Learn to Optimize: A Tutorial

Ke Tang, Shengcai Liu

Department of Computer Science and Engineering

Southern University of Science and Technology (SUSTech)

{tangk3,liusc3}@sustech.edu.cn

https://faculty.sustech.edu.cn/tangk3, http://nical.ai.shengcai

Outline

- **Introduction to Learning to Optimize (L2O)**
- Research Directions in L2O
  - Automatic Algorithm Selection
  - Automatic Algorithm Configuration
  - Neural Combinatorial Optimization
- Summary

Optimization

- It concerns finding the best solutions that *maximize* (or *minimize*) some criterion

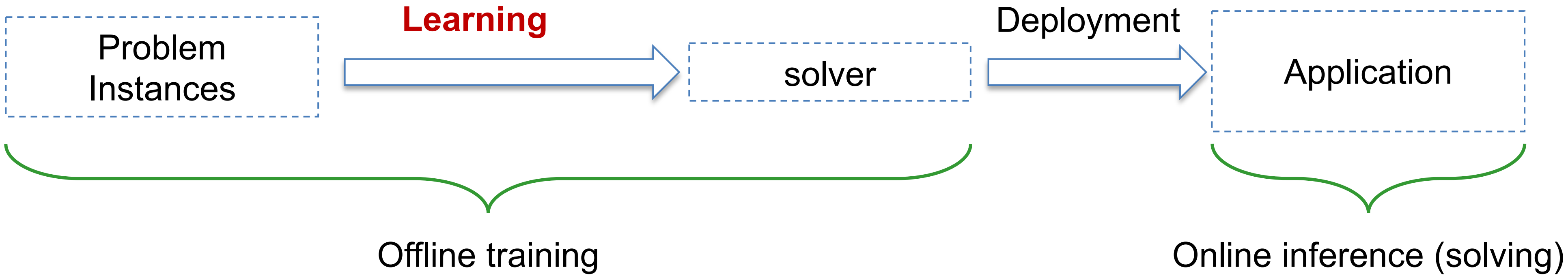
$$\begin{aligned} &\text{maximize} \quad f(x) \\ &\text{subject to: } g_i(x) \le 0, \quad i = 1 \ldots m \\ &\qquad\qquad\quad\; h_j(x) = 0, \quad j = 1 \ldots p \end{aligned}$$

- For optimization, the performances in terms of two aspects are often of interest
  - Effectiveness, e.g., solution quality
  - Efficiency, e.g., runtime, number of fitness evaluations

What is Learning to Optimize (L2O)

- Learning concerns improving performance with the *accumulation of experience*

- Generally, L2O leverages learning to *train* solvers for the problems of interest
  - The learning (training) phase is conducted offline
  - The learned solver will be deployed in production/application

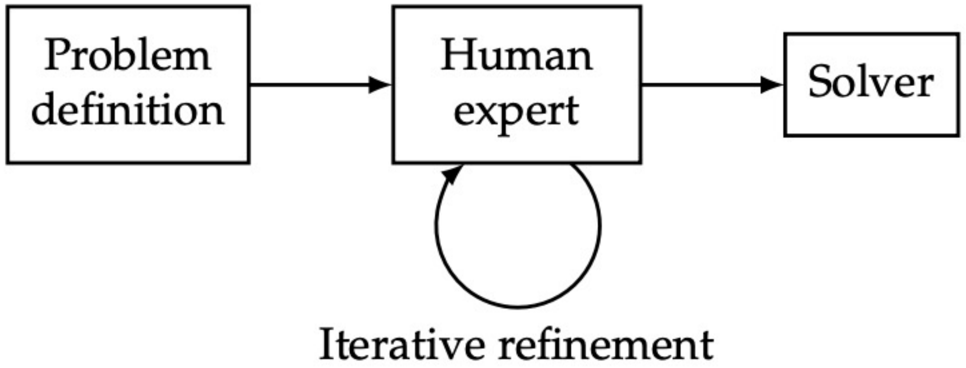


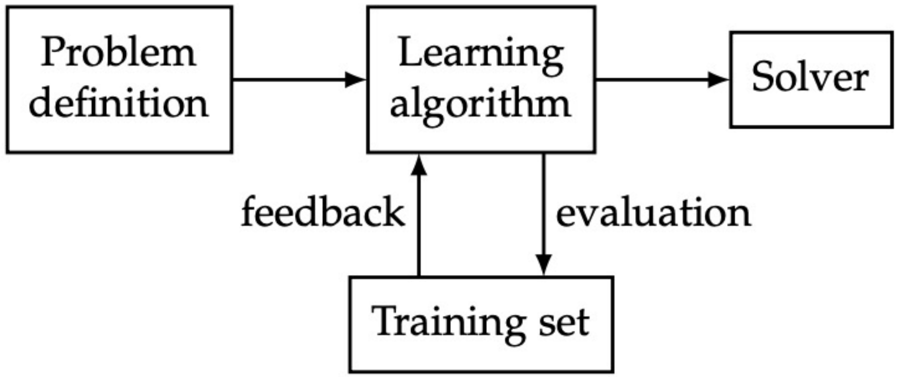
L2O vs. Classic Optimization

- Traditionally,
  - Experts hand-build algorithms/solvers based on theory/experience
  - Practitioners pick a solver to use

- L2O
  - Experts propose L2O frameworks and training procedures
  - Practitioners pick a L2O framework, prepare training data, and apply the training procedure to obtain a solver to use

L2O vs. Classic Optimization

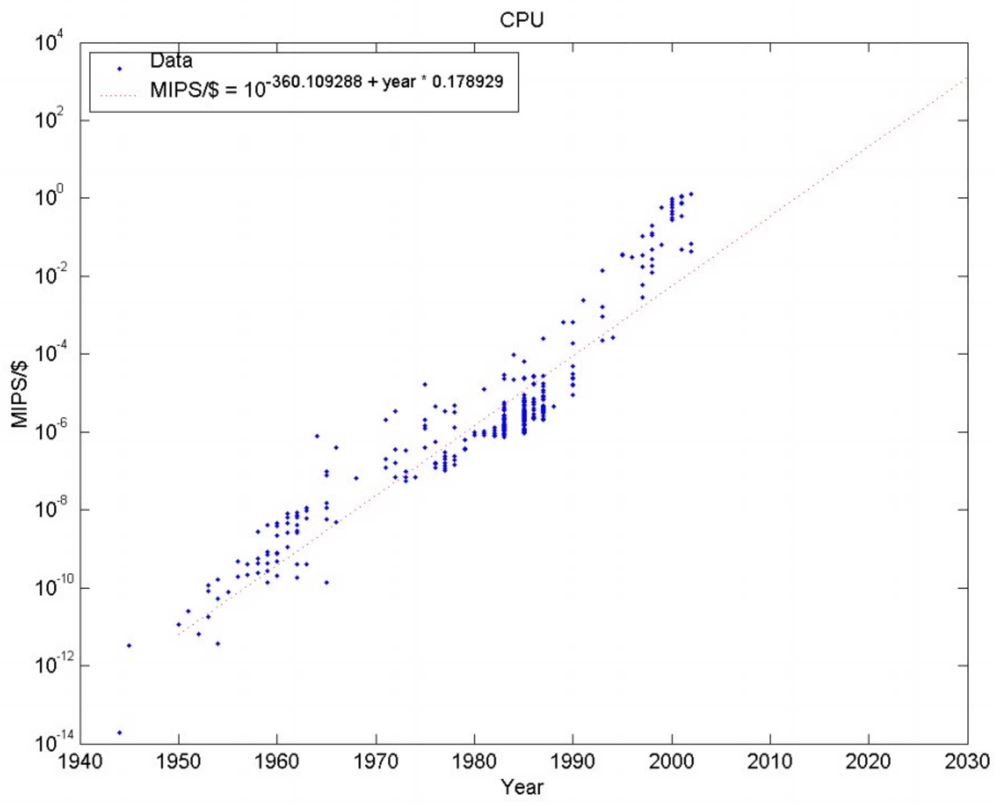


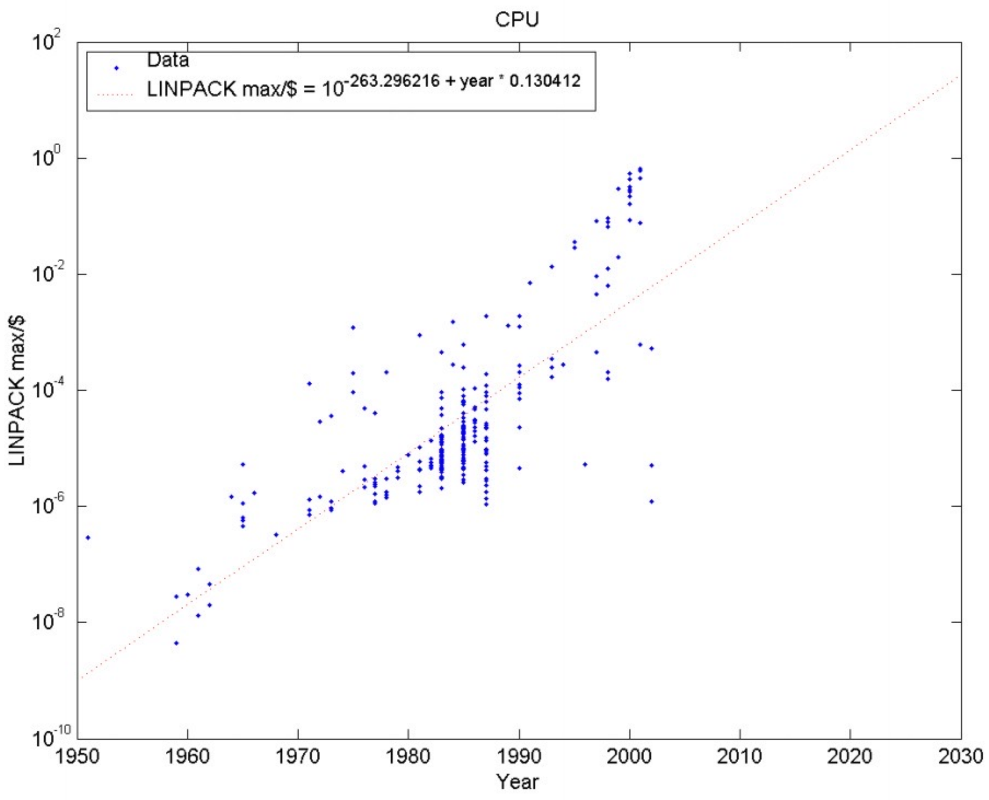


- Classic
  - Human centered
  - Heavily depends on domain expertise
  - Expensive in *human time*
- L2O
  - Learning centered
  - Requires much less domain knowledge
  - Expensive in *computation time*

L2O vs. Classic Optimization

Computing is getting cheaper and cheaper

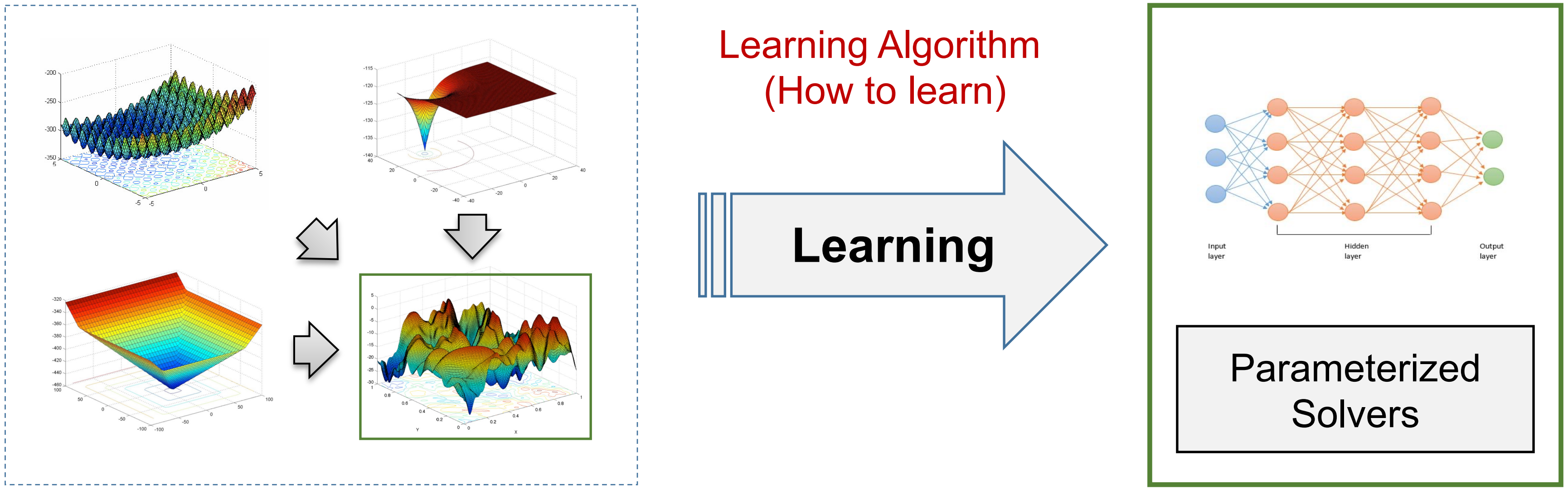




One dollar’s worth of computer power, measured in MIPS (left) and FLOPS (right)

