RWTH AACHEN UNIVERSITY

Decision Jungles

Tobias Pohlen

March 8, 2015

RWTH AACHEN UNIVERSITY

Outline

- Literature
- Introduction
- Training
- Implementation Details
- Experiments and Results

RWTH AACHEN UNIVERSITY

Literature

- Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision Jungles: Compact and rich models for classification. *Advances in Neural Information Processing System 26*, pages 234-242. Curran Associates, Inc., 2013
- Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision Jungles: Compact and rich models for classification. Supplemental material. 2013
- Piotr Dollár, Piotr's Image and Video Matlab Toolbox (PMT). `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`.

The Classification Problem

Objective

- Solve the *multiclass classification problem*

Definition *(Multiclass classification problem)*

Given

- A *training set* $X = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^n \times \{1, \ldots, C\}$
  - *training examples* $\mathbf{x}_i \in \mathbb{R}^n$
  - *class labels* $y_i \in \{1, \ldots, C\}$

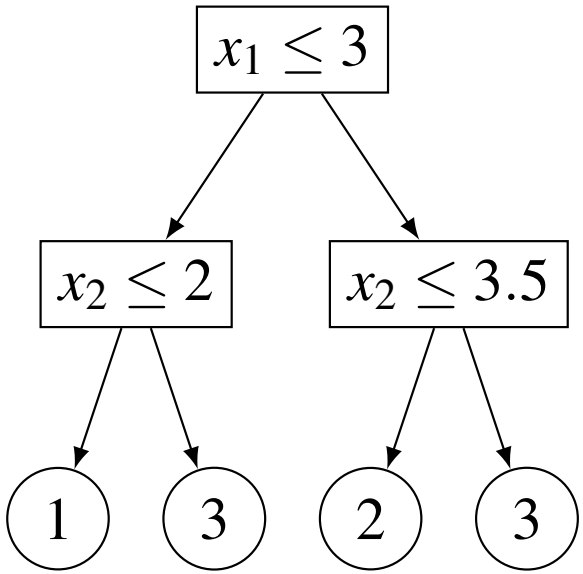
Problem

- Assign the previously unseen data point $\mathbf{x}$ to one of the classes $1, \ldots, C$

Binary Decision Trees

Definition *(Binary decision tree)*

A *binary decision tree* is a binary tree $G = (V, E)$ with the following properties:



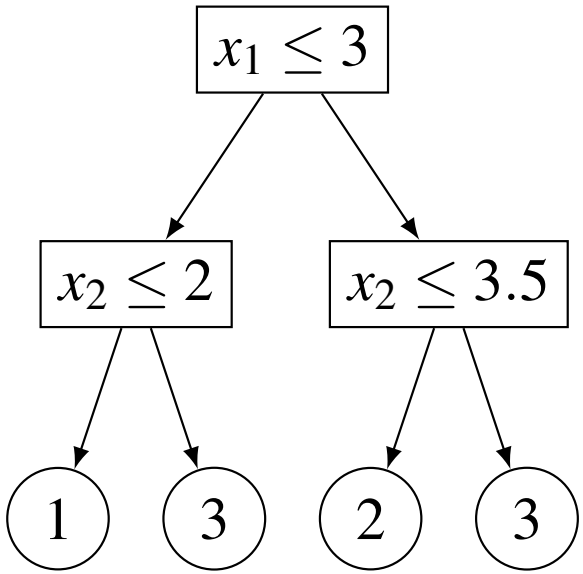
Binary Decision Trees

Definition *(Binary decision tree)*

A *binary decision tree* is a binary tree $G = (V, E)$ with the following properties:

An *internal node* v is augmented with

- *Feature dimension* $d_v \in \{1, \ldots, n\}$
- *Threshold* $\theta_v \in \mathbb{R}$



Binary Decision Trees

Definition *(Binary decision tree)*

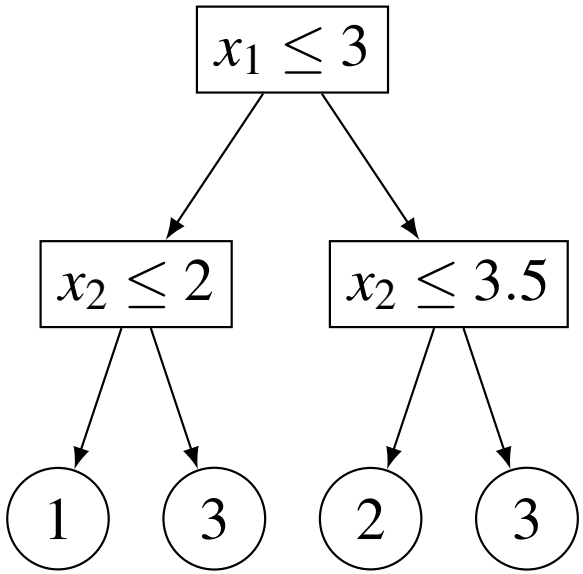
A *binary decision tree* is a binary tree $G = (V, E)$ with the following properties:

An *internal node* v is augmented with

- *Feature dimension* $d_v \in \{1, ..., n\}$
- *Threshold* $\theta_v \in \mathbb{R}$

A *leaf node* v is augmented with

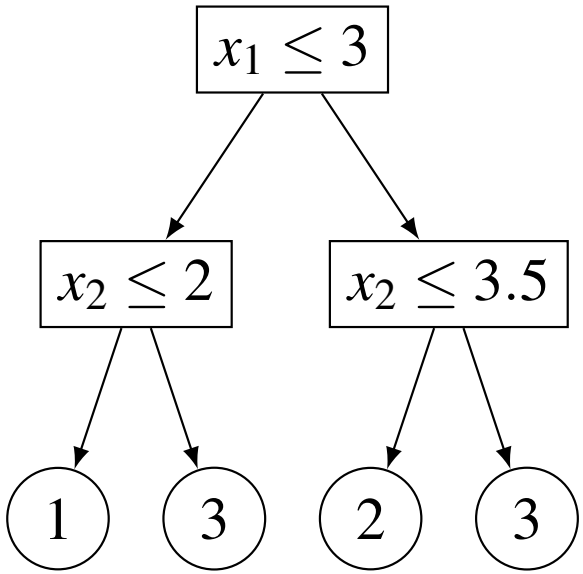
- Class label c_v
- **or** *class histogram* $h_v : \{1, ..., C\} \mapsto \mathbb{R}$



Classifying Data Points

Definition *(Classifier semantics)*

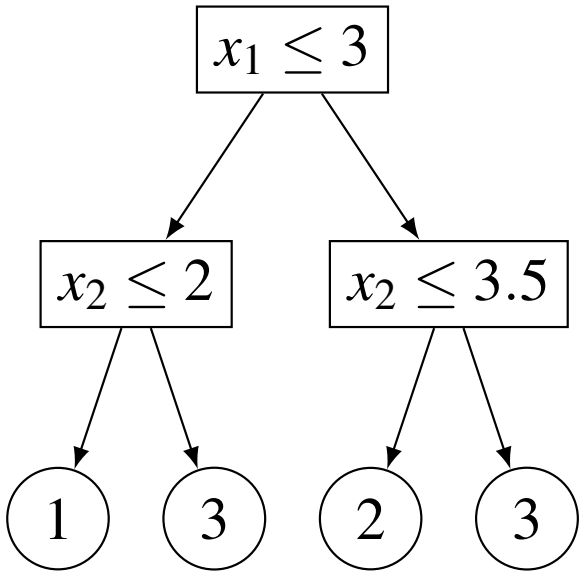
A data point $\mathbf{x} \in \mathbb{R}^n$ is assigned to a class by passing it along the tree according to the splits defined by d_v and θ_v.



Classifying Data Points

Definition *(Classifier semantics)*

A data point $\mathbf{x} \in \mathbb{R}^n$ is assigned to a class by passing it along the tree according to the splits defined by d_v and θ_v.



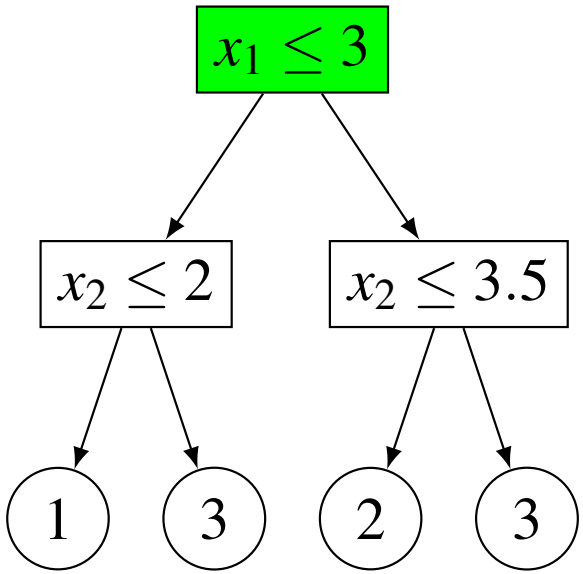
Example

Classify $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$

Classifying Data Points

Definition *(Classifier semantics)*

A data point $\mathbf{x} \in \mathbb{R}^n$ is assigned to a class by passing it along the tree according to the splits defined by d_v and θ_v.



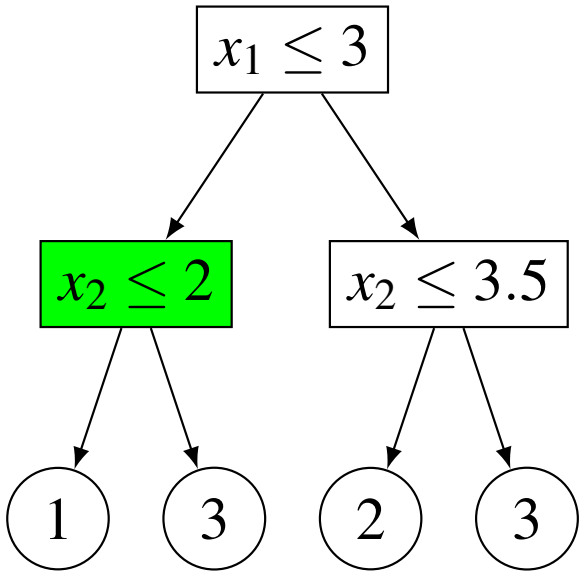
Example

Classify $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$

Classifying Data Points

Definition *(Classifier semantics)*

A data point $\mathbf{x} \in \mathbb{R}^n$ is assigned to a class by passing it along the tree according to the splits defined by d_v and θ_v.



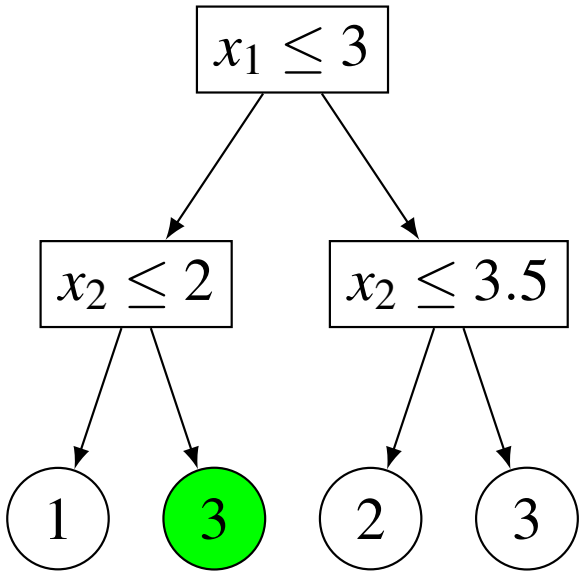
Example

Classify $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$

Classifying Data Points

Definition *(Classifier semantics)*

A data point $\mathbf{x} \in \mathbb{R}^n$ is assigned to a class by passing it along the tree according to the splits defined by d_v and θ_v.



Example

Classify $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$

Random Decision Trees

Let E be some objective function.

Deterministic decision trees

At each node v, determine d_v and θ_v such that

$$E(d_v, \theta_v) = \min_{d \in \{1,\ldots,n\}, \theta \in \mathbb{R}} E(d, \theta)$$

Random Decision Trees

Let E be some objective function.

Deterministic decision trees

At each node v, determine d_v and θ_v such that

$$E(d_v, \theta_v) = \min_{d \in \{1,\dots,n\}, \theta \in \mathbb{R}} E(d, \theta)$$

Random decision trees

At each node v, determine d_v and θ_v such that

$$E(d_v, \theta_v) = \min_{d \in \mathcal{F}, \theta \in \mathbb{R}} E(d, \theta)$$

where

- $\mathcal{F} \subseteq \{1, \dots, C\}$ is a **random** selection of features

Random Forests

Definition

A *random forest* $F = (G_1, \ldots, G_m)$ is an ensemble of random decision trees G_i.

Random Forests

Definition

A *random forest* $F = (G_1, \ldots, G_m)$ is an ensemble of random decision trees G_i.

Classification

A data point $\mathbf{x} \in \mathbb{R}^n$ is assigned to the class that receives the most votes.

