RWS-Diff: Flexible and Efficient Change Detection in Hierarchical Data

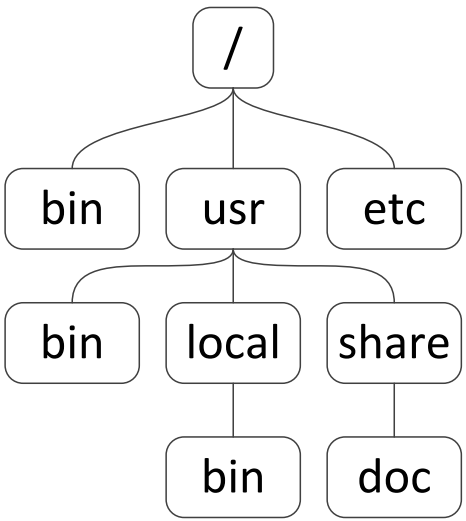
Jan Finis Martin Raiber Nikolaus Augsten
Robert Brunel Alfons Kemper Franz Färber

Technische Universität München

University of Salzburg

SAP AG

Hierarchical Data

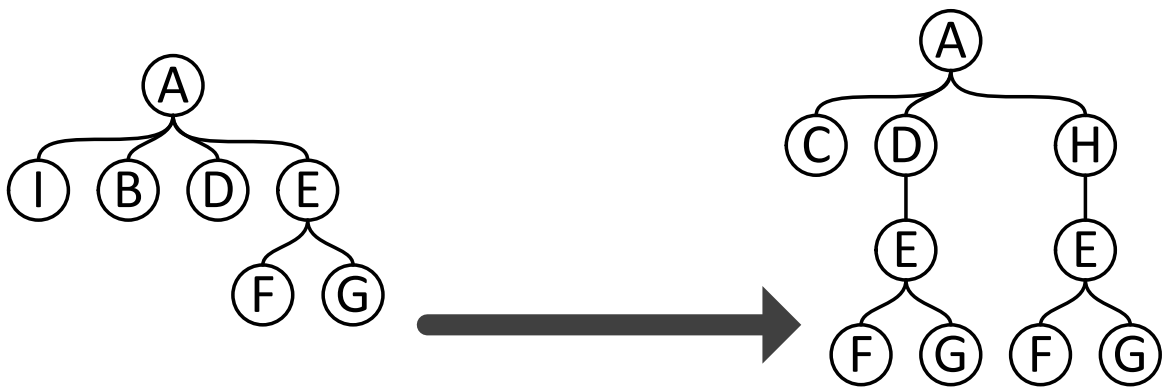


- Hierarchical Data: Rooted, labeled tree
  - XML, HTML
  - File System
  - Abstract Syntax Tree
  - Bills of Materials
  - ...
- Siblings are either ordered (e.g., XML) or unordered (e.g., file systems)
  - Greatly affects the complexity of various algorithms
  - We aim at supporting both

Diff'ing Hierarchical Data

- Task: Given two hierarchies A and B, determine an edit script that transforms A into B
- An edit script is a sequence of edit operations
- Various types of edit operations possible:
  - Leaf insertion/deletion/relocation
  - Subtree deletion/relocation/copy
  - ...
- A cost-minimal edit script is an edit script whose operations have the minimal cost
  - Finding a minimal script is computationally hard

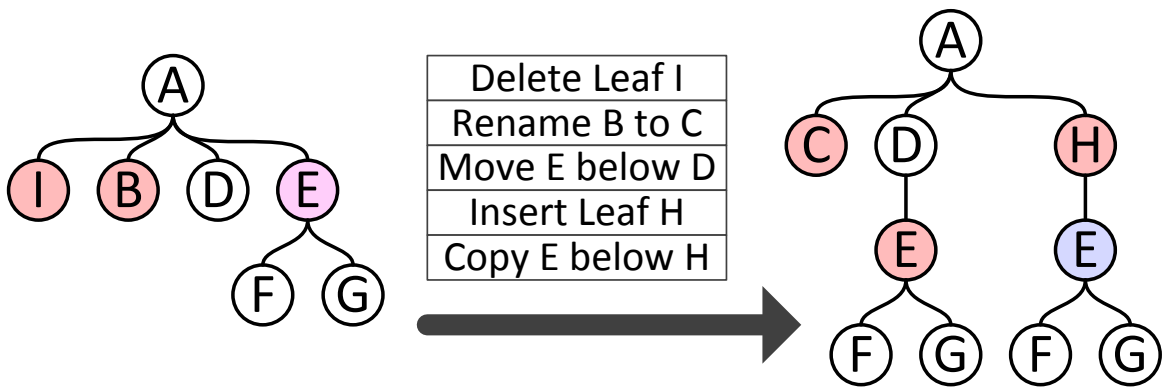
  $\Rightarrow$ Approximations required for scalability



