Hyper-parameter Optimization

- An example of knowledge graph learning

Dr. Quanming Yao

Assistant professor, EE Tsinghua

qyaoaa@tsinghua.edu.cn

https://lars-group.github.io/index.html

How to use AutoML

1. Define an AutoML problem

- Derive a search space from insights in specific domains
- Search objective is usually validation performance
- Search constraint is usually resource budgets
- Training objective usually comes from classical learning models

$$\min_{\lambda \in \mathcal{S}} M(F(w^*; \lambda), D_{\text{val}})$$

Search Space → $\lambda \in \mathcal{S}$; $M(F(w^*; \lambda), D_{\text{val}})$ ← Search Objective

$$\text{s.t.} \quad \min_{w} L(F(w; \lambda), D_{\text{tra}})$$

$L(F(w; \lambda), D_{\text{tra}})$ ← Training Objective

$$G(\lambda) \leq C$$

$G(\lambda) \leq C$ ← Search Constraints

2. Design or select proper search algorithm

- Reduce model training cost (time to get w^*)

Outline

- What's Hyper-parameter (HP) tuning
- What's Knowledge Graph (KG) learning
- Understand HP in KG learning
- An efficient two-stage HP search algorithm
- Experiments
- Key takeaway and future directions

What are Hyper-parameters?

Search λ Hyper-parameter

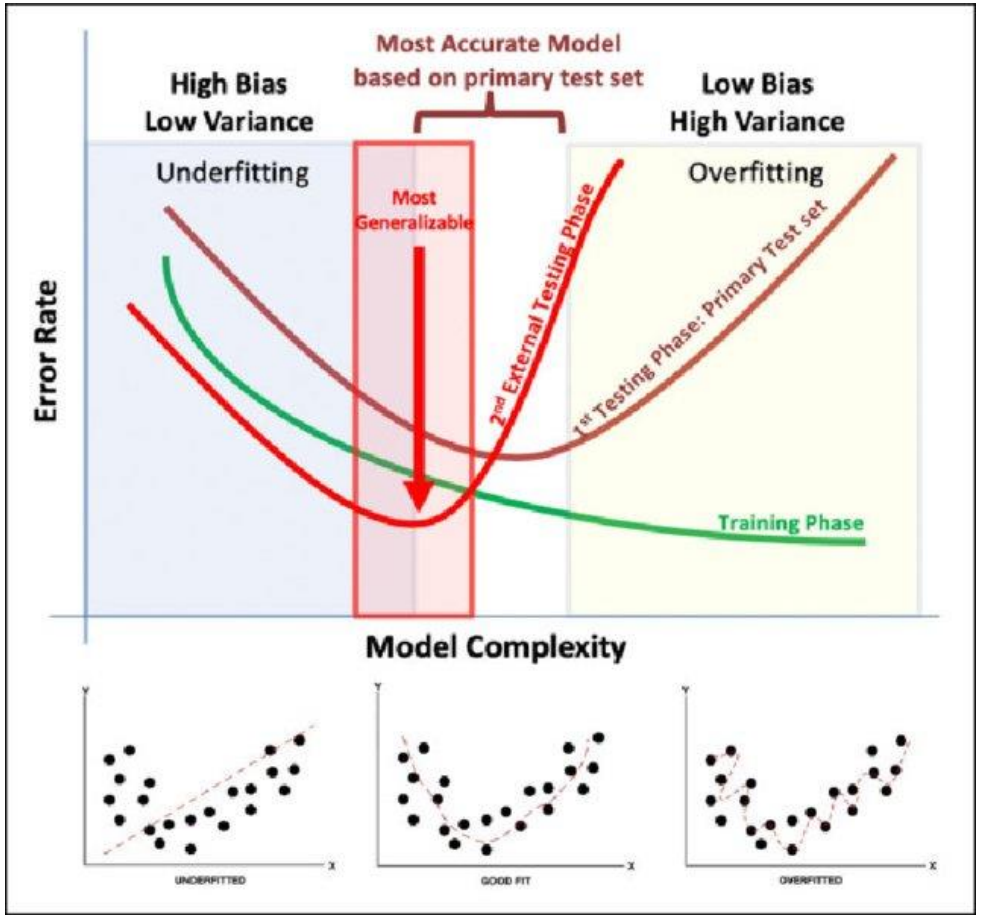
$$\max_{\lambda} \sum_{j} h(x_j; w^*) \quad \text{s.t.} \quad w^* = \min_{w} \sum_{i} f(x_i; w) + \lambda \|w\|_1$$

Validation Performance

Training objective

- Seek proper λ to maximize performance
- Grid search: enumerating $\lambda \in \{1,2,4,8, \dots\}$

Question: can we take some w as hyper-parameters?



[1]. Image source: Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods.

General form – Bi-level optimization

$$\min_{\lambda \in \mathcal{S}} M(F(w^*; \lambda), D_{\text{val}})$$

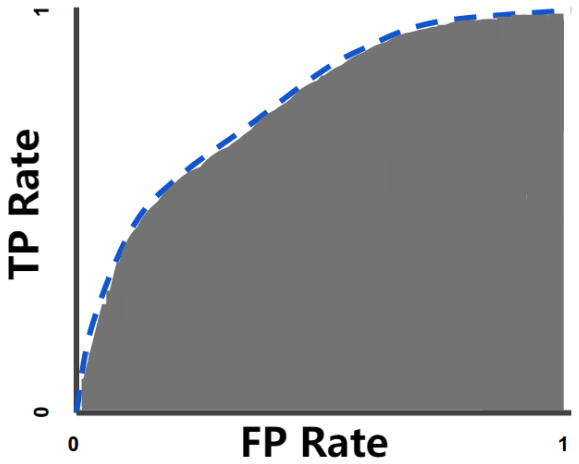
Upper level: minimize validation error

s. t.
$$w^* = \min_{w} L(F(w; \lambda), D_{\text{tra}})$$
$$G(\lambda) \leq C$$

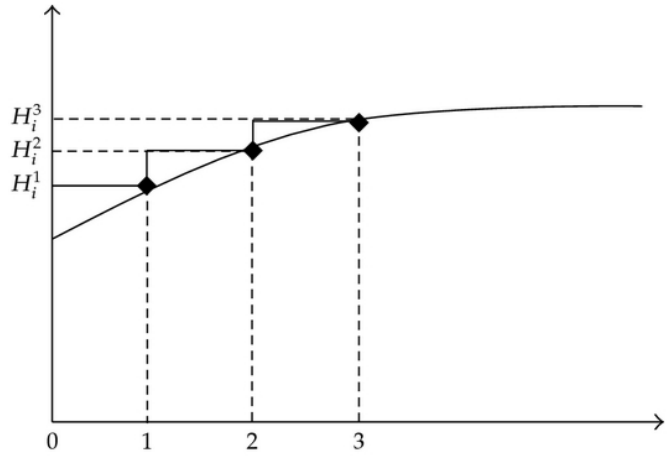
Lower level: obtain parameter by model training

Difficulties, e.g.,

- M can be discrete or undecomposable
- Each iteration to update λ is expensive

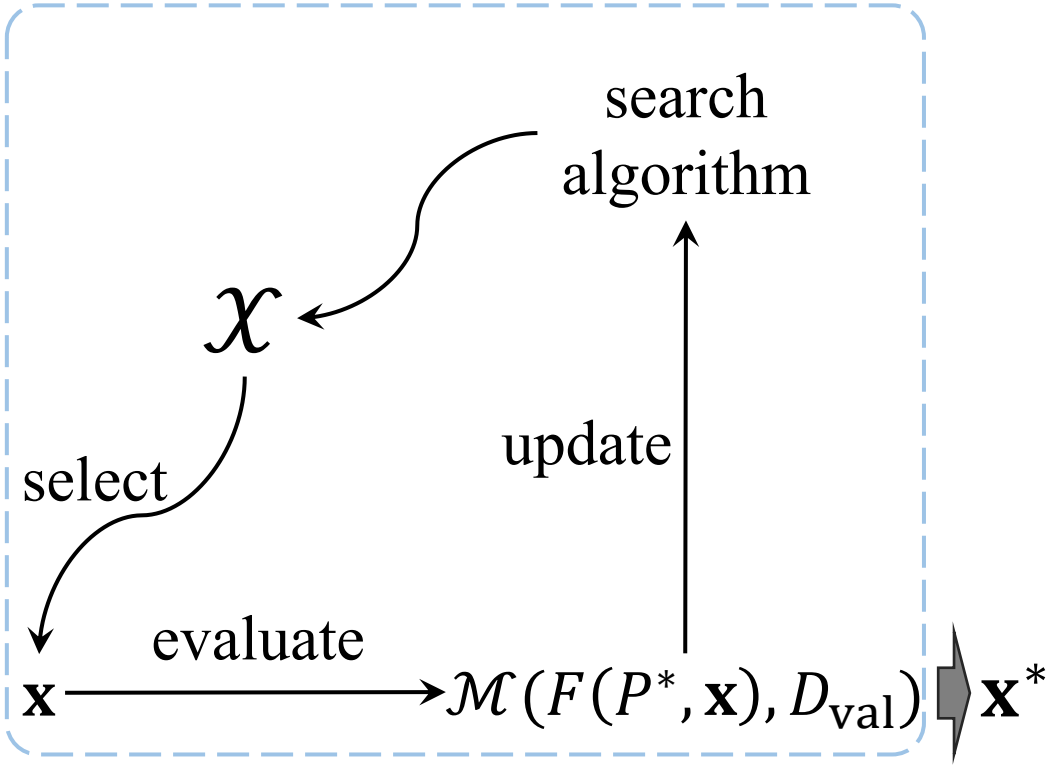


Area Under the ROC Curve
(undecomposable)



Accuracy
(discrete)

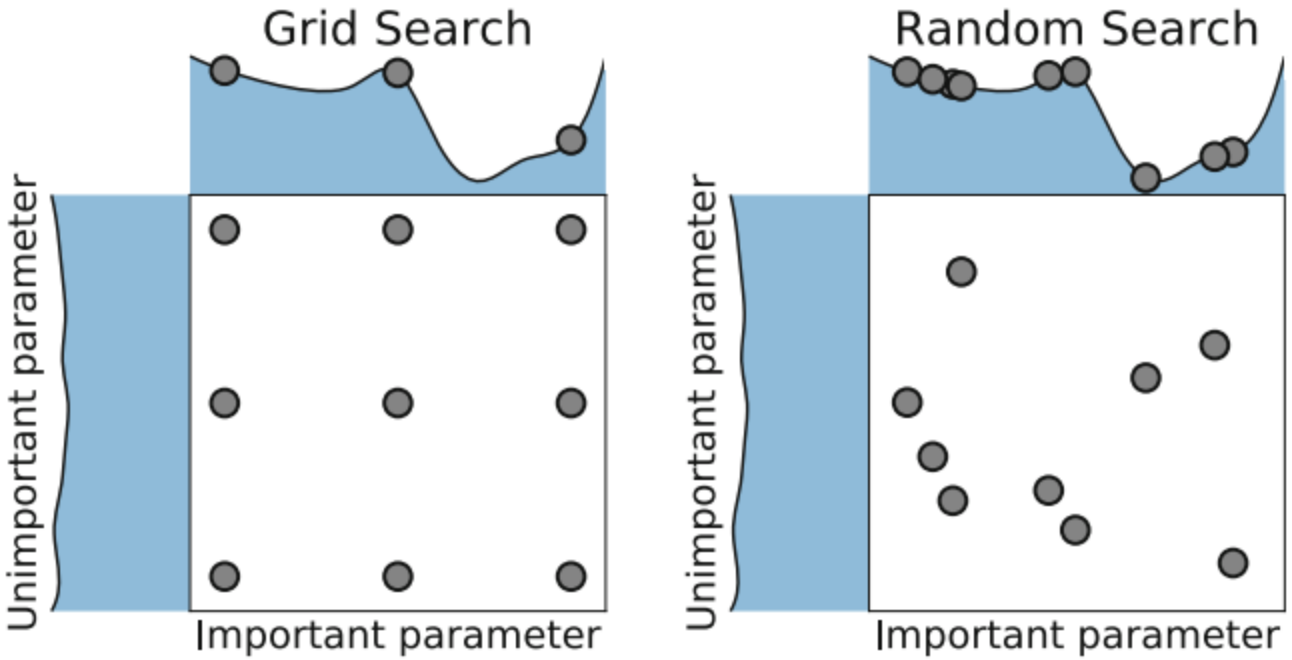
Optimization Framework



Popular search algorithms

- Random / grid search
- Bayesian optimization
- Gradient descent
- Genetic search

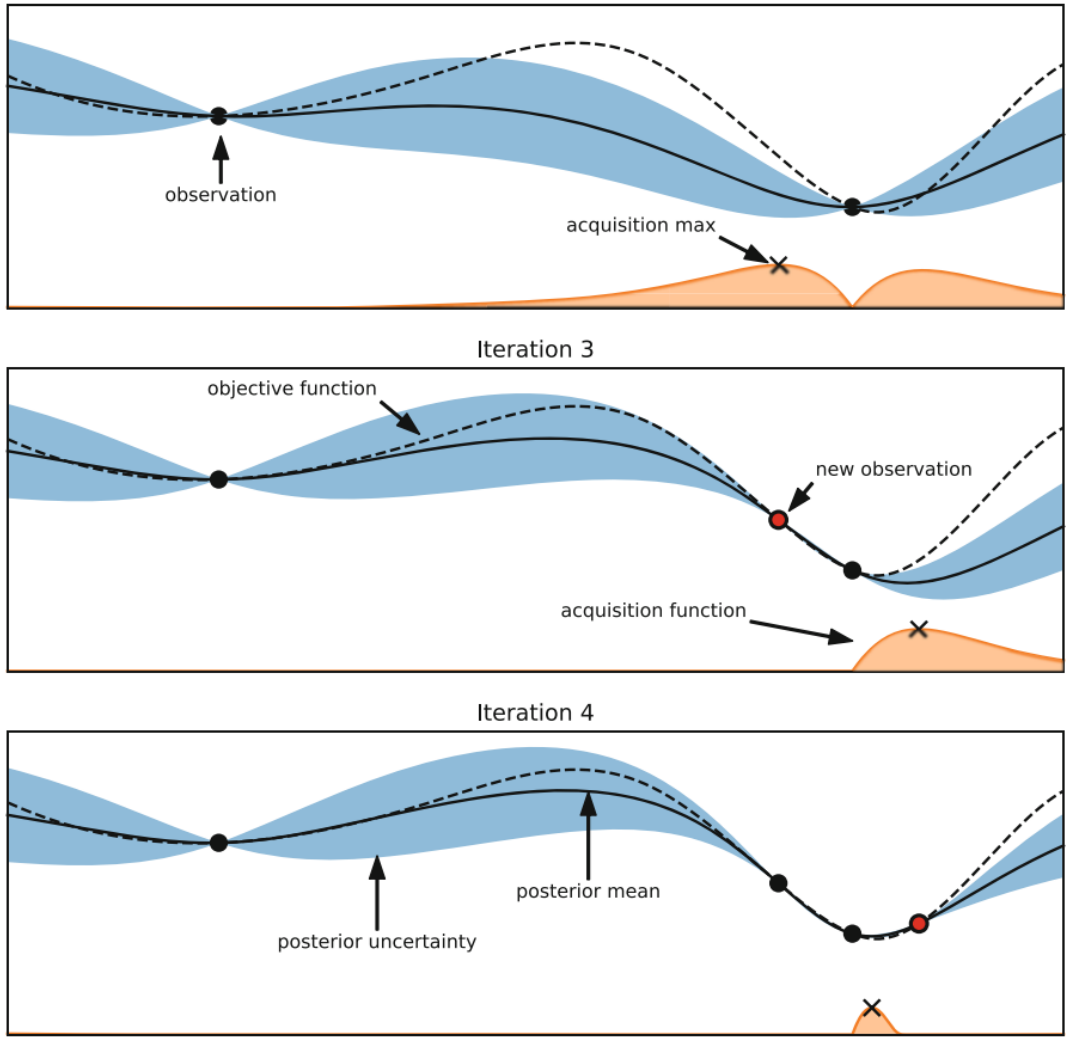
Exemplar Algorithm – Random Search



Random search works better than grid search when some hyperparameters are much more important than others

Random search for hyper-parameter optimization. JMLR 2012.

Exemplar Algorithm – Bayesian Optimization

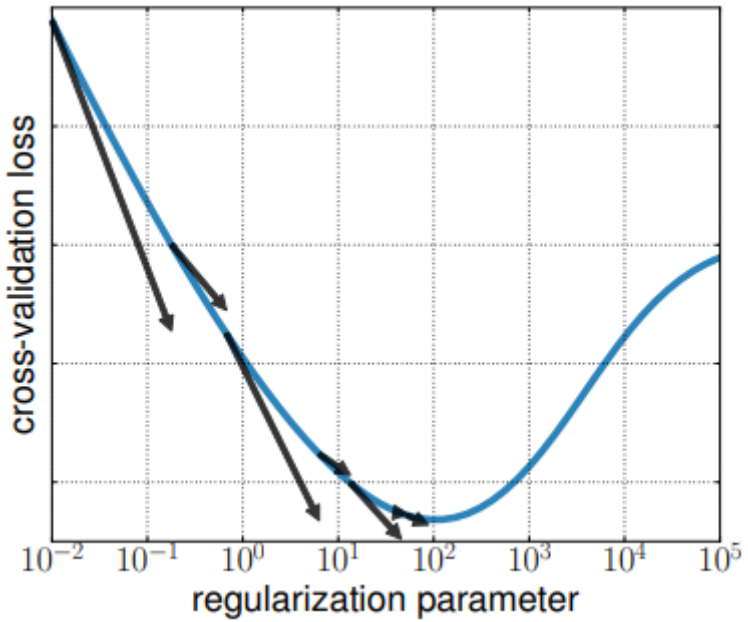


- a probabilistic surrogate model estimate M
- acquisition function to decide which point to evaluate next

Gaussian process is typically used as surrogate model, good at dealing with smooth M.