RWTH AACHEN UNIVERSITY

Random Forests: Discussion

- Initially proposed by Breiman in 2001 [1]

RWTH AACHEN UNIVERSITY

Random Forests: Discussion

- Initially proposed by Breiman in 2001 [1]
- High classification accuracy by learning uncorrelated trees

RWTH AACHEN UNIVERSITY

Random Forests: Discussion

- Initially proposed by Breiman in 2001 [1]
- High classification accuracy by learning uncorrelated trees
- Fast training due to random feature selection

RWTH AACHEN UNIVERSITY

Random Forests: Discussion

- Initially proposed by Breiman in 2001 [1]
- High classification accuracy by learning uncorrelated trees
- Fast training due to random feature selection
- Fast evaluation

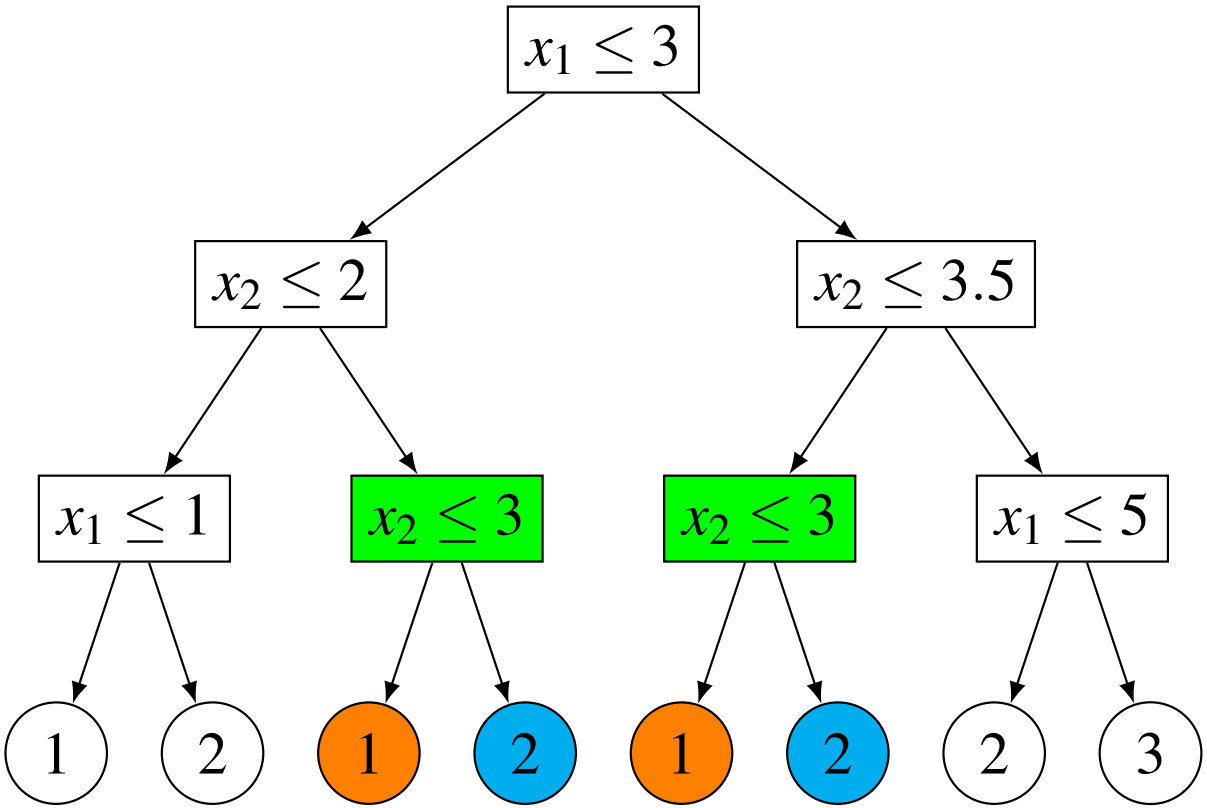
RWTH AACHEN UNIVERSITY

Random Forests: Problem

- High memory consumption: $O(2^d)$
  - Memory consumption grows exponentially with the depth d of the trees
- Especially a problem for memory constraint scenarios. E.g.
  - Embedded systems
  - Mobile devices

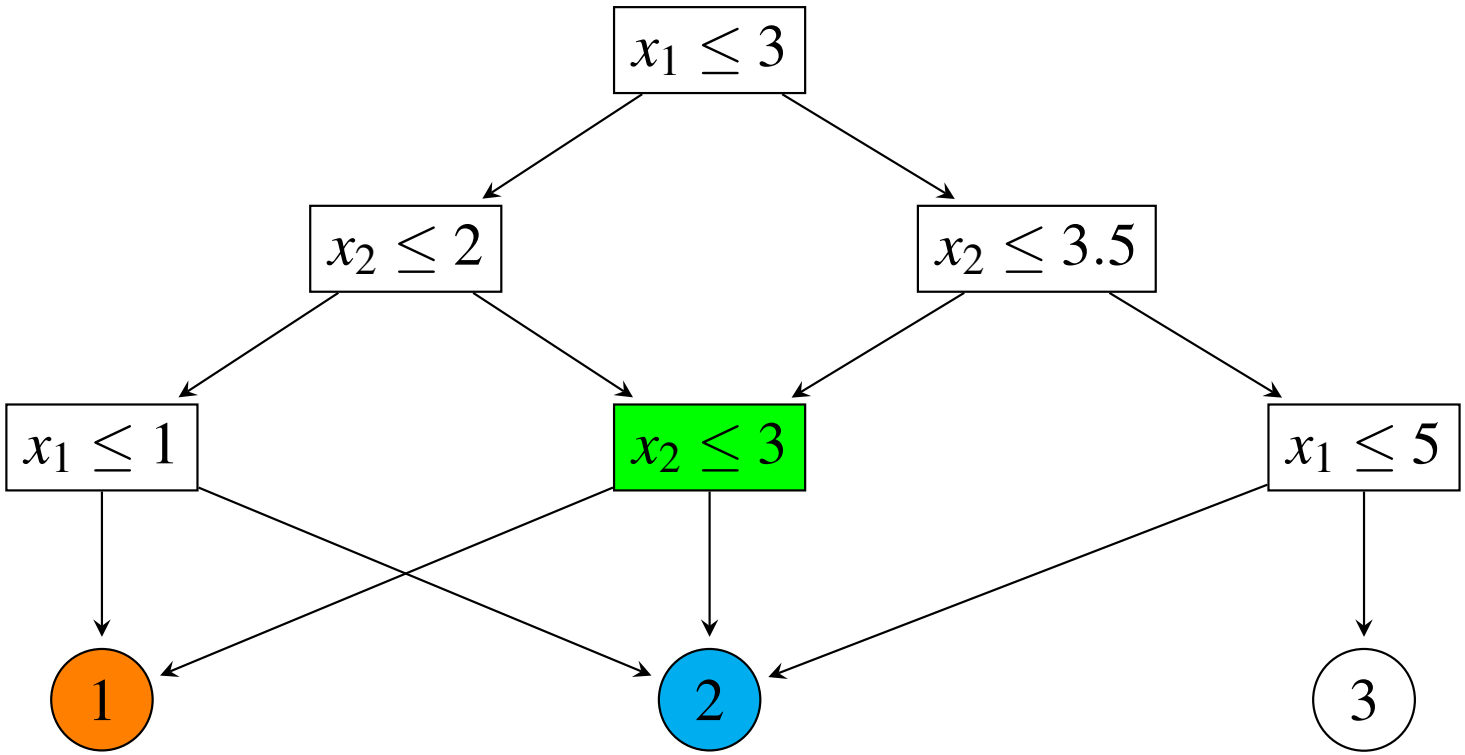
Decision DAGs: Concept

Idea: Instead of a tree graph, use a directed acyclic graph (DAG).



Decision DAGs: Concept

Idea: Instead of a tree graph, use a directed acyclic graph (DAG).

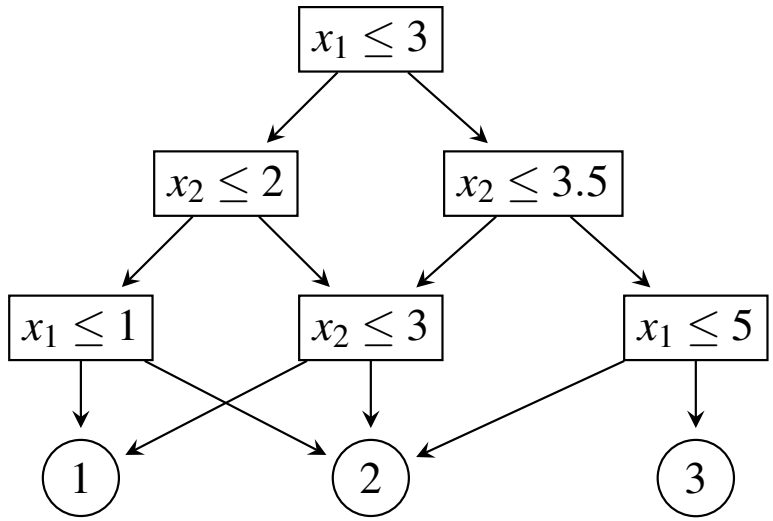


Decision DAGs: Memory consumption

Control the memory consumption by limiting the *width* of the DAG by a *merging schedule*

$$s : \mathbb{N} \mapsto \mathbb{N}, d \mapsto s(d)$$

If $s(d) \leq S \; \forall \; d \Rightarrow$, then the memory consumtion is $O(dS)$, where d is the depth.

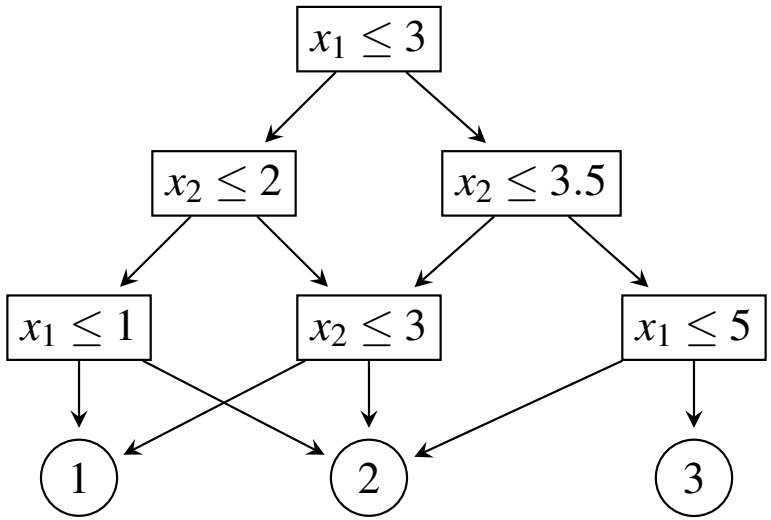


Decision DAGs: Memory consumption

A typical choice is

$$s : \mathbb{N} \mapsto \mathbb{N}, d \mapsto s(d) = \min(2^d, 2^D)$$

where $D \in \mathbb{N}$ is a constant.



Example *($D = 7$)*

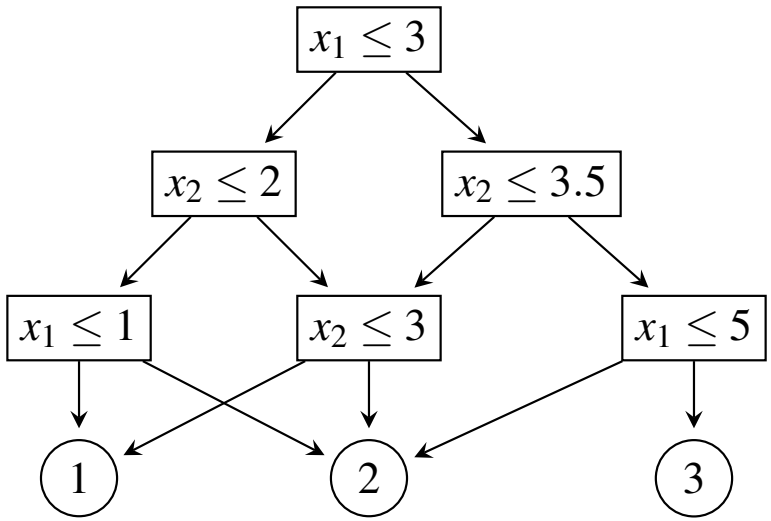
$$s : \mathbb{N} \mapsto \mathbb{N}, d \mapsto \min(2^d, 128)$$

Decision DAGs: Parameters

Definition

A *decision DAG* is a directed acyclic graph $G = (V, E)$ with the follwing properties:
An internal node v is augmented with

- Feature dimension $d_v \in \{1, \ldots, n\}$
- Threshold $\theta_v \in \mathbb{R}$
- Left child node $l_v \in V$
- Right child node $r_v \in V$



Decision Jungles

Definition

A *random decision DAG* is a decision DAG whose parameters are sampled from some probability distribution.

Decision Jungles

Definition

A *random decision DAG* is a decision DAG whose parameters are sampled from some probability distribution.

Definition

A *decision jungle* $J = (G_1, \ldots, G_m)$ is an ensemble of random decision DAGs G_i.

Decision jungles were proposed by J. Shotton et al. at NIPS 2013 [2].

