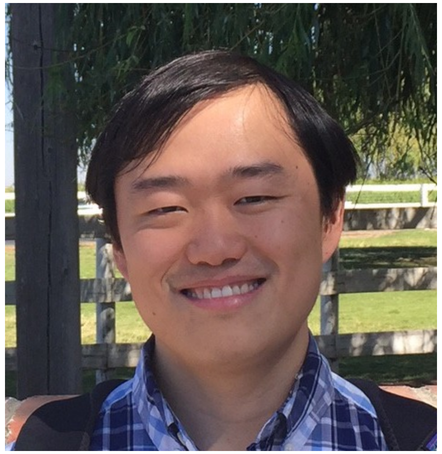
Machine Learning for Hard Optimization Problems in Computer System Design

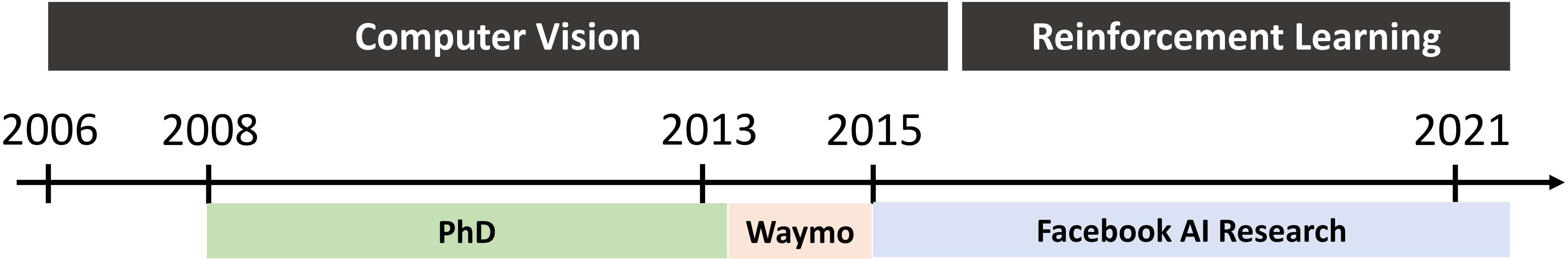
Yuandong Tian
Research Scientist and Manager

Facebook AI

facebook Artificial Intelligence

Career Path




Computer Vision
Reinforcement Learning
2006
2008
2013
2015
2021
PhD
Waymo
Facebook AI Research

Reinforcement Learning


Go


Chess


Shogi

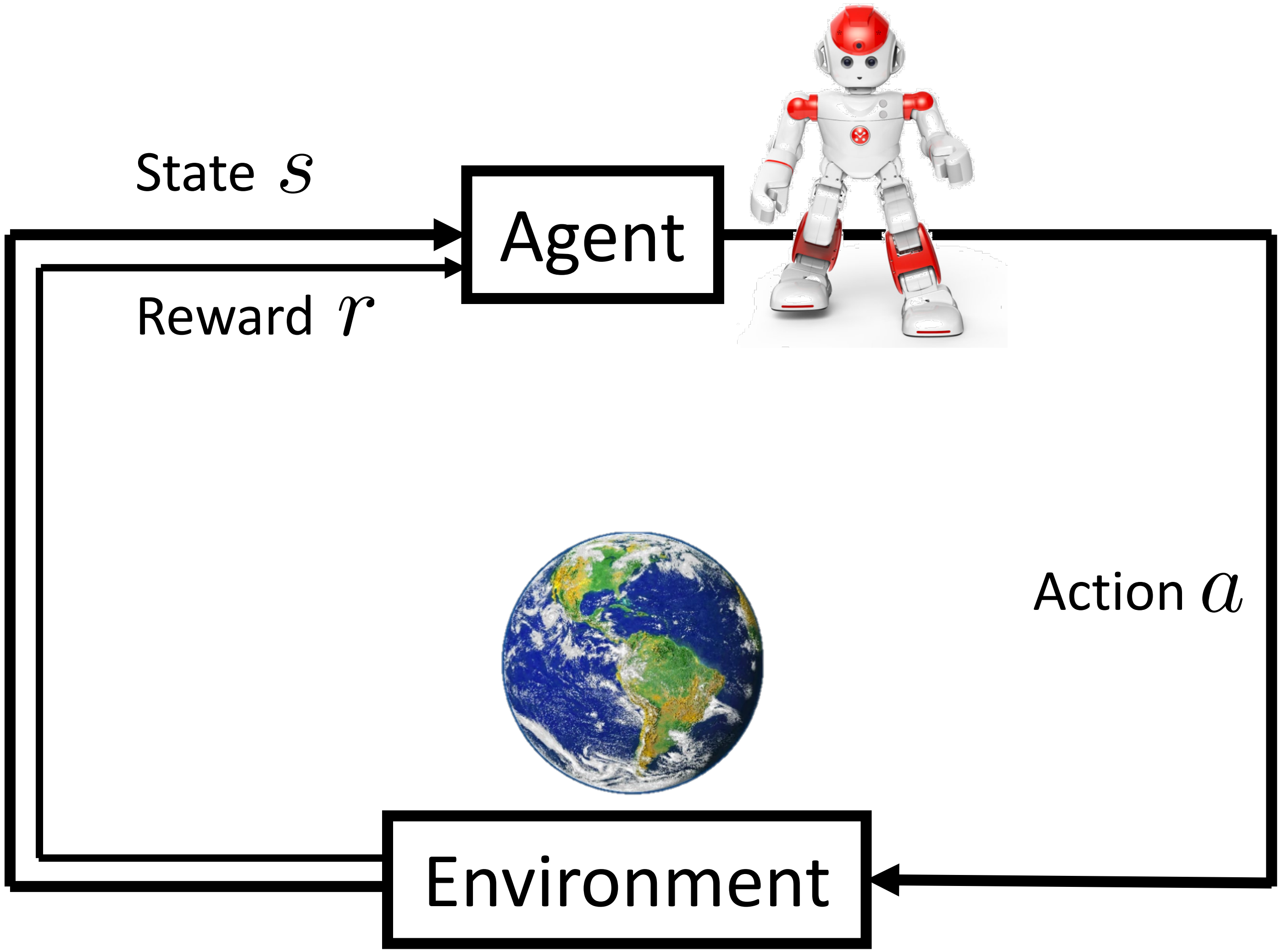

Poker


DoTA 2

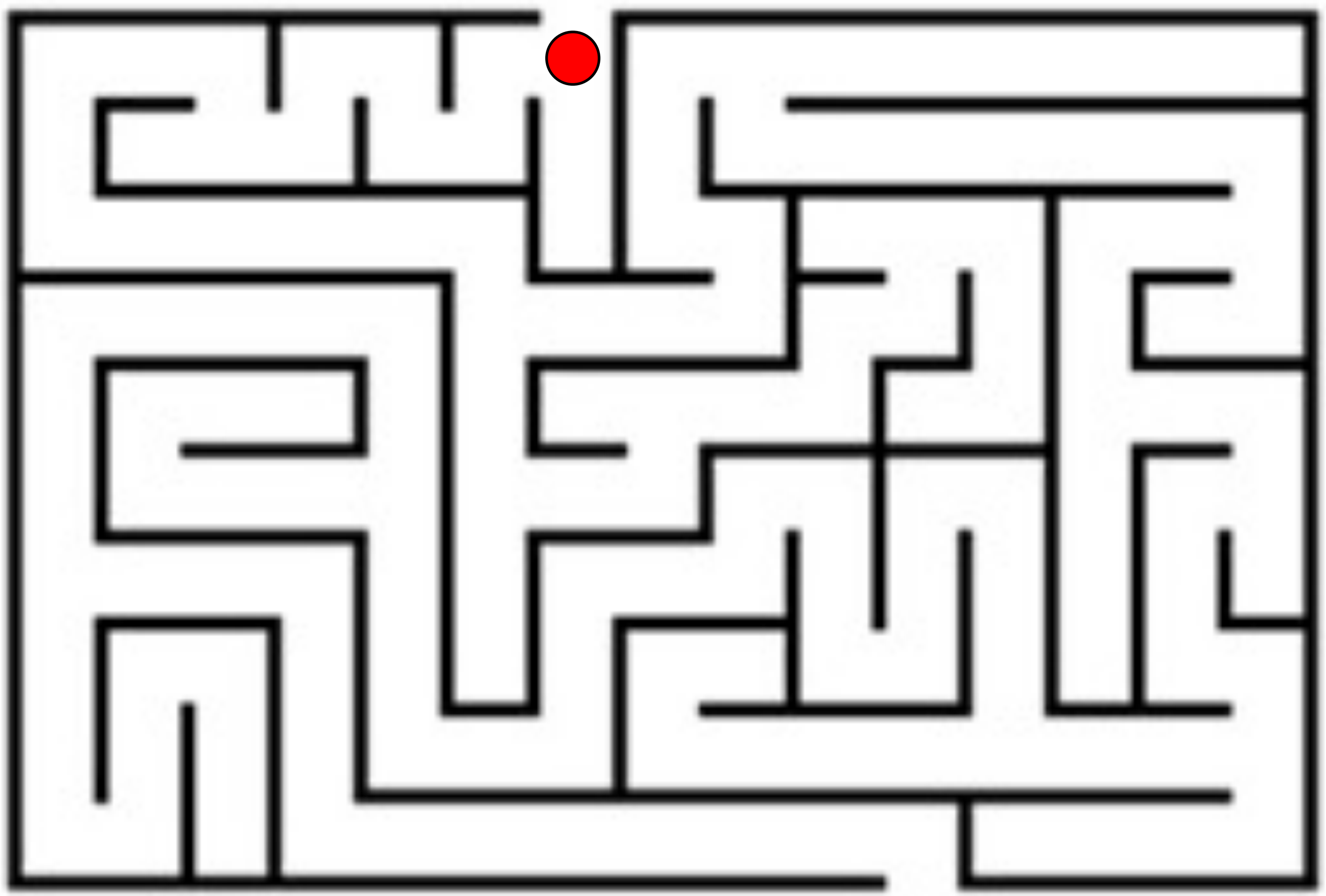

StarCraft II

Big Success in Games

What is Reinforcement Learning?



What is Reinforcement Learning?

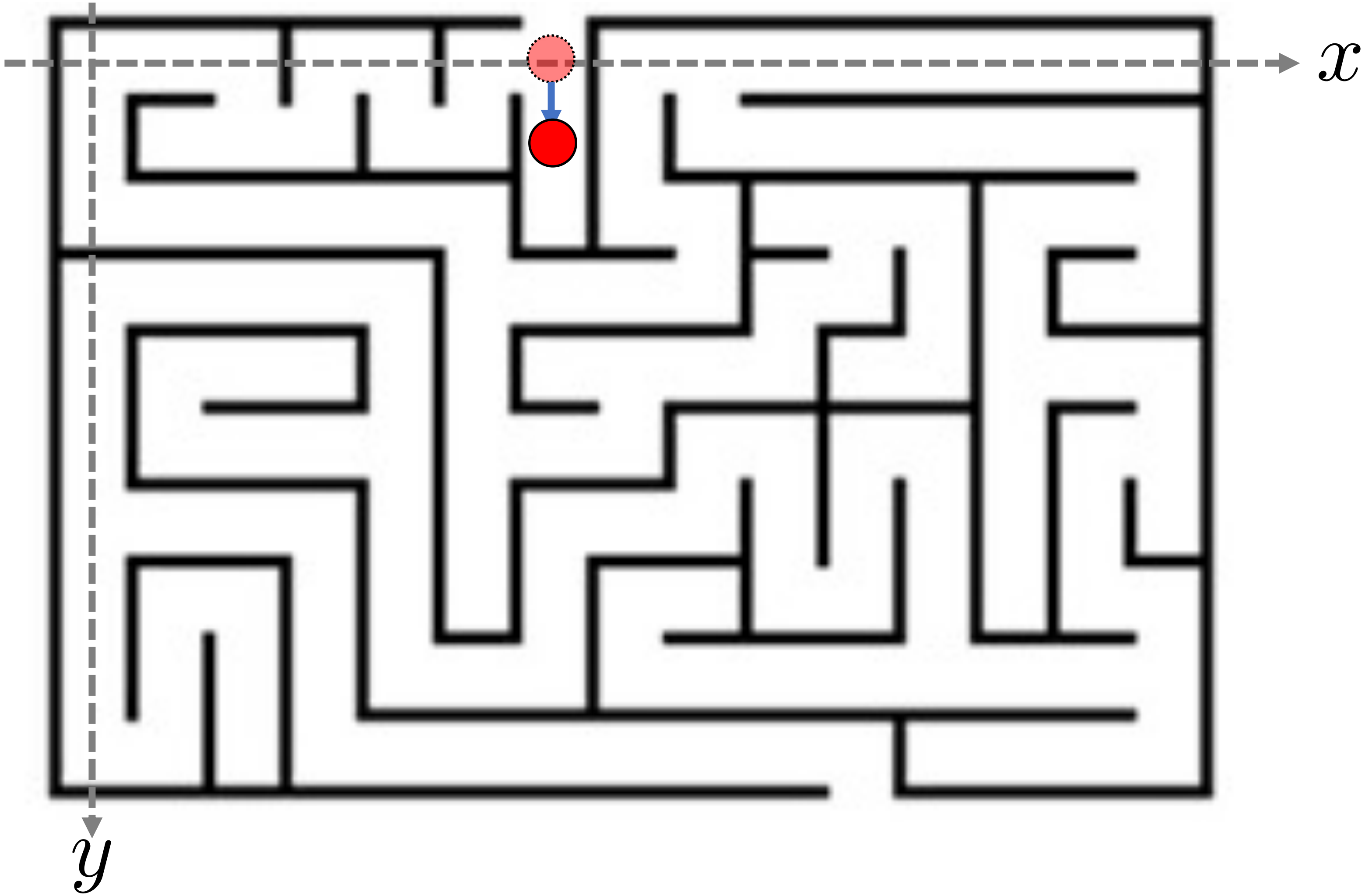


State:
where you are?

Action:
left/right/up/down

Next state:
where you are after the action?

What is Reinforcement Learning?



State:

$$s = (x, y) = (6, 0)$$

Actions:

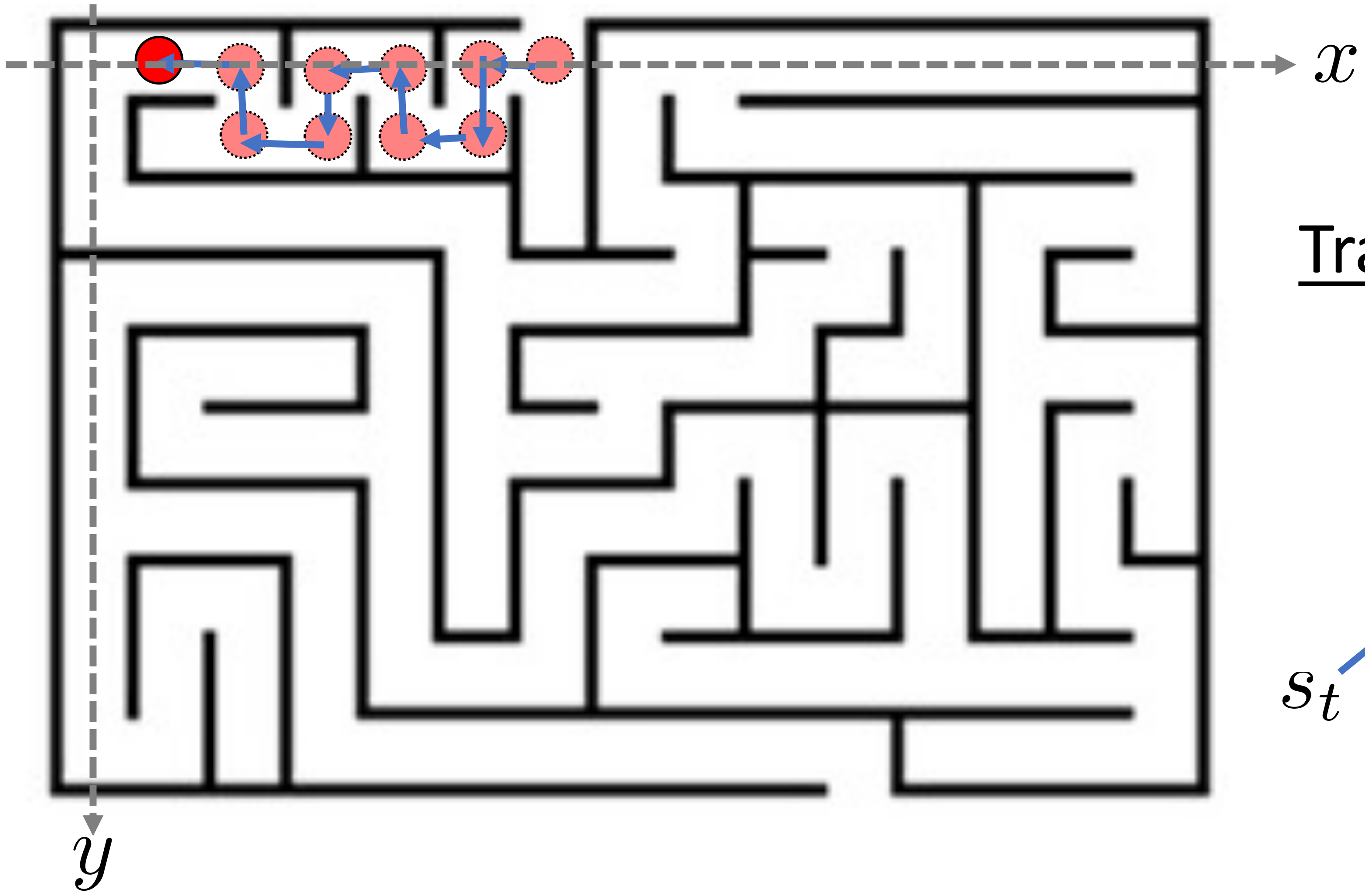
Left: $x \leftarrow x - 1$

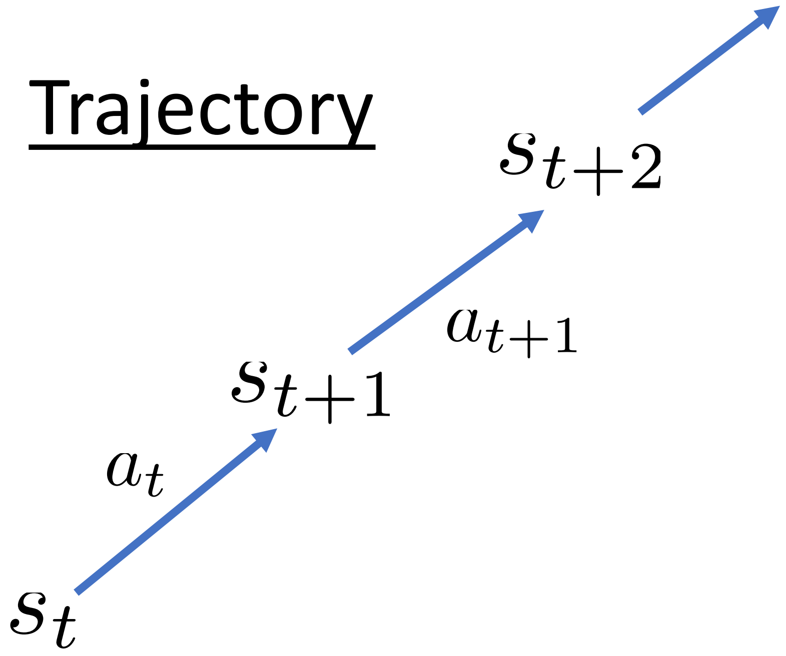
Right: $x \leftarrow x + 1$

Up: $y \leftarrow y - 1$

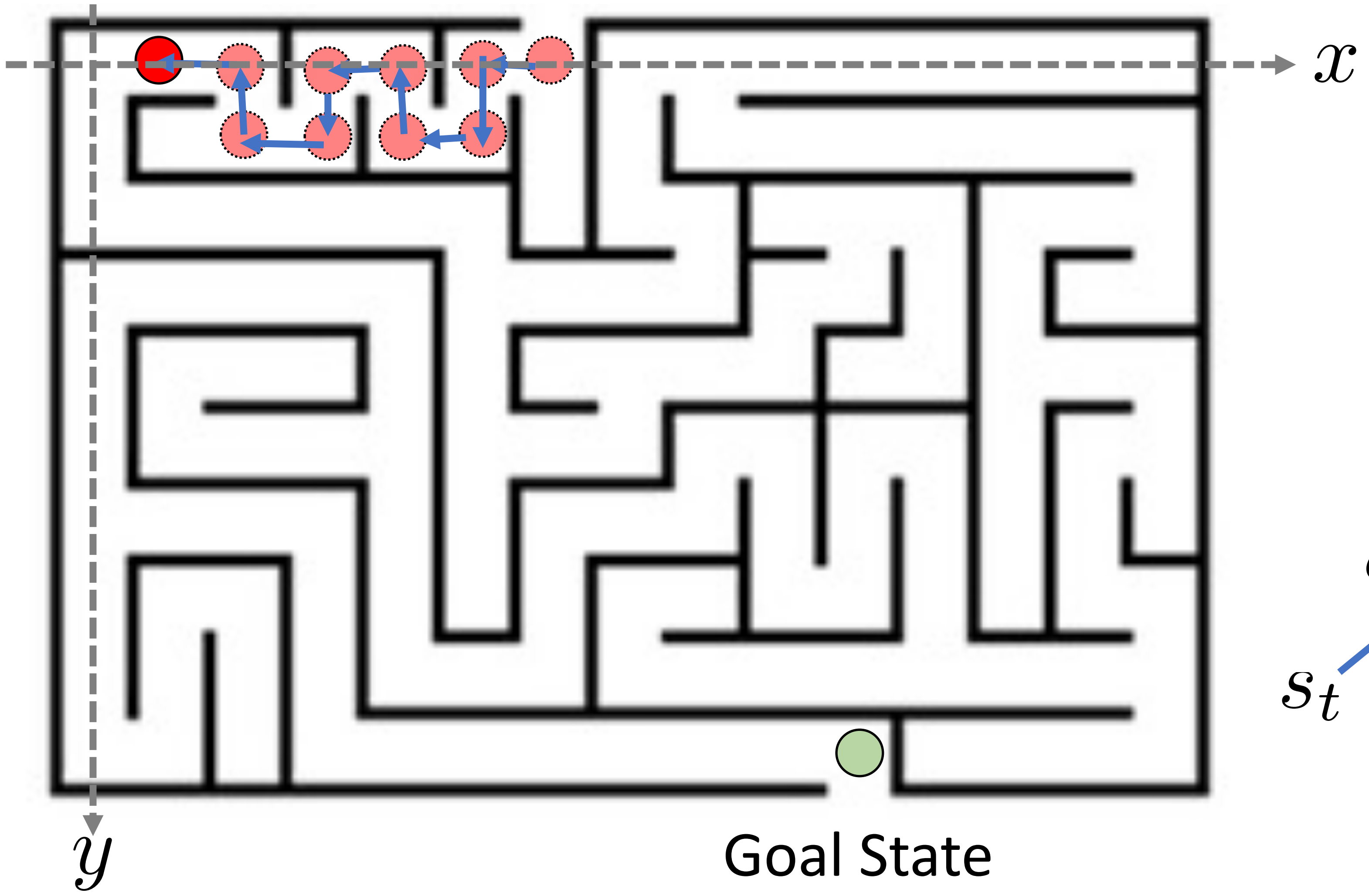
Down: $y \leftarrow y + 1$

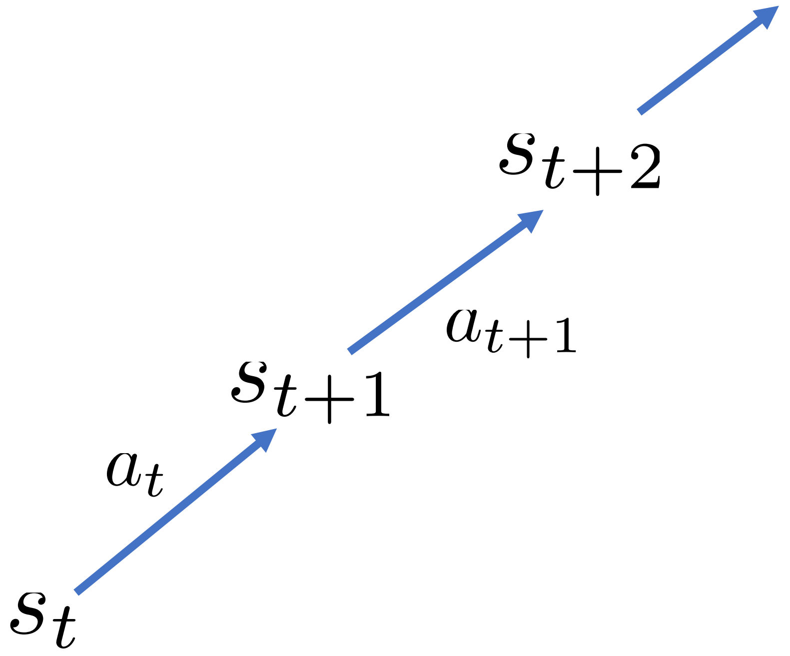
What is Reinforcement Learning?