Source: https://aiimpacts.org/trends-in-the-cost-of-computing

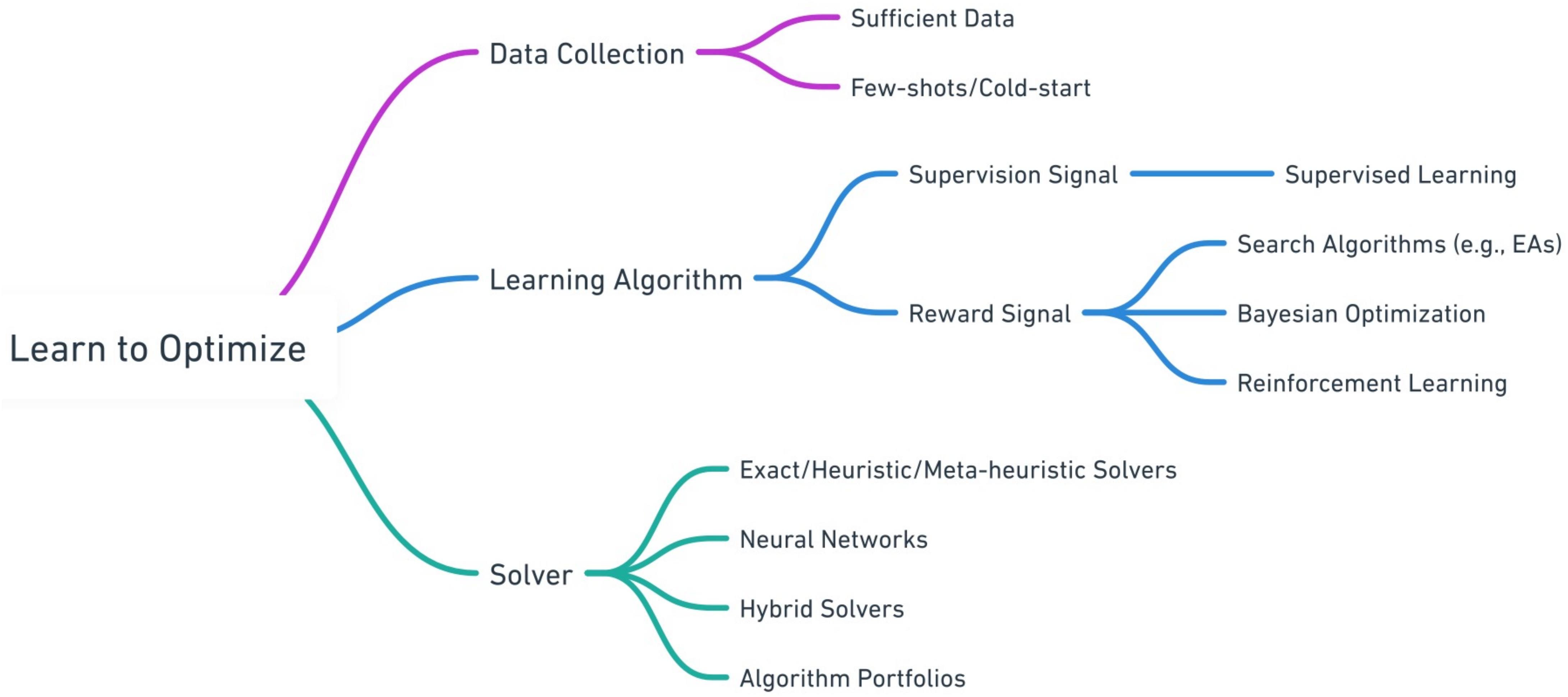
Key Research Questions in L2O



Training Data Collection
（Learn from what）

Learned Solvers
（What to learn）

A Taxonomy of L2O



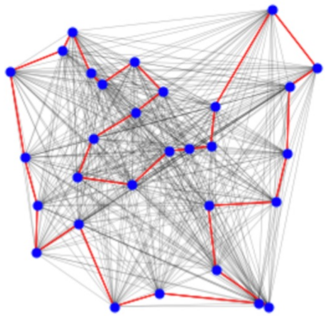
Research Directions in L2O

- This tutorial covers several widely studied directions in L2O
  - Automatic Algorithm Selection (AAS)
  - Automatic Algorithm Configuration (AAC)
  - Neural Combinatorial Optimization (NCO)

- Appropriate for handling different practical situations
  - AAS — there exist several powerful solvers for the problem, how to get the best of them?
  - AAC — the solver's performance heavily depends on its parameter configuration, how to identify the best configuration, or even to build a more powerful solver based on it?
  - NCO — how to build an effective solver in a unified framework that is applicable to a wide range of problems?

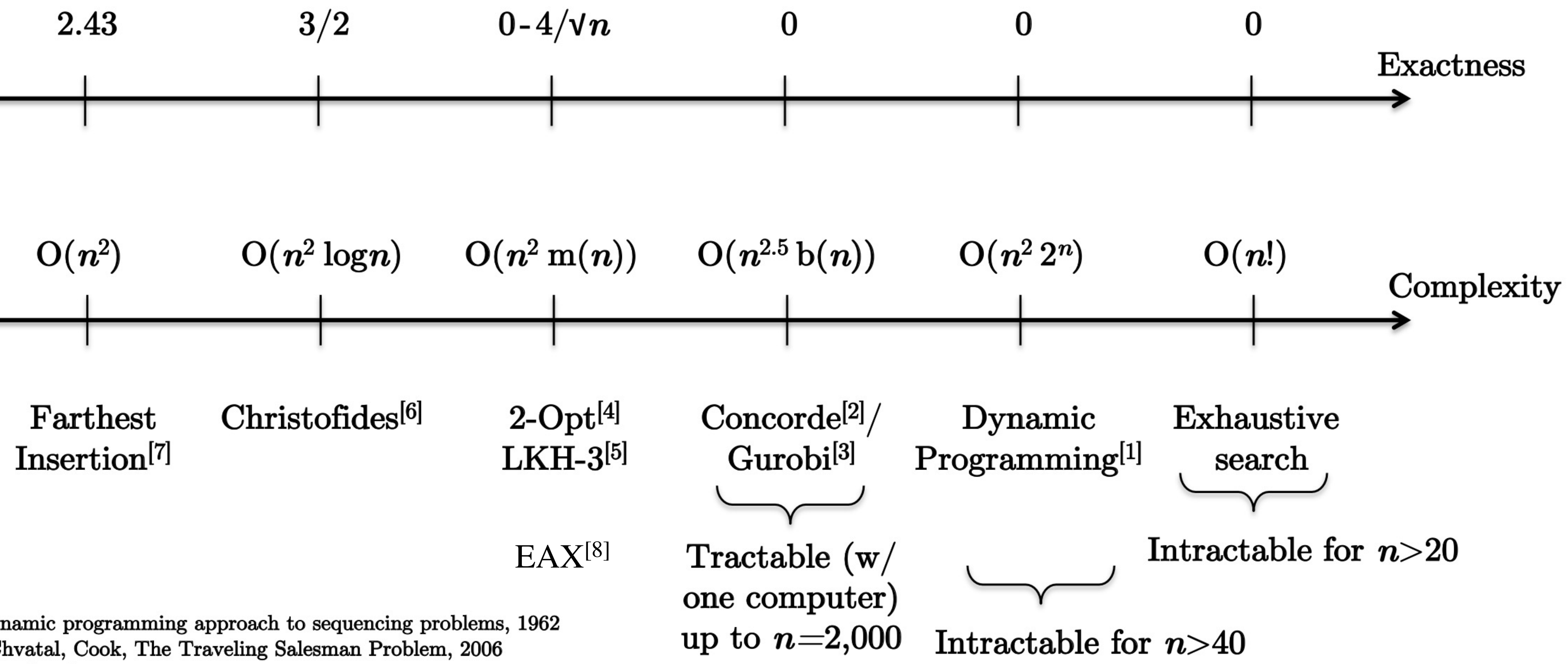
Quick Recap of TSP

- We use the well-known traveling salesman problem (TSP) as an example
  - One of the most representative optimization problem
  - Comprehensive empirical results available
  - Ideas presented in the tutorial also apply to other optimization problems



- TSP concerns finding the shortest Hamilton cycle on a complete graph
  - NP-hard, exhaustive search has a complexity of $O(n!)$

Traditional Solvers for TSP



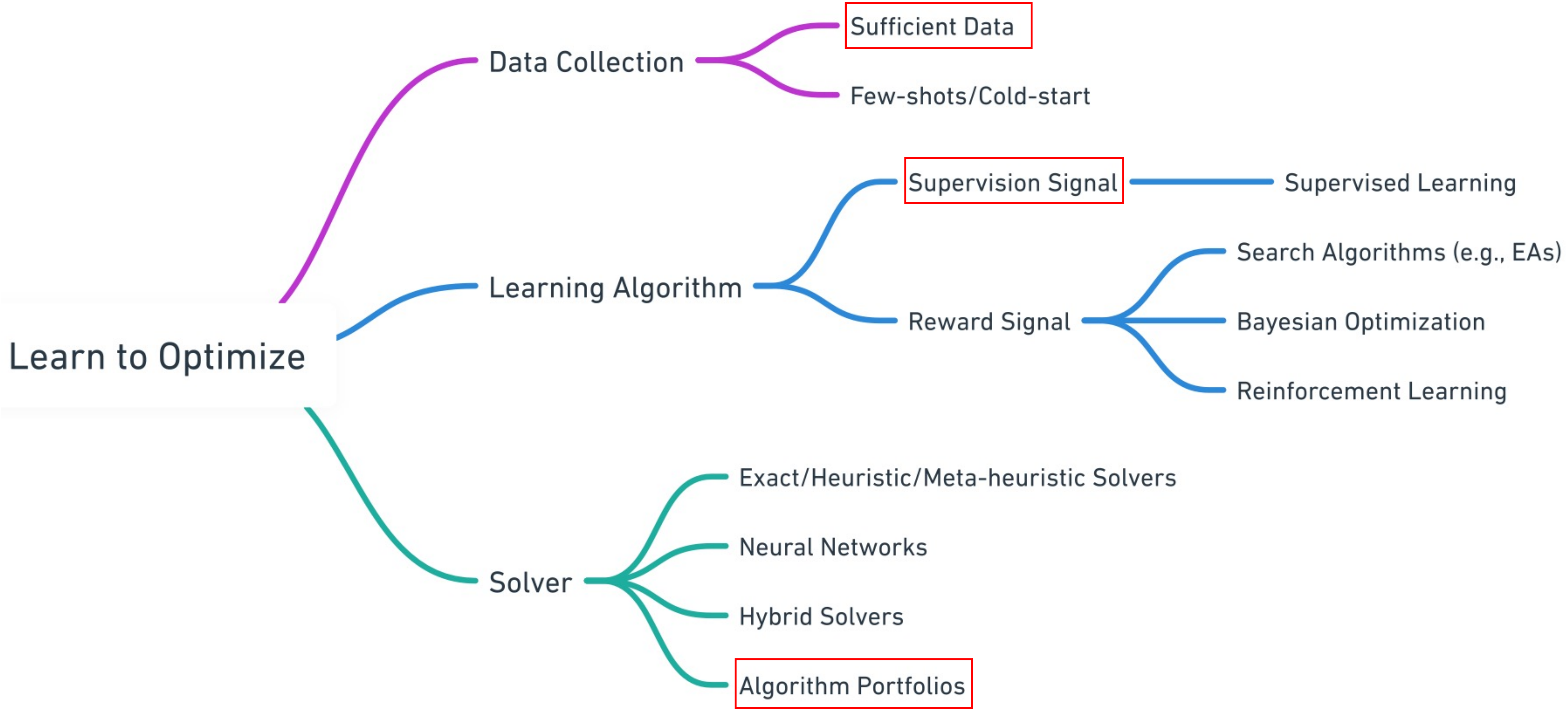
[1] Held, Richard, A dynamic programming approach to sequencing problems, 1962
[2] Applegate, Bixby, Chvatal, Cook, The Traveling Salesman Problem, 2006
[3] Gu, Rothberg, Bixby, Gurobi, 2008
[4] Johnson, McGeoch, The traveling salesman problem: A case study in local optimization, 1995
[5] Helsgaun, An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems, 2017
[6] Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, 1976
[7] Johnson, Local Optimization and the Traveling Salesman Problem, 1990
[8] Nagata, Yuichi, and Shigenobu Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. 2013

Source: https://www.ipam.ucla.edu/programs/workshops/deep-learning-and-combinatorial-optimization/

Outline

- Introduction to Learning to Optimize (L2O)
- Research Directions in L2O
  - **Automatic Algorithm Selection**
  - Automatic Algorithm Configuration
  - Neural Combinatorial Optimization
- Summary

Automatic Algorithm Selection



There exist several powerful solvers for the problem, how to get the best of them?

Automatic Algorithm Selection (AAS)

- AAS seeks to train an *algorithm selector* that chooses the best algorithm for a given problem instance

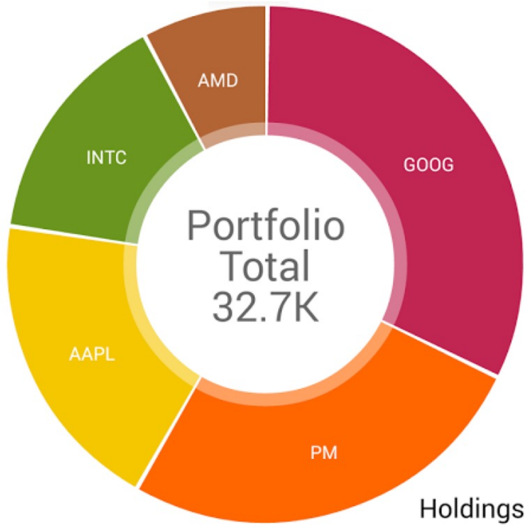
- It generally requires
  - An algorithm portfolio
  - A set of training instances
  - A set of instance features (unnecessary for deep learning-based approach)
  - Instance $\times$ Algorithm performance data

Algorithm Portfolios

- Comes from economics [Huberman et al., 1997]

- A portfolio of financial assets (e.g., stocks)
  - maximize profit and minimize risk

- A portfolio of algorithms
  - no “universal best” algorithm
  - the best algorithm is priori unknown
  - maximize overall performance



Holdings

#	Ticker	Value	Gain	% of Value
1	GOOG	10.5K	+4.8%	32.1%
2	PM	8526.00	+10.7%	26.1%
3	AAPL	6237.50	+1.8%	19.1%
4	INTC	4870.50	+18.1%	14.9%
5	AMD	2575.00	-19.5%	7.9%

Instance × Algorithm Performance Data

Instance	AS	MMAS	ACS
gr24.tsp	*1275.64*	1278.04	1277.52
berlin52.tsp	7746.16	7689.92	7729.16
st70.tsp	714.96	*703.12*	726.68
rd100.tsp	8717.36	8694.24	*8659.53*
ch150.tsp	7219.84	*6867.68*	6908.84
kroA200.tsp	40269.76	32858.68	*32643.76*
tsp225.tsp	4124.64	4095.8	*4039.4*
a280.tsp	3493.16	3020.28	*2973.72*
lin318.tsp	53175.84	47739.36	*47494.96*
pcb442.tsp	60781.08	*59118.32*	59264.12
rat575.tsp	9479.68	7795.44	*7717.48*
rat783.tsp	12684.6	10199.16	*10038.87*

A. Puris, R. Bello, F. Herrera, "Analysis of the Efficacy of a Two-Stage Methodology for Ant Colony Optimization: Case of Study with TSP and QAP," *Expert Systems with Applications 37(7)*

Problem Features

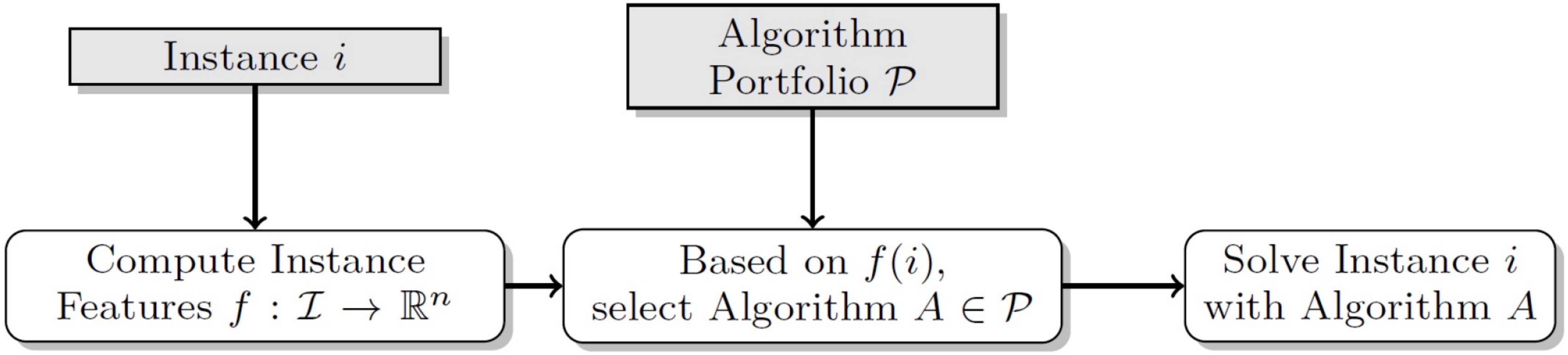
SUSTech Southern University of Science and Technology NICAL