Decision DAGs: Training

Binary decision trees

At each node v optimize

- the feature d_v
- the threshold θ_v

Decision DAGs

At each node v optimize

- the feature d_v
- the threshold θ_v
- the left child node l_v
- the right child node r_v

RWTH AACHEN UNIVERSITY

Decision DAGs: Training

Binary decision trees

At each node v optimize

- the feature d_v
- the threshold θ_v

Decision DAGs

At each node v optimize

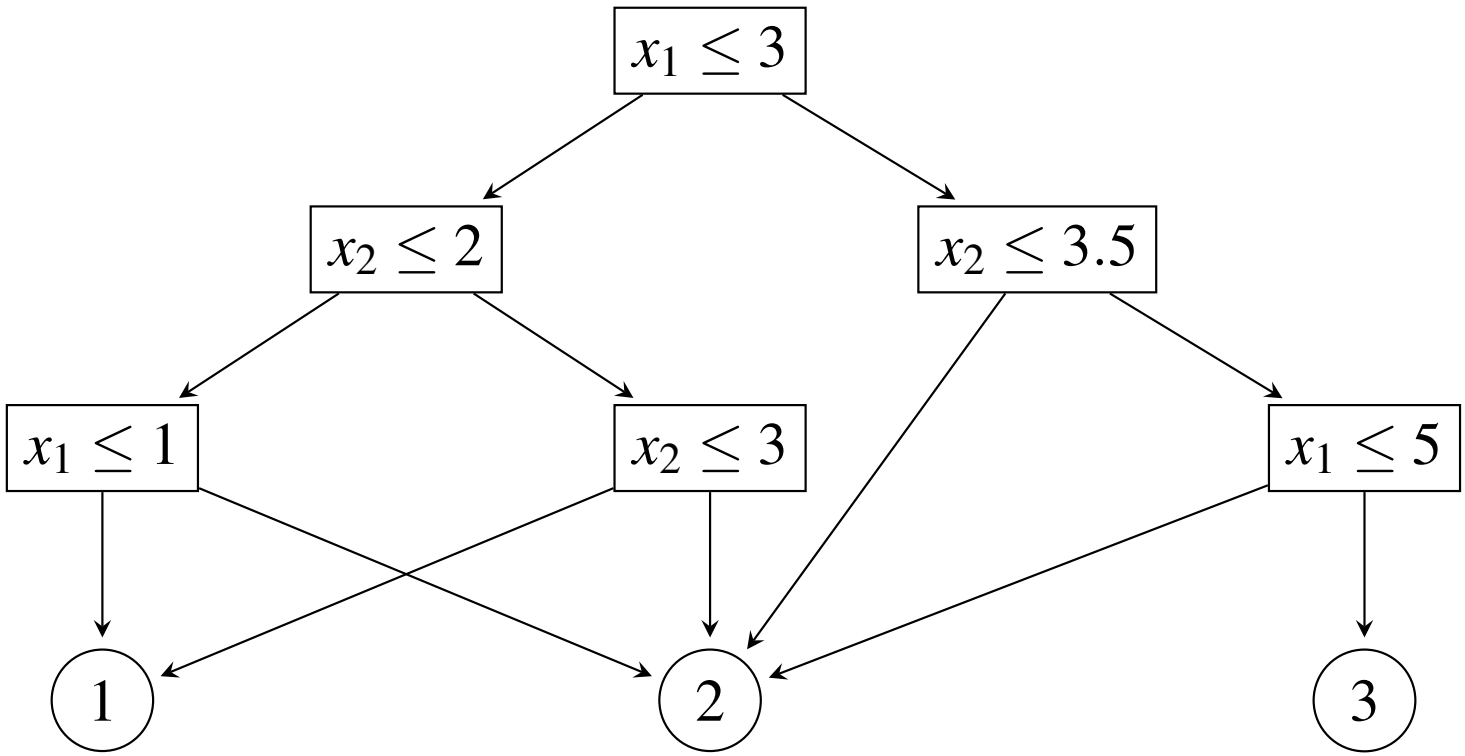
- the feature d_v
- the threshold θ_v
- the left child node l_v
- the right child node r_v

Conclusion

The graph structure and the thresholds/features need to be optimized simultaneously.

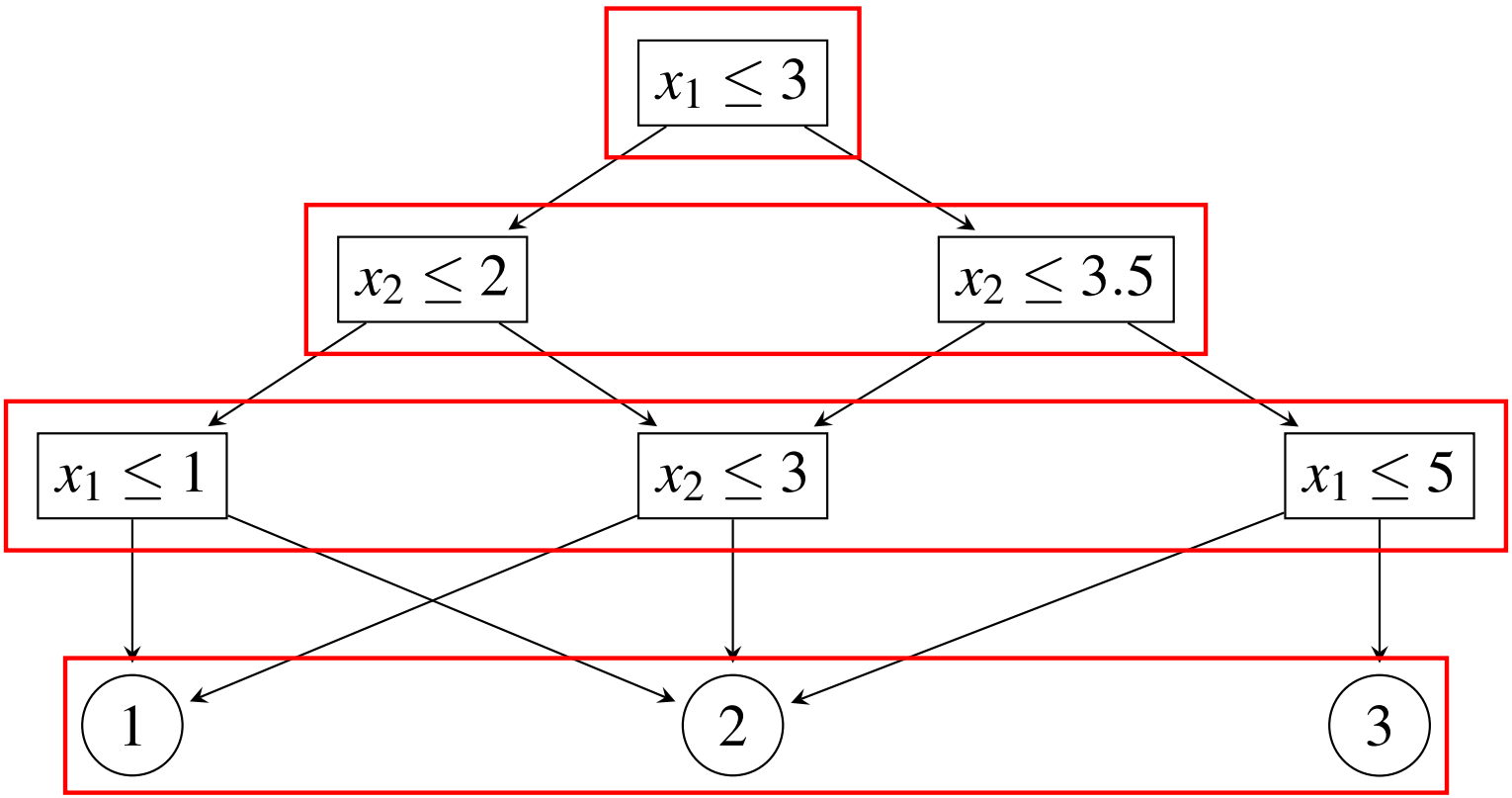
Decision DAGs: Training

Technically, this is also a decision DAG.



Decision DAGs: Training

Shotton et al. assumed a *level-wise* graph structure for optimization.



Decision DAGs: Training

The DAG is trained *level-wise*. Let s be a merging schedule (e.g. $s(d) = \min(2^d, 128)$).

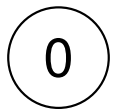
Decision DAGs: Training

The DAG is trained *level-wise*. Let s be a merging schedule (e.g. $s(d) = \min(2^d, 128)$).

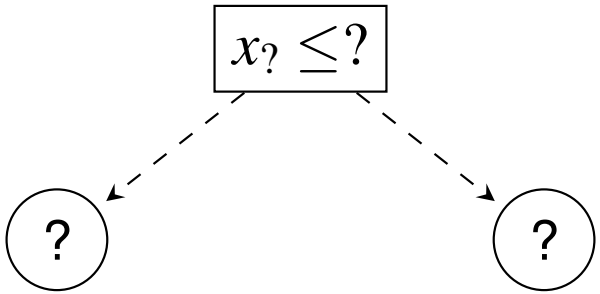
1: $G \leftarrow (\{root\}, \emptyset)$
2: **for** d = 1,2,.. **do**
3: Add $s(d)$ new nodes to G
4: Initialize the parameters of the former leaf nodes
5: Optimize the parameters of the former leaf nodes
6: **end for**

