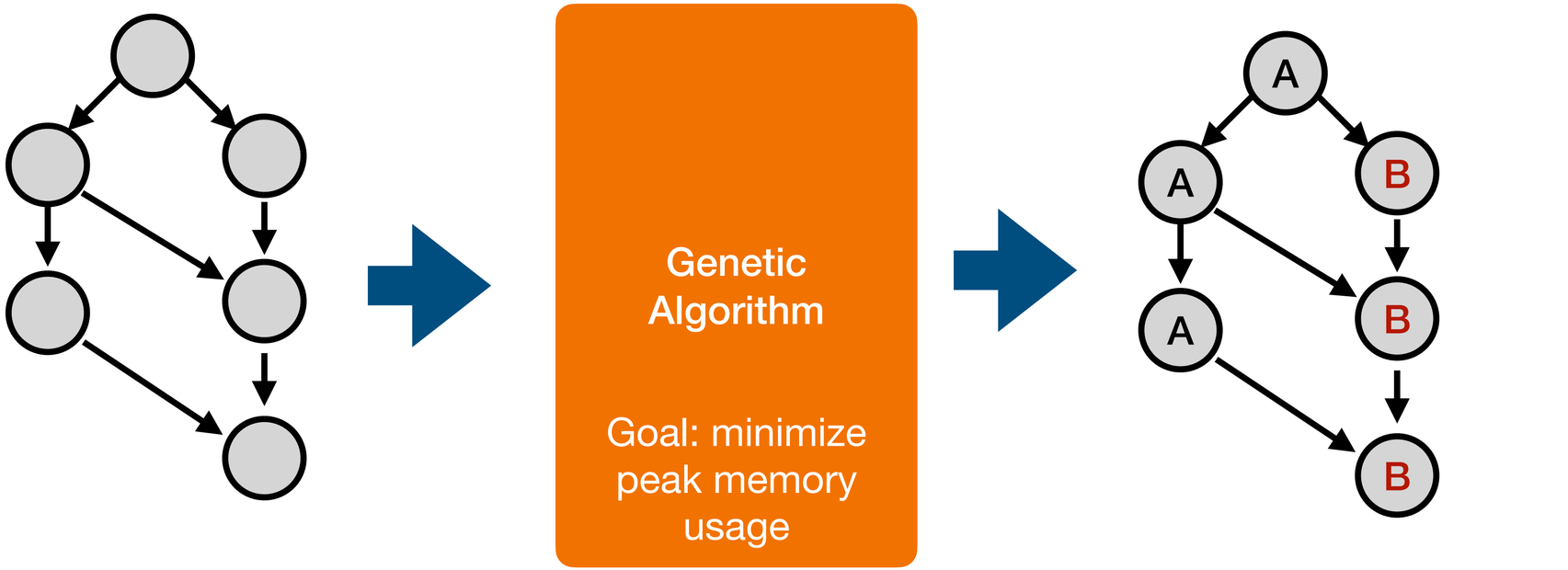
REGAL: Transfer Learning for Fast Optimization of Computation Graphs

Aditya Paliwal, Felix Gimeno, Vinod Nair, Yujia Li, Miles Lubin, Pushmeet Jhli, Oriol Vinyals

Zak Singh 08/11/21

REGAL optimizes **device placement** and **scheduling** for **static** computation graphs