- 100+ TSP manually designed features are now available [Kerschke et al., 2018]

Meta-feature	Description
V	Number of vertices
E	Number of edges
E_{low}	The lowest cost of edge
E_{hig}	The highest cost of edge
E_{avg}	Average of the edges costs
E_{std}	Standard deviation of the edges costs
E_{qmode}	Mode quantity of the edges costs
E_{fmode}	Frequency of the mode of the edges costs
E_{mode}	Average of the costs that occurs the most frequently in the edges
E_{median}	Median of the edges costs
E_{lavg}	Quantity of edges whose costs are lower than the average of the edges costs
V_{lower}	sum of the V edges of lower values

Basic Approach for AAS

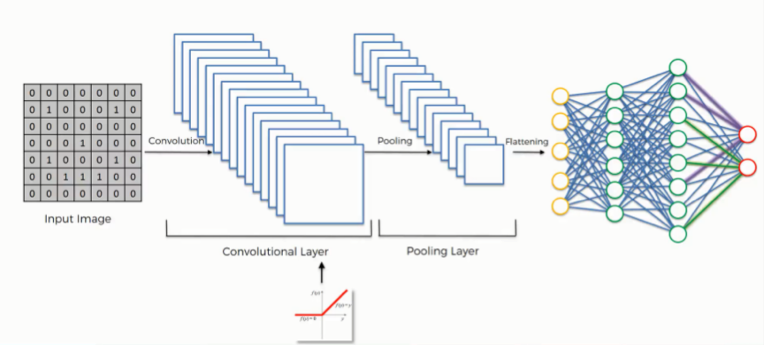
- Map instance features to the performance data by *regression*
  - Classification, Learning-to-rank, etc. can also be incorporated



- For a new problem instance:
  - Extract its features
  - Query the model the performance of the algorithms on this new instance
  - Apply the best expected algorithm

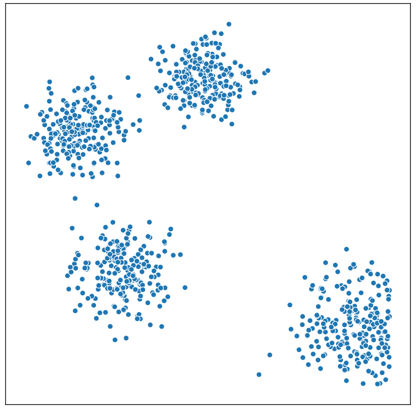
Deep Learning for AAS

- Identifying informative instance features is challenging
  - Domain expertise is required
  - Feature engineering/selection is generally necessary
  - More importantly, it needs to be done for every problem domain
- Deep learning could help address this issue — still in early research stage
  - Convert instance text files into images (ASCII code intro greyscale) [Loreggia et al., 2016]
  - Use visual representations of TSP instances, e.g., minimum spanning tree [Seiler et al., 2020]
  - CTAS [Zhao et al., 2021]: Convert the 2-D coordinates of TSP instances into images

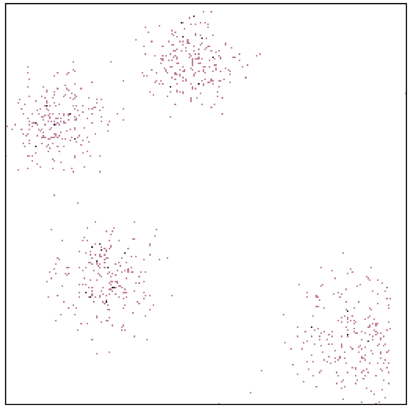


CTAS—TSP Instance to Image

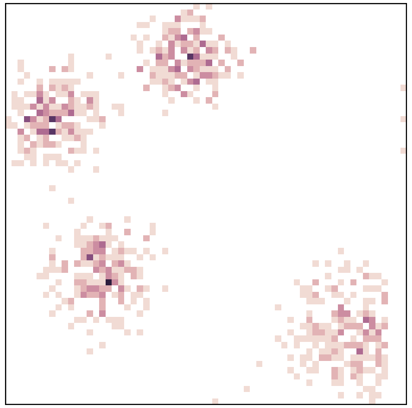
- 2D coordinates to density map



2D coordinate

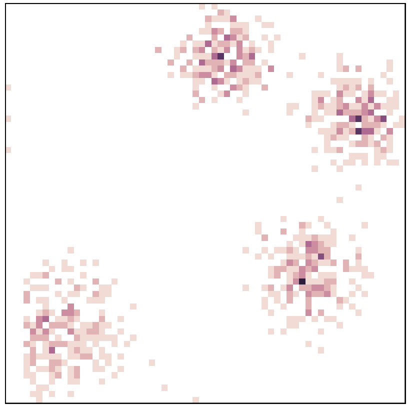


Grided density map

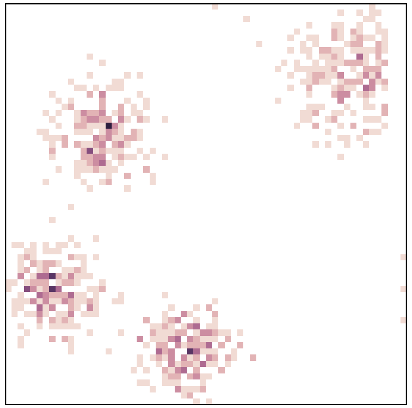


Interpolation enhancement

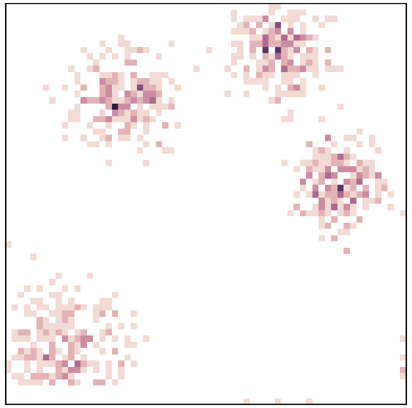
- Safe data augmentation



Vertical flip



Horizontal flip



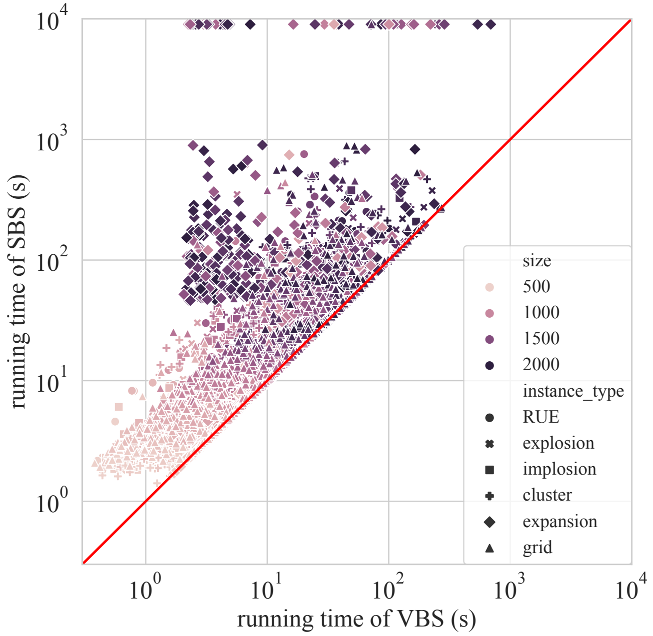
Rotation

CTAS—Data Preparation

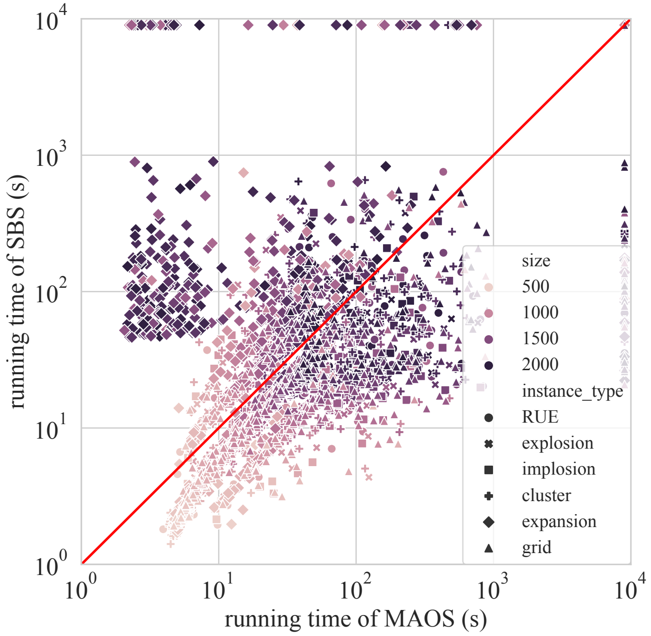
- 6,000 TSP instances, belonging to six different types
  - rue, explosion, implosion, expansion, cluster, grid
- Six TSP solvers: EAX, EAXr, LKH, LKHr, LKHc, MAOS

TABLE I: Solver performance statistics over the instances.

TSP set	Measure	EAX	EAXr	LKH	LKHr	LKHc	MAOS	VBS
rue	Unique	188	153	204	127	223	81	1000
	Shared	2	2	21	9	16	0	0
	Failed	254	1	11	10	9	6	0
	PAR10	2298.48	36.92	143.47	134.71	126.28	95.62	14.85
explosion	Unique	220	162	233	99	215	48	1000
	Shared	4	4	17	3	18	0	0
	Failed	194	1	5	3	3	5	0
	PAR10	1758.07	30.14	84.72	66.89	65.73	77.50	12.72
implosion	Unique	215	152	238	106	199	49	1000
	Shared	6	6	31	6	33	0	0
	Failed	193	1	10	10	11	4	0
	PAR10	1748.04	29.71	129.43	129.29	137.70	71.89	12.60
expansion	Unique	299	214	10	9	8	451	1000
	Shared	8	8	0	1	1	0	0
	Failed	507	42	316	318	319	11	0
	PAR10	4569.26	432.91	3001.87	3019.59	3026.38	123.02	19.67
cluster	Unique	239	191	184	83	177	90	1000
	Shared	6	6	24	9	27	0	0
	Failed	246	1	54	55	53	7	0
	PAR10	2225.21	34.35	546.62	555.76	541.13	100.63	14.51
grid	Unique	189	127	242	93	278	44	1000
	Shared	4	4	20	5	21	0	0
	Failed	234	1	15	13	14	34	0
	PAR10	2117.78	43.78	177.67	160.81	168.64	364.00	15.28
Total	Unique	1350	999	1111	517	1100	763	6000
	Shared	30	30	113	33	116	0	0
	Failed	1628	47	411	409	409	67	0
	PAR10	2452.81	101.30	680.63	677.84	677.64	138.78	14.94



SBS (EAXr) vs. VBS



SBS(EAXr) vs.
2nd best solver(MAOS)

CTAS—Experiment Results

TABLE IV: The test performance of CTAS and the baselines.

Selector Characteristics			Performance			
Learn. Stra.	**Model**	**Feat. Set / #Used Feat.**	**PAR10** (s)	**Avg. Rank**	**Impro.** (%)	**Notwo.** (%)
Reg.	DT	Pihera /4	41.01	3.08	7.50	91.17
Reg.	RF	UBC-cheap /8	58.39	3.05	14.89	85.22
Reg.	SVM	Pihera /3	68.47	3.00	16.72	85.28
Cla.	DT	Pihera /2	145.66	2.95	51.33	51.33
Cla.	SVM	UBC /11	164.64	2.90	53.22	53.22
Cla.	SVM	UBC-cheap /6	392.47	3.02	50.67	50.67
P-Reg.	SVM	UBC-cheap /5	118.35	3.00	49.56	49.56
P-Reg.	SVM	UBC /17	135.38	2.96	50.61	50.61
P-Reg.	DT	Pihera /2	147.05	4.28	17.83	17.83
Reg.	MLP	union /336	52.87	2.17	7.06	88.89
Cla.	MLP	union /336	57.66	2.12	7.22	90.67
Cla.	8-layer CNN	-	111.03	1.92	32.00	50.50
Reg.	ResNet18	-	**38.23**	2.30	7.70	80.17
Reg.	ResNet34	-	**36.89**	1.92	18.56	86.83
Cla.	ResNet18	-	40.91	2.11	6.70	92.09
Cla.	ResNet34	-	45.93	2.04	8.17	93.83
-	SBS	-	97.02	2.09	0.00	100.00
-	VBS	-	13.90	1.00	86.10	100.00

- CTAS achieves 2× speedup compared with SBS
- Regression is better then classification and pairwise regression

Literature on AAS

SUSTech Southern University of Science and Technology NICAL

[Kotthoff, 2016] http://larskotthoff.github.io/assurvey/

Comments? Suggestions? Corrections? Let me know!

Algorithm Selection Literature Summary

click headings to sort
click citations to expand

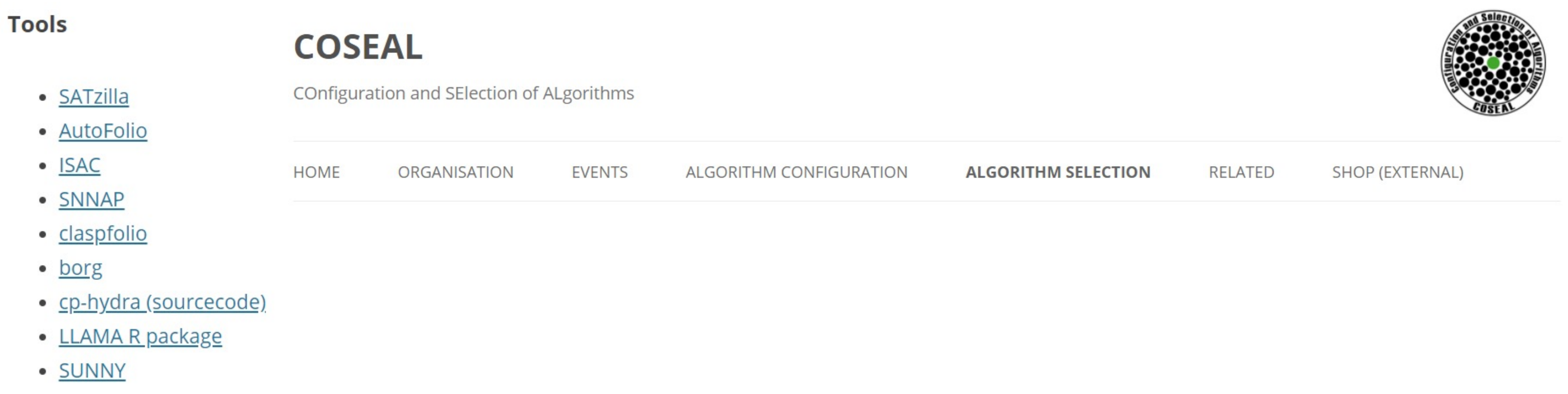
Fork me

Last update 10 July 2019

citation	domain	features	predict what	predict how	predict when	portfolio	year↑
Mantovani et al. 2019	machine learning	instance features	algorithm	classification	offline	static	2019
Loera et al. 2019	optimization	instance	algorithm	neural networks	offline	static	2019
Abdulrahman et al. 2019	machine learning	instance features	algorithm	ranking	offline	static	2019
Wang et al. 2018	CSP	instance features	algorithm	decision tree	offline	static	2018
Tripoul et al. 2018	pattern matching	simulation	constraint	hand-crafted model	online	static	2018
Silva et al. 2018	games	instance features	algorithm	logistic regression	online	dynamic	2018
Pavelski et al. 2018a, Pavelski et al. 2018b	flowshop	instance features	algorithm	decision trees, gradient boosting	offline	static	2018
Nikolić et al. 2018	theorem proving	instance features	algorithm, runtime performance	classification, regression	offline	static	2018
Kerschke et al. 2018	TSP	instance features	algorithm	classification, regression	offline	static	2018

Try it yourself!