Diff'ing Hierarchical Data

- Task: Given two hierarchies A and B, determine an edit script that transforms A into B
- An edit script is a sequence of edit operations
- Various types of edit operations possible:
  - Leaf insertion/deletion/relocation
  - Subtree deletion/relocation/copy
  - ...
- A cost-minimal edit script is an edit script whose operations have the minimal cost
  - Finding a minimal script is computationally hard

  $\Rightarrow$ Approximations required for scalability



Applications of Hierarchical Diffs

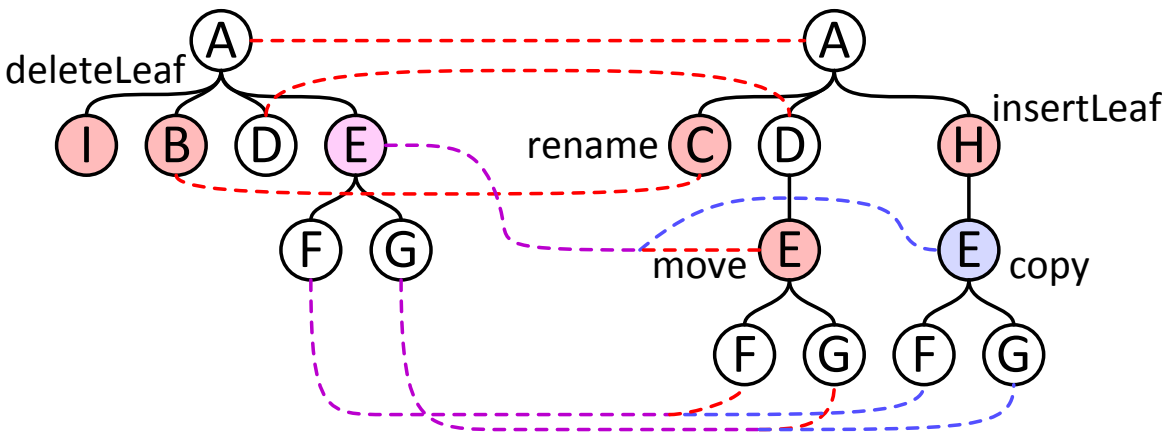
- Version Control
  - XML/HTML Warehousing
  - Source Code Revision Control
- Change visualization
- Synchronization
  - File Systems (cf. delete/insert versus move)
- Tree differencing in general

The Edit Mapping

- Given trees A and B, an edit mapping m is a function $V(B) \mapsto V(A)$ mapping corresponding nodes
- Given an edit mapping, inferring an edit script is simple

$\Rightarrow$ Finding a good edit mapping is the hardest part

- Good mapping maps as many nodes as possible
- Good mapping maps the "right" nodes



$\Rightarrow$ Rest of this talk: How to find a good mapping fast

The Problem

- Finding an exact solution (minimal edit script) is computationally hard
  - Exact approaches do not scale at all
  - Most existing contributions approximate the minimal edit script

The Problem

- Finding an exact solution (minimal edit script) is computationally hard
  - Exact approaches do not scale at all
  - Most existing contributions approximate the minimal edit script
- Finding a good approximation is still hard
  - Elaborated solutions: $O(n^2)$ or worse runtime complexity
  - Simple solutions: $O(n \log n)$ complexity but not robust

  $\Rightarrow$ No robust and scalable solutions exist

The Problem

- Finding an exact solution (minimal edit script) is computationally hard
  - Exact approaches do not scale at all
  - Most existing contributions approximate the minimal edit script
- Finding a good approximation is still hard
  - Elaborated solutions: $O(n^2)$ or worse runtime complexity
  - Simple solutions: $O(n \log n)$ complexity but not robust

  $\Rightarrow$ No robust and scalable solutions exist
- Tailoring the problem definition makes the problem even harder
  - Ordered versus unordered
  - Varying types of edit operations (e.g. no copy, no subtree move,...)