A simple approach: Genetic Algorithm

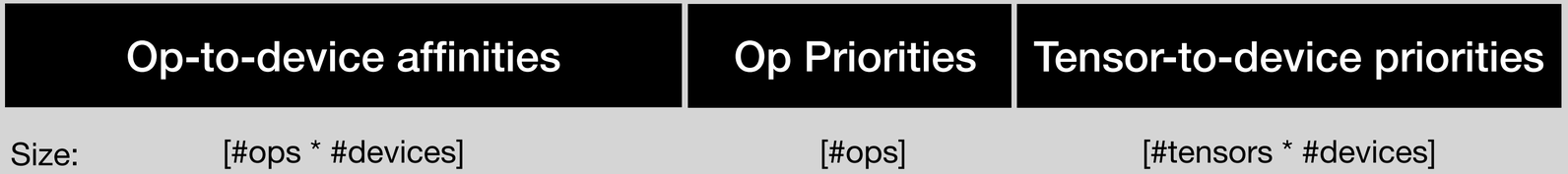


Computation graph

Placement, schedule

Biased Random-Key Genetic Algorithm

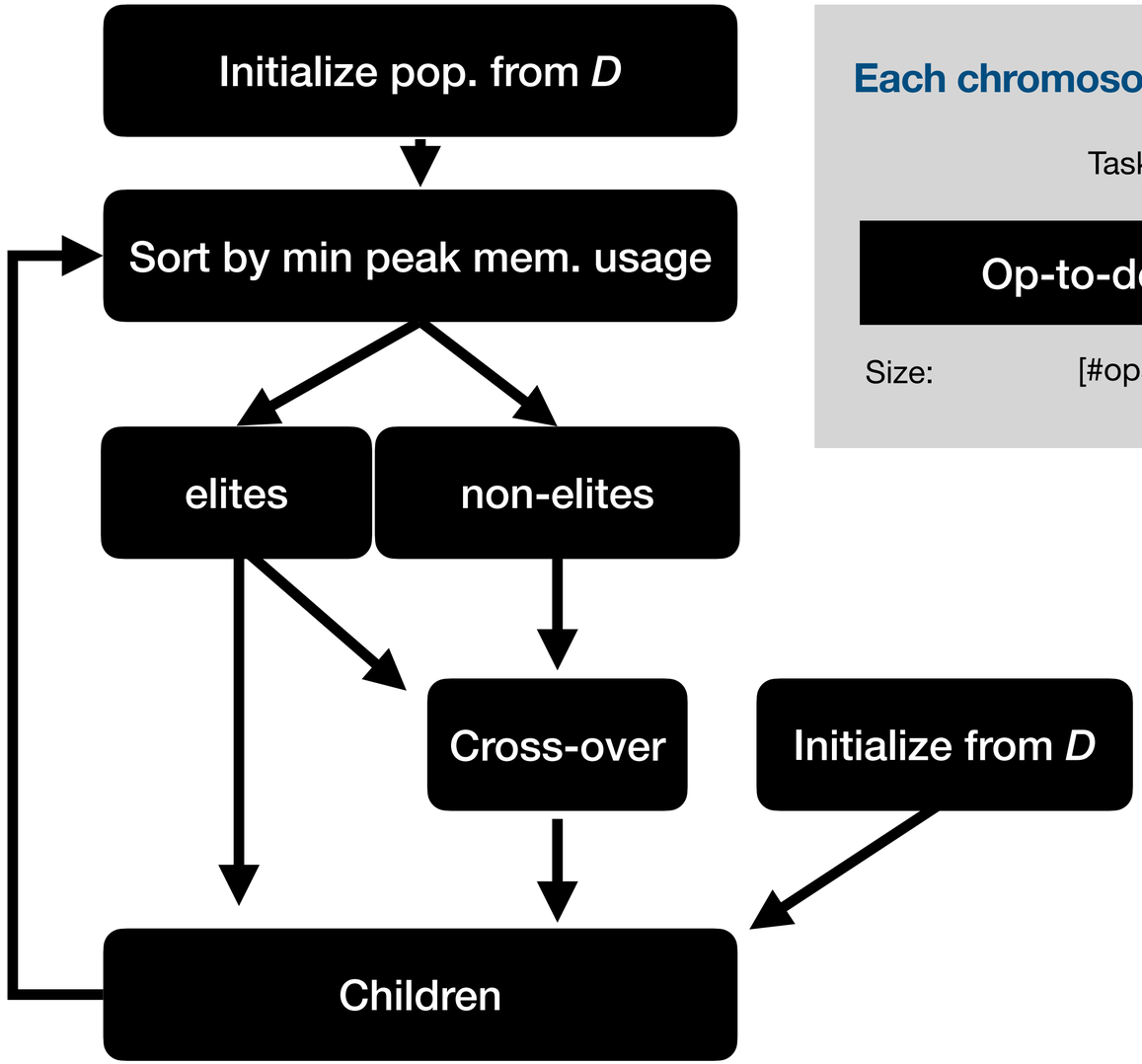
Each chromosome (i.e. schedule/placement):



- Made up of s=[devices*(ops + tensors) + ops] *genes*
- Each gene a float b/w 0 and 1