- State-of-the-art algorithm selection libraries are freely available online (www.coseal.net/algorithm-selection)

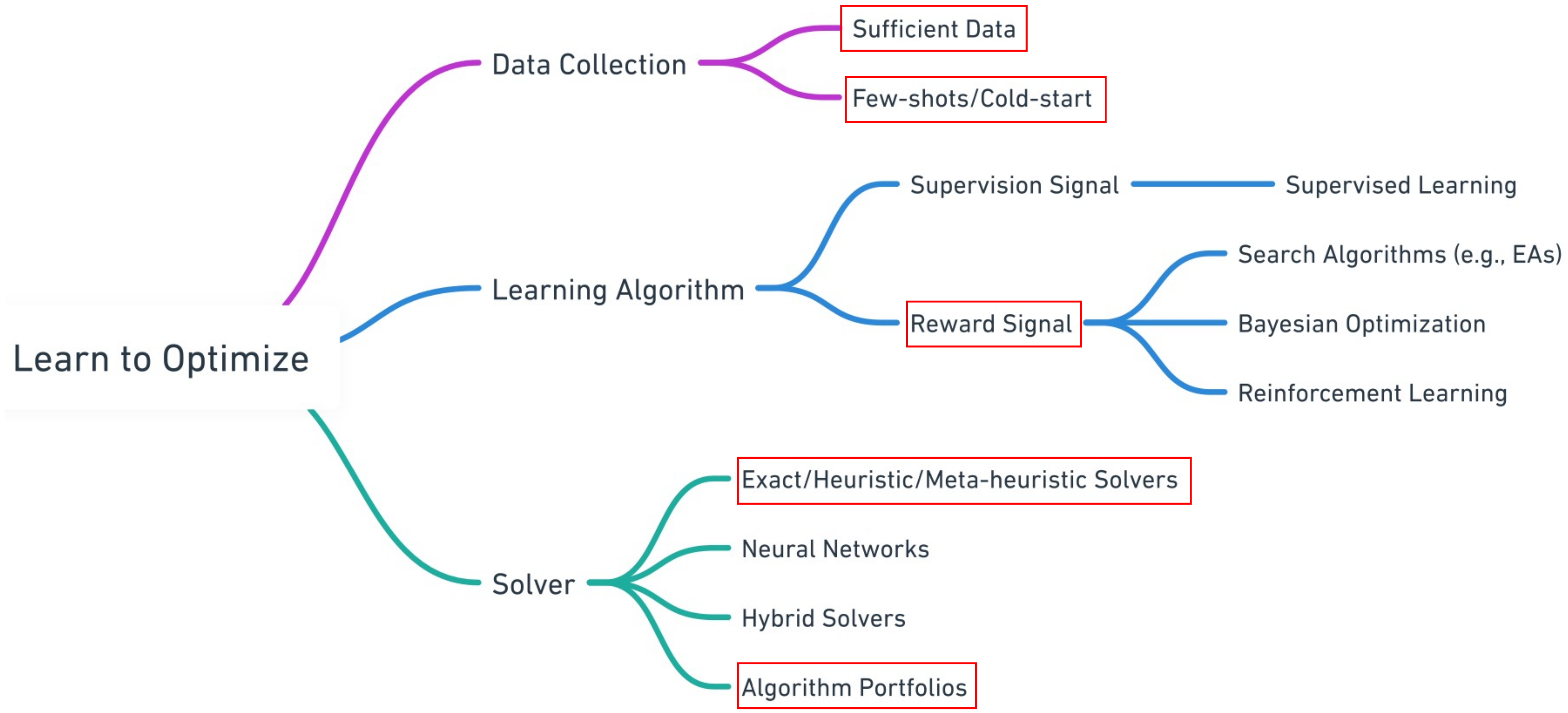


- You can try them for your problems
  - features for SAT, MIP, AI planning and TSP are available
  - you need to provide features for other problem domains
  - in many cases, the general ideas behind the features apply

Outline

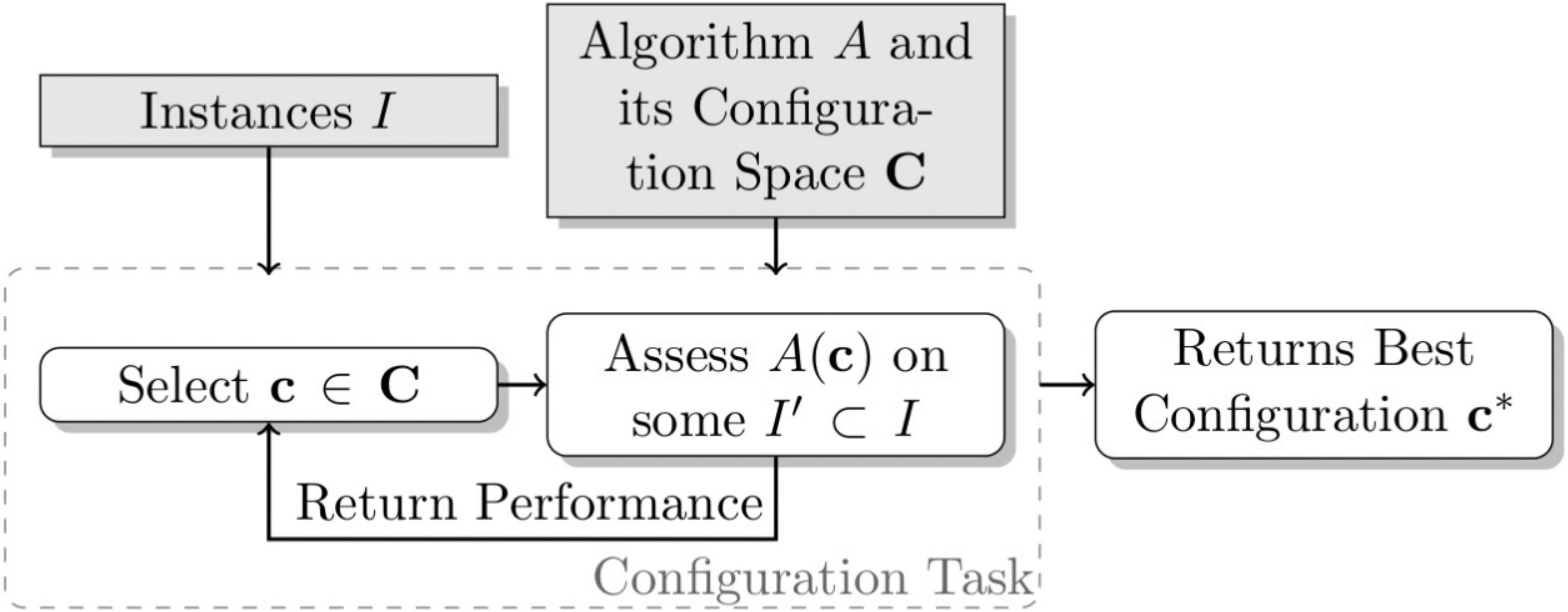
- Introduction to Learning to Optimize (L2O)
- Research Directions in L2O
  - Automatic Algorithm Selection
  - **Automatic Algorithm Configuration**
  - Neural Combinatorial Optimization
- Summary

Automatic Algorithm Configuration



The solver's performance heavily depends on its parameter configuration, how to identify the best configuration, or even to build a more powerful solver based on it?

Automatic Algorithm Configuration (AAC)



Source: https://www.coseal.net/algorithm-configuration/

Automatic Algorithm Configuration (AAC)

- How to select candidate configurations to evaluate?
  - Experimental design [Adenso-Diaz and Laguna, 2006] [Coy et al., 2001]
  - Heuristic/Meta-heuristic search [Hutter et al., 2009] [Ansótegui et al., 2009][Smit and Eiben, 2010]
  - Bayesian (model-based) optimization methods [Hutter et al., 2011] [Ansótegui et al., 2015]

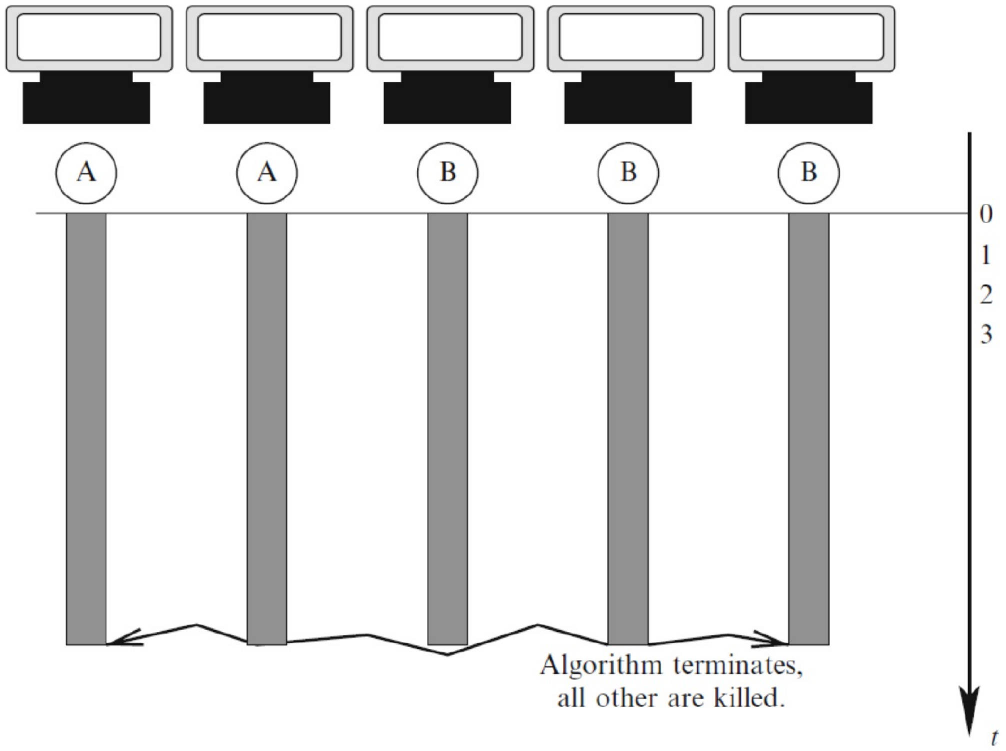
- How to evaluate candidate configurations with limited computational budget?
  - Split the budget as evenly as possible to all training problem instances [Birattari et al., 2007] [Liu et al., 2020]

Automatic Algorithm Configuration (AAC)

- AAC is a well studied area and is still in fast development
- Materials on AAC
  - Recent surveys: [Huang et al., 2019] [Stutzle and Lopez-Ibanez, 2019] [Eryoldas and Durmusoglu, 2021] [Schede et al., 2022]
  - Open-sourced Benchmarks: AClib (https://bitbucket.org/mlindauer/aclib2), and DAC (https://github.com/automl/DAC)
  - Open-sourced AAC tools: GPS [Pushak and Hoos, 2020], irace [Lopez-Ibanez et al., 2016], ParamILS [Hutter et al., 2009], SMAC [Hutter et al., 2011], GGA [Ansótegui et al., 2009], REVEC [Smit and Eiben, 2010]
- This tutorial focuses on a particular variant of AAC—*Automatic Construction of Parallel Algorithm Portfolios*

Parallel Algorithm Portfolios (PAPs)

- Run all component algorithms in parallel
- For decision problem, e.g., SAT
  - Once an algorithm terminates, all are killed
  - runtime = $\min\{t_1, t_2, \ldots, t_n\}$
- For optimization problem, e.g., TSP
  - algorithms terminate when time exhausted
  - cost = $\min\{c_1, c_2, \ldots, c_n\}$



- Features
  - Always achieve the best performance among the algorithms

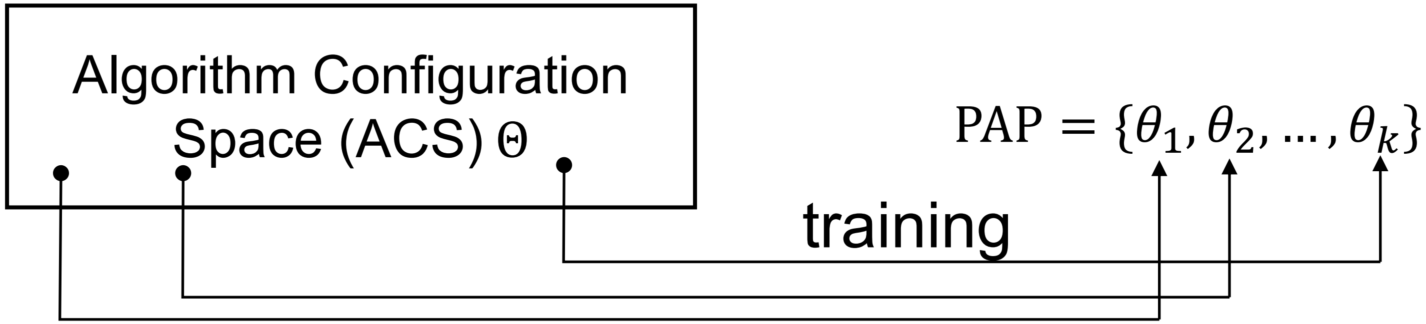
Why PAPs?

- Main advantages of PAPs
  - High-performance: by definition, “all” cannot be worse than “many”
  - Generality: applicable to nearly all kinds of computation problems
  - Easy to implement: friendly to modern computing facilities

- PAPs have shown promising performances in many areas, e.g.,
  - Boolean Satisfiability Problem (SAT) [Lindauer et al., 2017]
  - Classical/Satisficing/Agile Planning [Seipp et al., 2015]
  - Black-box Numerical Optimization [Tang et al., 2014]

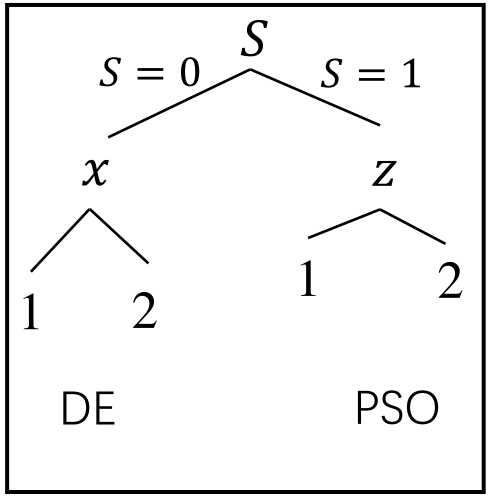
Automatic Construction of PAPs

- Automatically Identify algorithms (configurations) in PAPs



- ACS: a joint *parameter* space of *multiple* base algorithms

Two base algorithms DE and PSO
DE with one parameter x
PSO with one parameter z



It can involve as many base algorithms as you want!