Practical bayesian optimization of machine learning algorithms. NIPS 2012

Exemplar Algorithm – Gradient descent



$$\min_{\lambda \in \mathcal{S}} M(F(w^*; \lambda), D_{\text{val}})$$

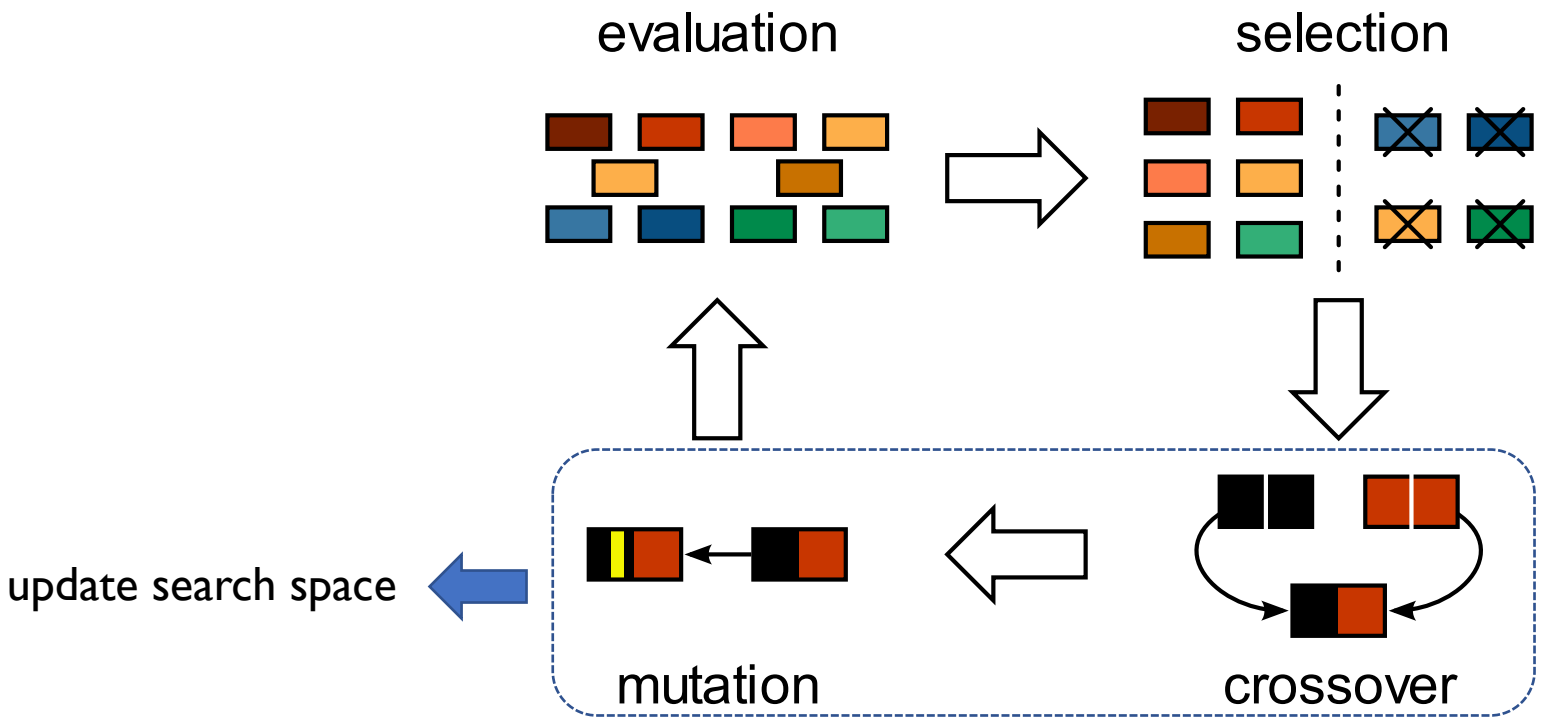
$$\text{s.t.} \quad w^* = \min_{w} L(F(w; \lambda), D_{\text{tra}})$$

w^* is also a function of λ

$$\nabla_\lambda M = \nabla_F M(F(w^*; \lambda), D_{val}) \nabla_\lambda F(w^*(\lambda); \lambda)$$
$$= \nabla_F M(F(w^*; \lambda), D_{val})(\nabla_{w^*} F(w^*; \lambda) \nabla_\lambda w^*(\lambda) + \nabla_\lambda F(w^*; \lambda))$$

Hyperparameter optimization with approximate gradient. *ICML* 2016

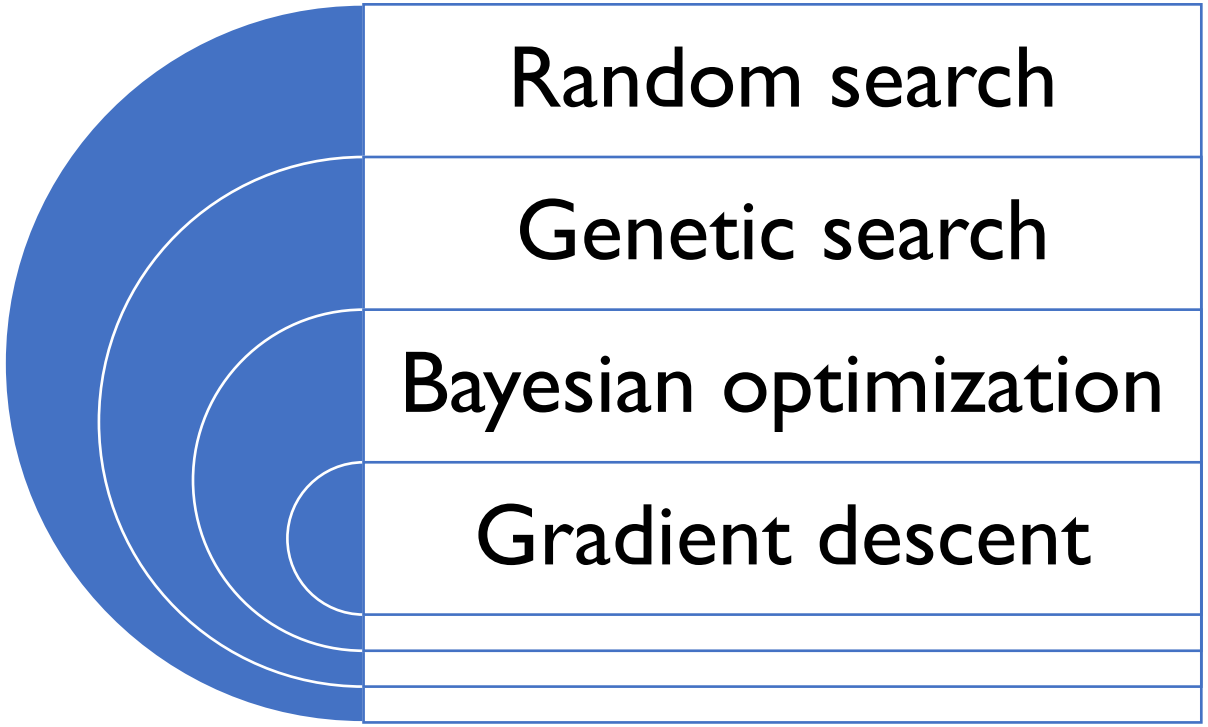
Exemplar Algorithm – Genetic search



Weakness of GS

- GS implementation is still an art
- design an objective function and getting the representation and operators right can be difficult
- computationally expensive i.e. time-consuming

General Comparison



Any function

Smooth one

More efficient

More restrictive

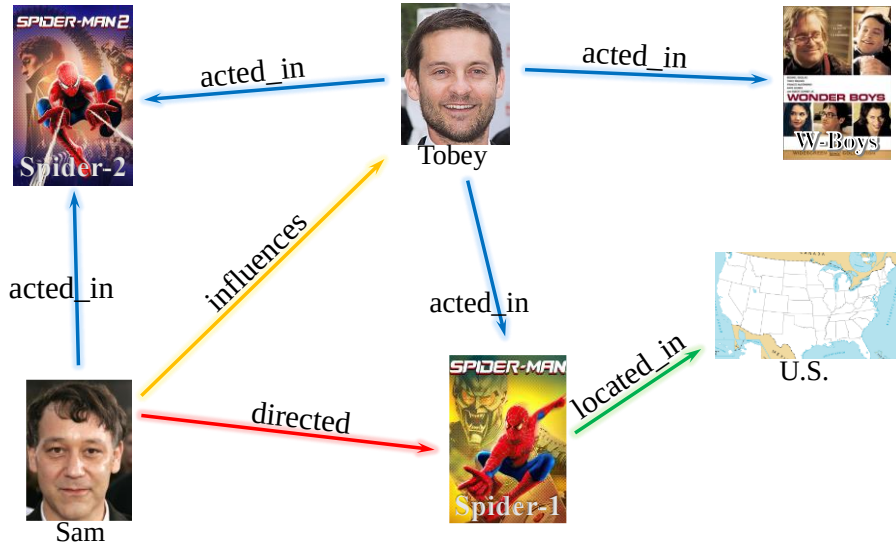
Some classical references:

- Random search for hyper-parameter optimization. JMLR 2012
- Genetic Algorithms in Search Optimization and Machine Learning, 1988
- Practical bayesian optimization of machine learning algorithms. NIPS 2012
- Hyperparameter optimization with approximate gradient. *ICML* 2016

Outline

- What's Hyper-parameter (HP) tuning
- What's Knowledge Graph (KG) learning
- Understand HP in KG learning
- An efficient two-stage HP search algorithm
- Experiments
- Key takeaway and future directions

Knowledge graph (KG)



Graph representation: $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$.

Entities $\mathcal{E}$: real world objects or abstract concepts.

Relations $\mathcal{R}$: interactions between/among entities.

Fact/triples $\mathcal{S}$: the basic unit in form of (head entity, relation, tail entity), (h, r, t).

Other related information:

- Types/attributes of entities/relations.
- Text descriptions on entities and relations.
- Ontologies: concept level description.
- Logic rules: regular expressions.

Google Knowledge Graph

范式星图 地址知识图谱工具

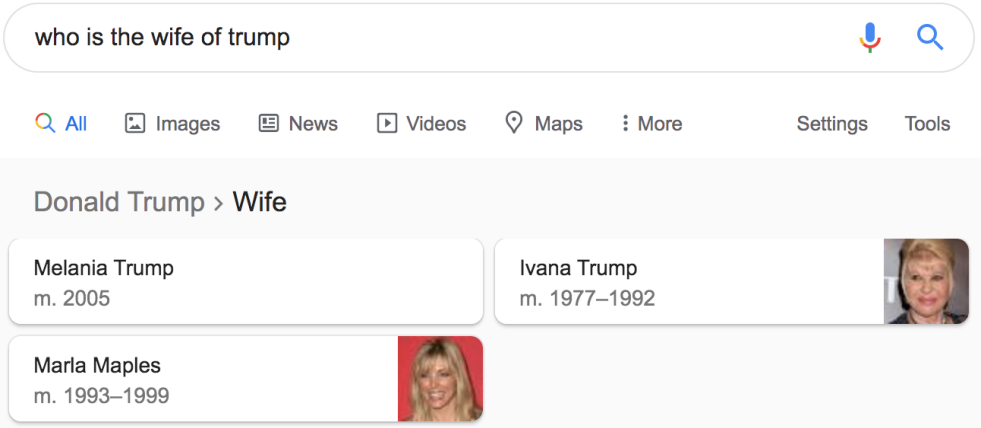
WIKIDATA

Freebase™

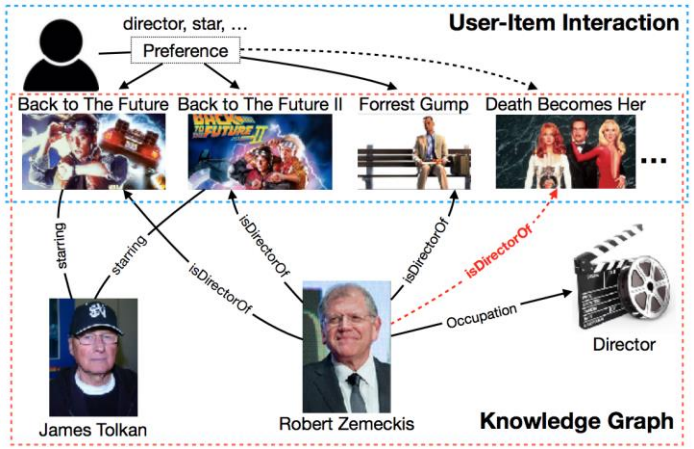
BIO2RDF

KG – important applications

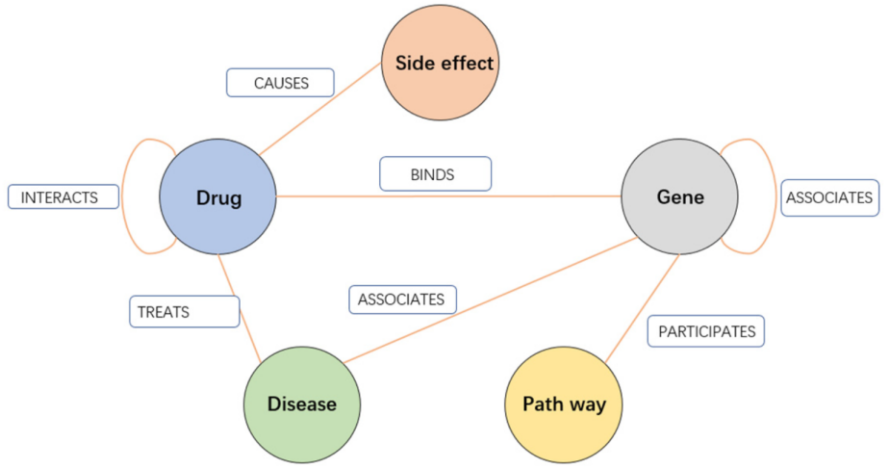
KGQA:



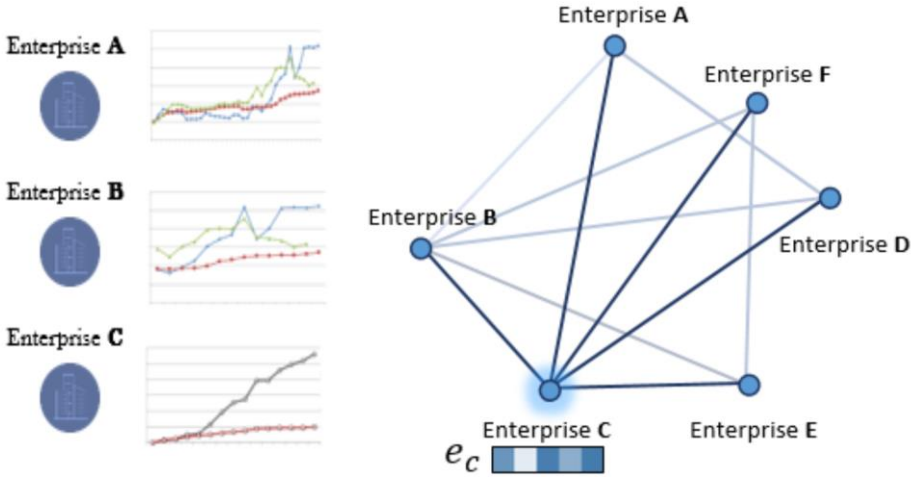
Recommendation:



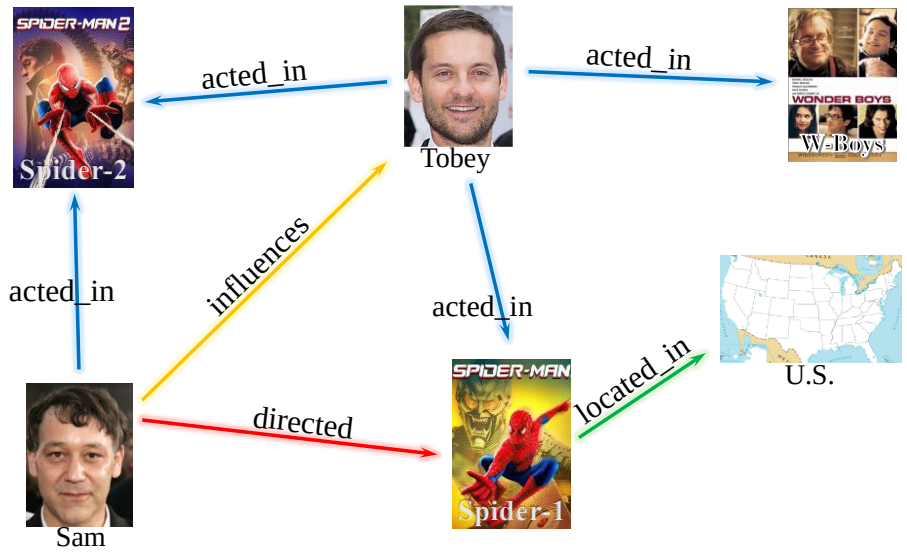
Drug discovery:



Stock prediction:



Representation learning

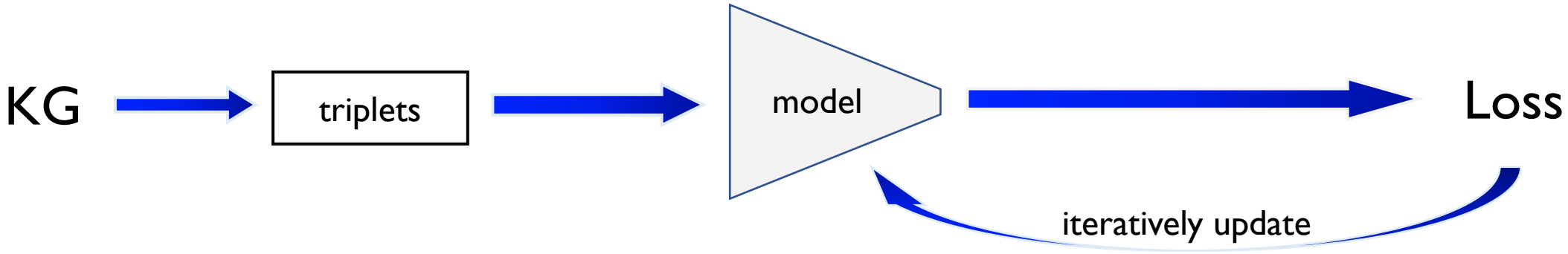


Observed triplets S^+:
maximize score

(Sam, directed, Spider-1)
(Tobey, acted_in, Spider-2)
(Spider-1, located_in, U.S.)
…

Unobserved triplets S^-:
minimize score

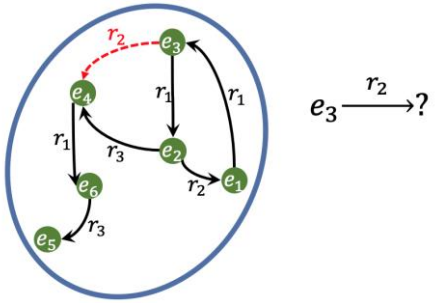
(Tobey, directed, Spider-1)
(Tobey, acted_in, U.S.)
(Spider-1, directed, U.S.)
…



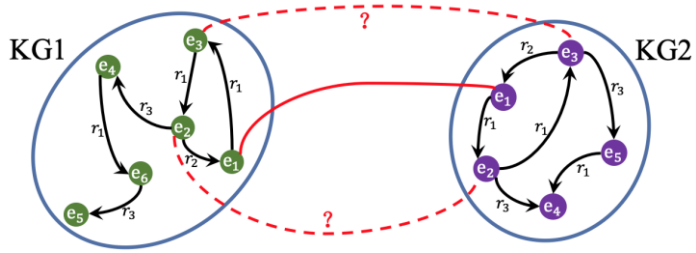
➢Objective: Preserve as much information on original graph as possible.