0.12	0.53	0.34	0.10	0.79	…

Biased Random-Key Genetic Algorithm (BRKGA)

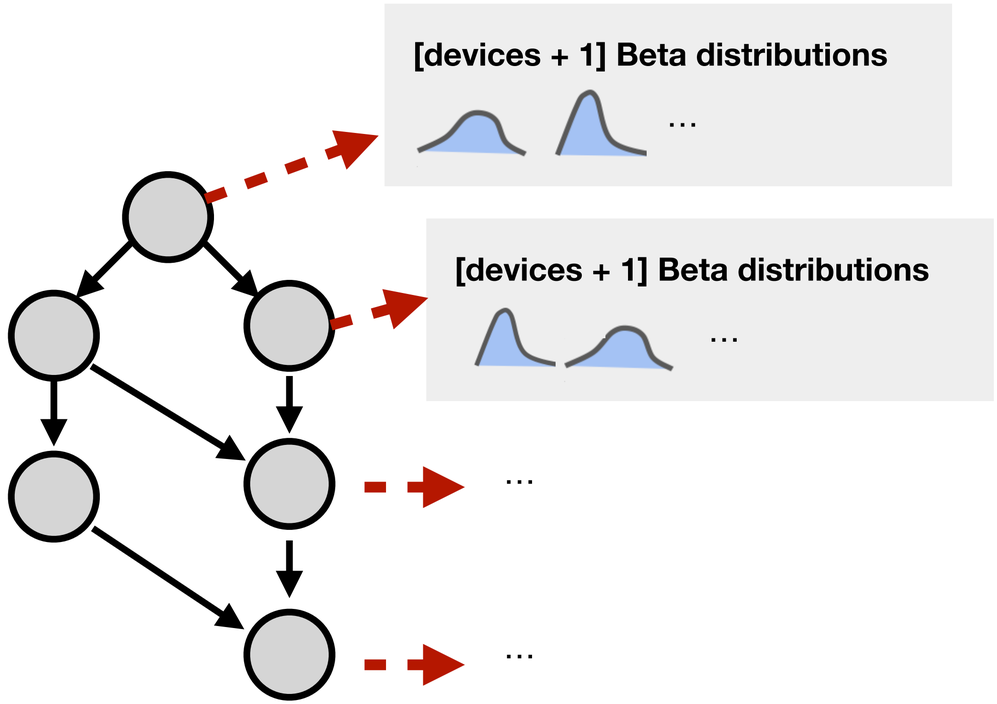


Each chromosome (i.e. schedule/placement):

	Task placement	Scheduling	Tensor placement
	Op-to-device affinities	Op Priorities	Tensor-to-device priorities
Size:	[#ops * #devices]	[#ops]	[#tensors * #devices]

- *D* is a set of Beta distributions (1 for each gene) from which each new chromosome is sampled
- *Elite Bias:* For cross-over, each gene of the elite has a probability p in [0.5, 1) of the offspring inheriting that gene

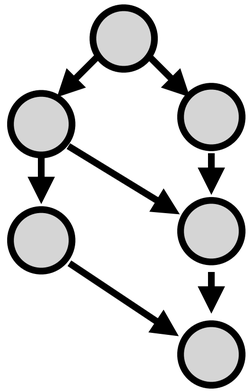
BRKGA Params, Revisualized



Each node (aka “op”) is mapped to [devices + 1] Beta distributions, corresponding to its device affinities and scheduling priority

Computation graph

How it works



Computation graph
(to be optimized)

Graph Neural Network

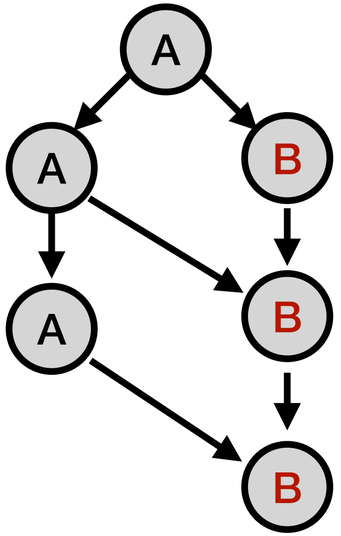
Output:
representations of nodes

Deep-RL Policy Network

Output:
GA params

BRKGA (Genetic Algorithm)