  $\Rightarrow$ All to be supported by our algorithm

Simple Mapping Strategies

Simple Mapping Strategies

- Bottom-up hash mapping
  - Start from the leaves and calculate subtree hashes
  - Map nodes with the same hash

  $\Rightarrow$ Problem: Changing, adding or removing leaf makes mapping fail for all subtrees including the leaf

Simple Mapping Strategies

- Bottom-up hash mapping
  - Start from the leaves and calculate subtree hashes
  - Map nodes with the same hash

  $\Rightarrow$ Problem: Changing, adding or removing leaf makes mapping fail for all subtrees including the leaf
- Top down mapping
  - Start from the root and match as long as nodes are equal

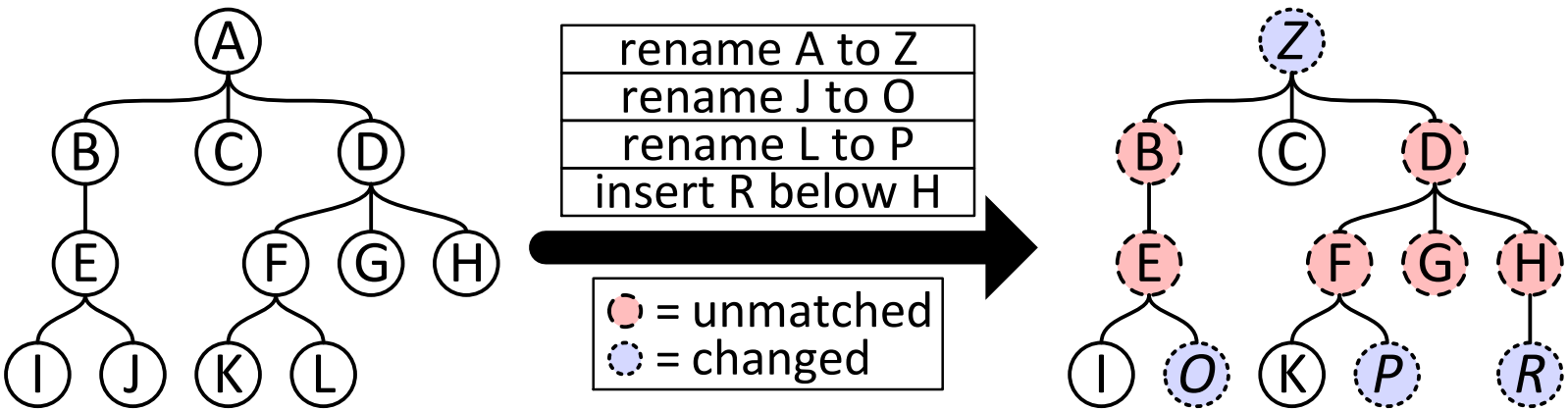
  $\Rightarrow$ Simple renaming of inner nodes prevents matching of all nodes below this node

Simple Mapping Strategies

- Bottom-up hash mapping
  - Start from the leaves and calculate subtree hashes
  - Map nodes with the same hash

  $\Rightarrow$ Problem: Changing, adding or removing leaf makes mapping fail for all subtrees including the leaf
- Top down mapping
  - Start from the root and match as long as nodes are equal

  $\Rightarrow$ Simple renaming of inner nodes prevents matching of all nodes below this node
- Both strategies only work as long as subtrees are equal

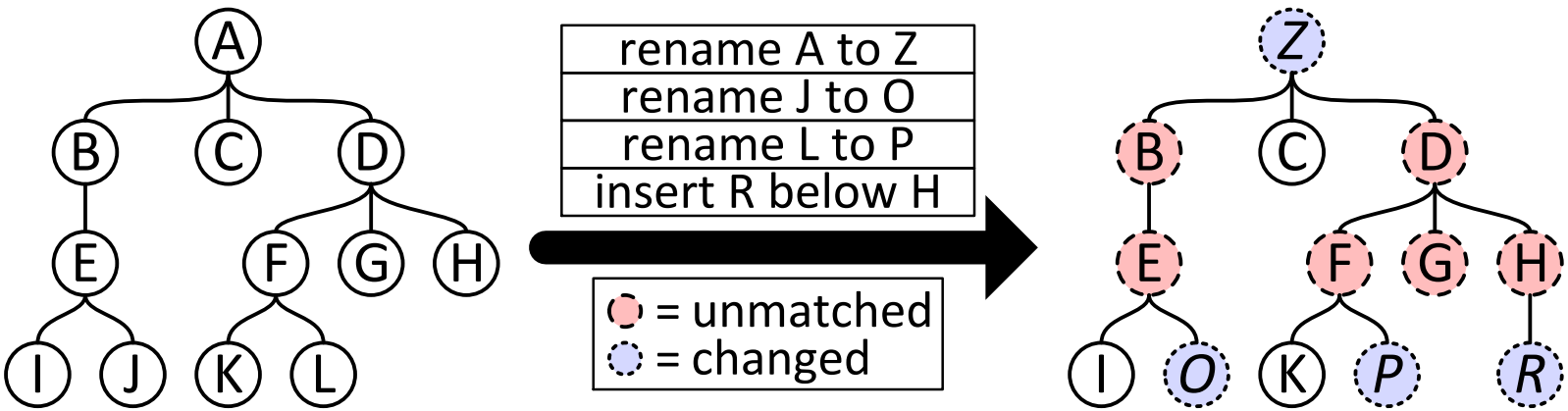


Simple Mapping Strategies

- Bottom-up hash mapping
  - Start from the leaves and calculate subtree hashes
  - Map nodes with the same hash

