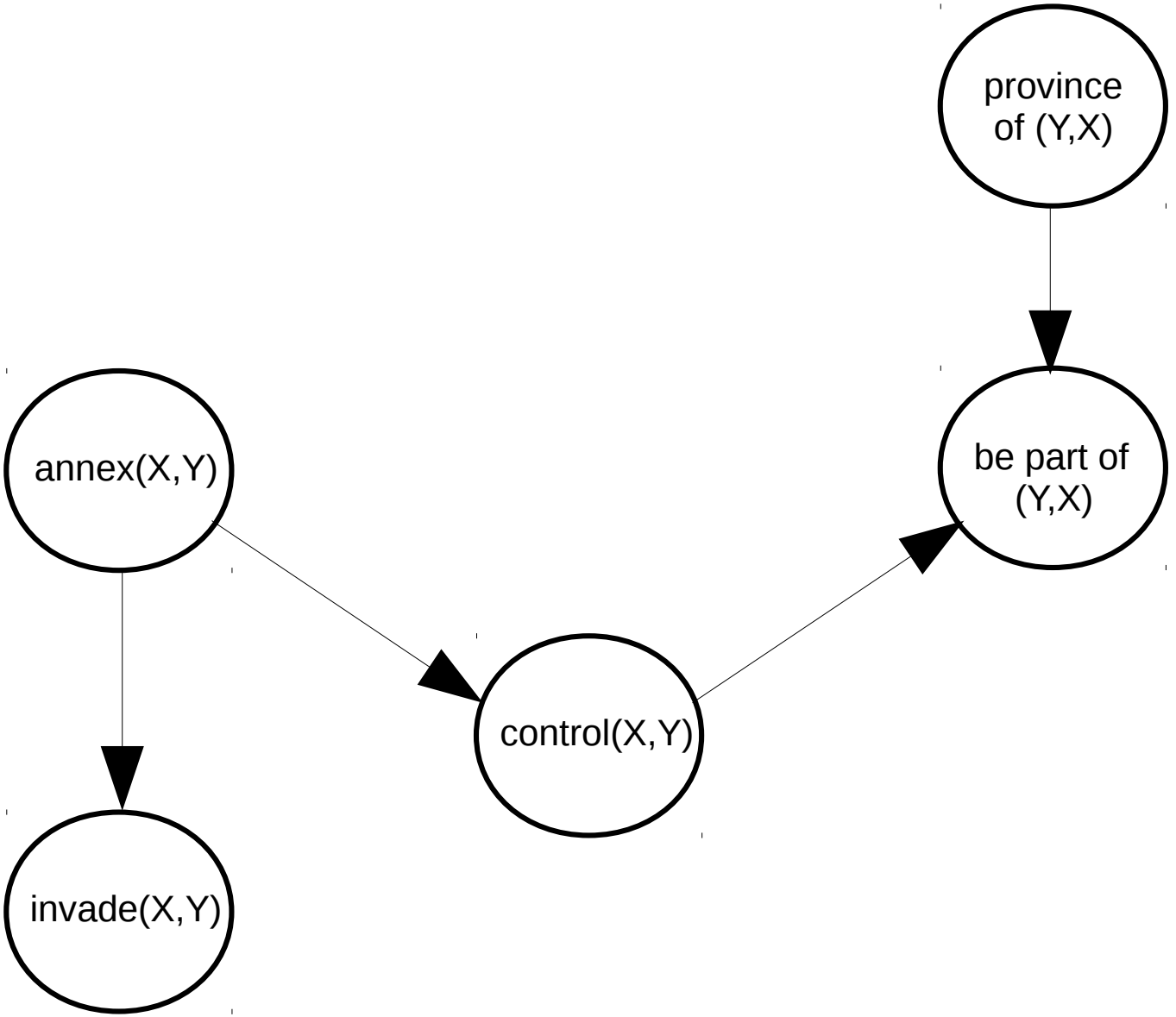
Entailment Rules

$$\text{annex}(X,Y) \rightarrow \text{control}(X,Y)$$

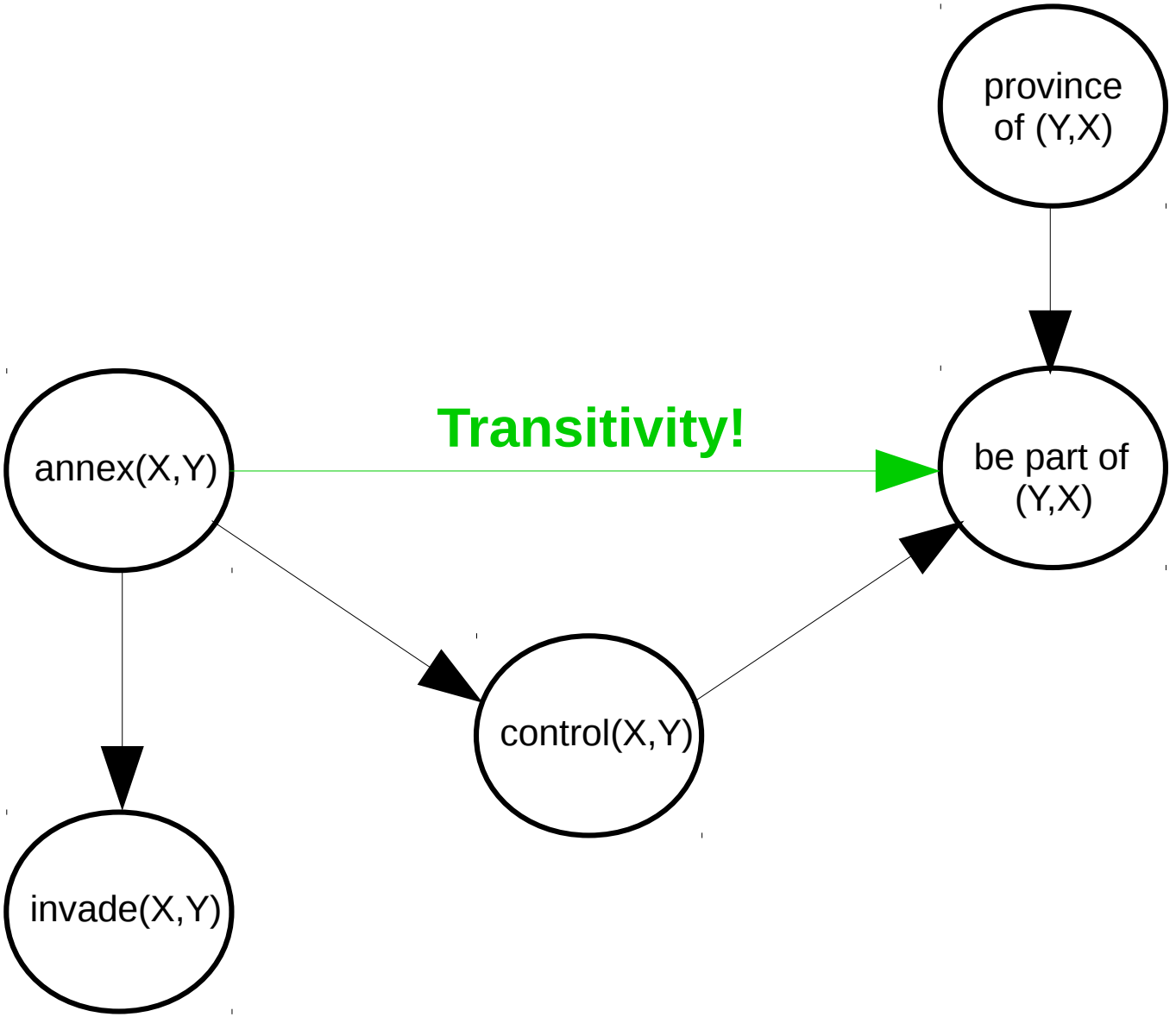
