Mix-and-Match: A Model-driven Runtime Optimisation Strategy for BFS on GPUs

Merijn Verstraaten[1,2], Ana Lucia Varbanescu[1] & Cees de Laat[1]

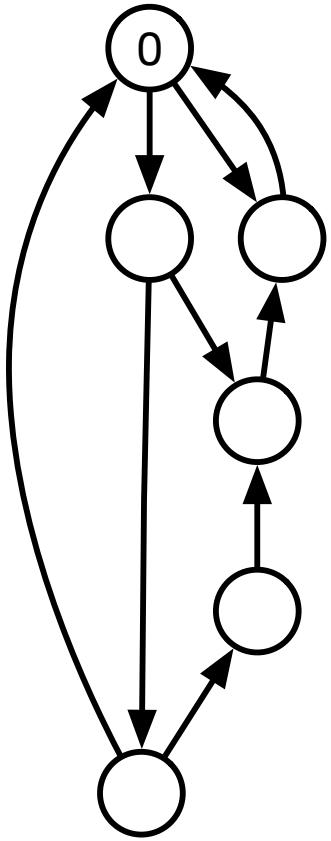
[1] University of Amsterdam

[2] Netherlands eScience Center

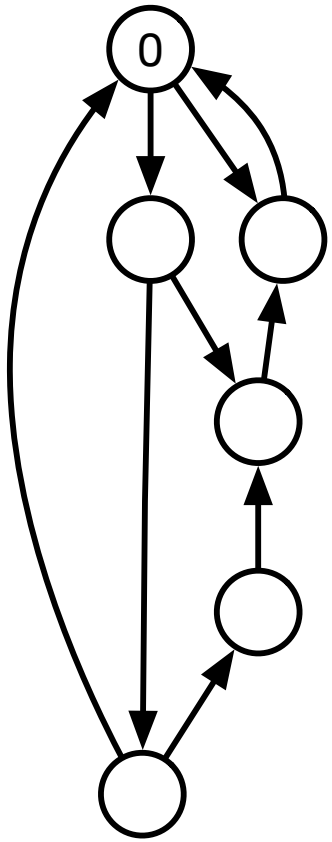
December 5, 2018

