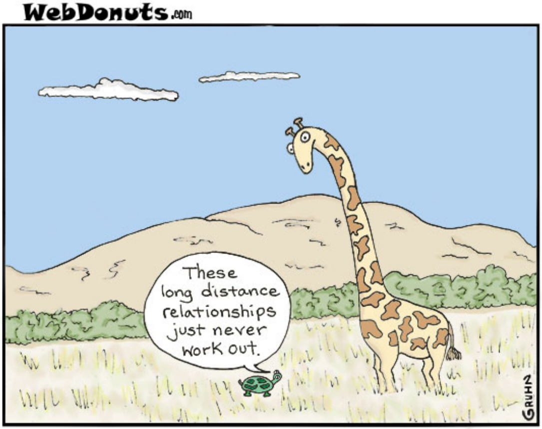
Faster nearest neighbor queries with simplified cover trees

by Mike Izbicki

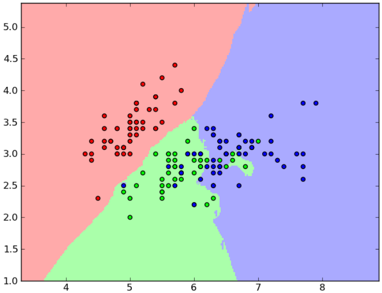


outline:

1. motivation
2. metric spaces
3. other data structures
4. simplified cover tree
5. nearest ancestor tree
6. experiments
7. open problems

Why care about cover trees?

They speed up nearest neighbor queries, e.g. in *k*-nn classification:

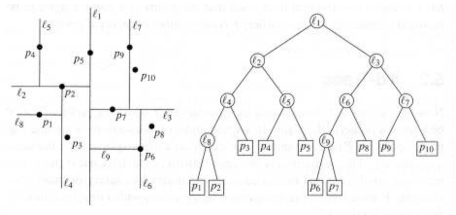


But neighbor queries are used in many other algorithms too:

- Localized support vector machines (Segata & Blanzieri, 2010)
- Dimensionality reduction (Lisitsyn *et. al.*, 2013)
- Reinforcement learning (Tziortziotis *et. al.*, 2014)

image from: `http://scikit-learn.org`

Nearest neighbor data structures for Euclidean distance

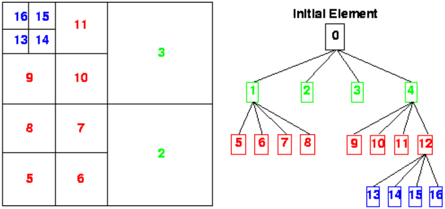


kd-tree

popular in machine learning

MLPack, scikit, R, matlab, weka

(Friedman *et al.*, 1977)

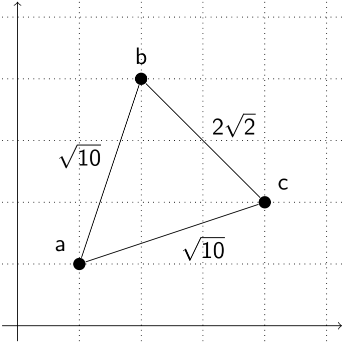


quad/oct-tree

not popular in machine learning

images from: `http://www.cs.sandia.gov/~kddevin/LB/figs`

Metric spaces generalize euclidean space



Euclidean distance:

$$\mathcal{X} = \mathbb{R}^n$$

$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^2 \right)^{\frac{1}{2}}$$

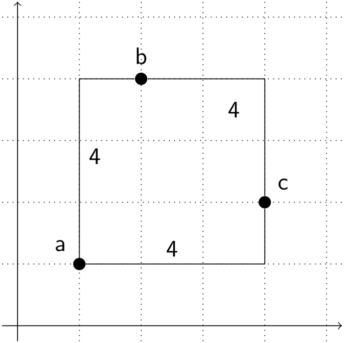
Runtime to calculate distance: $O(n)$

Definition

A *metric space* is a set $\mathcal{X}$ equiped with a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:

$$\begin{array}{ll} d(x,y) \geq 0 & d(x,y) = 0 \text{ iff } x = y \\ d(x,y) = d(y,x) & d(x,z) \leq d(x,y) + d(y,z) \end{array}$$

