Rule induction in knowledge graphs using linear programming

Sanjeeb Dash, Joao Goncalves
IBM Research AI

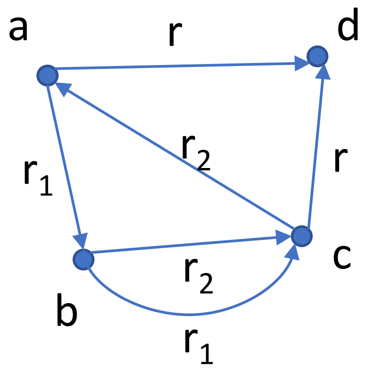
AAAI 2023

Knowledge graph completion

Knowledge Graph (KG): Directed node/edge-labeled multigraph; each edge is a "fact", and edge labels represent binary relations between nodes.

Example: (a, r_1, b) is a fact or $r_1(a, b)$ is true

a, b, c, d could be individuals,
r, r_1, r_2 could be *son_of, brother_of, related_to*



Knowledge graphs often have *missing* (and *incorrect)* facts.

KG completion problem: Find missing facts e.g., $(b, brother\_of, a), (c, brother\_of, a)$

Popular methods: Rule based & Embedding based

Rules

Examples: $(X, \text{son\_of}, Y) \wedge (Y, \text{son\_of}, Z) \rightarrow (X, \text{grandson\_of}, Z)$

KG Completion Problem: Answer query $(a, r, ?)$

Standard Approach:

1) Learn rule-based function $f_r\ (X, Y)$ which gives high scores to likely facts (X, r, Y) where X, Y are nodes in the graph, and r is an edge-label/relation

2) Answer query $(a, r, ?)$ by finding x such that $f_r\ (a, x)$ has highest score.

3) If the correct answer is b, measure accuracy by average rank/reciprocal rank of b (MR/MRR)

Prior work

Kok, Domingos '05, Richardson, Domingos '06 – *Markov Logic Networks*
Yang, Yang, Cohen '17 (NeuralLP) – *Neuro-symbolic methods*
Rochstätel, Riedel '17 (NTP) – ,,
Sadeghian, Armandpour, Ding, Wang '19 (DRUM) – ,,
Evans, Grefenstette '18 – *Differential ILP*
Das et al. '18 (Minerva) – *Reinforcement Learning*
Qu et. al. '21 (RNNLogic) – RNN + Probabilistic methods
Meilicke et. al. '19 (AnyBURL) – *Data mining*
Teru, Denis, Hamilton '20 (GraIL) – *Subgraph reasoning*

Advantages: (1) Inductive reasoning is possible.
(2) *Interpretable models when few rules are generated.*

Drawbacks: (1) Lower levels of accuracy compared to embedding methods
(2) *Current methods do not scale*

Embedding based methods

Approach: Find $v_a \in \mathbb{R}^k$ for each node a and a mapping $T_r: \mathbb{R}^k \to \mathbb{R}^k$ for each relation r such that the *score* $\|T_r(v_a) - v_b\|$ is small for each fact (a, r, b).

Bordes, Usunier, Garcia-Duran, Weston, Yakhnenko '13 (TransE)
Yang, Yih, He, Gao, Deng '15 (DistMult)
Trouillon, Welbl, Riedel, Gaussier, Bouchard '16 (ComplEx)
Dettmers, Pasquale, Pontus, Riedel '18 (ConvE)
Lacroix, Usunier, Obozinski '18 (ComplEx-N3)
Sun, Deng, Nie, Tang '19 (RotatE)

Advantages: (1) Reasonable accuracy
(2) Scalable

Drawbacks: (1) Not effective for inductive reasoning (works for transductive reasoning)
(2) Model is not interpretable.

Our work

Goals: Develop a scalable, rule-learner that returns compact sets of rules

- Interpretability is an explicit goal, and we return low-complexity rules
- We trade off complexity versus accuracy
- Scalability is attained by solving linear programming models instead of non-convex models

Our work

Approach: Learn few *(FOL)* rules $R_1, \ldots, R_p$ and positive weights $w_1, \ldots, w_p$ where each R_i has the form

$$r_1(X, X_1) \land r_2\,(X_1, X_2) \land \cdots \land r_l\,(X_{l-1}, Y) \rightarrow r(X, Y)$$

where $r_1, \ldots, r_l$ are relations in G.