Breadth-First Search: Implementations

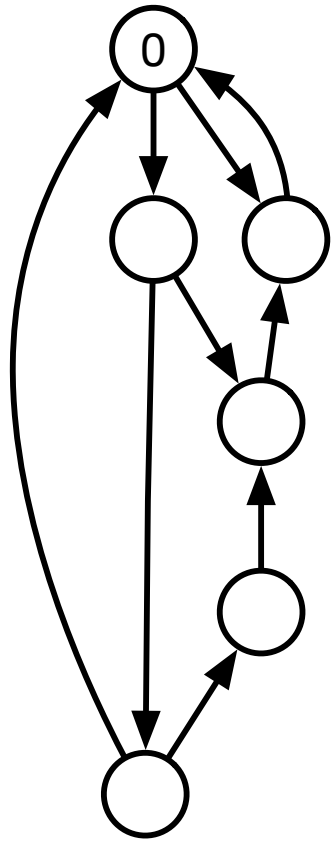
Edge-centric



Vertex Push



Vertex Pull



Useful Frontier Thread

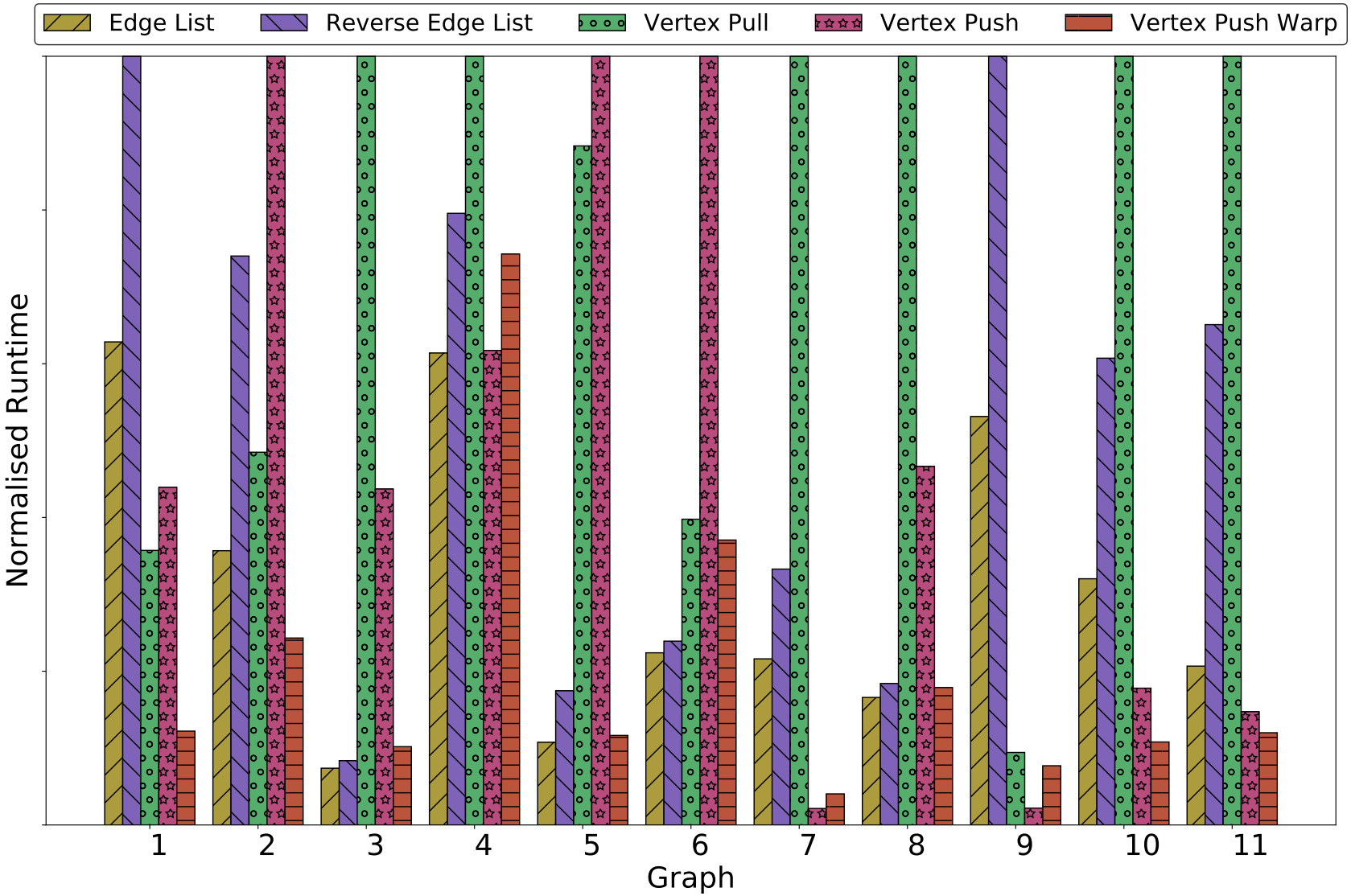
Useless Frontier Thread

Frontier Node

Updated Node

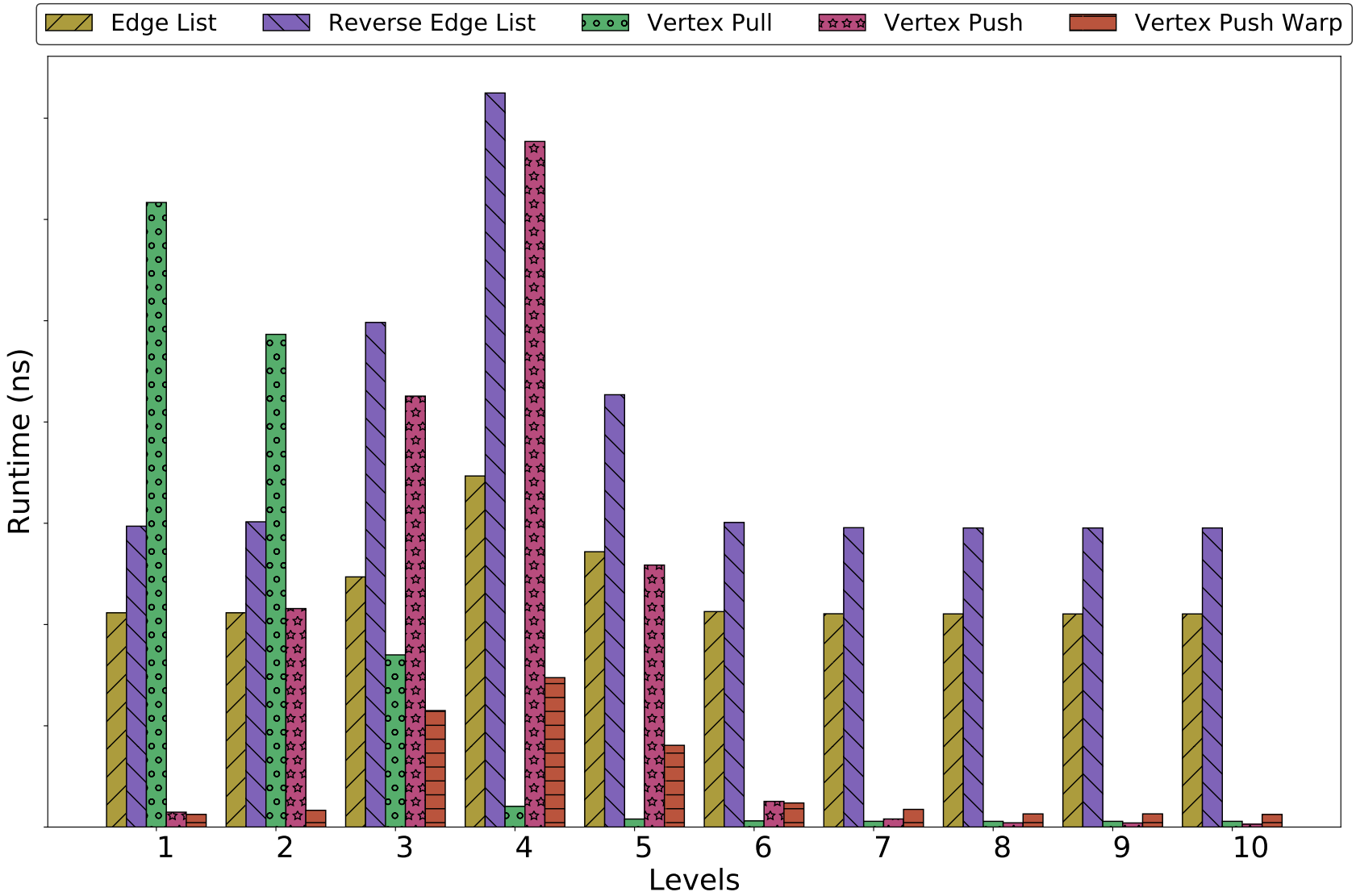
Accessed Node

Relative Performance of Implementations



There is no “best”!

Relative Performance Within a Single Traversal



Sticking to one implementation costs us!

Let's Choose an Algorithm!

Choosing the best algorithm:

- Depends on algorithm + platform + graph
- Predict "best" implementation each level

Challenge?