The PAP Construction Problem

Goal: Find the optimal PAP w.r.t metric m and target instance set I^*

$$\theta^*_{1:k} = \underset{\theta_{1:k} \in \boldsymbol{\Theta}^k}{\arg\min}\, \mathrm{m}(\theta_{1:k}, I^*)$$

$\mathrm{PAP} = \theta_{1:k} = \{\theta_1, \dots, \theta_k\}$

m: performance indicator (runtime, solution quality)

Target problem instance set

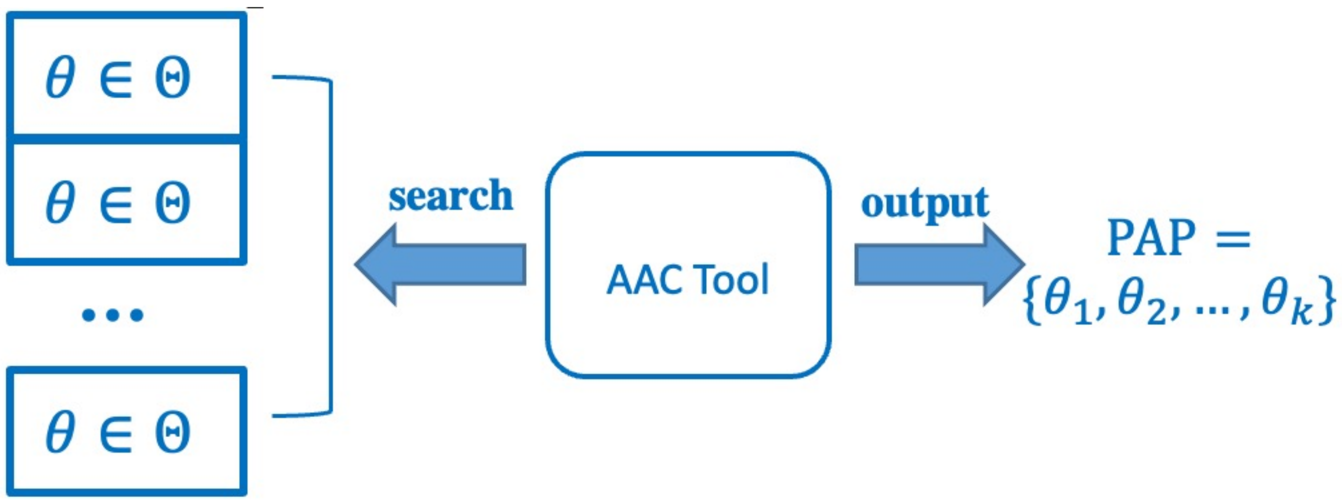
Approximated by a training set I

Main Research Questions

- If I is sufficient
  - How to identify a good set of algorithm configurations (i.e., PAP)?
- If I is insufficient
  - How to construct a PAP that can still generalize well?

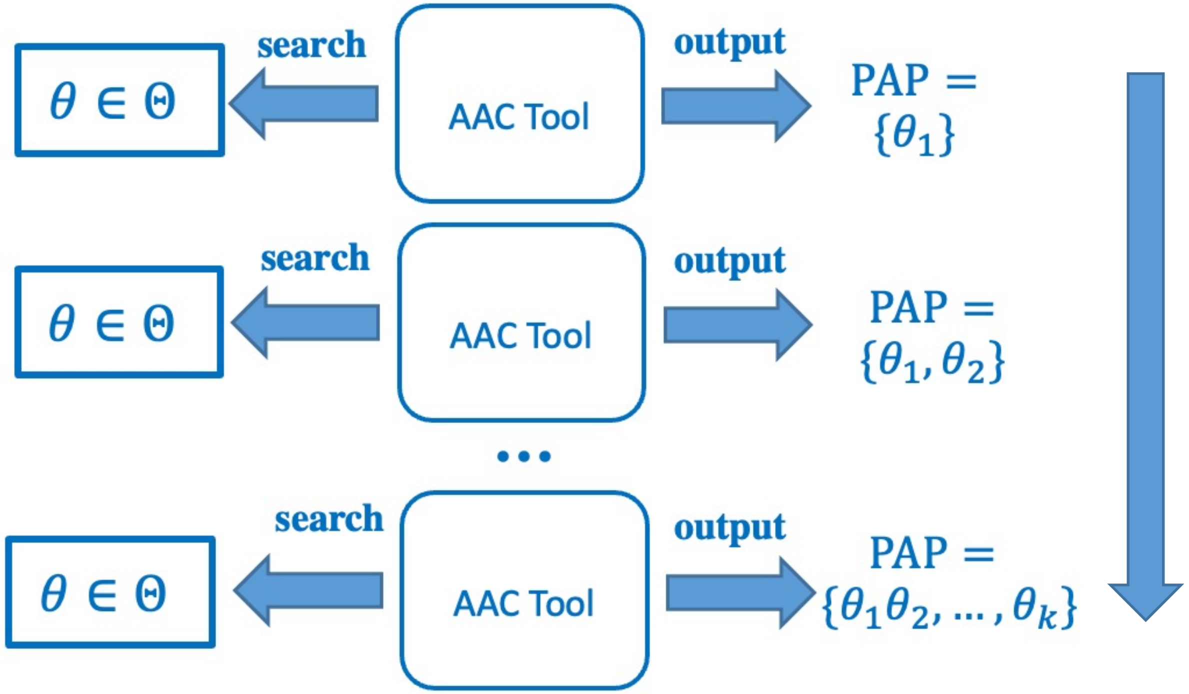
Training PAPs with Sufficient Data

- GLOBAL [Lindauer et al., 2017]
  - Treat the construction of PAPs as an algorithm configuration problem
  - configures all component solvers simultaneously
  - full configuration space size $|\Theta|^k$, increases exponentially with k



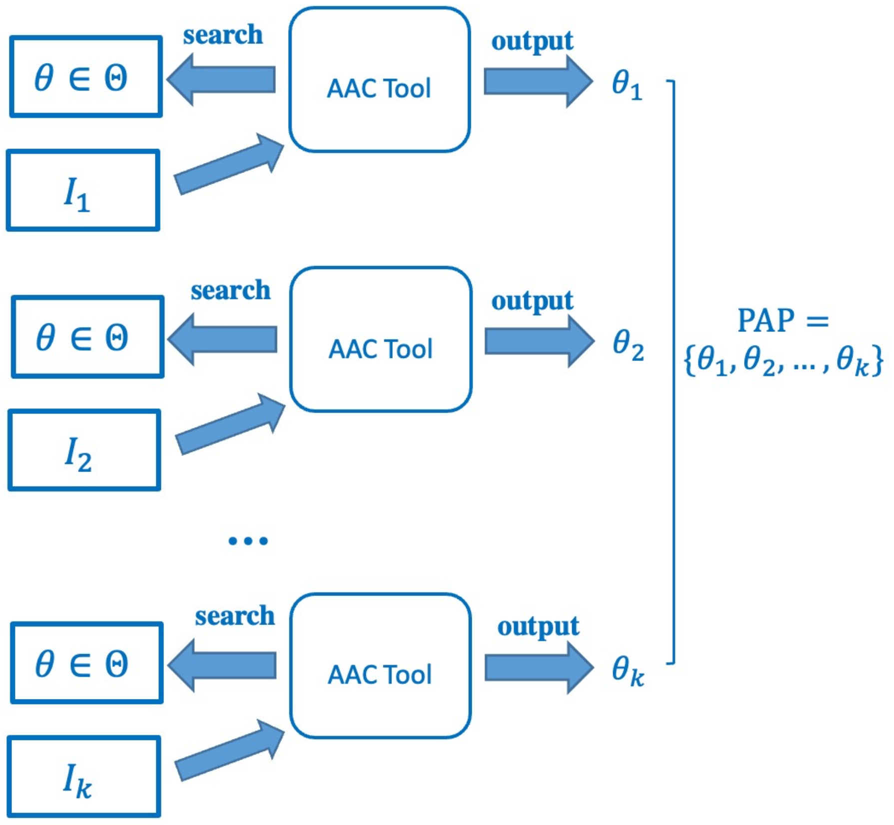
Training PAPs with Sufficient Data

- PARHYDRA [Lindauer et al., 2017]
  - An iterative method that configures b component solvers at each step
  - When $b = k$, PARHYDRA=GLOBAL
  - $b \uparrow$: limited scalability; $b \downarrow$: tend to stuck in local optimum



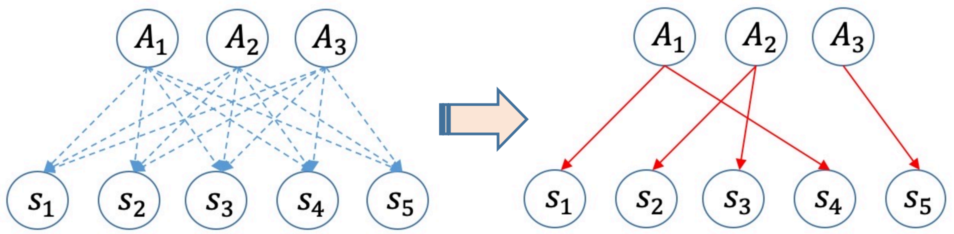
Training PAPs with Sufficient Data

- CLUSTERING [Kadioglu et al., 2010]
  - Split training set I into k disjoint subsets, based on the distances in feature space
  - Configure a component solver on each subset
  - The most informative features and the best normalization strategies are unknown in advance

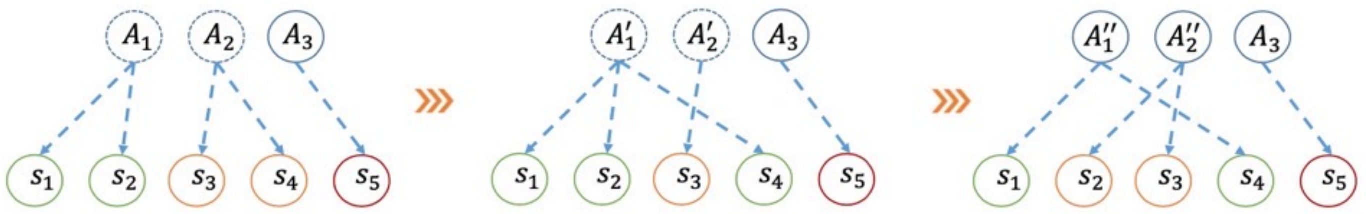


Training PAPs with Sufficient Data

- If the training set could be appropriately “clustered”, k solvers could be tuned separately



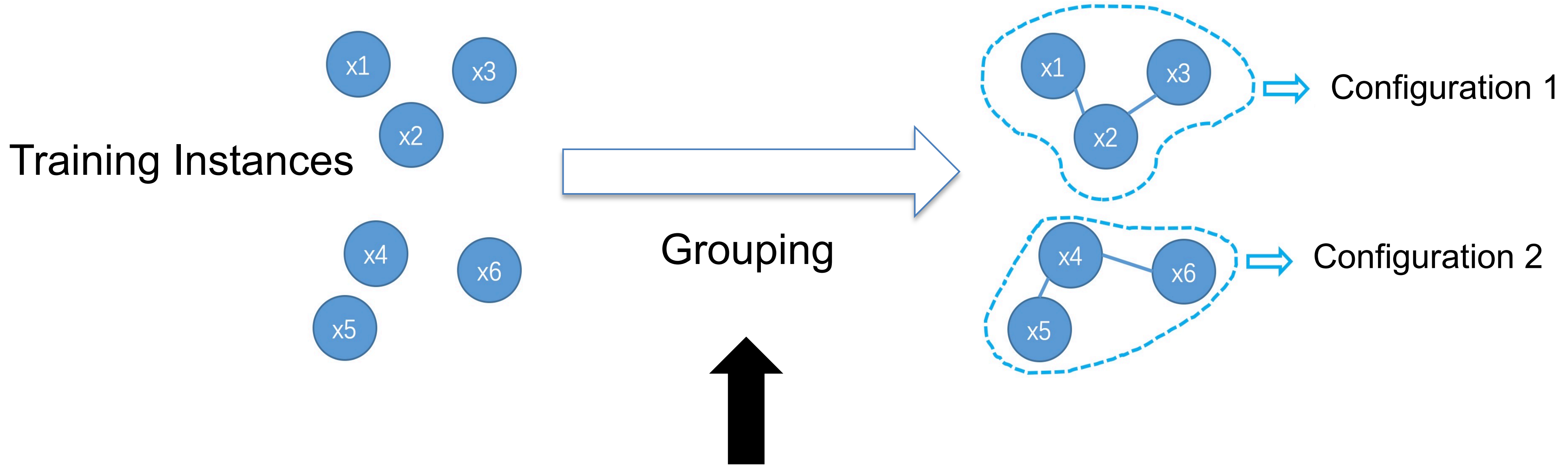
- But how to cluster training instances without tedious problem-specific feature engineering?
  - A problem-independent approach: using algorithm behavior data as instance feature



Evolve to find one’s niche

PCIT

- Parallel Configuration via explicit instance grouping (PCIT) [Liu et al., 2019]



Instance	Configuration	Performance	Problem feature (optional)
…	…	…	…
…	…	…	…
…	…	…	…

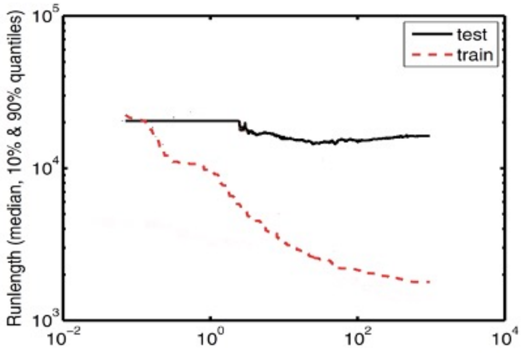
S. Liu, K. Tang and X. Yao, "Automatic Construction of Parallel Portfolios via Explicit Instance Grouping," *AAAI 2019*

Insufficient Training Data

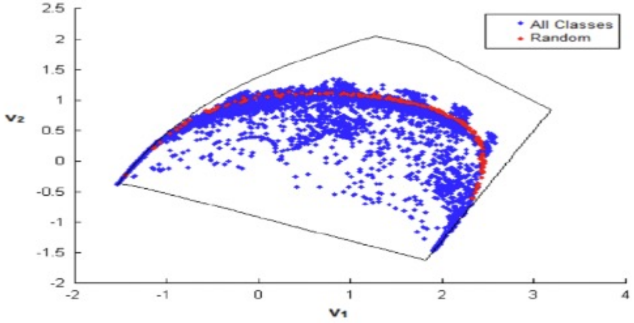
- What if we don’t have enough training instances?
  - Suppose a solver for combinatorial optimization problem is to be learned

Problem	No. of Instance	
TSP	143	http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html
CVRP	319	http://vrp.atd-lab.inf.puc-rio.br/index.php/en/
VRPSPDTW	85	https://github.com/senshineL/VRPenstein

Indeed, benchmark set is small



Overfitting may also occurs [1]



Randomly generated instances are biased [2]

Source [1]: https://www.cs.ubc.ca/~hoos/PbO/Tutorials/IJCAI-16
Source [2]: K. Smith-Miles, and B. Simon, “Generating new test instances by evolving in instance space.” *Computers & Operations Research* 2015 (63): 102-113.