Basic tasks

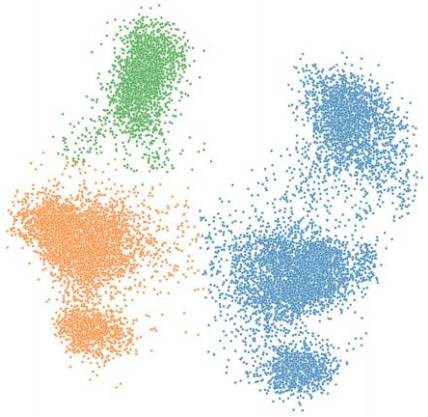
- Knowledge graph completion
  - Link prediction / triple classification: predicting missing links



- Entity alignment
  - Align the same entities in two KGs, e.g. cross-lingual KGs



- Entity classification
  - Predict the type of entities



Evaluation

- Link prediction as an example.

- Evaluation on (h, r, t):
  - head/tail prediction: $(?, r, t)$ or $(h, r, ?)$
  - get the rank of (h, r, t) over $\{(e, r, t): e \in \mathcal{E}\}$ or $\{(h, r, e): e \in \mathcal{E}\}$

- Metrics:

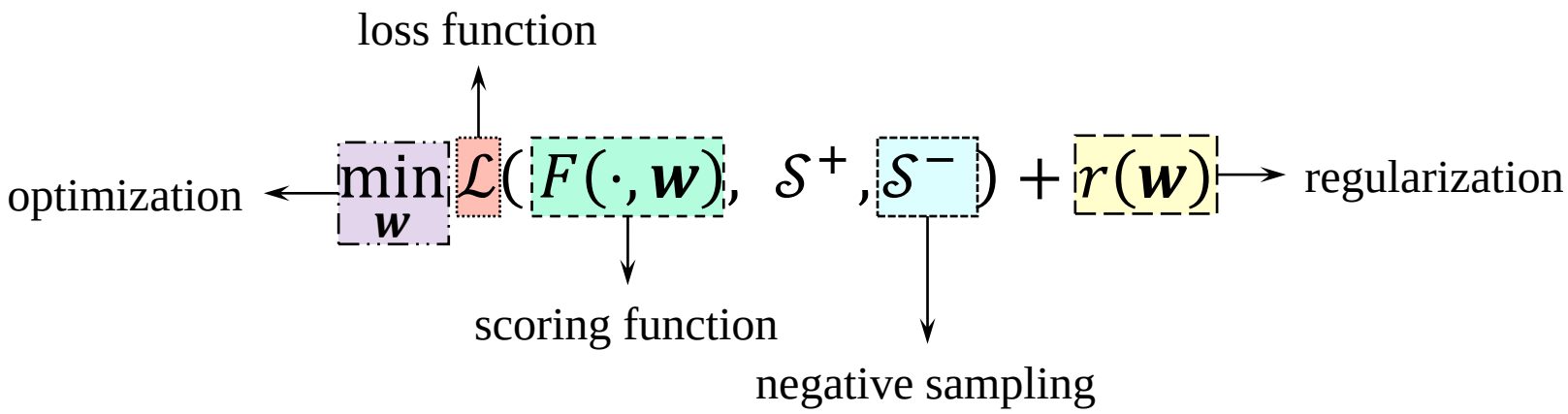
$$MR = \frac{1}{|\mathcal{S}_{\text{tst}}|} \sum_{i \in \mathcal{S}_{tst}} \text{rank}_i \qquad MRR = \frac{1}{|\mathcal{S}_{\text{tst}}|} \sum_{i \in \mathcal{S}_{tst}} \frac{1}{\text{rank}_i} \qquad H@K = \frac{|\{i \in \mathcal{S}_{\text{tst}}: \text{rank}_i \leq K\}|}{|\mathcal{S}_{\text{tst}}|}$$

Common Learning Objective

For setting up a KGE system, we need

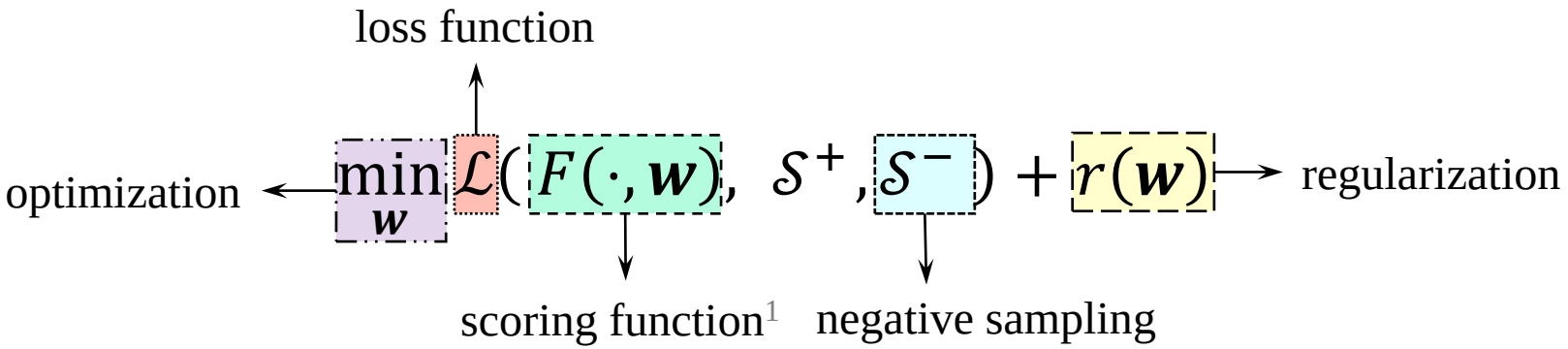
- A **scoring function F** to discriminate positive/negative triples
- A sampling scheme to generate **negative samples S^-**
- A **loss function L** and **regularization r** to define the learning problem
- An **optimization** strategy for convergence procedure

We can formulate the learning framework as:



Common Hyper-parameters

KGE components and related hyper-parameters (HPs)



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}

1: Note that HPs in SF are not covered here

Typical HP Configuration

KGE components and related hyper-parameters

hyper-parameter

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}

A configuration

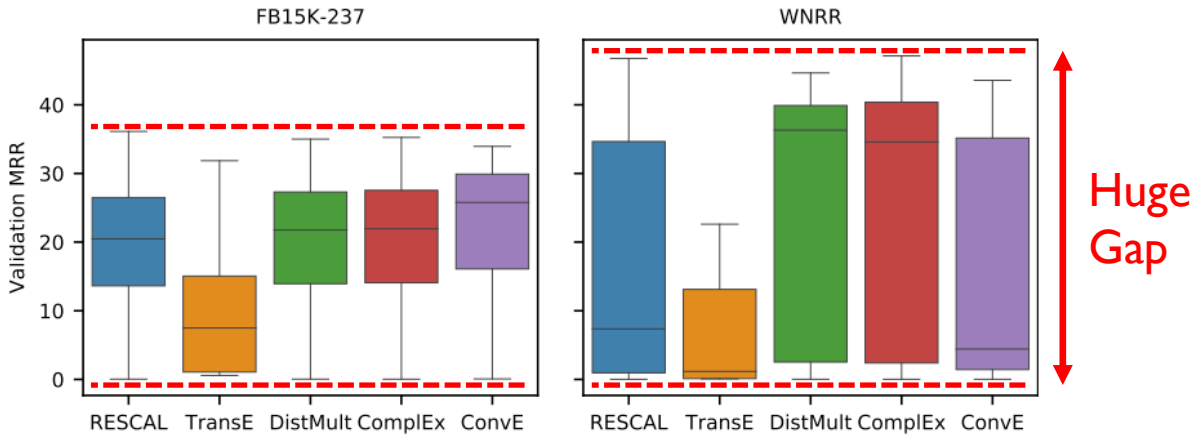
# negative samples	512
loss function	BCE_adv
gamma	0.00
adv. weight	0.57
regularizer	DURA
reg. weight	$8.64 * 10^{-3}$
dropout rate	0.25
optimizer	Adam
learning rate	$1.77 * 10^{-3}$
initializer	xavier_norm
batch size	512
dimension size	1000
inverse relation	False

Performance Comparison

		FB15k				WN18				FB15k-237				WN18RR				YAGO3-10			
		H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR
Tensor Decomposition Models	DistMult	73.61	86.32	173	0.784	72.60	94.61	675	0.824	22.44	49.01	199	0.313	39.68	50.22	5913	0.433	41.26	66.12	1107	0.501
	ComplEx	81.56	90.53	34	0.848	94.53	95.50	3623	0.949	25.72	52.97	202	0.349	42.55	52.12	4907	0.458	50.48	70.35	1112	0.576
	ANALOGY	65.59	83.74	126	0.726	92.61	94.42	808	0.934	12.59	35.38	476	0.202	35.82	38.00	9266	0.366	19.21	45.65	2423	0.283
	SimplE	66.13	83.63	138	0.726	93.25	94.58	759	0.938	10.03	34.35	651	0.179	38.27	42.65	8764	0.398	35.76	63.16	2849	0.453
	HolE	75.85	86.78	211	0.800	93.11	94.94	650	0.938	21.37	47.64	186	0.303	40.28	48.79	8401	0.432	41.84	65.19	6489	0.502
	TuckER	72.89	88.88	39	0.788	94.64	95.80	510	0.951	25.90	53.61	162	0.352	42.95	51.40	6239	0.459	46.56	68.09	2417	0.544
Geometric Models	TransE	49.36	84.73	45	0.628	40.56	94.87	279	0.646	21.72	49.65	209	0.31	2.79	49.52	3936	0.206	40.57	67.39	1187	0.501
	STransE	39.77	79.60	69	0.543	43.12	93.45	208	0.656	22.48	49.56	357	0.315	10.13	42.21	5172	0.226	3.28	7.35	5797	0.049
	CrossE	60.08	86.23	136	0.702	73.28	95.03	441	0.834	21.21	47.05	227	0.298	38.07	44.99	5212	0.405	33.09	65.45	3839	0.446
	TorusE	68.85	83.98	143	0.746	94.33	95.44	525	0.947	19.62	44.71	211	0.281	42.68	53.35	4873	0.463	27.43	47.44	19455	0.342
	RotatE	73.93	88.10	42	0.791	94.30	96.02	274	0.949	23.83	53.06	178	0.336	42.60	57.35	3318	0.475	40.52	67.07	1827	0.498
Deep Learning Models	ConvE	59.46	84.94	51	0.688	93.89	95.68	413	0.945	21.90	47.62	281	0.305	38.99	50.75	4944	0.427	39.93	65.75	2429	0.488
	ConvKB	11.44	40.83	324	0.211	52.89	94.89	202	0.709	13.98	41.46	309	0.230	5.63	52.50	3429	0.249	32.16	60.47	1683	0.420
	ConvR	70.57	88.55	70	0.773	94.56	95.85	471	0.950	25.56	52.63	251	0.346	43.73	52.68	5646	0.467	44.62	67.33	2582	0.527
	CapsE	1.93	21.78	610	0.087	84.55	95.08	233	0.890	7.34	35.60	405	0.160	33.69	55.98	720	0.415	0.00	0.00	60676	0.000
	RSN	72.34	87.01	51	0.777	91.23	95.10	346	0.928	19.84	44.44	248	0.280	34.59	48.34	4210	0.395	42.65	66.43	1339	0.511
	AnyBURL	81.09	87.86	288	0.835	94.63	95.96	233	0.951	24.03	48.93	480	0.324	44.93	55.97	2530	0.485	45.83	66.07	815	0.528

No best models

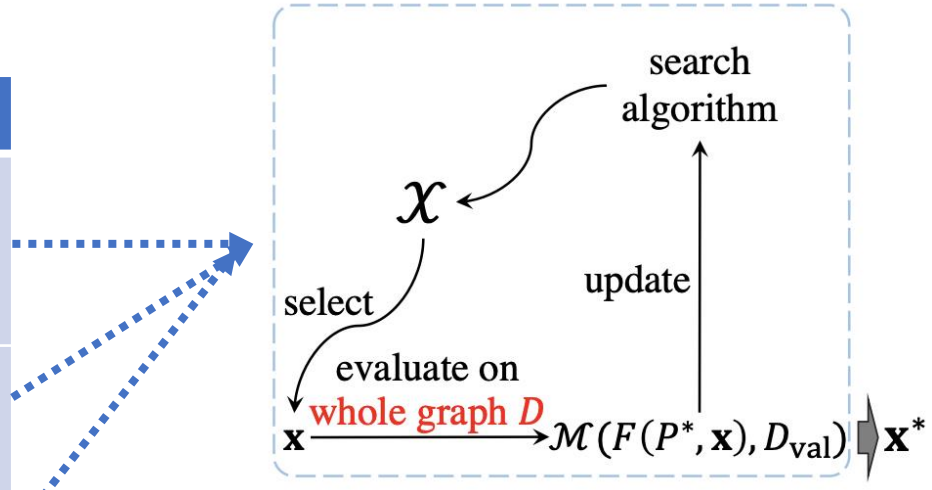
		RESCAL	TransE	DistMult	ComplEx	ConvE
FB15K-237	Valid. MRR	*36.1*	*31.5*	*35.0*	*35.3*	*34.3*
	Emb. size	128 *(-0.5)*	512 *(-3.7)*	256 *(-0.2)*	256 *(-0.3)*	256 *(-0.4)*
	Batch size	512 *(-0.5)*	128 *(-7.1)*	1024 *(-0.2)*	1024 *(-0.3)*	1024 *(-0.4)*
	Train type	1vsAll *(-0.8)*	NegSamp –	NegSamp *(-0.2)*	NegSamp *(-0.3)*	1vsAll *(-0.4)*
	Loss	CE *(-0.9)*	CE *(-7.1)*	CE *(-3.1)*	CE *(-3.8)*	CE *(-0.4)*
	Optimizer	Adam *(-0.5)*	Adagrad *(-3.7)*	Adagrad *(-0.2)*	Adagrad *(-0.5)*	Adagrad *(-1.5)*
	Initializer	Normal *(-0.8)*	XvNorm *(-3.7)*	Unif. *(-0.2)*	Unif. *(-0.5)*	XvNorm *(-0.4)*
	Regularizer	None *(-0.5)*	L2 *(-3.7)*	L3 *(-0.2)*	L3 *(-0.3)*	L3 *(-0.4)*
	Reciprocal	No *(-0.5)*	Yes *(-9.5)*	Yes *(-0.3)*	Yes *(-0.3)*	Yes –
WNRR	Valid. MRR	*46.8*	*22.6*	*45.4*	*47.6*	*44.3*
	Emb. size	128 *(-1.0)*	512 *(-5.1)*	512 *(-1.1)*	128 *(-1.0)*	512 *(-1.2)*
	Batch size	128 *(-1.0)*	128 *(-5.1)*	1024 *(-1.1)*	512 *(-1.0)*	1024 *(-1.3)*
	Train type	KvsAll *(-1.0)*	NegSamp –	KvsAll *(-1.1)*	1vsAll *(-1.0)*	KvsAll *(-1.2)*
	Loss	CE *(-2.0)*	CE *(-5.1)*	CE *(-2.4)*	CE *(-3.5)*	CE *(-1.4)*
	Optimizer	Adam *(-1.2)*	Adagrad *(-5.8)*	Adagrad *(-1.5)*	Adagrad *(-1.5)*	Adam *(-1.4)*
	Initializer	Unif. *(-1.0)*	XvNorm *(-5.1)*	Unif. *(-1.3)*	Unif. *(-1.5)*	XvNorm *(-1.4)*
	Regularizer	L3 *(-1.2)*	L2 *(-5.1)*	L3 *(-1.1)*	L2 *(-1.0)*	L1 *(-1.2)*
	Reciprocal	Yes *(-1.0)*	Yes *(-5.9)*	Yes *(-1.1)*	No *(-1.0)*	Yes –

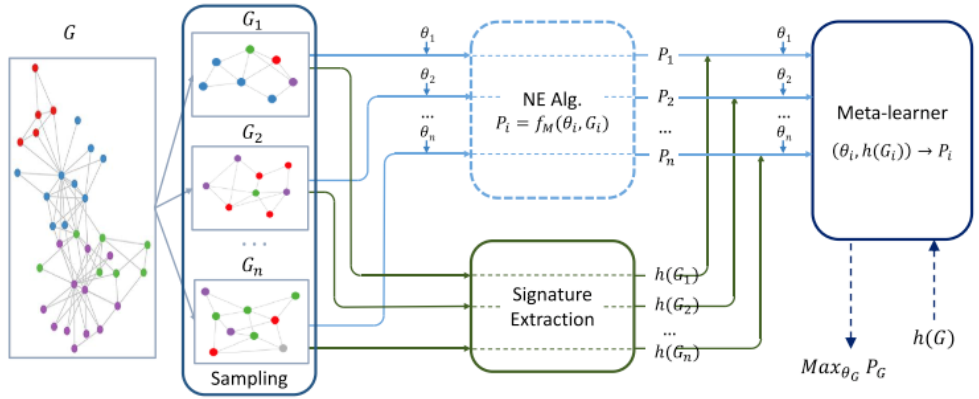


No best hyper-parameters

Existing HP tuning Methods

Taxonomy	Examples	Cons
Sampled-based	Grid search	Low efficiency Can not learn from historical records
	Random search	
Bayesian optimization	Hyperopt (TPE) [1]	Slow feedback from the original KG
	SMAC (RF) [2]	
	Ax (GP) [3]	
	AutoNE [4]	(Subgraph-based) No specialized designs for KGE
	e-AutoGR [5]	
	$\phi(HP\ configuration) \rightarrow Performance$	





[1] Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms.
[2] Sequential model-based optimization for general algorithm configuration.
[3] https://github.com/facebook/Ax
[4] Autone: Hyperparameter optimization for massive network embedding.
[5] Explainable automated graph representation learning with hyperparameter importance.