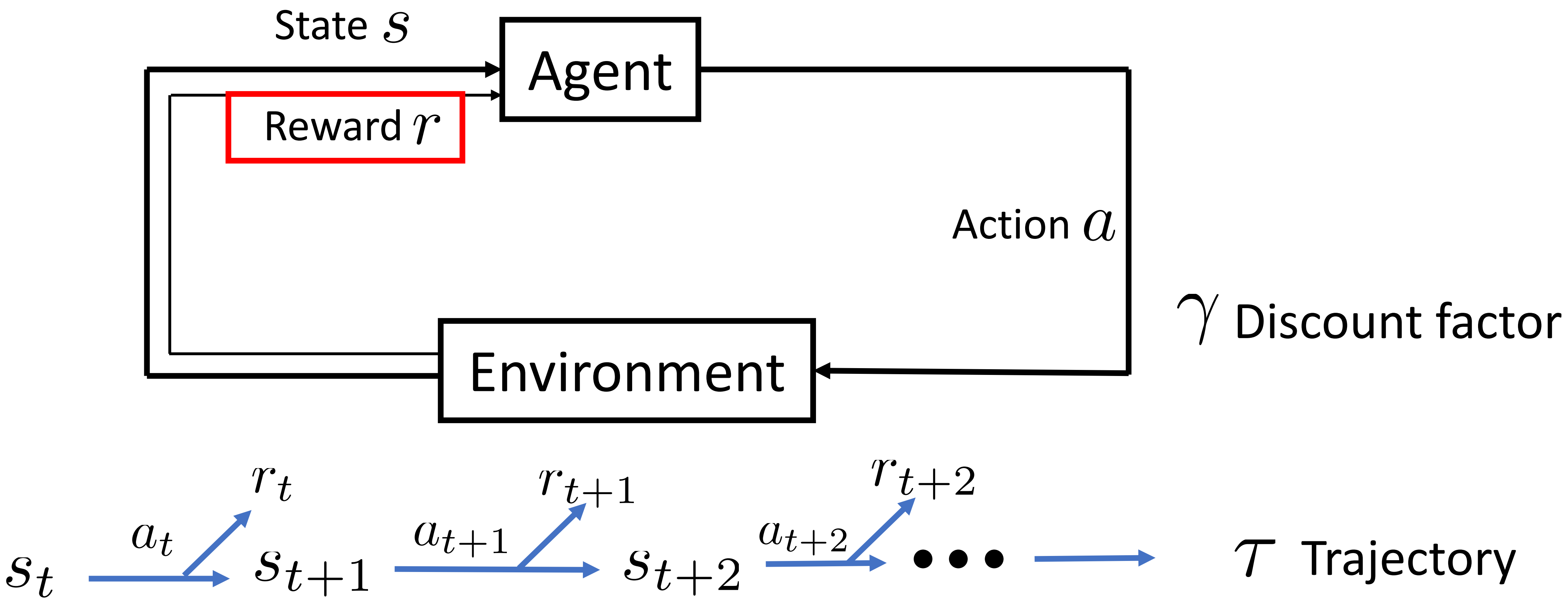


Goal of Reinforcement Learning



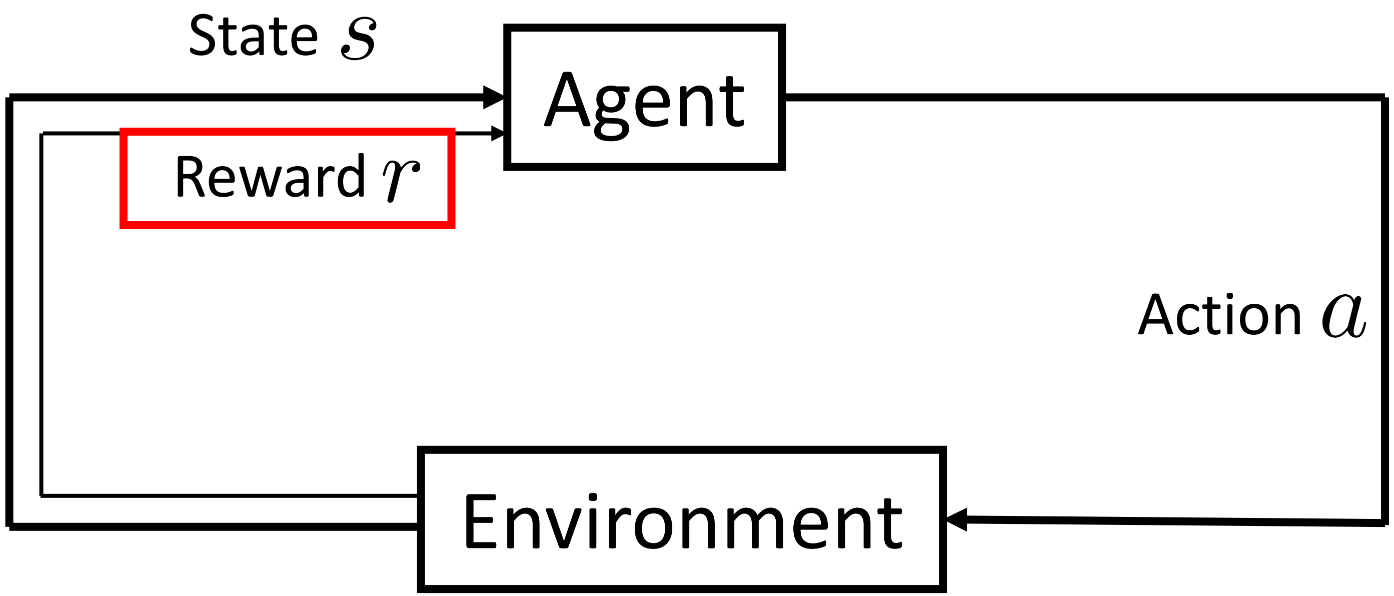


Goal of Reinforcement Learning



Maximize long-term reward: $\max \sum_{t'=t}^{+\infty} \gamma^{t'-t} r_t$

Key Quantities

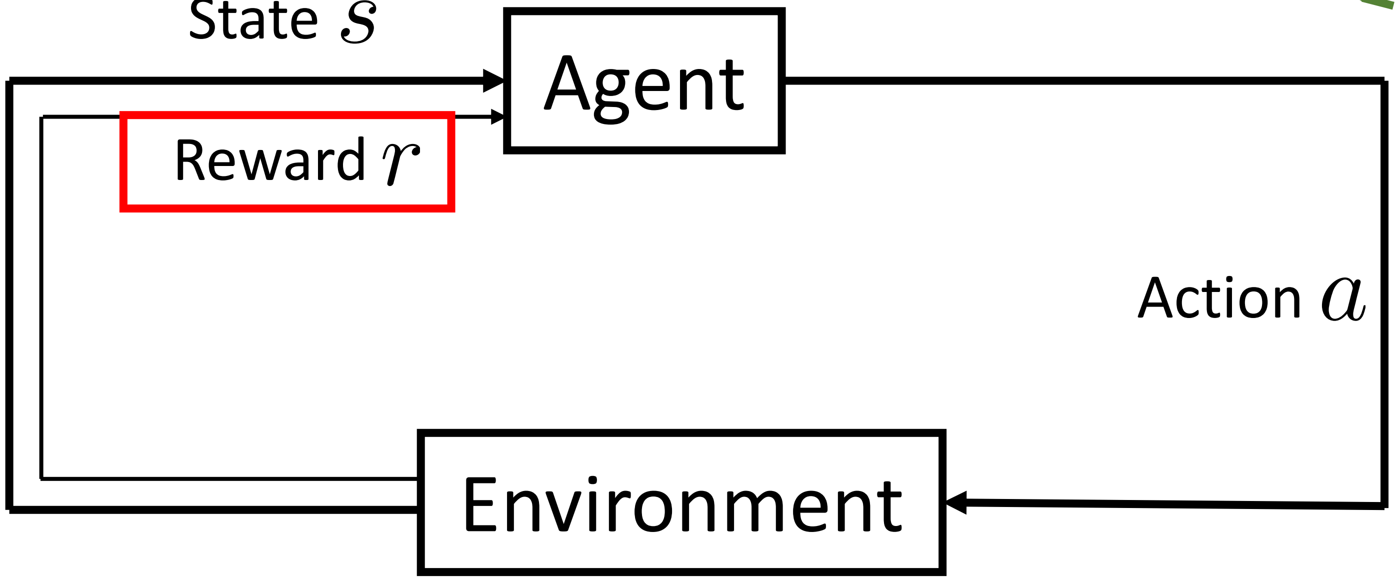


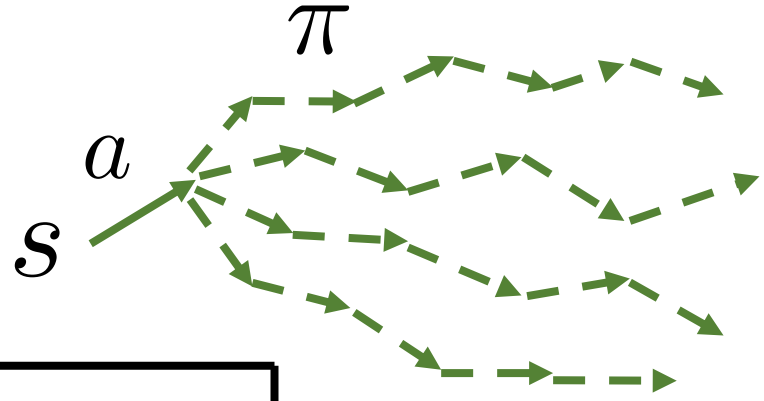
$V^*(s)$ Maximal reward you can get starting from state s

$Q^*(s, a)$ Maximal reward starting from s after taking action a

$\pi(a|s)$ Probability of taking action a given state s

Key Quantities





$V^{\pi}(s)$ Reward you can get, starting from s following policy π

$Q^{\pi}(s, a)$ Reward starting from s after taking action a and following π

Bellman Equations

$$\boxed{Q^*(s,a)} = r(s,a) + \gamma \max_{a'} \boxed{Q^*}(s'(s,a), a')$$

$$\boxed{V^*(s)} = \max_a r(s,a) + \gamma \boxed{V^*}(s'(s,a))$$

Optimal solution

Algorithm

Iteratively table filling

Tabular Q-learning

$$Q^{(n)}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q^{(n-1)}(s'(s,a),a')$$

Value Iteration

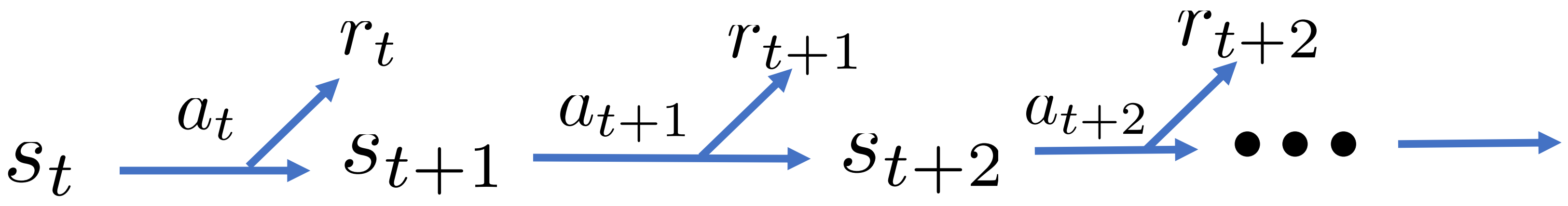
$$V^{(n)}(s) \leftarrow \max_{a} r(s,a) + \gamma V^{(n-1)}(s'(s,a))$$

As long as we can enumerate ***all possible*** states and actions

On trajectories

Q-learning

$$Q^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q^{(n-1)}(s_{t+1}, a')$$



On trajectories

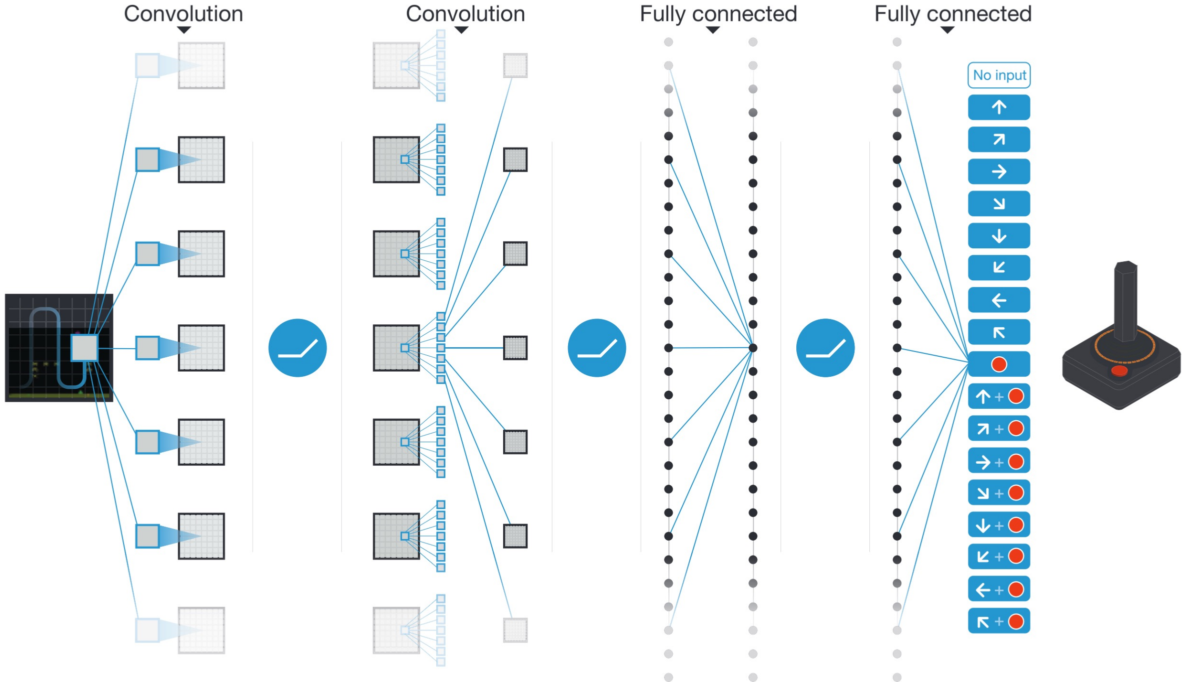
Parametric function

Q-learning

$$Q_\theta(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$$

$Q_\theta(s, a)$ now have generalization capability

How could you take the gradient w.r.t θ ?
Note that θ appears on both sides.



On trajectories

Q-learning

$$Q_\theta(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} \boxed{Q_{\boxed{\theta'}}(s_{t+1}, a')}$$

Old fixed parameters

Target network

Fixing RHS and learn θ from LHS.

On trajectories

Q-learning (make the target even smoother)

Smoothing factor

$$Q_\theta(s_t, a_t) \leftarrow (1-\alpha) Q_\theta(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') \right]$$

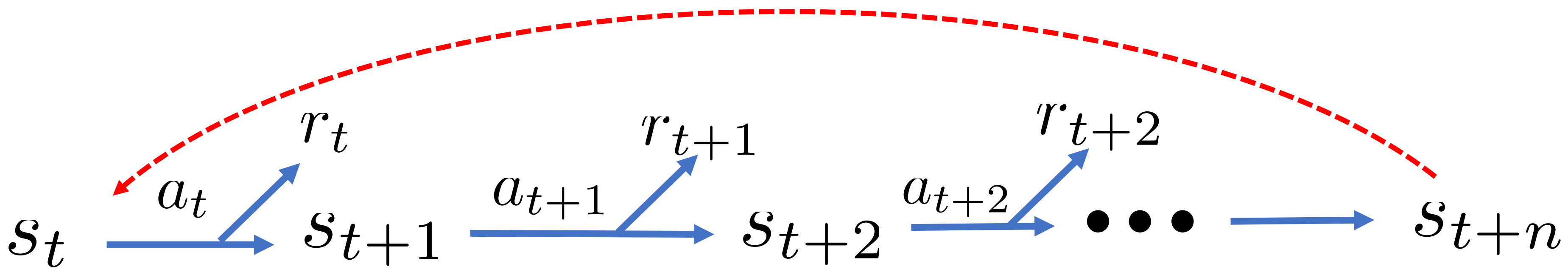
$$\Delta Q_\theta(s_t, a_t) \propto \underbrace{r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') - Q_\theta(s_t, a_t)}_{\text{Temporal Difference (TD) Error } \delta}$$

[*Mnih et al.* Human-level control through deep reinforcement learning, *Nature 2015*]

On trajectories

Multi-step Q-learning

$$\Delta Q_\theta(s_t, a_t) \propto \underbrace{\sum_{i=0}^{n-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^n \max_{a'} Q_\theta(s_{t+n}, a')}_{\text{n-step rollout}} - Q_\theta(s_t, a_t)$$



Trajectories from **replay buffer**

Sample trajectories

Q-learning

$$Q_\theta(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$$

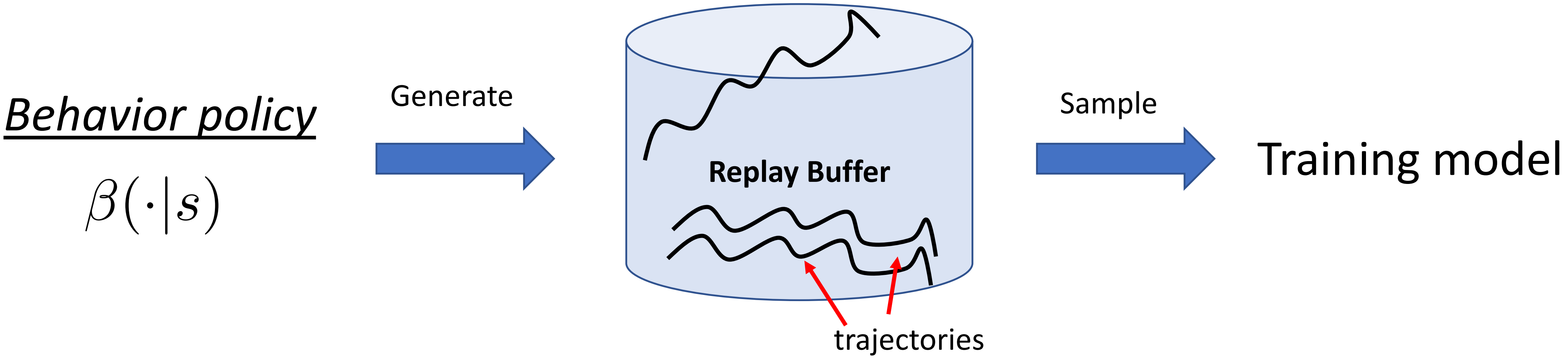
How could we sample a trajectory in the state space?

Replay Buffer

Dynamic “dataset”:

takes experience as input

provide data for training



On-policy versus Off-policy approaches

Off-policy, sampled by some behavior policy $\beta(\cdot|s)$

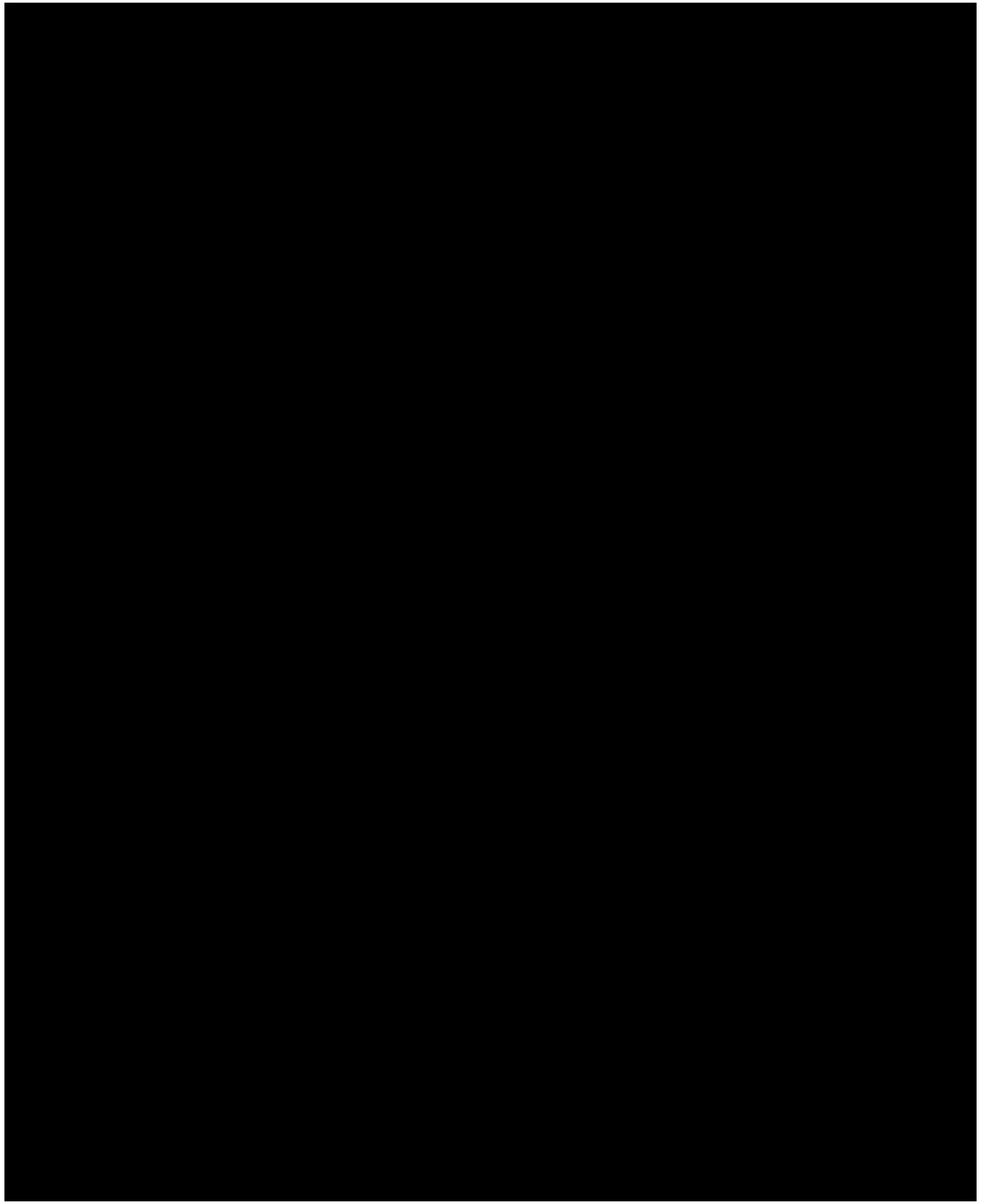
Expert behaviors (imitation learning)

Supervised learning

On-policy, sampled by the current models $Q(s,a)$ $\pi(\cdot|s)$

Agent not only learns from the data, but also chooses which data to learn.

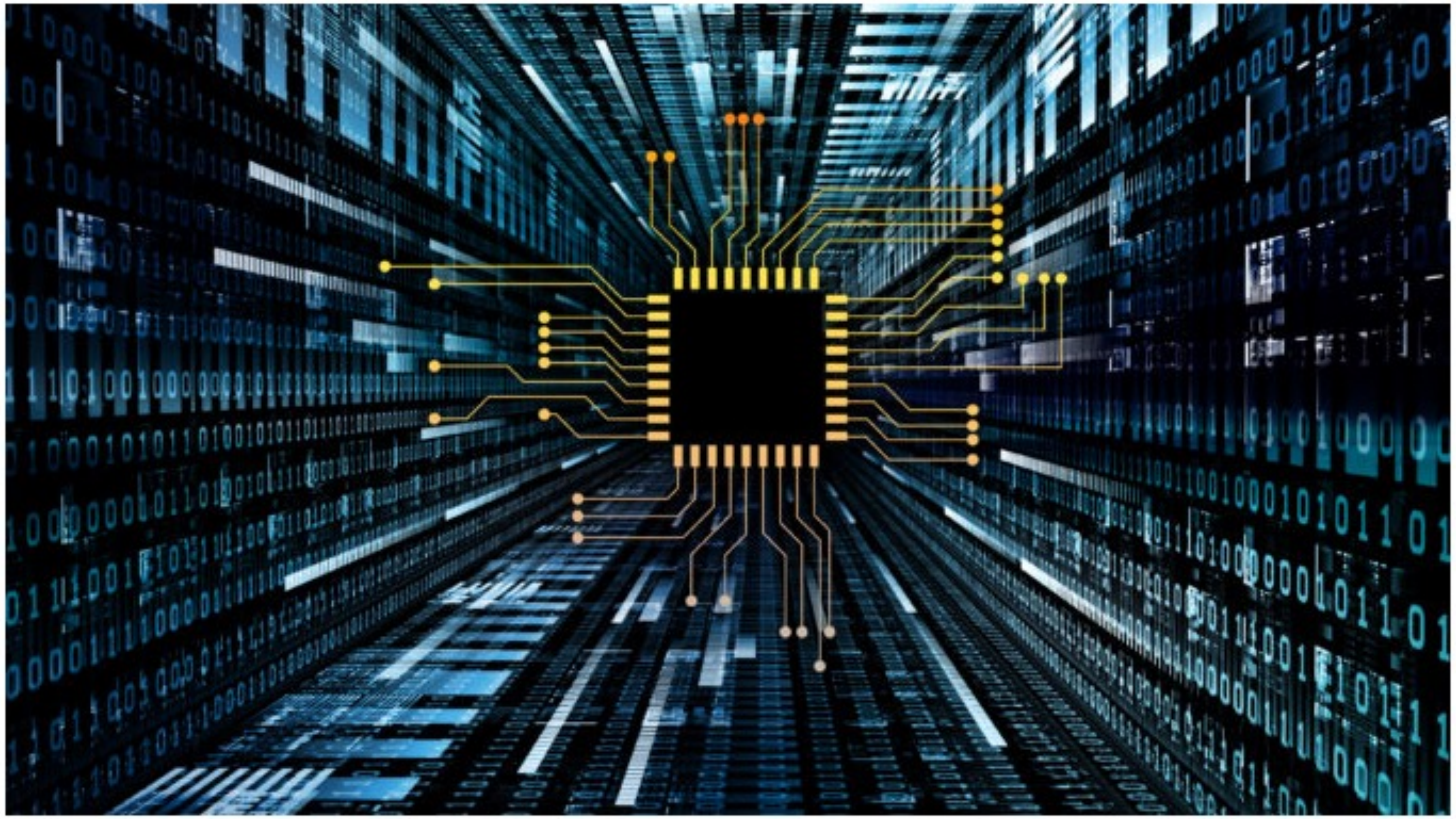
Deep Q-Learning



What's Beyond Games?

GOOGLE TEACHES AI TO PLAY THE GAME OF CHIP DESIGN

February 20, 2020 Timothy Prickett Morgan



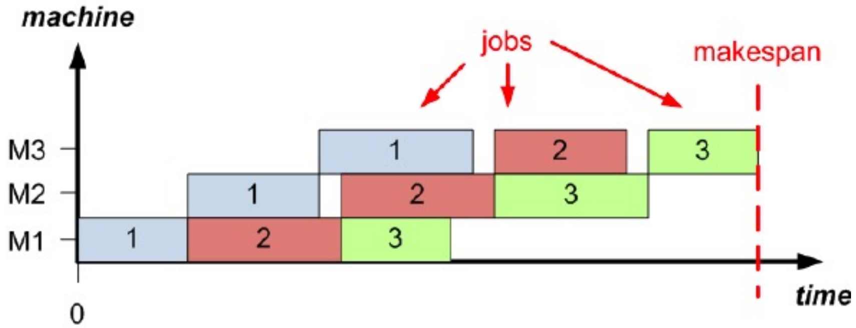
Several weeks with **human experts** in the loop

→

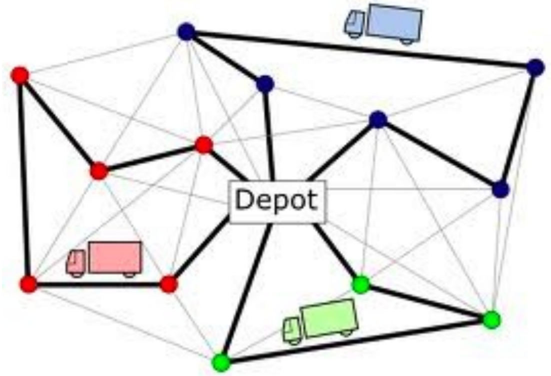
Fully automatic design in 6 hours

Optimization Problems

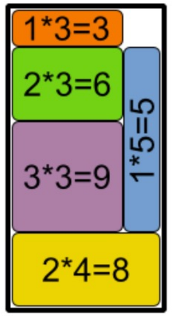
Travel Salesman Problem



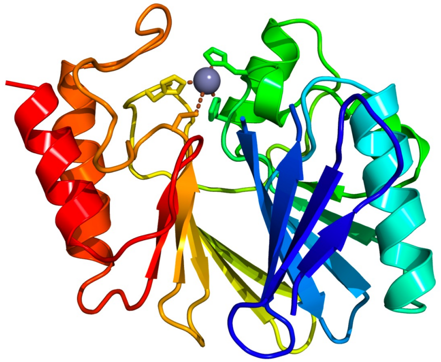
Job Scheduling



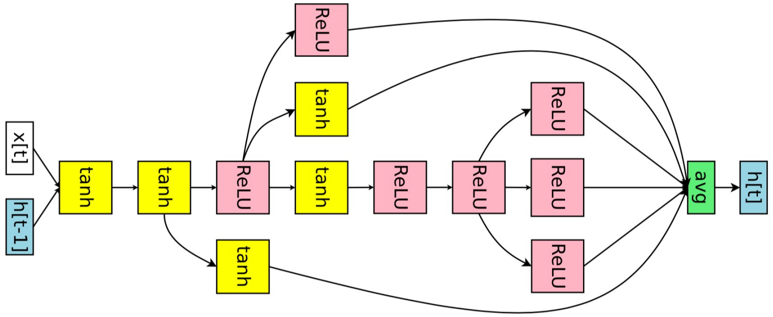
Vehicle Routing



Bin Packing



Protein Folding



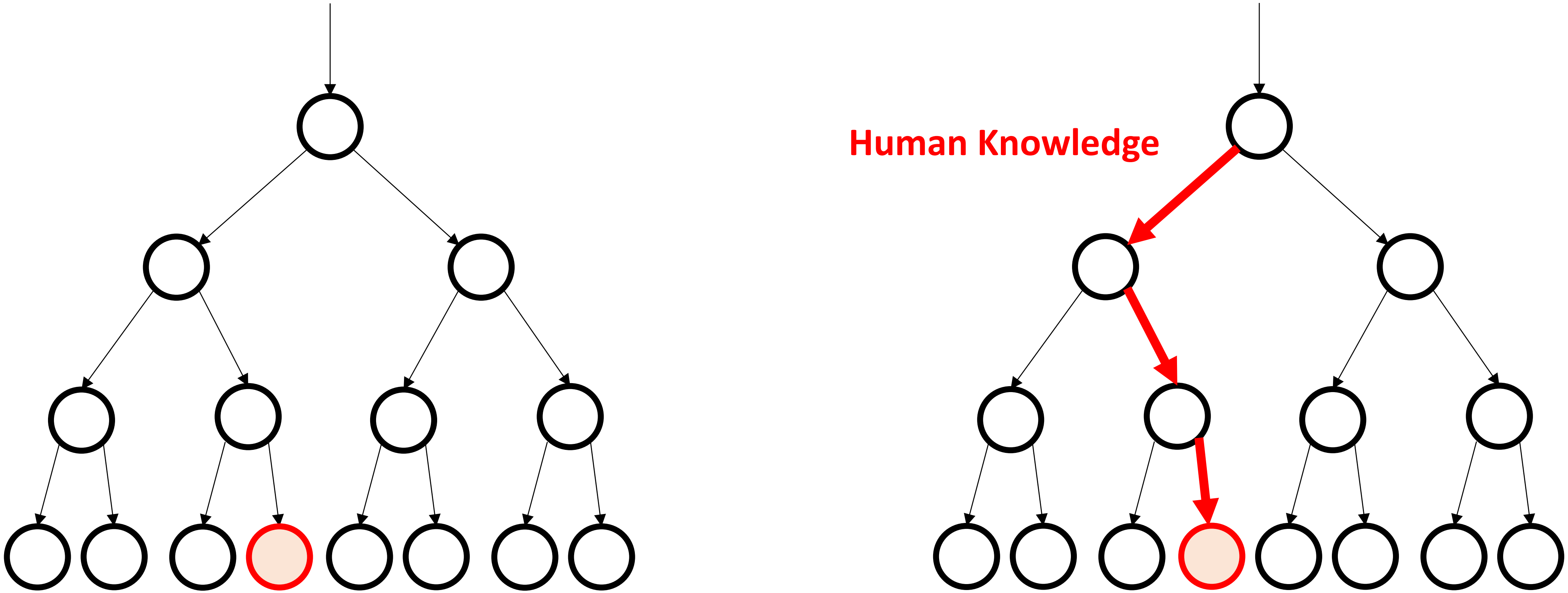
Model-Search

$$x^* = \arg\max_{x \in \Omega} f(x)$$

Wait...What?

