KGTuner: Efficient Hyper-parameter Search for Knowledge Graph Learning

Yongqi Zhang, zhangyongqi@4paradigm.com
Zhanke Zhou, cszkzhou@comp.hkbu.edu.hk
Quanming Yao, qyaoaa@tsinghua.edu.cn
Yong Li, liyong07@tsinghua.edu.cn

ACL 2022
22ND – 27TH MAY | 60TH MEETING | DUBLIN

https://github.com/AutoML-Research/KGTuner

4Paradigm 第四范式
香港浸會大學 HONG KONG BAPTIST UNIVERSITY
清華大学 Tsinghua University

Knowledge Graph (KG) Learning

Overall framework:

$$\min_{\mathbf{w}} \mathcal{L}(F(\cdot,\mathbf{w}), \mathcal{S}^{+}, \mathcal{S}^{-}) + r(\mathbf{w})$$

optimization · loss function · scoring function · negative sampling · regularization

Given a scoring function, the model is trained under configuration of negative sampling, loss function, regularization and optimization.

component	name	type	range
negative sampling	# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	loss function	cat	{MR, BCE_(mean, sum, adv), CE}
	gamma (MR)	float	[1, 24]
	adv. weight (BCE_adv)	float	[0.5, 2.0]
regularization	regularizer	cat	{FRO, NUC, DURA, None}
	reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
	dropout rate	float	[0, 0.5]
optimization	optimizer	cat	{Adam, Adagrad, SGD}
	learning rate	float	$[10^{-5}, 10^{0}]$
	initializer	cat	{uniform, normal, xavier_uniform, xavier_norm}
	batch size	int	{128, 256, 512, 1024}
	dimension size	int	{100, 200, 500, 1000, 2000}
	inverse relation	bool	{True, False}

Hyper-parameter (HP) Optimization for KG Learning

Denote $\mathbf{x}$ as an configuration of HPs.

Definition 1 (Hyper-parameter search for KG embedding). *The problem of HP search for KG embedding model is formulated as*

$$\mathbf{x}^{*} = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathcal{M}(F(\boldsymbol{P}^{*},\mathbf{x}), D_{val}), \quad (2)$$

$$\boldsymbol{P}^{*} = \arg\min_{\boldsymbol{P}} \mathcal{L}(F(\boldsymbol{P},\mathbf{x}), D_{tra}). \quad (3)$$

Three major aspects for efficiency in Def. 1
1. the **size** of search space $\mathcal{X}$
2. the validation **curvature** of $\mathcal{M}$
3. the evaluation **cost** in solving $\arg\min_{P} \mathcal{L}$

The conventional HPO methods do not seriously consider the three aspects.