Goal: minimize memory usage

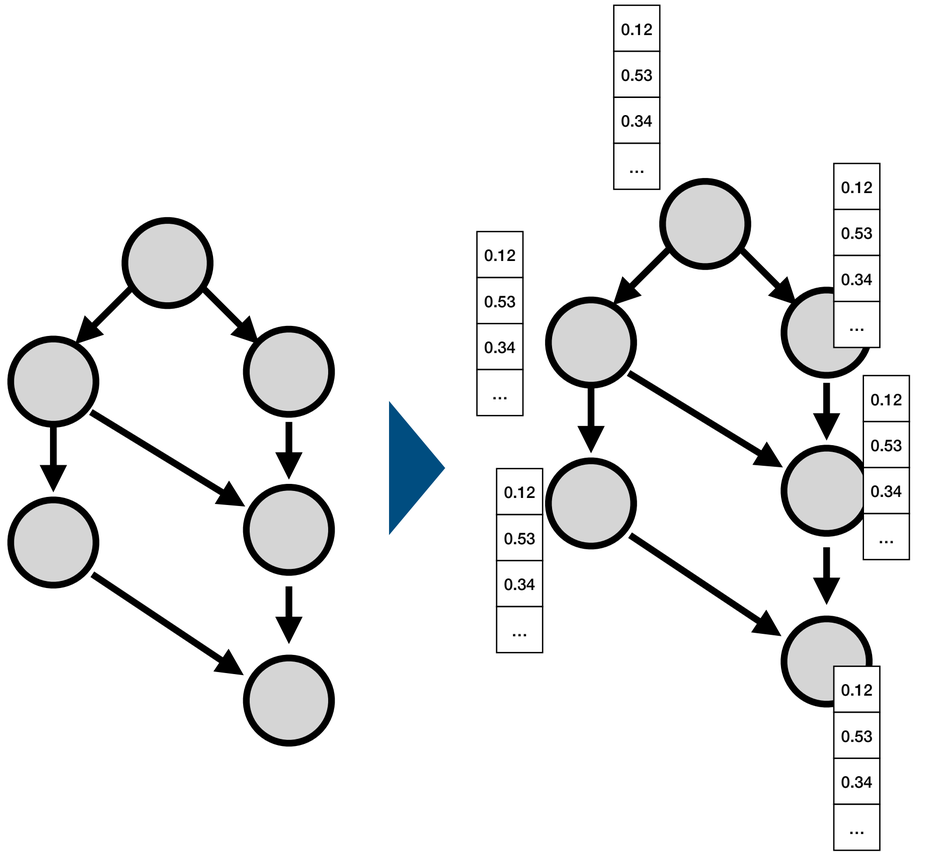


Placement,
Schedule

Contribution (metaoptimizer)