- Many problems are NP-hard problems.
  - No good algorithm unless $P = NP$
- These guarantees are worst-case ones.
  - To prove a lower-bound, construct an adversarial example to fail the algorithm
- For specific distribution, there might be better heuristics.
  - Human heuristics are good but may not be suitable for everything

More Efficient Search for Optimization



Exhaustive search to get **a good solution**

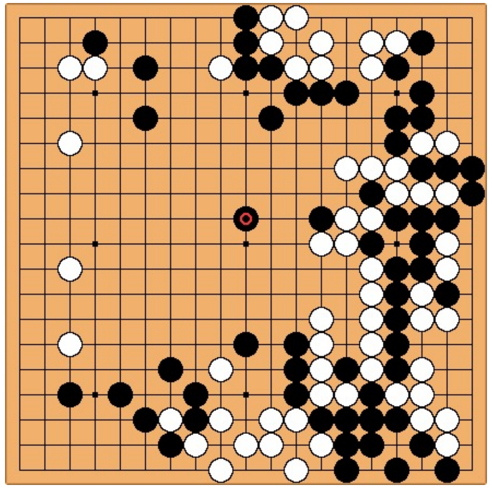
More Efficient Search for Optimization

Can we use Machine Learning?

Exhaustive search to get **a good solution**

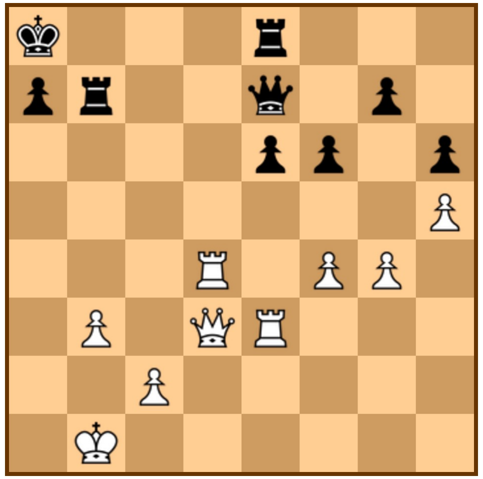
Efficient Search for Games

Go



DarkForest (2015)
AlphaGo (2016)
AlphaZero (2017)
OpenGo (2018)

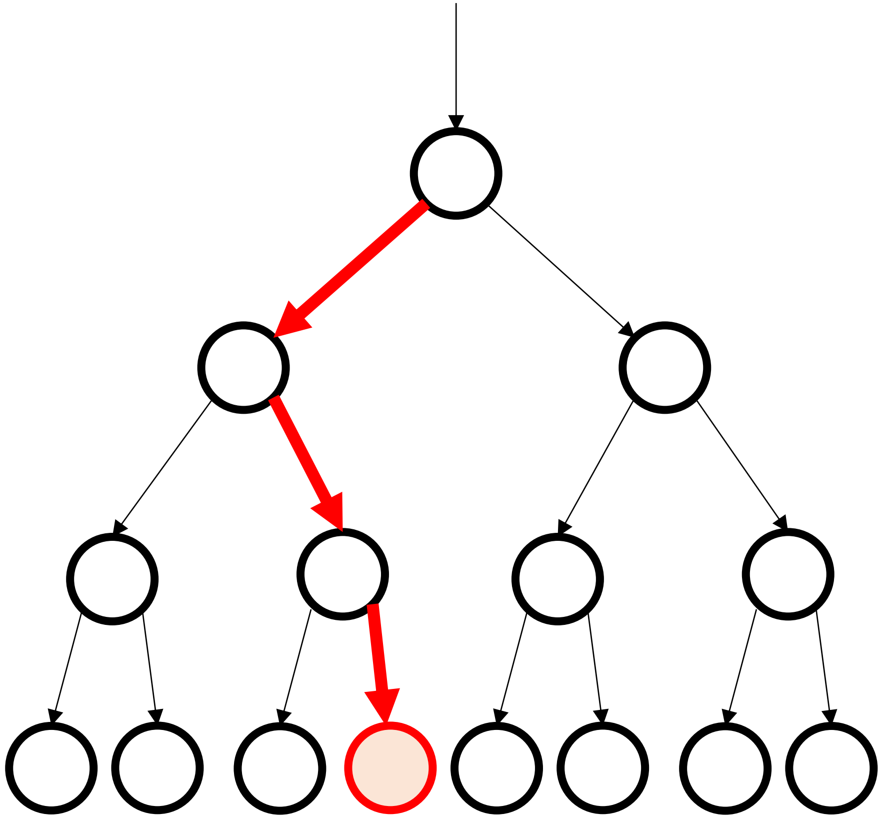
Chess



Deep Blue (2002)
AlphaZero (2017)

Shogi

AlphaZero (2017)



~~Human Knowledge~~

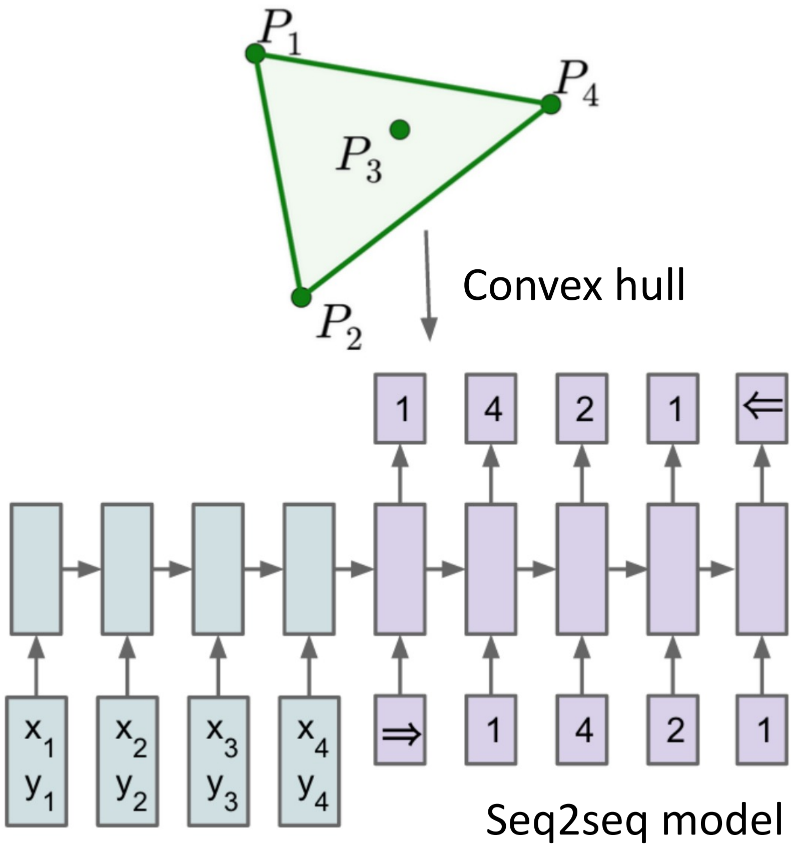
Machine learned models

Optimization → Reinforcement Learning

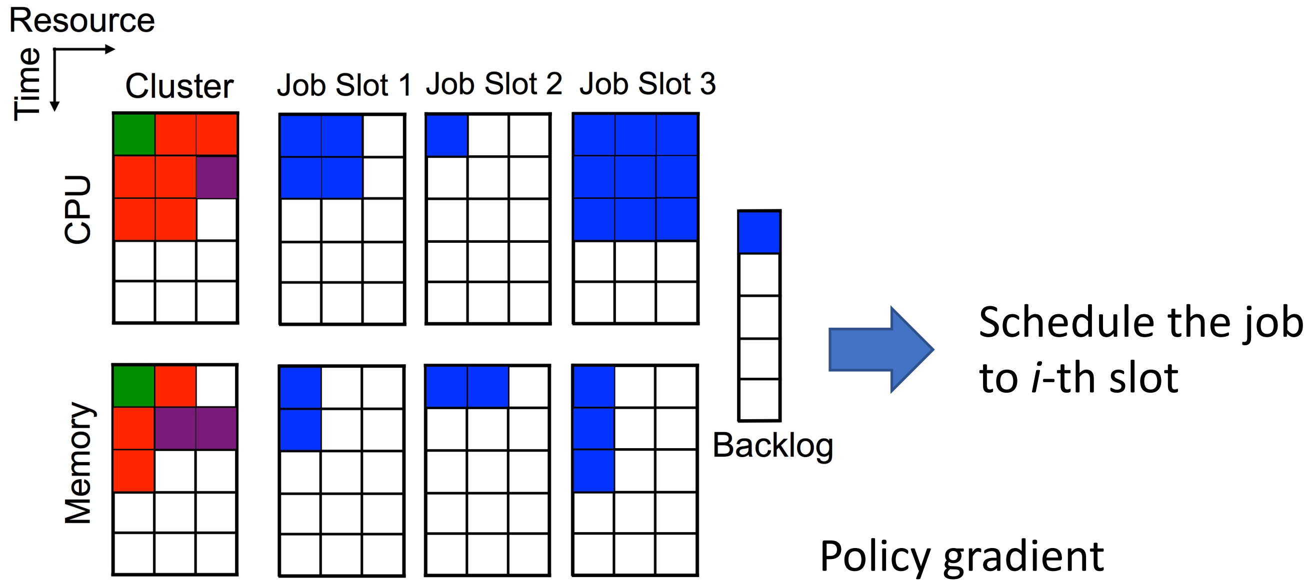
Name	Ways of Parameterization
One-shot Prediction	Spec → Solution
Progressive Prediction	Spec → SolPart1 → SolPart2 → SolPart3
Iterative Refinement	Spec → Sol1 → Sol2 (improved) → Sol3 (Better Improved)
Learned Action Space	Spec → All solution space → Small solution space → …

Representation Matters!

Direct predicting solutions

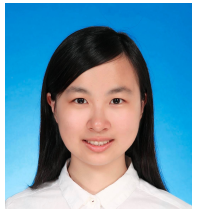


[O. Vinyals. et al, Pointer Networks, NIPS 2015]

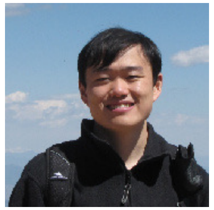


[H. Mao et al, Resource Management with Deep Reinforcement Learning, ACM Workshop on Hot Topics in Networks, 2016]

Local Rewriting Framework

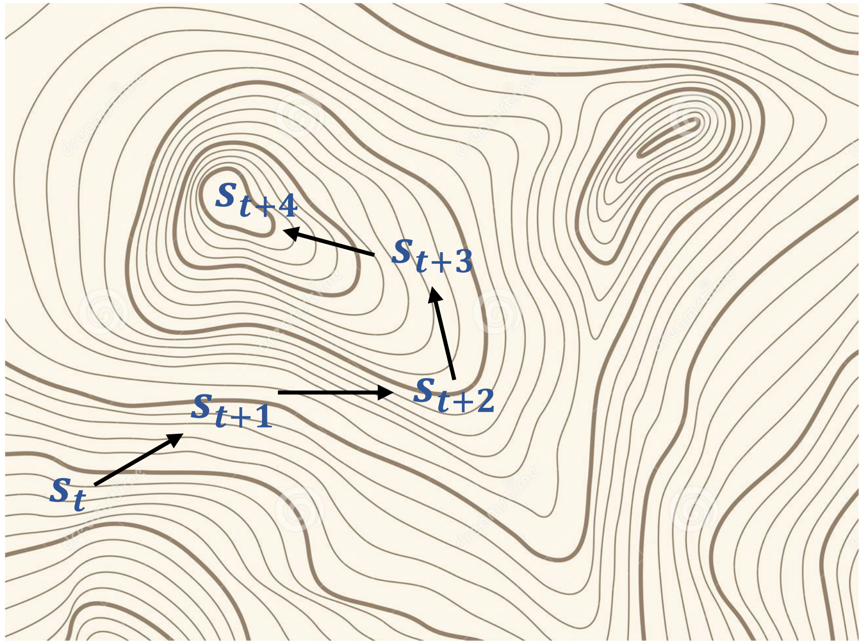


Xinyun Chen

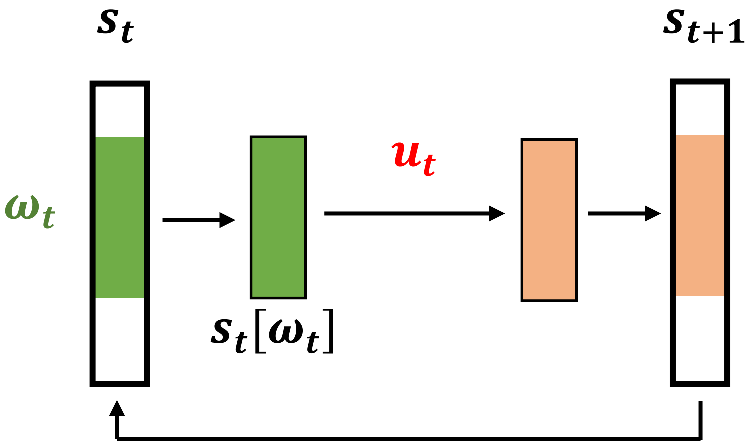


Yuandong Tian

[X. Chen and Y. Tian, ***Learning to Perform Local Rewriting for Combinatorial Optimization***, NeurIPS 2019]



Start from a feasible solution and iteratively converges to a good solution



Current State (i.e. Solution) | Region-Picker | Rule-Picker

$$s_t \longrightarrow \omega_t \sim \pi_\omega(\cdot\,|s_t) \longrightarrow u_t \sim \pi_u(\cdot\,|s_t\,[\omega_t])$$

$$s_{t+1} = f(s_t, \omega_t, u_t)$$

Q-Actor-Critic Training

How to train two policies $\boldsymbol{\pi_\omega}(\cdot\,|\boldsymbol{s_t})$ and $\boldsymbol{\pi_u}(\cdot\,|\boldsymbol{s_t}\,[\boldsymbol{\omega_t}])$?

Learn Q to fit cumulative rewards:

$$L_\omega(\theta) = \frac{1}{T}\sum_{t=0}^{T-1}\Big(\sum_{t'=t}^{T-1}\gamma^{t'-t} r(s'_t, (\omega'_t, u'_t)) - Q(s_t, \omega_t; \theta)\Big)^2$$

$\boldsymbol{\pi_\omega}(\cdot\,|\boldsymbol{s_t})$: Q-learning with soft policy:

$$\pi_\omega(\omega_t|s_t;\theta) = \frac{\exp(Q(s_t,\omega_t;\theta))}{\sum_{\omega_t}\exp(Q(s_t,\omega_t;\theta))}$$

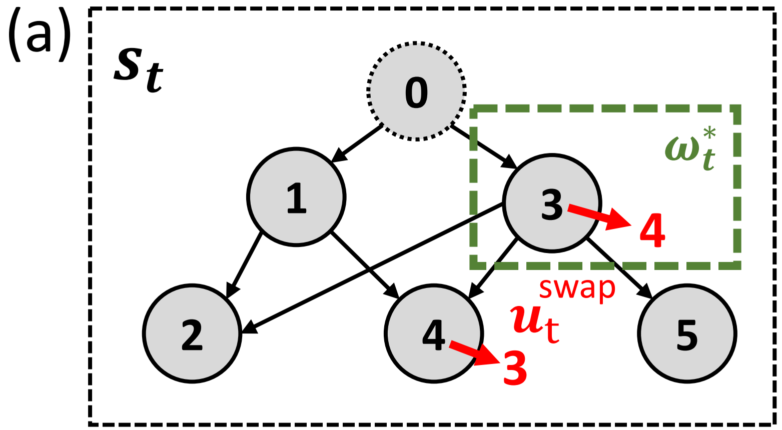
$\boldsymbol{\pi_u}(\cdot\,|\boldsymbol{s_t}\,[\boldsymbol{\omega_t}])$: Actor-Critic with learned Q:

$$L_u(\phi) = -\sum_{t=0}^{T-1}\Delta(s_t,(\omega_t,u_t))\log\pi_u(u_t|s_t[\omega_t];\phi)$$

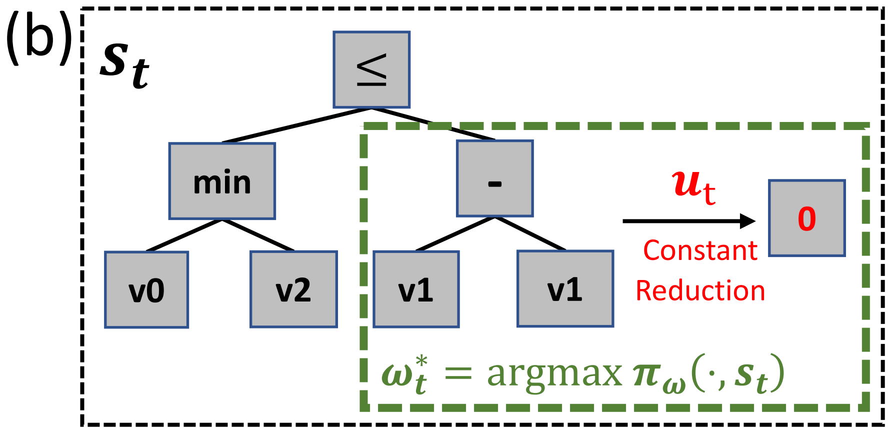
<u>Advantage:</u>

$$\Delta(s_t,(\omega_t,u_t)) \equiv \sum_{t=t'}^{T-1}\gamma^{t'-t} r(s'_t,(\omega'_t,u'_t)) - Q(s_t,\omega_t;\theta)$$

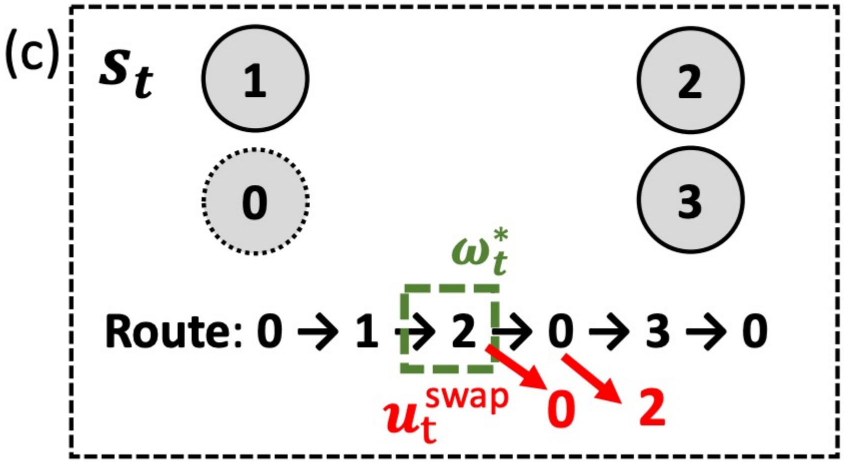
Different Action Spaces for Different Applications