Intelligently Generating More Instances

- Given a training set I, to generate a set $\bar{I}$ of additional instances

$$\mathrm{m}(\bar{P}, I^*) - \mathrm{m}(P, I^*) := \frac{1}{|I^*|}\sum_{z \in I^*} \mathrm{m}(\bar{P}, z) - \frac{1}{|I^*|}\sum_{z \in I^*} \mathrm{m}(P, z)$$

$$\boxed{\begin{aligned} &= \frac{1}{|I^*|}\Big(\sum_{z \in I} \mathrm{m}(\bar{P}, z) + \sum_{z \in \bar{I}} \mathrm{m}(\bar{P}, z) + \sum_{z \in I^* \setminus (I \cup \bar{I})} \mathrm{m}(\bar{P}, z)\Big) \\ &- \frac{1}{|I^*|}\Big(\sum_{z \in I} \mathrm{m}(P, z) + \sum_{z \in \bar{I}} \mathrm{m}(P, z) + \sum_{z \in I^* \setminus (I \cup \bar{I})} \mathrm{m}(P, z)\Big)\end{aligned}}$$

➢ $\forall z \in I^*$, $\mathrm{m}(\bar{P}, z) \leq m(P, z)$ holds when $\mathrm{P} \subset \bar{P}$

➢ Thus

$$\mathrm{m}(\bar{P}, I^*) - \mathrm{m}(P, I^*) \leqslant \boxed{\frac{1}{|I^*|}\Big(\sum_{z \in \bar{I}} \mathrm{m}(\bar{P}, z) - \sum_{z \in \bar{I}} \mathrm{m}(P, z)\Big).}$$

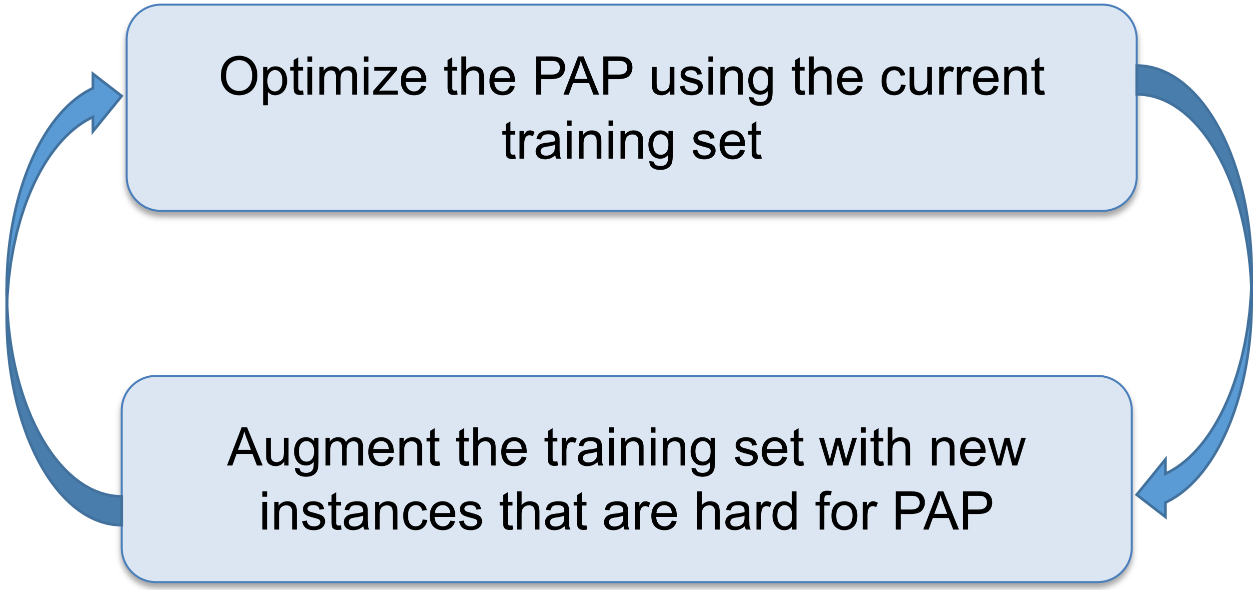
Computable Upper-bound of the improvement on generalization

Competitive Game

- Iterative two-step procedure to minimize the upper bound [Liu et al., 2020]

  a) Generate $\bar{I}$ to maximize $\sum_{z \in \bar{I}} m(P, z)$ — find hard instances for current PAP

  b) Train $\bar{P}$ with $I \cup \bar{I}$ to minimize $\sum_{z \in \bar{I}} m(\bar{P}, z)$ — improve PAP on new instances



S. Liu, K. Tang and X. Yao, "Generative Adversarial Construction of Parallel Portfolios," *IEEE Transactions on Cybernetics*, 2022, 52(2): *784-795.*

CEPS: Co-evolving PAPs and Training Instances

- A co-evolutionary framework to construct generalizable PAP [Tang et al., 2021]
  - Two competitive populations PAP and I
  - Conflicting objectives: minimize $\sum_{z \in \bar{I}} m(\bar{P}, z)$ and maximize $\sum_{z \in \bar{I}} m(P, z)$
- Widely applicable to nearly all problems and algorithms
  - Need to design mutation and crossover operators to handle Θ and I^*
  - Available at https://github.com/senshineL/CEPS

Evolve I to generate hard instances for PAP

Evolve PAP to better solve I

K. Tang, S. Liu, P. Yang and X. Yao, "Few-shots parallel algorithm portfolio construction via co-evolution." *IEEE Transactions on Evolutionary Computation*, 2021, 25(3): 595-607.

Experiment Results

SUSTech Southern University of Science and Technology NICAL

Training time < 7 days, with 40core Intel Xeon machines with 128 GB RAM (2.20 GHz, 30 MB Cache)

SAT_PAP[1]	TSP_PAP[2]	VRPSPDTW_PAP[3]
Competitive to SOTA	Significantly outperform SOTA	New best solution

	Agile Track		Parallel Track	
	#TOs	PAR-10(s)	#TOs	PAR-10(s)
SAT_PAP	**181**	**119**	**35**	**1164**
Priss6	225	146	-	-
PfolioUZK	-	-	**36**	**1185**
Plingeling-bbc	452	276	**33**	**1090**

Priss6: SAT'16 Competition (Agile Track) Winner

PfolioUZK: SAT'12 Competition (Parallel Track) Winner

Plingeling-bbc: SAT'16 Competition (Parallel Track) Winner

	#TOs	PAR-10(s)	ADR(‰)
TSP_PAP	**7**	**2369.38**	**0.02**
LKH	50	14242.85	0.56
EAX	16	4928.88	0.24
LKH-TUNED	43	12319.73	0.51
EAX-TUNED	14	4364.25	0.16
EAX_LKH	9	2921.51	0.12

LKH, EAX: **Best** TSP solvers so far

LKH/EAX-TUNED: LKH & EAX with **further finetuning**

EAX_LKH: **PAP** consisted with LKH & EAX

Instance Type	#instances	#better	#not-worse
Rdp	23	19	22
Cdp	17	9	9
RCdp	16	16	16
count	56	**44**	**47**

BKS: Best Known Solutions by September 2020

Solver \ Cost	Rdp103	Rdp206	Cdp108	RCdp102	RCdp105
VRPSPDTW_PAP	**2594.64**	**1206.14**	**1932.49**	**2770.28**	**2946.36**
Co-GA	2616.16	1261.32	1951.24	2897.05	2981.26
p-SA	2626.77	1259.94	2063.73	2822.76	2981.54
ALNS-PR	2597.01	1213.68	1932.88	2783.62	2948.96

[1] S. Liu, K. Tang and X. Yao, "Automatic Construction of Parallel Portfolios via Explicit Instance Grouping," *AAAI* 2019

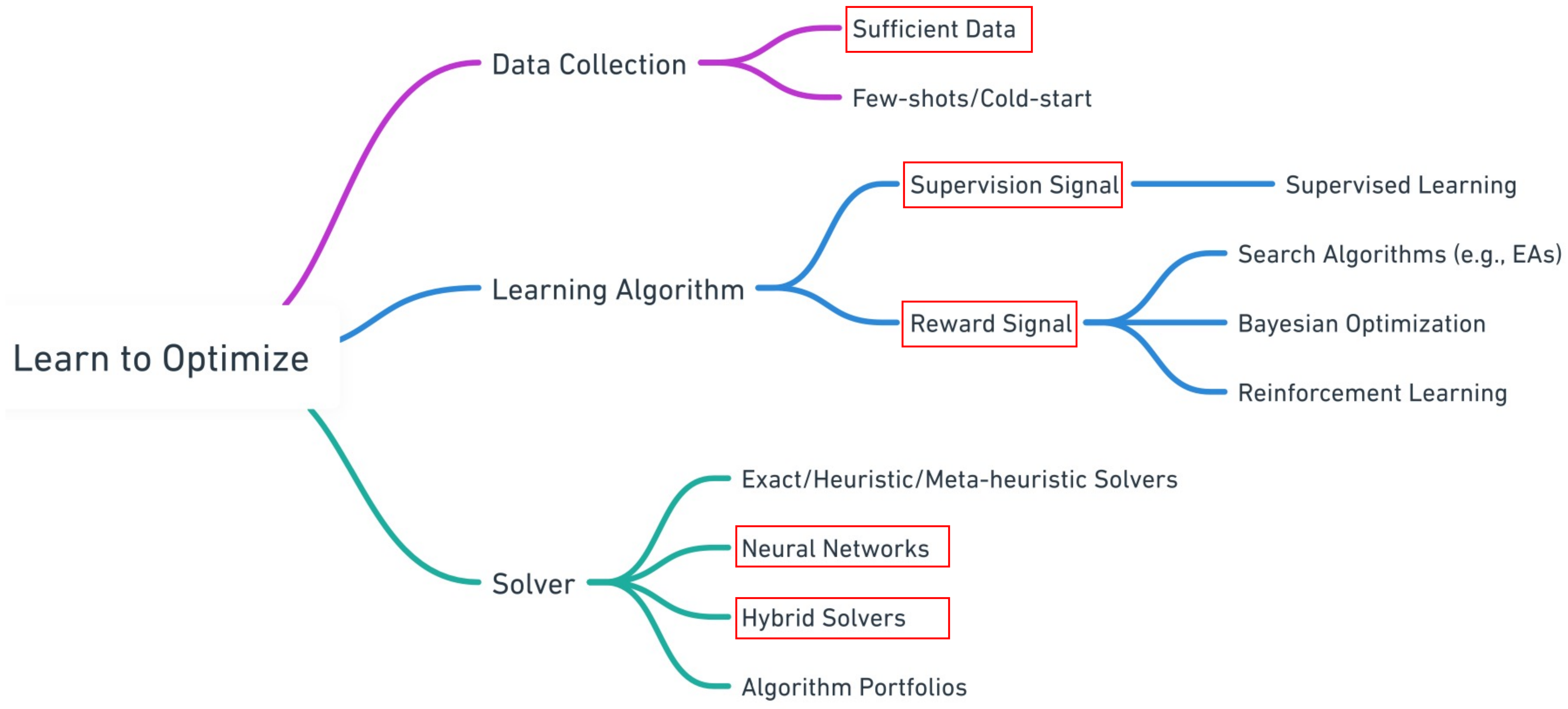
[2] S. Liu, P. Yang, K. Tang. Approximately optimal construction of parallel algorithm portfolios by evolutionary intelligence (in Chinese). *Sci Sin Tech*, 2022

[3] K. Tang, S. Liu, P. Yang and X. Yao, "Few-shots Parallel Algorithm Portfolio Construction via Co-evolution," *IEEE Transactions on Evolutionary Computation*, 2021, 25(3): 595-607.

Outline

- Introduction to Learning to Optimize (L2O)
- Research Directions in L2O
  - Automatic Algorithm Selection
  - Automatic Algorithm Configuration
  - **Neural Combinatorial Optimization**
- Summary

Neural Combinatorial Optimization



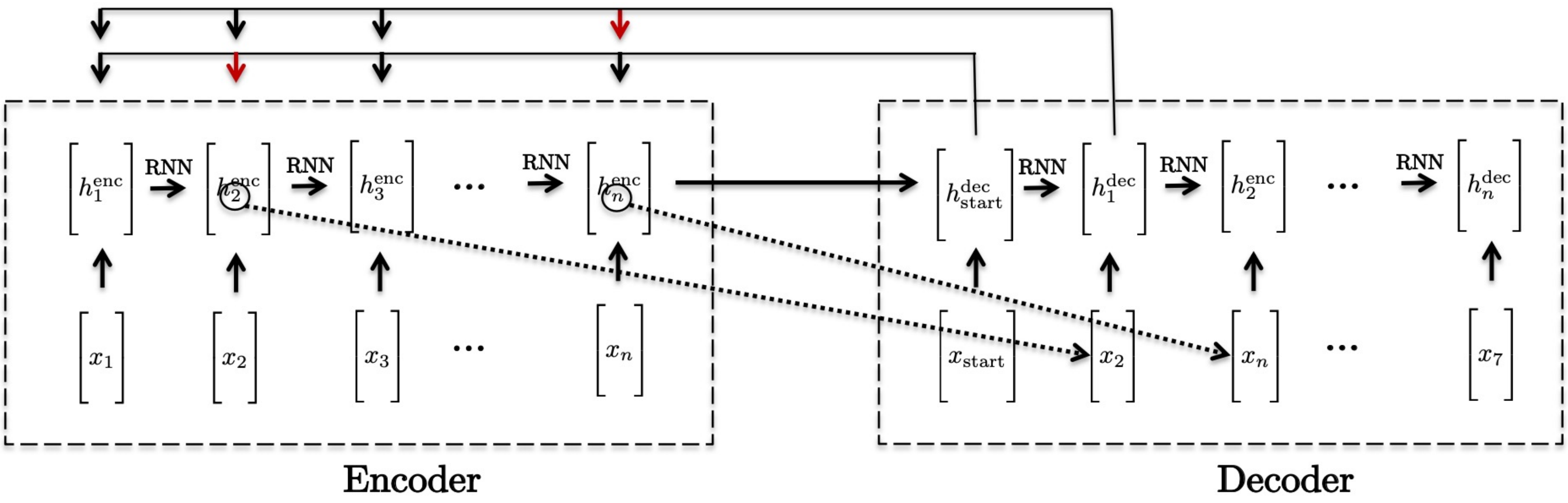
How to build an effective solver in a unified framework that is applicable to a wide range of problems?