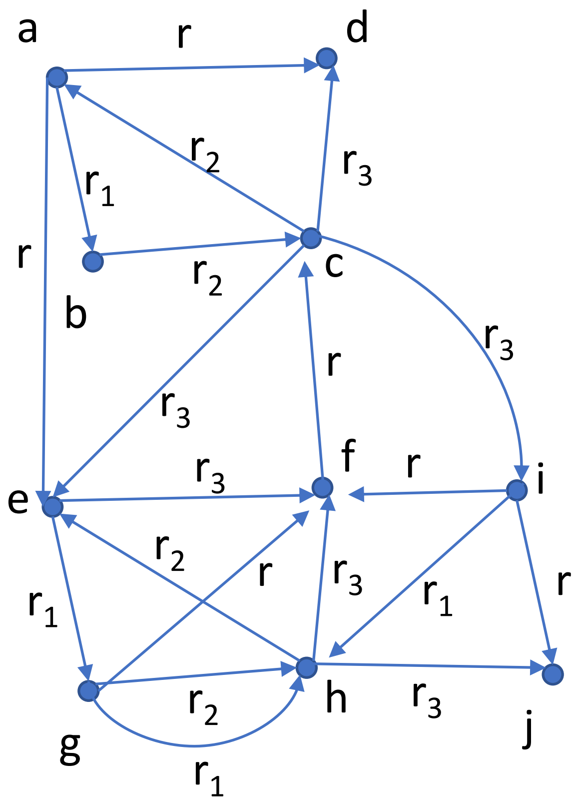
The *length* of this rule is l, and the left-hand-side of the rule is the clause $C_i\colon \mathrm{V} \times \mathrm{V} \rightarrow \{0,1\}$

The learned prediction/scoring function $f_r\colon \mathrm{V} \times \mathrm{V} \rightarrow \mathbb{R}_+$ for r is:

$$f_r\,(X, Y) = \sum_{i=1}^{p} w_i C_i(X, Y) \quad \forall\, X, Y \in V$$

Details

KG:
a-j are entities
r, r_1, r_2, r_3 are relations



$C_1(X,Y)$

Rule $r_1(X,X_1) \wedge r_2\,(X_1,X_2) \wedge r_3\,(X_2,Y) \rightarrow r(X,Y)$ and associated clause-edge vector

edge	$r_1 \wedge r_2 \wedge r_3$	r
(a,d)	1	**1**
(a,e)	1	**1**
(f,c)	0	**1**
(g,f)	1	**1**
(i,f)	1	**1**
(i,j)	0	**1**
(e,f)	**1**	0
(a,i)	**1**	0
(e,j)	**1**	0

a_{i1}

$r_1(a,b) \wedge r_2\,(b,c\,) \wedge r_3\,(c,d)$ is true and $r(a,d)$ is true

positive instances:
edges in KG = E_r

negative instances:
non-edges (sample)

LP Model

Minimize error for weighted collection of rules:

loss on positive instances — loss on negative instances

$$\min_{w,\xi} \sum_{i:y_i=1} \xi_i + \tau \sum_{k \in K} \text{neg}_k w_k$$

cover positives → $\xi_i + \underbrace{\sum_{k \in K} a_{ik} w_k}_{\text{value of scoring fn.}} \geq 1, \quad \xi_i \geq 0, \; (t_i, h_i) \in E_r$

complexity bound → $\sum_{k \in K} c_k w_k \leq C$

select clause *k* or not → $w_k \in [0,1], \qquad k \in K$

value of scoring fn.

Model details

- E_r = set edges labeled by r, and (t_i, h_i) = ith edge in E_r
- w_k variable gives weight for rule k; $w_k > 0$ implies rule k is chosen
- a_{ik} is a constant = $C_k(t_i, h_i)$
- c_k is a constant = 1+ rule length
- C is a parameter bounding weighted complexity of chosen rules
- τ is a parameter, neg_k is a constant

Modeling
– Use all positive facts for a relation + sample some negative facts for the LP model

Algorithmic issues
– Use simple shortest path heuristics to find relational paths, and associated rules
– Iterate over different values of tau and complexity