Online Job Scheduling

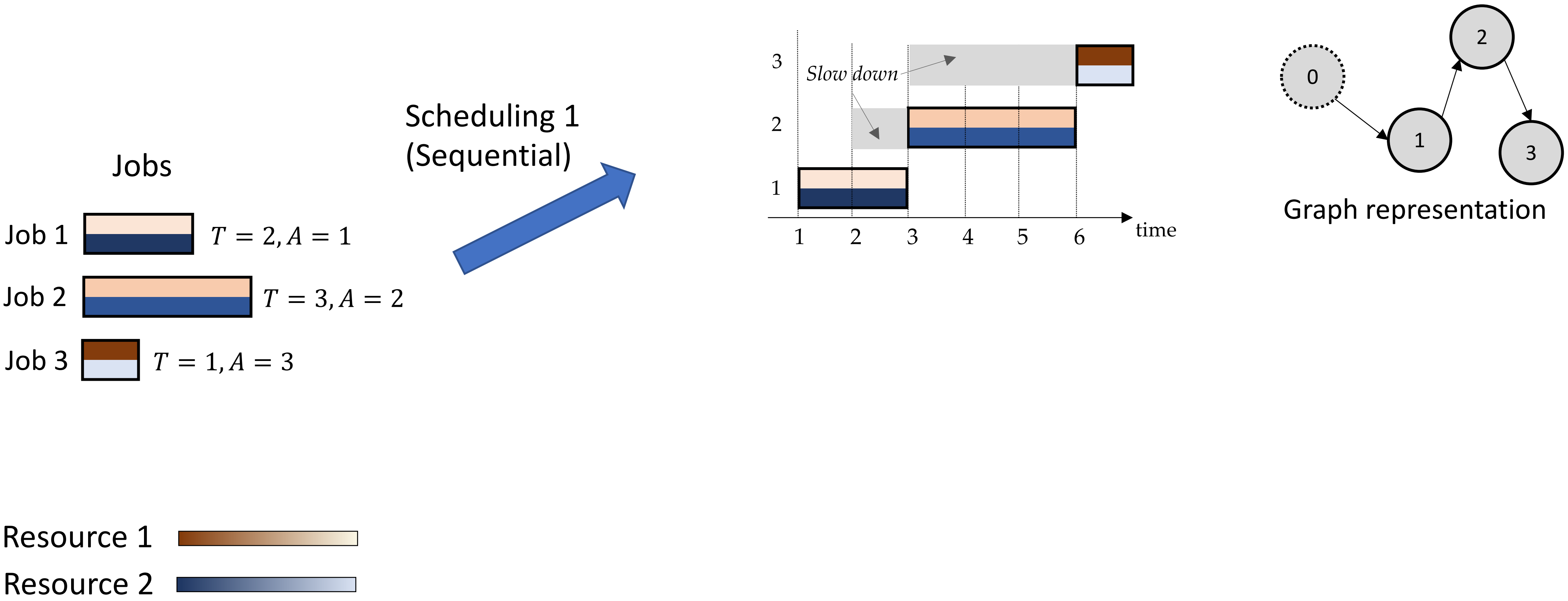


Expression Simplification

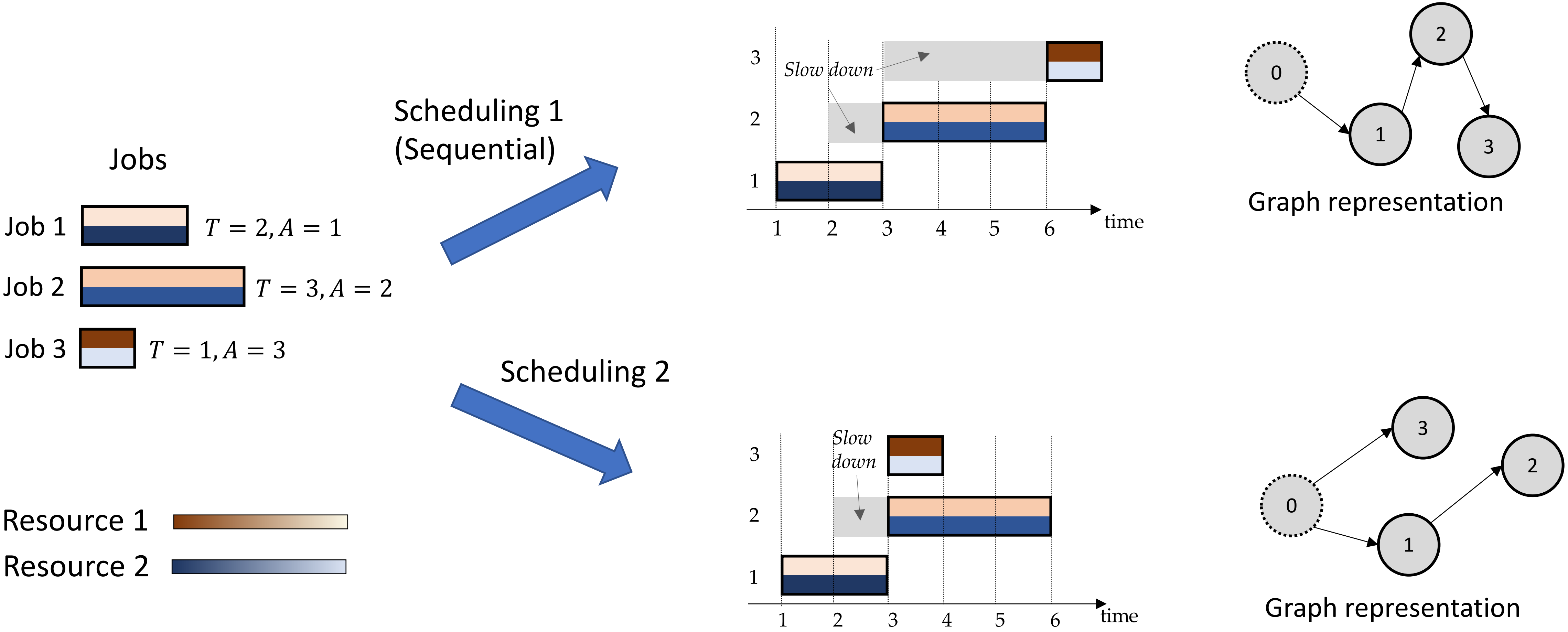


Vehicle routing

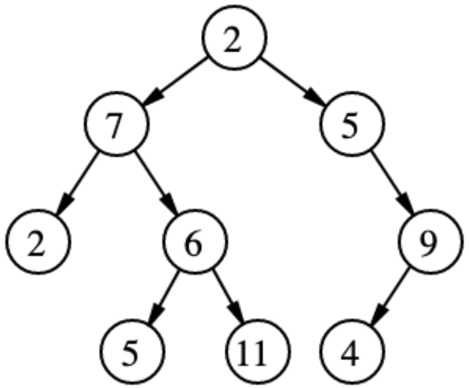
Online Job Scheduling

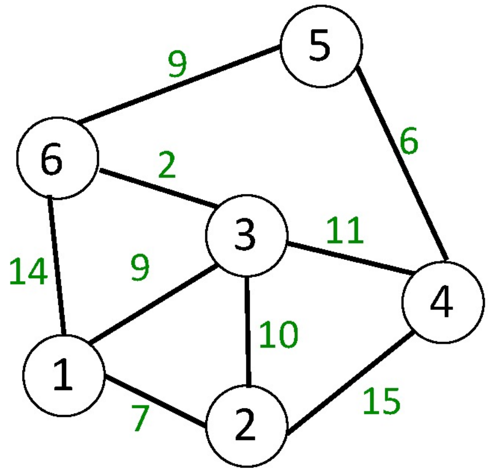


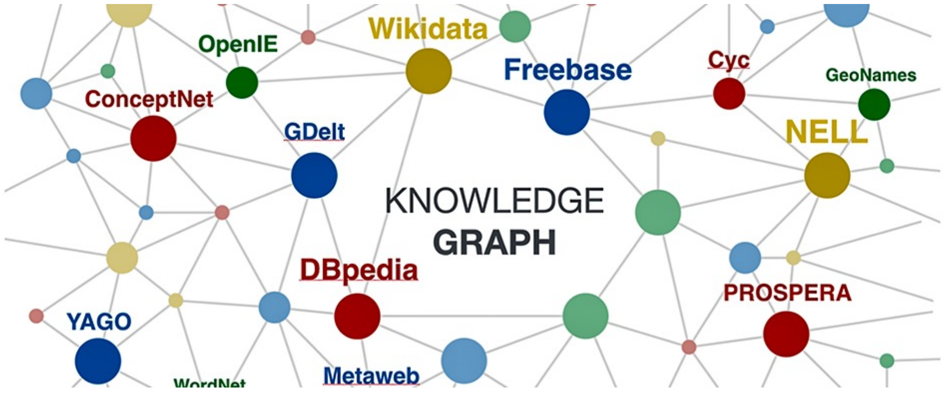
Online Job Scheduling

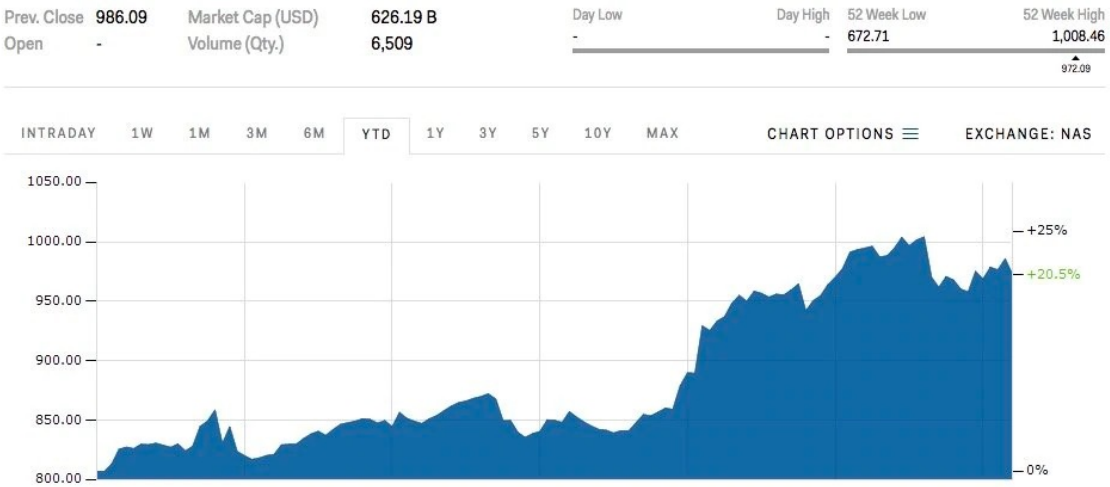


Structured Data



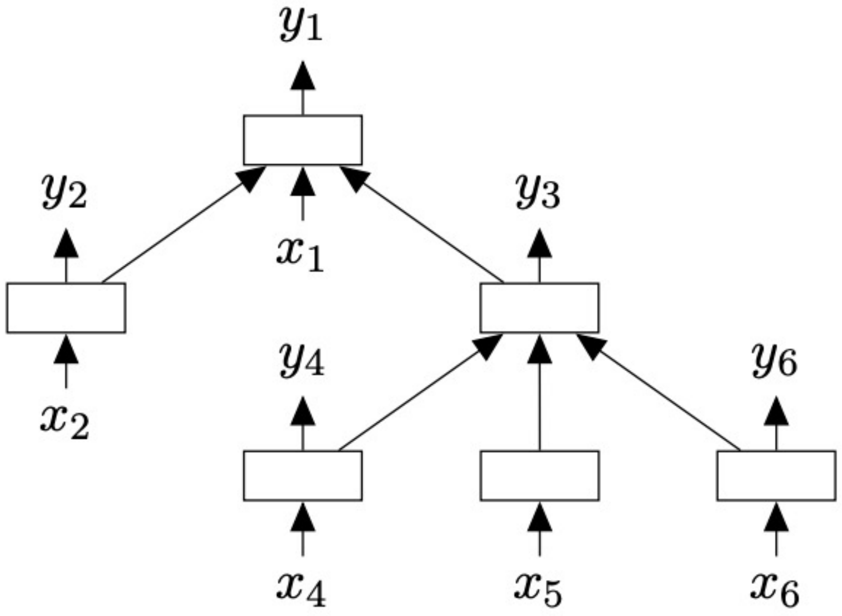






Delaunay triangulation timings (seconds)									
Number of points	10,000			100,000			1,000,000		
Point distribution / Algorithm	Uniform Random	Boundary of Circle	Tilted Grid	Uniform Random	Boundary of Circle	Tilted Grid	Uniform Random	Boundary of Circle	Tilted Grid
Div&Conq, alternating cuts									
robust	0.33	0.57	0.72	4.5	5.3	5.5	58	61	58
non-robust	0.30	0.27	0.27	4.0	4.0	3.5	53	56	44
Div&Conq, vertical cuts									
robust	0.47	1.06	0.96	6.2	9.0	7.6	79	98	85
non-robust	0.36	0.17	failed	5.0	2.1	4.2	64	26	failed
Sweepline									
non-robust	0.78	0.62	0.71	10.8	8.6	10.5	147	119	139
Incremental									
robust	1.15	3.88	2.79	24.0	112.7	101.3	545	1523	2138
non-robust	0.99	2.74	failed	21.3	94.3	failed	486	1327	failed

How to encode Structure Data



[Improved Semantic Representation From Tree-Structured Long Short-Term Memory Networks. K. Tai et al]

Child-Sum LSTM

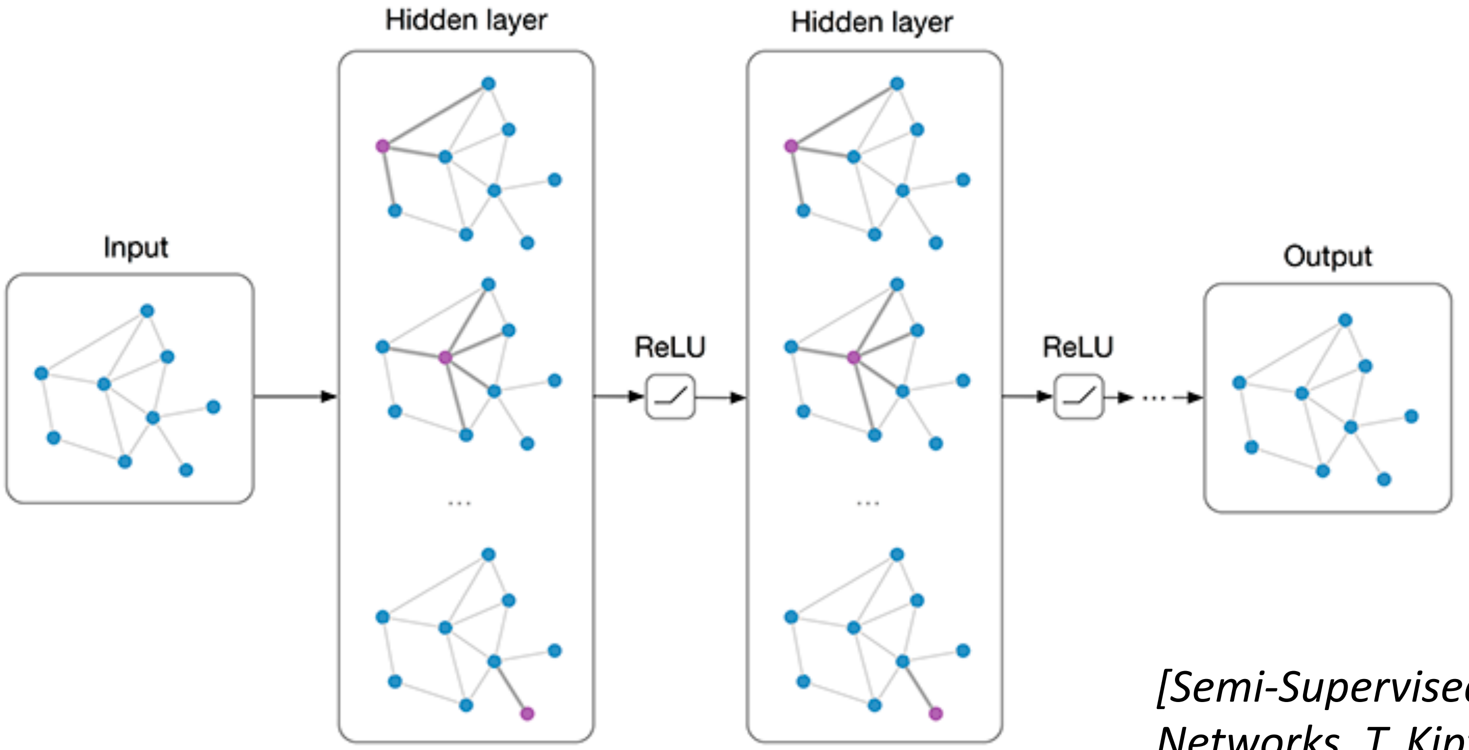
$$y_1 = f(y_2, y_3, x_1)$$

f can be very complicated:

$$\begin{aligned}
\tilde{h}_j &= \sum_{k \in C(j)} h_k, \\
i_j &= \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right), \\
f_{jk} &= \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right), \\
o_j &= \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right), \\
u_j &= \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right), \\
c_j &= i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \\
h_j &= o_j \odot \tanh(c_j),
\end{aligned}$$

How to encode Structure data

- Graph Convolutional Network (GCN)



[Semi-Supervised Classification with Graph Convolutional Networks, T. Kipf and M. Welling, ICLR 2017]

How to encode Structure data

- Graph Convolutional Network (GCN)

A: Affinity matrix of a graph

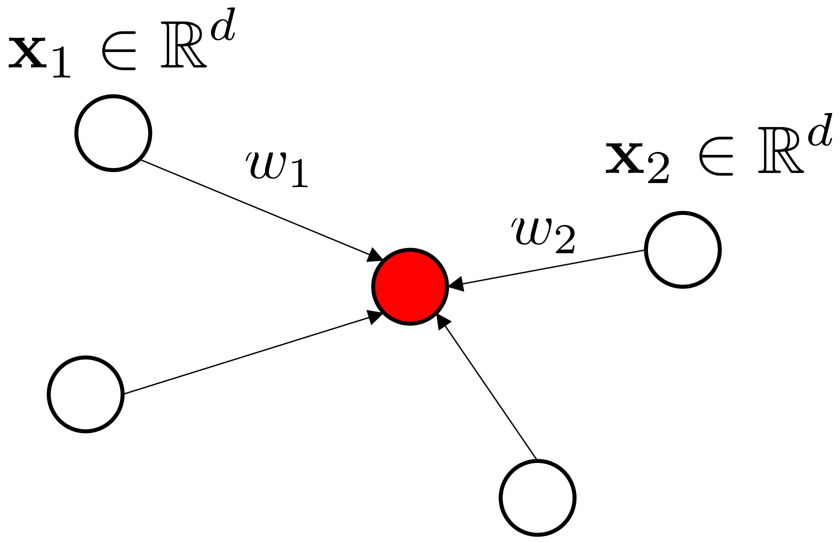
$$\hat{A} = D^{-1/2} A D^{-1/2}$$

Node embedding at layer l:

$$X_l \in \mathbb{R}^{n \times d_l}$$

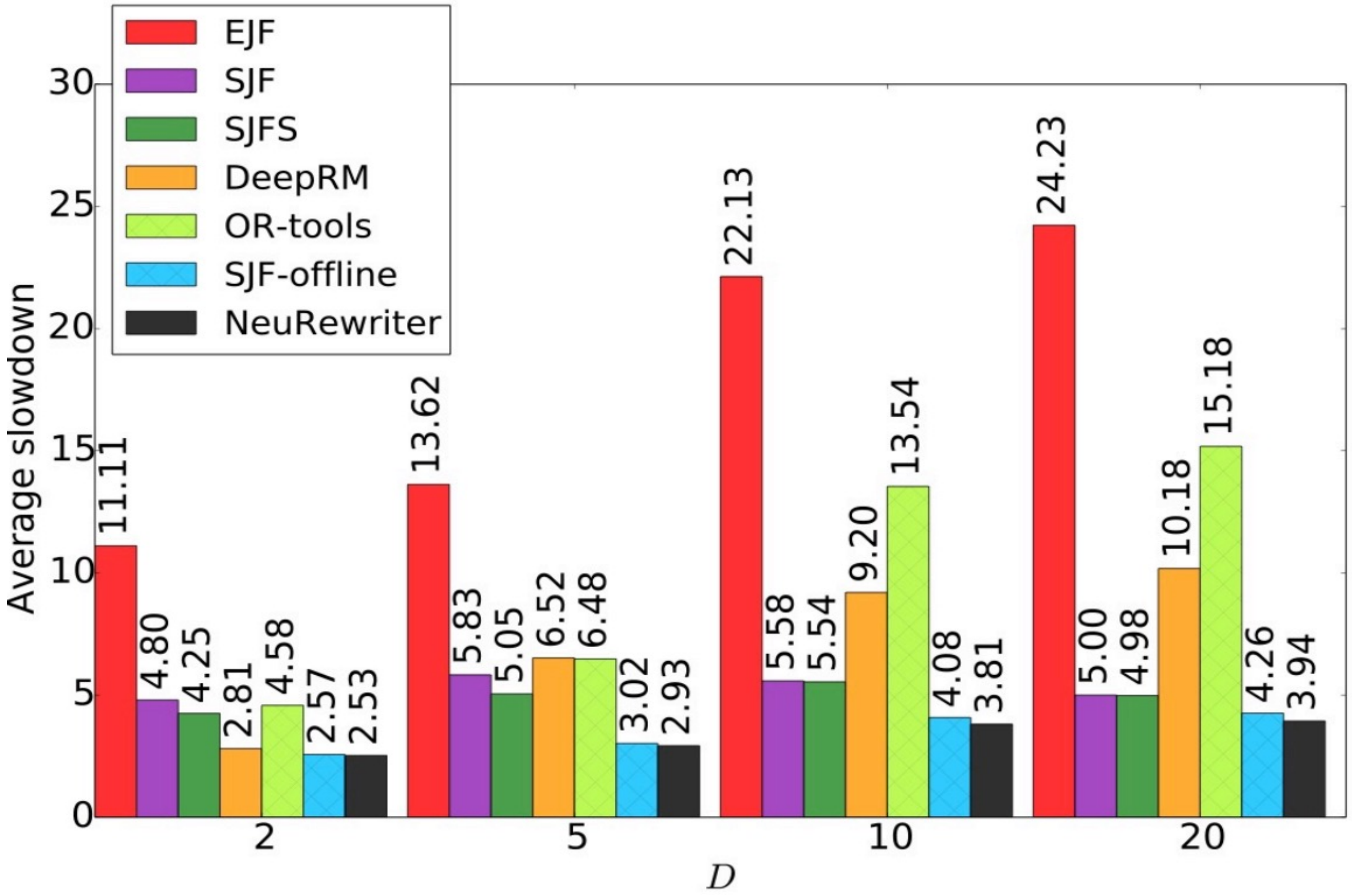
Node Embedding at layer $l+1$:

$$X_{l+1} = \mathrm{ReLU}(\hat{A} X_l W)$$



[Semi-Supervised Classification with Graph Convolutional Networks, T. Kipf and M. Welling, ICLR 2017]

Online Job Scheduling



D: Number of resources

Baselines:

- Earliest Job First (EJF)
- Shortest Job First (SJF)
- Shortest First Search (SJFS)
- DeepRM

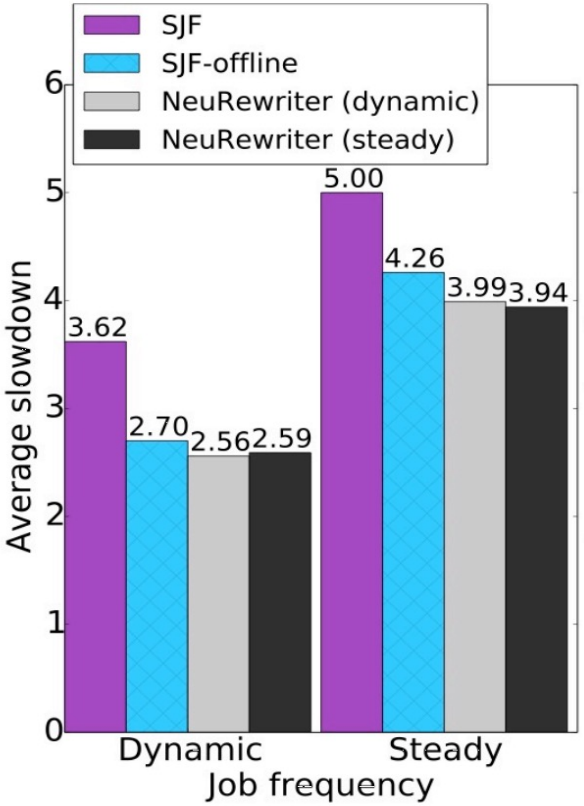
Offline baselines:

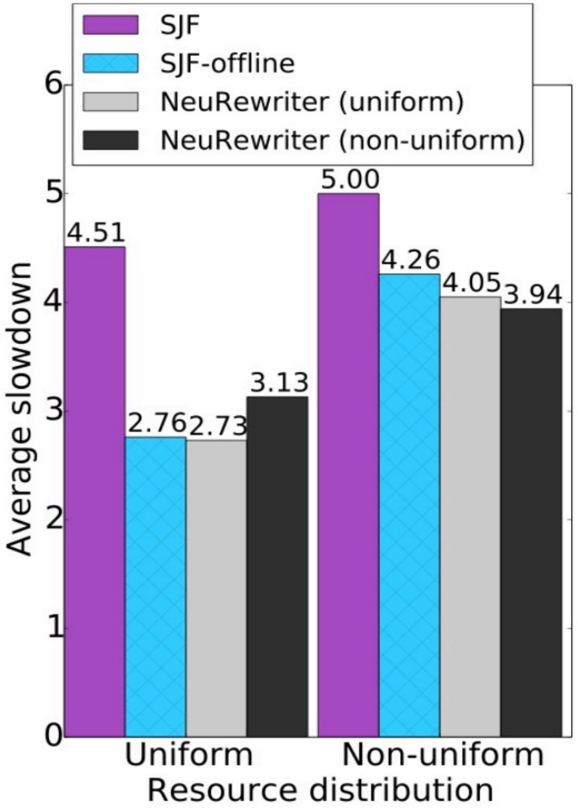
- Google OR-tools (OR-tools)
- SJF-offline

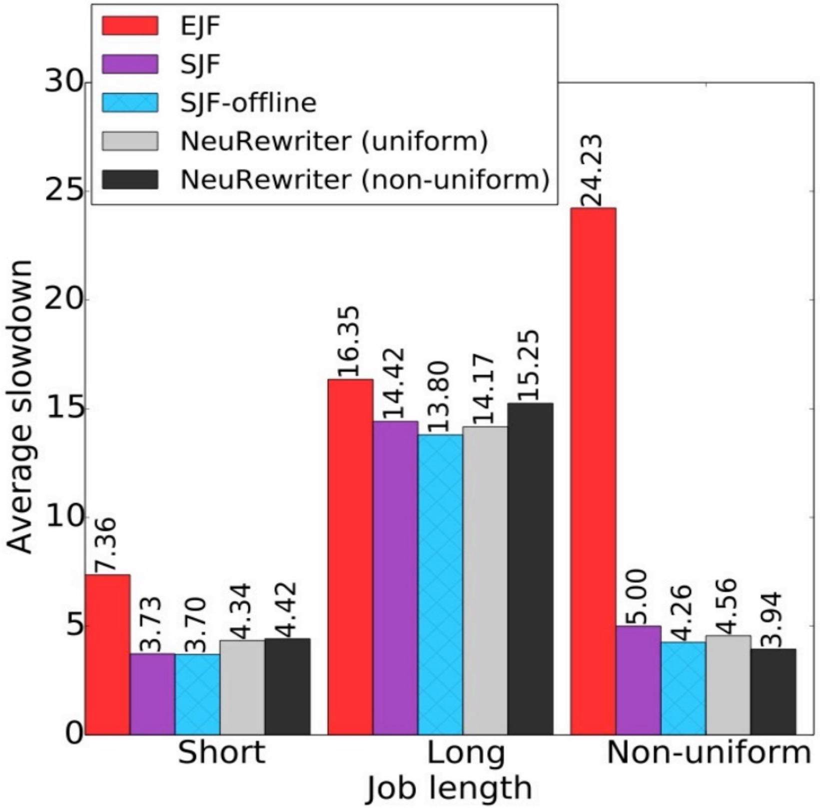
	Time (s)
OR-tools	10.0
DeepRM	0.020
NeuRewriter	0.037

Online Job Scheduling: Ablation Study

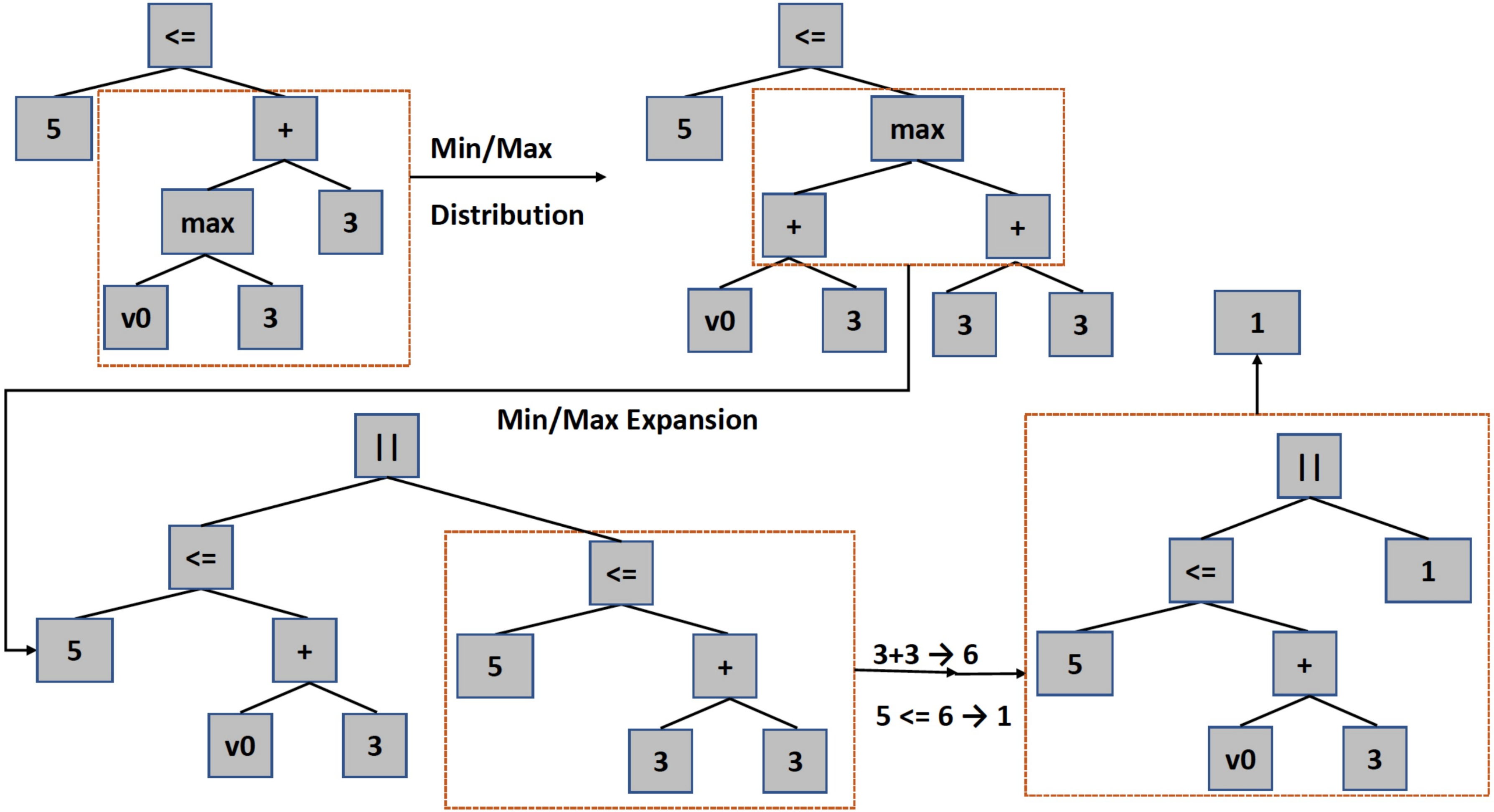
The learned model can generalize to different job distributions.