  $\Rightarrow$ Problem: Changing, adding or removing leaf makes mapping fail for all subtrees including the leaf
- Top down mapping
  - Start from the root and match as long as nodes are equal

  $\Rightarrow$ Simple renaming of inner nodes prevents matching of all nodes below this node
- Both strategies only work as long as subtrees are equal



$\Rightarrow$ Solution: Match subtrees that are similar

Tree Similarity: P,Q-Grams

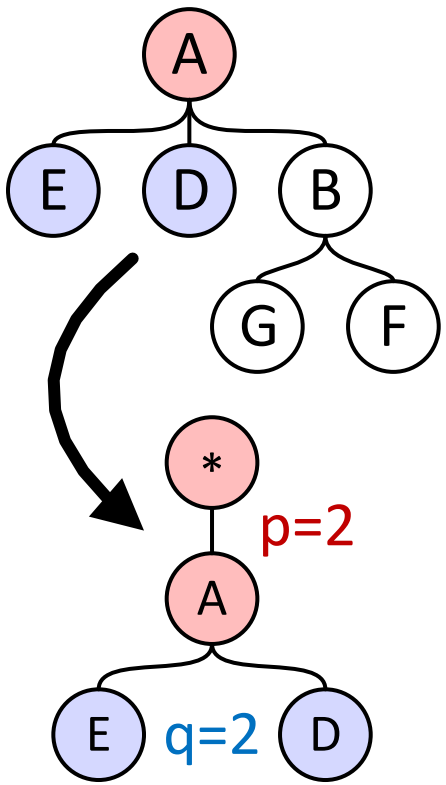
P,Q-Grams are used for computing tree similarity

Tree Similarity: P,Q-Grams

P,Q-Grams are used for computing tree similarity

- "p-grams for trees"
- Idea: Cut trees a into small excerpts $P_a = p_1, p_2, \ldots$(grams)
  - $\Rightarrow$ More grams equal $\Leftrightarrow$ subtrees more similar
    - Symmetric bag distance $D_{\text{bag}}(a, b) = |(P_a \setminus P_b) \cup (P_b \setminus P_a)|$ measures dissimilarity
    - If trees are equal, then $D_{\text{bag}}(a, b) = 0$

Tree Similarity: P,Q-Grams

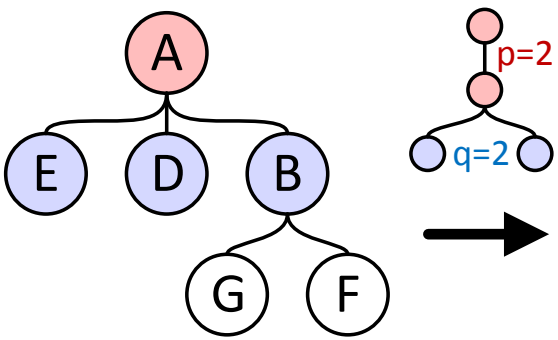


P,Q-Grams are used for computing tree similarity

- "p-grams for trees"
- Idea: Cut trees a into small excerpts $P_a = p_1, p_2, \ldots$(grams)
  - $\Rightarrow$ More grams equal $\Leftrightarrow$ subtrees more similar
    - Symmetric bag distance $D_{\text{bag}}(a, b) = |(P_a \setminus P_b) \cup (P_b \setminus P_a)|$ measures dissimilarity
    - If trees are equal, then $D_{\text{bag}}(a, b) = 0$
- Structure:
  - chain of p ancestors (stem)
  - q leaves (base)
  - Dummy elements (*) for missing ancestors
  - $\Rightarrow$ Capture ancestry and sibling relationsips