Designing the metaoptimizer

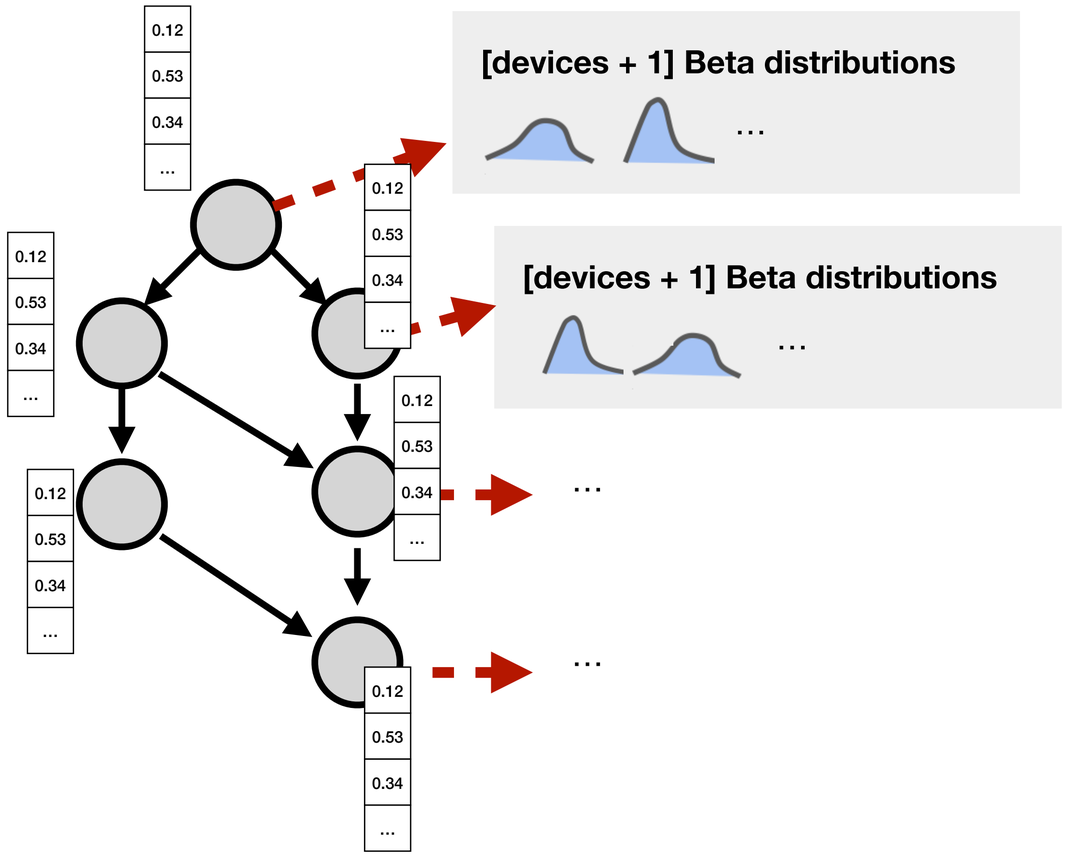
- **We want each node in the computation graph to be mapped to a representation which encodes the structure of its neighborhood**
  - Roughly: think convolution on an image (pixel gets weighted average value of its neighborhood)
  - Implemented via a Graph Neural Network with message-passing
  - *Why do we need this?* **To make the metaoptimizer generalizable to all graphs**
    - Nodes from different computation graphs with similar neighborhoods map to similar representations



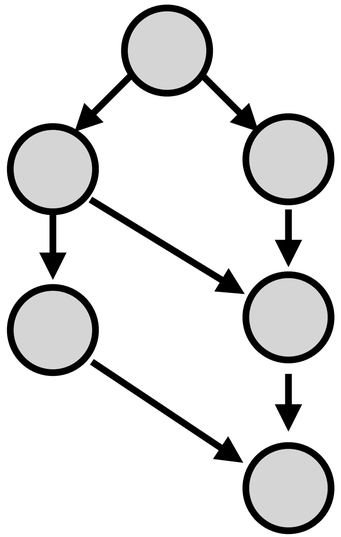
Learned Representations

Training the Policy Network

- Our policy network maps node representations (states) to a quantized mean and variance of the Beta distributions for that node (actions)
- The Beta distributions serve as our GA parameters
- To find the reward, we plug the outputted distributions into BRKGA and evaluate the resulting schedule and placement based on our maximum memory usage criteria
- The policy network is then updated via the REINFORCE algorithm (similar to gradient descent and backprop method we're familiar with)

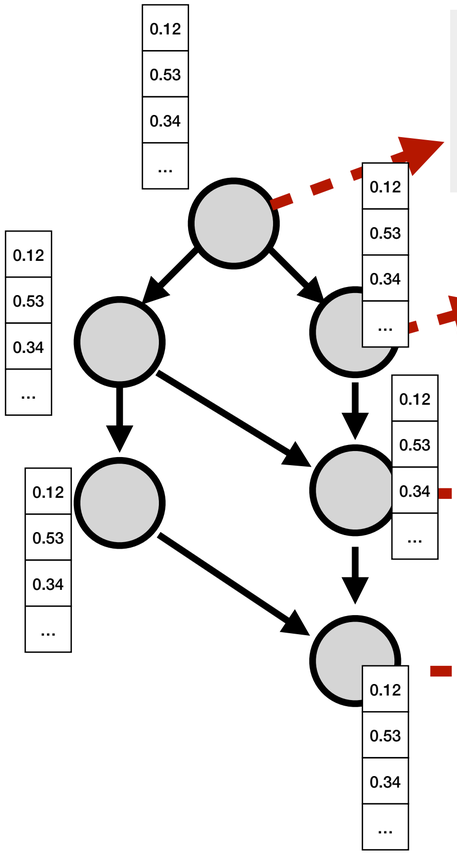


Metaoptimizer Pipeline



Computation graph

Graph Neural Network



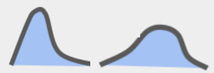
Learned Representations

Deep-RL Policy Network

[devices + 1] Beta distributions

...