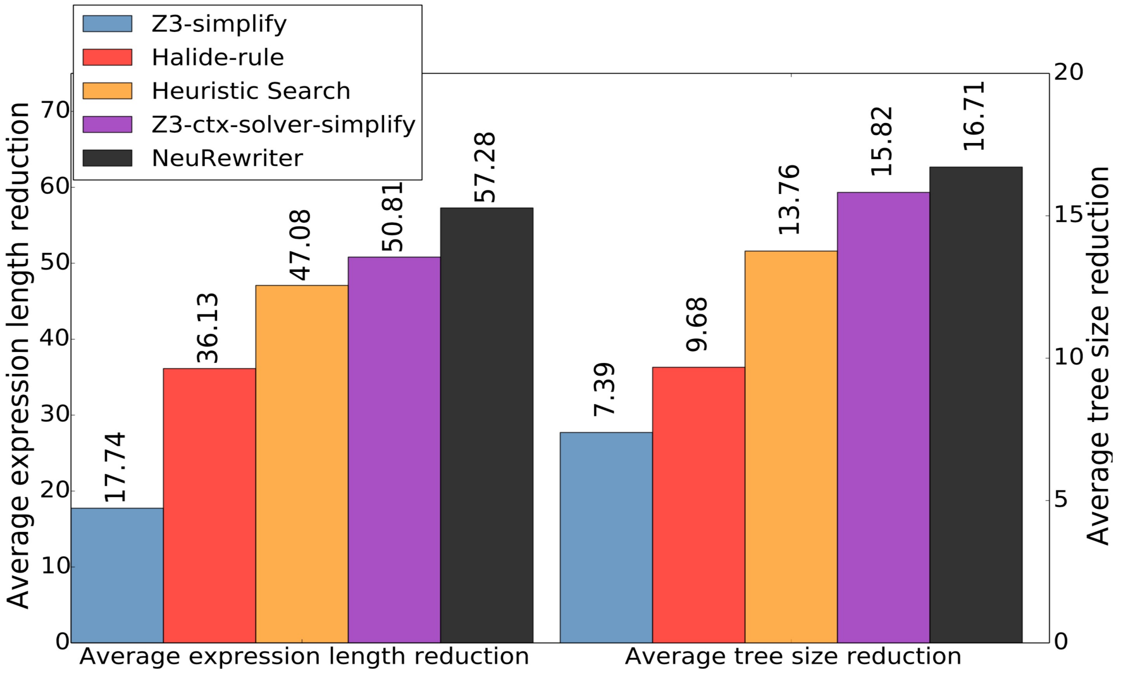




Expression Simplification



Expression Simplification



Baselines:

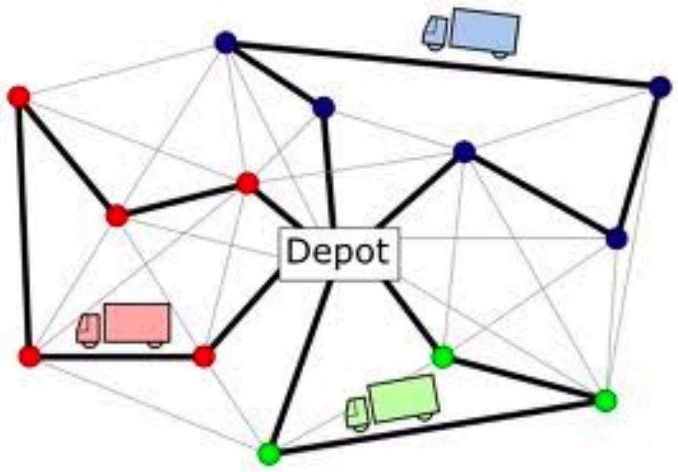
- Z3-simplify
- Z3-ctx-solver-simplify
- Heuristic Search
- Halide rules

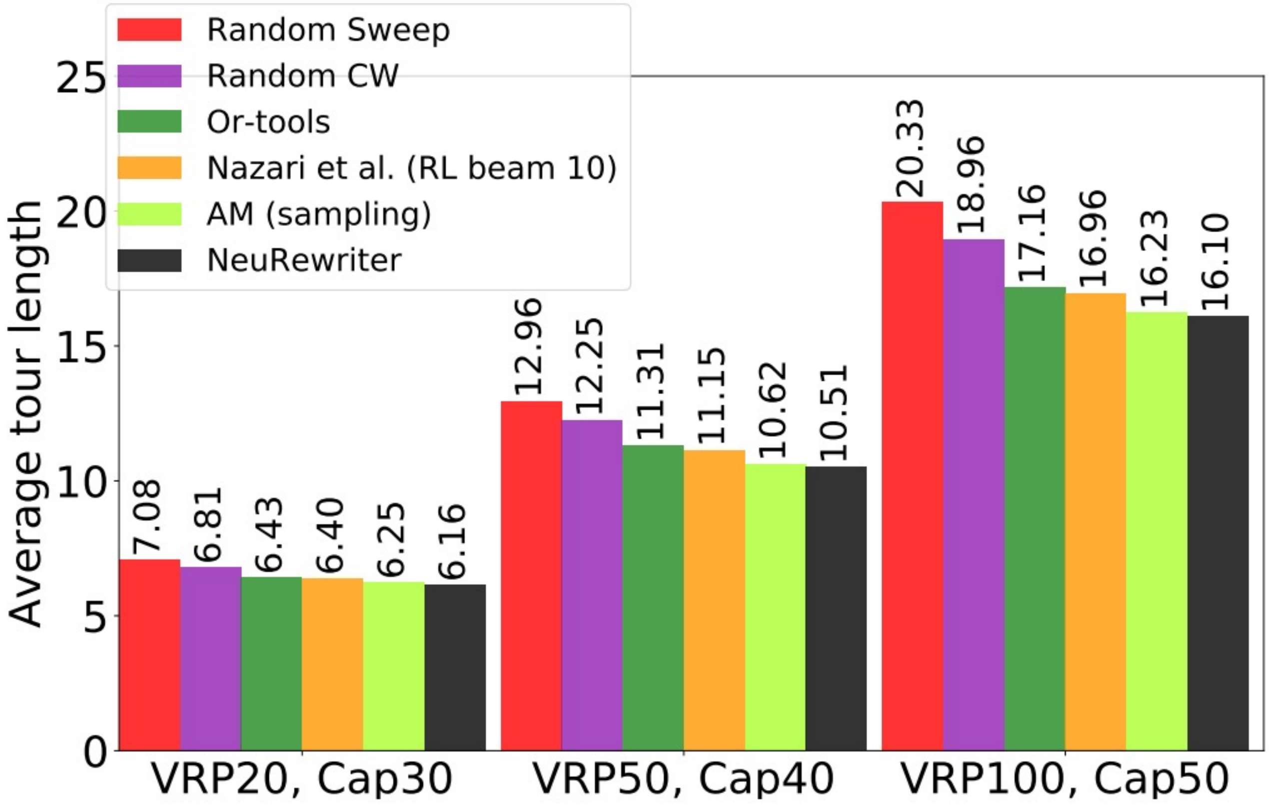
	Time (s)
Z3-solver	1.375
NeuRewriter	0.159

Follow-up work: Getting rid of manually specified rules

[H. Shi et al., Deep Symbolic Superoptimization without Human Knowledge, ICLR 2020]

Capacitated Vehicle Routing





Code is available: https://github.com/facebookresearch/neural-rewriter

Coda: An End-to-End Neural Program Decomplier

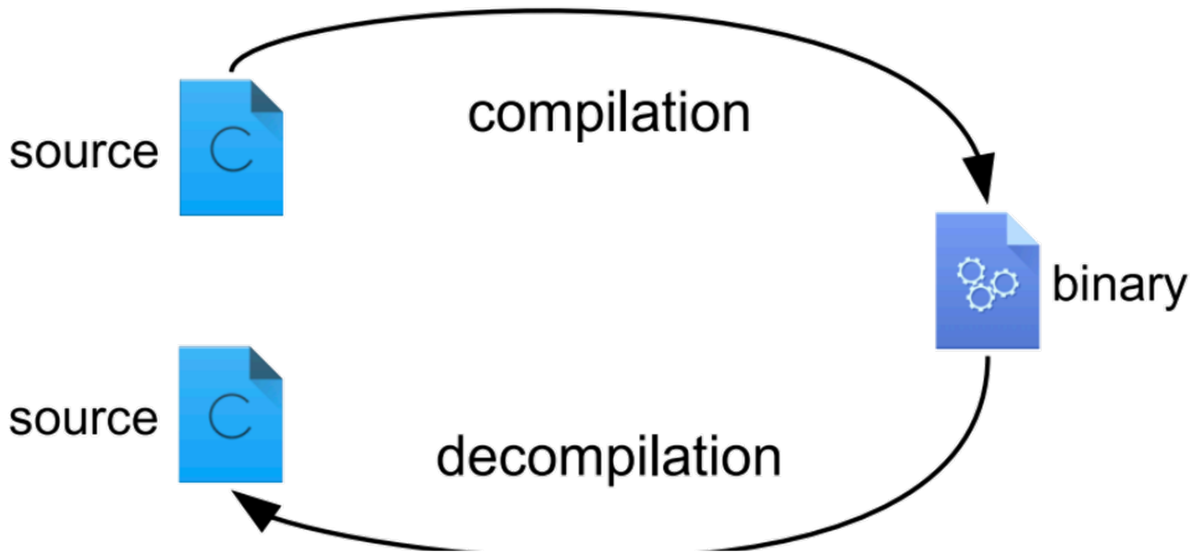
Cheng Fu[1], Huili Chen[1], Haolan Liu[1], Xinyun Chen[3], Yuandong Tian[2], Farinaz Koushanfar[1], Jishen Zhao[1]

[1]UC San Diego, [2]Facebook AI Research, [3]UC Berkeley

NeurIPS 2019

Background: Decompilation

- Goal of Decompilation
  - From Binary Execution to High-level program language

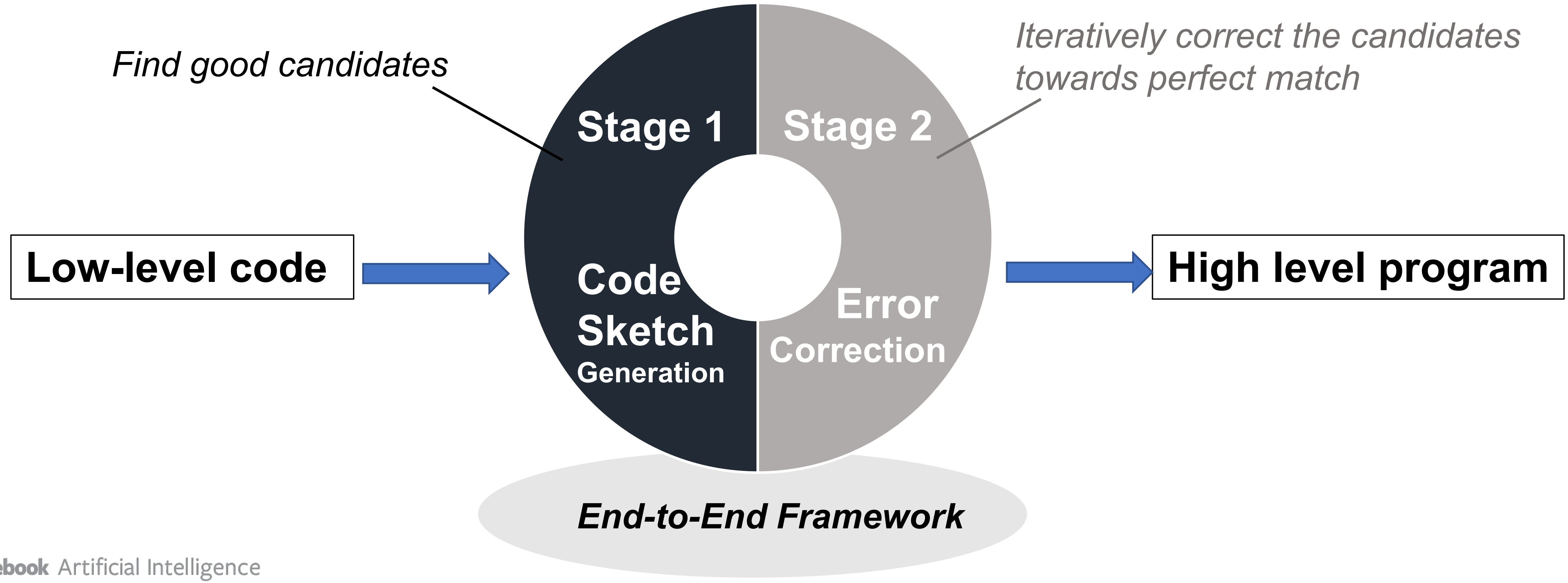


Challenges

- Many hardware architectures (ISA): x86, MIPS, ARM
- Many Programming Languages (PL)
  - Extra Human effort to extend to the new version of the hardware architectures or programming languages
- Our goals:
  - Maintain both the functionality and semantics of the binary executables
  - Make the design process end-to-end (generalizable to various ISAs and PLs)

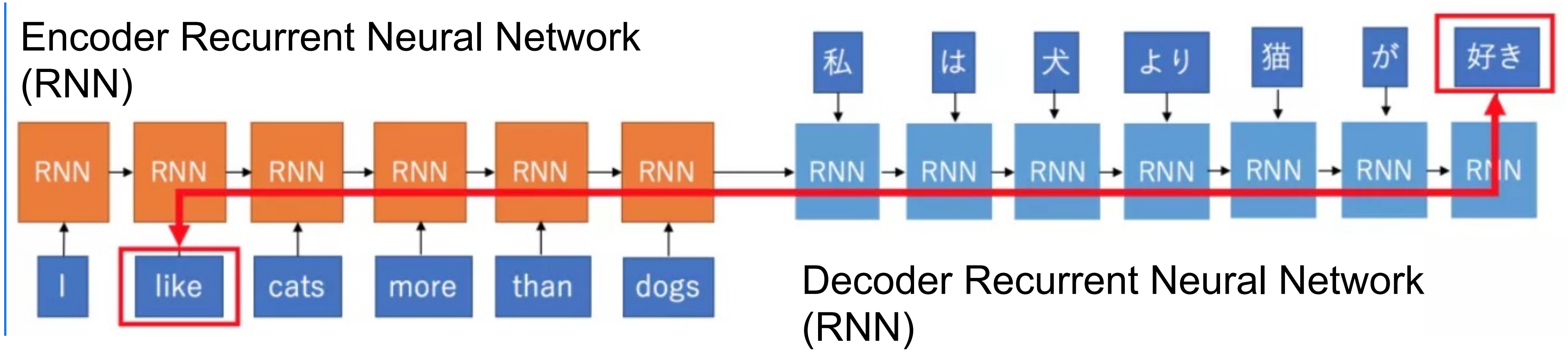
Coda Design

Leverage both syntax and dynamic information



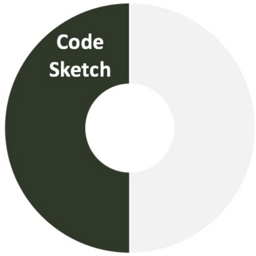
Stage 1: Coda Sketch Generation

- Is Decompilation simply a translation problem?



More than a translation problem!

Stage 1: Coda Sketch Generation

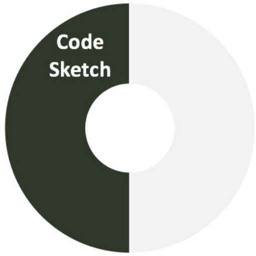


- **Encoder**
  - N-ary Tree Encoder to capture **inter** and **intra** dependencies of the low-level code.
  - Opcode and its operands are encoded together
  - Different encoder is used for different instruction types
    - memory (mem)
    - branch (br)
    - arithmetic (art).

```
# source C program
 a = b * c ;
 if( a > c ){
   c = a * c - b;
 }
```

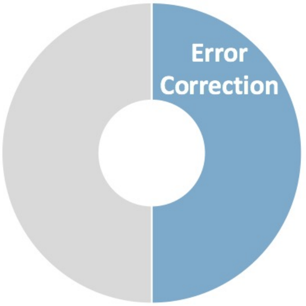
```
0  lw    $1, 24($fp)     mem h0
1  lw    $2, 20($fp)     mem h1
2  mul   $1, $1, $2      art h2
3  sw    $1, 28($fp)     mem h3
4  lw    $1, 28($fp)     mem h4
5  lw    $2, 20($fp)     mem h5
6  slt   $1, $2, $1      art h6
7  beqz  $1, $BB0_3       br h7
8  j  $B2                 br h8
9  $B2:                   br h9
10 lw    $1, 28($fp)     mem h10
11 lw    $2, 20($fp)     mem h11
12 mul   $1, $1, $2      art h12
13 lw    $2, 24($fp)     mem h13
14 subu  $1, $1, $2      art h14
15 j     $B3              br h15
16 sw    $1, 20($fp)     mem h16
```

Stage 1: Coda Sketch Generation



- **Decoder**
  - Generate Abstract Syntax Tree (AST)
  - AST can be equivalently translated into its corresponding high level Program
  - Advantages:
    - Prevent error propagation/ Preserve node dependency / capture PL grammar
    - Boundaries are more explicit (terminal nodes)
  - Using Attention Mechanism

Stage 2: Iterative Error Correction



- The sketch generated in Stage 1 may contain errors:
  - mispredicted tokens, missing lines, redundant lines

Golden program

```
If( a > c ) {
    a = b + c * a;
    b = a – c;
}
```

Wrongly predicted

```
If( a > b ) {
    a = b + c * a;
    b = a - b;
}
```

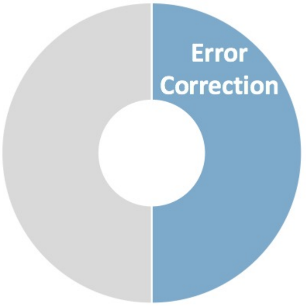
Missing lines

```
If( a > c ) {
    a = b + c * a;
}
```

Redundant lines

```
If( a > c ) {
    a = b + c * a;
    b = a;
    b = a;
}
```

Stage 2: Iterative Error Correction



- Correct the error by iterative Error Predictor (EP)
  - Iterative rewriting!
  - Spot errors in the generated assembly codes
  - Fix errors and recompile
  - Repeat 10 times

Experimental Setup

- Compiler configuration: Clang **–O0** (no code optimization)
- Benchmarks:
  - Synthetic programs:
    - **Karel library (Karel)** – only function calls
    - **Math library (Math)** – function calls with arguments
    - **Normal expressions (NE)** – (^,&,*,-,<<,>>,|,% ....)
    - **Math library + Normal expressions (Math+NE)** – replaces the variables in NE with a return value of math function.
- Metrics:
  - Token Accuracy
  - Program Accuracy

Result – Stage 1 Performance

- Token accuracy (%) across benchmarks

Benchmarks	Seq2Seq	Seq2Seq+Attn	Seq2AST+Attn	Inst2seq+Attn	Inst2AST+Attn
Karel_S	51.61	97.13	99.81	98.83	**99.89**
Math_S	23.12	94.85	99.12	96.20	**99.72**
NE_S	18.72	87.36	90.45	88.48	**94.66**
(Math+NE)_S	14.14	87.86	91.98	89.67	**97.90**
Karel_L	33.54	94.42	98.02	98.12	**98.56**
Math_L	11.32	91.94	96.63	93.16	**98.63**
NE_L	11.02	81.80	85.92	85.97	**91.92**
(Math+NE)_L	6.09	81.56	85.32	86.16	**93.20**

- Highest token accuracy across all benchmarks (96.8% on average) compared to baselines.
- 10.1% and 80.9% margin over a naive Seq2Seq model with and without attention.
- More tolerant to the growth of program length.

Result – Stage 2 Performance

- Program accuracy (%)

BenchMarks	(i) Error Detection		(ii) Befor EC		After EC	
	$s2s$,10	$i2a$,10	$s2s$	$i2a$	$s2s$	$i2a$
Math_S	91.4	94.2	40.1	64.8	91.2	**100.0**
NE_S	83.5	88.7	6.6	12.2	53.0	**78.6**
$(\text{Math+NE})_S$	83.6	90.1	3.5	43.2	63.6	**89.2**
Math_L	87.5	91.3	21.7	51.8	83.9	**99.5**
NE_L	78.1	84.5	0.2	2.6	33.1	**56.4**
$(\text{Math+NE})_L$	80.2	85.3	0.1	4.9	38.3	**67.2**

Baseline

Ours

s2s = sequence-to-sequence with attention

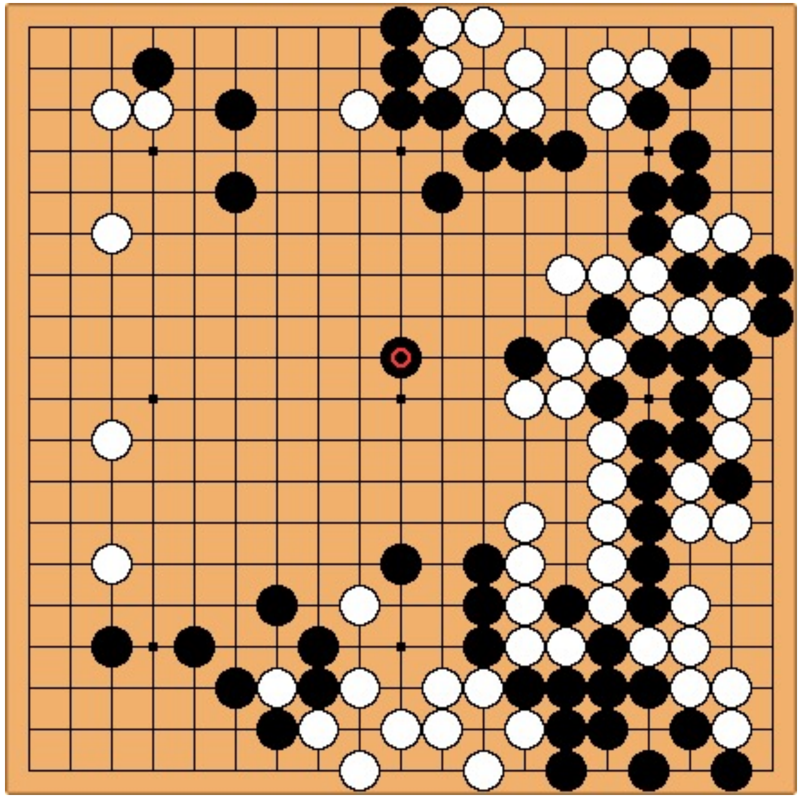
I2a = instruction encoder to AST decoder with attention

Result – Overall

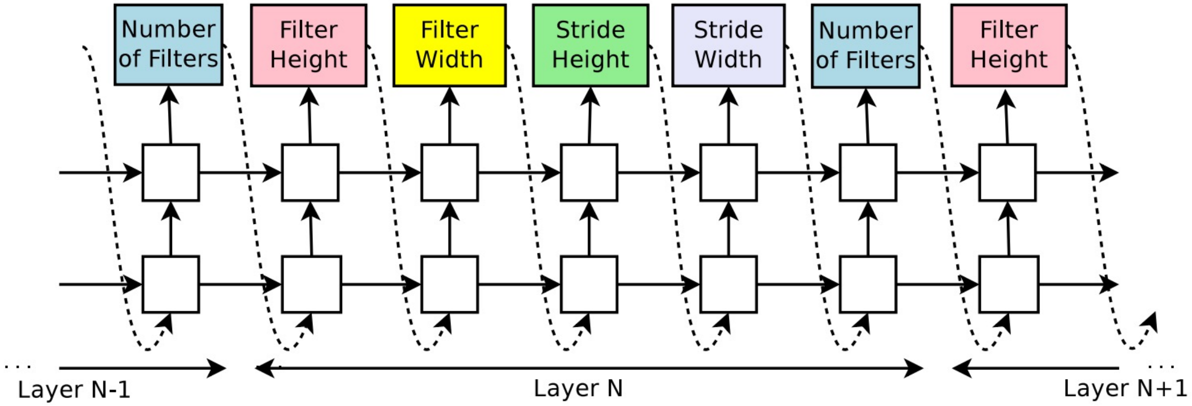
- Coda vs. traditional decompiler (RetDec)
  - Lines of code: ~10K vs. ~500K **-- 50x reduction**
  - Toolkit size: ~10MB Neural network size vs. ~5GB toolkit size **-- 500x reduction**

- Summary:
  - First neural-based decompiler
    - Generative models with iterative error corrections.
  - Significantly outperforms seq2seq models.

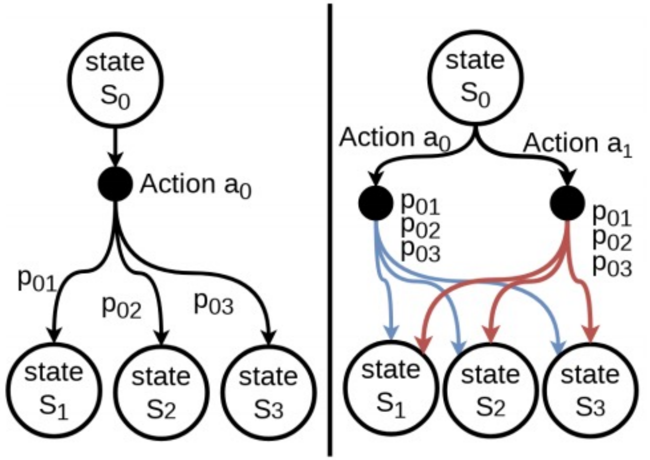
Predefined Action Space



Fixed action space = R^{361}



[B. Zoph and Q. Le, Neural Architecture Search with Reinforcement Learning, 2016]



[G. Malazgirt, TauRieL: Targeting Traveling Salesman Problem with a deep reinforcement learning inspired architecture]

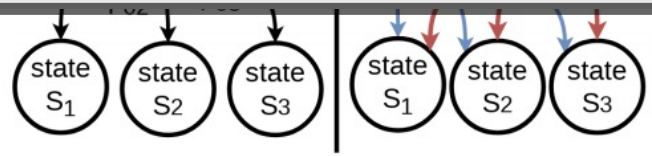
Predefined Action Space

Fixed action space = R^{361}

Why Predefined Action Space?

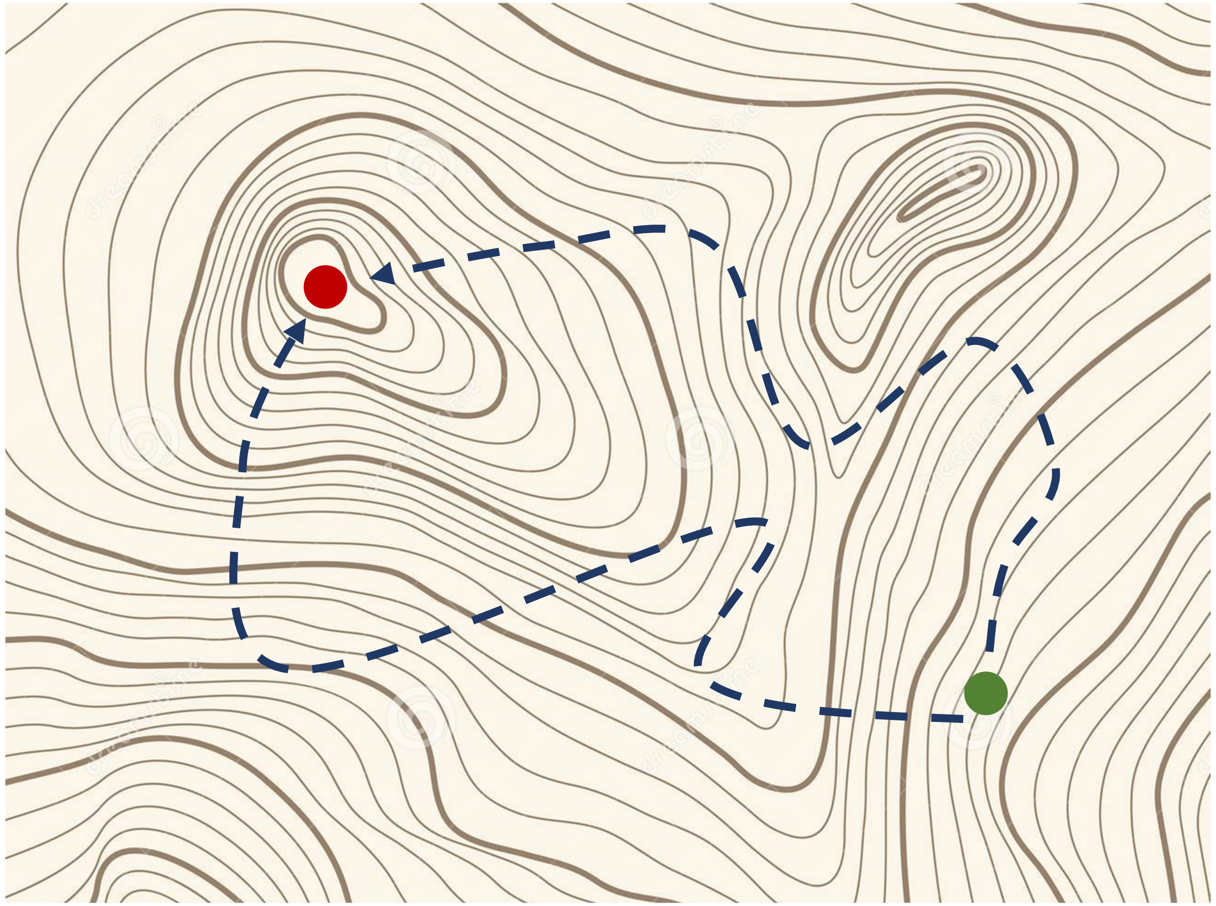
N+1

2016]



[G. Malazgirt, TauRieL: Targeting Traveling Salesman Problem with a deep reinforcement learning inspired architecture]

Why Predefined Action Space?