Understanding the HP Search Problem

Search space $\mathcal{X}$

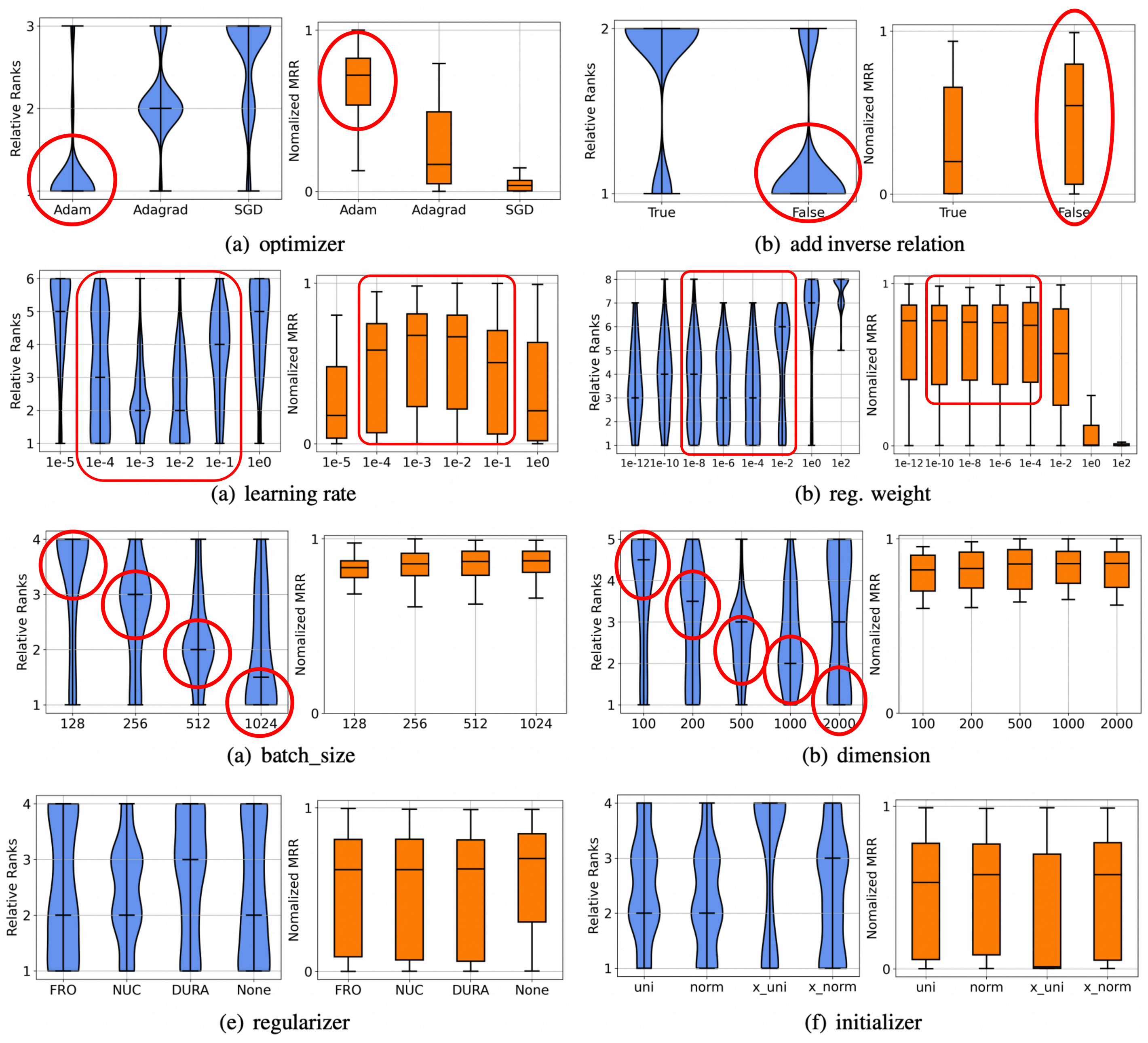
We classify the HPs into four categories according to **distributions of performance**.

