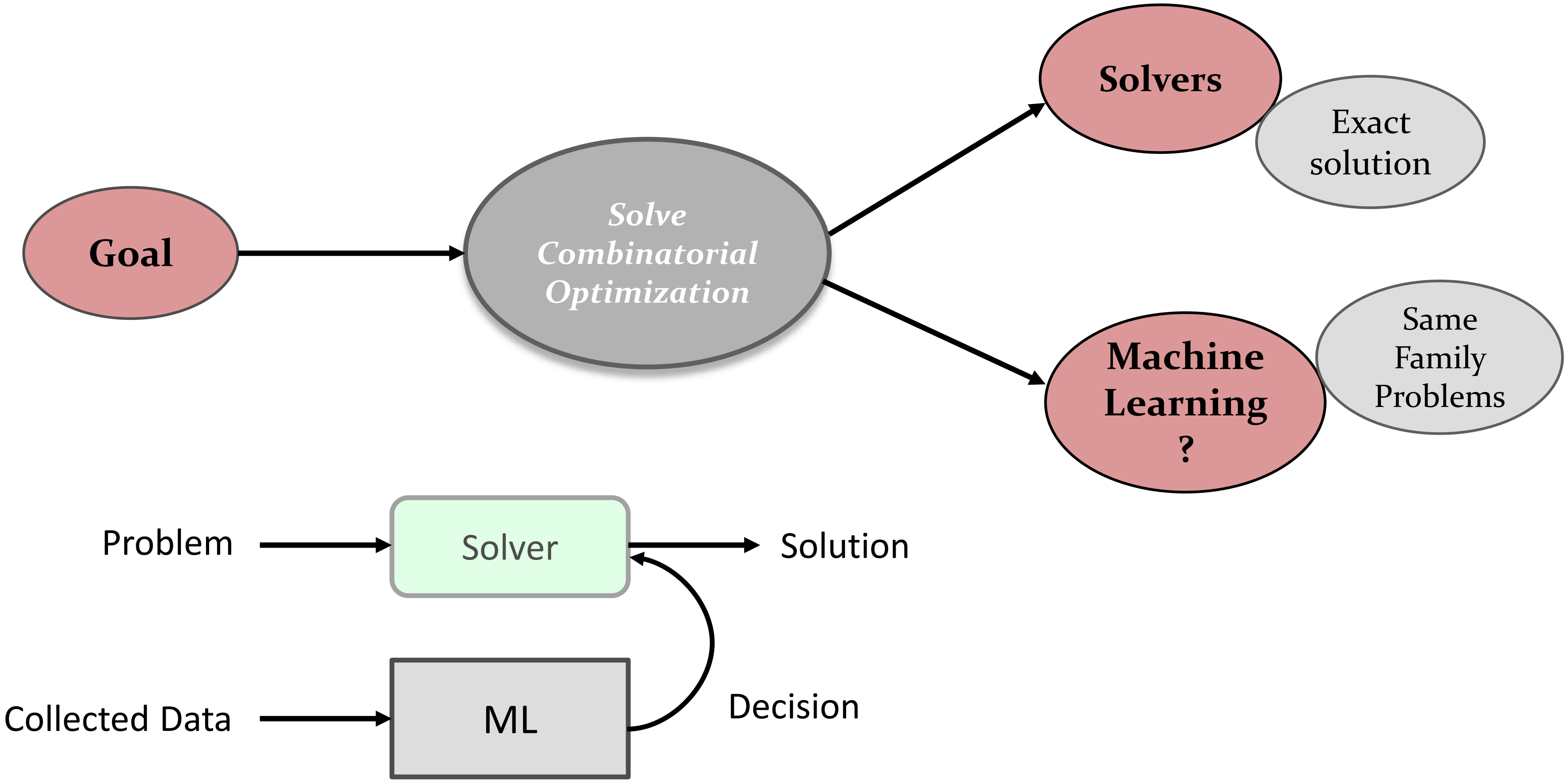
Combining Algorithms and NNs

SEMINAR IN DEEP IN DEEP NEURAL NETWORKS

Kadoglou Maria Eleni, 18.04.21

Exact Combinatorial Optimization with Graph Convolutional Neural Networks



Branch & Bound Algorithm

Mixed-Integer Linear Program (MILP)

$$
\begin{aligned}
\arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\
& \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \\
& \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}
\end{aligned}
$$

NP-Hard Problem

Branch & Bound Algorithm

Linear Program Relaxation

$$
\begin{aligned}
\arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\
& \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \\
& \mathbf{x} \in \mathbb{R}^n
\end{aligned}
$$

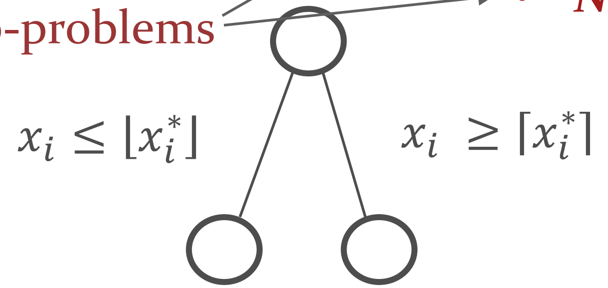
Convex Problem → lower bound to the original MILP

Branch & Bound Algorithm

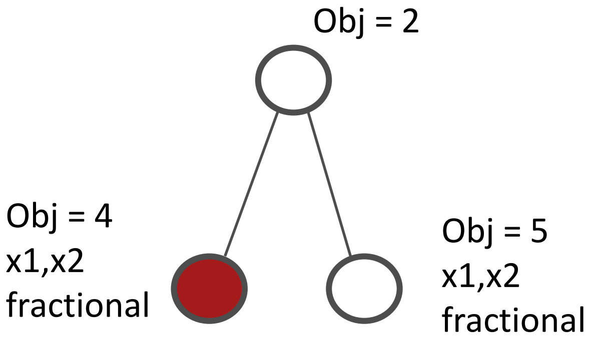
LP

- $x^* \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$ → solution to the original problem (lucky!)
- $x^* \notin \mathbb{Z}^p \times \mathbb{R}^{n-p}$ → decompose into two sub-problems

- ***Branching selection***
- ~~***Node Selection***~~

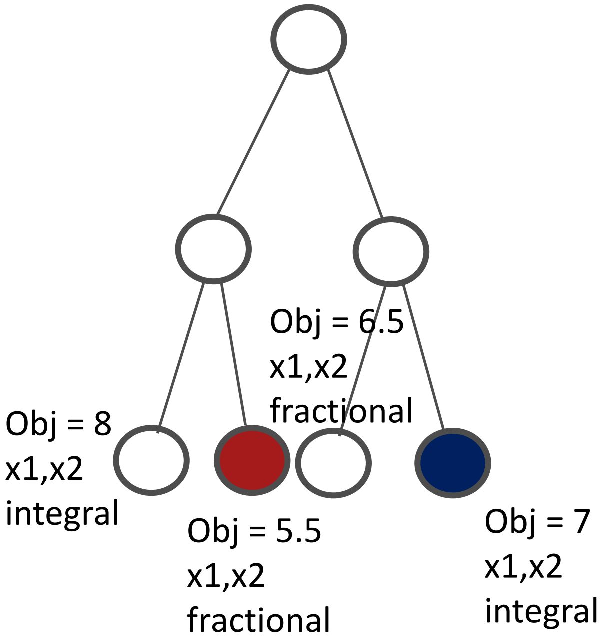


1. Pick a fractional variable to branch on



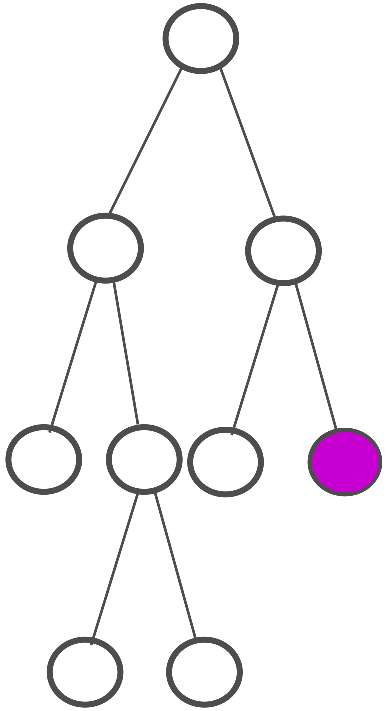
- **Lower Bound**: minimum of leaf nodes
- **Upper Bound**: minimum of leaf nodes with **integer solution**