Neural Combinatorial Optimization (NCO)

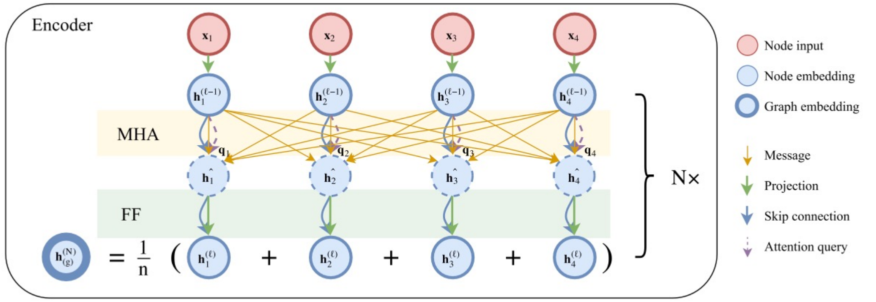
- Deep learning (DL) has achieved huge success over the past decade
  - ➢ Huge amount of training data and massively parallel computing platforms (GPUs)
  - ➢ Replacing hand-crafted features by features learned from data
- Can DL be used to learn heuristics (solvers) for optimization problems?
- The last five years have seen the emergence of such promising techniques
- This tutorial focuses on combinatorial optimization (NCO)

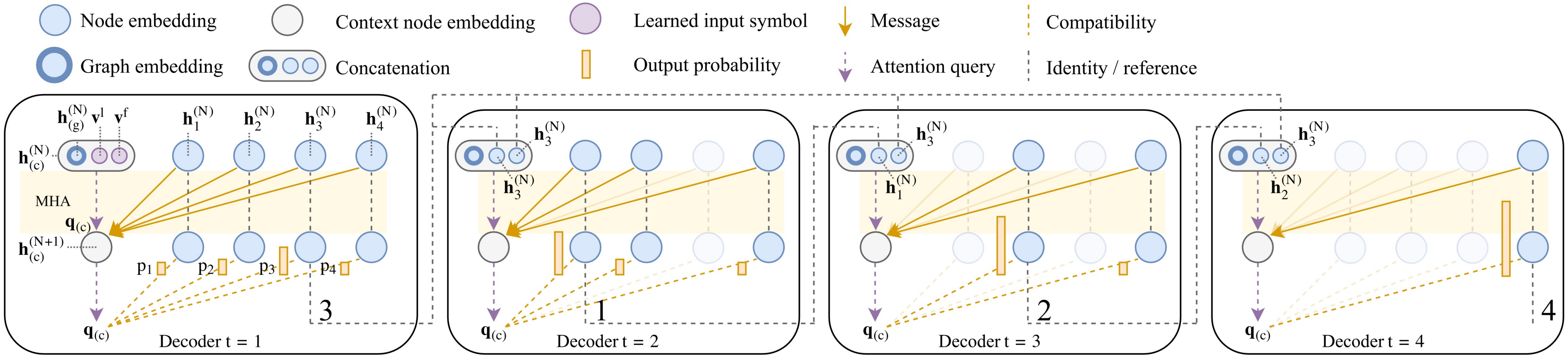
Neural Combinatorial Optimization (NCO)

- Many CO problems are similar to Translation in Natural Language Processing
  - mapping from a set (sequence) to a sequence
- The seminal work—Pointer Networks (Ptr-Net) for solving TSP [Vinyals et al., 2015]
  - Use RNNs to encode the cities and decode the node sequence of the tour sequentially
  - Trained by supervised learning with approximately optimal TSP solutions



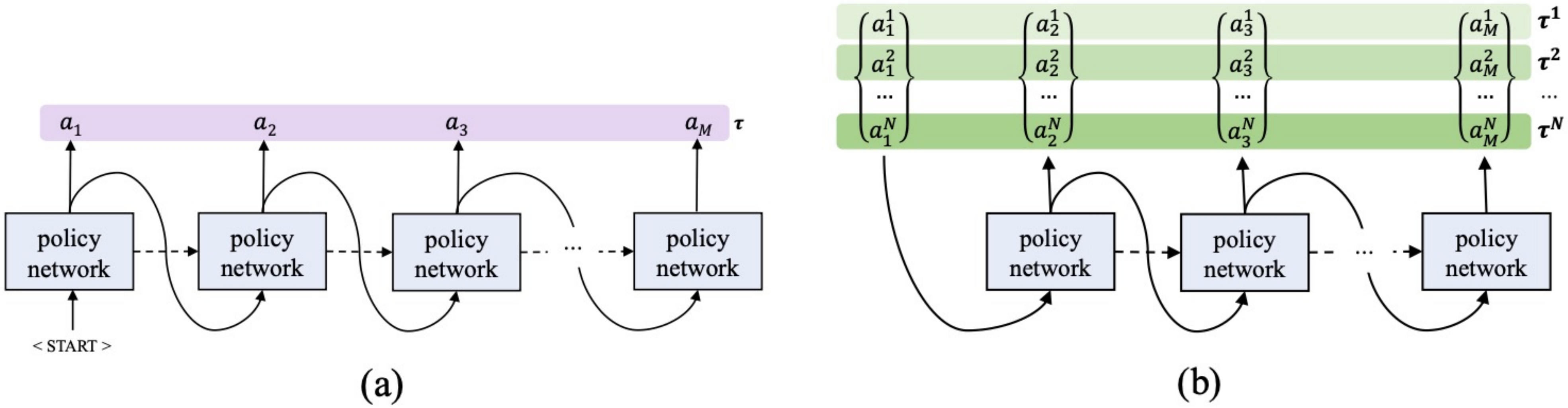
- Attention Model (AM) [Kool et al., 2018] improves based upon Ptr-Net
  - ➢ Attention-based Encoder learns the embedding of each node and the problem instance
  - ➢ Attention-based Decoder decodes the sequence
  - ➢ Training the solvers with Reinforcement Learning



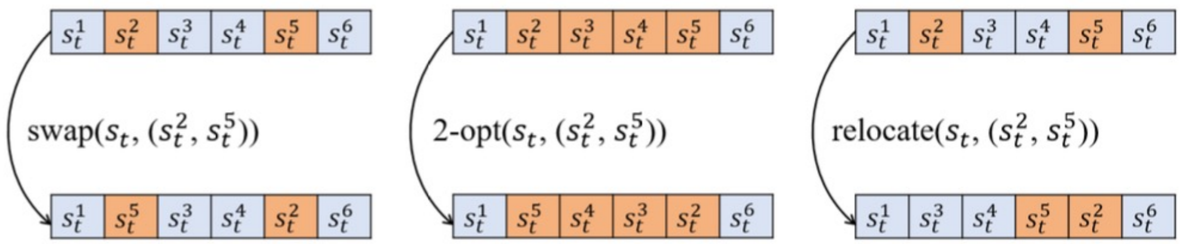


Learning Constructive Heuristics

- POMO [Kwon et al., 2020] improves based upon AM
  - Leverages multiple trajectories in parallel rather than the single one in AM
  - Uses data augmentation (during inference/solving stage)



Learning Improvement Heuristics



- LIH [Wu et al., 2021] combines DL with traditional move operators
  - ➢ Learns a policy to pick node pair to perform 2-opt/local swap operators
  - ➢ The policy is a transformer-based model, trained by reinforcement learning
  - ➢ Later, LIH was improved in [Ma et al. 2021], dubbed DACT

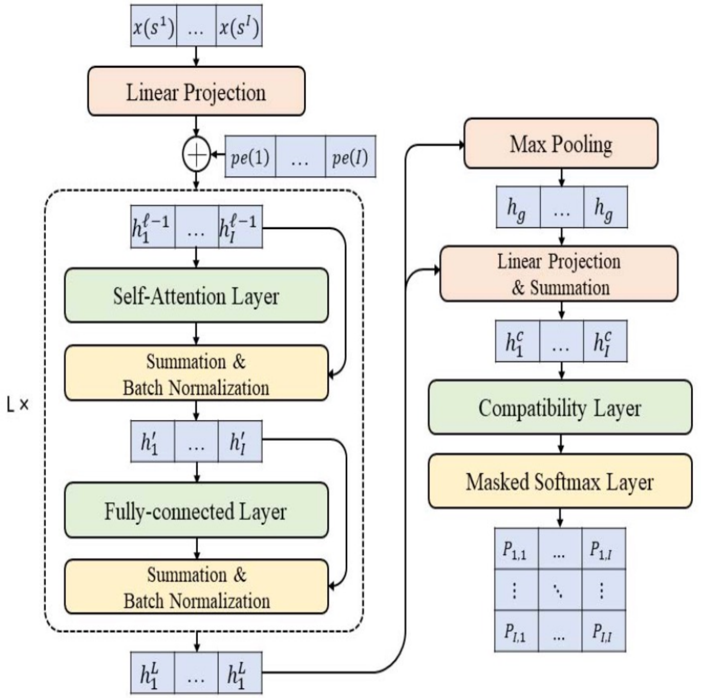


Fig. 2. Architecture of policy network (left: node embedding; right: node pair selection).

Learning Hybrid Solvers

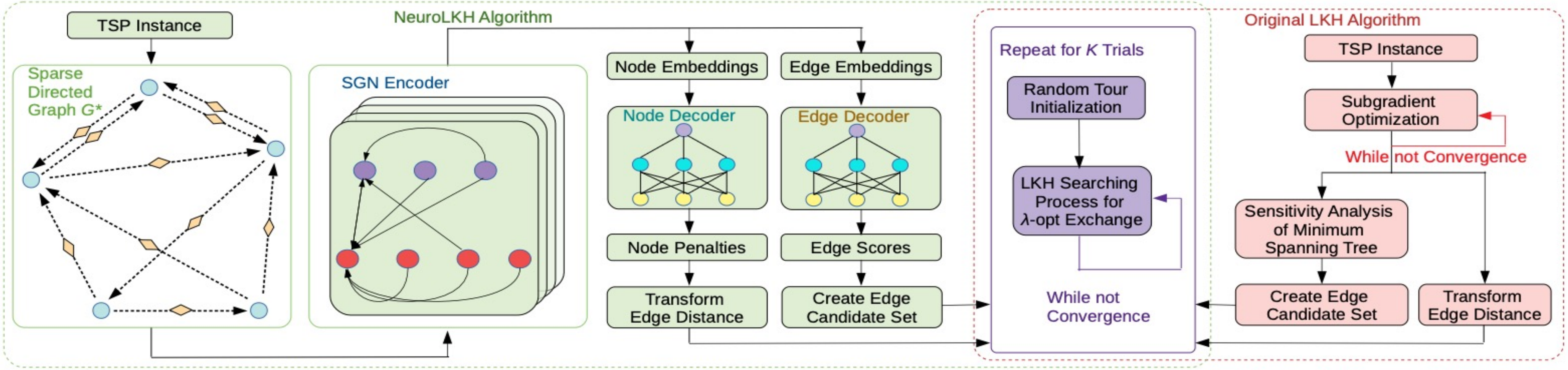


Figure 1: NeuroLKH algorithm and the original LKH algorithm.

- NeuroLKH [Xin et al., 2021] combines DL with the well-known LKH solver for TSP
  - ➢ Learns a Sparse Graph Network (SGN) for edge scores to create edge candidate set for LKH
  - ➢ The network is trained by supervised learning with optimal solutions

A Comparative Study on TSP

- Benchmark Instances
  - Random uniform instances (rue)
  - “Clustered” instances (clust)
  - 1,000,000 instances for training, 10,000 instances for testing

- Competitors—the best NCO approaches and traditional solvers
  - POMO (learning constructive heuristic)
  - DACT (learning improvement heuristic)
  - NeuroLKH (learning hybrid solver)
  - LKH [Helsgaun, 2017], EAX [Nagata and Kobayashi, 2013], and their tuned variants

A Comparative Study on TSP

- Performance Metrics (the smaller the values are, the better)
  - Solution quality, measured by the gap between the found solution and the optimal solution
  - Wall-clock runtime
  - Consumed energy

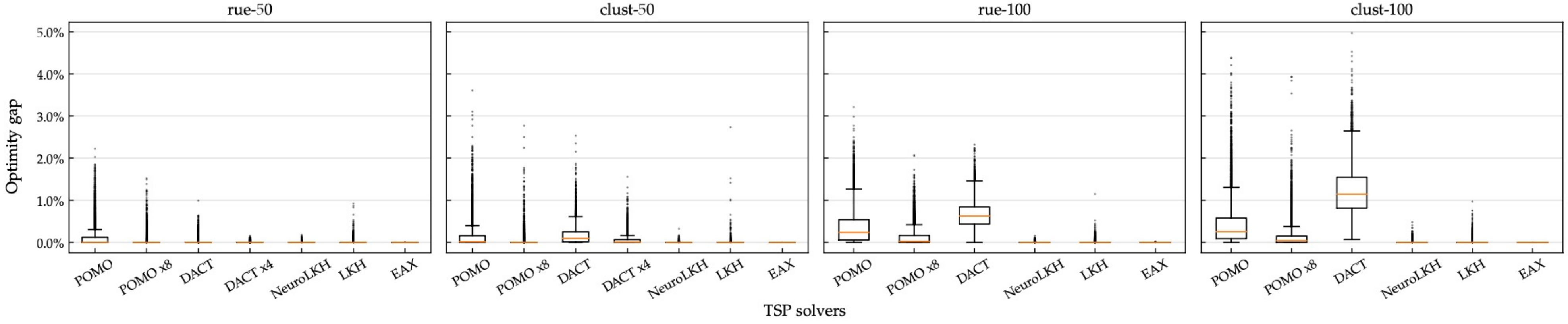
- Experiment Environment
  - NVIDIA TITAN RTX GPU (24 GB video memory) for DNN-based solvers
  - Xeon Gold 6240 CPU (2.60 GHz, 24.75 MB Cache) for traditional solvers (running *in parallel* with 32 CPU threads)

Experiment Results on Small-scale Instances

SUSTech Southern University of Science and Technology NICAL

TABLE 1
Testing results of experiment 1 which is designed to assess the effectiveness, efficiency and stability of the solvers on small-size problem instances. The results are presented in terms of the average optimum gap, the total computation time and energy consumed by the solver. For each metric, the best performance is indicated in grey. Note for DACT with instance augmentation mechanism, its results on *rue*-100 and *clust*-100 are missing because it runs prohibitively long to solve the testing instances.