Reduced options

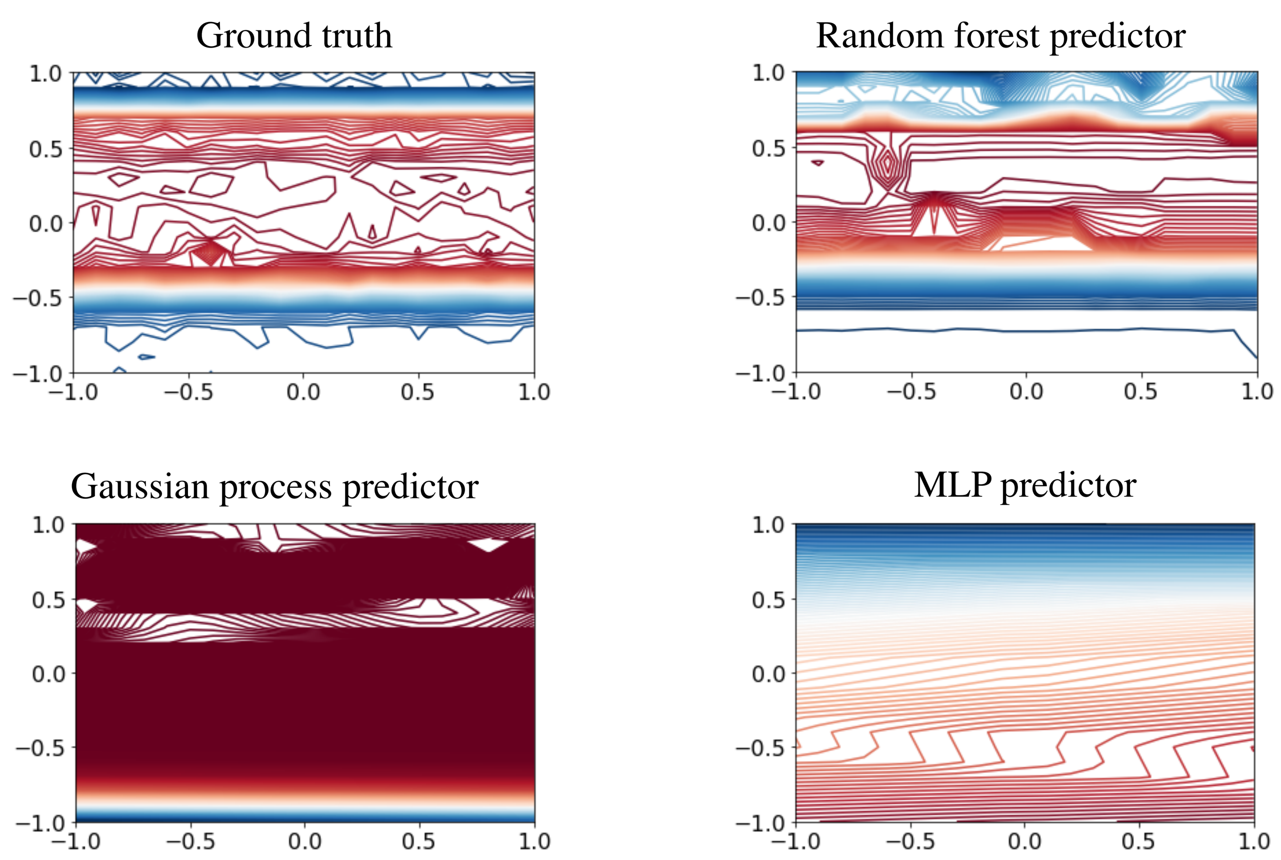
Shrunken range

Monotonously related

No obvious patterns



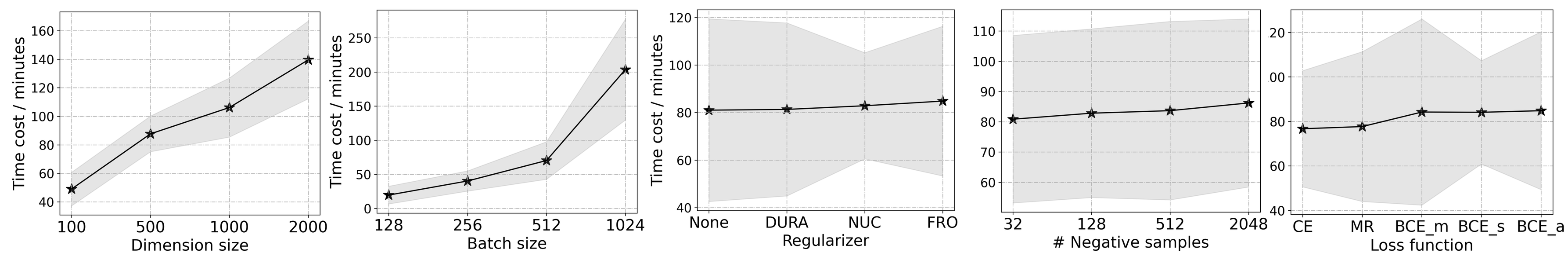
Validation curvature $\mathcal{M}$



Random forest can approximate the validation curvature better.