Graph-theoretic Model



Graph-theoretic Model



Graph-theoretic Model



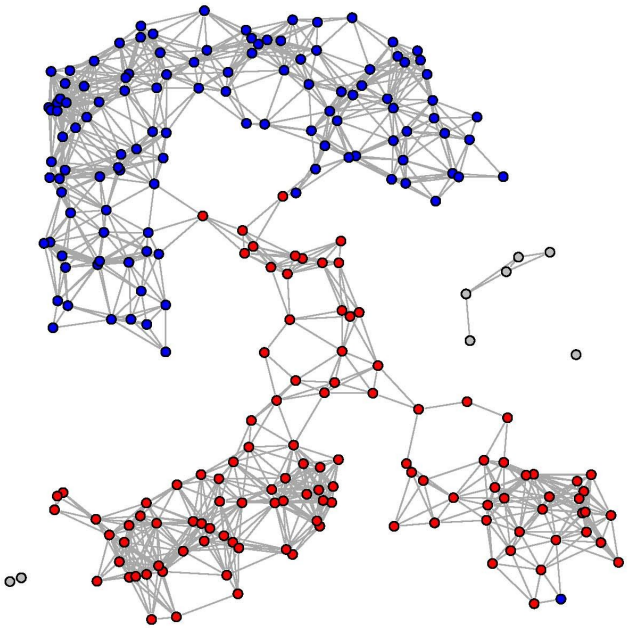
Graph-theoretic Model: Challenges