Outline

- What's Hyper-parameter (HP) tuning
- What's Knowledge Graph (KG) learning
- Understand HP in KG learning
- An efficient two-stage HP search algorithm
- Experiments
- Key takeaway and future directions

Motivation and Objective

Major disadvantages of existing works

Low efficiency in searching for HP configuration

- usually in a time-consuming trial-and-error way
- interaction, importance, and tunability of HPs are unclear
- lacking understanding of KG learning components

Objective of KGTuner:

- *Design a searching algorithm,*
- *for any given dataset and embedding model with limited budget,*
- *to **efficiently** search for the hyper-parameter configuration.*

KGTuner: Efficient Hyper-parameter Search for Knowledge Graph Learning. ACL 2022

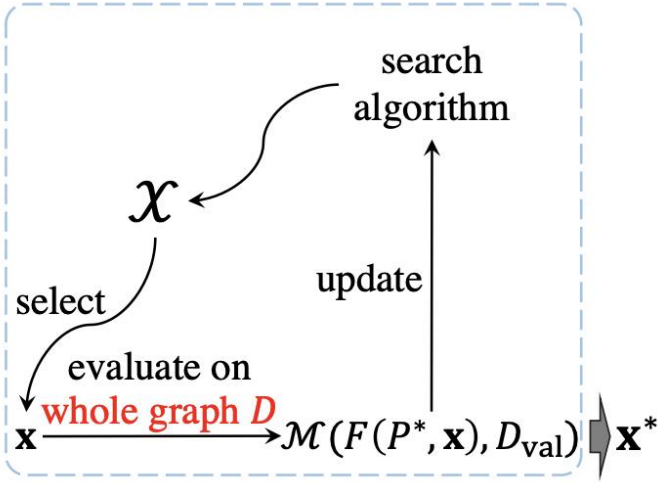
Problem Definition

HP searching problem setup

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

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

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



Three major aspects for efficiency in Def. I

the **size** of search space χ	make the space smaller
the validation **curvature** of $\mathcal{M}$	capture the curvature better
the evaluation **cost** in solving $argmin_{\mathcal{P}}$	obtain cheap and accurate evaluations

Outline

- Understand HP in KG learning
  - Search space
  - Validation curvature
  - Evaluation cost

Understanding the Search Space

Recall the search space

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	[0, 0.5]
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}

Three major aspects for efficiency in Def. 1

1. the size of search space χ
2. the validation curvature of $\mathcal{M}$
3. the evaluation cost in solving $argmin_{\mathcal{P}}$

Questions to be answered

- What are the properties of each HP?
  - ranking distribution
  - consistency
  - computing cost
- Can we decrease the range for each HP?
- Can we decouple some HPs?

Understanding the Search Space

Excavating properties of HPs

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	[0, 0.5]
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}

enumerating

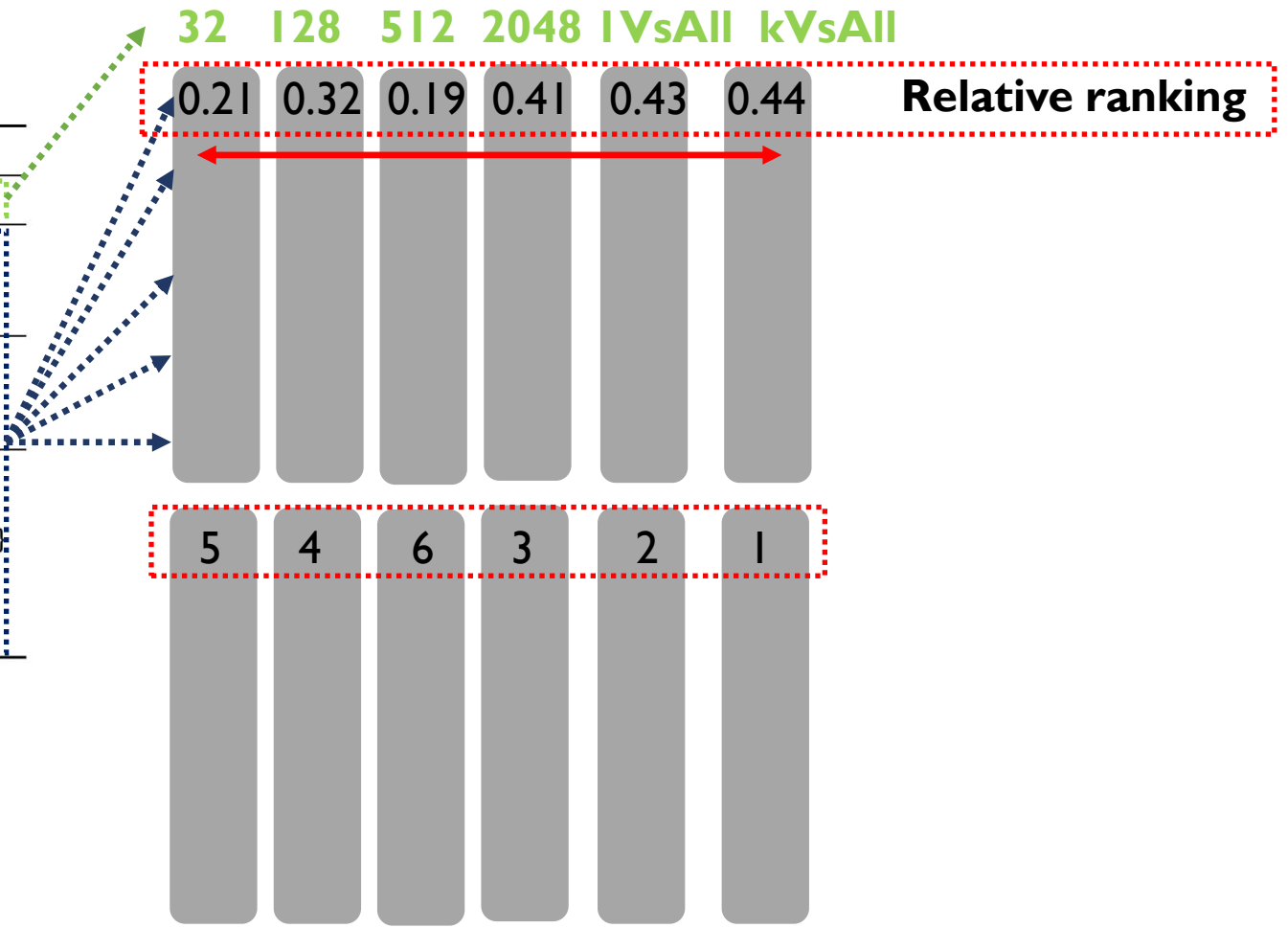
sampling [1]

[1] Design Space for Graph Neural Networks

Understanding the Search Space

Excavating properties of HPs

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	[0, 0.5]
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}



enumerating

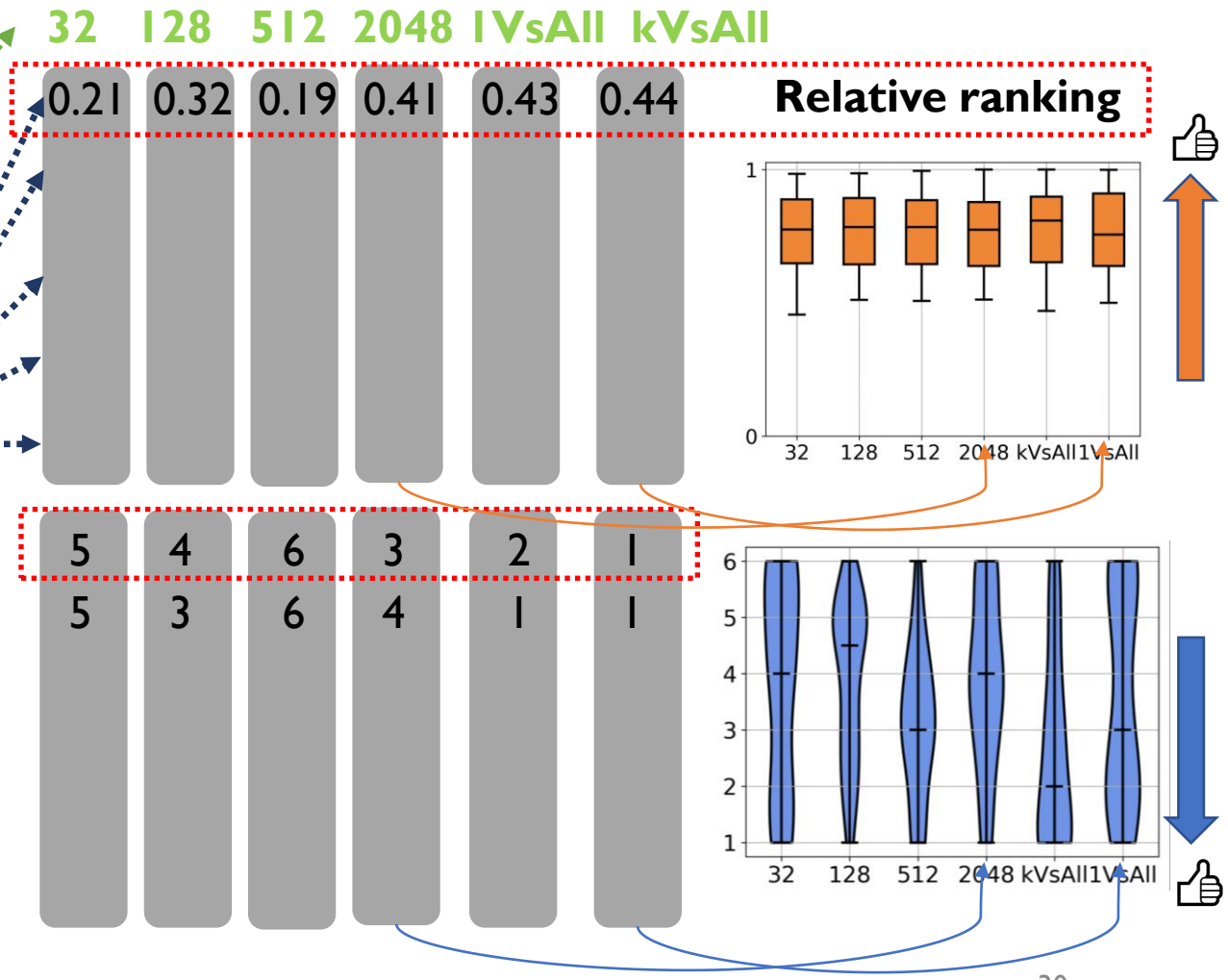
sampling [1]

[1] Design Space for Graph Neural Networks

Understanding the Search Space

Excavating properties of HPs

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	[0, 0.5]
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}



enumerating

sampling [1]

[1] Design Space for Graph Neural Networks

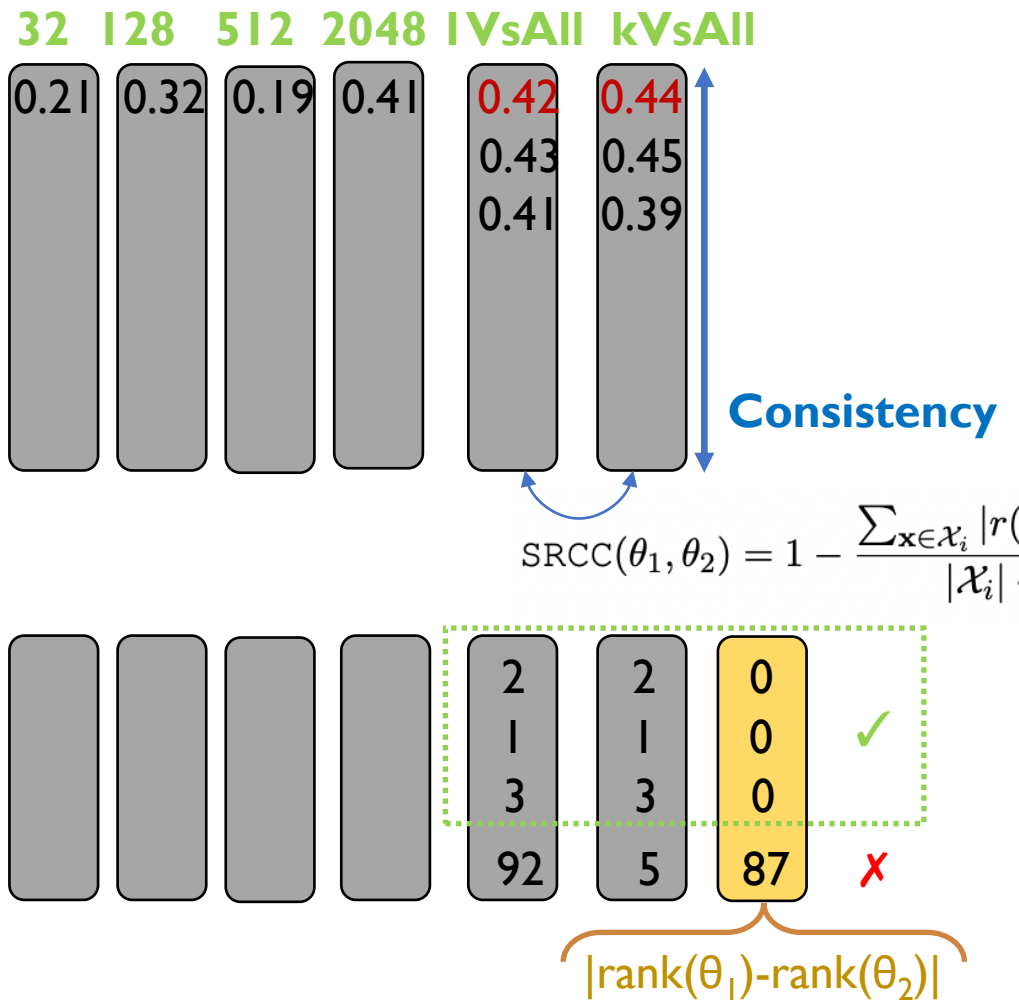
Understanding the Search Space

Excavating properties of HPs

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	[0, 0.5]
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}