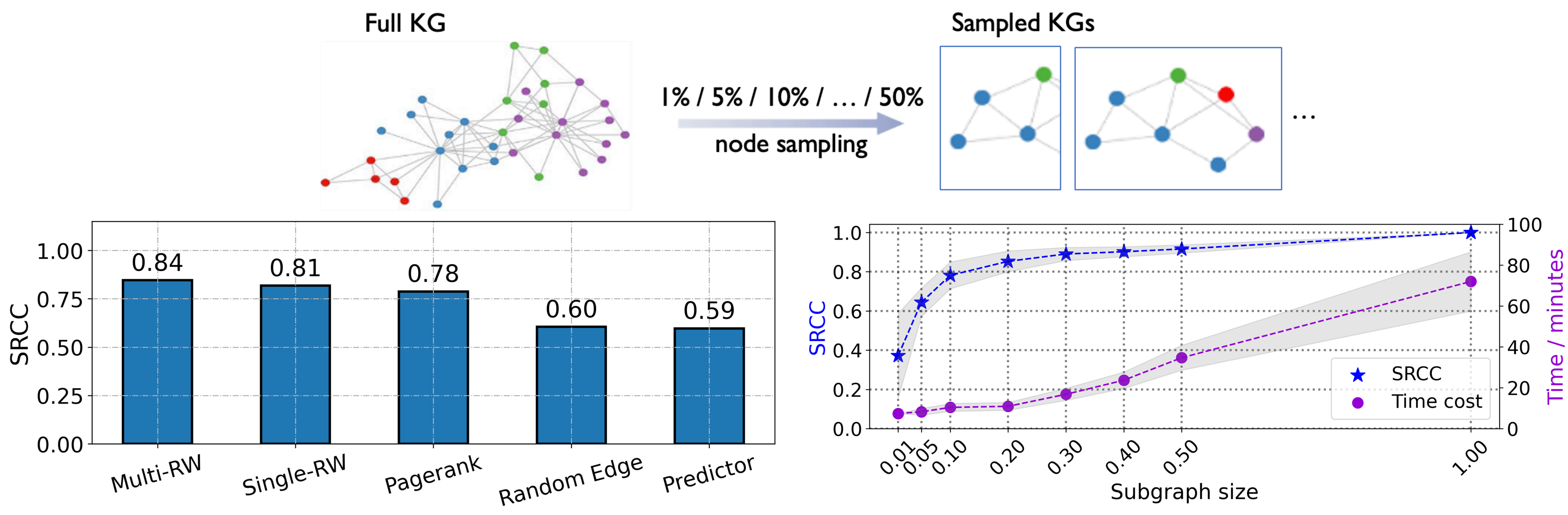
Understanding the HP Search Problem (Continued)