1) Ambiguity

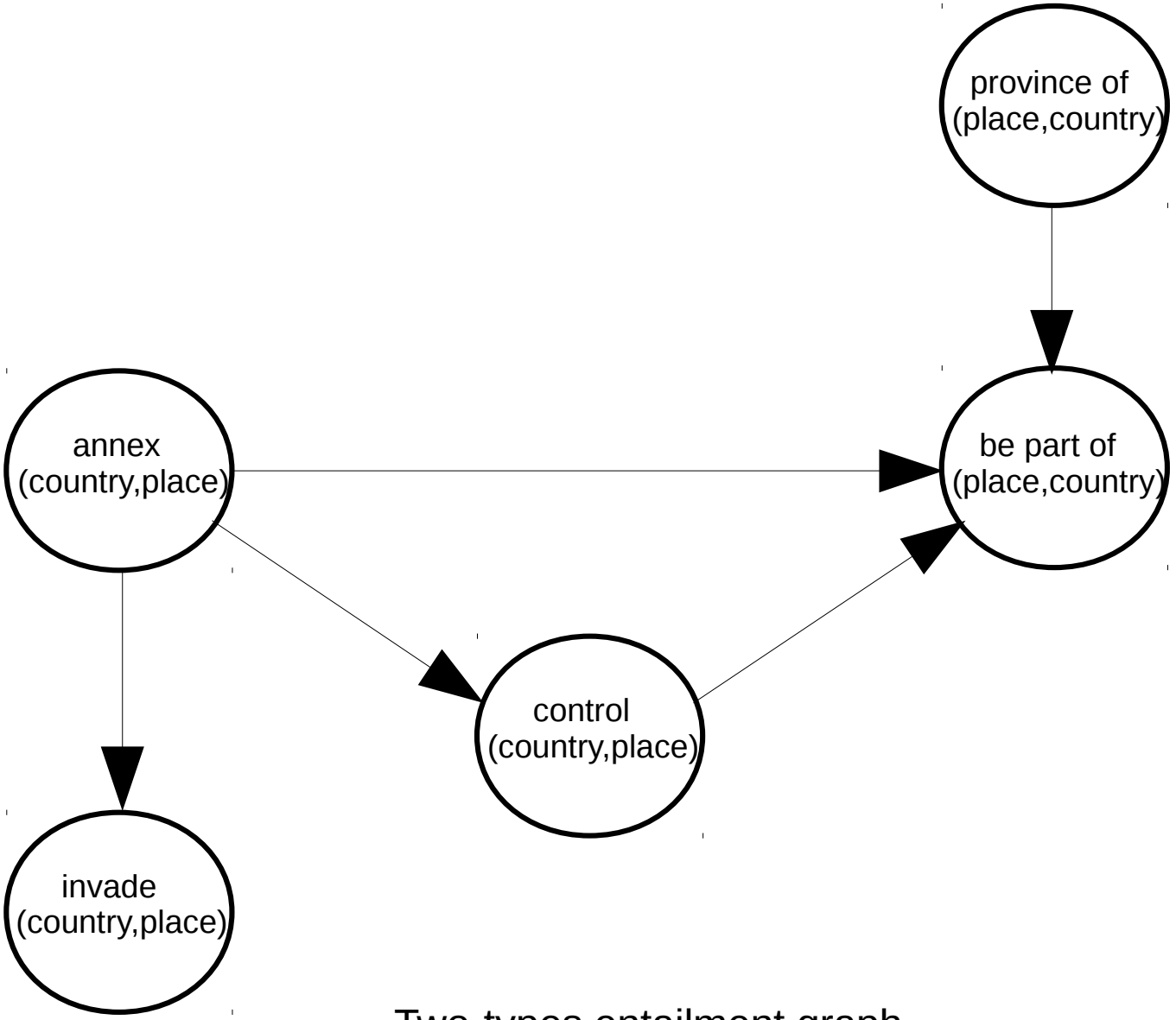
buy(X,Y) → acquire(X,Y)
acquire(X,Y) → learn(X,Y)

buy(X,Y) $\not\rightarrow$ learn(X,Y)