RWTH AACHEN UNIVERSITY

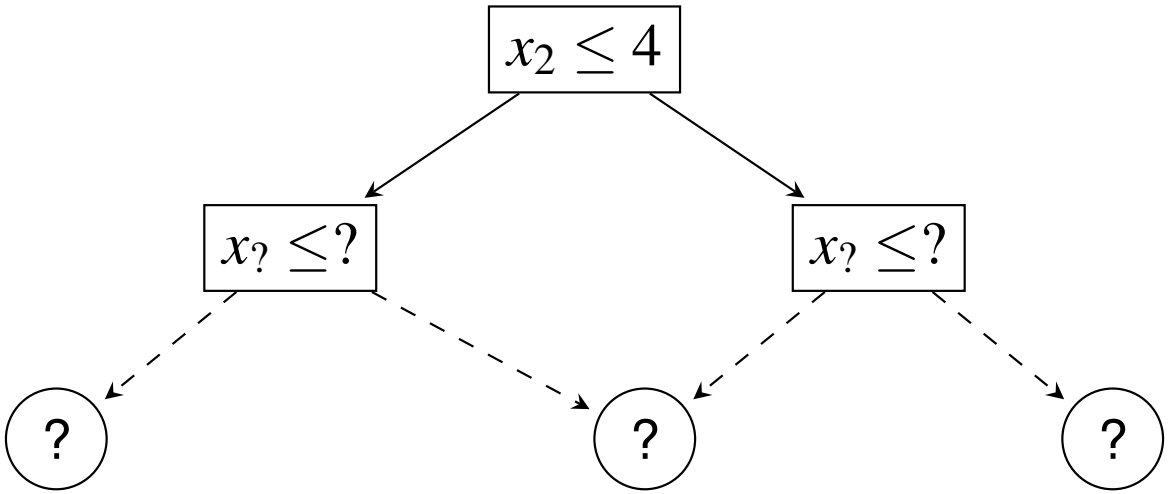
Decision DAGs: Training



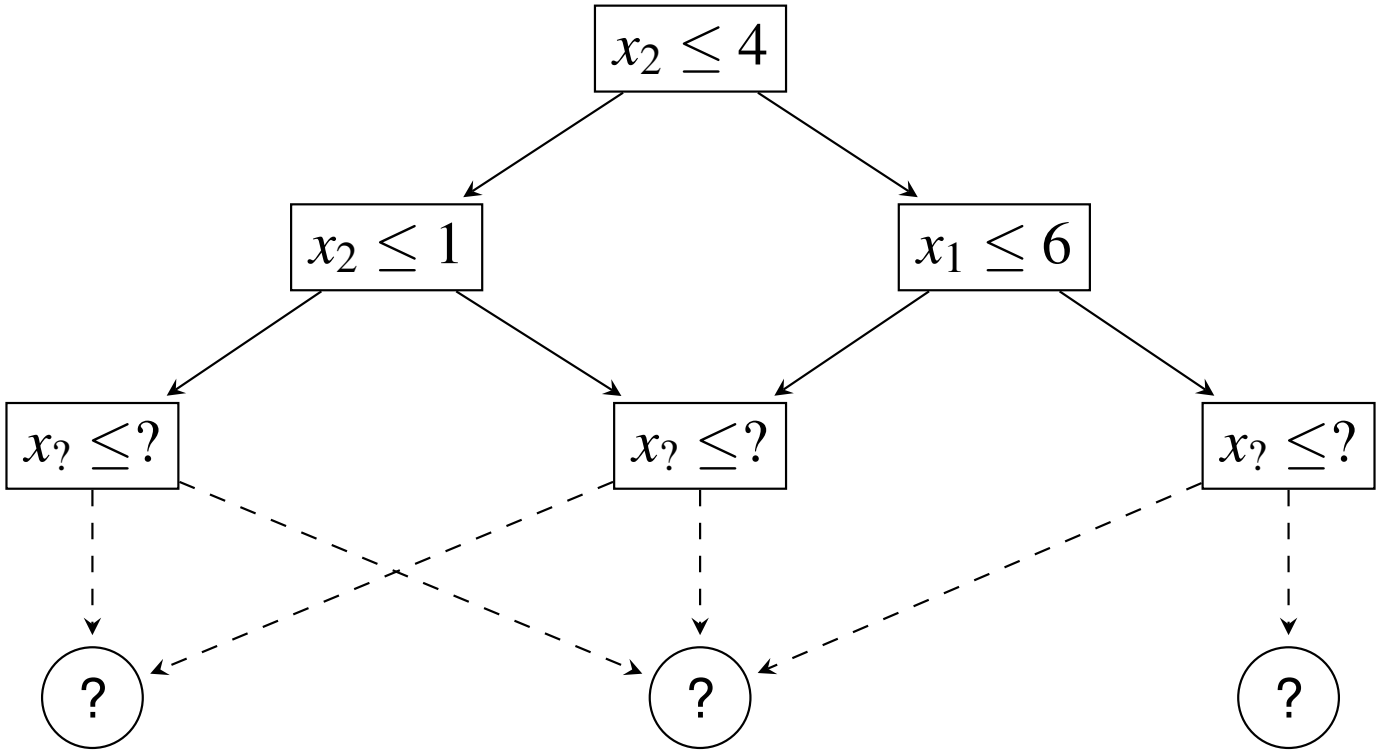
Decision DAGs: Training



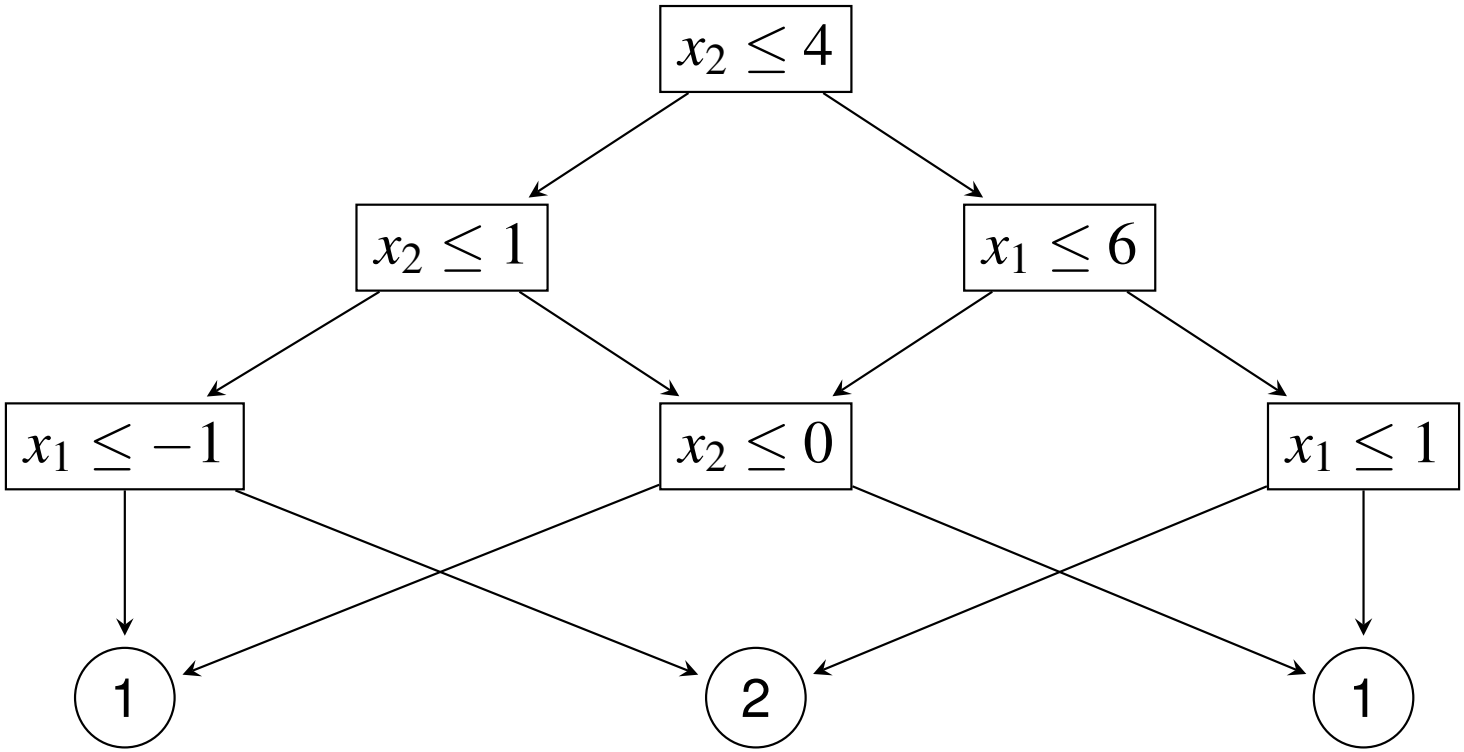
Decision DAGs: Training



Decision DAGs: Training

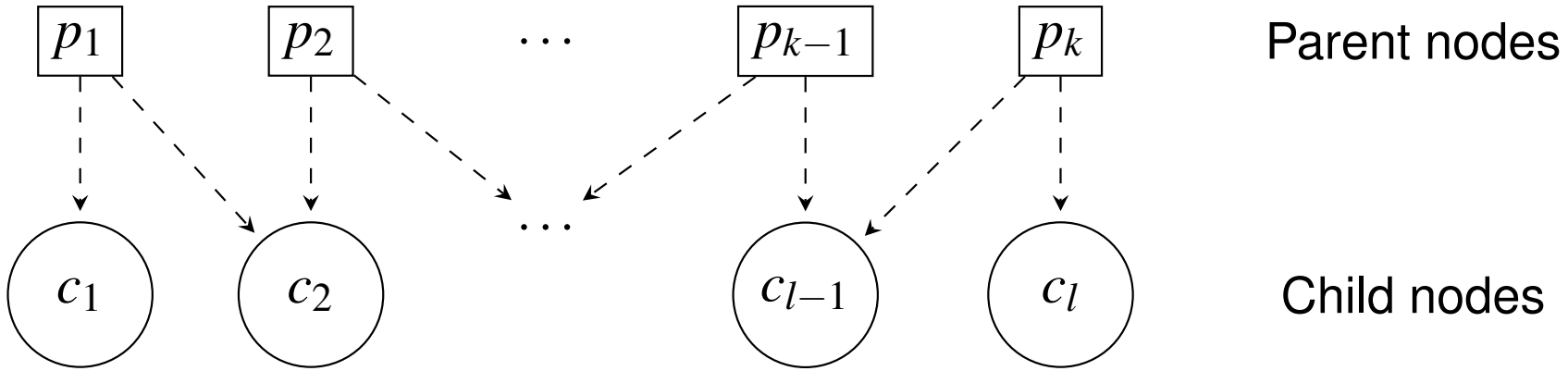


Decision DAGs: Training



Level Optimization

Naming convention:



- feature dimension d_{p_i}
- threshold θ_{p_i}
- left child node l_{p_i}
- right child node r_{p_i}

- S_{p_i} and S_{c_j} are the training sets at nodes p_i and c_j respectively

Objective Function I

Goal: Find the *optimal* parameters for the parent nodes in terms of an objective function E.

Definition

Let $X \subset \mathbb{R}^n \times \{1, \ldots, C\}$ be a training set. The entropy $H(X)$ is defined as

$$H(X) = -\sum_{i=1}^{C} p(i) \log_2 p(i)$$

where

$$p(i) = \frac{|\{(\mathbf{x}, y) \in X : y = i\}|}{|X|}$$

Objective Function II

The objective function E is defined in terms of the entropies at the child nodes.

$$E(\Theta_1, \ldots, \Theta_k) = \sum_{i=1}^{l} |S_{c_i}| H(S_{c_i})$$

where

- $\Theta_i = (d_{p_i}, \theta_{p_i}, l_{p_i}, r_{p_i})$ are the parameters of p_i

Objective Function III

The connection between the $\Theta_1, \ldots, \Theta_k$ and the $S_{c_1}, \ldots, S_{c_l}$ becomes apparent when looking at the definition of S_{c_i}:

$$S_{c_i} = \bigcup_{j=1,..,k \,:\, l_{p_j}=c_i} \{(\mathbf{x}, y) \in S_{p_j} : x_{d_{p_j}} \leq \theta_{p_j}\} \cup \bigcup_{j=1,..,k \,:\, r_{p_j}=c_i} \{(\mathbf{x}, y) \in S_{p_j} : x_{d_{p_j}} > \theta_{p_j}\}$$

LSEARCH Optimization Algorithm

1: **function** LSEARCH($\Theta_{p_1}, \ldots, \Theta_{p_k}$)
2: **while** something changes **do**
3: **for** $i = 1, \ldots, k$ **do**
4: $\mathcal{F} \leftarrow$ random feature selection
5: $(d_{p_i}, \theta_{p_i}) \leftarrow \arg\min_{d \in \mathcal{F}, \theta \in R} E(\ldots, \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, \ldots)$
6: **end for**
7: **for** $i = 1, \ldots, k$ **do**
8: $l_{p_i} \leftarrow \arg\min_{l=c_1,\ldots,c_l} E(\ldots, \Theta_{p_{i-1}}, (d_{p_i}, \theta_{p_i}, l, r_{p_i}), \Theta_{p_{i+1}}, \ldots)$
9: $r_{p_i} \leftarrow \arg\min_{r=c_1,\ldots,c_l} E(\ldots, \Theta_{p_{i-1}}, (d_{p_i}, \theta_{p_i}, l_{p_i}, r), \Theta_{p_{i+1}}, \ldots)$
10: **end for**
11: **end while**
12: **return** $\Theta_{p_1}, \ldots, \Theta_{p_k}$
13: **end function**

RWTH AACHEN UNIVERSITY

Intermediate Discussion

This is where the technical section of the paper ends.

Intermediate Discussion

This is where the technical section of the paper ends.

Open questions

- Does the algorithm converge to a local minimum?
- Does the algorithm terminate in a finite number of steps?
- How to implement the two minimization steps efficiently?

Intermediate Discussion

This is where the technical section of the paper ends.

Open questions

- Does the algorithm converge to a local minimum?
- Does the algorithm terminate in a finite number of steps?
- How to implement the two minimization steps efficiently?

Main issue

- There is no code available

In the following, I present the findings of my research.

Termination Theorem

Theorem

The LSEARCH algorithm terminates.

Termination Theorem Proof I

Proof

- E takes on a finite number of discrete values
  - There are only finitely many combinations of d_v, l_r and r_v
  - There are infinitely many choices for θ_v
    - We can factorize $\mathbb{R}$ using the following relation

$$x \sim y :\Leftrightarrow \forall \lambda \in [0,1] : E(d,x,l,r) = E(d,\lambda x + (1-\lambda)y,l,r)$$

    - $\mathbb{R}/\sim$ is finite
  - The joined parameter space is finite

Termination Theorem Proof II

Proof

- If the algorithm did not terminate, it would cycle through some configurations
- Let $\gamma_1, \ldots, \gamma_r$ be those configurations of $\Theta_1, \ldots, \Theta_k$
  - $\gamma_{i+1} = \text{LSEARCH}(\gamma_i)$ and $\gamma_1 = \text{LSEARCH}(\gamma_r)$
- Observation: Parameters only change when the objective function decreases
- Hence

$$E(\gamma_1) > E(\gamma_2) > \ldots > E(\gamma_{r-1}) > E(\gamma_r)$$

Termination Theorem Proof III

Proof

- But because of the cycling, it must also hold

$$E(\gamma_r) > E(\gamma_1)$$

Therefore

$$E(\gamma_1) > E(\gamma_1)$$

Hence, the algorithm terminates.

Optimality Theorem

From the termination proof, the following theorem follows immidiately.

Theorem

The LSEARCH optimization algorithm converges to a local minimum of the objective function in a finite number of iterations.

Optimality Theorem

From the termination proof, the following theorem follows immidiately.

Theorem

The LSEARCH optimization algorithm converges to a local minimum of the objective function in a finite number of iterations.

Proof

Termination theorem + Only parameters which decrease the objective function are accepted.

Implementation

- Efficiently implementing the algorithm is not trivial
- Evaluating the objective function is expensive

$$E(\Theta_1, \ldots, \Theta_k) = \sum_{i=1}^{l} |S_{c_i}| H(S_{c_i})$$

  - First the S_{c_i} have to be determined
  - Then the entropies have to be calculated
- Exploit the problem structure in order to find an efficient implementation

RWTH AACHEN UNIVERSITY

Threshold Optimization I

1: $(d_{p_i}, \theta_{p_i}) \leftarrow \arg\min_{d \in \mathcal{F}, \theta \in R} E(\ldots, \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, \ldots)$

First we note that only $S_{l_{p_i}}$ and $S_{r_{p_i}}$ can change.

Threshold Optimization I

1: $(d_{p_i}, \theta_{p_i}) \leftarrow \arg\min_{d \in \mathcal{F}, \theta \in R} E(\ldots, \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, \ldots)$

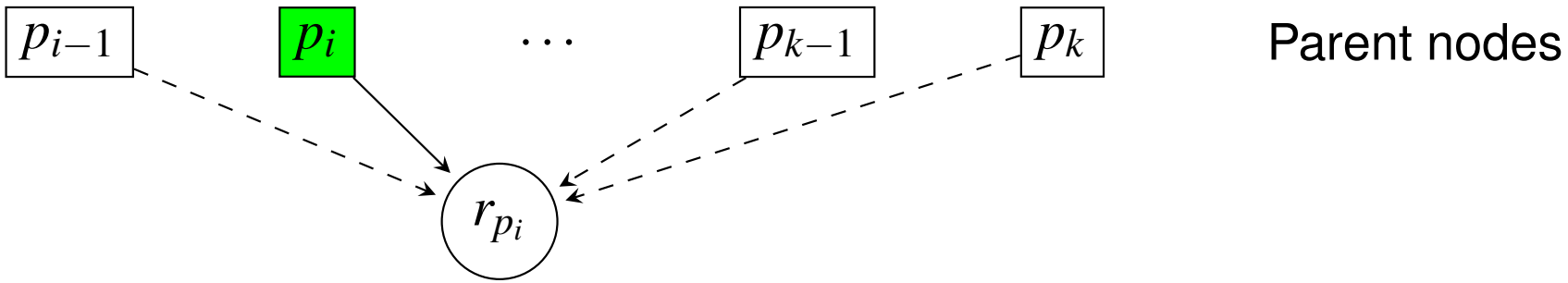
First we note that only $S_{l_{p_i}}$ and $S_{r_{p_i}}$ can change.