Metric spaces generalize euclidean space



L_1 (Manhattan, taxicab) distance:

$$\mathcal{X} = \mathbb{R}^n$$

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

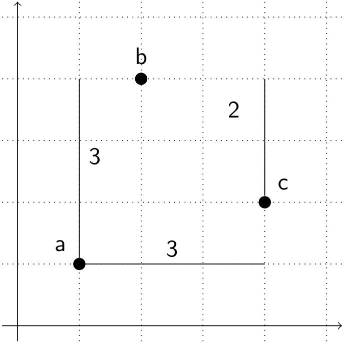
Runtime to calculate distance: $O(n)$

Definition

A *metric space* is a set $\mathcal{X}$ equiped with a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:

$$d(x, y) \geq 0 \qquad d(x, y) = 0 \text{ iff } x = y$$
$$d(x, y) = d(y, x) \qquad d(x, z) \leq d(x, y) + d(y, z)$$

Metric spaces generalize euclidean space



L_∞ (sup) distance:

$$\mathcal{X} = \mathbb{R}^n$$
$$d(x, y) = \sup_{i \in \{1..n\}} |x_i - y_i|$$

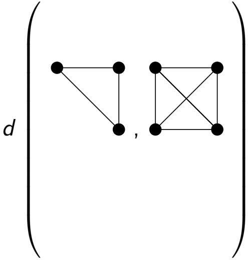
Runtime to calculate distance: $O(n)$

Definition

A *metric space* is a set $\mathcal{X}$ equiped with a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:

$$\begin{array}{ll} d(x, y) \geq 0 & d(x, y) = 0 \text{ iff } x = y \\ d(x, y) = d(y, x) & d(x, z) \leq d(x, y) + d(y, z) \end{array}$$

Metric spaces generalize euclidean space



graph metrics are distances between graphs

$$\mathcal{X} = \text{set of all graphs}$$
$$d(x, y) = \text{many possibilities}$$

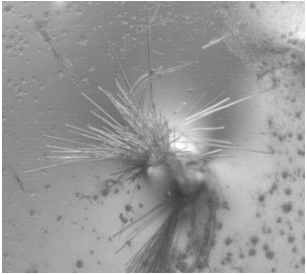
Runtime to calculate distance: varies

Definition

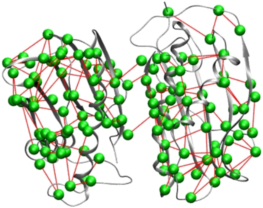
A *metric space* is a set $\mathcal{X}$ equiped with a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:

$$\begin{array}{ll} d(x,y) \geq 0 & d(x,y) = 0 \text{ iff } x = y \\ d(x,y) = d(y,x) & d(x,z) \leq d(x,y) + d(y,z) \end{array}$$

Metrics for proteins



$\rightarrow$



Protein structure (and hence function) can be modeled as a graph

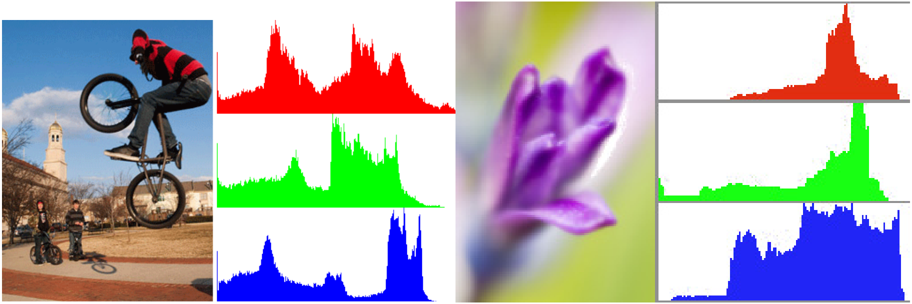
The random walk graph kernel is a commonly used protein metric

This is an *expensive* metric, taking time $O(v^3)$

(Vishwanathan *et. al.*, 2010)

images from: `http://www.lunenfeld.ca`, and `http://vishgraph.mbu.iisc.ernet.in/GraProStr/`

Metrics for images



There are *lots* of distance metrics for images. Histogram metrics follow the two step process:

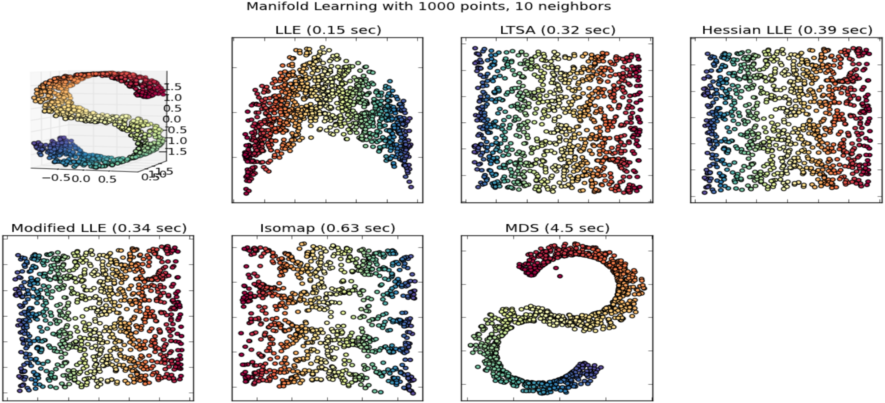
- generate histograms (usually of colors)
- define a distance between histograms

Earth mover's distance is a popular but slow metric. It runs in time $O(b^3 \log b)$, where b is the size of the histogram. (Rubner *et. al.*, 1998)

images from: `http://billmill.org/the_histogram.html`

Metrics can be learned

$$d'_\phi(x,y) = d(\phi(x), \phi(y))$$
$$\phi = \arg\min_\phi \sum_{(x,y)\in\mathcal{X}\times\mathcal{X}} \ell(x,y;\phi)$$



Nearest neighbor data structures for arbitrary metric spaces

	Cover Tree	Nav. Net	Met. Skip List	Ball Tree
Construction space	$O(n)$	$c^{O(1)}n$	$c^{O(1)}n \log n$	$O(n)$
Construction time	$O(c^6 n \log n)$	$c^{O(1)}n \log n$	$c^{O(1)}n \log n$	$O(n^2)$
Insertion time (1 pt)	$O(c^6 \log n)$	$c^{O(1)} \log n$	$c^{O(1)} \log n$	$O(n)$
Query time (1 pt)	$O(c^{12} \log n)$	$c^{O(1)} \log n$	$c^{O(1)} \log n$	$O(n)$
Query time (n pts)	$O(c^{16} n)$	$c^{O(1)}n \log n$	$c^{O(1)}n \log n$	$O(n^2)$
	Beygelzimer *et. al.*, 2006	Krauthgamer and Lee, 2004	Karger and Ruhl, 2002	Omohundro, 1989

The variable c is a measure of dimension (defined on next slide).

Recent research either:

- Extends the analysis on cover trees (Ram *et. al.*, 2010; Curtin *et. al.*, 2013)
- Focuses on *approximate* queries (too many papers to list)

The expansion constant c is a type of dimension

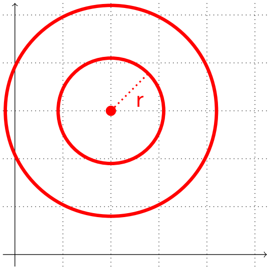
The **expansion constant** is defined as:

$$c = \sup_{p \in X, r \geq 0} \left\{ \frac{|B(p, 2r)|}{|B(p, r)|} \right\}$$

where

$$B(p, r) = \{q \in X : d(p, q) \leq r\}$$

is the ball of radius r centered at point p.



Example 1: In the metric space $\mathbb{R}^2$,

$$c = \frac{\pi(2r)^2}{\pi r^2} = 4$$

Define the **expansion dimension** as $\log_2 c$.
Then the expansion dimension of $\mathbb{R}^n$ is n.

The expansion constant c is a type of dimension

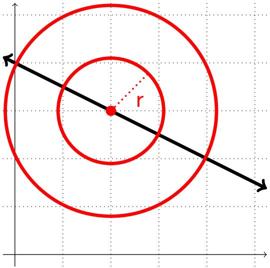
The **expansion constant** is defined as:

$$c = \sup_{p \in X, r \geq 0} \left\{ \frac{|B(p, 2r)|}{|B(p, r)|} \right\}$$

where

$$B(p, r) = \{q \in X : d(p, q) \leq r\}$$

is the ball of radius r centered at point p.