2. Continue branching to the fractional variables



3. The process stops:

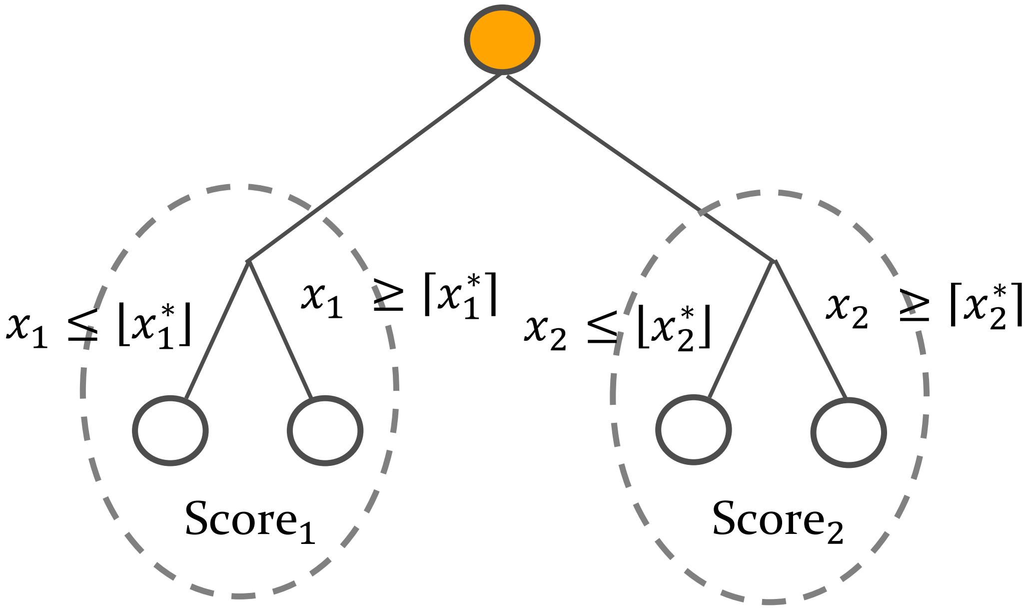
- LB = UB
- LB – UB = ε
- The feasible regions do not decompose anymore



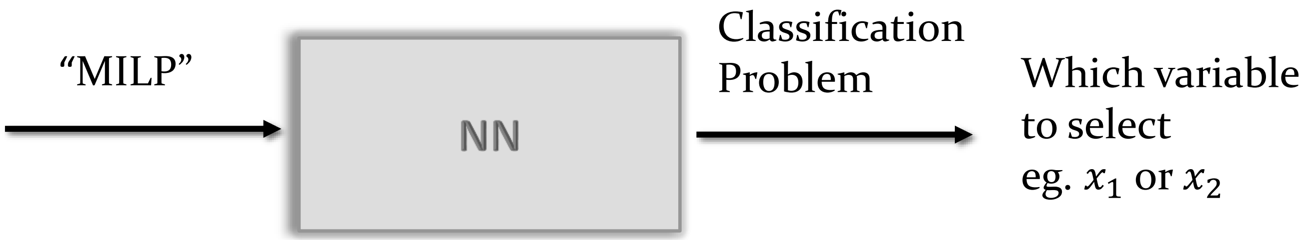
Branch & Bound Algorithm

*Fundamental Questions: Which **variable** to choose for branching?*

- SOTA: Strong Branching



- Proposed Method: **The Neural Network will say**



1. Collect expert state-action pairs $D = \{(s_i, a_i^*)\}_{i=1}^N$
2. Learn the policy by minimizing:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(\mathbf{s},\mathbf{a}^*)\in\mathcal{D}} \log \pi_\theta(\mathbf{a}^* \mid \mathbf{s})$$

Proposed Method

How to represent the states? → Original problem, bounds

How to represent the states? → Relaxation problem, solution to LPs, added constraints

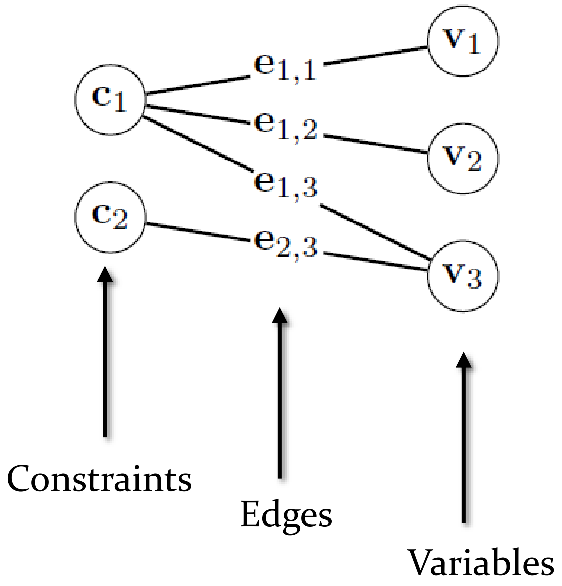
Growing structure → **Graph**

1. ***State Encoding*** → *bipartite graphs with attributes*

Mixed-Integer Linear Program (MILP)

$$
\begin{aligned}
\arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\
& \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \\
& \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}
\end{aligned}
$$

$$s_t = (G, C, E, V)$$



Proposed Method

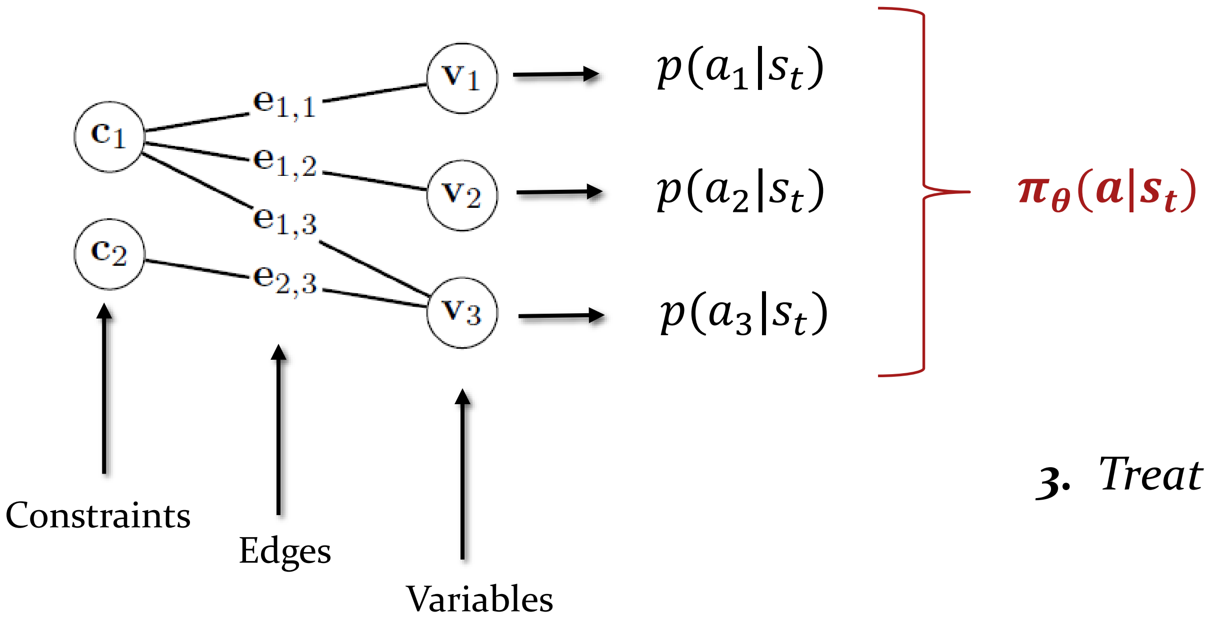
How can we model the Graph to the Neural Netwok?

2. **Policy $\pi_\theta(a|s_t)$** → *Graph Convolutional Neural Network (GCNN)*

$$\mathbf{c}_i \leftarrow \mathbf{f}_{\mathcal{C}}\left(\mathbf{c}_i, \sum_{j}^{(i,j)\in\mathcal{E}} \mathbf{g}_{\mathcal{C}}\left(\mathbf{c}_i, \mathbf{v}_j, \mathbf{e}_{i,j}\right)\right) \qquad \mathbf{v}_j \leftarrow \mathbf{f}_{\mathcal{V}}\left(\mathbf{v}_j, \sum_{i}^{(i,j)\in\mathcal{E}} \mathbf{g}_{\mathcal{V}}\left(\mathbf{c}_i, \mathbf{v}_j, \mathbf{e}_{i,j}\right)\right)$$

Why GCNN?