Corollary

It holds

$$\begin{aligned}
&\arg\min_{d \in \mathcal{F}, \theta \in R} E(\ldots, \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, \ldots) \\
=&\arg\min_{d \in \mathcal{F}, \theta \in R} \sum_{i=1}^{l} |S_{c_i}| H(S_{c_i}) \\
=&\arg\min_{d \in \mathcal{F}, \theta \in R} |S_{l_{p_i}}| H(S_{l_{p_i}}) + |S_{r_{p_i}}| H(S_{r_{p_i}})
\end{aligned}$$

Threshold Optimization II



Observation

- There is only a constant contribution from the other parents
- Only the contribution from p_i varies

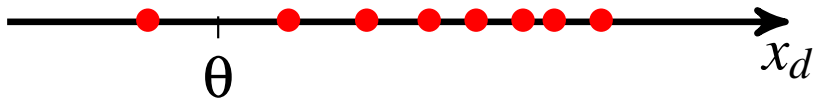
Idea

- Precompute the contribution from the other parent nodes in histograms

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points

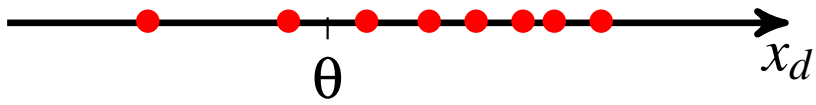


- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points

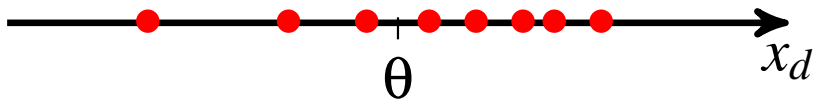


- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points

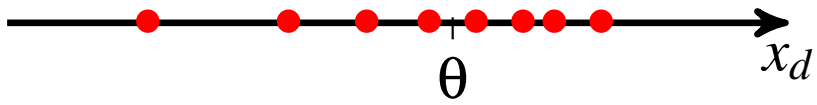


- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points

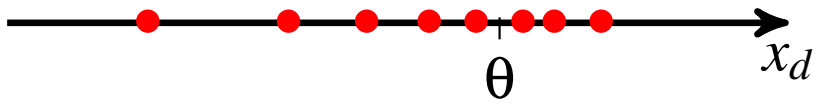


- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points

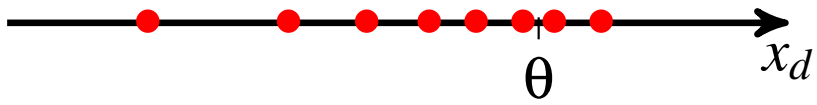


- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points

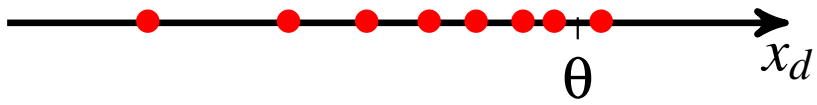


- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization III

Testing multiple thresholds for a fixed feature dimension efficiently

- Sort the training set according the the feature dimension
- Subsequently test thresholds between neighboring points



- At each iteration, only a single data points moves from the right to the left child node
- This technique is due to Piotr Dollár [3]

Threshold Optimization: Discussion

In Summary

- Precompute the contributions from the other parents
- Subsequently test different thresholds
- These steps allow us to evaluate the objective function at each iteration in constant time

Threshold Optimization: Discussion

In Summary

- Precompute the contributions from the other parents
- Subsequently test different thresholds
- These steps allow us to evaluate the objective function at each iteration in constant time

Notes

- The steps are proven to be correct
- Derivations are rather technical
- See the seminar paper for formal details

Experiments from the Paper

Kincet Body Dataset [5]

- Estimate a human pose from a single depth image
- 31 classes

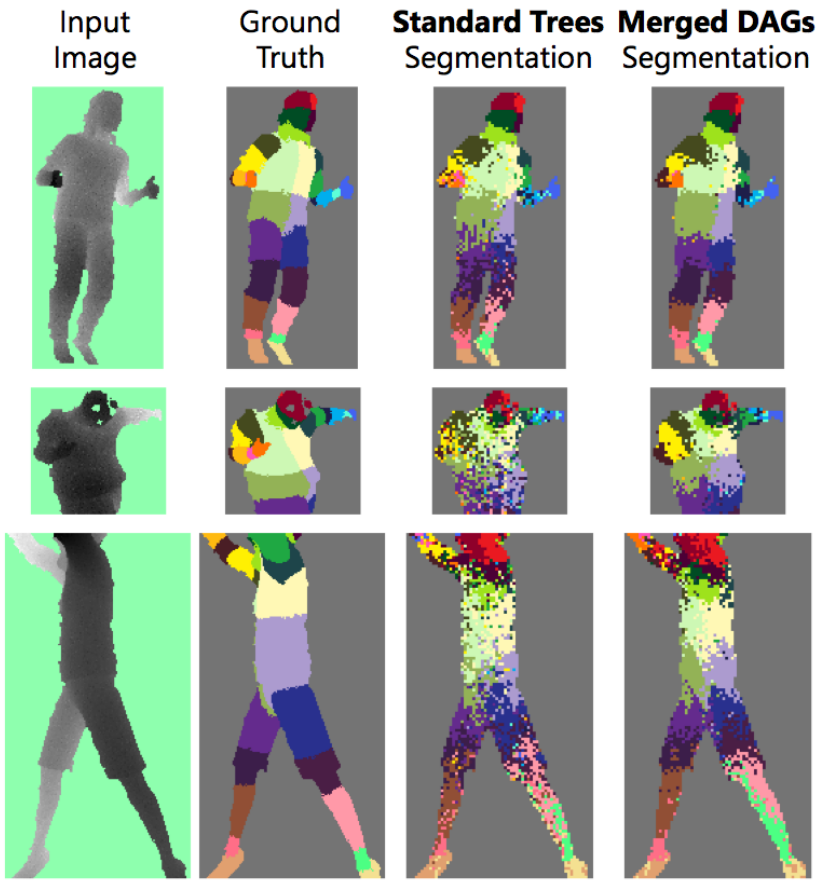


Image by Shotton et al. [2]

Results: Test accuracy

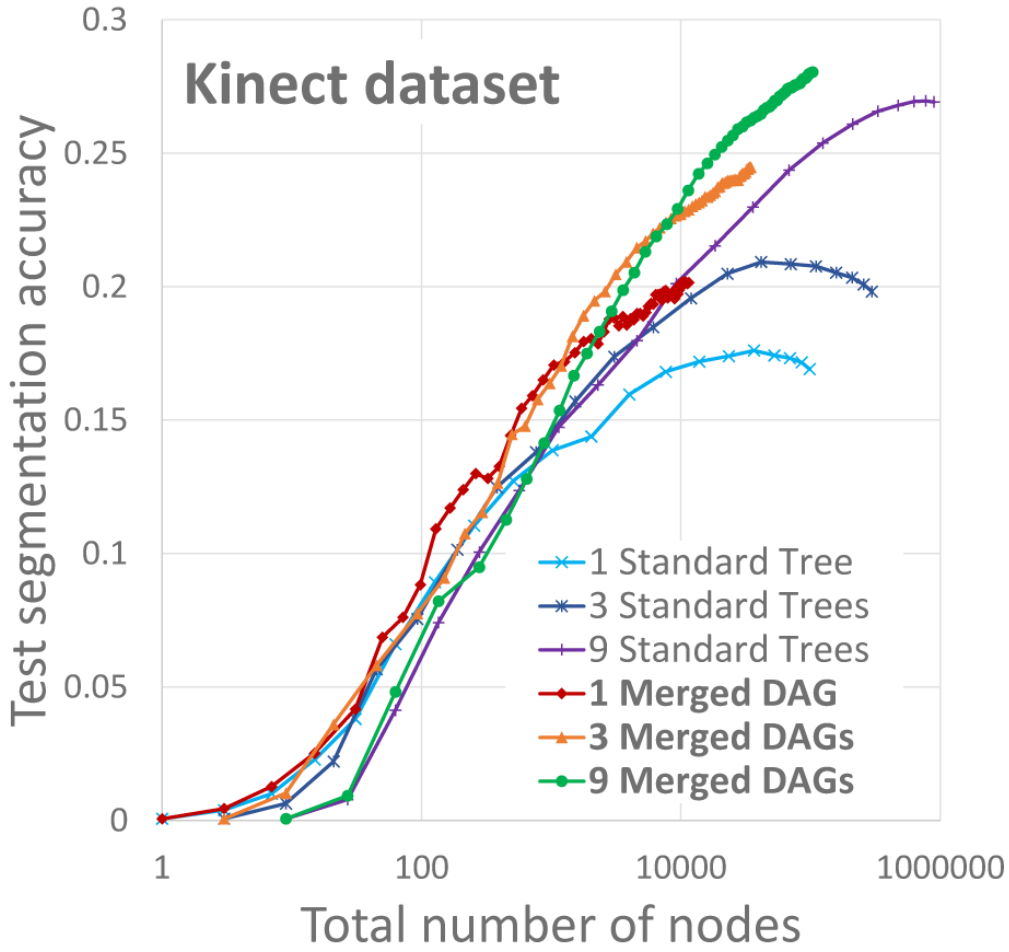


Image by Shotton et al. [2]

Results: Feature evaluations

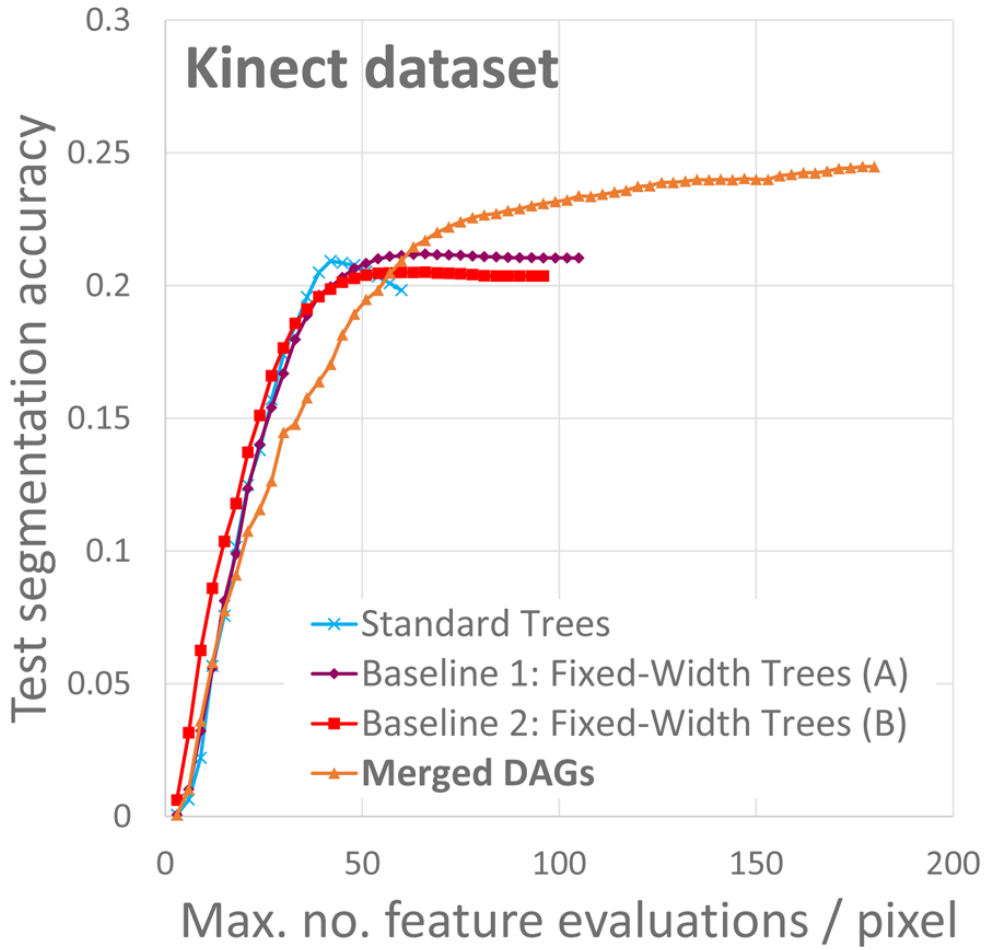


Image by Shotton et al. [2]

Interpretation

Conclusions

- Decision DAGs trained using the LSEARCH algorithm...
  - consume less memory than binary decision trees
  - perform significantly better when compared to trees of the same size (i.e. same number of nodes)

Interpretation

Conclusions

- Decision DAGs trained using the LSEARCH algorithm...
  - consume less memory than binary decision trees
  - perform significantly better when compared to trees of the same size (i.e. same number of nodes)
- The proposed DAG structure works better than trees of fixed width
  - Fixed-width tree: At each level, only split the M nodes that have the highest entropy

Interpretation

Conclusions

- Decision DAGs trained using the LSEARCH algorithm...
  - consume less memory than binary decision trees
  - perform significantly better when compared to trees of the same size (i.e. same number of nodes)
- The proposed DAG structure works better than trees of fixed width
  - Fixed-width tree: At each level, only split the M nodes that have the highest entropy

Questions

- How do decision jungles perform compared to random forests (disregarding model size)?
  - Training time?
  - *Absolute* test accuracy?
  - Evaluation time?