Example 2: In the subspace of $\mathbb{R}^2$ given by

$$\{(x, y) : y = -0.5x + 4\}$$

we have

$$c = \frac{2r}{r} = 2$$

The expansion constant c is a type of dimension

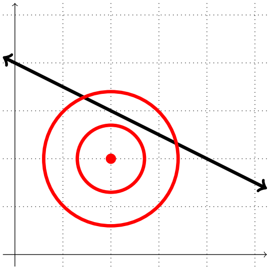
The **expansion constant** is defined as:

$$c = \sup_{p \in X, r \geq 0} \left\{ \frac{|B(p, 2r)|}{|B(p, r)|} \right\}$$

where

$$B(p, r) = \{q \in X : d(p, q) \leq r\}$$

is the ball of radius r centered at point p.

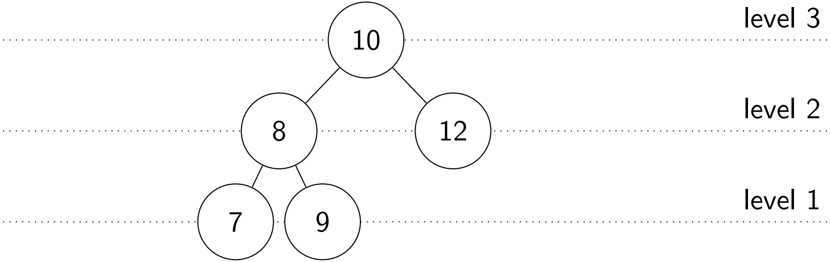


Example 3: In the subspace of $\mathbb{R}^2$ given by

$$\{(x, y) : y = -0.5x + 4\} \cup \{(2, 2)\}$$

we have $c = \infty$

The simplified cover tree



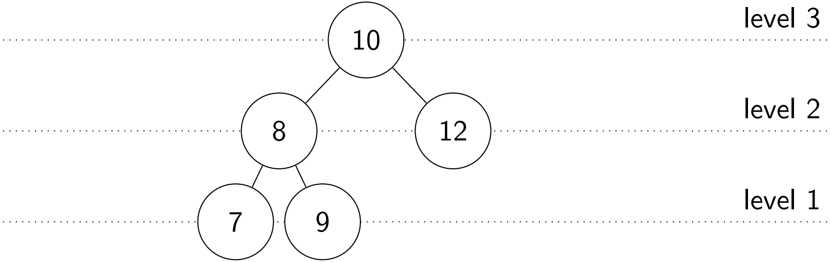
The covering invariant. For every node p, define the function $\texttt{covdist}(p) = 2^{\texttt{level}(p)}$. For each child q of p

$$d(p, q) \leq \texttt{covdist}(p)$$

The separating invariant. For every node p, define the function $\texttt{sepdist}(p) = 2^{\texttt{level}(p)-1}$. For all distinct children q_1 and q_2 of p

$$d(q_1, q_2) \geq \texttt{sepdist}(p)$$

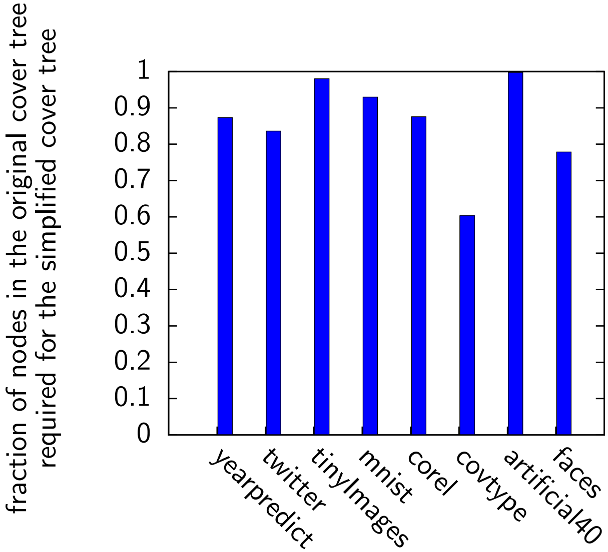
The simplified cover tree



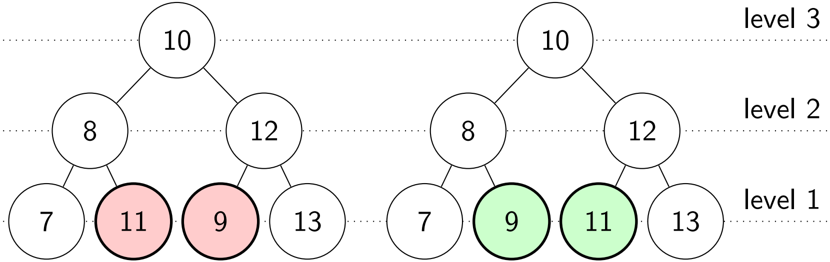
Advantages of the simplified cover tree:

- Maintains all runtime guarantees of the original cover tree.
- Significantly easier to understand and implement.
  The original cover tree was described in terms of an infinitely large tree, only a subset of which actually gets implemented.
- Requires exactly n nodes instead of $O(n)$ nodes.
  Fewer nodes means a faster constant factor for all algorithms.

The simplified cover tree



The nearest ancestor cover tree

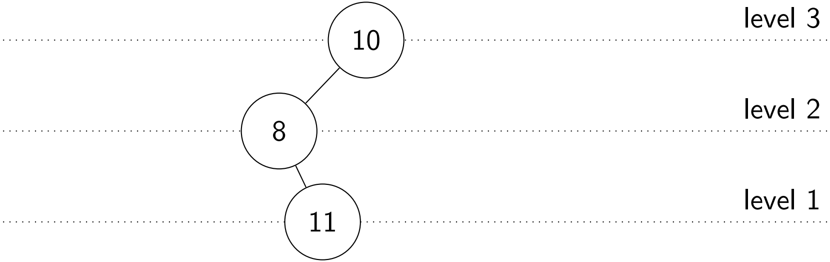


A **nearest ancestor cover tree** is a simplified cover tree where every point p satisfies the additional invariant that if q_1 is an ancestor of p and q_2 is a sibling of q_1, then

$$d(p, q_1) \leq d(p, q_2)$$

Inserting into a nearest ancestor cover tree

Inserting into a nearest ancestor cover tree can require rebalancing.

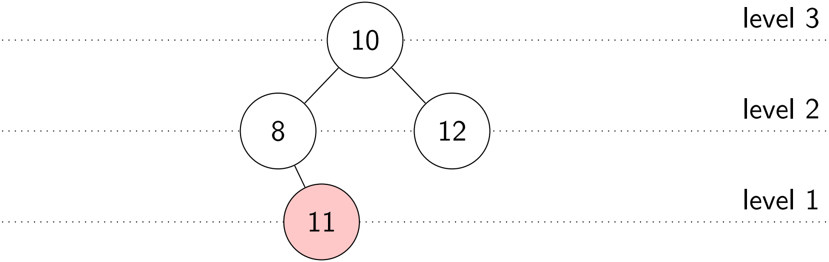


No runtime bounds on the rebalancing step.

In practice, queries are faster but construction is slower.

Inserting into a nearest ancestor cover tree

Inserting into a nearest ancestor cover tree can require rebalancing.

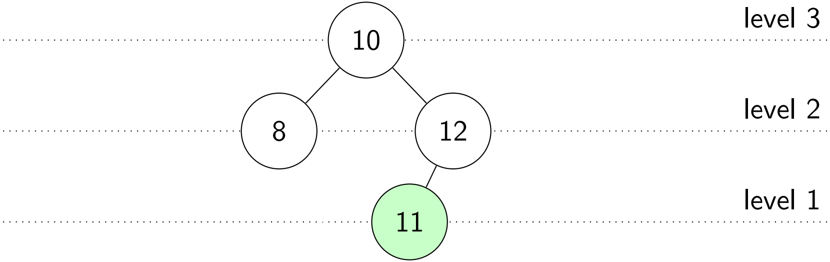


No runtime bounds on the rebalancing step.

In practice, queries are faster but construction is slower.