- They have permutation invariance
- Combine the node with each neighbors
- From variable to constraints
- From constraints to variables



3. *Treat the problem as a* ***classification*** *one*

Evaluation Results

- ***Benchmarks :*** 4 Np Hard Problems
- ***Solver :*** SCIP 6.0.1 open source solver
- ***Baselines :*** Hybrid Branching [RBP], Full Strong Branching Expert [FSB], SVRRANK, LMART, and the regression approach of Alvarez [TREES]

Comparing accuracy of ML models

	Set Covering			Combinatorial Auction			Capacitated Facility Location			Maximum Independent Set		
model	acc@1	acc@5	acc@10	acc@1	acc@5	acc@10	acc@1	acc@5	acc@10	acc@1	acc@5	acc@10
TREES	51.8±0.3	80.5±0.1	91.4±0.2	52.9±0.3	84.3±0.1	94.1±0.1	63.0±0.4	97.3±0.1	99.9±0.0	30.9±0.4	47.4±0.3	54.6±0.3
SVMRANK	57.6±0.2	84.7±0.1	94.0±0.1	57.2±0.2	86.9±0.2	95.4±0.1	67.8±0.1	98.1±0.1	99.9±0.0	48.0±0.6	69.3±0.2	78.1±0.2
LMART	57.4±0.2	84.5±0.1	93.8±0.1	57.3±0.3	86.9±0.2	95.3±0.1	68.0±0.2	98.0±0.0	99.9±0.0	48.9±0.3	68.9±0.4	77.0±0.5
GCNN	**65.5**±0.1	**92.4**±0.1	**98.2**±0.0	**61.6**±0.1	**91.0**±0.1	**97.8**±0.1	**71.2**±0.2	**98.6**±0.1	**99.9**±0.0	**56.5**±0.2	**80.8**±0.3	**89.0**±0.1

Evaluation Results

Performance regarding Solution:

Train on small (Easy) instances and evaluate ***generalization*** on medium (Medium) and large (Hard)

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	17.30 ± 6.1%	0 / 100	17 ±13.7%	411.34 ± 4.3%	0 / 90	171 ± 6.4%	3600.00 ± 0.0%	0 / 0	n/a ± n/a %
RPB	8.98 ± 4.8%	0 / 100	**54** ±20.8%	60.07 ± 3.7%	0 / 100	1741 ± 7.9%	1677.02 ± 3.0%	4 / 65	47 299 ± 4.9%
TREES	9.28 ± 4.9%	0 / 100	187 ± 9.4%	92.47 ± 5.9%	0 / 100	2187 ± 7.9%	2869.21 ± 3.2%	0 / 35	59 013 ± 9.3%
SVMRANK	8.10 ± 3.8%	1 / 100	165 ± 8.2%	73.58 ± 3.1%	0 / 100	1915 ± 3.8%	2389.92 ± 2.3%	0 / 47	42 120 ± 5.4%
LMART	7.19 ± 4.2%	14 / 100	167 ± 9.0%	59.98 ± 3.9%	0 / 100	1925 ± 4.9%	2165.96 ± 2.0%	0 / 54	45 319 ± 3.4%
GCNN	**6.59** ± 3.1%	**85** / 100	134 ± 7.6%	**42.48** ± 2.7%	**100** / 100	**1450** ± 3.3%	**1489.91** ± 3.3%	**66** / 70	**29 981** ± 4.9%

Set Covering

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	4.11 ±12.1%	0 / 100	6 ±30.3%	86.90 ± 12.9%	0 / 100	72 ±19.4%	1813.33 ± 5.1%	0 / 68	400 ± 7.5%
RPB	2.74 ± 7.8%	0 / 100	**10** ±32.1%	17.41 ± 6.6%	0 / 100	689 ±21.2%	136.17 ± 7.9%	13 / 100	5511 ±11.7%
TREES	2.47 ± 7.3%	0 / 100	86 ±15.9%	23.70 ± 11.2%	0 / 100	976 ±14.4%	451.39 ±14.6%	0 / 95	10 290 ±16.2%
SVMRANK	2.31 ± 6.8%	0 / 100	77 ±15.0%	23.10 ± 9.8%	0 / 100	867 ±13.4%	364.48 ± 7.7%	0 / 98	6329 ± 7.7%
LMART	**1.79** ± 6.0%	**75** / 100	77 ±14.9%	14.42 ± 9.5%	1 / 100	873 ±14.3%	222.54 ± 8.6%	0 / 100	7006 ± 6.9%
GCNN	1.85 ± 5.0%	25 / 100	70 ±12.0%	**10.29** ± 7.1%	**99** / 100	**657** ±12.2%	**114.16** ±10.3%	**87** / 100	**5169** ±14.9%

Combinatorial Auction

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	30.36 ±19.6%	4 / 100	14 ±34.5%	214.25 ± 15.2%	1 / 100	76 ±15.8%	742.91 ± 9.1%	15 / 90	55 ± 7.2%
RPB	26.55 ±16.2%	9 / 100	**22** ±31.9%	156.12 ± 11.5%	8 / 100	**142** ±20.6%	631.50 ± 8.1%	14 / 96	**110** ±15.5%
TREES	28.96 ±14.7%	3 / 100	135 ±20.0%	159.86 ± 15.3%	3 / 100	401 ±11.6%	671.01 ±11.1%	1 / 95	381 ±11.1%
SVMRANK	23.58 ±14.1%	11 / 100	117 ±20.5%	130.86 ± 13.6%	13 / 100	348 ±11.4%	586.13 ±10.0%	21 / 95	321 ± 8.8%
LMART	23.34 ±13.6%	16 / 100	117 ±20.7%	128.48 ± 15.4%	23 / 100	349 ±12.9%	582.38 ±10.5%	15 / 95	314 ± 7.0%
GCNN	**22.10** ±15.8%	**57** / 100	107 ±21.4%	**120.94** ± 14.2%	**52** / 100	339 ±11.8%	**563.36** ±10.7%	**30** / 95	338 ±10.9%

Capacitated Facility Location

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	23.58 ±29.9%	9 / 100	7 ±35.9%	1503.55 ± 20.9%	0 / 74	38 ±28.2%	3600.00 ± 0.0%	0 / 0	n/a ± n/a %
RPB	8.77 ±11.8%	7 / 100	**20** ±36.1%	**110.99** ± 24.4%	41 / 100	**729** ±37.3%	2045.61 ±18.3%	22 / 42	**2675** ±24.0%
TREES	10.75 ±22.1%	1 / 100	76 ±44.2%	1183.37 ± 34.2%	1 / 47	4664 ±45.8%	3565.12 ± 1.2%	0 / 3	38 296 ± 4.1%
SVMRANK	8.83 ±14.9%	2 / 100	46 ±32.2%	242.91 ± 29.3%	1 / 96	546 ±26.0%	2902.94 ± 9.6%	1 / 18	6256 ±15.1%
LMART	7.31 ±12.7%	30 / 100	52 ±38.1%	219.22 ± 36.0%	15 / 91	747 ±35.1%	3044.94 ± 7.0%	0 / 12	8893 ± 3.5%
GCNN	**6.43** ±11.6%	**51** / 100	43 ±40.2%	192.91 ±110.2%	**42** / 82	1841 ±88.0%	**2024.37** ±30.6%	**25** / 29	2997 ±26.3%

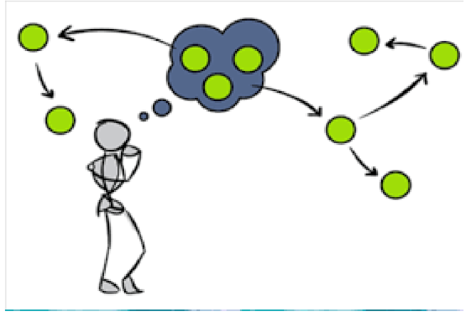
Maximum Independent Set

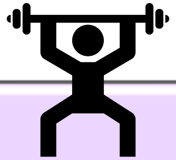
Results

+ better in terms of **solving time**
+ **generalizes** to fairly larger instances

- **performance decreases** as the model is evaluated on **larger** problems
- **outside** of the training distribution ??
- **need data** for training

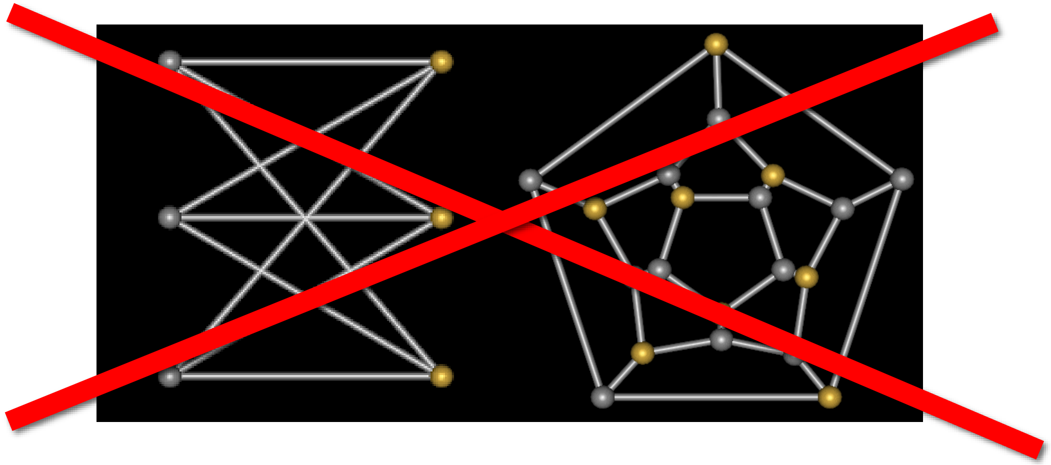
*How Combinatorial Optimization **can help** in Deep Learning?*