2) Scalability



Graph-theoretic Model: Typed Predicates



Two-types entailment graph

Typed Entailment Graphs: Learning Entailment Rules

Step 1: Get a set of typed predicates

Step 2: Train an entailment classifier

Step 3: Optimize the edges between predicates using the classifier

Typed Entailment Graphs: Learning Entailment Rules

Step 1: Get a set of typed predicates

Step 2: Train an entailment classifier

Step 3: Optimize the edges between predicates using the classifier

→ The edges are the learned entailment rules

Step 1:
Get a set of typed predicates

- Get tuples extracted from text (e.g. TextRunner)
- Map tuples to typed predicates using Hearst patterns (e.g. “X such as Y”)
- Decompose predicates by type

Step 2: Train an entailment classifier

- Generate examples (pairs of predicates; positive and negative) using WordNet
- Represent each example with a feature vector
- Use the data to train a probabilistic classifier in order to compute

$$P_{uv} = P\left(X_{uv} = 1 | F_{uv}\right)$$

where X_{uv} is an indicator of whether an edge exists between predicates u and v, and F_{uv} is the feature vector representing the pair

Step 3:
Optimize the entailment graph edges

- Using the trained classifier, find the optimal graph Ĝ:

$$\hat{G} = arg\,max \sum_{u \neq v} f(u, v) \cdot X_{uv}$$

f: Score function for every pair (u,v)

X_{uv}: Binary value that indicates whether an edge u → v exists

Graph Optimization: Score Function

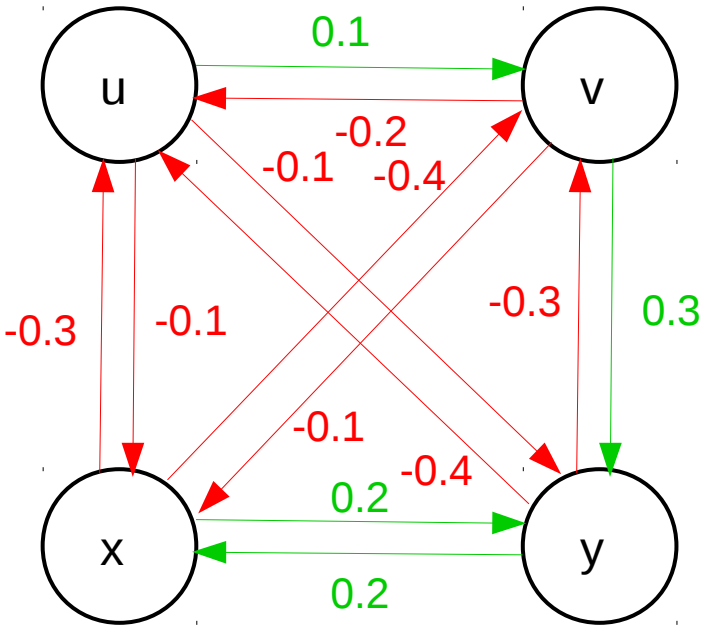
$$f(u,v) = \log\left(\frac{P_{uv}\cdot P(X_{uv}=1)}{(1-P_{uv})\cdot P(X_{uv}=0)}\right)$$

$$= \log\left(\frac{P_{uv}}{1-P_{uv}}\cdot\frac{P(X_{uv}=1)}{P(X_{uv}=0)}\right)$$