Inserting into a nearest ancestor cover tree

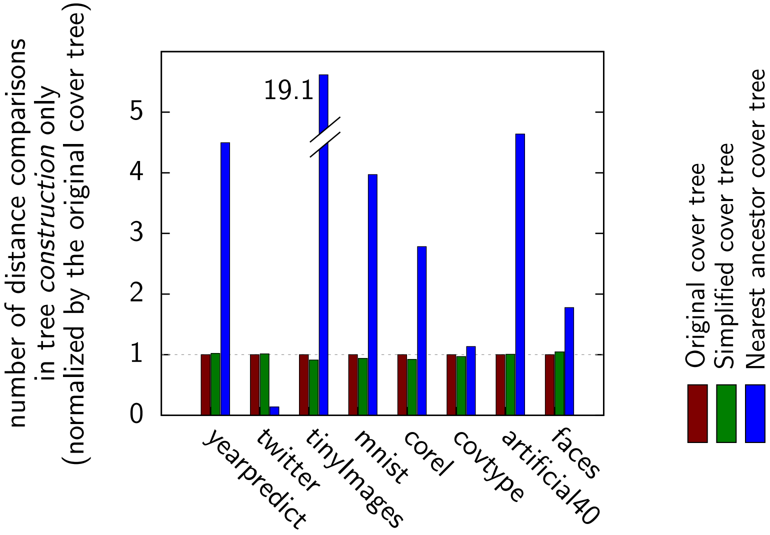
Inserting into a nearest ancestor cover tree can require rebalancing.



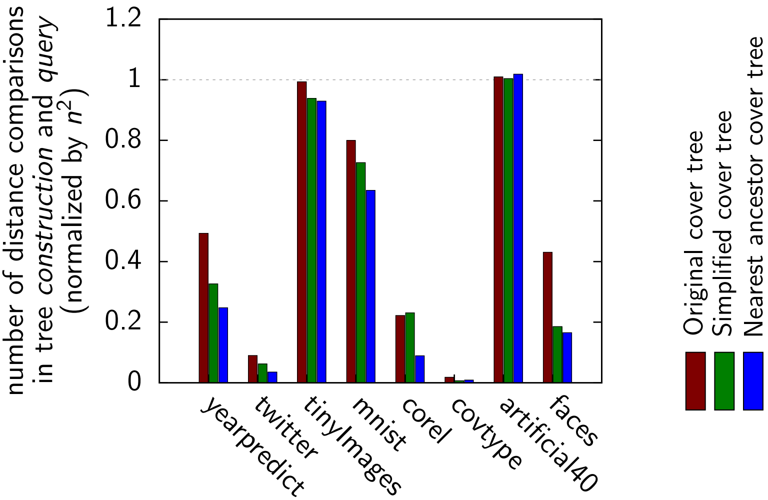
No runtime bounds on the rebalancing step.

In practice, queries are faster but construction is slower.

Comparing cover trees on *construction* time

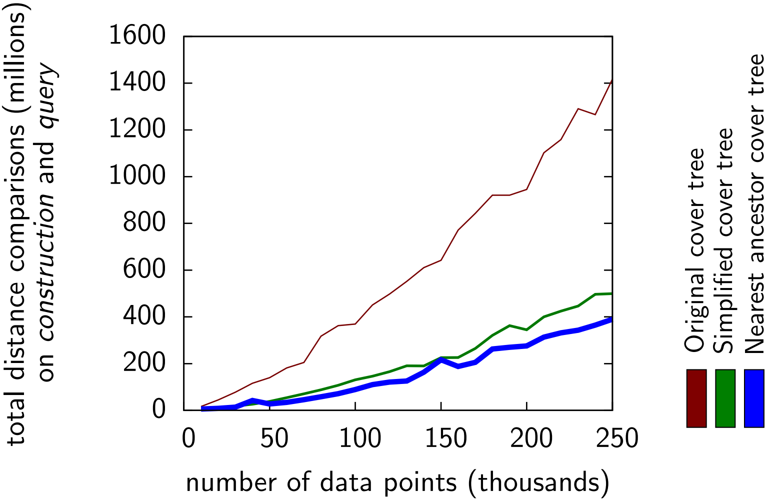


Comparing cover trees on *construction and query* time



All of the cover trees scale similarly

This experiment uses the protein data and the random walk graph kernel.