RWTH AACHEN UNIVERSITY

My Experiments

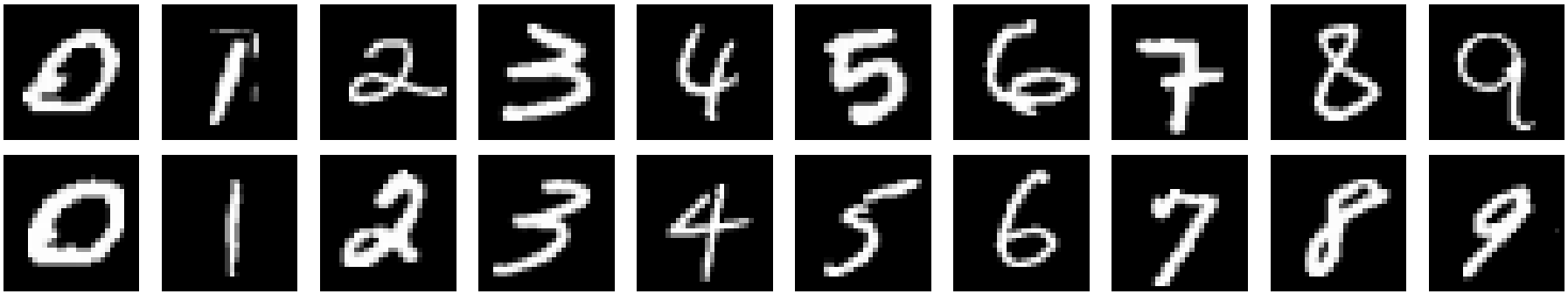
- Decision jungle results are obtained using my LibJungle C++ library [4]
  - Efficient multi-threaded implementation of decision jungles

RWTH AACHEN UNIVERSITY

My Experiments

- Decision jungle results are obtained using my LibJungle C++ library [4]
  - Efficient multi-threaded implementation of decision jungles
- Baseline results are obtained using Piotr Dollár's MATLAB Toolbox [3]
  - Very efficient and well tested implementation of random forests
  - Fair comparison: Crucial parts are implemented in C

Evaluation Data: MNIST Data Set



- Handwritten digits 0-9 (10 classes)
  - Grayscale images
  - 28×28 pixels
- 60,000 training images
- 10,000 test images
- Available under `http://yann.lecun.com/exdb/mnist/`

Experiment 1: Iteration Limit

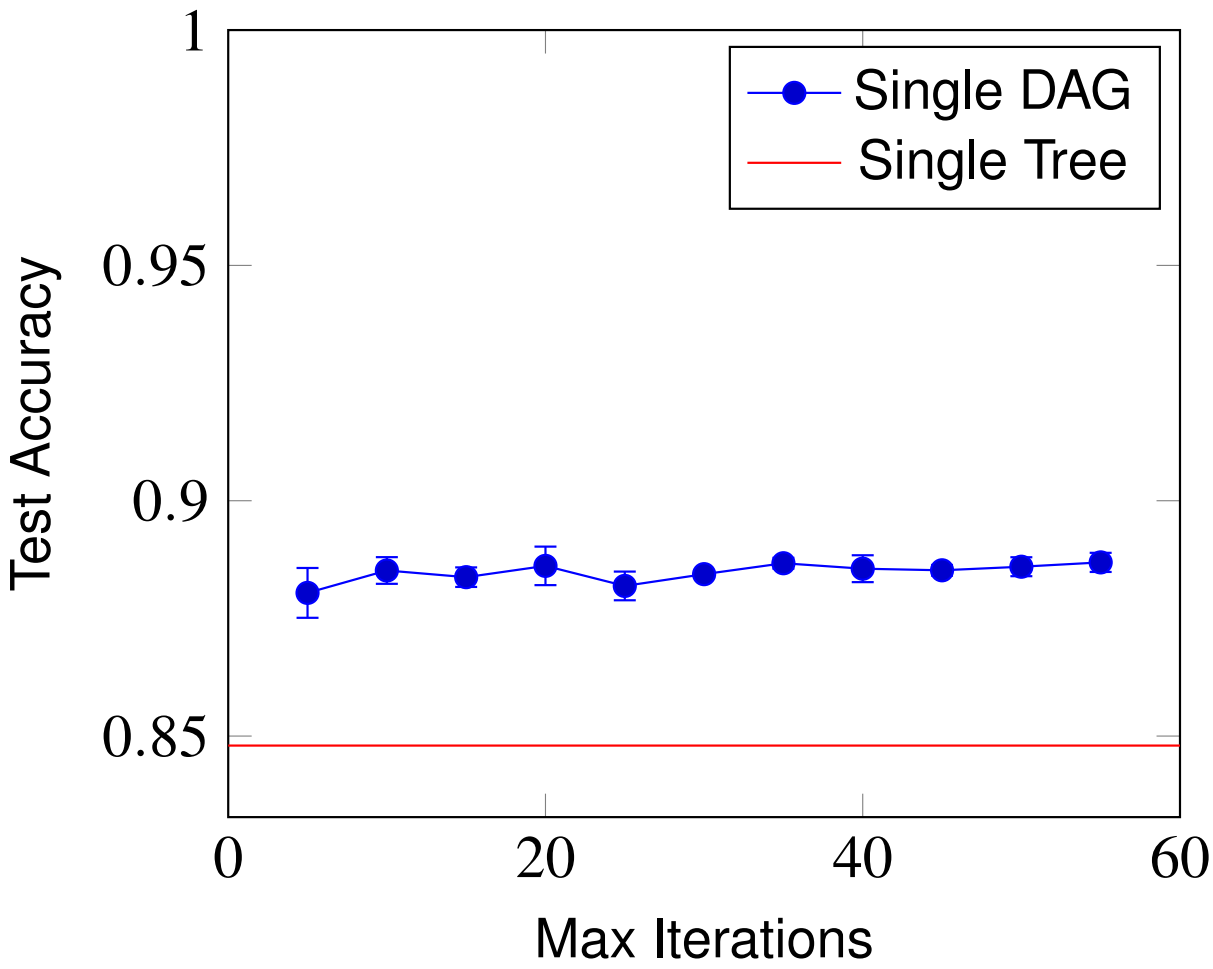
Algorithm

1: **function** LSEARCH($\Theta_{p_1}, \ldots, \Theta_{p_k}$)
2: **while** something changes **do**
3: ...
4: **end while**
5: **return** $\Theta_{p_1}, \ldots, \Theta_{p_k}$
6: **end function**

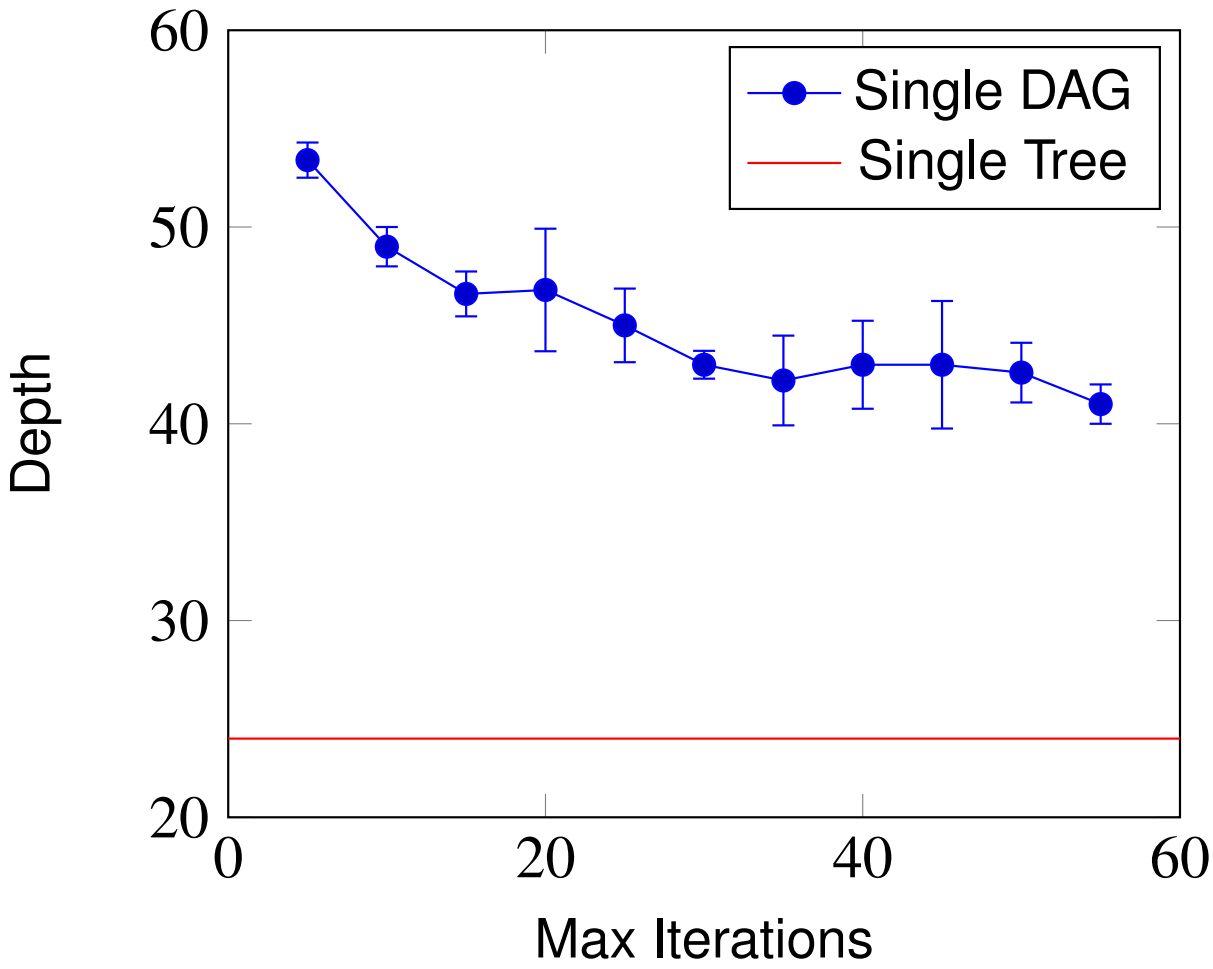
Experiment

- We set an iteration limit on the outer `while`-loop in the LSEARCH optimization algorithm
- Evaluate the performance of a single DAG vs. a single tree