Deep Learning

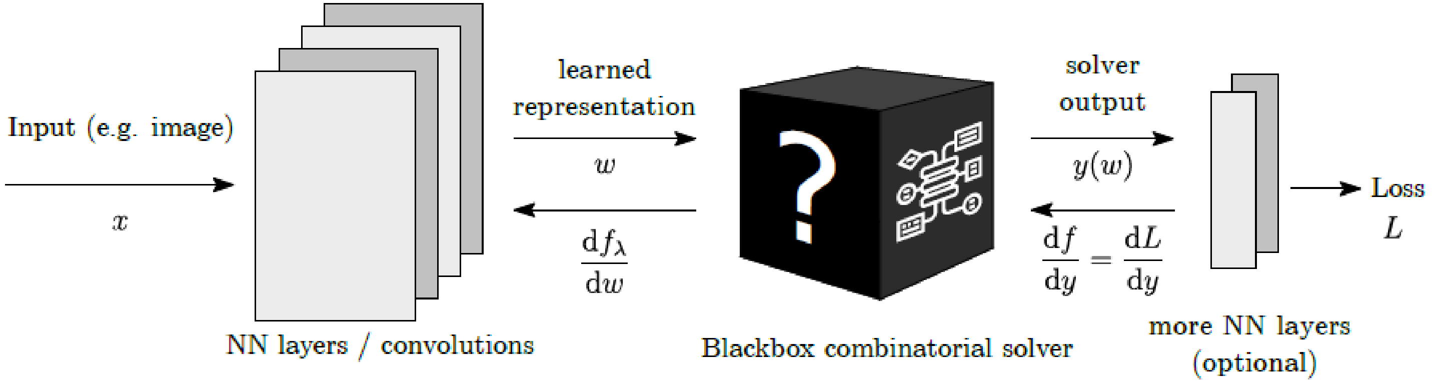
e.g. Predict the quickest routes in Google Maps based on map input as an image



Construct Hybrid Architectures

- Problem: **differentiability** of the combinatorial components
- SOTA approaches : *Solve a relaxation problem* → Sub- optimal:
  - Runtime
  - Performance
  - Optimality Quarantees

Differentiation of Blackbox Combinatorial Solvers



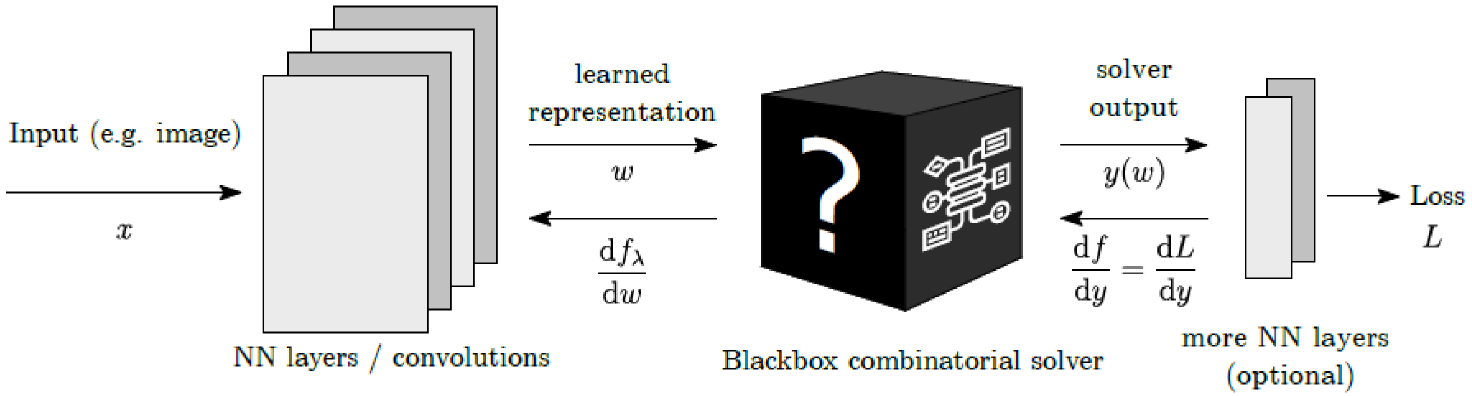
Gradients of Blackbox Solver

Continuous Input w → Solver → Discrete Output y

$\omega \mapsto y(\omega)$ such that $y(w) = \arg\ \min_{y \in Y} \mathbf{c}(w, y)$

- Cost Function: $\mathbf{c}(w, y) = w \cdot \phi(y)$
- ω → edge weights of a graph

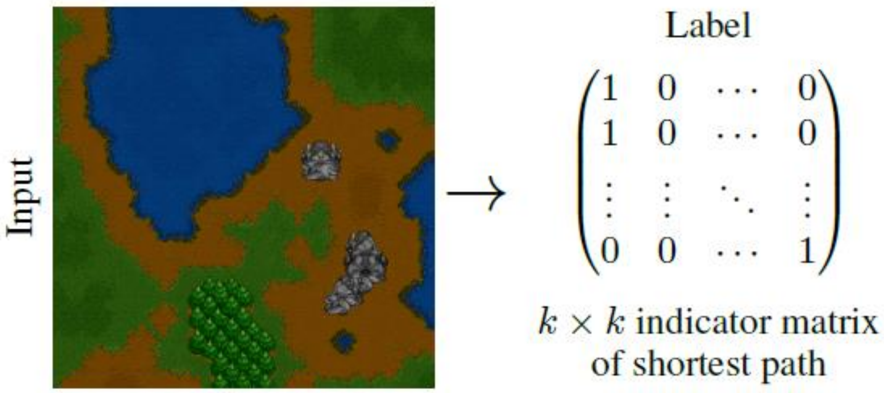
Differentiation of Blackbox Combinatorial Solvers



Shortest Path Problem from raw Images

w → representation

output: predicted shortest path for the respective map

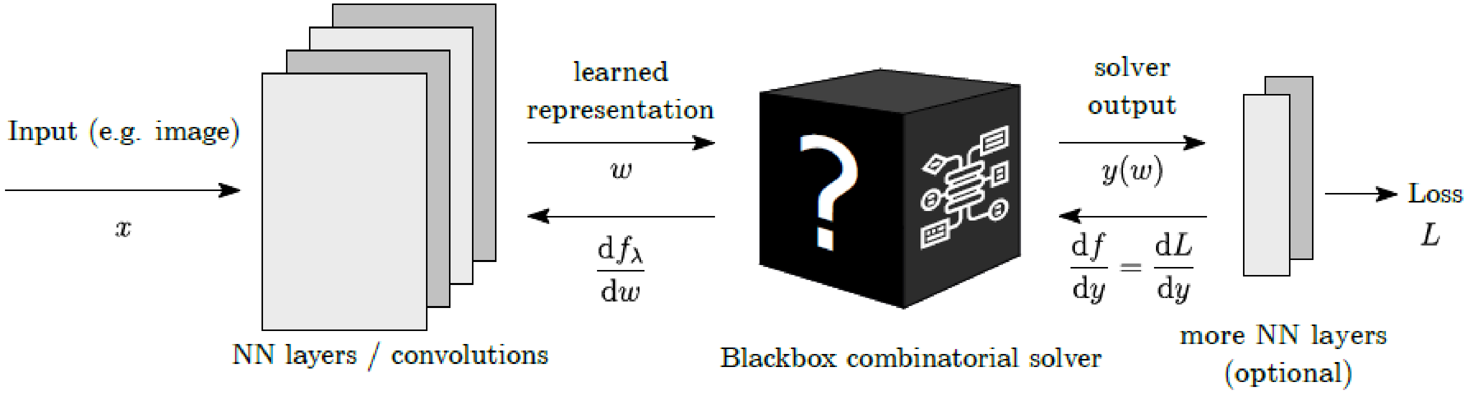


L

Loss: Hamming distance between the true and the predicted SP

Piecewise function

Differentiation of Blackbox Combinatorial Solvers



Forward propagation

x → NN → ω → Solver → y → L(y)