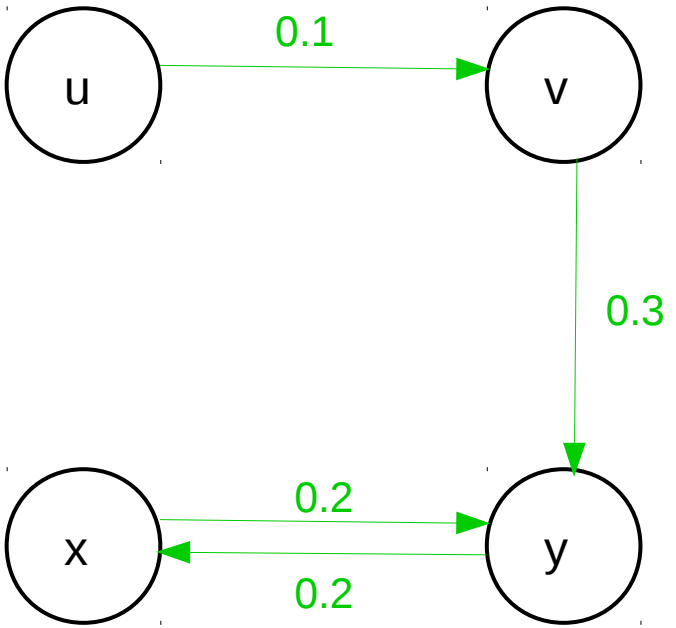
Likelihood component

Edge prior (η)

Graph Optimization: Score only



Graph Optimization: Score only



Graph Optimization: Constraints

$$\hat{G} = arg\,max \sum_{u \neq v} f(u, v) \cdot X_{uv}$$

$$\forall_{u,v,w \in V}\ X_{uv} + X_{vw} - X_{uw} \leq 1 \quad \text{(Transitivity)}$$

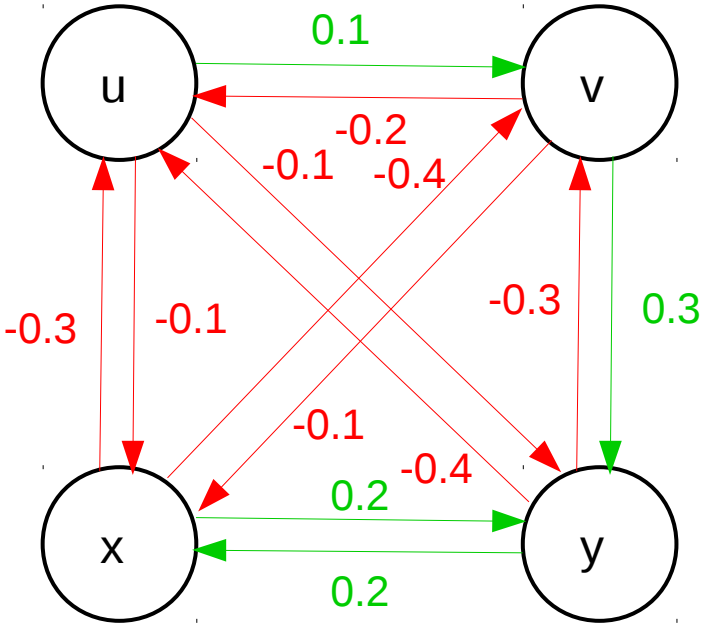
$$\forall_{u,v \in A_{yes}}\ X_{uv} = 1$$

$$\forall_{u,v \in A_{no}}\ X_{uv} = 0$$

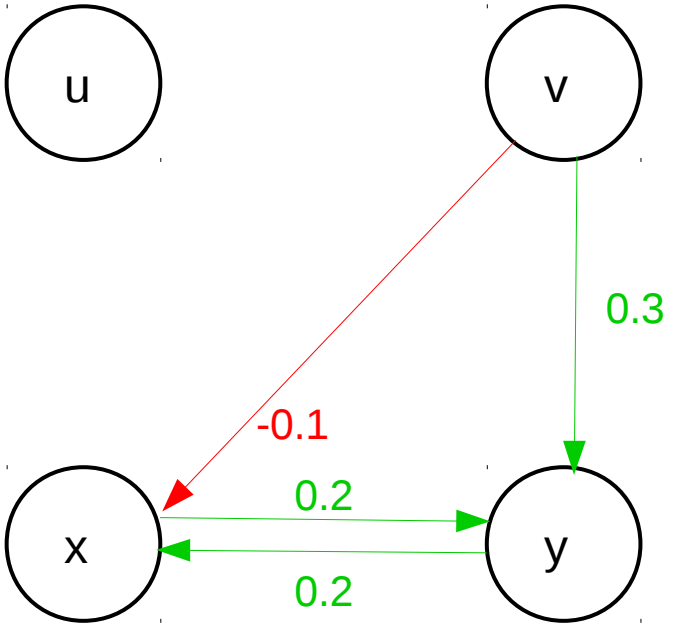
(Background knowledge)

$$\forall_{u \neq v}\ X_{uv} \in \{0,1\}$$

Graph Optimization: Score and Transitivity



Graph Optimization: Score and Transitivity



Optimization Techniques: Graph Decomposition

- Take advantage of graph sparsity

  → most predicates do not entail each other

- Divide the graph into connected components and solve them separately
- This method will still return an optimal solution