Code available at: https://github.com/IBM/LPRules

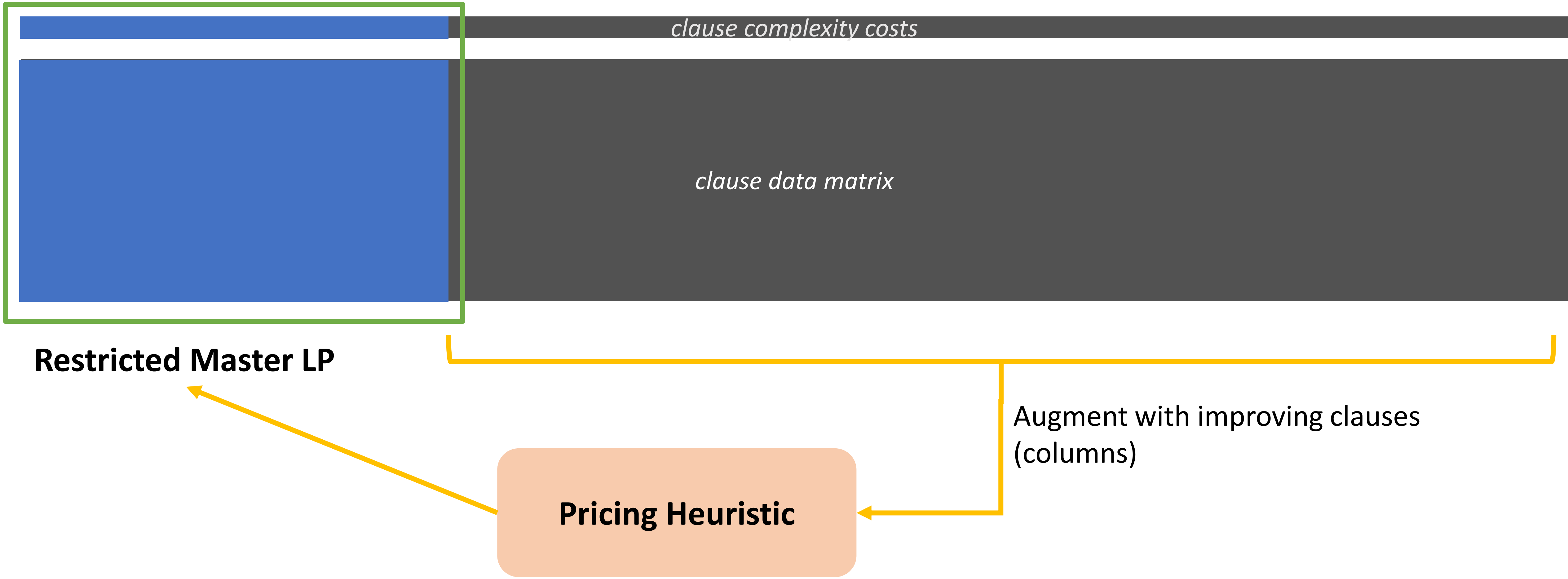
Related work

Linear Programming based boosting methods for classification that use column generation

Demirez, Bennett, Shawe-Taylor '02
Eckstein, Goldberg '12
Eckstein, Kagawa, Goldberg '19
Dash, Gunluk, Wei '18

Column Generation

Solve LP over small subsets of rules



Column Generation

Step 0 – Fix an initial complexity and tau value

Step 1 – Use simple heuristics to create an initial collection of rules

Step 2 – Set up LP model and solve it

Step 3 – Obtain dual values of LP model

Step 4 – Dual values indicate which facts are "well-covered" and which are not. Heuristically generate new rules that "cover" facts that are not well-covered.

Step 5 – Repeat Steps 2 – 4 till termination criterion

Sizes of datasets

Datasets	# Relations	# Entities	# Train	# Test	# Valid
Kinship	25	104	8544	1074	1068
UMLS	46	135	5216	661	652
FB15k-237	237	14541	272115	20466	17535
WN18RR	11	40943	86835	3134	3034
YAGO3-10	37	123182	1079040	5000	5000

Neuro-symbolic methods take a long time on FB15k-237 and cannot handle YAGO3-10

Experiments (accuracy)