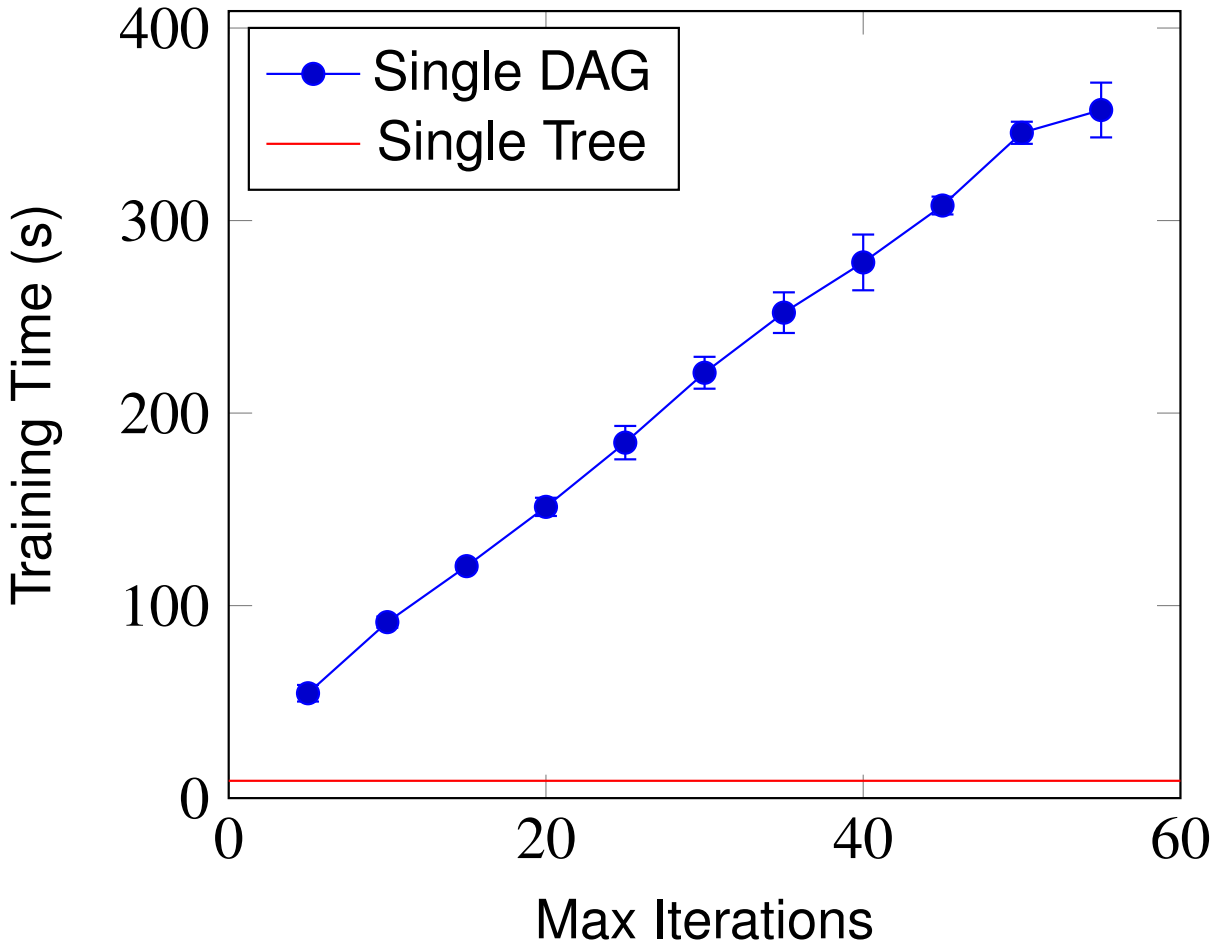
Results: Test Accuracy



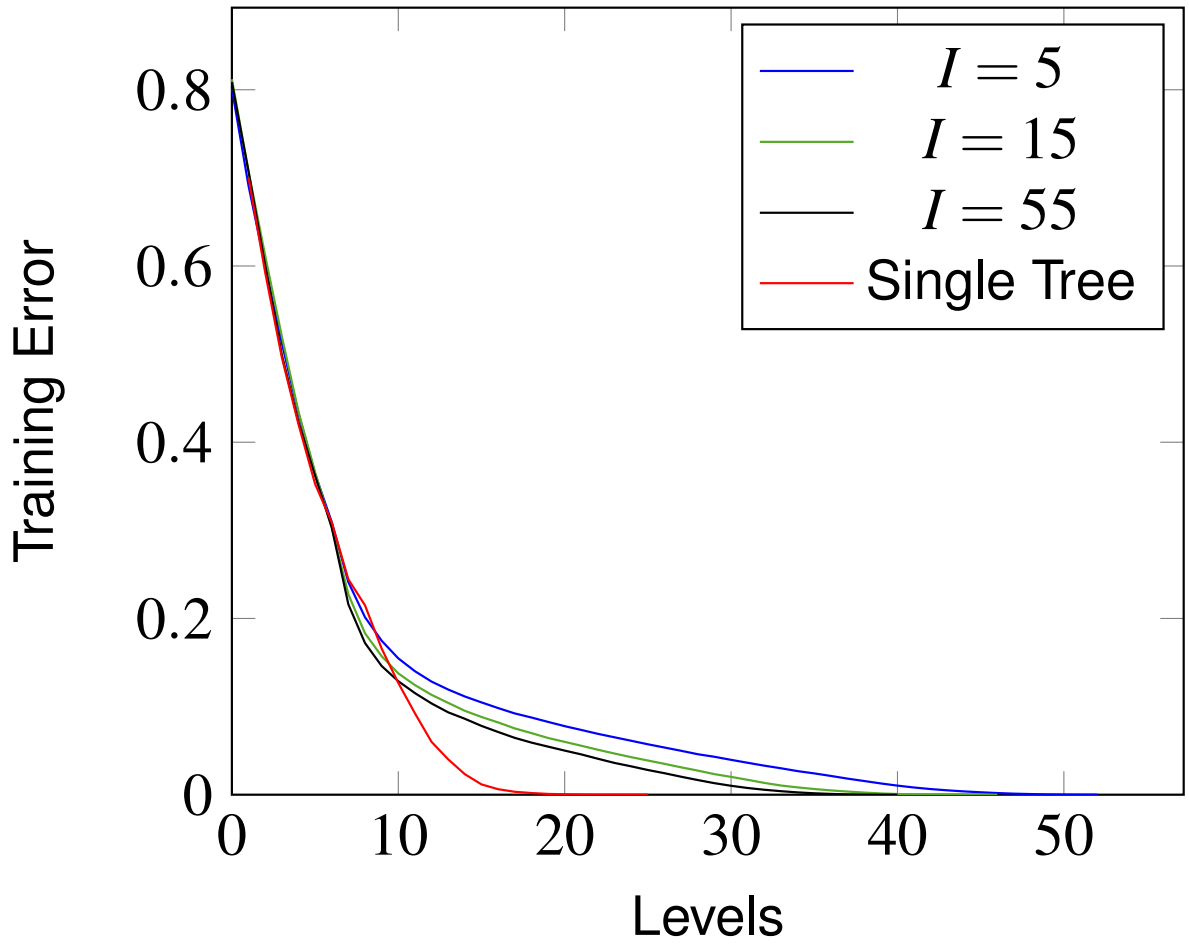
Results: Depth



Results: Training Time



Results: Convergence Speed



Experiment 1: Interpretation

Pros

- DAGs outperform trees by a large margin
- DAGs consume considerable less memory

Experiment 1: Interpretation

Pros

- DAGs outperform trees by a large margin
- DAGs consume considerable less memory

Cons

- Evaluation time for DAGs is twice the time for trees
- Training DAGs takes significantly longer than training trees

Experiment 2: Ensembles

Question

- How do decision jungles perform compared to random forests?

Experiment 2: Ensembles

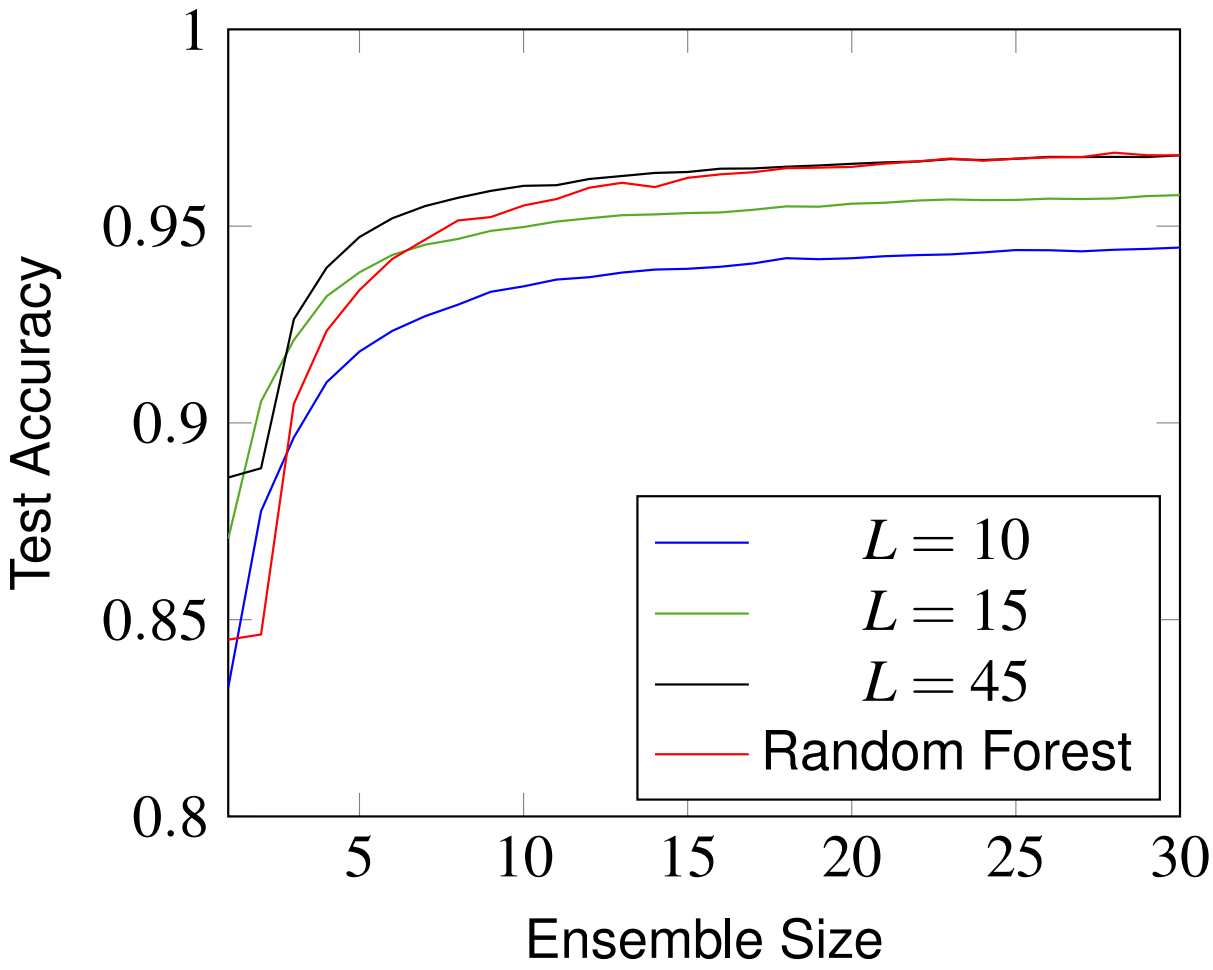
Question

- How do decision jungles perform compared to random forests?

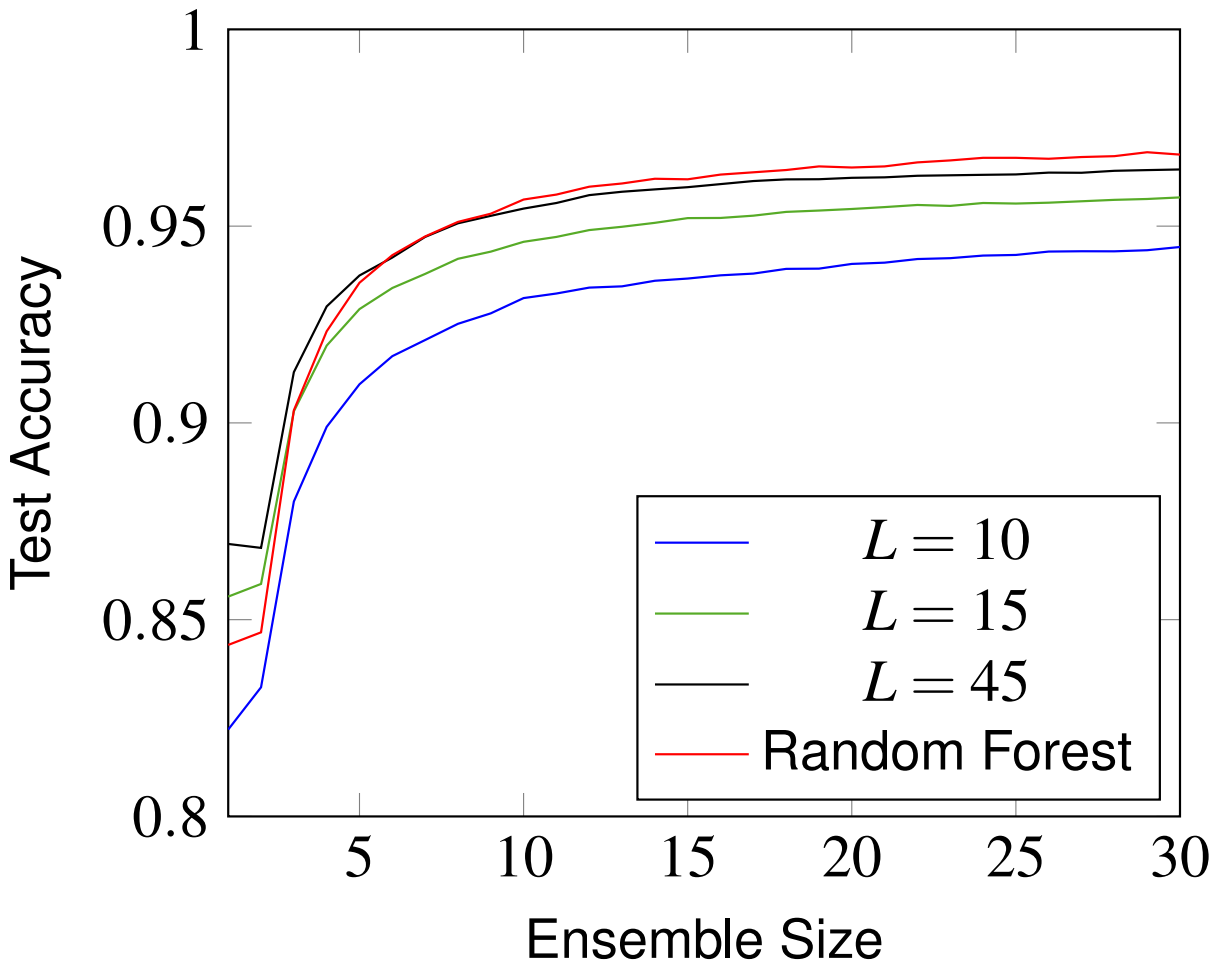
Experiment

- Train up to 30 DAGs/trees
- Evaluate the performance of the ensemble each time after adding a DAG/tree
- Perform the experiment for different depth limits (10,15,45)
- Perform the experiment with bagging and without bagging