We only care the final solution

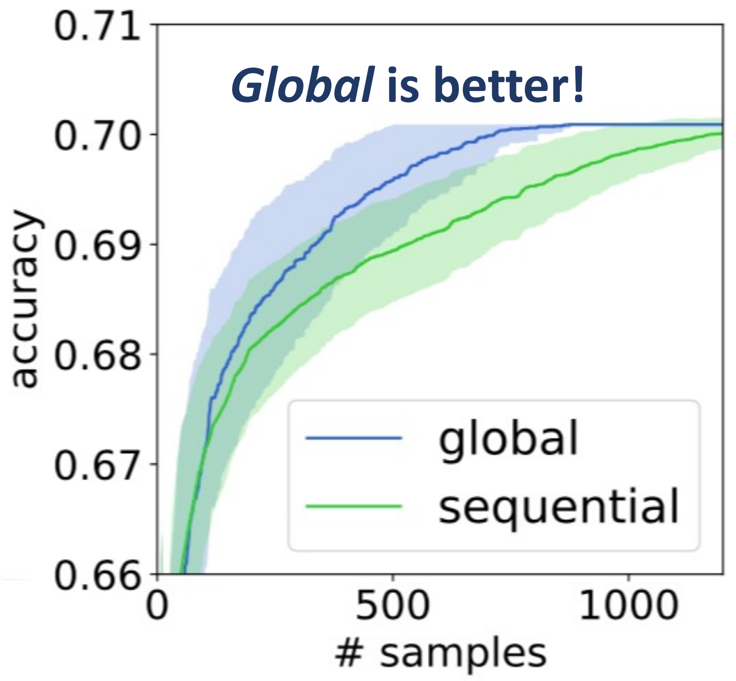
We don’t care how we get it.

Different Representation matters

Depth = {1, 2, 3, 4, 5}
Channels = {32, 64}
KernelSize = {3x3, 5x5}

1364 networks.

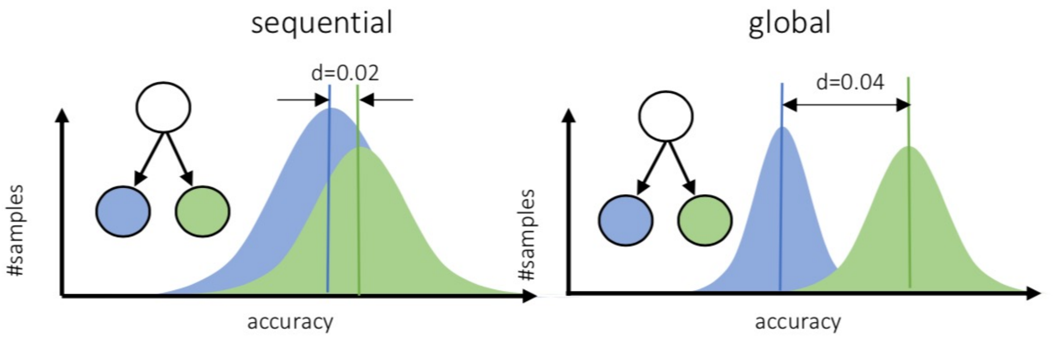
Goal: Find the network with the best accuracy using fewest trials.



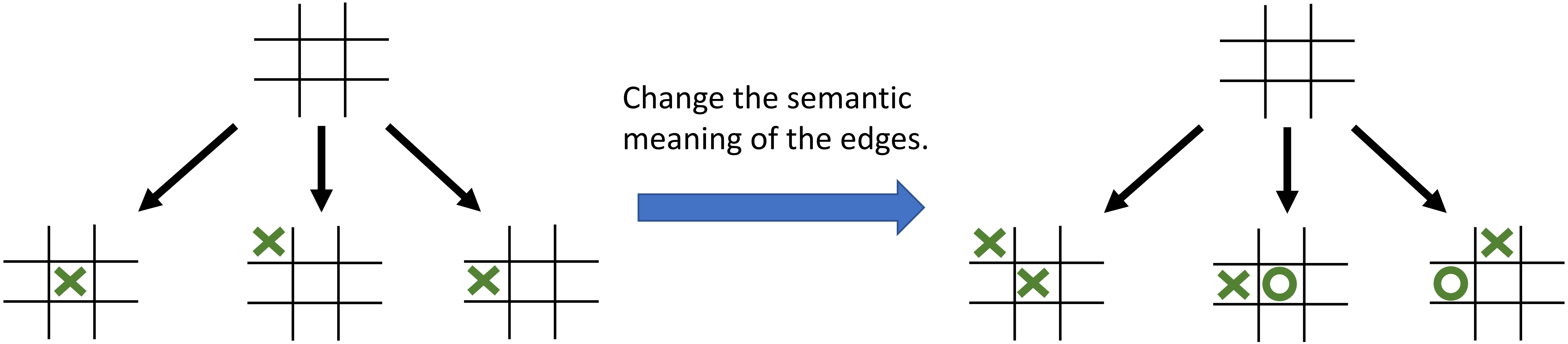
Representation of action space

Sequential = { add a layer, set K, set C }

Global = { Set depth, set all K, set all C }



The Meaning of Learning Action Space



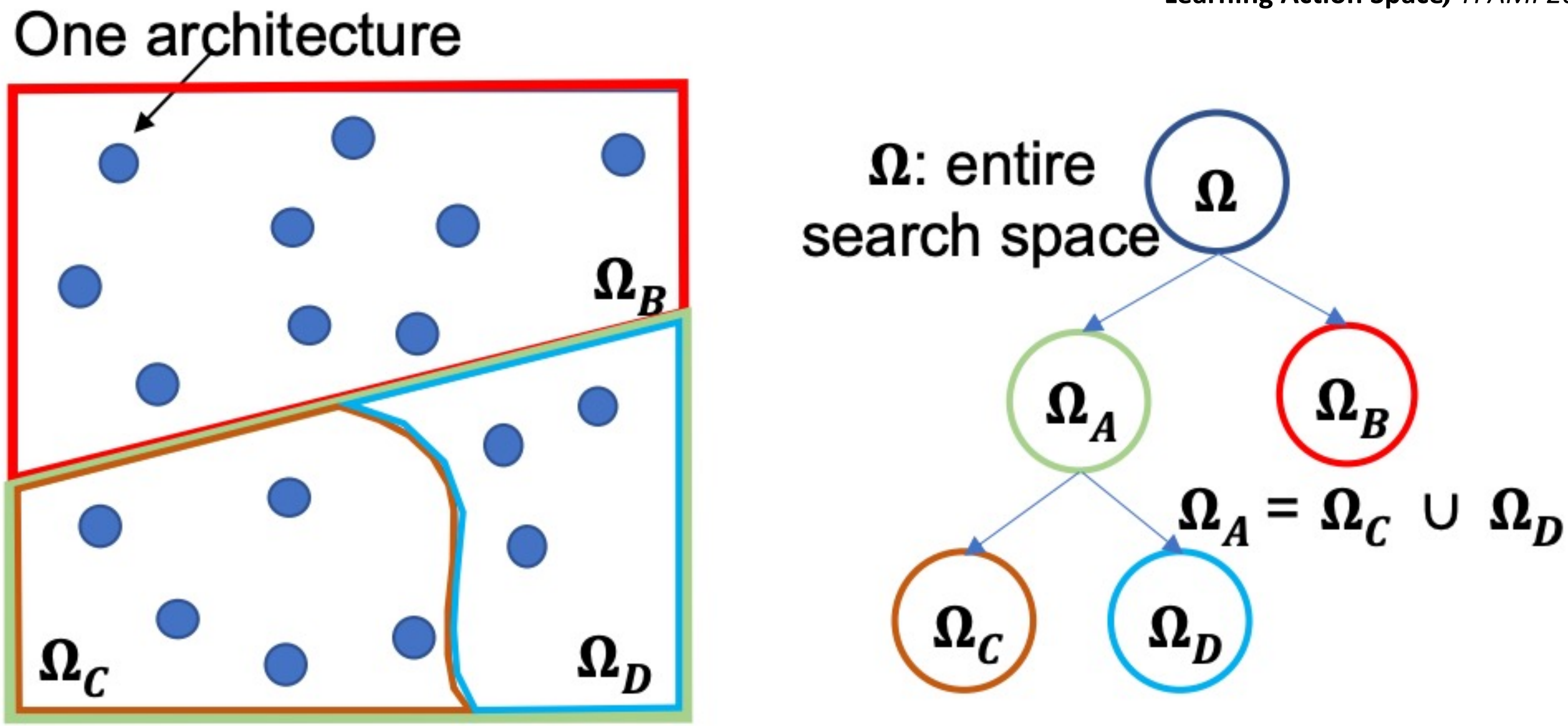
Not allowed in games, but doable in optimization.

Learning Action Space

Linnan Wang Saining Xie Teng Li Rodrigo Fonseca Yuandong Tian

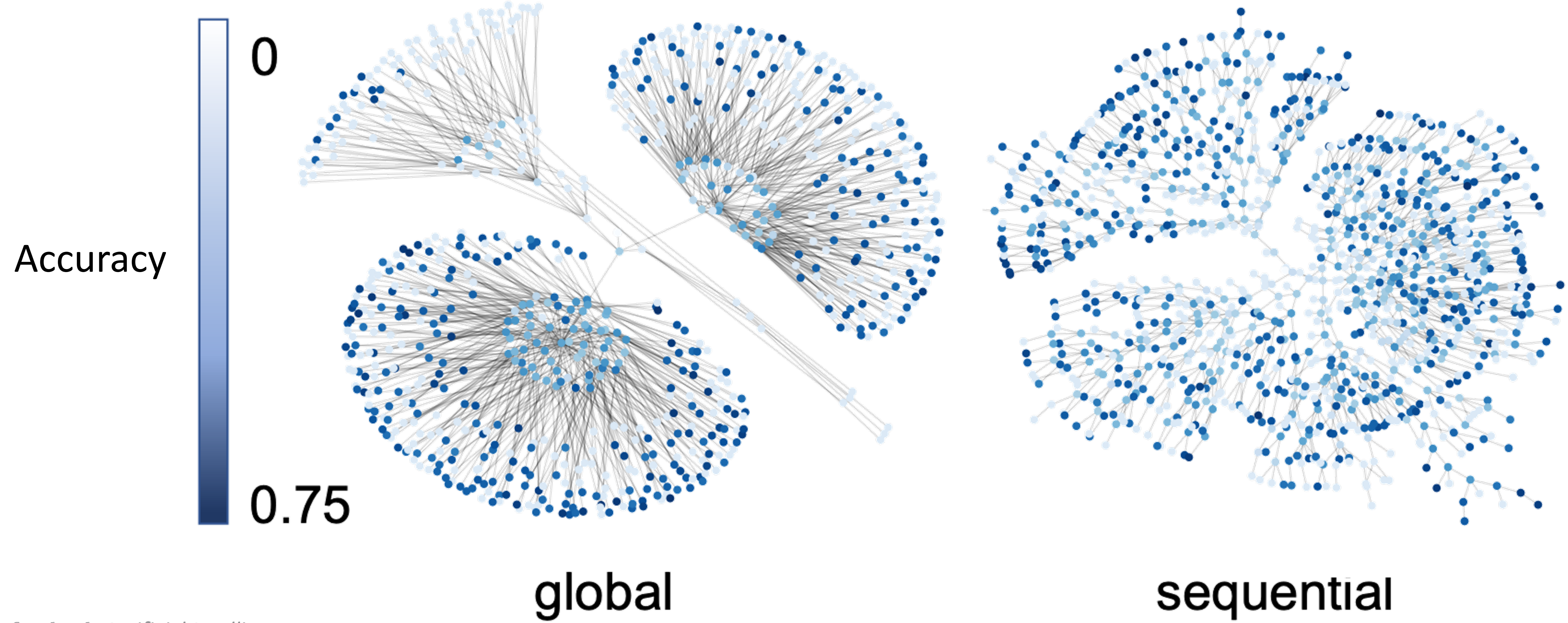
[L. Wang, R. Fonseca, Y. Tian, **Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search,** *NeurIPS 2020]*

[L. Wang, S. Xie, T. Li, R. Fonseca, Y. Tian, **Sample-Efficient Neural Architecture Search by Learning Action Space,** *TPAMI 2021]*

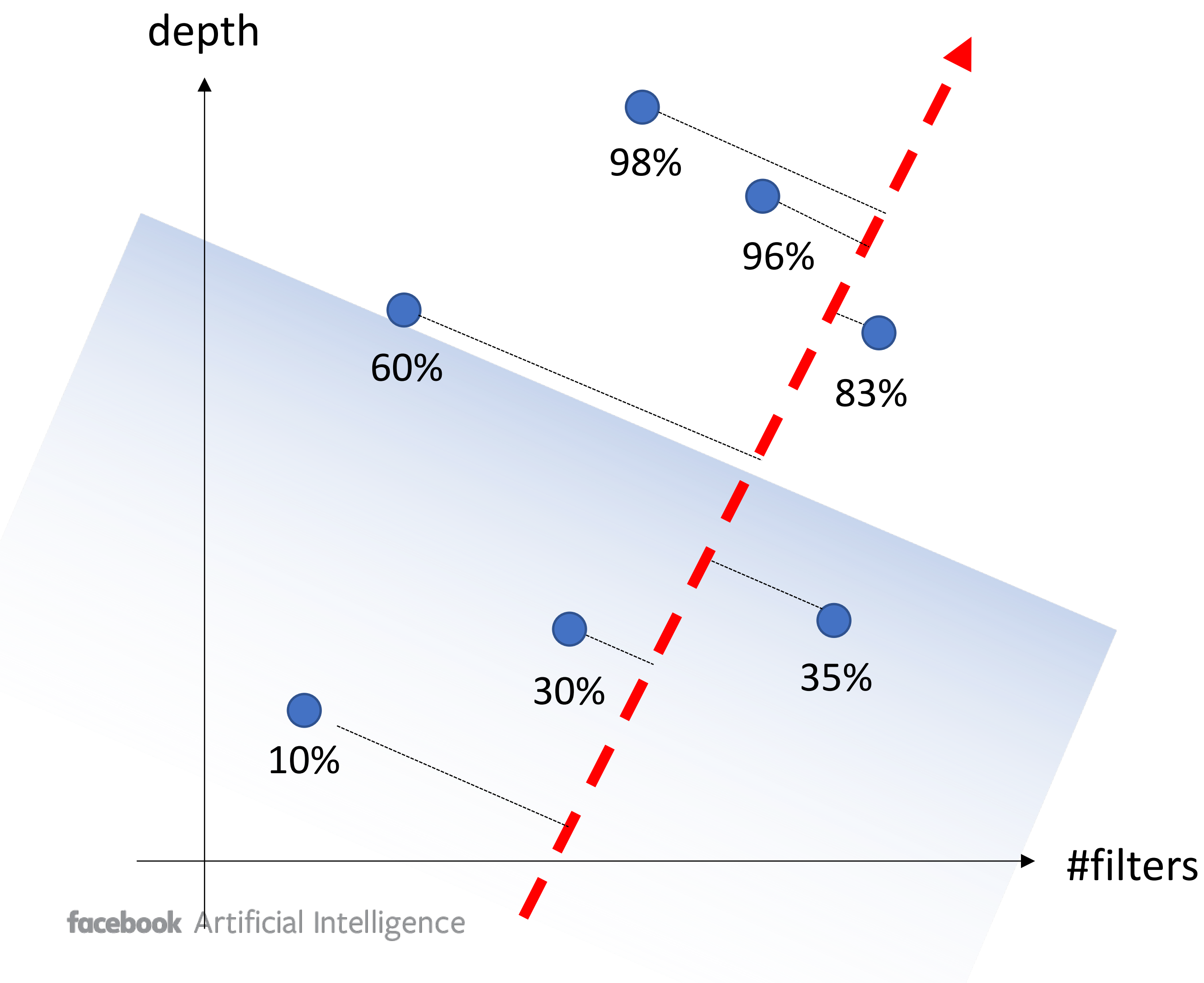


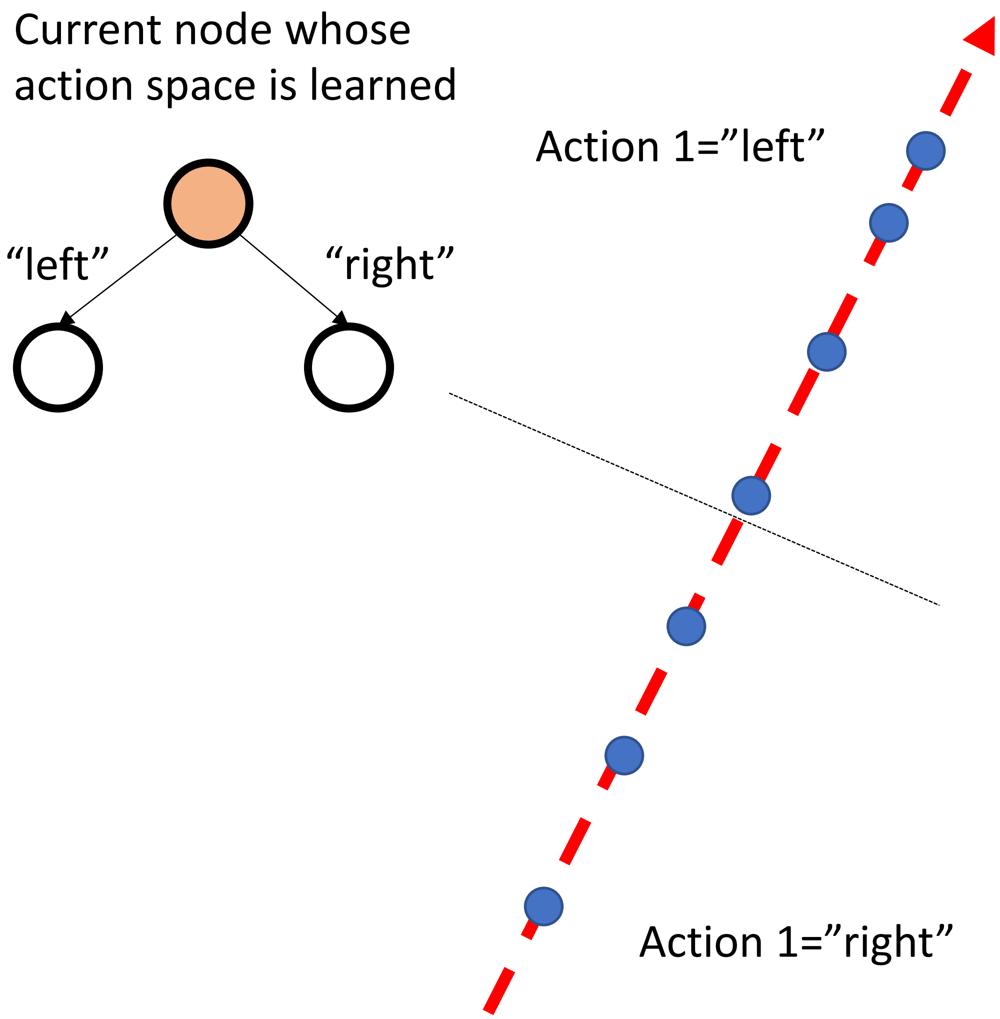
Partition = Action

Different Partition → Different Value Distribution

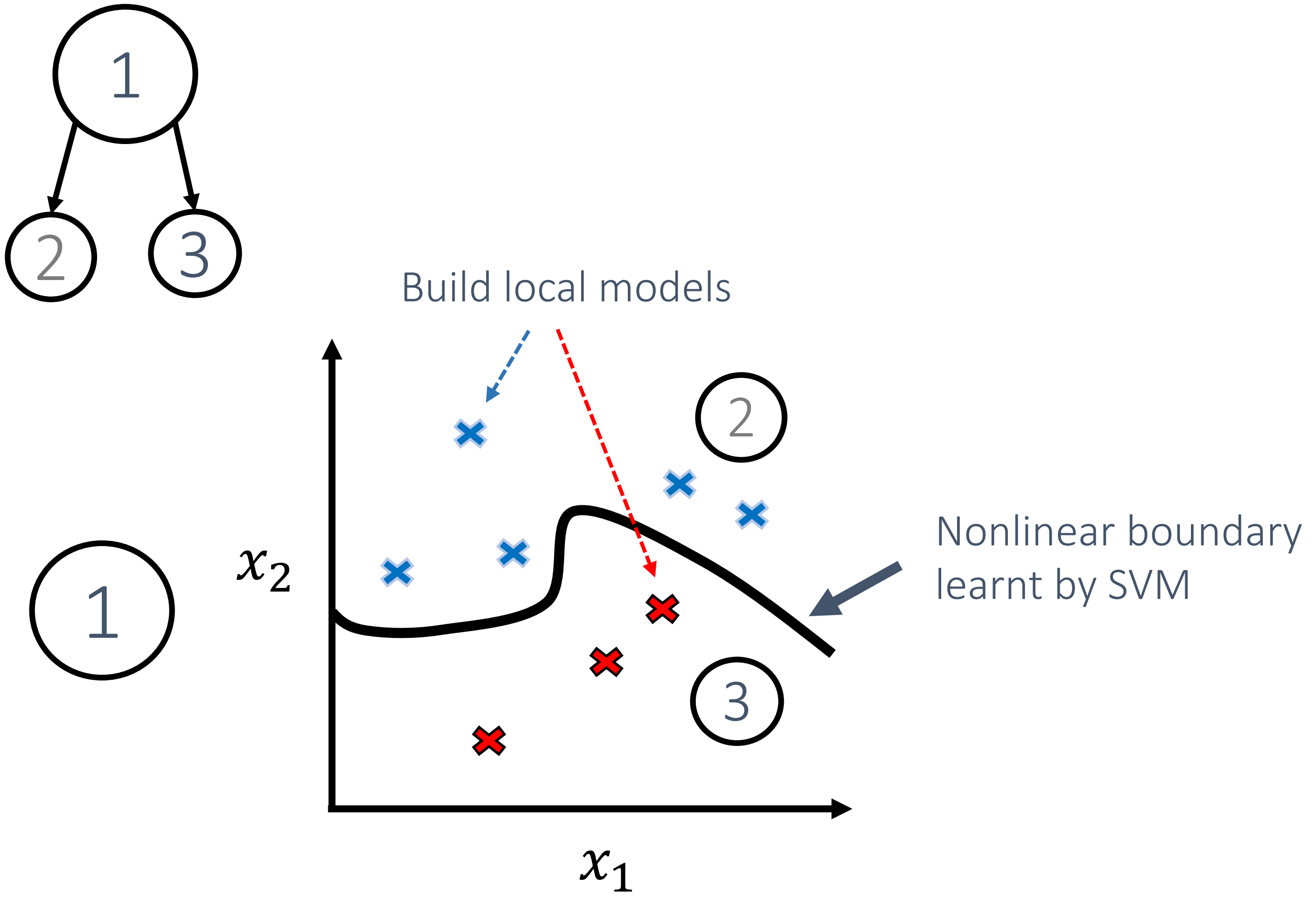


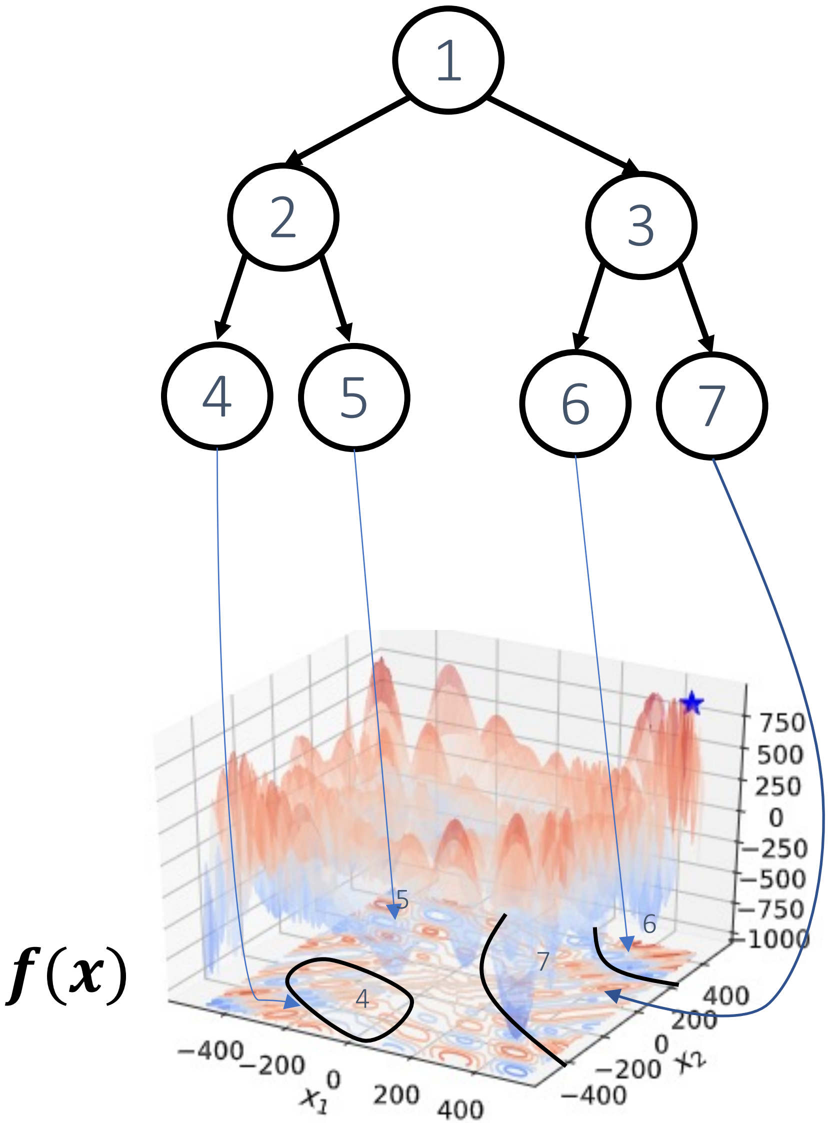
Learn action space





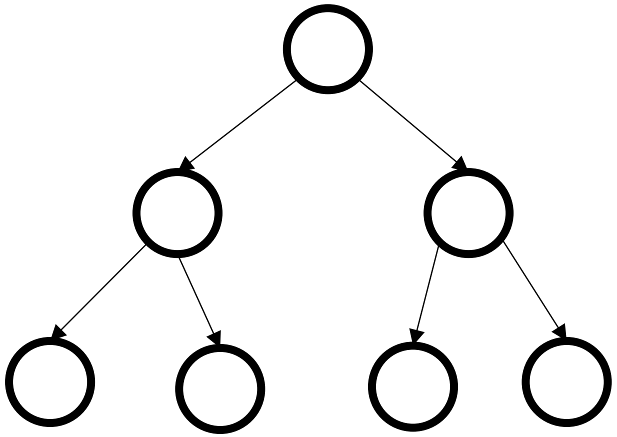
Nonlinear Partition





[L. Wang, et al, Learning Search Space Partition for Black-box Optimization using MCTS, NeurIPS 2020]

Approach



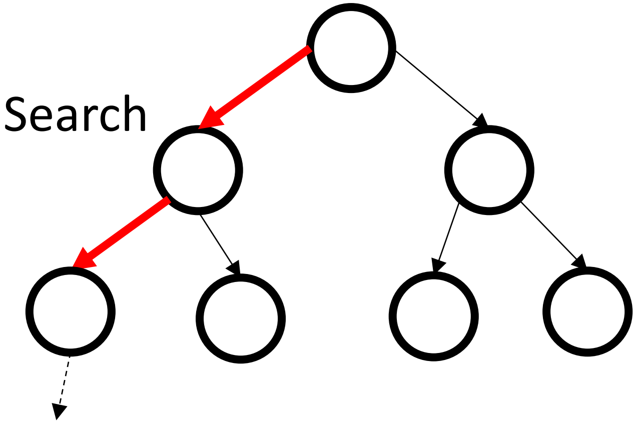
Fixed action branches
(but not action space)

(a) Train the action space.

	Accuracy
(filter=2, depth=5)	85%
(filter=3, depth=7)	92%

(b) Search using learned action space until a fixed #rollouts are used.