Datasets	ComplEx-N3	AnyBURL	NeuralLP	DRUM	RNNLogic	LPRules
Kinship	0.889	0.626	0.652	0.566	*0.687*	**0.746**
UMLS	0.962	**0.940**	0.750	0.845	0.748	*0.869*
FB15k-237	0.362	0.226	0.222	0.225	†**0.288**	*0.255*
WN18RR	0.469	*0.454*	0.381	0.381	0.451	**0.459**
YAGO3-10	0.574	**0.449**				**0.449**

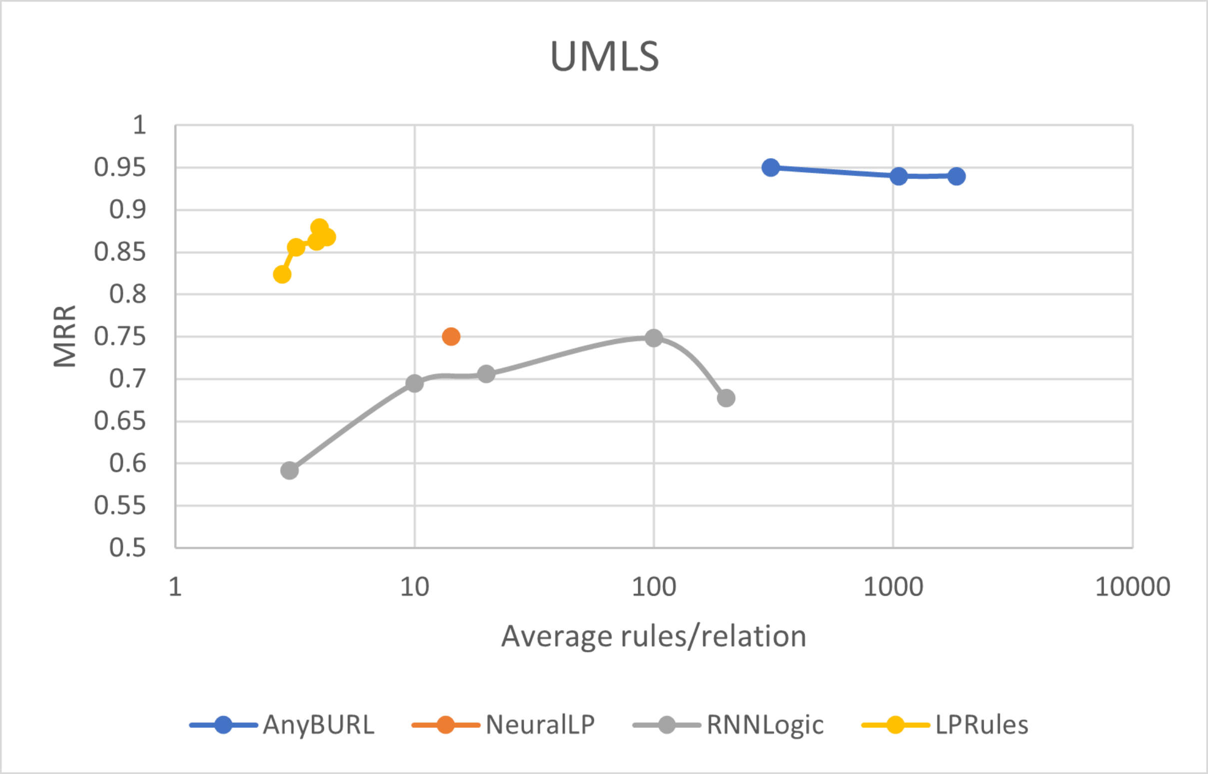
† We could not run RNNLogic on FB15k-237 and report numbers taken from Qu et al. (2021)