Optimization Techniques: Incremental ILP

- Omit transitivity constraints at first

  → given a good classifier, we should expect most constraints to be satisfied in the first place
- Add constraints only when they are violated
- Repeat until no constraints are violated

Experimental Evaluation: Scalability

- Compare the algorithm with and without scaling techniques

  → Which graphs are too big to learn?

- Different η modify graph sparsity

Experimental Evaluation: Scalability

$\log \eta$	# unlearned	# rules	$\triangle$	Red.
-1.75	9/0	6,242 / 7,466	20%	75%
-1	9/1	16,790 / 19,396	16%	29%
-0.6	9/3	26,330 / 29,732	13%	14%

Table 3: Impact of scaling techinques (ILP^{-}/ILP_{scale}).

- Graph decomposition in sparse graphs results in very small subgraphs
- Even for greater η, most large graphs can still be learned
- This results in more overall learned rules

Experimental Evaluation: Rule Learning

- Manually annotate 10 graphs (gold standard)
- Apply the algorithm to the set of predicates
- Compare the results to some baseline algorithms as well as a no-transitivity approach (each with and without background knowledge)

Experimental Evaluation: Rule Learning

- ILP_{scale} performs better than all other algorithms in most regards

	micro-average			
	R (%)	P (%)	F_1 (%)	AUC
ILP_{scale}	**43.4**	42.2	**42.8**	**0.22**
$clsf_k$	30.8	37.5	33.8	0.17
$Sherlock_k$	20.6	**43.3**	27.9	N/A
$BInc_k$	31.8	34.1	32.9	0.17
SR_k	38.4	23.2	28.9	0.14
$DIRT_k$	25.7	31.0	28.1	0.13

Table 2: micro-average F_1 and AUC for the algorithms.

Experimental Evaluation: Rule Learning

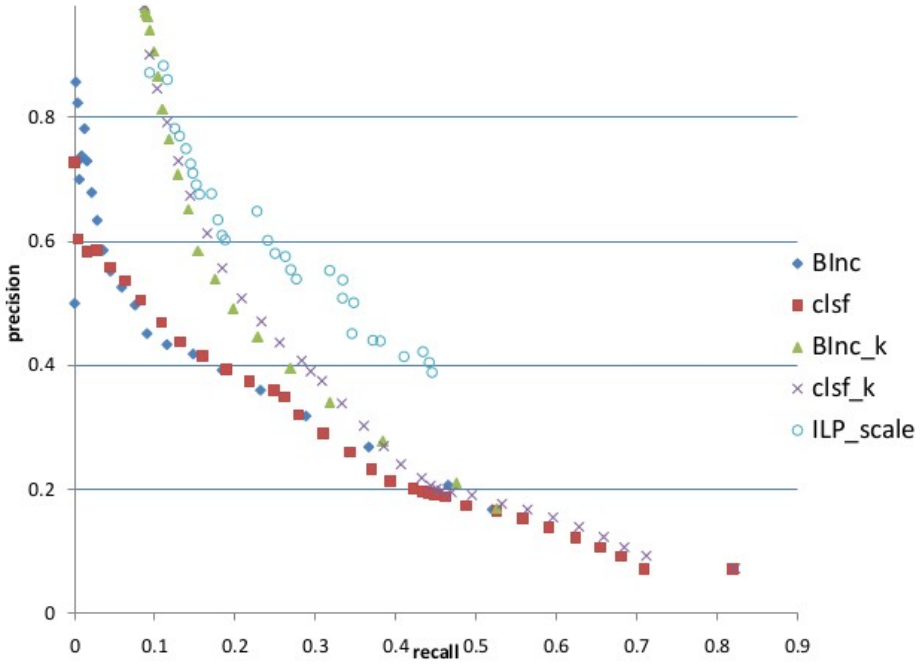


Figure 2: Precision-recall curve for the algorithms.

- Transitivity constraints are useful for eliminating false positives