enumerating

sampling

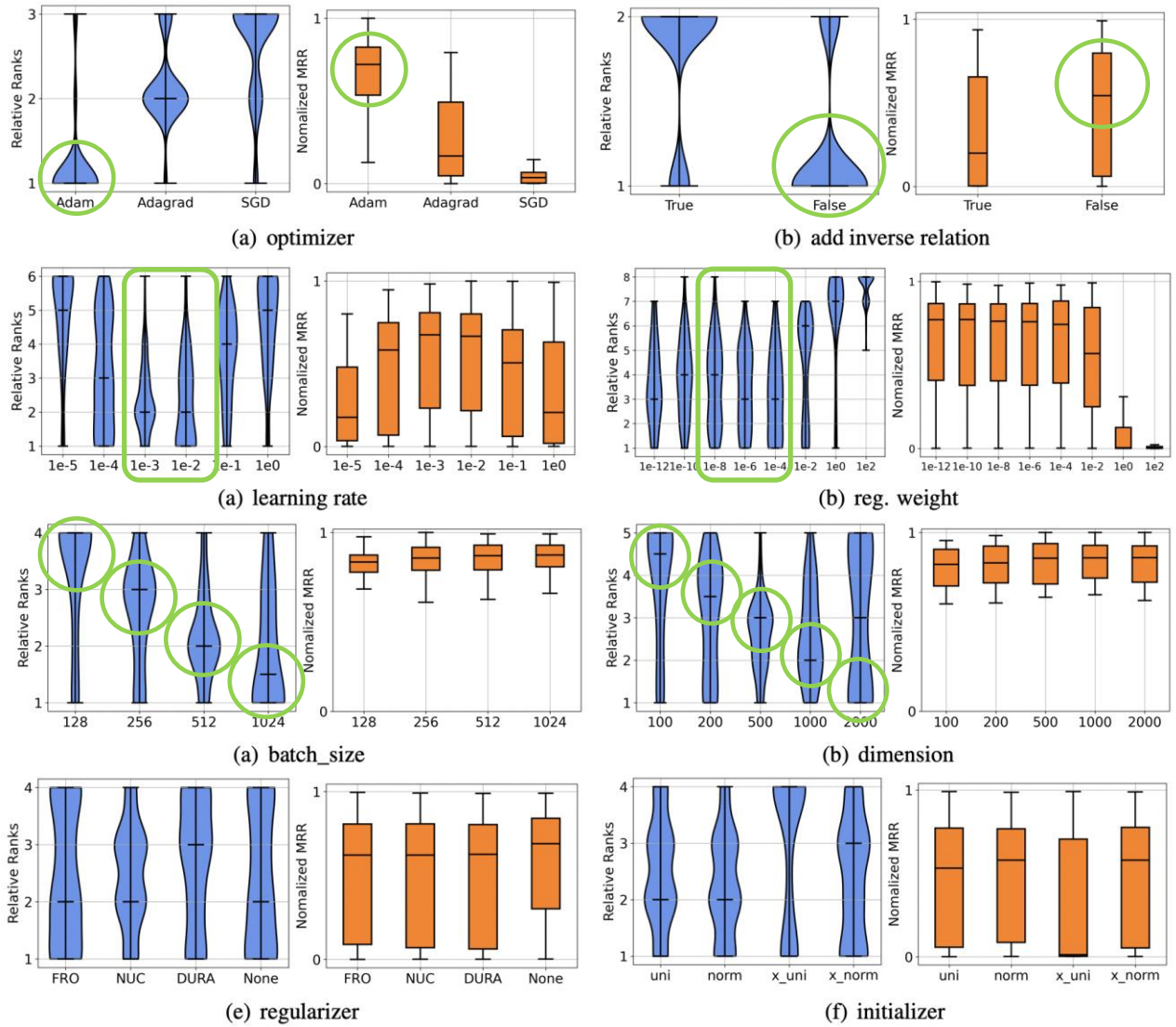


Understanding the Search Space

Excavating properties of HPs | Ranking/Performance distribution

The HPs can be classified into 4 groups

1. fixed choice
2. limited range
3. monotonously related
4. no obvious patterns



✗ lower rankings

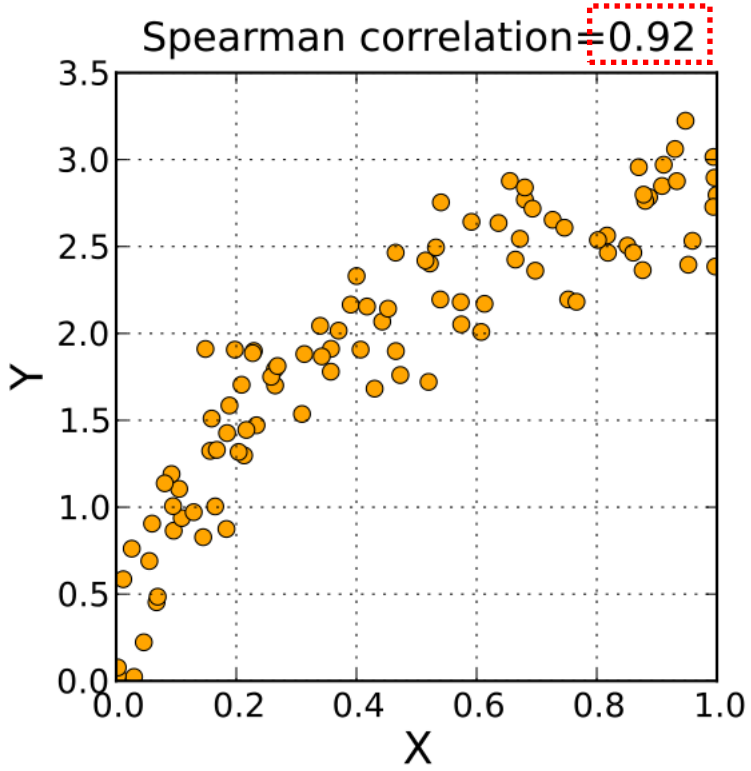
✓ higher rankings

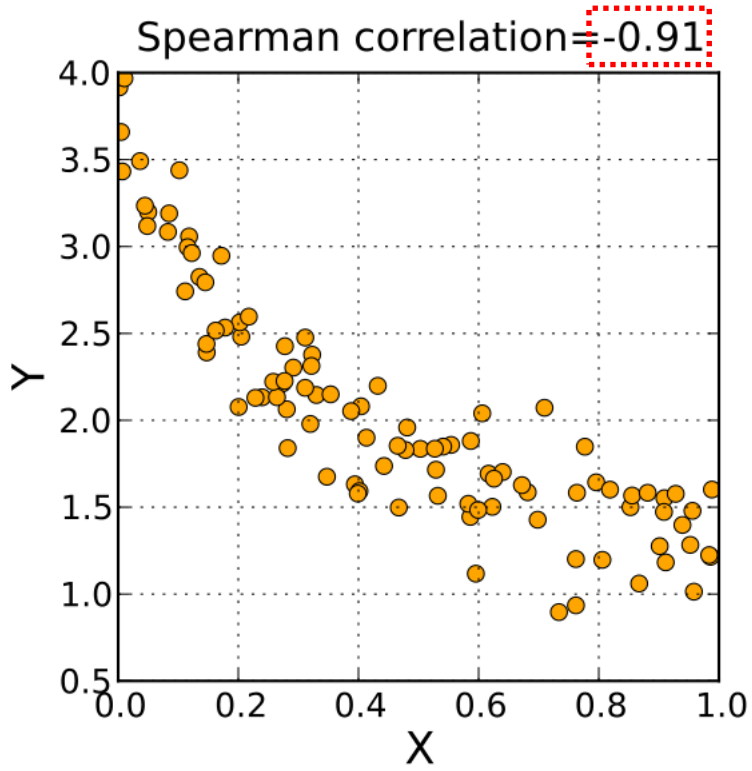
Understanding the Search Space

Positive and negative Spearman rank correlations

$$\mathrm{SRCC}(\theta_1, \theta_2) = 1 - \frac{\sum_{\mathbf{x} \in \mathcal{X}_i} |r(\mathbf{x}, \theta_1) - r(\mathbf{x}, \theta_2)|}{|\mathcal{X}_i| \cdot (|\mathcal{X}_i|^2 - 1)}$$

positive





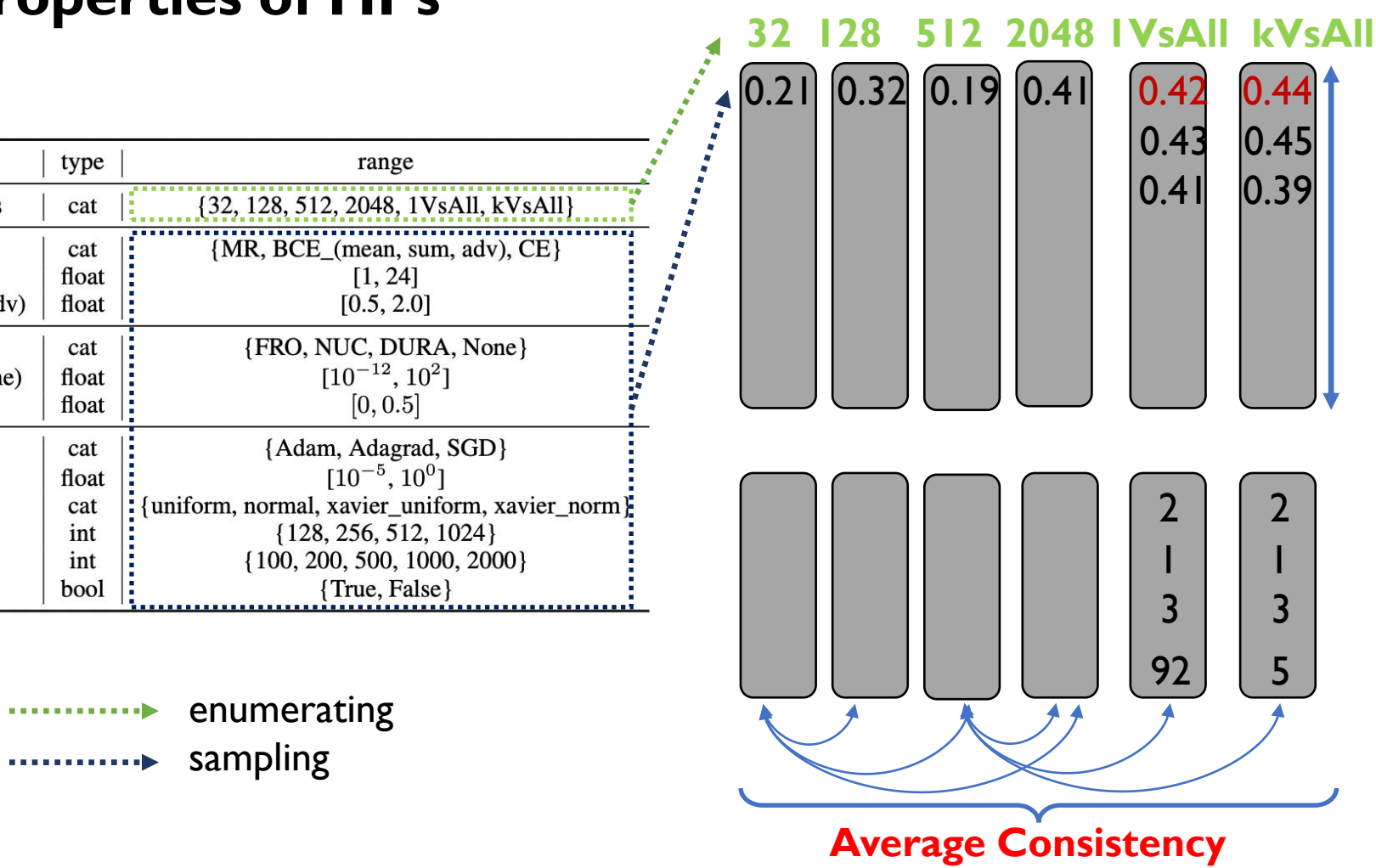
negative

https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

Understanding the Search Space

Excavating properties of HPs

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	[0, 0.5]
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}



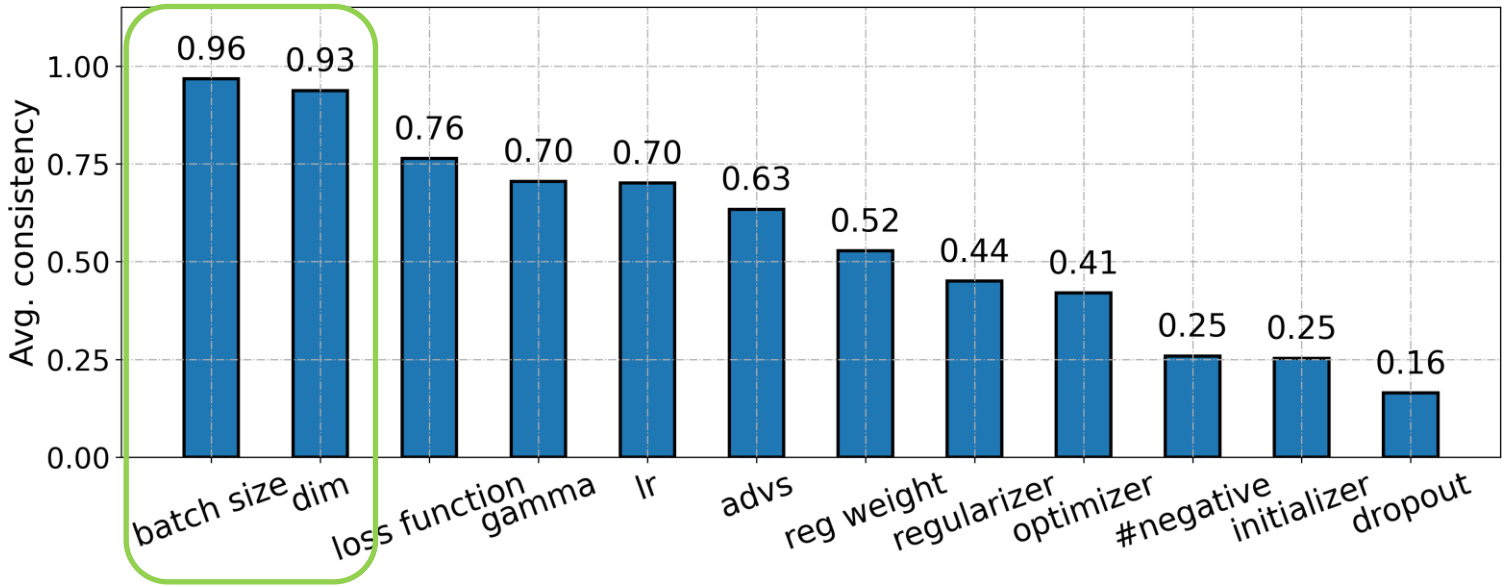
enumerating

sampling

Understanding the Search Space

Excavating properties of HPs | Consistency

Observation:
the batchsize and dimension show higher consistency than the other HPs.

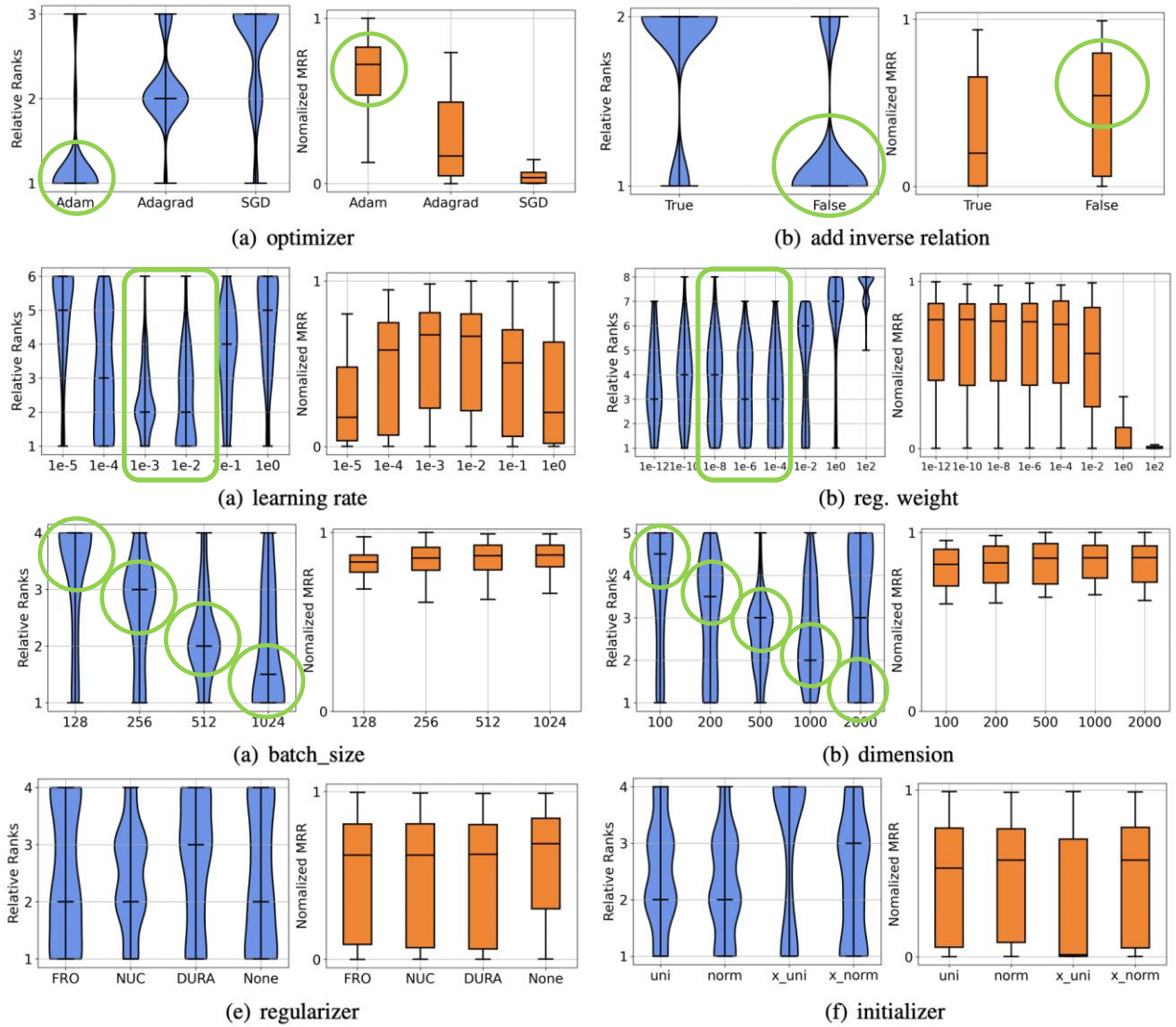


Understanding the Search Space

Excavating properties of HPs | Ranking/Performance distribution

The HPs can be classified into 4 groups

1. fixed choice
2. limited range
3. monotonously related
4. no obvious patterns



✗ lower rankings

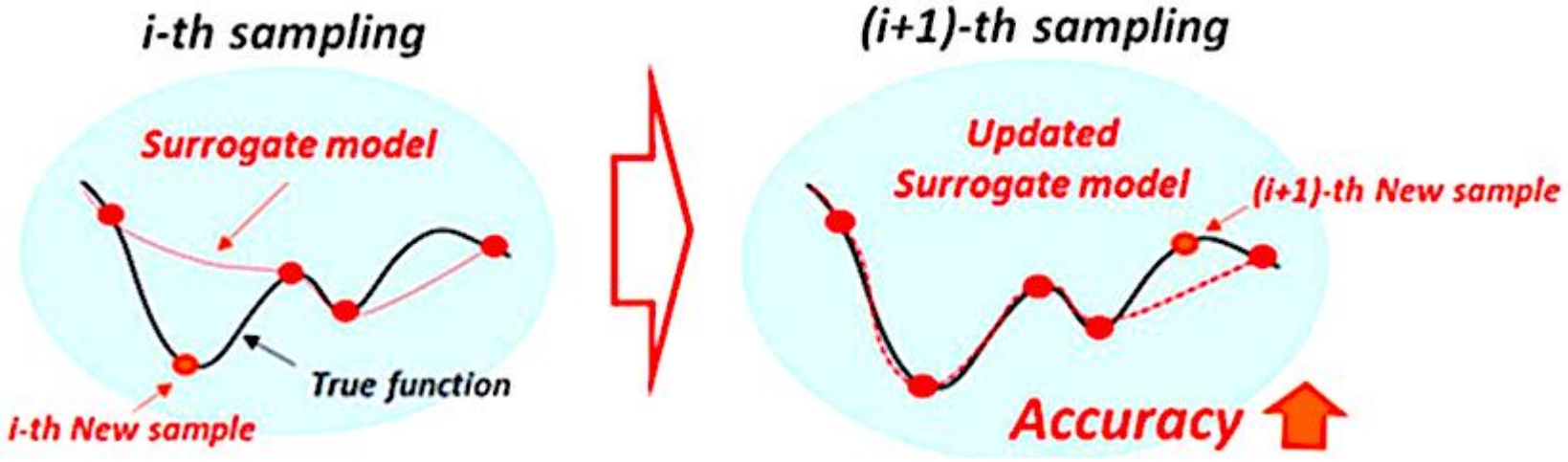
✓ higher rankings

Outline

- Understand HP in KG learning
  - Search space
  - Validation curvature
  - Evaluation cost

Understanding the Curvature

Surrogate: estimate the ground-truth function with given observations



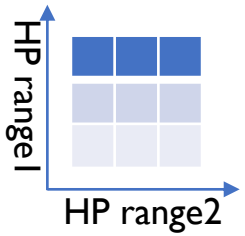
Requirements

- Accurate approximation to the ground-truth
- Easy to be updated with new observations

What can be a bad surrogate?