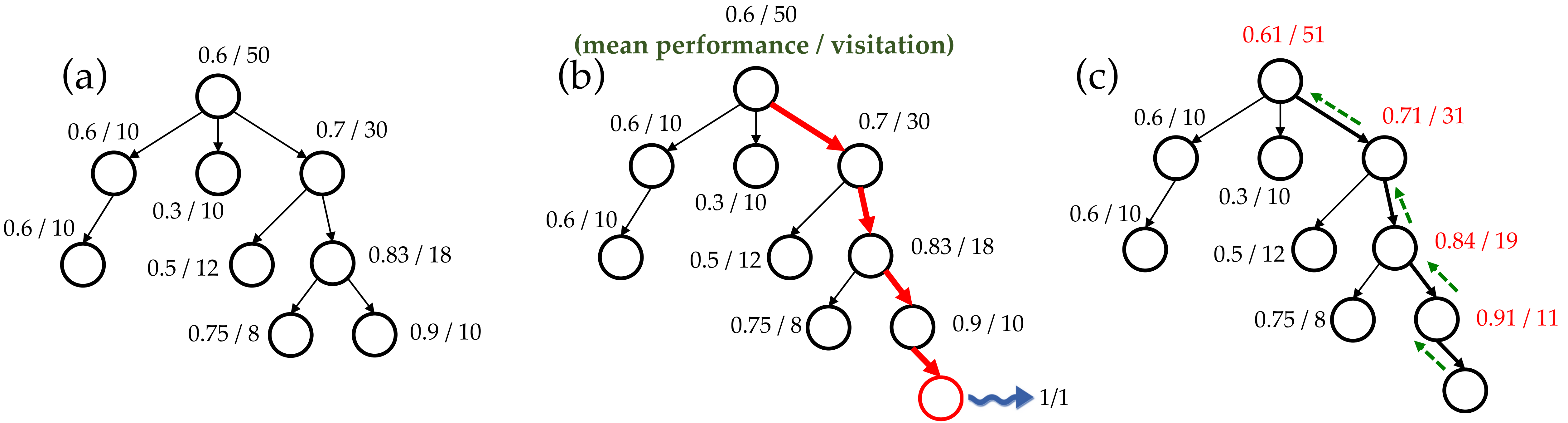
Monte Carlo Tree Search
(MCTS)



Getting the true quality $f(x)$ for the solution x

Monte Carlo Tree Search

Search towards the good nodes while keeping exploration in mind

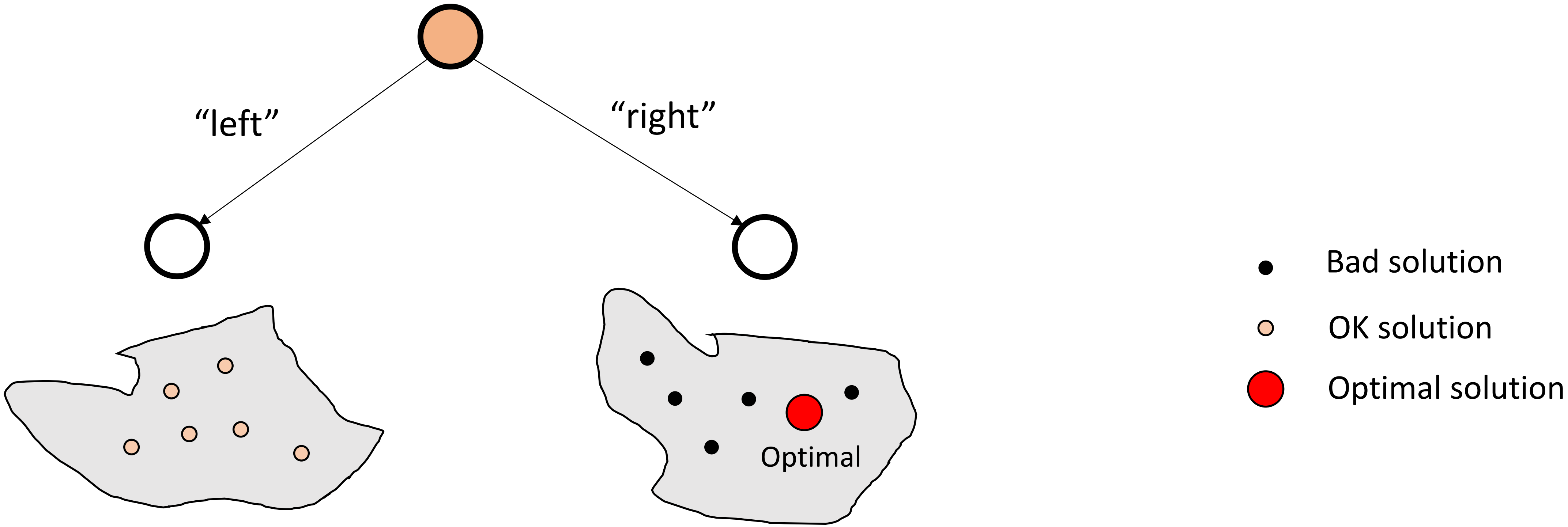


$$a_t = \arg\max_a Q(s_t, a) + u(s_t, a)$$

Exploration

$$u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

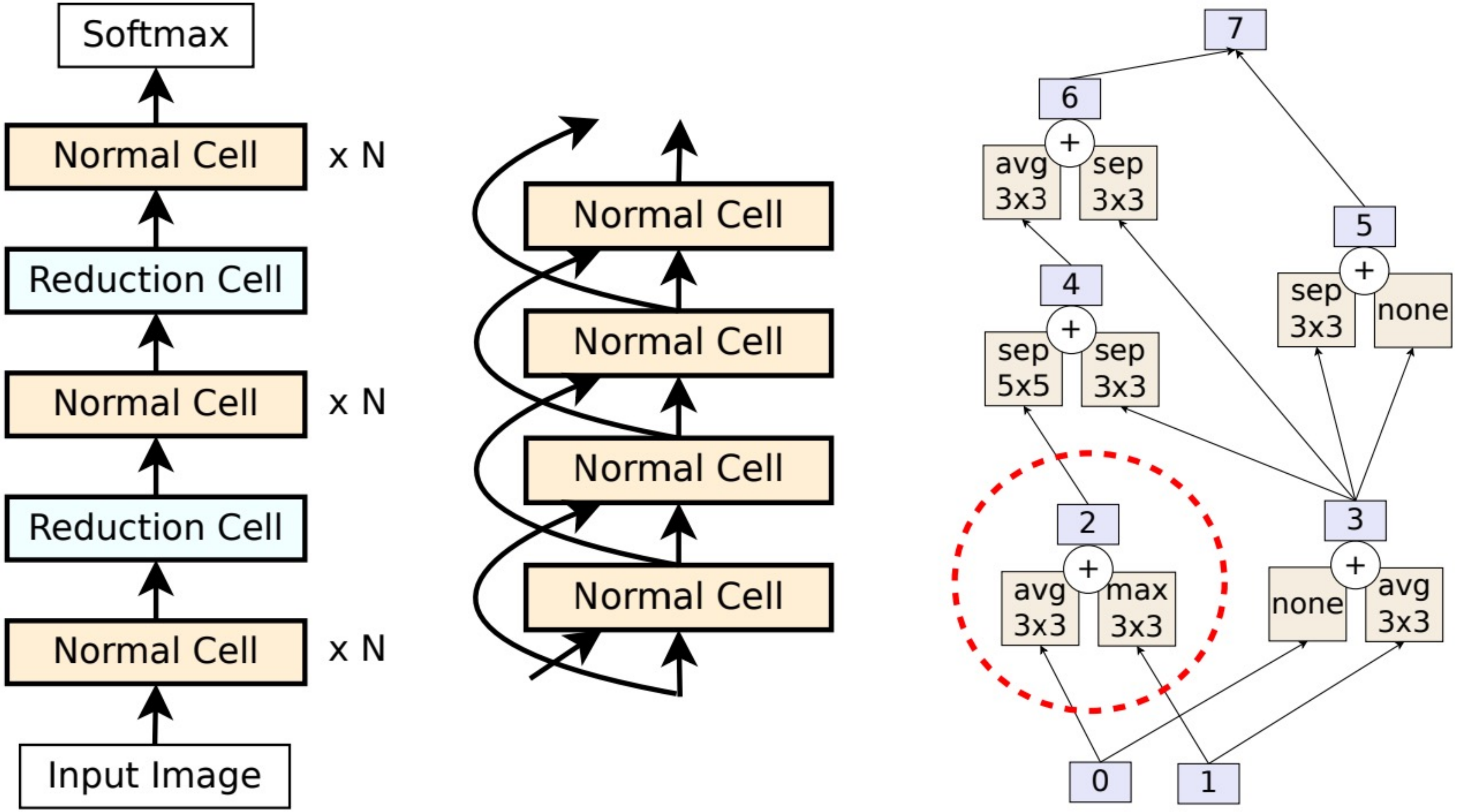
Why Exploration is Important



OK solutions
but not optimal

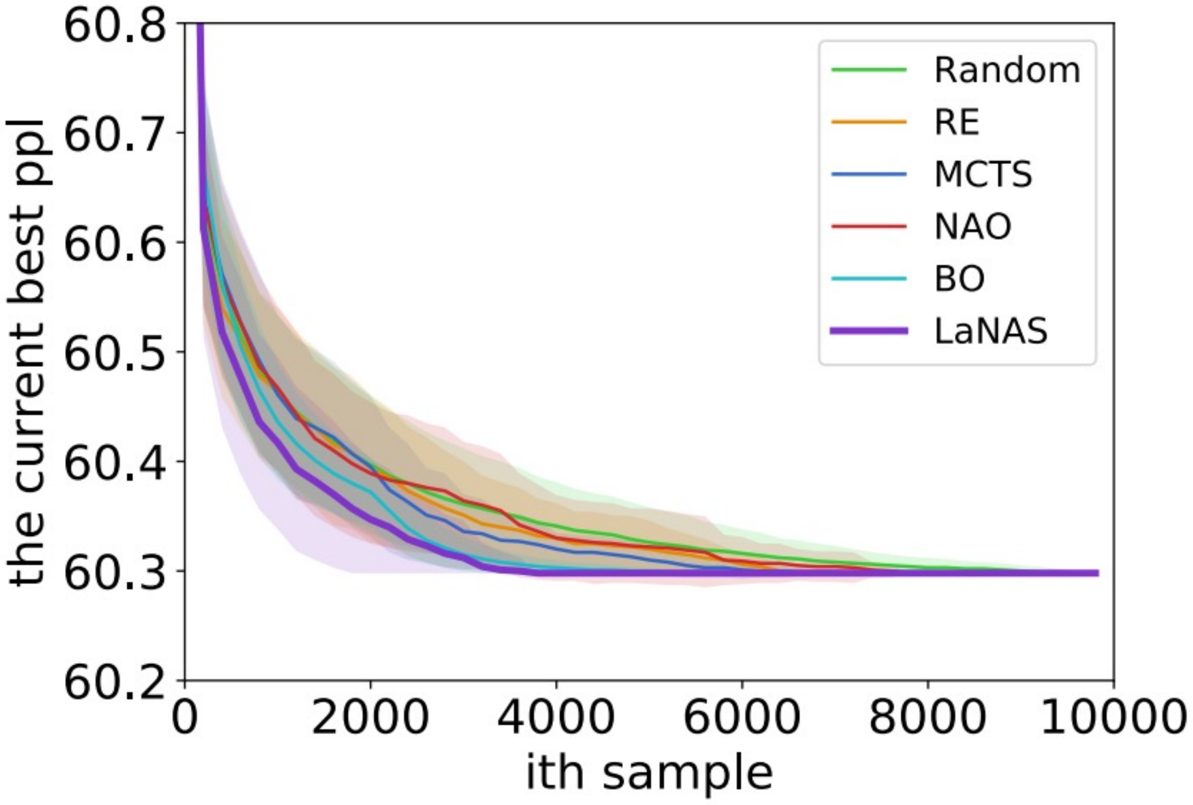
Most solutions are bad but there
exists an optimal one

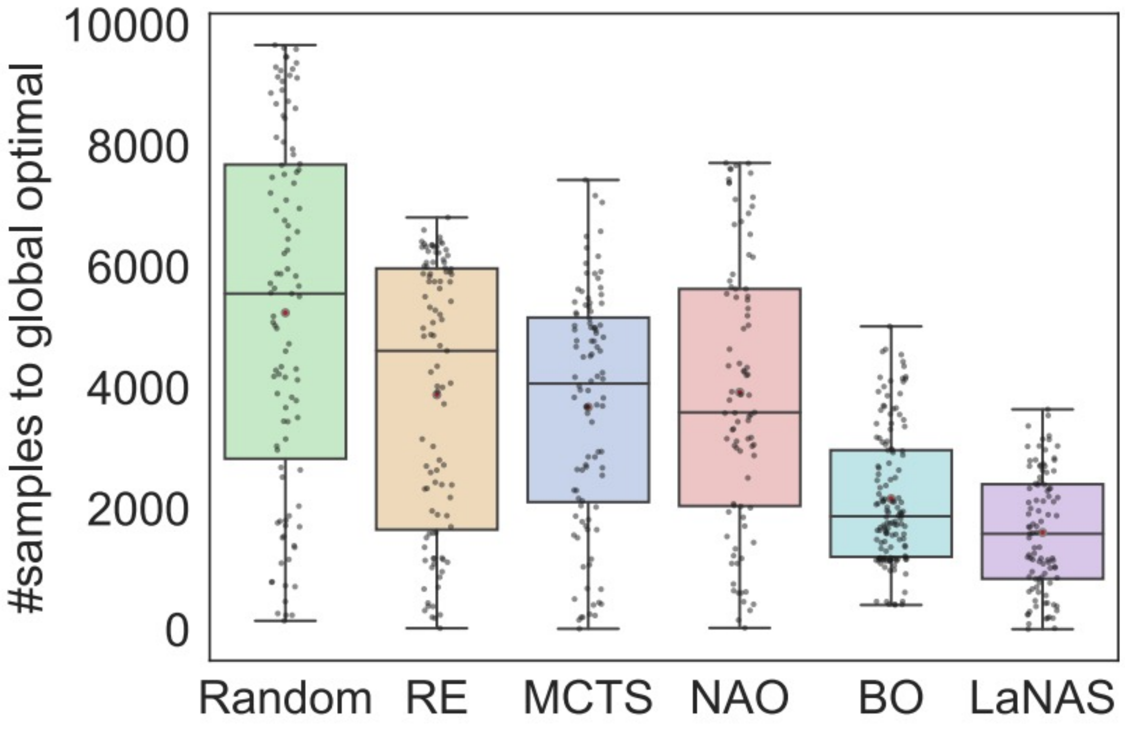
NASNet Search Space



Performance

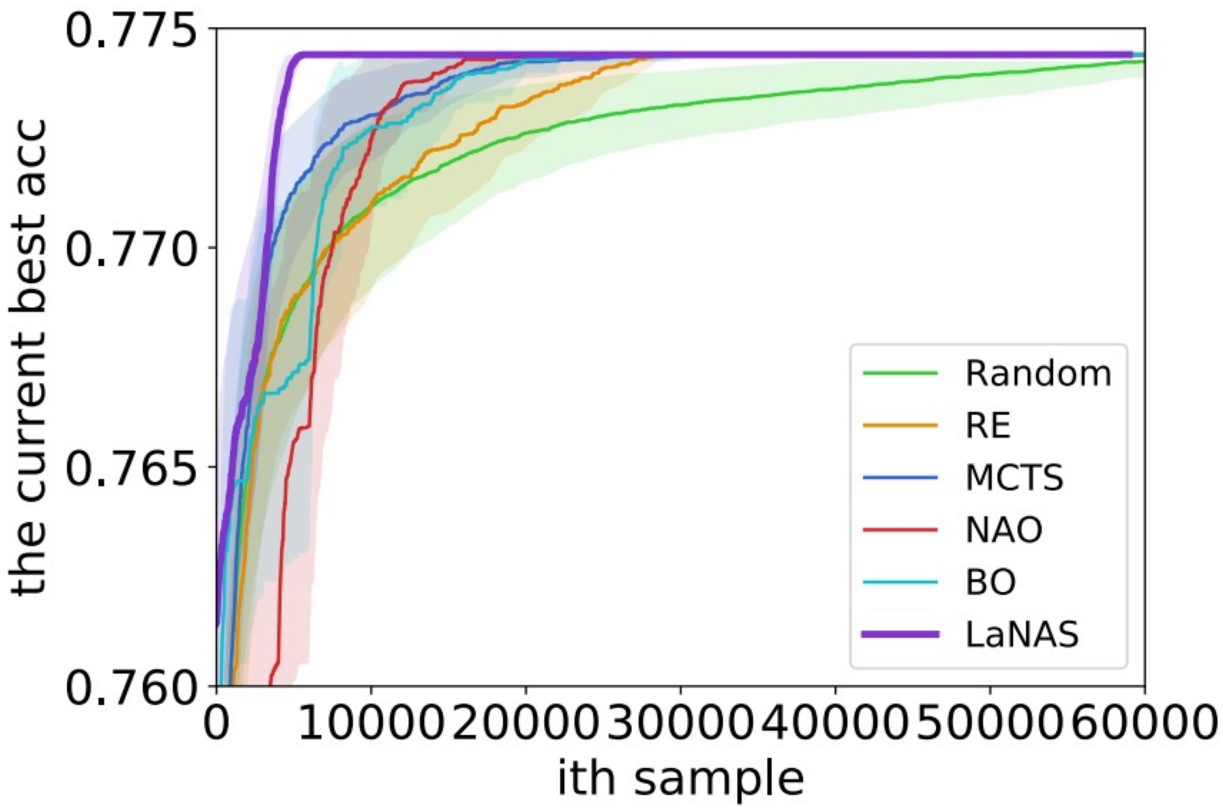
Customized dataset: LSTM-10K (PTB)

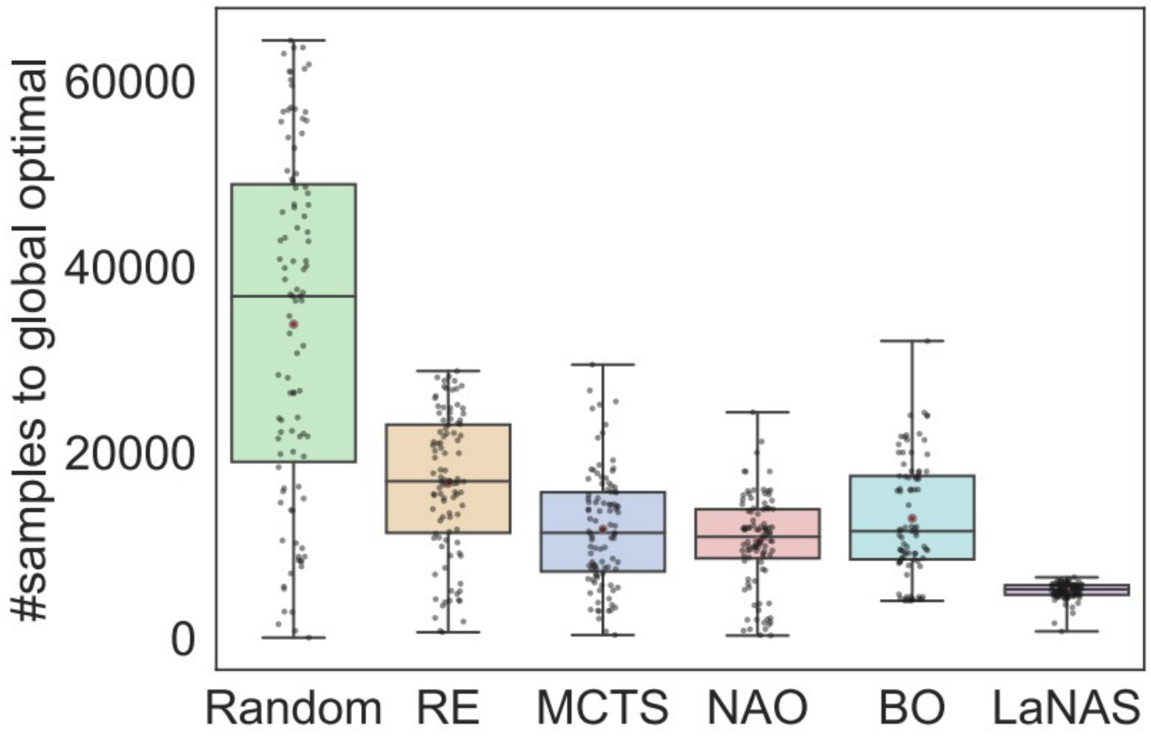




Performance

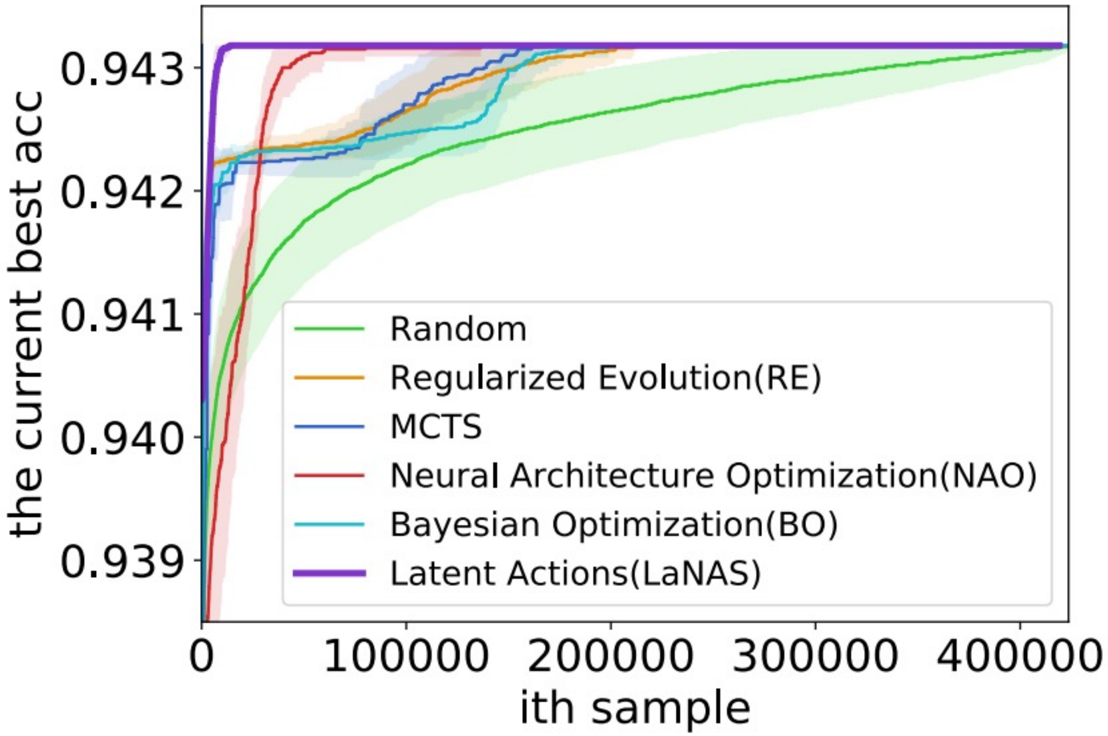
Customized dataset: ConvNet-60K (CIFAR-10, VGG style models)

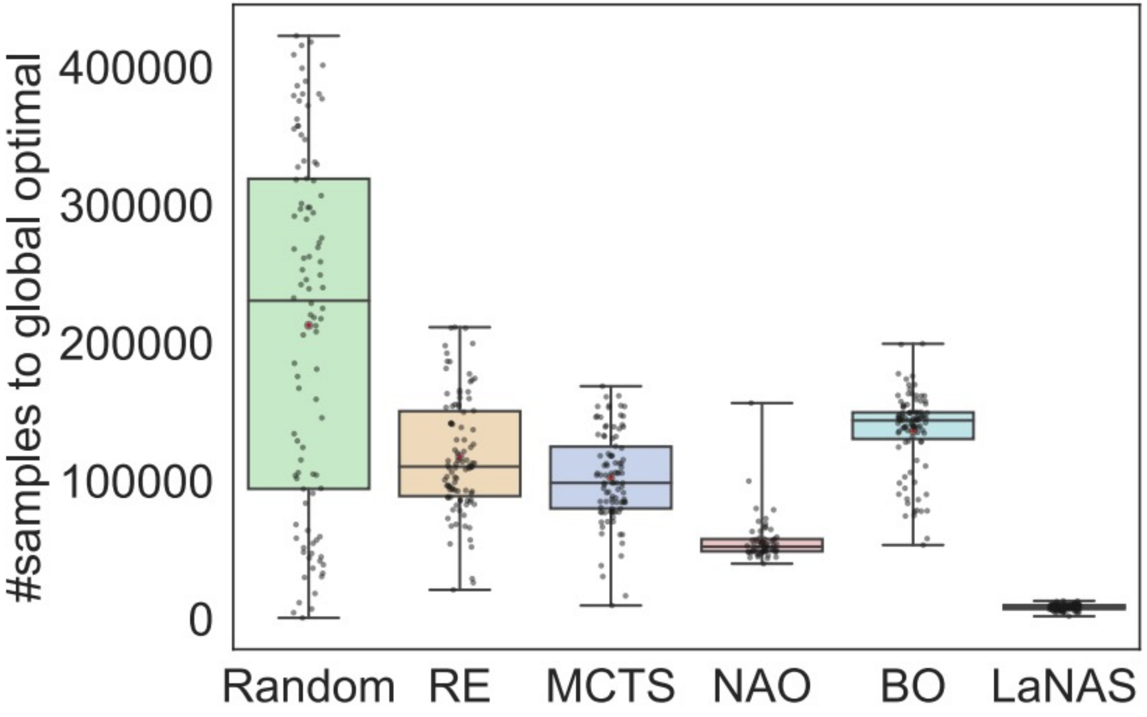




Performance

NASBench-101 (CIFAR-10, 420k models, NASNet Search Space)





Each curve is repeated 100 times. We randomly pick 2k models to initialize.

Open Domain

CIFAR-10 (NASNet style architecture)

Model	Using ImageNet	Params	Top1 err	M	GPU days
		search based methods			
NASNet-A+c/o [22]	X	3.3 M	2.65	20000	2000
AmoebaNet-B+c/o [10]	X	2.8 M	$2.55_{\pm 0.05}$	27000	3150
PNASNet-5 [29]	X	3.2 M	$3.41_{\pm 0.09}$	1160	225
NAO+c/o [30]	X	128.0 M	2.11	1000	200
AmoebaNet-B+c/o	X	34.9 M	$2.13_{\pm 0.04}$	27000	3150
EfficientNet-B7	√	64M	1.01		
BiT-M	√	60M	1.09		
LaNet+c/o	X	3.2 M	$\mathbf{1.63}_{\pm 0.05}$	800	150
LaNet+c/o	X	44.1 M	$\mathbf{0.99}_{\pm 0.02}$	800	150
		one-shot NAS based methods			
ENAS+c/o [18]	X	4.6 M	2.89	-	0.45
DARTS+c/o [20]	X	3.3 M	$2.76_{\pm 0.09}$	-	1.5
BayesNAS+c/o [31]	X	3.4 M	$2.81_{\pm 0.04}$	-	0.2
ASNG-NAS+c/o [32]	X	3.9 M	$2.83_{\pm 0.14}$	-	0.11
XNAS+c/0 [33]	X	3.7 M	1.81		0.3
oneshot-LaNet+c/o	X	3.6 M	$\mathbf{1.68}_{\pm 0.06}$	-	3
oneshot-LaNet+c/o	X	45.3 M	$\mathbf{1.2}_{\pm 0.03}$	-	3

M: number of samples selected.