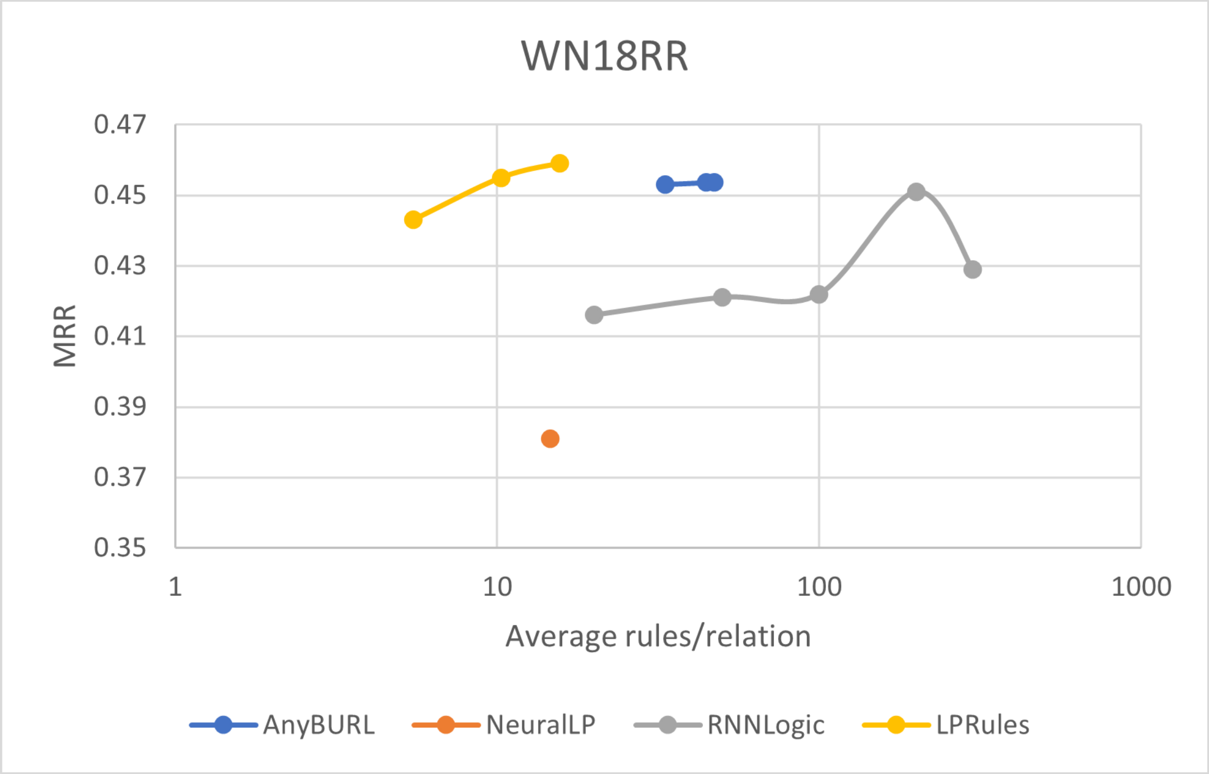
Running time + number of rules

Metric	Datasets	AnyBURL	NeuralLP	RNNLogic	LPRules
Average # rules per relation	Kinship	6653.1	10.4	200.0	21.0
	UMLS	1837.6	15.1	100.0	4.2
	FB15k-237	79.9	8.1		14.2
	WN18RR	47.3	14.3	200.0	15.6
	YAGO3-10	63.0			7.8
Running time	Kinship	1.7	1.6	108.8	0.5
	UMLS	1.9	1.1	133.4	0.2
	FB15k-237	3.9	14565.9		234.5
	WN18RR	1.8	399.9	104.0	11.0
	YAGO3-10	34.3			1648.4

Avg number of rules per relation and wall clock running time on a 60 core machine