Cache oblivious cover tree

Need to consider cache accesses for fast, modern data structures

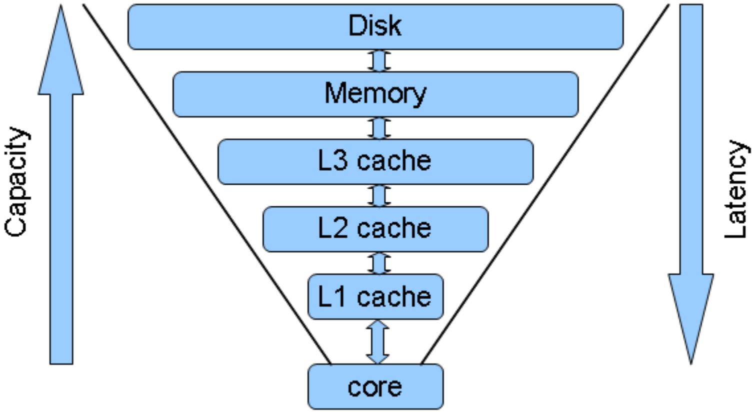


image from: `http://1024cores.net`

Cache oblivious cover tree

Arrange nodes in memory according to a preorder traversal of the tree (van Emde Boas *et al.*, 1966; Demaine, 2002)

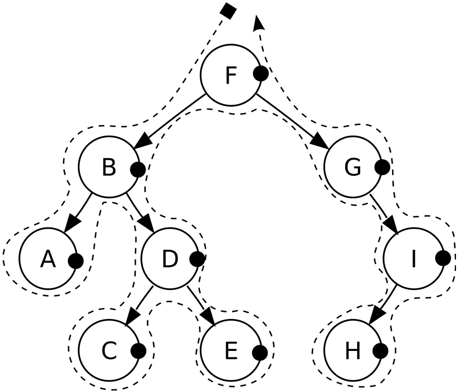
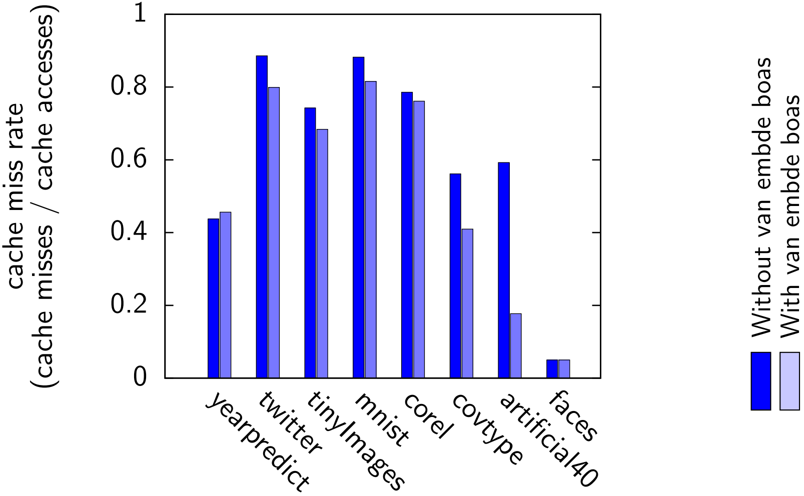


image from: Wikipedia

The cache efficiency of three cover tree implementations

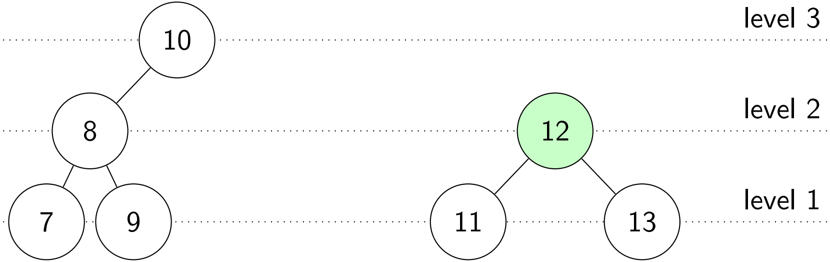


Measured using Linux's perf stat utility on an Amazon AWS instance

Merging cover trees

Merging cover trees gives us a parallel tree construction algorithm

Sometimes, merging cover trees is **easy**:

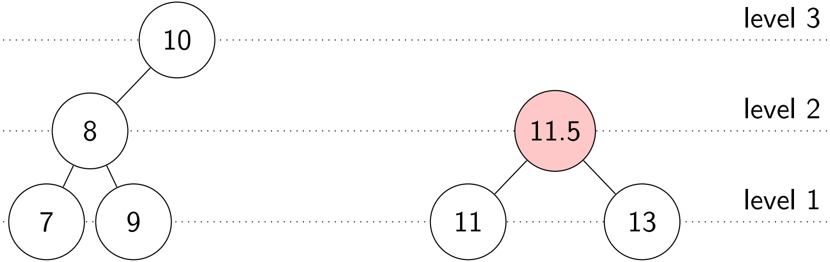


No runtime bound on the merge operation, but it is fast in practice

Merging cover trees

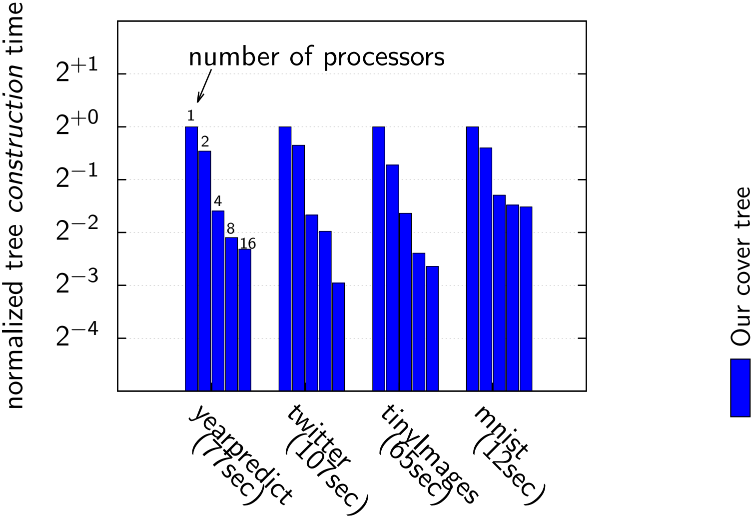
Merging cover trees gives us a parallel tree construction algorithm

Sometimes, merging cover trees is **hard**:



No runtime bound on the merge operation, but it is fast in practice

The effect of parallel tree *construction* on small datasets



Experiments run on an Amazon AWS instance with 16 true cores

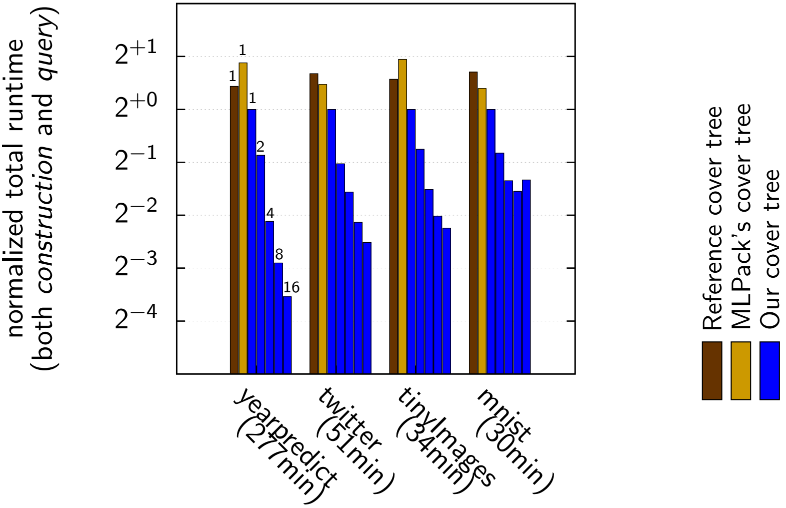
Parallel tree construction really matters on larger data sets

On large datasets with an expensive metric, parallelism is more useful

Yahoo! Flickr dataset with 1.5 million images and earth mover distance

num cores	simplified tree		nearest ancestor tree	
	time	speedup	time	speedup
1	70.7 min	1.0	210.9 min	1.0
2	36.6 min	1.9	94.2 min	2.2
4	18.5 min	3.8	48.5 min	4.3
8	10.2 min	6.9	25.3 min	8.3
16	6.7 min	10.5	12.0 min	17.6

The effect of parallel tree *construction and query*



Experiments run on an Amazon AWS instance with 16 true cores

Summary

You should use cover trees.

We made them easier to implement and faster.

All the code is licensed under the BSD3 and available at:

`http://github.com/mikeizbicki/hlearn`