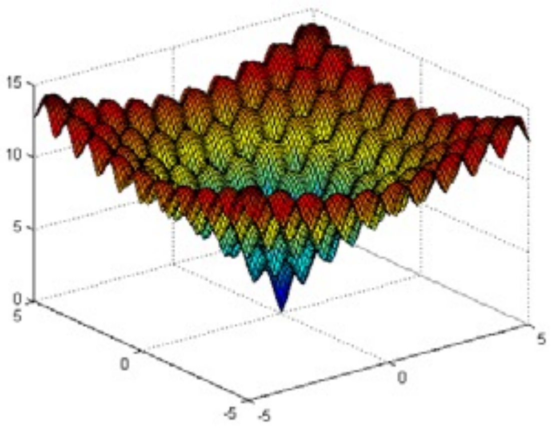
Open Domain

ImageNet
(mobile setting
Flop < 600M)

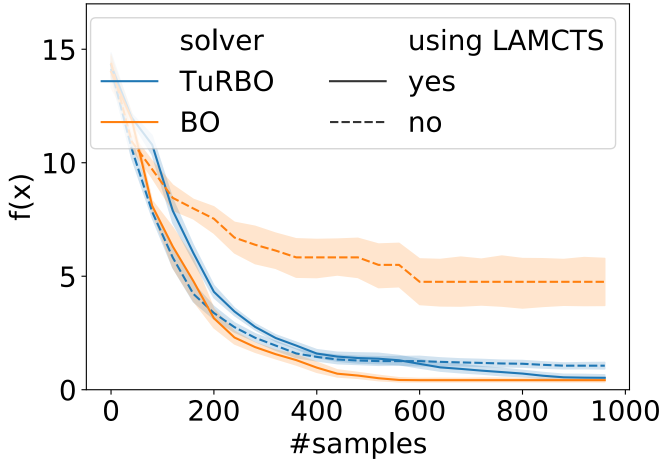
Model	FLOPs	Params	top1 / top5 err
NASNet-A (Zoph et al. (2018))	564M	5.3 M	26.0 / 8.4
NASNet-B (Zoph et al. (2018))	488M	5.3 M	27.2 / 8.7
NASNet-C (Zoph et al. (2018))	558M	4.9 M	27.5 / 9.0
AmoebaNet-A (Real et al. (2018))	555M	5.1 M	25.5 / 8.0
AmoebaNet-B (Real et al. (2018))	555M	5.3 M	26.0 / 8.5
AmoebaNet-C (Real et al. (2018))	570M	6.4 M	**24.3 / 7.6**
PNASNet-5 (Liu et al. (2018a))	588M	5.1 M	25.8 / 8.1
DARTS (Liu et al. (2018b))	574M	4.7 M	26.7 / 8.7
FBNet-C (Wu et al. (2018))	375M	5.5 M	25.1 / -
RandWire-WS (Xie et al. (2019))	583M	5.6 M	25.3 / 7.8
BayesNAS (Zhou et al. (2019))	-	3.9 M	26.5 / 8.9
LaNet	570M	5.1 M	**25.0 / 7.7**

La-MCTS as a meta method

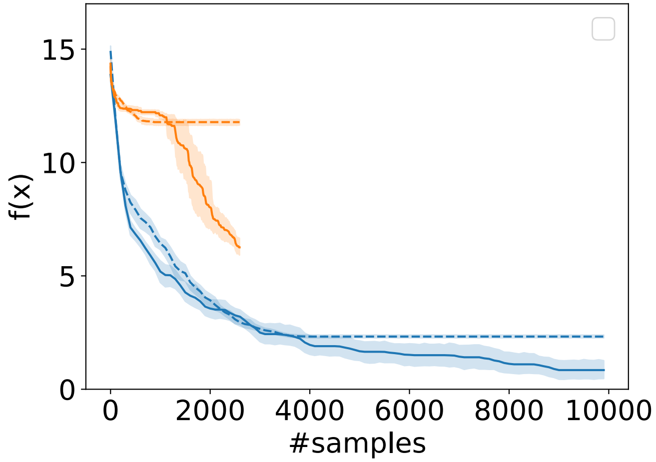
$$x^* = \arg\min_{x \in \Omega} f(x)$$



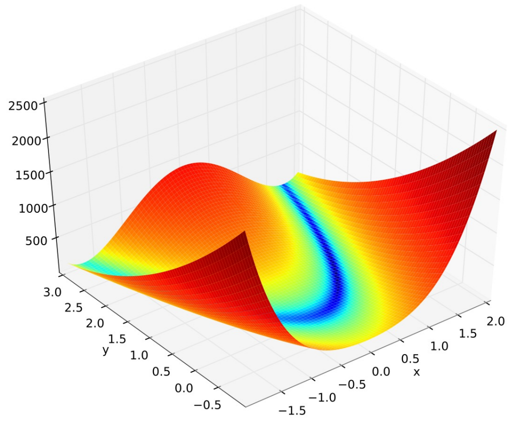
Ackley-2d



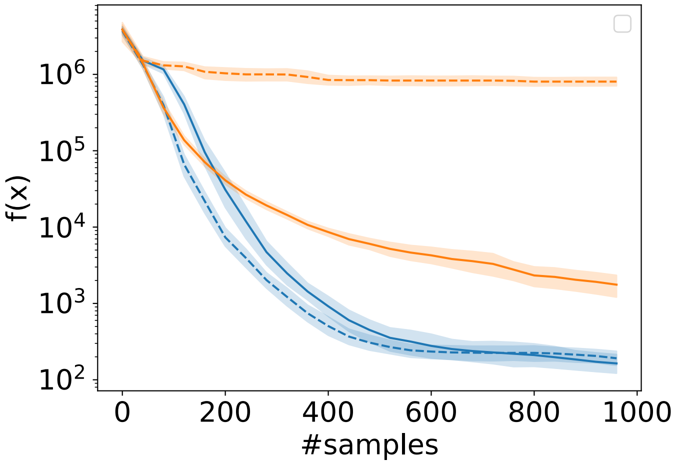
Ackley-20d



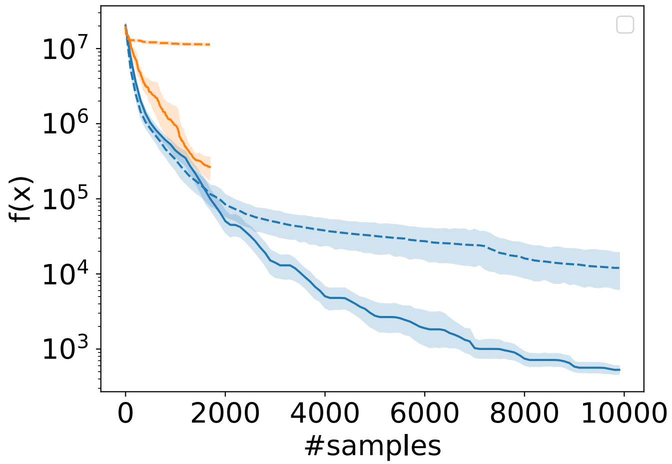
Ackley-100d



Rosenbrock-2d

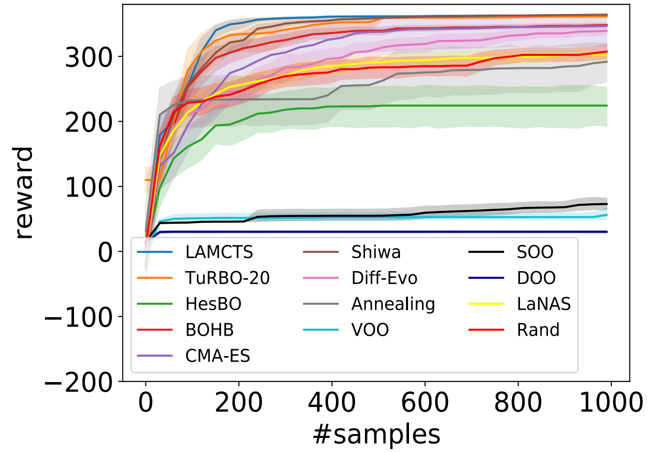


Rosenbrock-20d

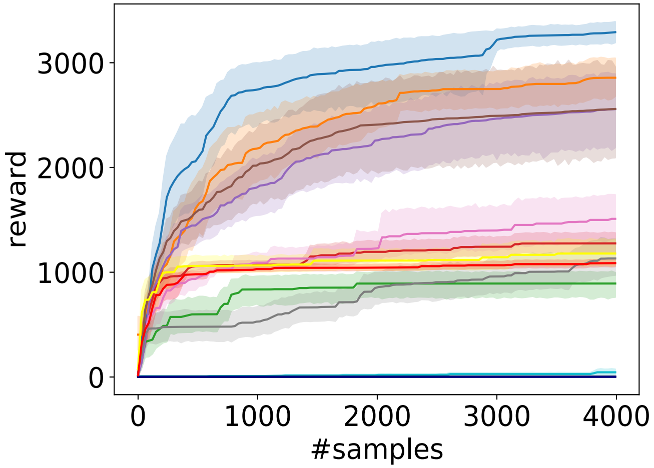


Rosenbrock-100d

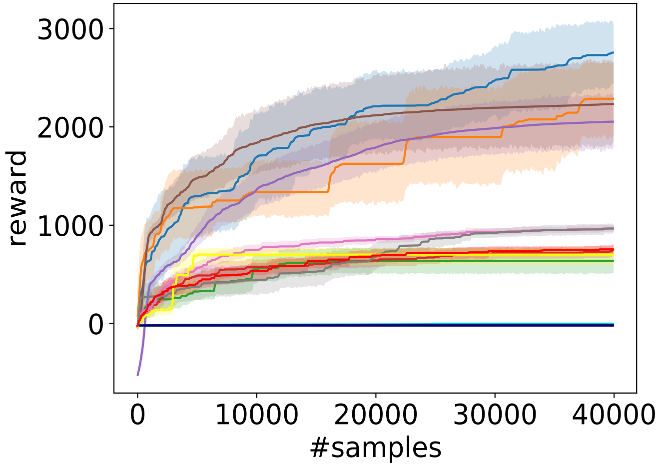
Optimizing linear policy for Mujoco tasks



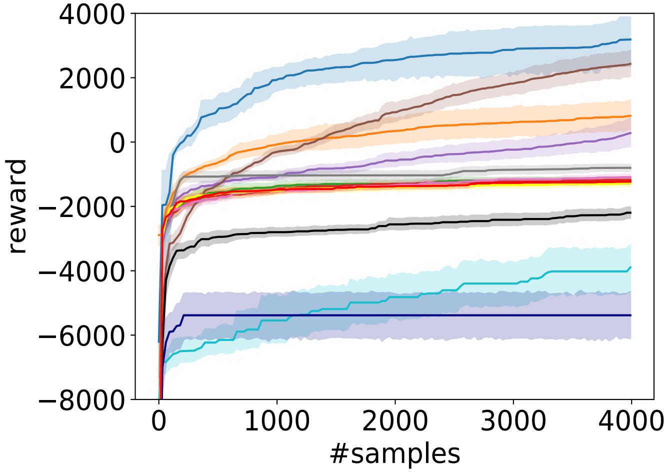
(a) Swimmer, #params = 16



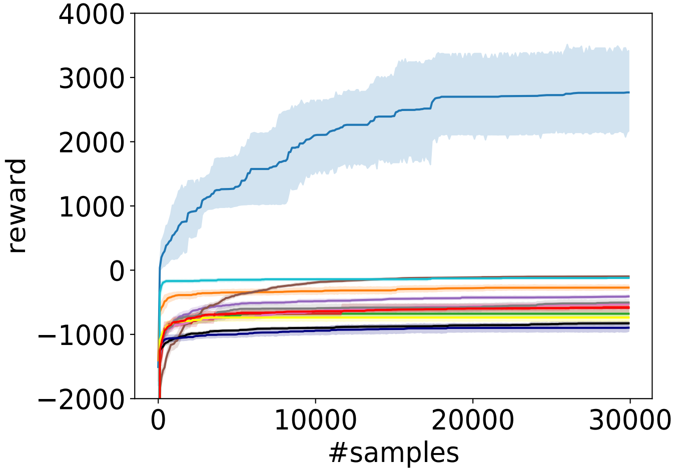
(b) Hopper, #params = 33



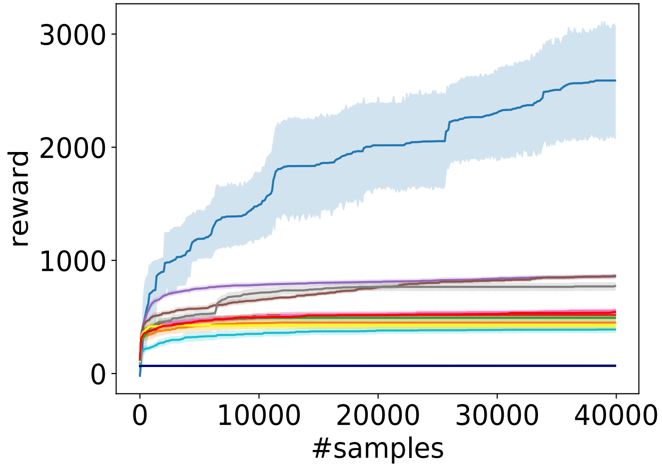
(c) Walker-2d, #params = 102



(d) Half-Cheetah, #params =102



(e) Ant, #params = 888



(f) Humanoid, #params = 6392

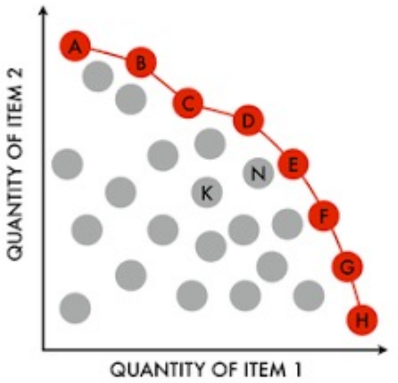
Limitations

Task	Reward Threshold	The average episodes (#samples) to reach the threshold				
		LA-MCTS	**ARS V2-t [54]**	**NG-lin [55]**	**NG-rbf [55]**	**TRPO-nn [54]**
Swimmer-v2	325	**132**	427	1450	1550	N/A
Hopper-v2	3120	2897	1973	13920	8640	10000
HalfCheetah-v2	3430	3877	1707	11250	6000	4250
Walker2d-v2	4390	N/A($r_{best} = 3314$)	24000	36840	25680	14250
Ant-v2	3580	N/A($r_{best} = 2791$)	20800	39240	30000	73500
Humanoid-v2	6000	N/A($r_{best} = 3384$)	142600	130000	130000	unknown

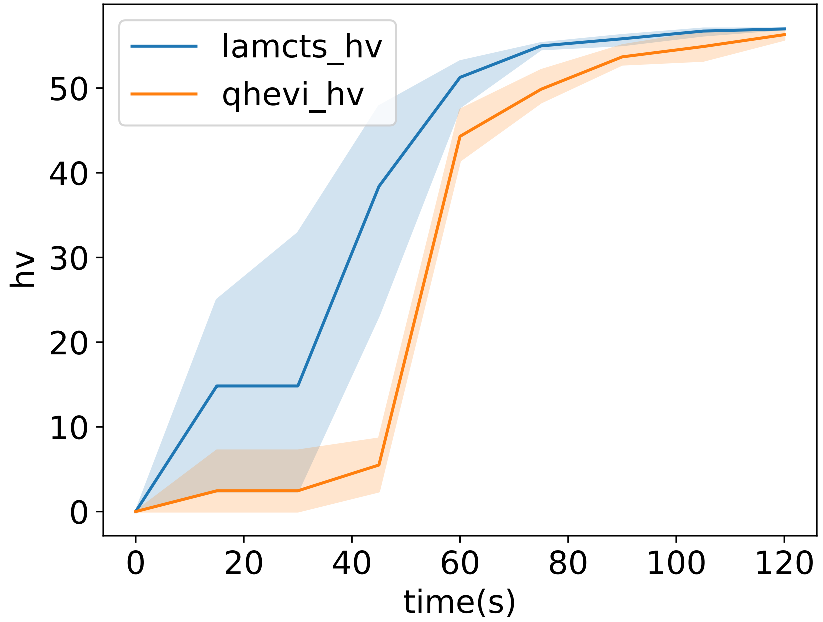
N/A stands for not reaching reward threshold.
r_{best} stands for the best reward achieved by LA-MCTS under the budget in Fig. 3.

Too many explorations might hurt in Mujoco tasks.

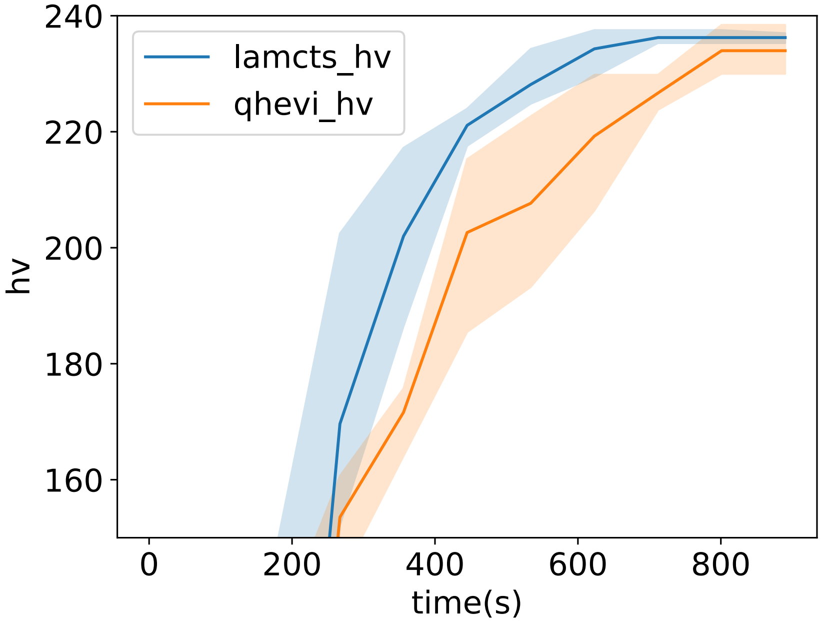
Multi-Objective Optimization



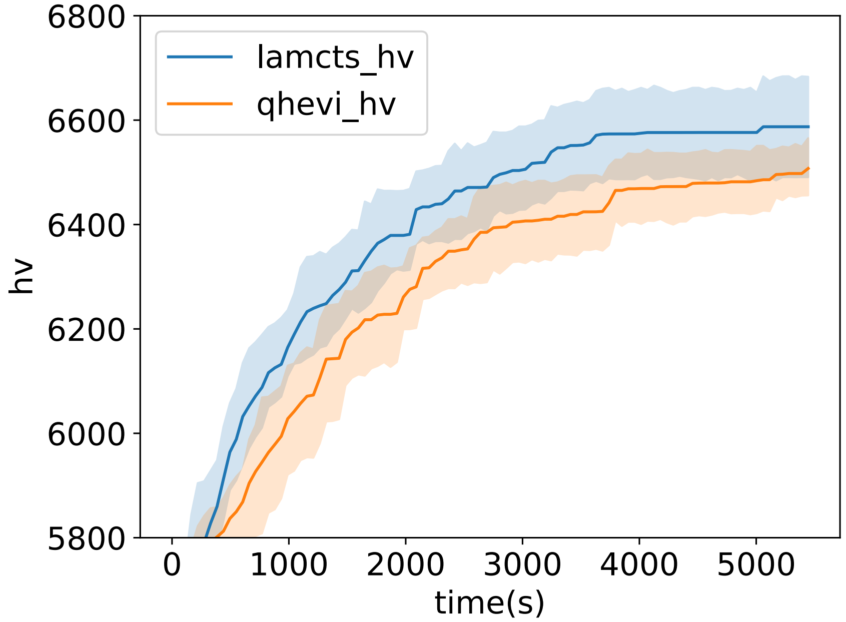
HV: Hyper Volume of the Pareto Frontier



Branin-Currin problem
(2 objective)



Vehicle Safety
(3 objective)



Waveguide

qEHVI: https://arxiv.org/pdf/2006.05078.pdf

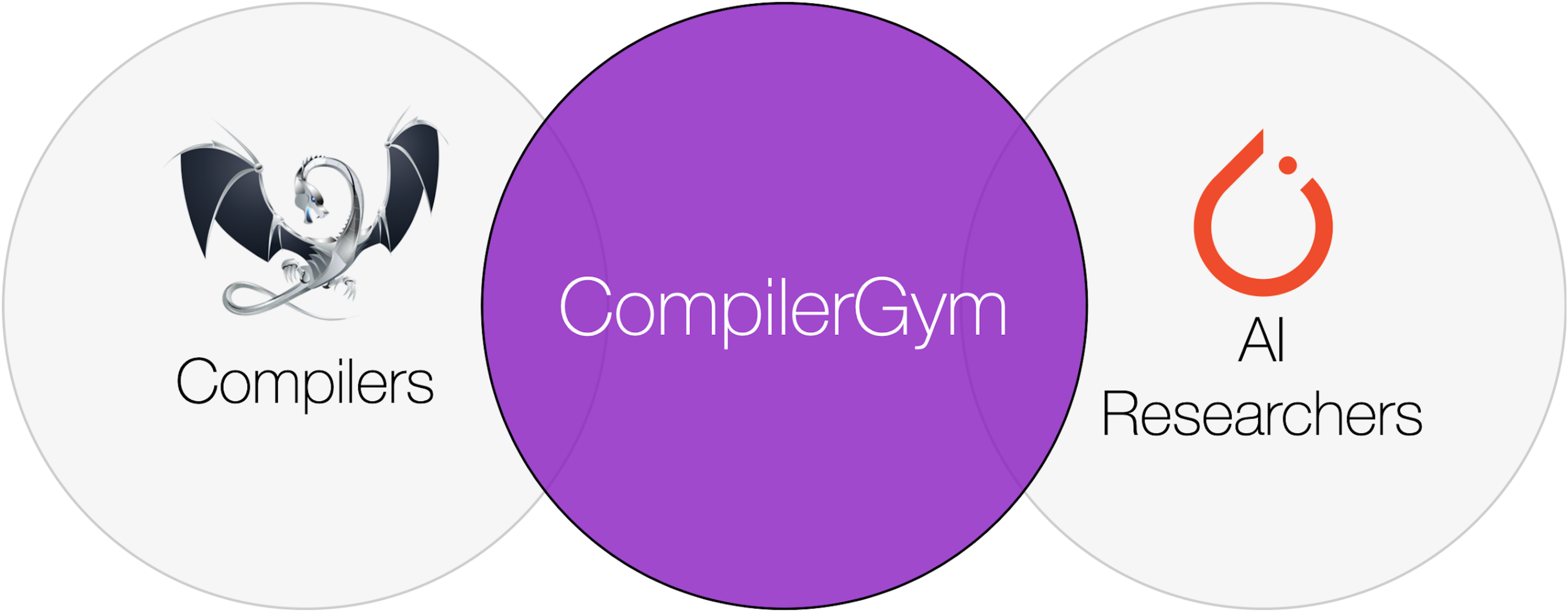
Code is public now!

LA-MCTS

https://github.com/facebookresearch/LaMCTS

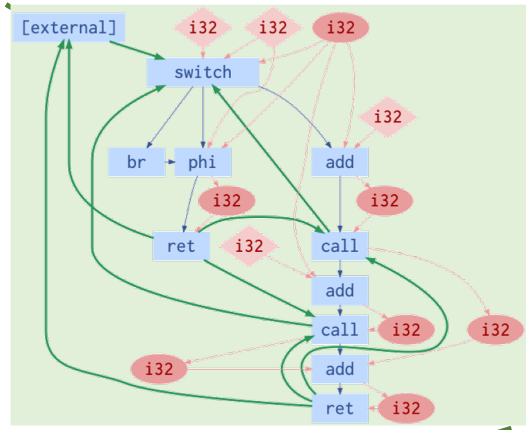
Both 3rd and 8th teams in NeurIPS 2020 Black-box optimization competition use our method!

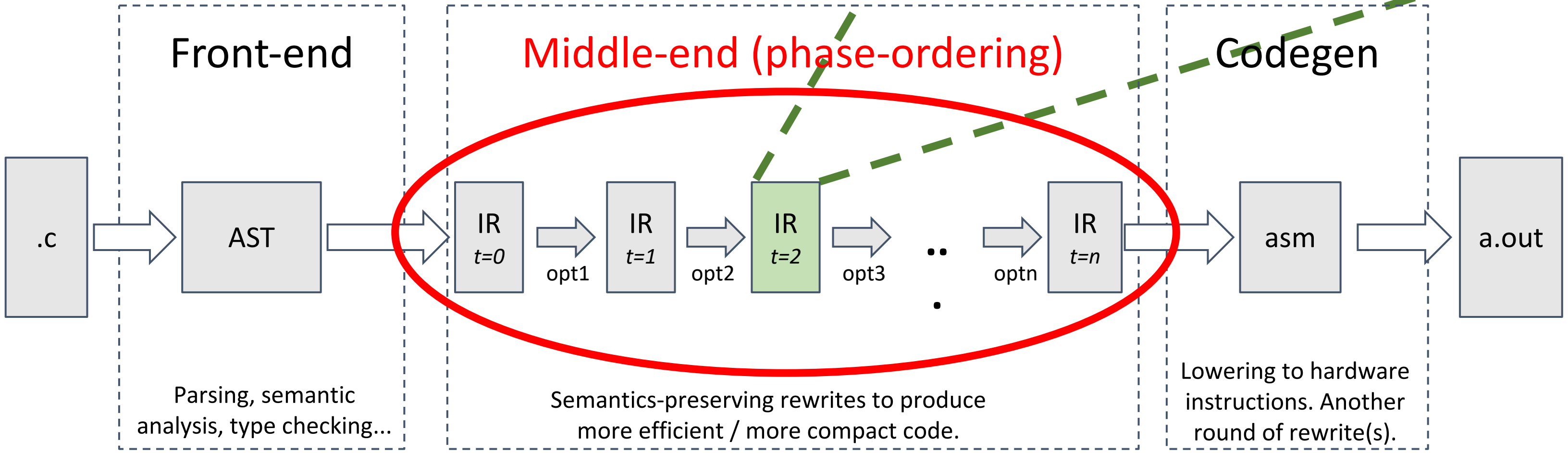
https://github.com/facebookresearch/CompilerGym



Robust, high-performance reinforcement learning environments for compiler optimization tasks

CompilerGym





* not even slightly to scale

An iterative decision-making process

Challenges
1. Huge state and action space
2. Many irrelevant actions
3. Graph-structured observations
4. Learned policy needs to transfer well

Goals

1. Lower the barrier to entry to AI for compilers research.
2. Provide common benchmarks for compiler optimization tasks.
   - e.g. "ImageNet for Compilers", CodeXGLUE for performance.
3. Advance the state-of-the-art in AI for compilers

Long term

1. Enable every single compiler decision to be controlled by an agent.
2. Build a family of "SysML Gyms" and tools for making new ones.

There are a lot of Programs available

Dataset	License	Num. Benchmarks	Validatable? [1]	Difficulty [2]
blas-v0	BSD 3-Clause	300	No	0.3
cBench-v1	BSD 3-Clause	23	Partial	0.8
github-v0	CC BY 4.0	50,708	No	0.7
linux-v0	GPL-2.0	13,920	No	0.4
mibench-v0	BSD 3-Clause	40	No	0.8
npb-v0	NASA v1.3	122	No	0.4
opencv-v0	Apache 2.0	442	No	0.3
poj104-v0	BSD 3-Clause	49,628	No	0.7
tensorflow-v0	Apache 2.0	1,985	No	0.3

Leader Board

LLVM Instruction Count

Author	Algorithm	Links	Date	Walltime (mean)	Codesize Reduction (geomean)
Facebook	Random search (t=10800)	write-up, results	2021-03	10,512.356s	**1.062×**
Facebook	Random search (t=3600)	write-up, results	2021-03	3,630.821s	1.061×
Facebook	Greedy search	write-up, results	2021-03	169.237s	1.055×
Facebook	Random search (t=60)	write-up, results	2021-03	91.215s	1.045×
Facebook	e-Greedy search (e=0.1)	write-up, results	2021-03	152.579s	1.041×
Jiadong Guo	Tabular Q (N=5000, H=10)	write-up, results	2021-04	2534.305	1.036×
Facebook	Random search (t=10)	write-up, results	2021-03	**42.939s**	1.031×
Jiadong Guo	Tabular Q (N=2000, H=5)	write-up, results	2021-04	694.105	0.988×

Summarization and Future Works

- Summary
  - Machine Learning can be used to learn heuristics for optimization problems.
  - Many system problem are optimization problems
  - Use ML to make the system smarter ☺

- Many Challenges ahead
  - Huge state / action space.
  - Irrelevant actions
  - Slow evaluation (sim2real problem)

Thanks!