Evaluation cost $\arg\min_{P} \mathcal{L}$



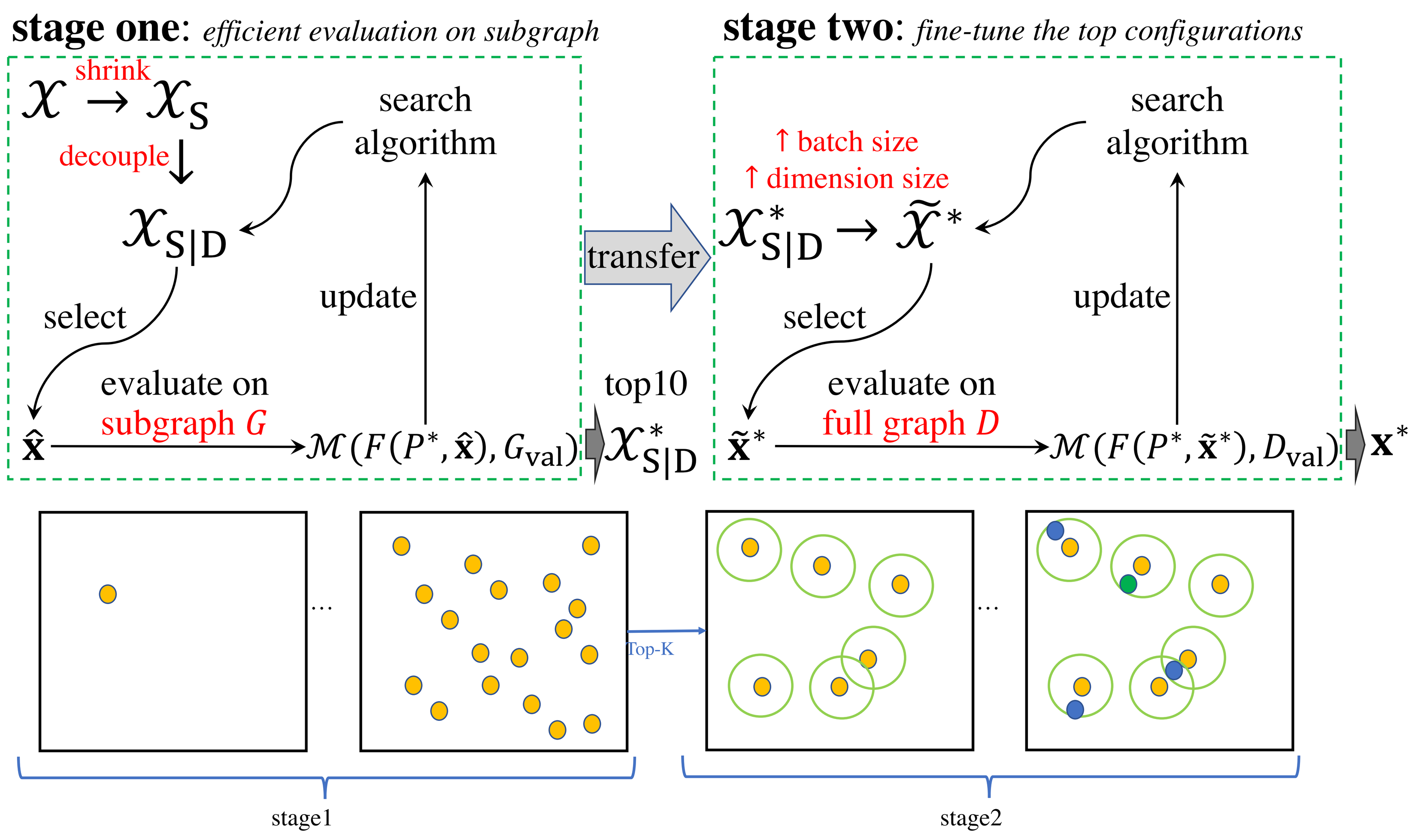
Observation: batch size↑ or dimension↑ ⇒ time cost↑

Using subgraph evaluation to approximate full graph evaluation:



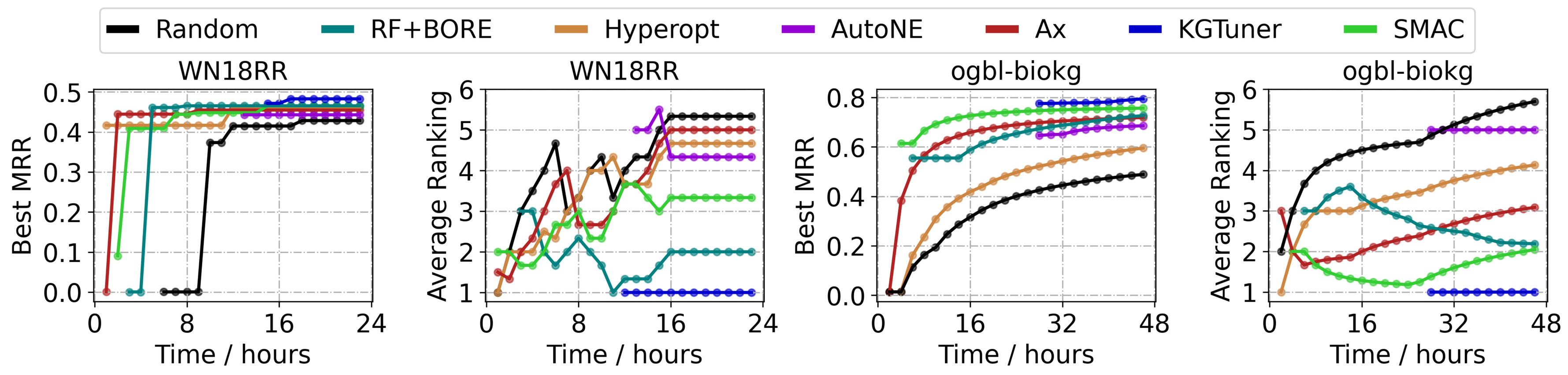
Observations:
- Multi-start random walk is a better strategy to sample subgraphs.
- To balance the consistency and cost, the subgraph with 20% nodes is the better choice