[devices + 1] Beta distributions



...

...

...

Learned Policy

BRKGA (Genetic Algorithm)

Evaluation

- Used dataset of 372 unique real-world Tensorflow graphs to train
  - Each graph is augmented 100 times, altering the tensor sizes for each
- Tested on a 2-machine cluster

- Baselines:
  - **CP-SAT**: Guaranteed to find optimum with enough time
  - **BRKGA 5K:** BRKGA w/ with uniform distributions, ran for 5k iterations

Results

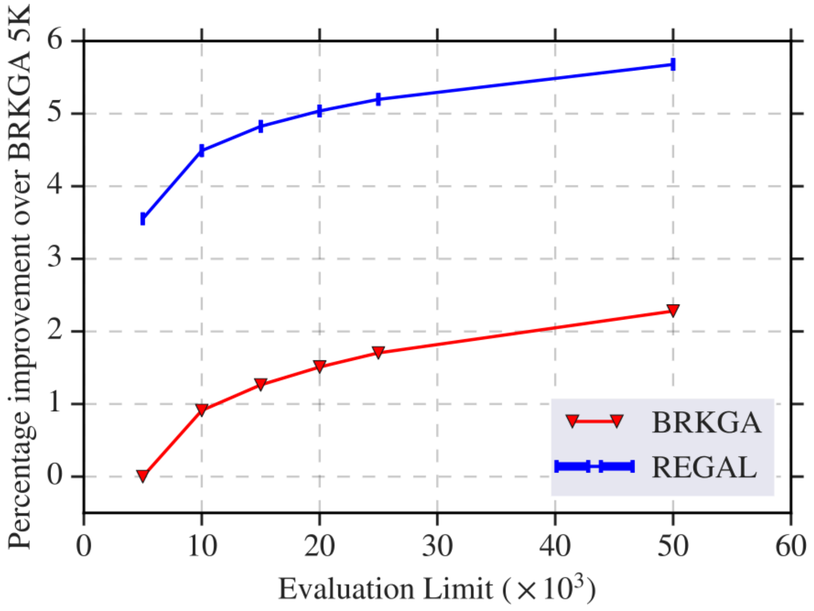


Table 1: Performance for all methods, averaged over the graphs in the test set of the TensorFlow and XLA datasets.

Algorithm	TensorFlow dataset (test)		XLA dataset ←	
	% Improv. over BRKGA5K	% Gap from best	% Improv. over BRKGA5K	% Gap from best
CP SAT	-1.77%	13.89%	-47.14%	71.35%
GP + DFS	-6.51%	16.63%	-21.43%	39.86%
Local Search	0.63%	8.65%	-6.69%	21.98%
BRKGA 5K	0%	9.65%	0%	14.04%
Tuned BRKGA	0.8%	8.54%	0.452%	13.52%
GAS	0.16%	9.33%	-1.1%	15.36%
REGAL	**3.56%**	**4.44%**	**3.74%**	**9.40%**

Not used for training
Avg. ~9X more nodes than TensorFlow dataset, ~13x more edges

Results

Table 2: Average running times for all methods.

Algorithm	TensorFlow dataset (test)	XLA dataset
CP SAT	~2 hours	12+ hours
GP + DFS	144 sec	500 sec
Local Search	122 sec	1343 sec
BRKGA 5K	0.89 sec	8.82 sec
Tuned BRKGA	1.04 sec	10.0 sec
GAS	1.04 sec	10.1 sec
REGAL	1.04 sec	10.1 sec