Accuracy versus Complexity tradeoff

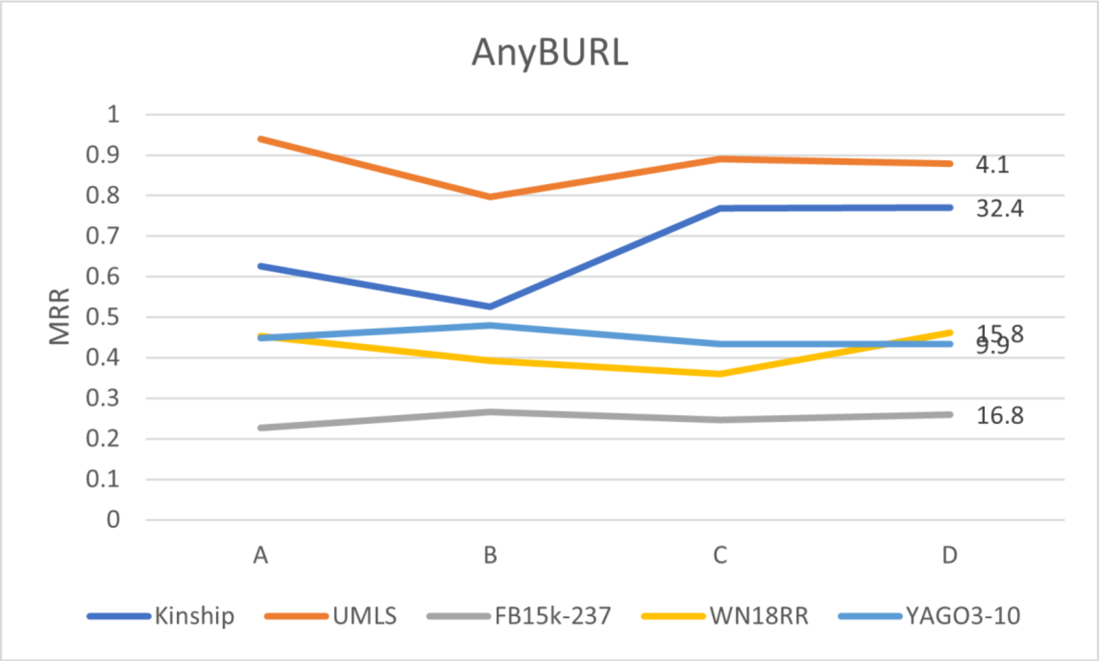


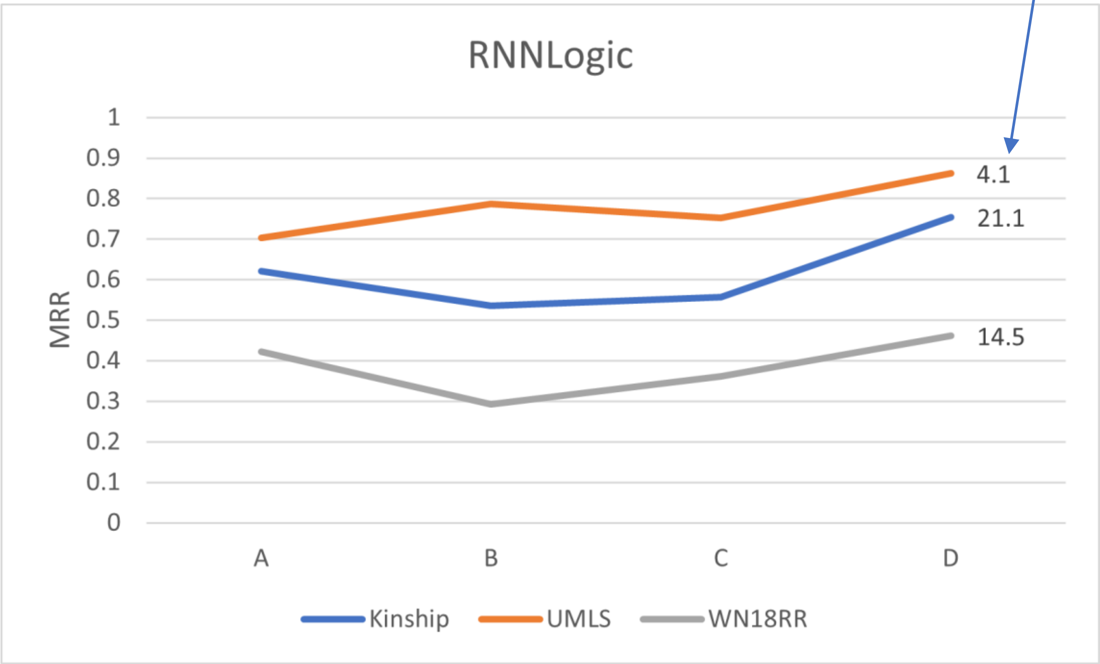


Change in MRR with change in average rules per relation

LPRules + rules from other code

Avg # rules/relation





MRR values using rules generated by AnyBURL and RNNLogic (in experiments A-D)

A – Use other rule-based code
B – Take rules and weights and use in our prediction function
C – Recalculate weights using complexity bound
D – Add our rules and recalculate weights

Concluding remarks

Features

– Our LP model performs well: it chooses a small set of rules that yield high accuracy

– Our simple rule generation heuristics suffice for small datasets

– Column generation is essential for large datasets such as YAGO3-10

Directions for improvement

– More general rules

– Better sampling (for better scaling & accuracy)