Backpropagation

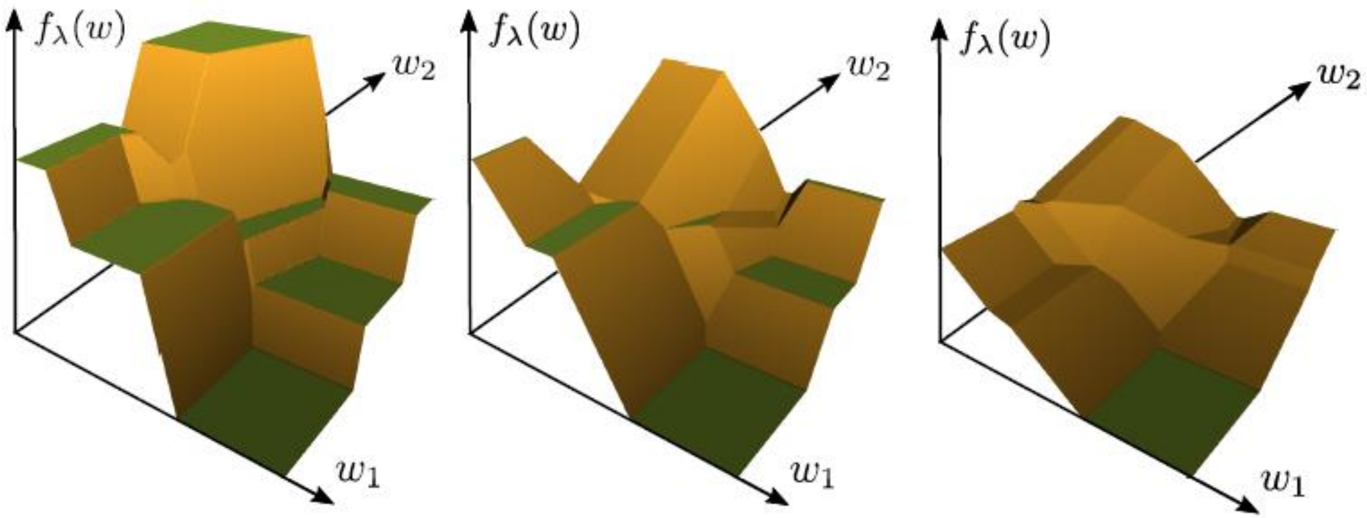
dL / dx ← NN ← dL/dω ← Solver ← dL/dy ← L

Problem!! dL/dω ⟶ *Useless in Optimization*

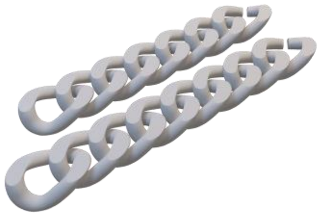
Method of Interpolation

General Approaches → Relaxation → Loose a lot of information

Solution → **Interpolation**



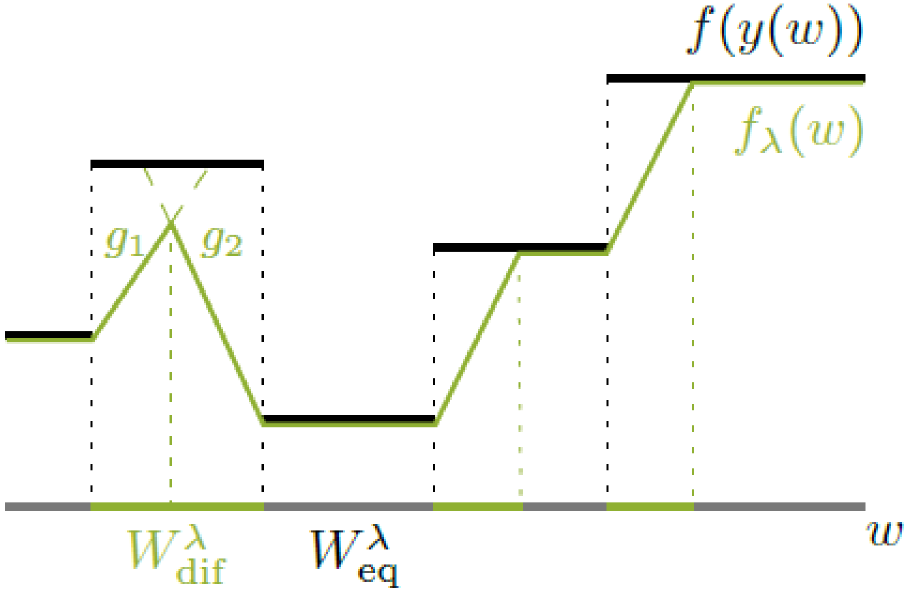
Method of Interpolation



$$\frac{dL}{d\omega} = \frac{dL}{dy}\frac{dy}{d\omega} \longrightarrow$$ ***We want a trick!***

Linearization

$$f(y) = L(\hat{y}) + \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y}) \cdot (y - \hat{y}) \longrightarrow \frac{\mathrm{d}f\big(y(w)\big)}{\mathrm{d}w} = \frac{\mathrm{d}L}{\mathrm{d}w}$$



Interpolation

$$f_\lambda(w) = f\big(y_\lambda(w)\big) - \frac{1}{\lambda}\Big[\mathbf{c}\big(w, y(w)\big) - \mathbf{c}\big(w, y_\lambda(w)\big)\Big]$$

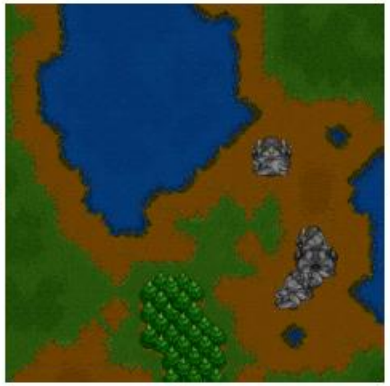
$$y_\lambda(w) = \arg\min_{y \in Y}\{\mathbf{c}(w, y) + \lambda f(y)\}$$

Gradient

$$\nabla f_\lambda(w) = -\frac{1}{\lambda}\Big[\frac{\mathrm{dc}}{\mathrm{d}w}\big(w, y(w)\big) - \frac{\mathrm{dc}}{\mathrm{d}w}\big(w, y_\lambda(w)\big)\Big] = -\frac{1}{\lambda}\big[y(w) - y_\lambda(w)\big]$$

Experiments

Input



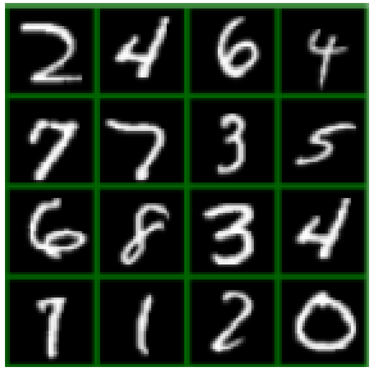
$\rightarrow$

Label

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

$k \times k$ indicator matrix of shortest path

Input



$\rightarrow$

$$\begin{pmatrix} 0 \\ 1 \\ \vdots \\ 1 \\ 0 \end{pmatrix}$$

Input
k flags

$\longrightarrow$

Label

$$\begin{pmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{pmatrix}$$

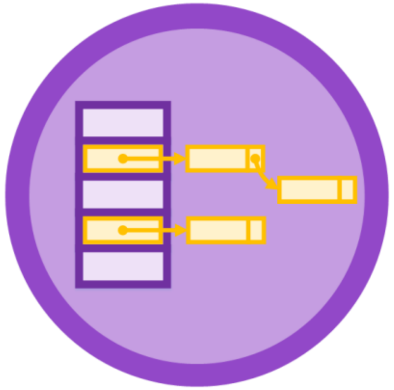
$k \times k$ adjacency matrix with optimal TSP tour

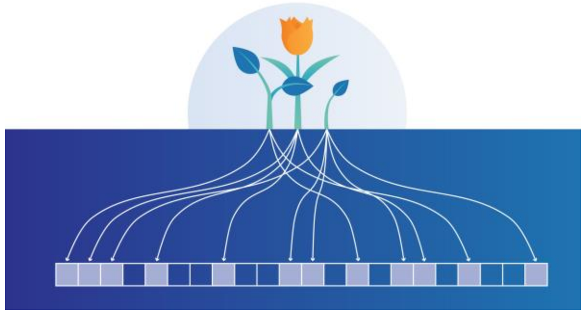
	Embedding Dijkstra		ResNet18	
k	Train %	Test %	Train %	Test %
12	99.7 ± 0.0	96.0 ± 0.3	100.0 ± 0.0	23.0 ± 0.3
18	98.9 ± 0.2	94.4 ± 0.2	99.9 ± 0.0	0.7 ± 0.3
24	97.8 ± 0.2	94.4 ± 0.6	100.0 ± 0.0	0.0 ± 0.0
30	97.4 ± 0.1	94.0 ± 0.3	95.6 ± 0.5	0.0 ± 0.0

The Case for Learned Index Structures

Can indexing data structures be replaced with machine learning models?