Method	rue-50			clust-50			rue-100			clust-100		
	Gap (%) ± std (%)	Time (s)	Energy (J)	Gap (%) ± std (%)	Time (s)	Energy (J)	Gap (%) ± std (%)	Time (s)	Energy (J)	Gap (%) ± std (%)	Time (s)	Energy (J)
POMO, no aug.	0.1185 ± 0.0000	2.57	290.83	0.1353 ± 0.0000	2.58	292.14	0.3646 ± 0.0000	12.59	2588.90	0.4318 ± 0.0000	12.83	2675.18
POMO, ×8 aug.	0.0228 ± 0.0000	16.98	3361.87	0.0213 ± 0.0000	17.04	4193.39	0.1278 ± 0.0000	87.73	25873.38	0.1405 ± 0.0000	93.08	27299.18
DACT	0.0167 ± 0.0291	1991.49	402635.40	0.1770 ± 0.1117	1921.99	393994.52	0.6596 ± 0.5216	6141.72	1269009.13	1.2220 ± 0.4773	6517.2110	1390740.24
DACT, ×4 aug.	0.0006 ± 0.0013	8534.42	1742933.29	0.0576 ± 0.0390	8735.50	1676140.80		-			-	
NeuroLKH	0.0003 ± 0.0003	34.22	4006.11	0.0004 ± 0.0007	131.92	12698.78	0.0004 ± 0.0005	74.90	9819.77	0.0021 ± 0.0031	309.75	29397.05
LKH	0.0035 ± 0.0035	291.22	10458.70	0.0022 ± 0.0018	257.95	9512.24	0.0044 ± 0.0048	313.73	12868.38	0.0048 ± 0.0040	340.58	14264.64
EAX	0.0000 ± 0.0000	343.79	15089.39	0.0000 ± 0.0000	321.06	12541.47	0.0000 ± 0.0000	598.34	35145.37	0.0000 ± 0.0000	561.41	27893.49



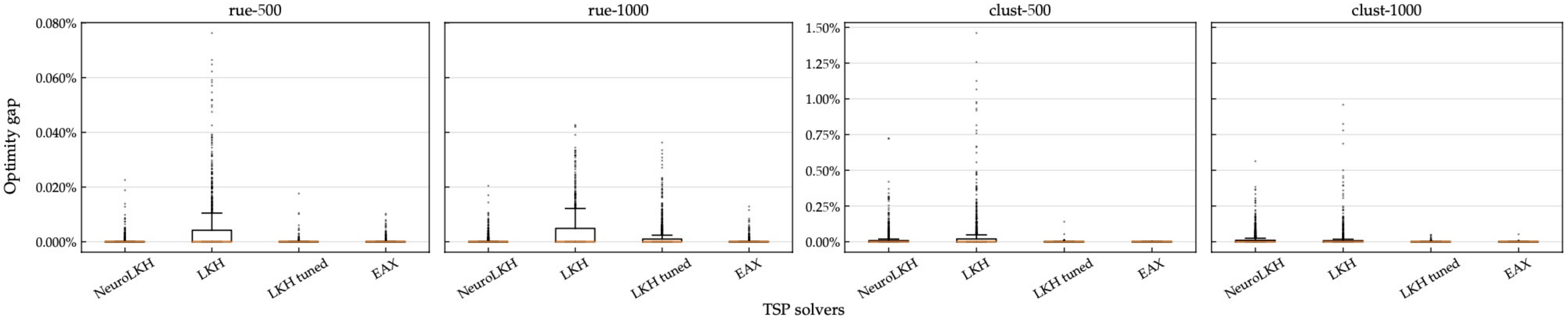
➢ Traditional solvers consistently obtain better solutions

➢ POMO exhibits excellent efficiency in terms of both runtime and energy

Experiment Results on Medium-scale Instances

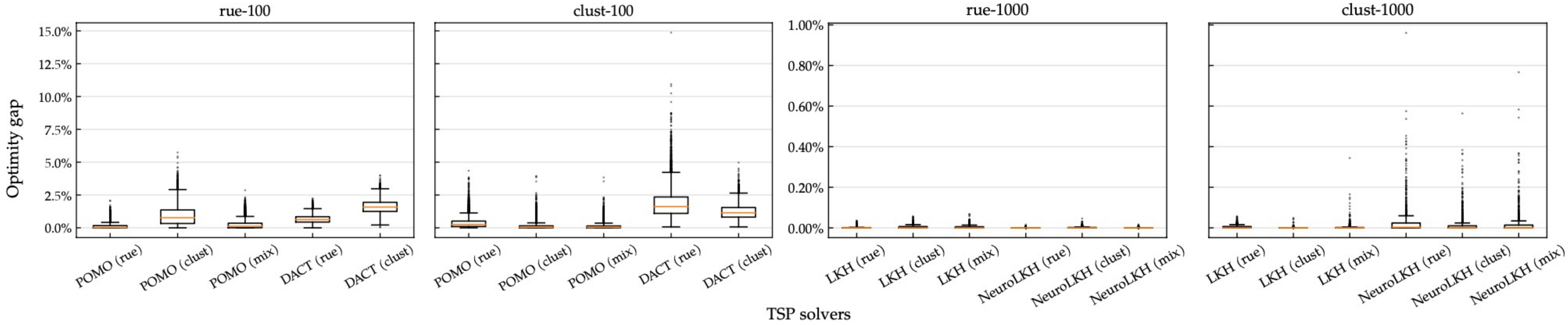
TABLE 2
Testing results of experiment 2 which is designed to assess the effectiveness, efficiency and stability of the solvers on large-size problem instances. The results are presented in terms of the average optimum gap, the total computation time and energy consumed by the solver. For each metric, the best performance is indicated in grey. Note in this experiment POMO and DACT are not tested due to their poor scalability.

Method	rue-500			clust-500			rue-1000			clust-1000		
	Gap (‱) ± std (‱)	Time (s)	Energy (J)	Gap (‱) ± std (‱)	Time (s)	Energy (J)	Gap (‱) ± std (‱)	Time (s)	Energy (J)	Gap (‱) ± std (‱)	Time (s)	Energy (J)
NeuroLKH	0.0273 ± 0.0441	242.71	21455.69	1.8080 ± 2.5469	2695.06	171826.14	0.0417 ± 0.0510	728.41	58569.96	1.8599 ± 2.2236	5094.77	334771.95
LKH	0.4356 ± 0.5084	143.36	17077.63	4.2779 ± 3.6307	313.80	35213.13	0.3620 ± 0.3624	413.43	48867.49	1.8761 ± 1.2798	859.39	100514.45
LKH (tuned)	0.0086 ± 0.0165	162.63	19087.78	0.0345 ± 0.0398	279.80	32685.72	0.1732 ± 0.1770	406.16	52252.97	0.0460 ± 0.0537	522.41	66511.40
EAX	0.0140 ± 0.0291	269.41	32370.64	0.0006 ± 0.0012	209.09	24456.69	0.0182 ± 0.0242	620.30	75168.05	0.0086 ± 0.0122	631.68	75222.30



- ➢ NeuroLKH can improve LKH in solution quality, but needs to consume much more runtime and energy
- ➢ Automatic Algorithm Configuration can improve LKH in solution quality, runtime and energy

On Generalization over Instance Types



- The performance of a learned solver would be degraded over different problem types
- Even if a mixed training set is used, the learned/trained solver still cannot achieve the best possible performance

On Generalization over Instance Sizes

SUSTech Southern University of Science and Technology NICAL

TABLE 3
Testing results of experiment 4 which is designed to assess the learned solvers' generalization ability over different problem sizes. The results are presented in terms of the average optimum gap.

Method (training set)	rue-100 Gap (%) ± std (%)	Method (training set)	clust-100 Gap (%) ± std (%)
POMO (rue-50)	0.6703 ± 0.0000	POMO (clust-50)	0.6829 ± 0.0000
POMO (rue-100)	0.1278 ± 0.0000	POMO (clust-100)	0.1405 ± 0.0000
DACT (rue-50)	27.5437 ± 31.4449	DACT (clust-50)	21.4630 ± 5.3292
DACT (rue-100)	0.6596 ± 0.5216	DACT (clust-100)	1.2220 ± 0.4773

- when applying the solvers learned by POMO and DACT on the testing instances having larger sizes than the training instances, the performance would be significantly degraded

Materials on NCO

- NCO is a rapidly evolving area that absorbs ideas from both DL and CO

[1] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. “Machine Learning for Combinatorial Optimization: A Methodological Tour d’horizon.” 2020

[2] Natalia Vesselinova, Rebecca Steinert, Daniel F. Perez-Ramirez, and Magnus Boman. “Learning Combinatorial Optimization on Graphs: A Survey With Applications to Networking. ” 2020

[3] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. “Reinforcement Learning for Combinatorial Optimization: A survey.” 2021

[4] Quentin Cappart, Didier Chetelat, Elias B. Khalil, Andrea Lodi, Christopher Morris, Petar Velickovic, “Combinatorial Optimization and Reasoning with Graph Neural Networks.” 2021

[5] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck and Bryan Wilder, “End-to-End Constrained Optimization Learning: A Survey.” 2021

[6] Jiayi Zhang, Chang Liu, Junchi Yan, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, “A Survey for Solving Mixed Integer Programming via Machine Learning.” 2022

Outline

- Introduction to Learning to Optimize (L2O)
- Research Directions in L2O
  - Automatic Algorithm Selection
  - Automatic Algorithm Configuration
  - Neural Combinatorial Optimization
- **Summary**

Summary

- L2O represents a shift in algorithm/solver design paradigm, from human-centered paradigm to learning centered
- L2O is able to improve traditional solvers in both effectiveness and efficiency
- L2O can be implemented in many different ways
  - Learning a selector that always selects the best algorithm
  - Learning an algorithm configuration (or a set of configurations) with strong performance
  - Learning a constructive/improvement heuristic

Summary

- L2O has been successfully applied to may problems over the past years

AAS	SAT, MIP, PLANNING, VRP, CSP, QBF, BBO,GAMES…
AAC	SAT, SMT, ASP, PLANNING, MIP, VRP, Protein Folding…
NCO	VRP, BPP, OP, EDA, MIP, IM, JSSP, QAP, MCS, SAT…

- Future Directions
  - ➢ Effective combination of different L2O frameworks
  - ➢ Convenient integration of domain knowledge into L2O
  - ➢ More complex real-word problems with no effective solutions

Thanks!
Comments/Questions are most welcome!

References

[1] Huberman, Bernardo A., Rajan M. Lukose, and Tad Hogg. "An economics approach to hard computational problems." Science 275.5296 (1997): 51-54.

[2] Loreggia, Andrea, et al. "Deep learning for algorithm portfolios." Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[3] Kerschke, Pascal, et al. "Leveraging TSP solver complementarity through machine learning." Evolutionary computation 26.4 (2018): 597-620.

[4] Zhao, Kangfei, et al. "Towards Feature-free TSP Solver Selection: A Deep Learning Approach." 2021 IJCNN. IEEE, 2021.

[5] Kotthoff, Lars. "Algorithm selection for combinatorial search problems: A survey." Data mining and constraint programming. Springer, Cham, 2016. 149-190.

[6] Adenso-Diaz, Belarmino, and Manuel Laguna. "Fine-tuning of algorithms using fractional experimental designs and local search." Operations research 54, no. 1 (2006): 99-114.

[7] Coy, Steven P., Bruce L. Golden, George C. Runger, and Edward A. Wasil. "Using experimental design to find effective parameter settings for heuristics." Journal of Heuristics 7, no. 1 (2001): 77-97.

[8] Hutter, Frank, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. "ParamILS: an automatic algorithm configuration framework." Journal of Artificial Intelligence Research 36 (2009): 267-306.

[9] Ansótegui, Carlos, Meinolf Sellmann, and Kevin Tierney. "A gender-based genetic algorithm for the automatic configuration of algorithms." In International Conference on Principles and Practice of Constraint Programming, pp. 142-157. Springer, Berlin, Heidelberg, 2009.

[10] Smit, Selmar K., and Agoston E. Eiben. "Beating the 'world champion'evolutionary algorithm via REVAC tuning." In Evolutionary Computation (CEC), 2010 IEEE Congress on, pp. 1-8. IEEE, 2010.

[11] Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration." In International Conference on Learning and Intelligent Optimization, pp. 507-523. Springer, Berlin, Heidelberg, 2011.

References

[12] Ansótegui, Carlos, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. "Model-Based Genetic Algorithms for Algorithm Configuration." In IJCAI, pp. 733- 739. 2015.

[13] Birattari, Mauro. The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective. Ph.D. Dissertation, Universit´e Libre de Bruxelles, Brussels, Belgium. 2004.

[14] Liu, Shengcai, et al. "On performance estimation in automatic algorithm configuration." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 03. 2020.

[15] Changwu Huang, Yuanxiang Li, and Xin Yao. A survey of automatic parameter tuning methods for metaheuristics. IEEE transactions on evolutionary computation, 24(2):201–216, 2019.

[16] Thomas St¨utzle and Manuel L´opez-Ib´a˜nez. Automated design of metaheuristic algorithms. In Michel Gendreau and Jean-Yves Potvin, editors, Handbook of Metaheuristics, volume 272 of International Series in Operations Research & Management Science, pages 541–579. Springer, 2019. doi: 10.1007/978-3-319-91086-4 17.

[17] Yasemin Eryolda¸s and Alptekin Durmu¸so˘glu. A literature survey on instance specific algorithm configuration methods. In Proceedings of the 11th Annual International Conference on Industrial Engineering and Operations Management, pages 2983–2990. IEOM Society International, 2021.

[18] Schede, Elias, et al. "A Survey of Methods for Automated Algorithm Configuration." arXiv preprint arXiv:2202.01651 (2022).

[19] Yasha Pushak and Holger H. Hoos. Golden parameter search: exploiting structure to quickly configure parameters in parallel. In Genetic and Evolutionary Computation Conference, GECCO, pages 245–253. ACM, 2020.

[20] Manuel L´opez-Ib´anez, J´er´emie Dubois-Lacoste, P´erez C´aceres Leslie, Mauro Birattari, and Thomas St¨utzle. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3:43–58, 2016.

References

[21] Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly. "Pointer networks." Advances in neural information processing systems 28 (2015).

[22] M. Seiler, J. Pohl, J. Bossek, P. Kerschke, and H. Trautmann. Deep Learning as a Competitive Feature-Free Approach for Automated Al- gorithm Selection on the Traveling Salesperson Problem. In Proc. of PPSN'2020.

[23] Kadioglu, Serdar, et al. "ISAC–instance-specific algorithm configuration." ECAI 2010. IOS Press, 2010. 751-756.

[24] Lindauer, Marius, et al. "Automatic construction of parallel portfolios via algorithm configuration." Artificial Intelligence 244 (2017): 272-290.

[25] Tang, Ke, et al. "Population-based algorithm portfolios with automated constituent algorithms selection." Information Sciences 279 (2014): 94-104.

[26] S. Liu, K. Tang and X. Yao, "Automatic Construction of Parallel Portfolios via Explicit Instance Grouping," AAAI 2019

[27] Kool, Wouter, Herke Van Hoof, and Max Welling. "Attention, learn to solve routing problems!." arXiv preprint arXiv:1803.08475 (2018).

[28] Kwon, Yeong-Dae, et al. "Pomo: Policy optimization with multiple optima for reinforcement learning." Advances in Neural Information Processing Systems 33 (2020): 21188-21198.

[29] Wu, Yaoxin, et al. "Learning improvement heuristics for solving routing problems.." IEEE transactions on neural networks and learning systems (2021).

[30] Ma, Yining, et al. "Learning to iteratively solve routing problems with dual-aspect collaborative transformer." Advances in Neural Information Processing Systems 34 (2021): 11096-11107.

[31] Xin, Liang, et al. "NeuroLKH: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem." Advances in Neural Information Processing Systems 34 (2021): 7472-7483.