Understand HP in KG Learning

Excavating properties of HPs | from the aspect of predictor



Predictors

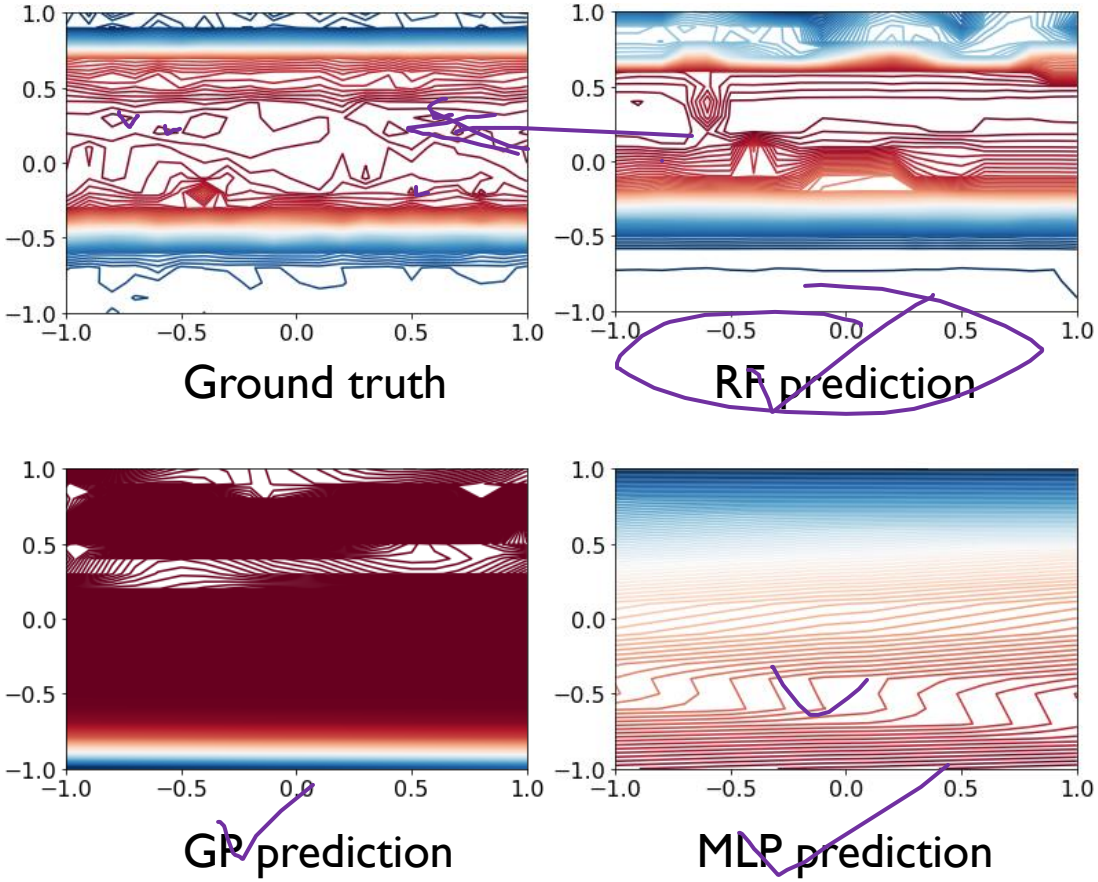
- Gaussian process (GP)
- Multi layer perceptron (MLP)
- Random forest (RF)

Observation:
RF is better in approximating the curvature

# train configurations	10	20	30
GP	0.0693±0.02	0.029±0.01	0.019±0.01
MLP	2.121±0.4	2.052±0.3	0.584±0.1
RF	**0.003±0.002**	**0.002±0.001**	**0.001±0.001**

MSE results

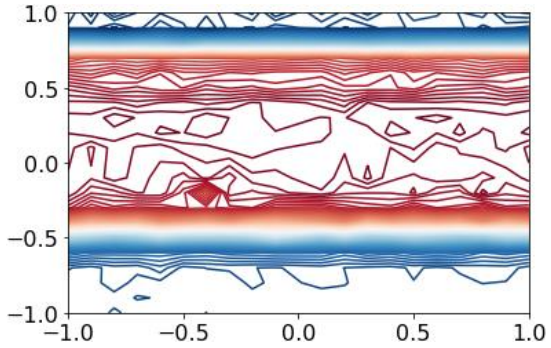
Validation curvature



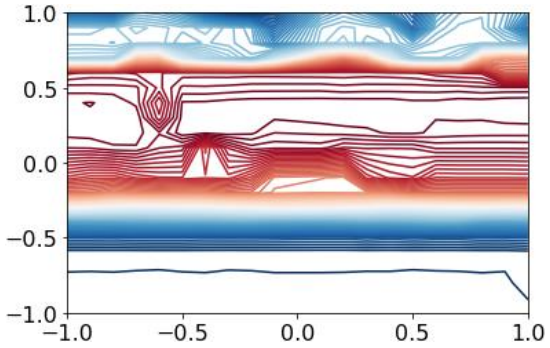
Understand HP in KG Learning

Excavating properties of HPs | from the aspect of predictor

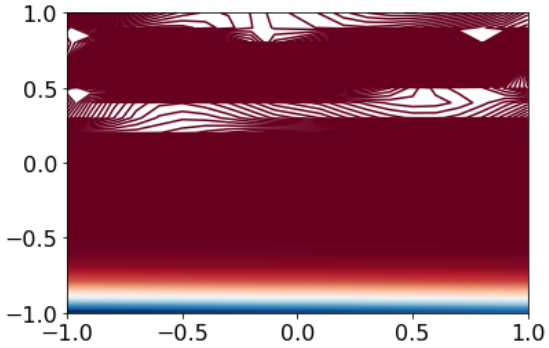
Ground truth



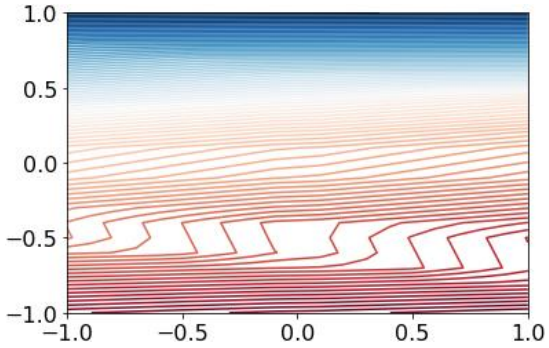
RF prediction

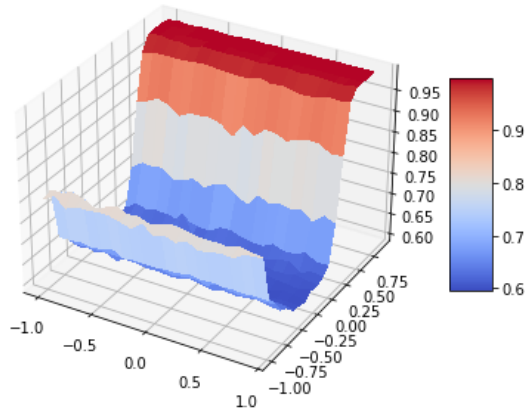


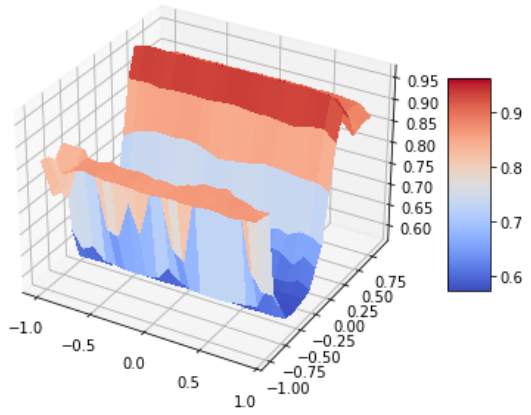
GP prediction

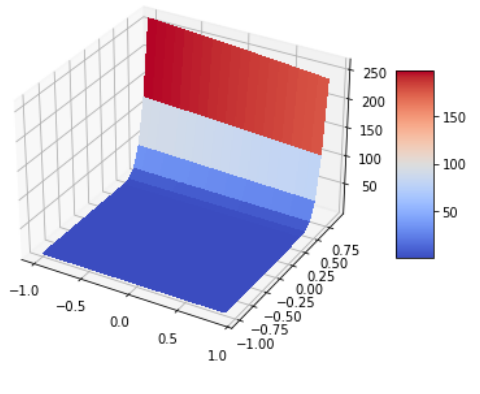


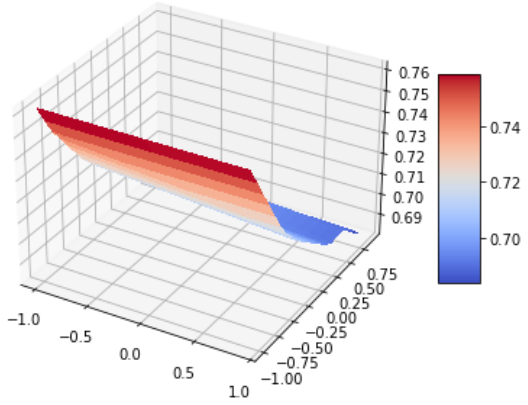
MLP prediction











Outline

- Understand HP in KG learning
  - Search space
  - Validation curvature
  - Evaluation cost

Understand HP in KG Learning

Excavating properties of HPs | Time cost

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

dataset	#entity	#relation	#train	#validate	#test
WN18RR (Dettmers et al., 2017)	41k	11	87k	3k	3k
FB15k-237 (Toutanova and Chen, 2015)	15k	237	272k	18k	20k
ogbl-biokg (Hu et al., 2020)	94k	51	4,763k	163k	163k
ogbl-wikikg2 (Hu et al., 2020)	2,500k	535	16,109k	429k	598k

Average evaluation time cost:

~2.1h
~3.5h
~17.3h
~21.7h

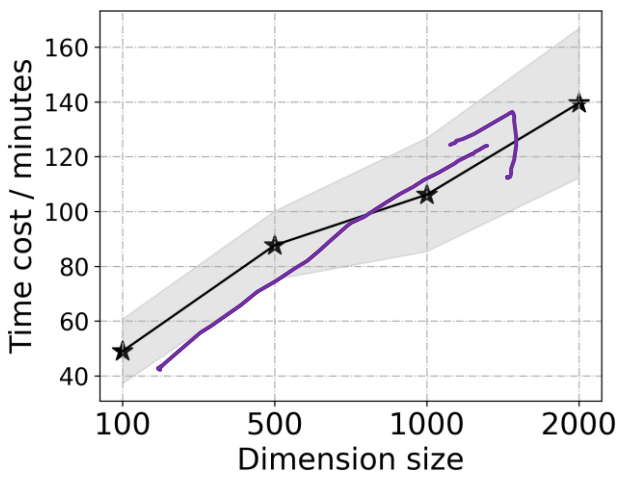
Expensive

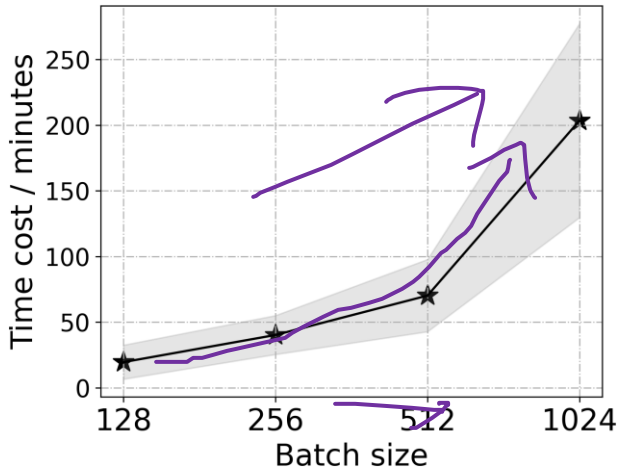
How to reduce the evaluation cost?

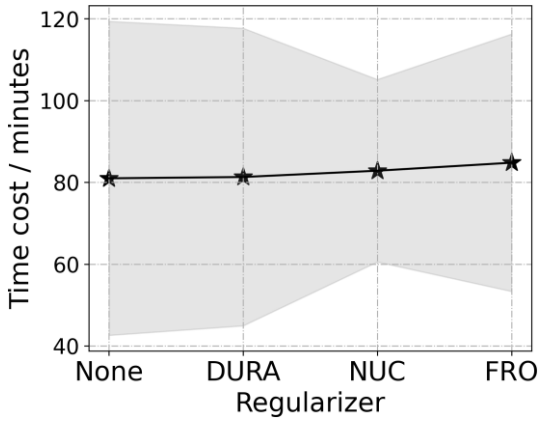
Understand HP in KG Learning

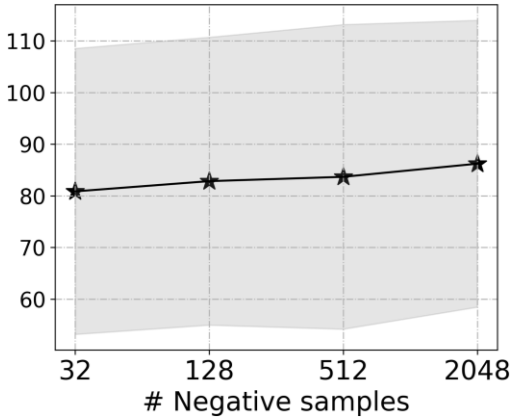
Excavating properties of HPs | Time cost[1]

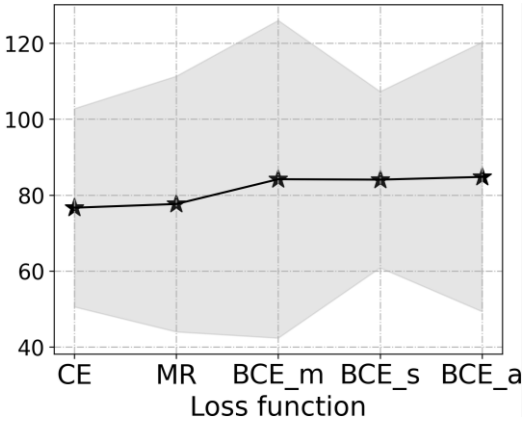
Observation:
batchsize↑ or dimension↑
⇒ time cost↑







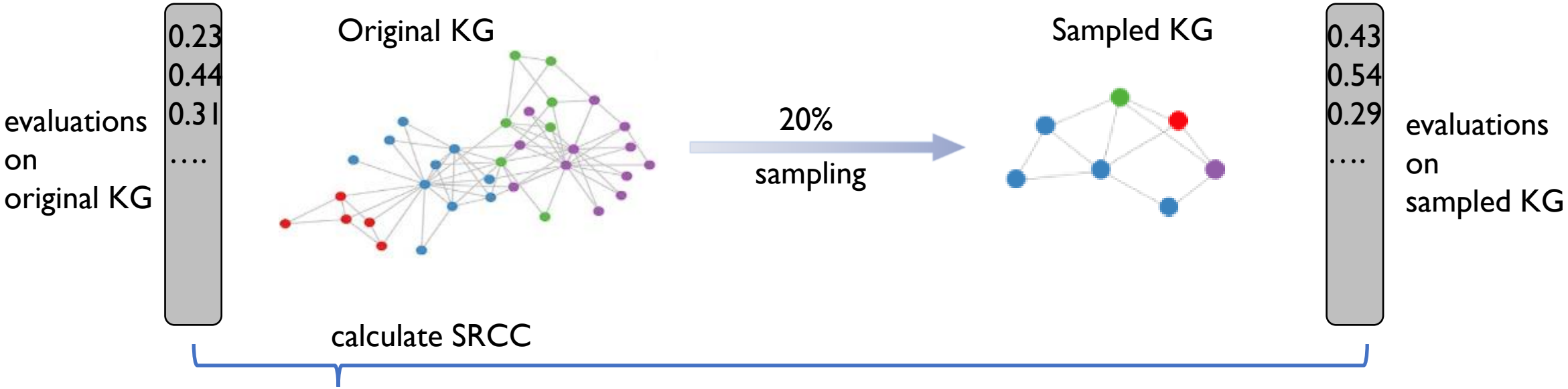




1: The experiments are implemented with PyTorch framework,
on a machine with Intel Xeon 6230R CPUs, 754 GB memory and RTX 3090 GPUs with 24 GB.

Understand HP in KG Learning

Excavating properties of HPs | Transferability of subgraphs

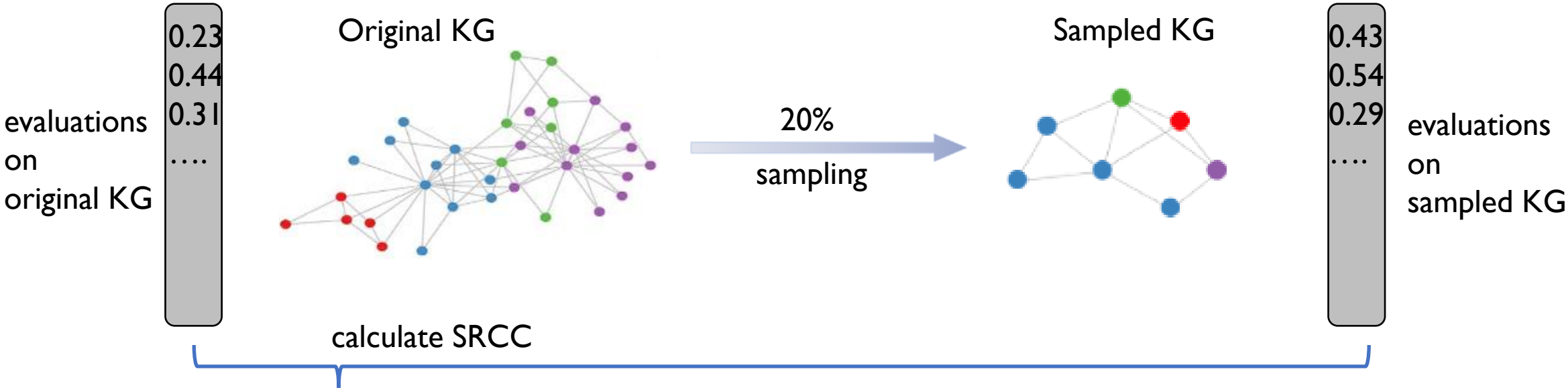


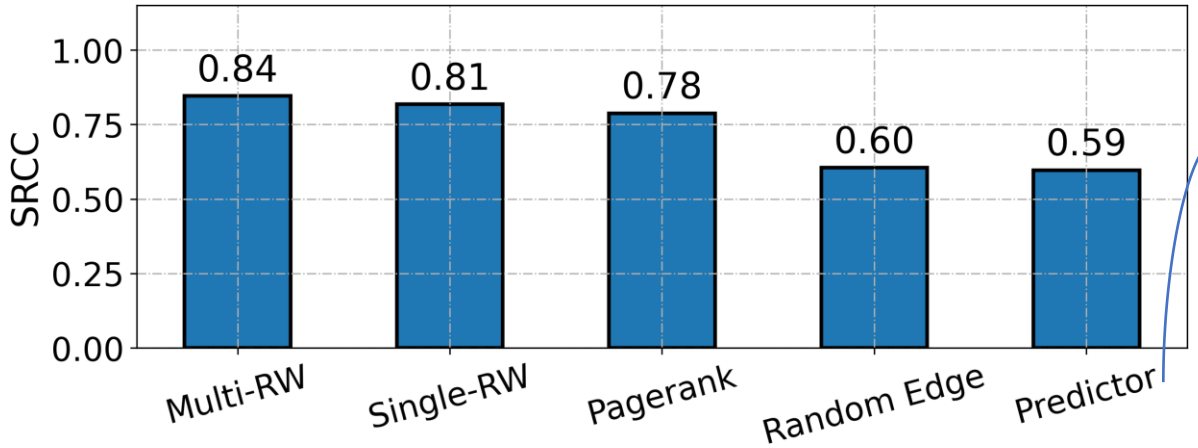
Observation:

- Good correlation between subgraph and original graph
- Multi-start random walk is the best choice

Understand HP in KG Learning

Excavating properties of HPs | Transferability of subgraphs

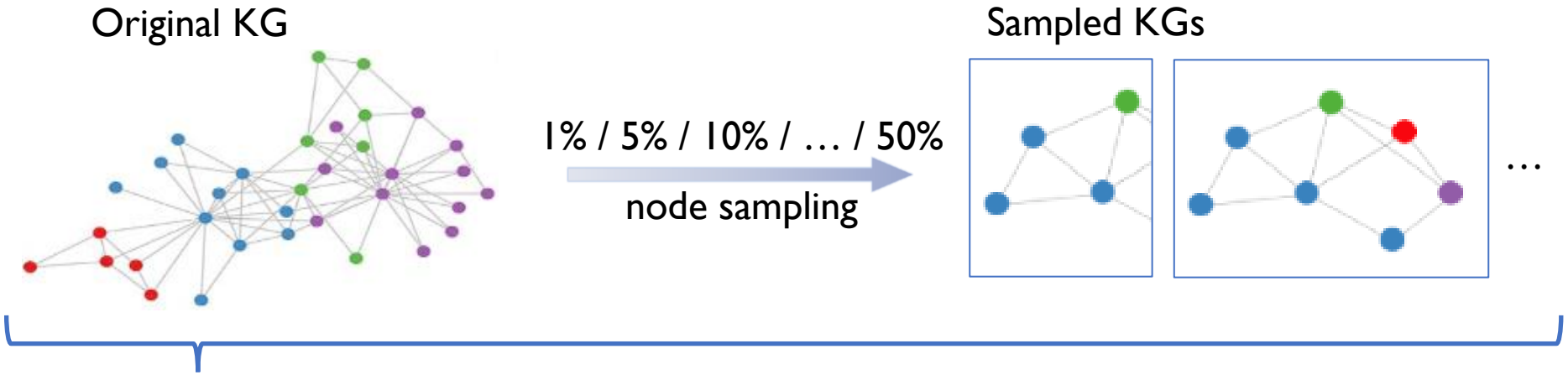


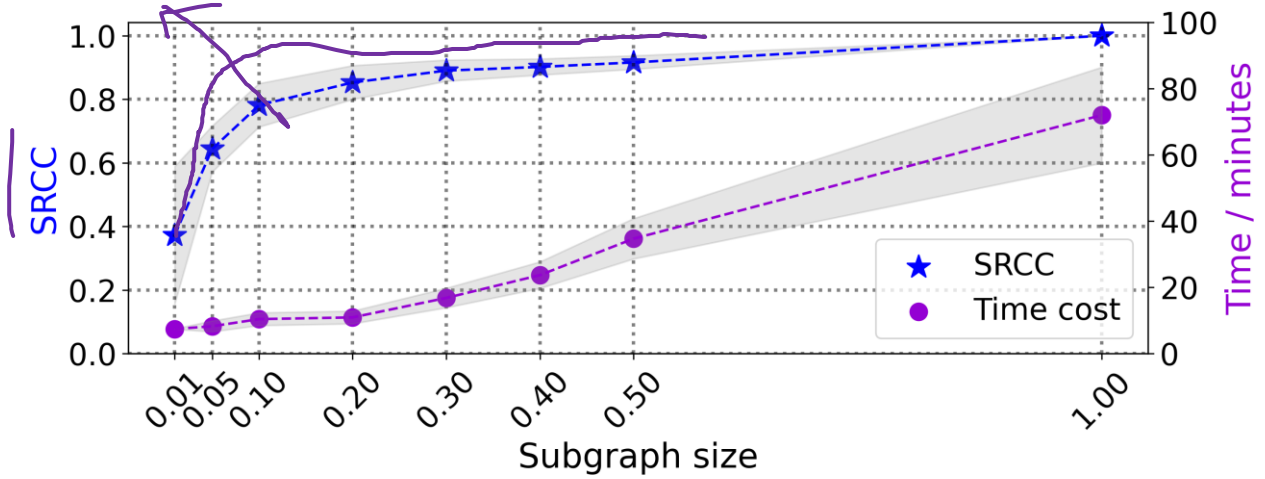


Predictor
- train with sampled KG + original KG
- lower SRCC compared with direct transfer
- similar cases for GP/MLP/RF

Understand HP in KG Learning

Excavating properties of HPs | Transferability of subgraphs





Observation:

- When the subgraph size↑, the consistency↑ and the cost↑
- To balance the consistency and cost, the subgraphs with 20% nodes are the better choices

Understand HP in KG Learning

Summary of the observations

- Ranking distribution/consistency for each HP's values
  - dimension/batchsize
- Full HP range can be reduced and decoupled

- The validation curvature is pretty complex
- RF is better than GP/MLP as the predictor

- Sampling with multi-start random walk can reduce cost while possessing high performance consistency

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

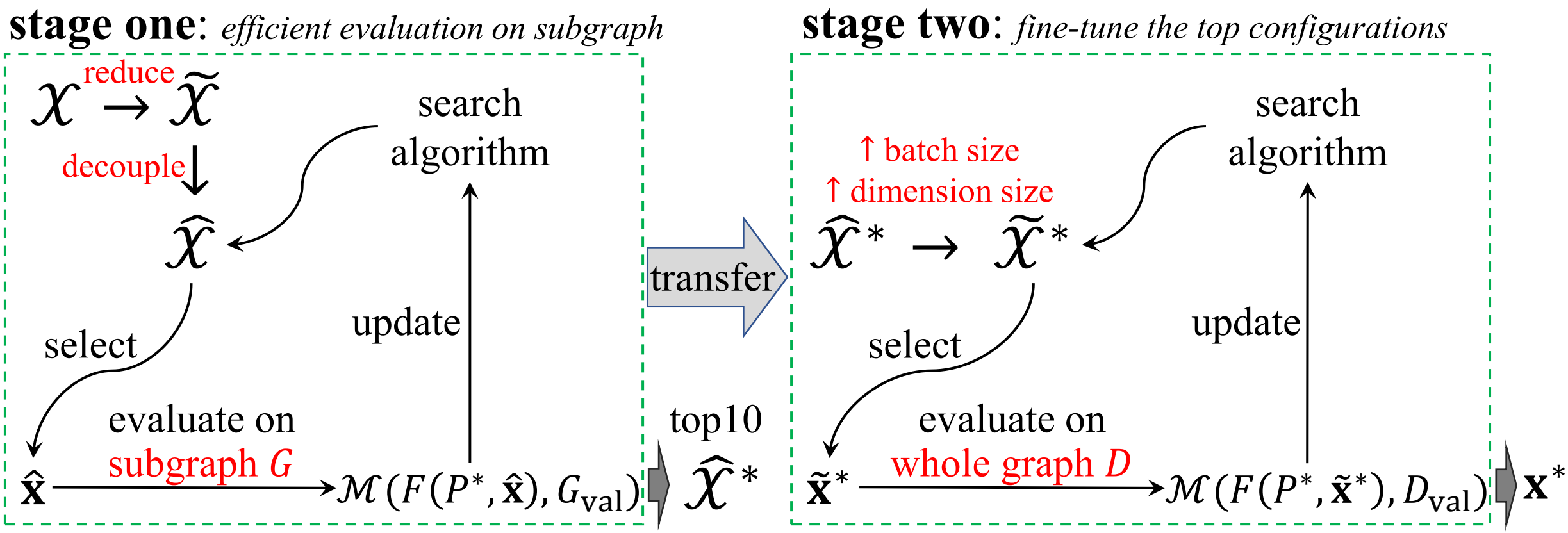
$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathcal{M}\big(F(\boldsymbol{P}^*, \mathbf{x}), D_{val}\big),$$
$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P}} \mathcal{L}\big(F(\boldsymbol{P}, \mathbf{x}), D_{tra}\big).$$

How to design algorithm based on the above observations? 🤔

Outline

- What's Hyper-parameter (HP) tuning
- What's Knowledge Graph (KG) learning
- Understand HP in KG learning
- An efficient two-stage HP search algorithm
- Experiments
- Key takeaway and future directions

General Idea



Efficient Two-stage Search Algorithm

Pre-processing step

name	ranges in the whole space	revised ranges
optimizer	{Adam, Adagrad, SGD}	Adam
learning rate	$[10^{-5}, 10^{0}]$	$[10^{-4}, 10^{-1}]$
reg. weight	$[10^{-12}, 10^{2}]$	$[10^{-8}, 10^{-2}]$
dropout rate	[0, 0.5]	[0, 0.3]
inverse relation	{True, False}	{False}
batch size	{128, 256, 512, 1024}	128
dimension size	{100, 200, 500, 1000, 2000}	100

shrink range:

- Fixed choice
- Limited range

decoupled HPs:

- Monotonously related

Reduced space = shrinkage range HPs + decoupled HPs

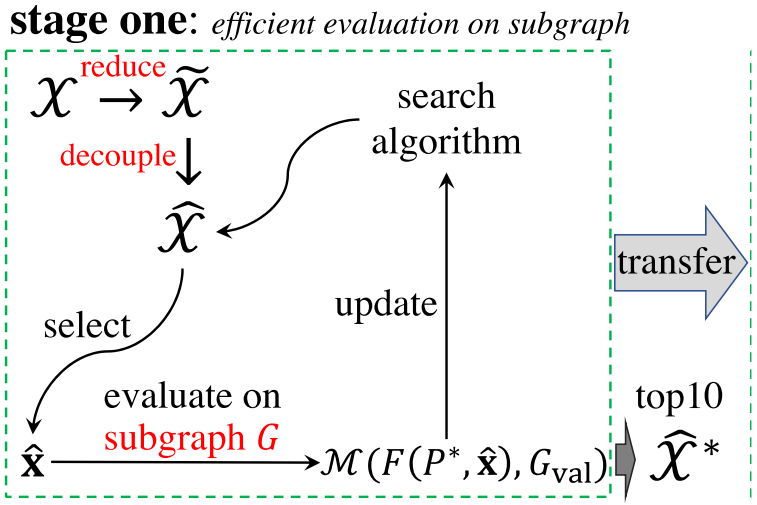
The reduced space is about **1/700** size of the full space

Stage I: Exploration on reduced space

KGTuner: two-stage search algorithm

Stage I: exploration on reduced space

- quickly search HP on sampled KG
- with predictor RF and acquisition BORE



Algorithm 1 KGTuner: two-stage search algorithm

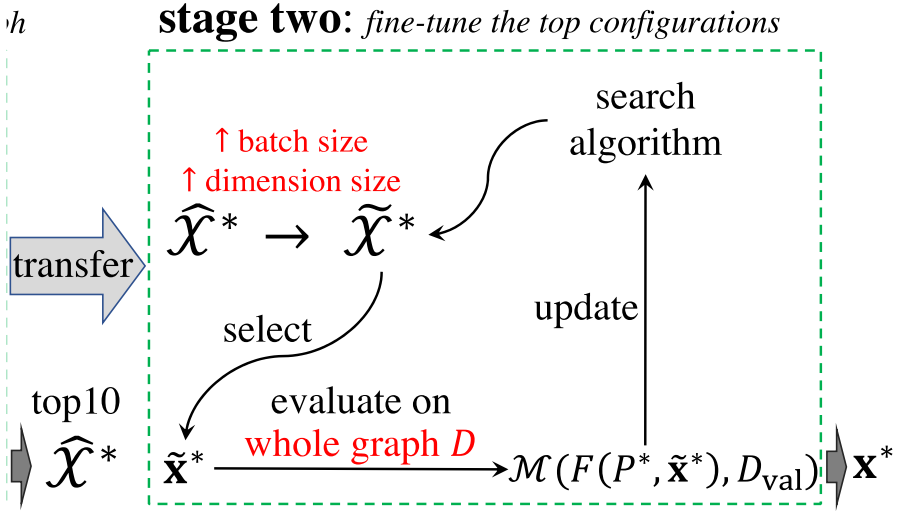
Require: KG embedding model F, dataset D, and budget B;
shrink the search space $\mathcal{X}$ to $\mathcal{X}_S$ and decouple $\mathcal{X}_S$ to $\mathcal{X}_{\text{SID}}$;
 # state one: *efficient evaluation on subgraph*
sample a subgraph (with 20% entities) G from D_{tra} by multi-start random walk;
repeat
  sample a configuration $\hat{\mathbf{x}}$ from $\mathcal{X}_{S|D}$ by RF+BORE;
  evaluate $\hat{\mathbf{x}}$ on the subgraph G to get the performance;
  update the RF with record $\left(\hat{\mathbf{x}}, \mathcal{M}(F(P^*, \hat{\mathbf{x}}), G_{\text{val}})\right)$;
until $B/2$ budget exhausted;
save the *top10* configurations in $\mathcal{X}^*_{\text{SID}}$;
 # state two: *fine-tune the top configurations*
increase the batch/dimension size in $\mathcal{X}^*_{\text{SID}}$ to get $\tilde{\mathcal{X}}^*$;
set $y^* = 0$ and re-initialize the RF surrogate;
repeat
  select a configuration $\tilde{\mathbf{x}}^*$ from $\tilde{\mathcal{X}}^*$ by RF+BORE;
  evaluate on full graph G to get the performance;
  update the RF with record $\left(\tilde{\mathbf{x}}^*, \mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{\text{val}})\right)$;
  if $\mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{\text{val}}) > y^*$ **then**
  $y^* \leftarrow \mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{\text{val}})$ and $\mathbf{x}^* \leftarrow \tilde{\mathbf{x}}^*$; **end if**
until the remaining $B/2$ budget exhausted;
return $\mathbf{x}^*$.

Stage2: Exploitation with fine-tuning

KGTuner: two-stage search algorithm

Stage2: exploitation with fine-tuning

- transfer top10 configurations from stage 1
- finetune configuration on original KG, with higher dimension and batchsize



Algorithm 1 KGTuner: two-stage search algorithm

Require: KG embedding model F, dataset D, and budget B;
1: shrink the search space $\mathcal{X}$ to $\mathcal{X}_S$ and decouple $\mathcal{X}_S$ to $\mathcal{X}_{S|D}$;
 # state one: *efficient evaluation on subgraph*
2: sample a subgraph (with 20% entities) G from D_{tra} by multi-start random walk;
3: **repeat**
4: sample a configuration $\hat{\mathbf{x}}$ from $\mathcal{X}_{S|D}$ by RF+BORE;
5: evaluate $\hat{\mathbf{x}}$ on the subgraph G to get the performance;
6: update the RF with record $(\hat{\mathbf{x}}, \mathcal{M}(F(P^*, \hat{\mathbf{x}}), G_{val}))$;
7: **until** $B/2$ budget exhausted;
8: save the *top10* configurations in $\mathcal{X}^*_{S|D}$;
 # state two: *fine-tune the top configurations*
9: increase the batch/dimension size in $\mathcal{X}^*_{S|D}$ to get $\tilde{\mathcal{X}}^*$;
10: set $y^* = 0$ and re-initialize the RF surrogate;
11: **repeat**
12: select a configuration $\tilde{\mathbf{x}}^*$ from $\tilde{\mathcal{X}}^*$ by RF+BORE;
13: evaluate on full graph G to get the performance;
14: update the RF with record $(\tilde{\mathbf{x}}^*, \mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{val}))$;
15: **if** $\mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{val}) > y^*$ **then** $y^* \leftarrow \mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{val})$ and $\mathbf{x}^* \leftarrow \tilde{\mathbf{x}}^*$; **end if**
16: **until** the remaining $B/2$ budget exhausted;
17: **return** $\mathbf{x}^*$.