Fundamental Algorithms & Data Structure



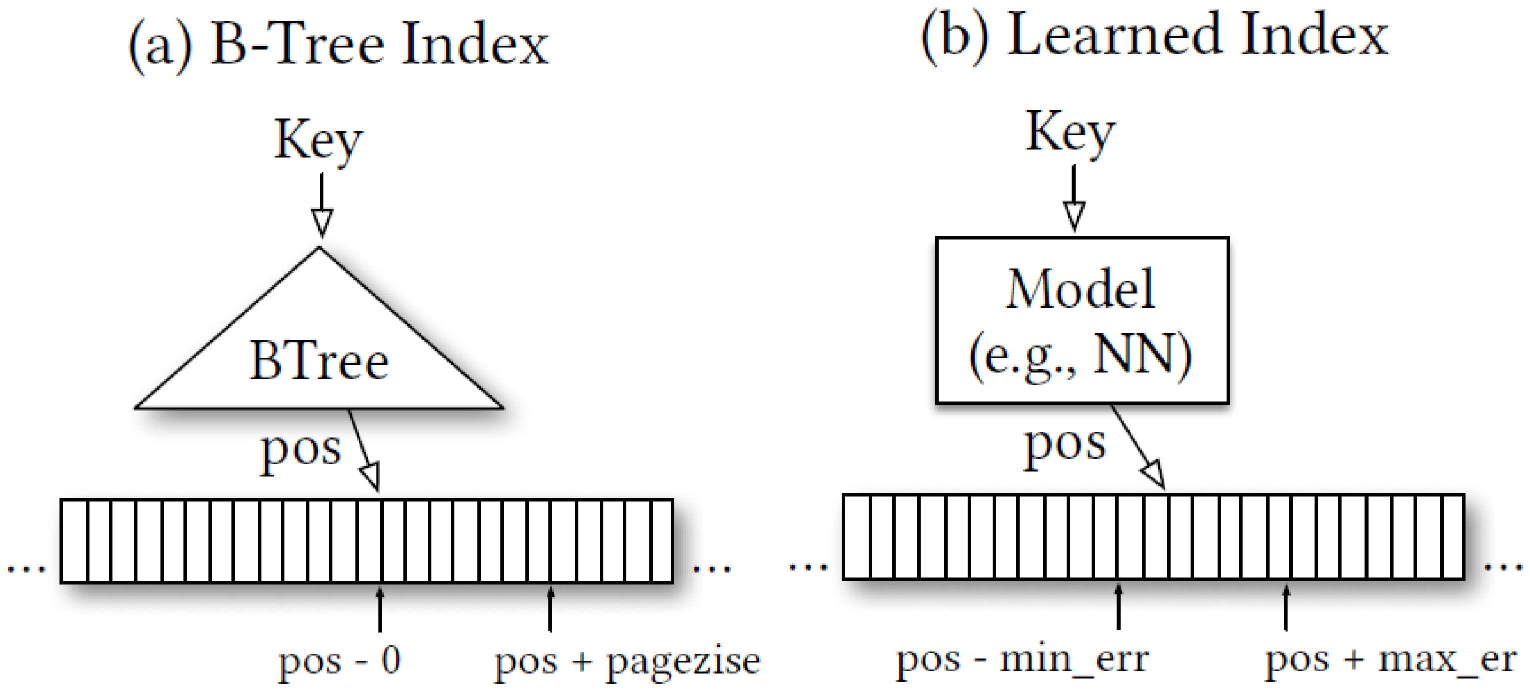


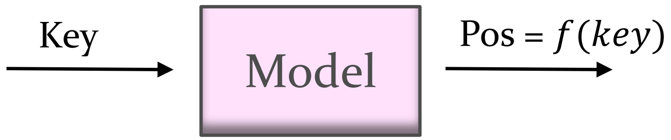
Why ML?

- Powerful GPU → Parallelism
- Speed and Memory usage
- Benefit from data distributions
- Read-only in-memory data

B-Tree Index

Basic Idea: "Predict" the position





1.

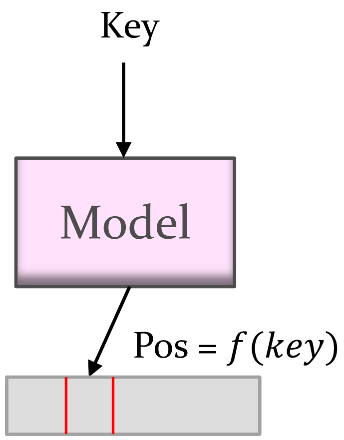
2. Do Binary search inside:
   [Pos –min_error, Pos +max_error]

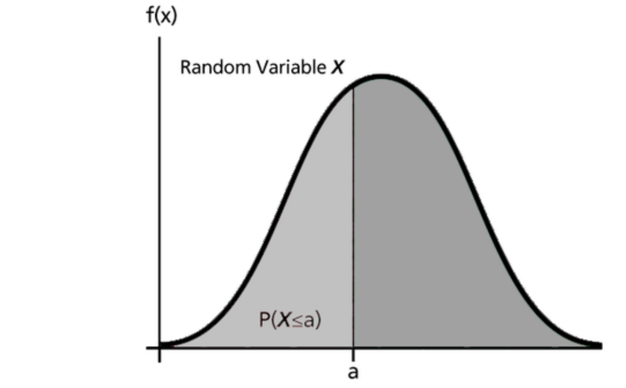
3. How to find this error?

4. Run all the keys through the model and take the maximum (over, under)miss-predictions

B-Tree Index

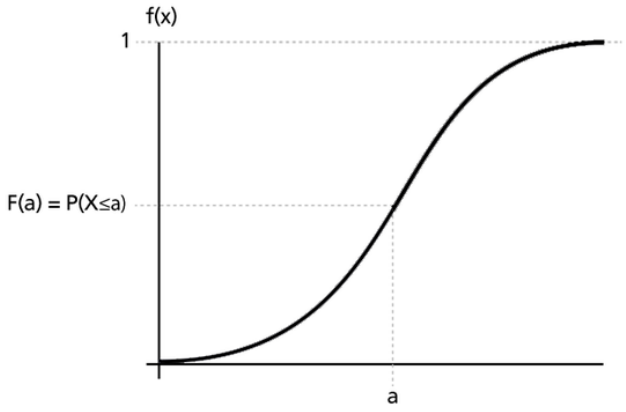
What eventually does the model? ***"Modeling"*** *the* ***CDF*** *of the key distribution*





$$Position = P(X \leq key) * Number_{keys}$$

- So first need to learn the data distribution
- Benefit because CDFs in ML are well studied over decades



First Attempt

TensorFlow

- 200M web-server log records by timestamp sorted
- 2 layer NN with 32 neurons/layer + ReLU

Goal : given the timestamp (index) → predict the position

+ Measure the look-up time

B-Tree Index

Result?

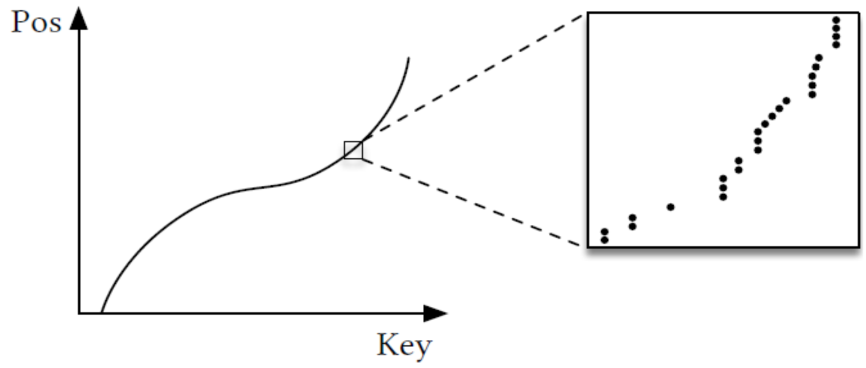
250 ns

TensorFlow

80.000 ns

But why?

1. Tensorflow → designed for large models
2. B-Trees are good in overfitting
3. B-Trees → cache and operation efficient
4. Predict the region not the exact position



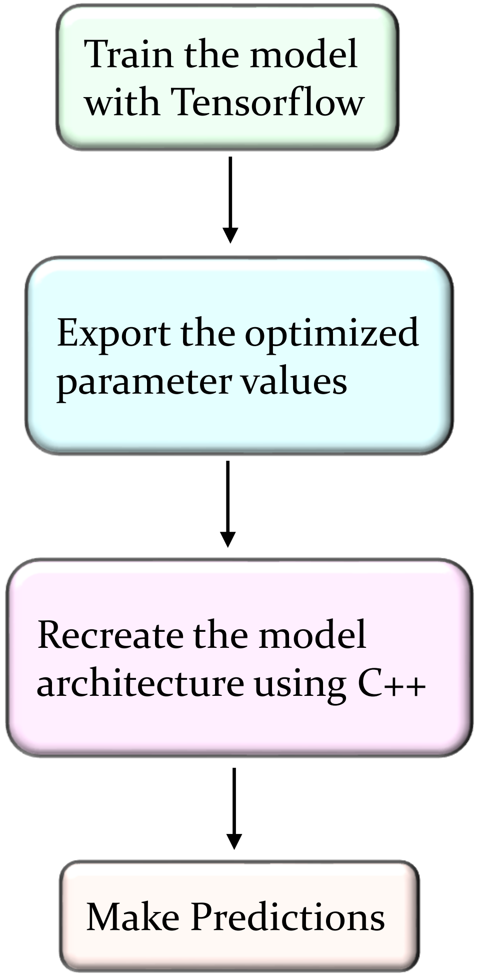
↓

Need to apply a search method (e.g. binary search)