How to model the algorithm + graph + platform

Is it worth it? It depends on…

- …gain
- …prediction cost
- …data representation for implementations

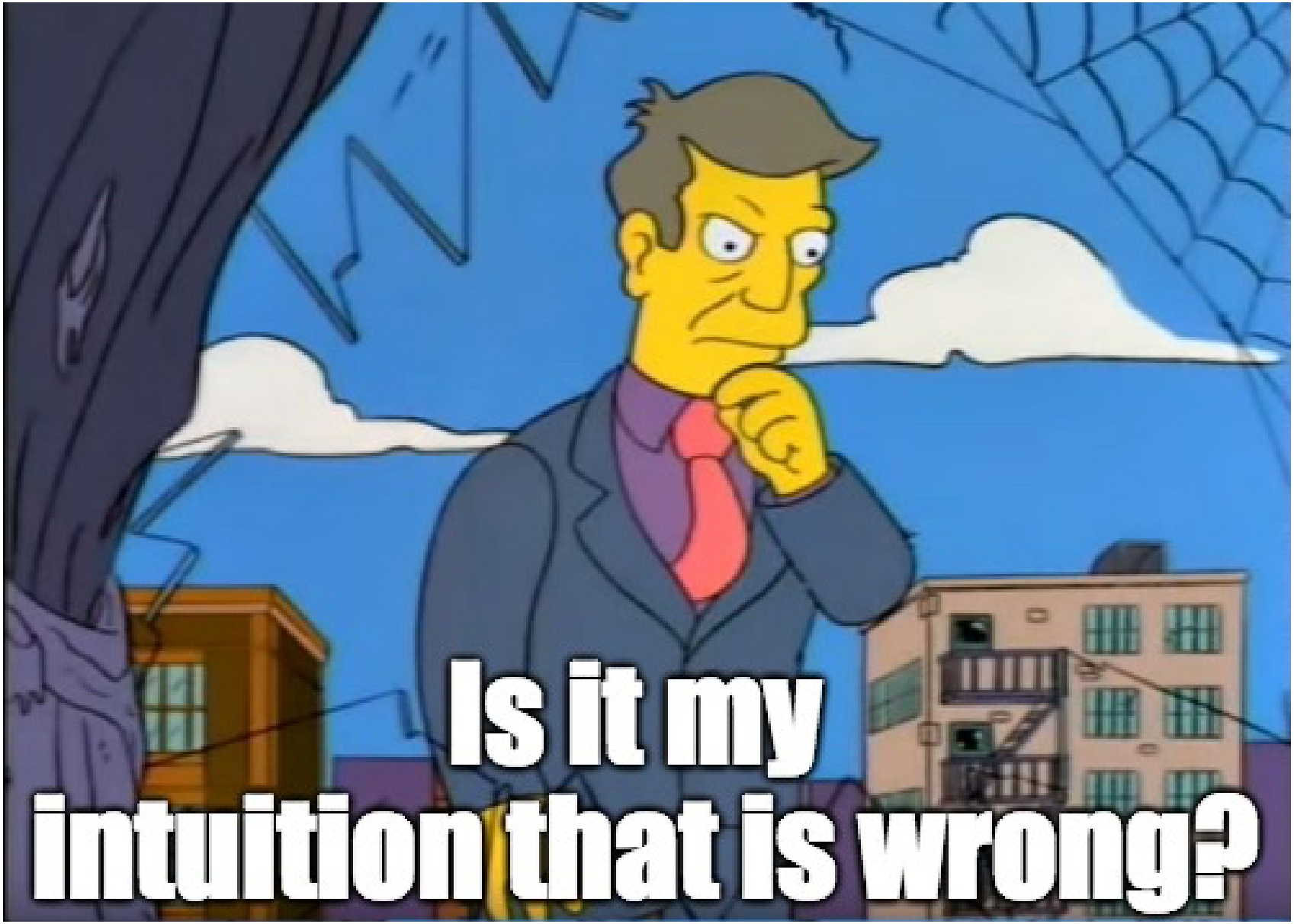
Modelling the Problem

Analytical model:

1. Build a parametrised work model of the algorithm
2. Use graph properties as parameters
3. Calibrate using hardware microbenchmarking

Result: Prediction accuracy below 50%…

Intuition vs Results



Intuition vs Results

Problem: Sequential workload $\rightarrow$ Parallel GPU execution

But: Best implementation stable over several GPU generations

What now?

Intuition vs Results

Machine Learning: Decision Trees

Training Parameters:

- Degree distribution
- Frontier size
- Percentage discovered
- Vertex count
- Edge count

Pros:

- Black-box approach
- Fast! (Training & prediction)
- **Variable importance!**

Cons:

- Can overfit on non-uniform parameters
- Bad with large numbers of parameters

Two Questions

Do the models actually work?

Do the models match our intuitions?