Results: Without Bagging



Results: With Bagging



Experiment 3

Algorithm

1: $G \leftarrow (\{root\}, \emptyset)$
2: **for** d = 1,2,.. **do**
3: Add $s(d)$ new nodes to G
4: Initialize the parameters of the former leaf nodes
5: Optimize the parameters of the former leaf nodes
6: **end for**

Experiment 3

Algorithm

1: $G \leftarrow (\{root\}, \emptyset)$
2: **for** d = 1,2,.. **do**
3: Add $s(d)$ new nodes to G
4: Initialize the parameters of the former leaf nodes
5: Optimize the parameters of the former leaf nodes
6: **end for**

Two possibilities

- Initialize parameters randomly
- Initialize l_{p_i} and r_{p_i} such that parent nodes with high entropy do not have common child nodes

Experiment 3

Algorithm

1: $G \leftarrow (\{root\}, \emptyset)$
2: **for** d = 1,2,.. **do**
3: Add $s(d)$ new nodes to G
4: Initialize the parameters of the former leaf nodes
5: Optimize the parameters of the former leaf nodes
6: **end for**

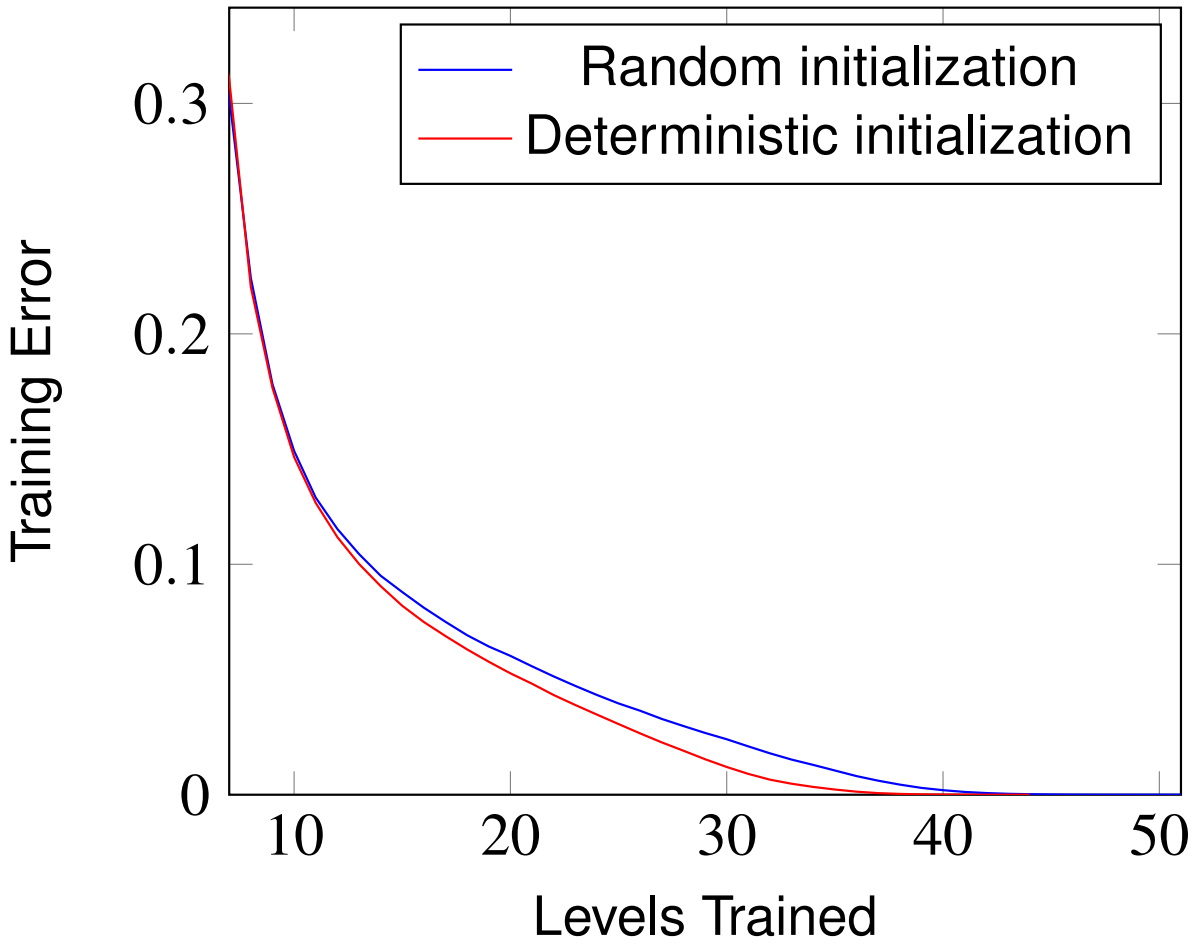
Two possibilities

- Initialize parameters randomly
- Initialize l_{p_i} and r_{p_i} such that parent nodes with high entropy do not have common child nodes

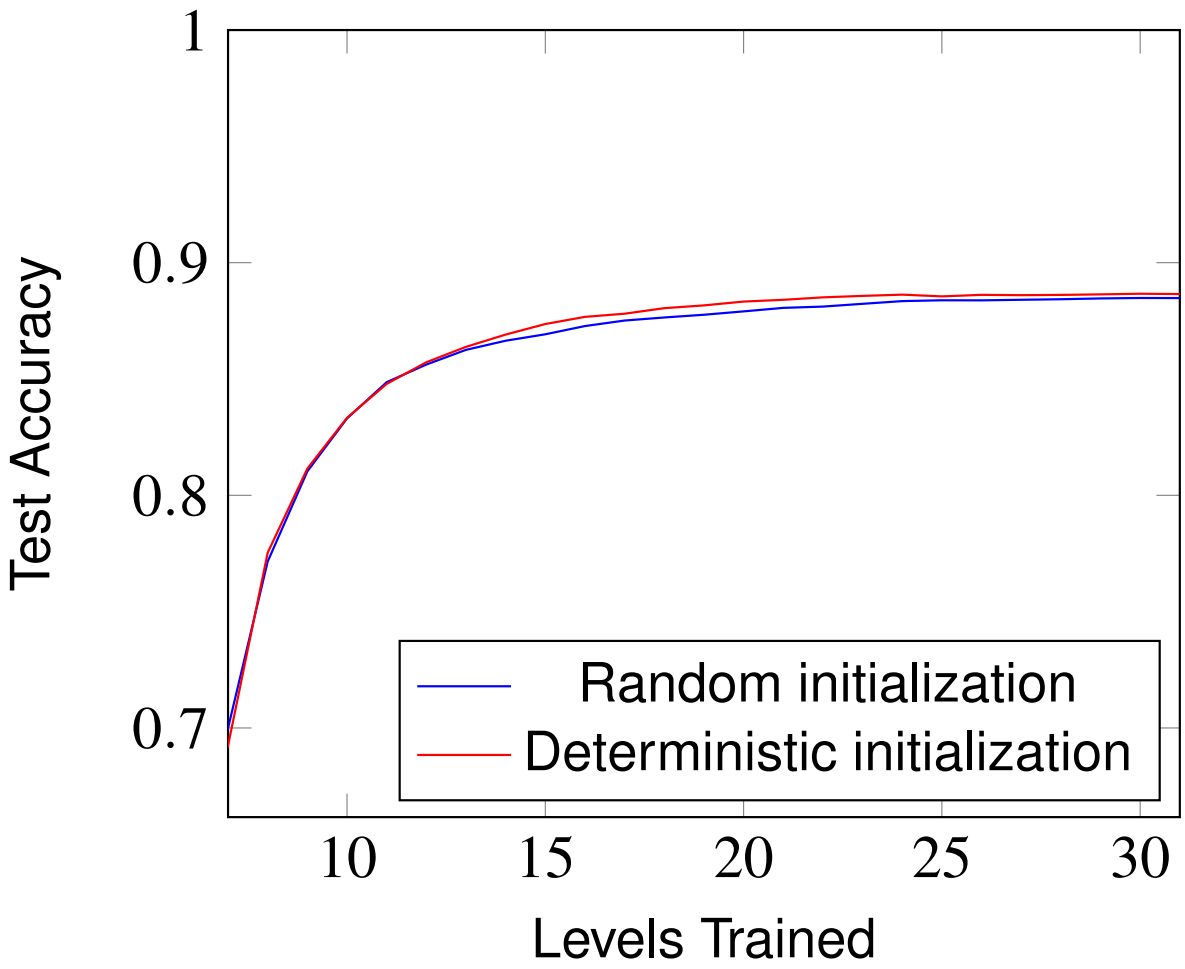
Goal

- Speed up convergence

Results: Convergence speed



Results: Test accuracy



RWTH AACHEN UNIVERSITY

Experiment 4: Various Data Sets

We compare the test accuracy of decision jungles and random forests.

Data set	Size	Features	Attributes	#DAGs
MNIST	60,000/10,000	784	numerical	8/15
USPS	3,823/1,797	64	numerical	8/15
CONNECT 4	67,557/-	42(126)	categorical	8/15
LETTER RECOG.	20,000/-	16	numerical	8/15
SHUTTLE	43,500/14,500	9	numerical	8/15

Data sets are from the UCI Machine Learning Repository [6].

RWTH AACHEN UNIVERSITY

Experiment 4: Results I

Data set	Decision jungles 8 DAGs Mean	Decision jungles 8 DAGs Stdev.	Random forests 8 Trees Mean	Random forests 8 Trees Stdev.
MNIST	**95.72%**	0.13%	95.14%	0.20%
USPS	**94.65%**	0.5%	94.44%	0.30%
CONNECT 4	**81.17%**	0.22%	80.99%	0.46%
LETTER RECOGNITION	**94.73%**	0.57%	94.29%	0.43%
SHUTTLE	99.98%	0.01%	**99.99%**	0.00%

DAGs are trained without bagging.

Experiment 4: Results II

Data set	Decision jungles 15 DAGs Mean	Decision jungles 15 DAGs Stdev.	Random forests 15 Trees Mean	Random forests 15 Trees Stdev.
MNIST	**96.38%**	0.09%	96.23%	0.16%
USPS	**95.95%**	0.2%	95.93%	0.52%
CONNECT 4	**81.98%**	0.15%	81.47%	0.66%
LETTER RECOGNITION	**95.73%**	0.55%	95.58%	0.48%
SHUTTLE	**99.99%**	0.01%	**99.99%**	0.01%

DAGs are trained without bagging.

RWTH AACHEN UNIVERSITY

LibJungle C++ Library

- C++ implementation of decision jungles
- Implements all speed-ups discussed in the seminar paper
- Can be used as a static library
- Open source license (BSD)
- Available under `https://bitbucket.org/geekStack/libjungle`

RWTH AACHEN UNIVERSITY

Summary

- Goal: Find memory efficient alternative to random forests

RWTH AACHEN UNIVERSITY

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width

RWTH AACHEN UNIVERSITY

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width
- Train ensembles of random decision DAGs (called decision jungle)

RWTH AACHEN UNIVERSITY

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width
- Train ensembles of random decision DAGs (called decision jungle)
- Train a DAG level-wise by minimizing an objective function

RWTH AACHEN UNIVERSITY

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width
- Train ensembles of random decision DAGs (called decision jungle)
- Train a DAG level-wise by minimizing an objective function
- Efficiently implement the optimization using histograms

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width
- Train ensembles of random decision DAGs (called decision jungle)
- Train a DAG level-wise by minimizing an objective function
- Efficiently implement the optimization using histograms
- Decision jungles perform as well as random forests
  - Sometimes even better

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width
- Train ensembles of random decision DAGs (called decision jungle)
- Train a DAG level-wise by minimizing an objective function
- Efficiently implement the optimization using histograms
- Decision jungles perform as well as random forests
  - Sometimes even better
- Evaluation is twice as expensive

Summary

- Goal: Find memory efficient alternative to random forests
- Idea: Use DAGs and limit their width
- Train ensembles of random decision DAGs (called decision jungle)
- Train a DAG level-wise by minimizing an objective function
- Efficiently implement the optimization using histograms
- Decision jungles perform as well as random forests
  - Sometimes even better
- Evaluation is twice as expensive
- Training takes significantly longer

Thanks for your attention

Questions are welcome

Seminar paper available under `geekstack.net/paper`

RWTH AACHEN UNIVERSITY

Further Reading I

Leo Breiman.
Random Forests.
Machine Learning 45, 2001.

Shotton, Jamie and Sharp, Toby and Kohli, Pushmeet and Nowozin, Sebastian and Winn, John and Criminisi, Antonio.
Decision Jungles: Compact and Rich Models for Classification.
Advances in Neural Information Processing Systems 26, 2013.

Piotr Dollár.
Piotr's Image and Video Matlab Toolbox (PMT).
`http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`.

Further Reading II


Tobias Pohlen.
LibJungle - Decision Jungle Library.
https://bitbucket.org/geekStack/libjungle.


Jamie Shotton and Ross Girshick and Andrew Fitzgibbon and Toby Sharp and Mat Cook and Mark Finocchio and Richard Moore and Pushmeet Kohli and Antonio Criminisi and Alex Kipman and Andrew Blake.
Efficient Human Pose Estimation from Single Depth Images.
IEEE Trans. Pattern Anal. Mach. Intell., 35, pages 2821-2840, 2013.


K. Bache and M. Lichman.
UCI Machine Learning Repository.
http://archive.ics.uci.edu/ml.