KGTuner: Two-stage Algorithm



Experiments

Search algorithm comparison



models		ogbl-biokg	ogbl-wikikg2
original	TransE	0.7452	0.4256
	RotatE	0.7989	0.2530
	DistMult	0.8043	0.3729
	ComplEx	0.8095	0.4027
	AutoSF	0.8320	0.5186
KGTuner	TransE	0.7781 (4.41%↑)	0.4739 (11.34%↑)
	RotatE	0.8013 (0.30%↑)	0.2944 (16.36%↑)
	DistMult	0.8241 (2.46%↑)	0.4837 (29.71%↑)
	ComplEx	0.8385 (3.58%↑)	0.4942 (22.72%↑)
	AutoSF	0.8354 (0.41%↑)	0.5222 (0.69%↑)
average improvement		2.23%	16.16%

Performance comparison

		WN18RR				FB15k-237			
		MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10
Original	ComplEx	0.440	0.410	0.460	0.510	0.247	0.158	0.275	0.428
	DistMult	0.430	0.390	0.440	0.490	0.241	0.155	0.263	0.419
	RESCAL	0.420	-	-	0.447	0.270	-	-	0.427
	ConvE	0.430	0.400	0.440	**0.520**	0.325	<u>0.237</u>	0.356	0.501
	TransE	0.226	-	-	0.501	0.294	-	-	0.465
	RotatE	<u>0.476</u>	**0.428**	<u>0.492</u>	<u>0.571</u>	**0.338**	<u>0.241</u>	**0.375**	**0.533**
	TuckER	<u>0.470</u>	**0.443**	<u>0.482</u>	<u>0.526</u>	**0.358**	**0.266**	**0.394**	**0.544**
LibKGE (Ruffinelli et al., 2019)	ComplEx	<u>0.475</u>	<u>0.438</u>	<u>0.490</u>	<u>0.547</u>	<u>0.348</u>	<u>0.253</u>	<u>0.384</u>	**0.536**
	DistMult	<u>0.452</u>	**0.413**	<u>0.466</u>	<u>0.530</u>	<u>0.343</u>	<u>0.250</u>	**0.378**	**0.531**
	RESCAL	<u>0.467</u>	**0.439**	<u>0.480</u>	<u>0.517</u>	<u>0.356</u>	<u>0.263</u>	**0.393**	**0.541**
	ConvE	**0.442**	**0.411**	**0.451**	<u>0.504</u>	**0.339**	**0.248**	**0.369**	<u>0.521</u>
	TransE	<u>0.228</u>	**0.053**	<u>0.368</u>	<u>0.520</u>	<u>0.313</u>	<u>0.221</u>	<u>0.347</u>	<u>0.497</u>
KGTuner (ours)	ComplEx	**0.484**	**0.440**	**0.506**	**0.562**	**0.352**	**0.263**	**0.387**	<u>0.530</u>
	DistMult	**0.453**	<u>0.407</u>	**0.468**	**0.548**	**0.345**	**0.254**	<u>0.377</u>	<u>0.527</u>
	RESCAL	**0.479**	<u>0.436</u>	**0.496**	**0.557**	**0.357**	**0.268**	<u>0.390</u>	<u>0.535</u>
	ConvE	<u>0.437</u>	<u>0.399</u>	<u>0.449</u>	0.515	<u>0.335</u>	<u>0.242</u>	<u>0.368</u>	**0.523**
	TransE	**0.233**	<u>0.032</u>	**0.399**	**0.542**	**0.327**	**0.228**	**0.369**	**0.522**
	RotatE	**0.480**	<u>0.427</u>	**0.501**	**0.582**	**0.338**	**0.243**	<u>0.373</u>	<u>0.527</u>
	TuckER	**0.480**	<u>0.437</u>	**0.500**	**0.557**	<u>0.347</u>	<u>0.255</u>	<u>0.382</u>	<u>0.534</u>