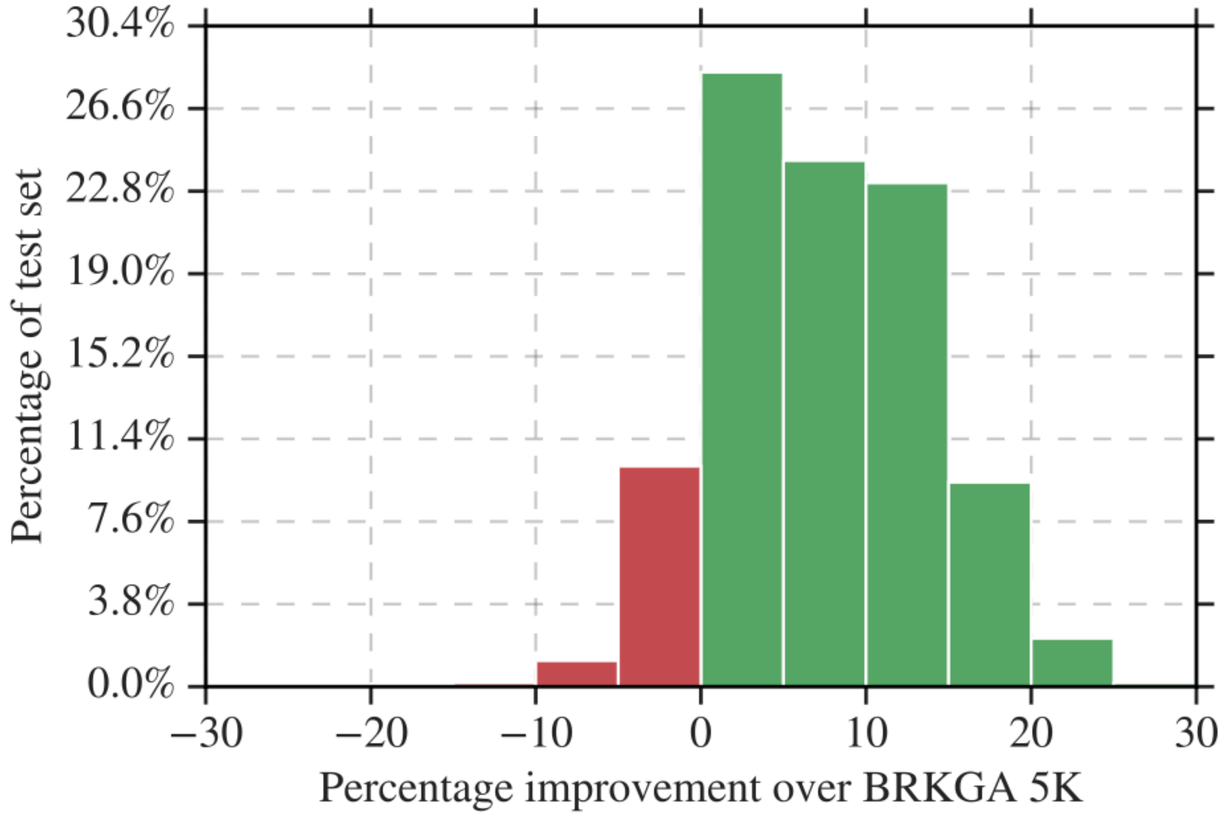
Results

- Elite biases during crossover resulted in worse performance than not using them!

Placement	Scheduling	Elite Bias	Valid	Test	XLA
Yes	No	No	-0.4%	-0.2%	-0.4%
No	Yes	No	4.4%	**3.65%**	1%
Yes	Yes	No	**4.67%**	3.56%	**3.74%**
Yes	No	Yes	-1.53%	-1.1%	-2.2%
No	Yes	Yes	2.47%	1.4%	-0.4%
Yes	Yes	Yes	2.58%	1.88%	-0.7%

Results

- Performance not always better



Criticism

- Overly complex — buzzword soup? (Graph Neural Networks, Deep Reinforcement Learning, Genetic Algorithms, etc.)
- Some aspects of implementation details, hyperparams are unexplained (especially for the RL agent)
- Only 3.56% reduced memory usage— worth the trouble?
- More exhaustive benchmarks
  - More than 2 devices
  - Optimize for a different metric (execution time?)

In conclusion

- REGAL jointly addresses the placement and scheduling problems for static computation graphs
- It does so by adding a metaoptimizer on top of BRKGA (a genetic algorithm)
- The metaoptimizer consists of:
  - a GNN to learn node representations which encode graph structure,
  - a deep RL Policy Network which converts each node representation into Beta distributions for use in BRKGA
- **Key difference from prior work:** metaoptimizer only needs to be trained once, and is afterwards generalizable to any computation graph
- May be overly complex for the small improvements yielded

Questions?