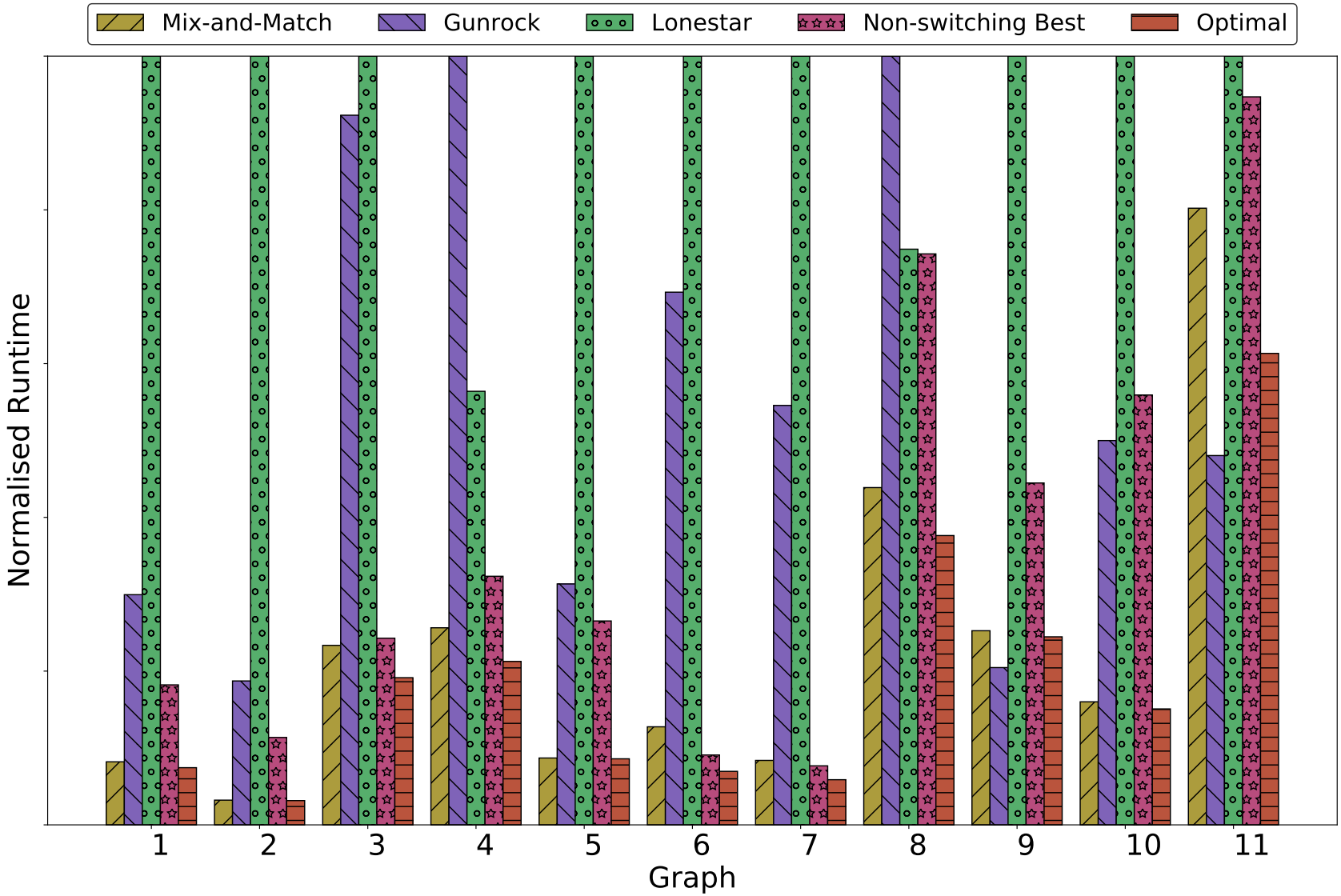
Trained Models

Feasibility:

Average Prediction Time:	144 ns ($\sigma = 165$ ns)
Mininum BFS Step:	20 ms
(Re)loading graph representation:	Stupidly slow

Classic time-space trade-off.

Comparison with State-of-the-Art: Best & Worst



Even better if we include Gunrock in model?

Overall Results

Algorithm	1–2×	>5×	>20×	Average	Worst
Mix-and-Match	92%	2.5%	0.4%	2.04×	498×
Non-switching Best	65%	8%	0%	2.44×	37×
Edge List	49%	22%	2.2%	4.16×	61×
Rev. Edge List	39%	33%	8.8%	7.04×	108×
Vertex Pull	16%	58%	30%	48.41×	2,495×
Vertex Push	23%	53%	28%	55.61×	1,980×
Vertex Push Warp	18%	25%	4.9%	5.42×	88×

Averaged over 248 KONECT graphs.

Model & Intuition

Parameter importance matches intuition

Investigating “poor” predictions reveals new insights

Not investigated (yet):

- Handle implementations with similar results
- Minimising training data
- Model portability across datasets, hardware & algorithm
- Relating BDTs to analytical model

Summary

The Good:

- Prediction works!
- Predictions are fast enough at runtime
- Our Mix-and-Match outperforms state-of-the-art (on average)
- Models provide new insights

The Bad:

- Training set too big
- Training set non-uniformity

Takeaway

Large potential performance gains for graph algorithms

Significant performance improvement for many graphs

Method applicable to any BSP graph algorithm

Science would be less dull with less text and more memes…

Questions?