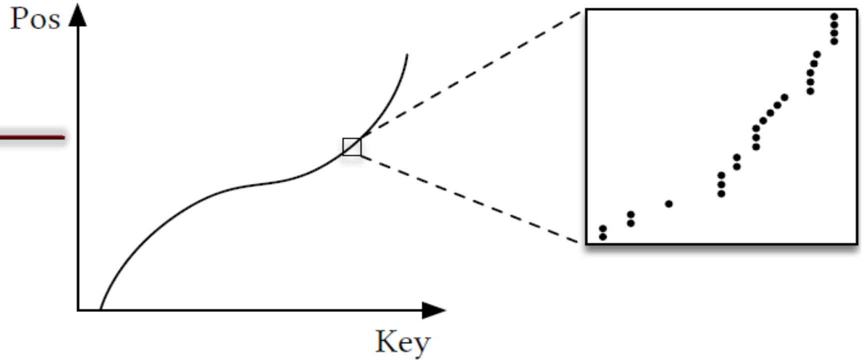
1. Learning Index Framework (LIF)

2. Recursive Model Index

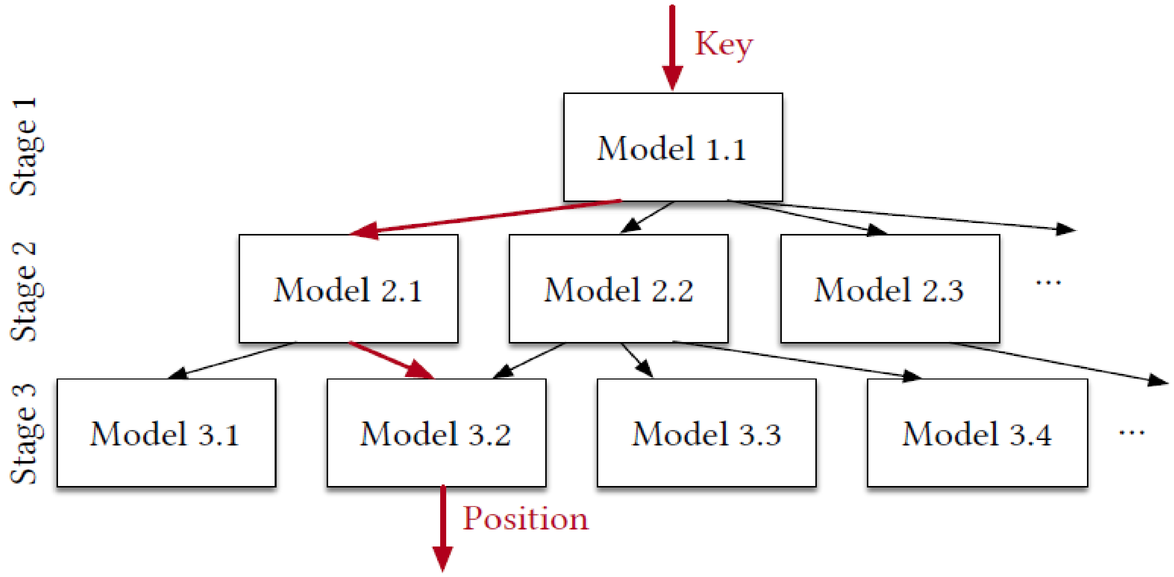
- Index synthesis system



Still Problems!

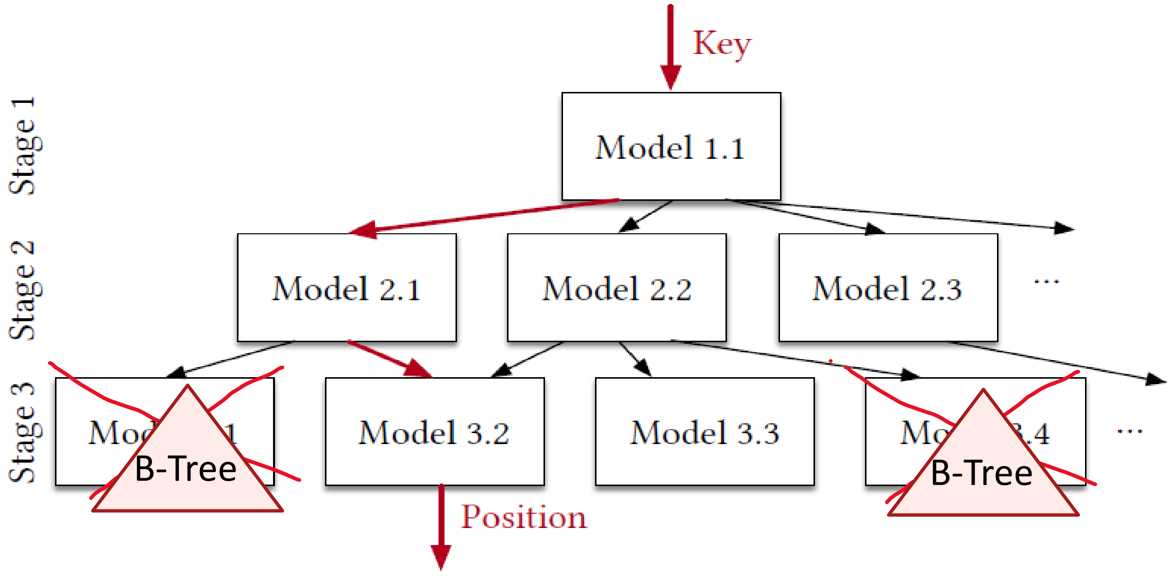


Solution



3. Hybrid Indexes

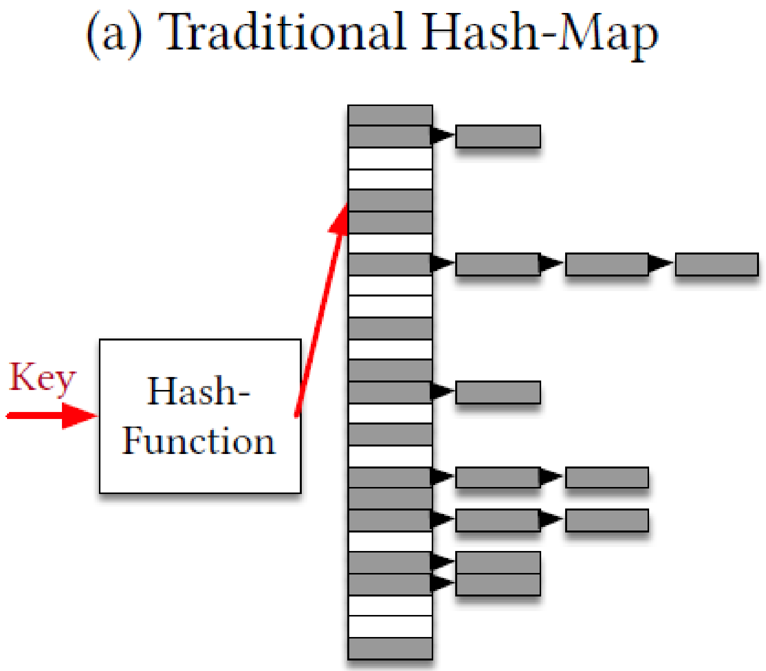
- Build Mixtures of Models
- Worse case Scenario : **B-Tree!**

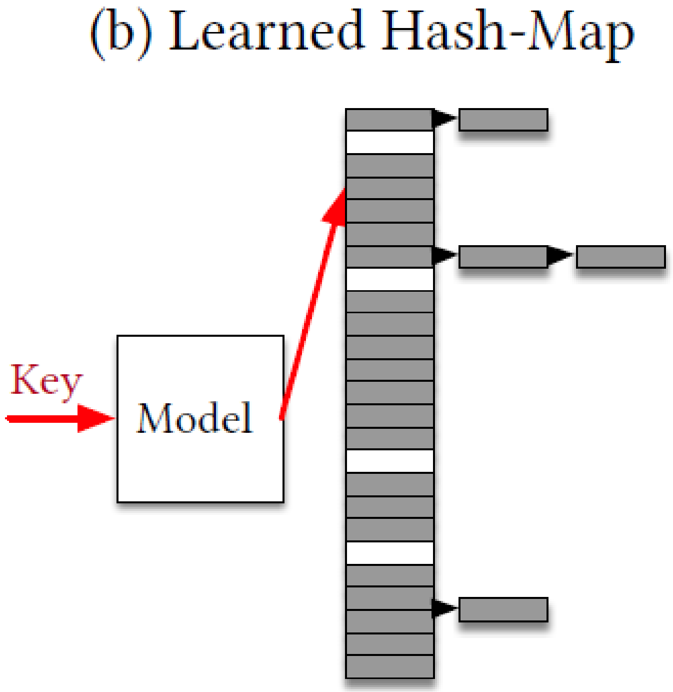


Results

Type	Config	Map Data Size (MB)	Map Data Lookup (ns)	Map Data Model (ns)	Web Data Size (MB)	Web Data Lookup (ns)	Web Data Model (ns)	Log-Normal Data Size (MB)	Log-Normal Data Lookup (ns)	Log-Normal Data Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