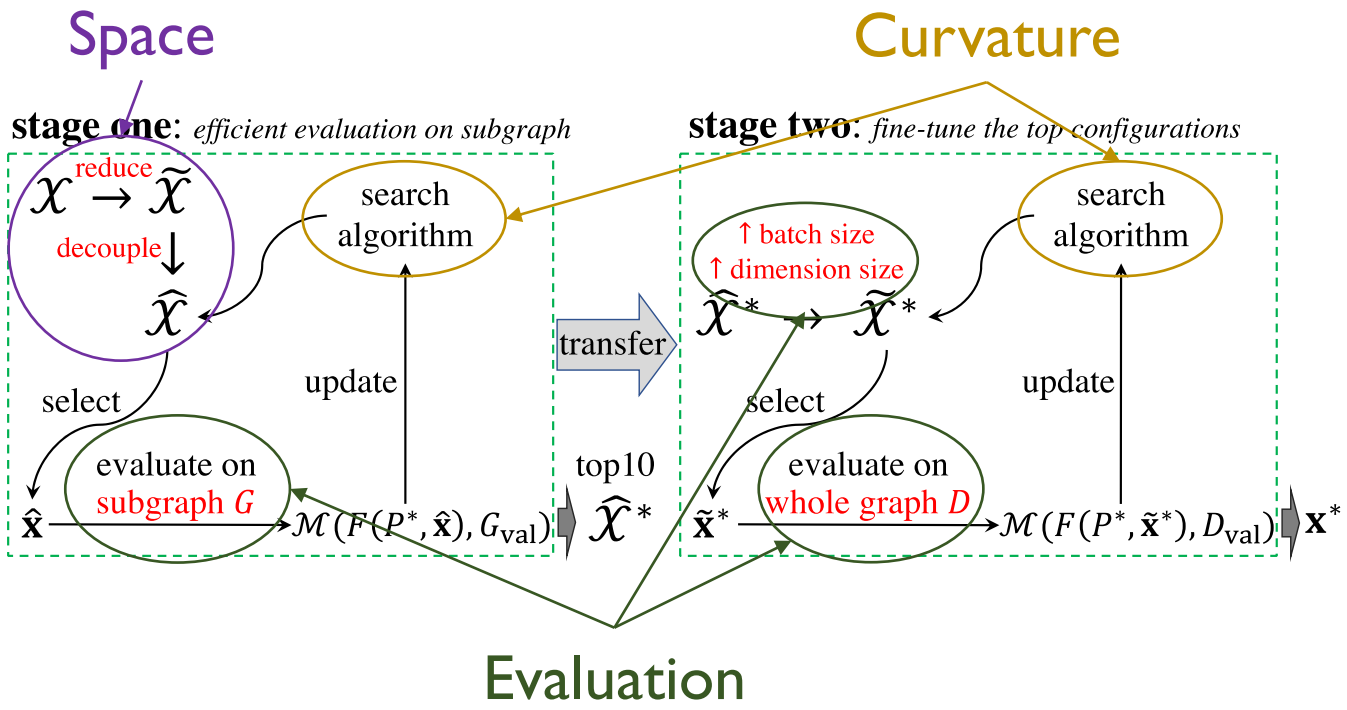
Summary of the Search Algorithm

Preprocessing: from the original to reduce space

Stage1: exploration on reduced space

Stage2: exploitation with fine-tuning

Where observations are used -

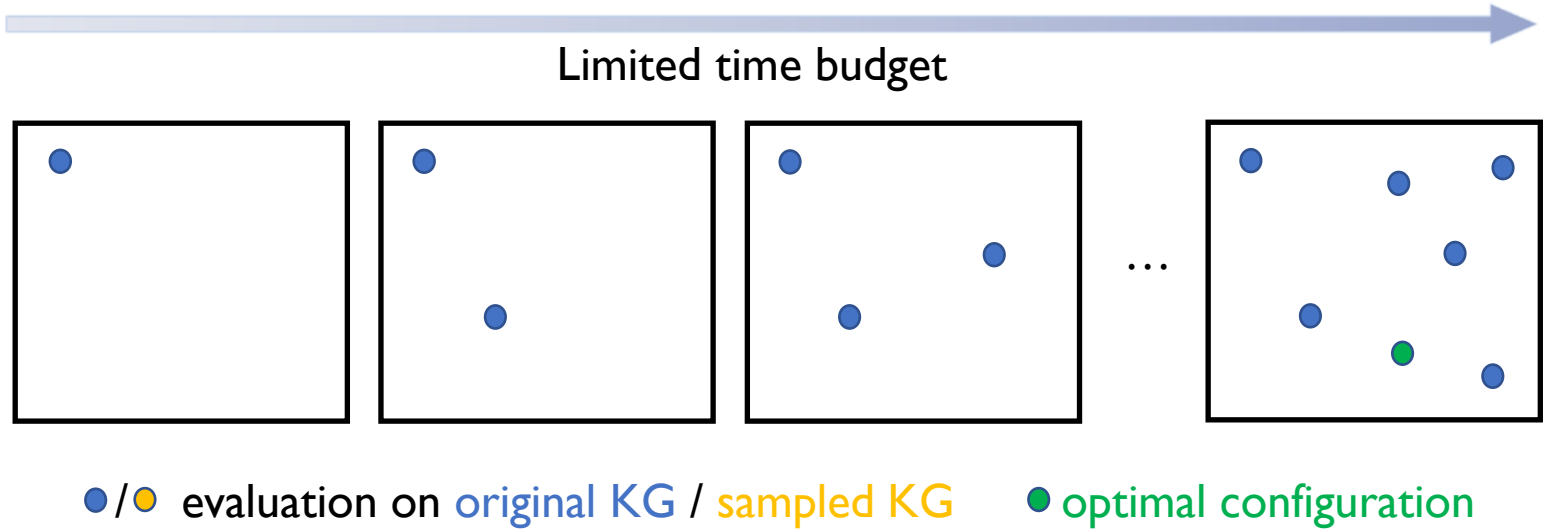


Algorithm 1 KGTuner: two-stage search algorithm

Require: KG embedding model F, dataset D, and budget B;
shrink the search space $\mathcal{X}$ to $\mathcal{X}_S$ and decouple $\mathcal{X}_S$ to $\mathcal{X}_{S|D}$;
 # state one: *efficient evaluation on subgraph*
sample a subgraph (with 20% entities) G from D_{tra} by multi-start random walk;
repeat
  sample a configuration $\hat{\mathbf{x}}$ from $\mathcal{X}_{S|D}$ by RF+BORE;
  evaluate $\hat{\mathbf{x}}$ on the subgraph G to get the performance;
  update the RF with record $\left(\hat{\mathbf{x}}, \mathcal{M}(F(P^*, \hat{\mathbf{x}}), G_{val})\right)$;
until $B/2$ budget exhausted;
save the *top10* configurations in $\mathcal{X}^*_{S|D}$;
 # state two: *fine-tune the top configurations*
increase the batch/dimension size in $\mathcal{X}^*_{S|D}$ to get $\tilde{\mathcal{X}}^*$;
set $y^* = 0$ and re-initialize the RF surrogate;
repeat
  select a configuration $\tilde{\mathbf{x}}^*$ from $\tilde{\mathcal{X}}^*$ by RF+BORE;
  evaluate on full graph G to get the performance;
  update the RF with record $\left(\tilde{\mathbf{x}}^*, \mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{val})\right)$;
  if $\mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{val}) > y^*$ **then**
  $y^* \leftarrow \mathcal{M}(F(P^*, \tilde{\mathbf{x}}^*), D_{val})$ and $\mathbf{x}^* \leftarrow \tilde{\mathbf{x}}^*$; **end if**
until the remaining $B/2$ budget exhausted;
return $\mathbf{x}^*$.

Efficient two-stage HP search algorithm

Searching process diagram



(a) Conventional methods.

(b) TOSS.

Use AutoML? HP tuning example

1. Define an AutoML problem

- Derive a search space from insights in specific domains
- Search objective is usually validation performance
- Search constraint is usually resource budgets
- Training objective usually comes from classical learning models

Search Space → $\min_{\lambda \in \mathcal{S}} M(F(w^*; \lambda), D_{\text{val}})$ ← Search Objective

$\min_{w} L(F(w; \lambda), D_{\text{tra}})$ ← Training Objective

s. t. $G(\lambda) \leq C$ ← Search Constraints

2. Design or select proper search algorithm

- Reduce model training cost (time to get w^*)

Outline

- What's Hyper-parameter (HP) tuning
- What's Knowledge Graph (KG) learning
- Understand HP in KG learning
- An efficient two-stage HP search algorithm
- Experiments
- Key takeaway and future directions

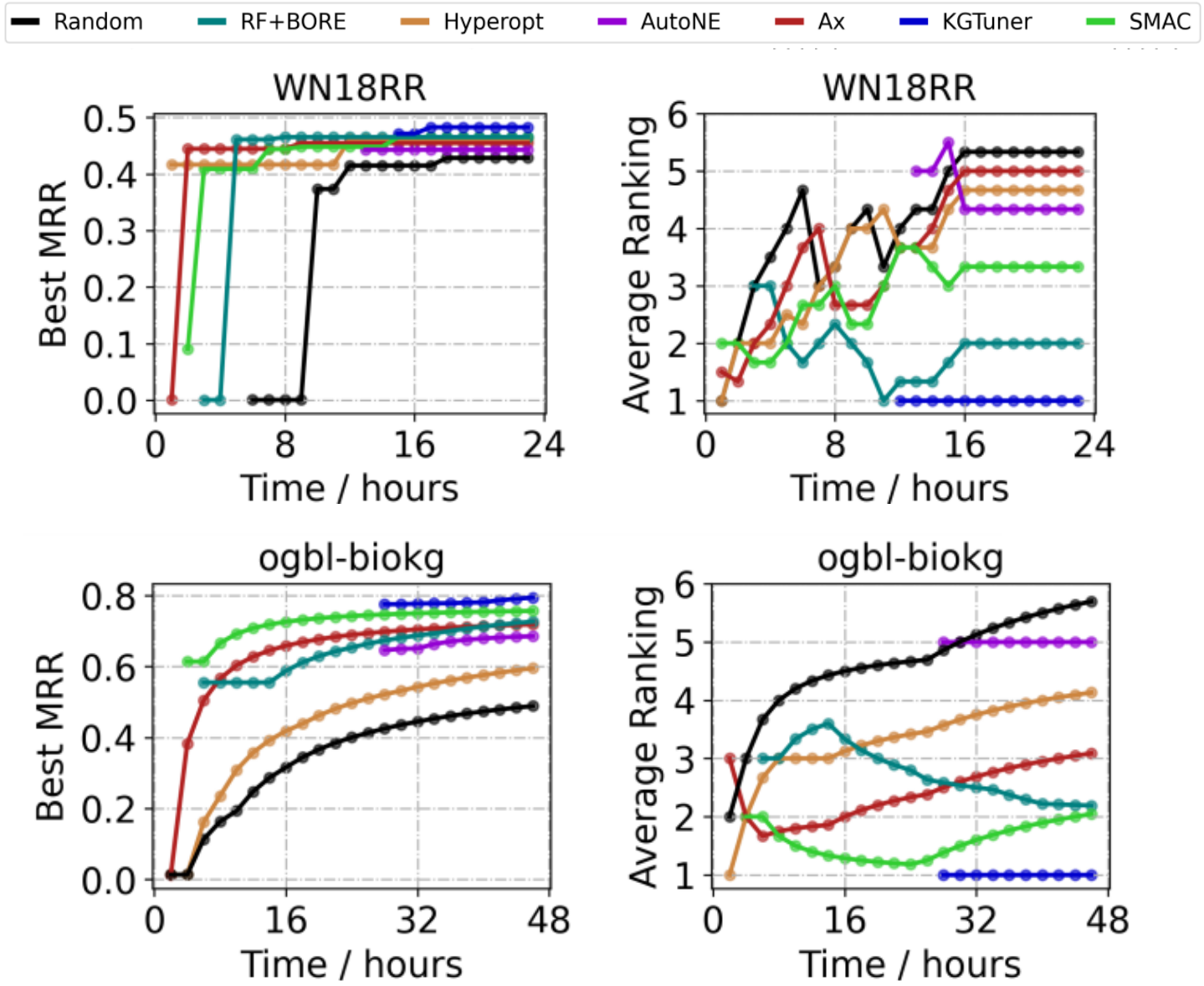
Experiment

Search algorithm comparison

Observation

- Random search is the worst due to the full randomness.
- SMAC and RF+BORE achieve better performance than Hyperopt and Ax since RF can fit the space better than TPE and GP.
- Due to the weak approximation and transferability, AutoNE also performs bad.
- TOSS is much better than all the baselines

	search space reduce	search space decouple	surrogate model	fast evaluation
Random	×	×	×	×
Hyperopt	×	×	TPE	×
Ax	×	×	GP	×
SMAC	×	×	RF	×
RF+BORE	×	×	RF	×
AutoNE	×	×	GP	√
TOSS	√	√	RF	√



Experiment

Searched configuration performance

Table 3: MRR of models with HPs tuned in different methods. The bold numbers mean the best performance of the same model.

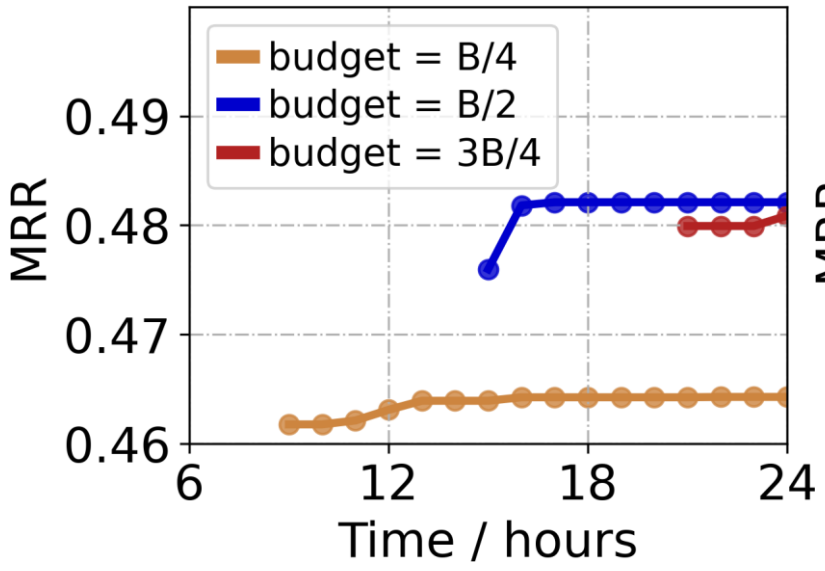
source	models	WN18RR	FB15k-237
original	TransE	0.226	0.296
	ComplEx	0.440	0.247
	ConvE	0.430	0.325
LibKGE	TransE	0.228	0.313
	ComplEx	0.475	0.348
	ConvE	**0.442**	**0.339**
KGTuner	TransE	**0.233**	**0.327**
	ComplEx	**0.484**	**0.352**
	ConvE	0.437	0.335

Table 4: Performance in MRR in OGB link prediction board `https://ogb.stanford.edu/docs/leader_linkprop/` *and those reproduced by KGTuner on ogbl-biokg and ogbl-wikikg2. Relative improvements are in parenthesize.*

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%

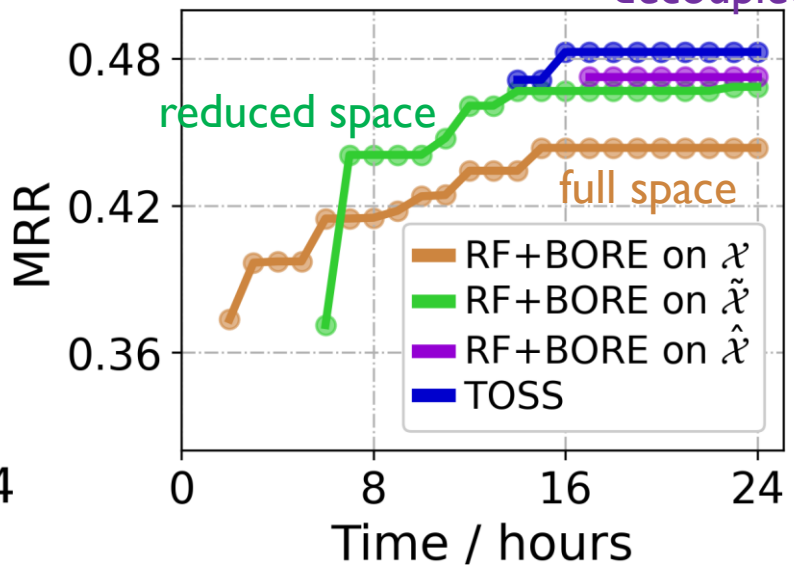
Experiment | Ablation study

First-stage budget



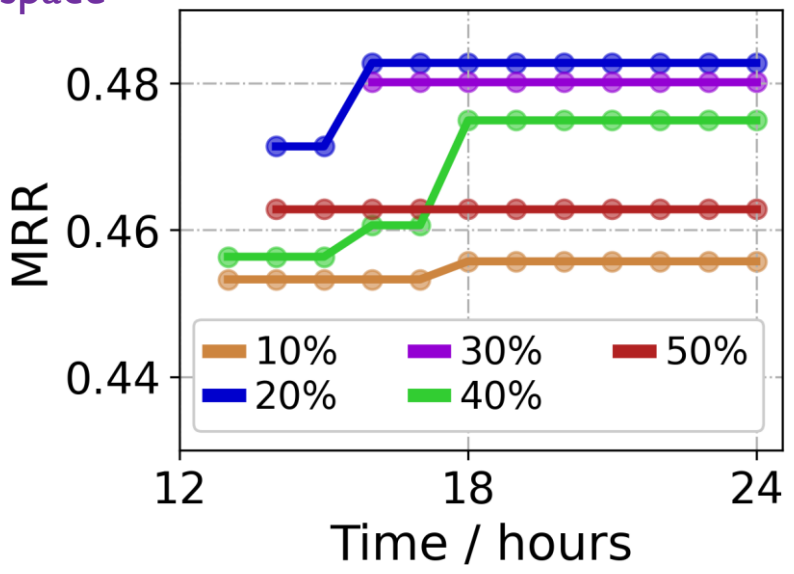
budget = B/2

Search space



Subgraph + Decouple

Subgraph size



Subgraph size = 20%/30%

OGB Leaderboard & Toolbox

Our result on ogbl-biokg:

	Test MRR	Validation MRR	Institution
AutoSF+RP	**0.8543**	**0.8550**	Tsinghua
ComplEx-RP	0.8492	0.8497	UCL NLP & FAIR
TripleRE	0.8348	0.8360	360AI
AutoSF	0.8309	0.8317	4Paradigm

Our result can be reproduced by our toolbox KGBench

- a toolbox for automatic search of hyperparameters and model structure
- will be open sourced this summer

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

Outline

- What's Hyper-parameter (HP) tuning
- What's Knowledge Graph (KG) learning
- Understand HP in KG learning
- An efficient two-stage HP search algorithm
- Experiments
- Key takeaway and future directions

Key takeaways

Recall the difficulties	KGTuner
Lacking understanding of KGE components	A comprehensive understanding of HPs
Low efficiency in searching for hyperparameter	An efficient two-stage HP search algorithm

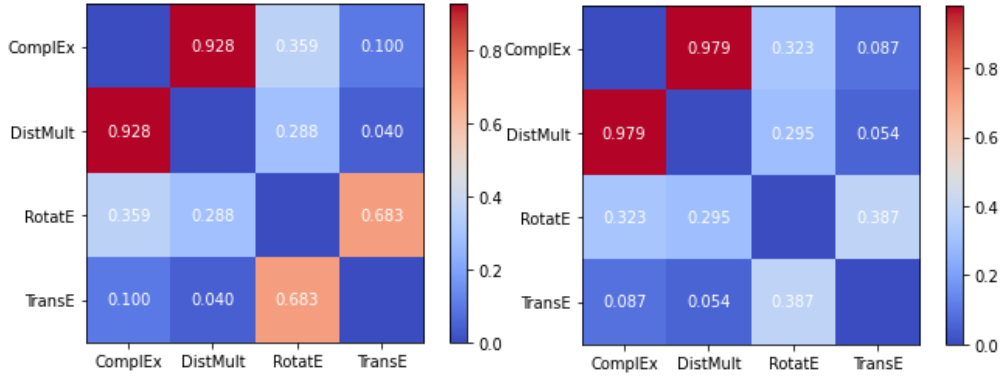
Limitation and future directions

Limitation

- Limited to pure embedding models
- Not considering HPs inside the SF model
- Lacking of theoretical analysis and guarantees

Potential directions

- apply with GNN to solve the scaling problem
- combine HPO with NAS
- transferability across datasets/models/tasks



WN18RR FB15k-237

Q&A

Thanks for your attention!