Hash Maps





Goal: Reduce Conflicts

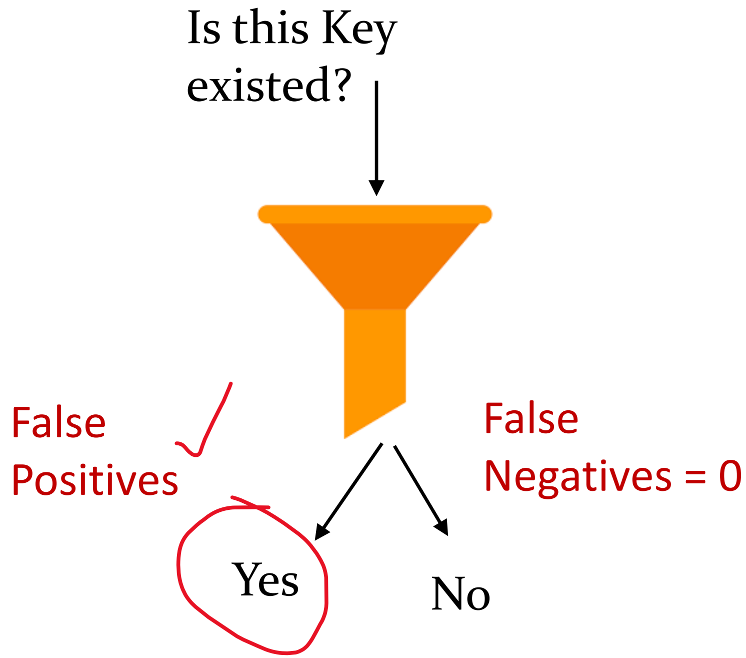
Again Learn Distributions!

$$h(K) = P(X \leq K) * M$$

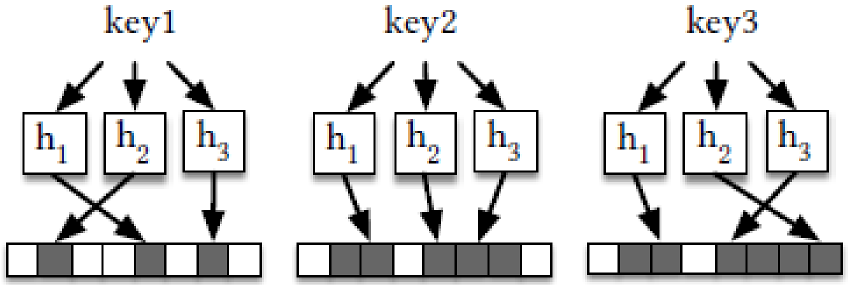
Hash Maps - Results

	% Conflicts Hash Map	% Conflicts Model	Reduction
Map Data	35.3%	07.9%	77.5%
Web Data	35.3%	24.7%	30.0%
Log Normal	35.4%	25.9%	26.7%

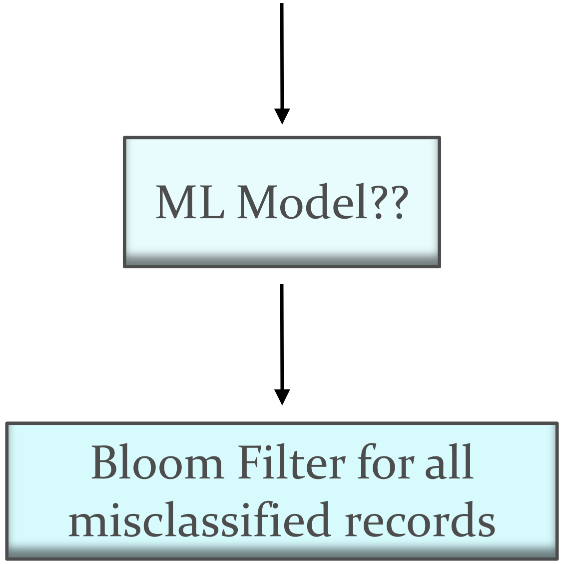
Bloom Filter



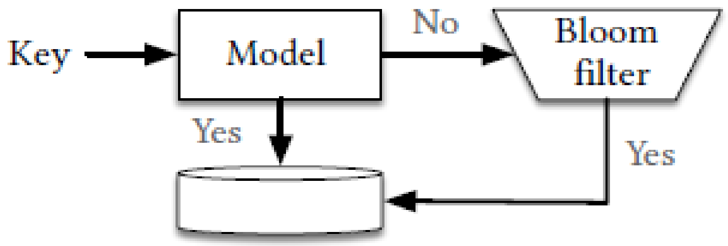
(a) Traditional Bloom-Filter Insertion



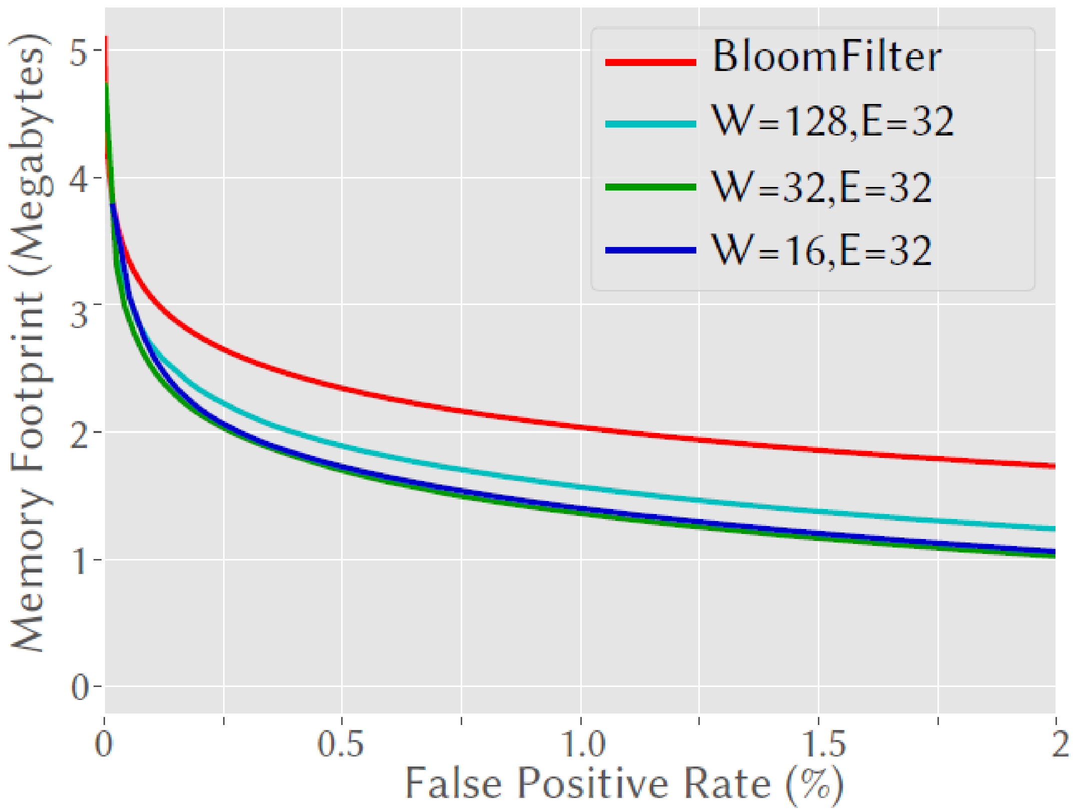
- For the No we want to be sure!



(c) Bloom filters as a classification problem

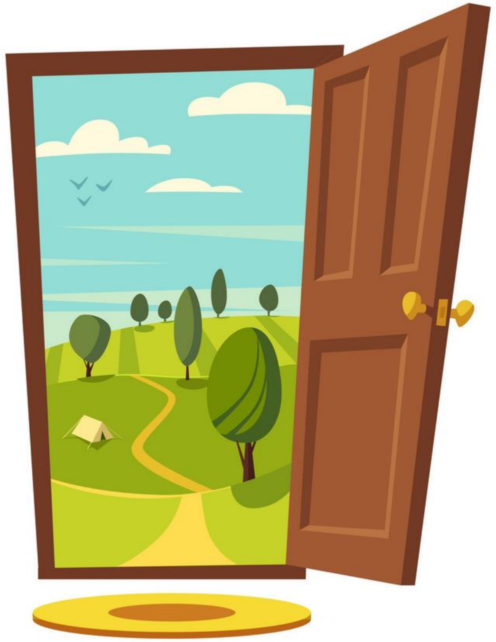


Bloom Filter - Results



36% Memory Reduction!

The Case for Learned Index Structures – Future Work



References

- Exact Combinatorial Optimization with Graph Convolutional Neural Networks, https://arxiv.org/abs/1906.01629
- Black Box Combinatorial Solver, https://arxiv.org/abs/1912.02175
- The Case for Learned Index Structures, https://arxiv.org/pdf/1712.01208.pdf
- https://medium.com/syncedreview/googles-tpu-chip-goes-public-in-challenge-to-nvidia-s-gpu-78ced56776b5
- https://towardsdatascience.com/graph-convolutional-networks-deep-99d7fee5706f