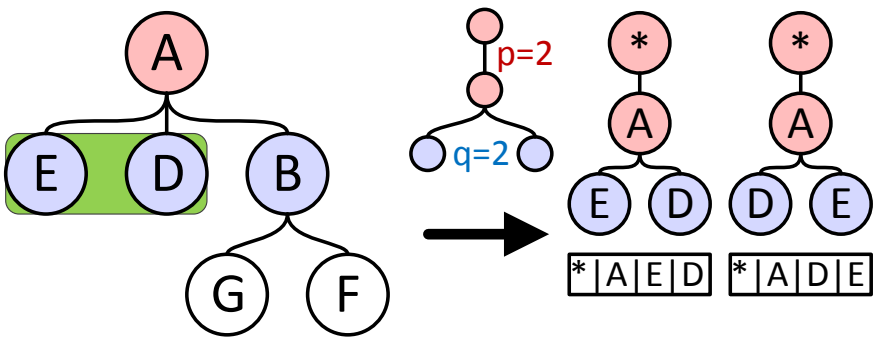
Generating P,Q-Grams

- p,q-grams are generated by sliding a window over the children of a node
- p,q-grams from subtrees can be reused for overall construction time of $O(n)$



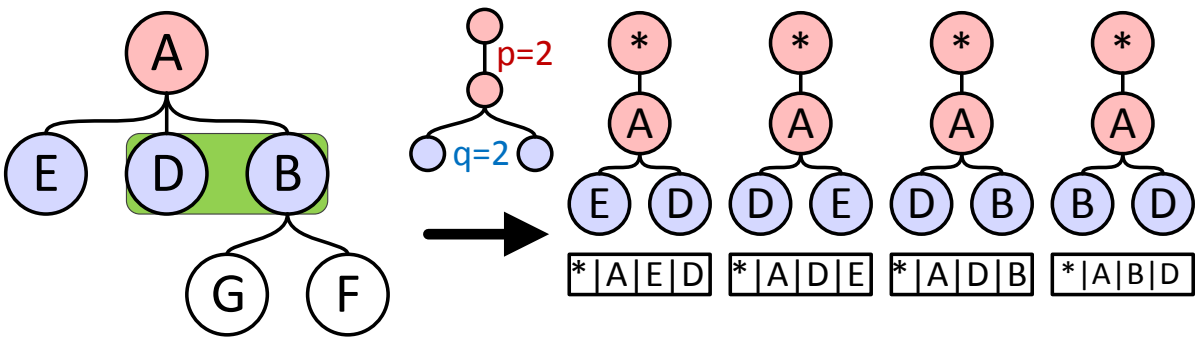
Generating P,Q-Grams

- p,q-grams are generated by sliding a window over the children of a node
- p,q-grams from subtrees can be reused for overall construction time of $O(n)$



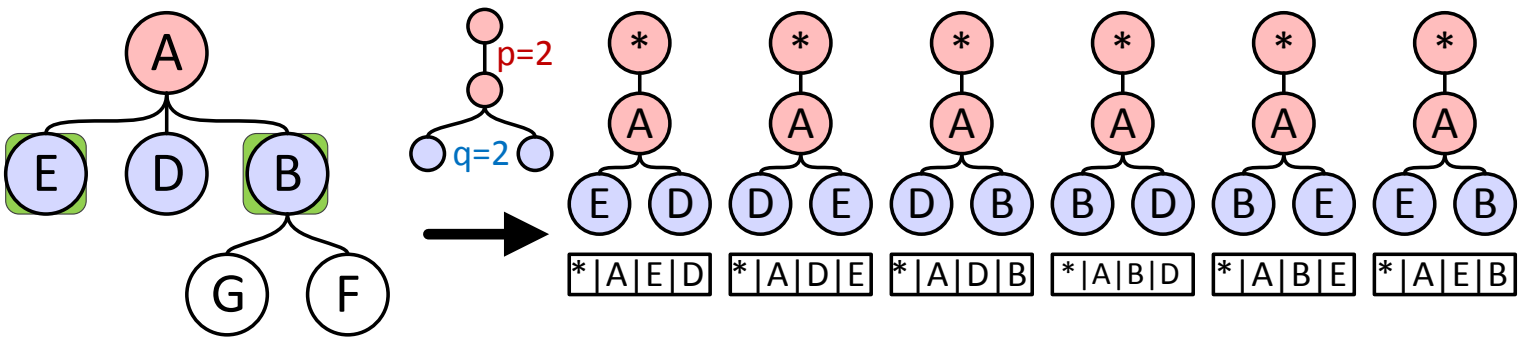
Generating P,Q-Grams

- p,q-grams are generated by sliding a window over the children of a node
- p,q-grams from subtrees can be reused for overall construction time of $O(n)$



Generating P,Q-Grams

- p,q-grams are generated by sliding a window over the children of a node
- p,q-grams from subtrees can be reused for overall construction time of $O(n)$



The Problem with Similarity Mapping

Similarity mapping:

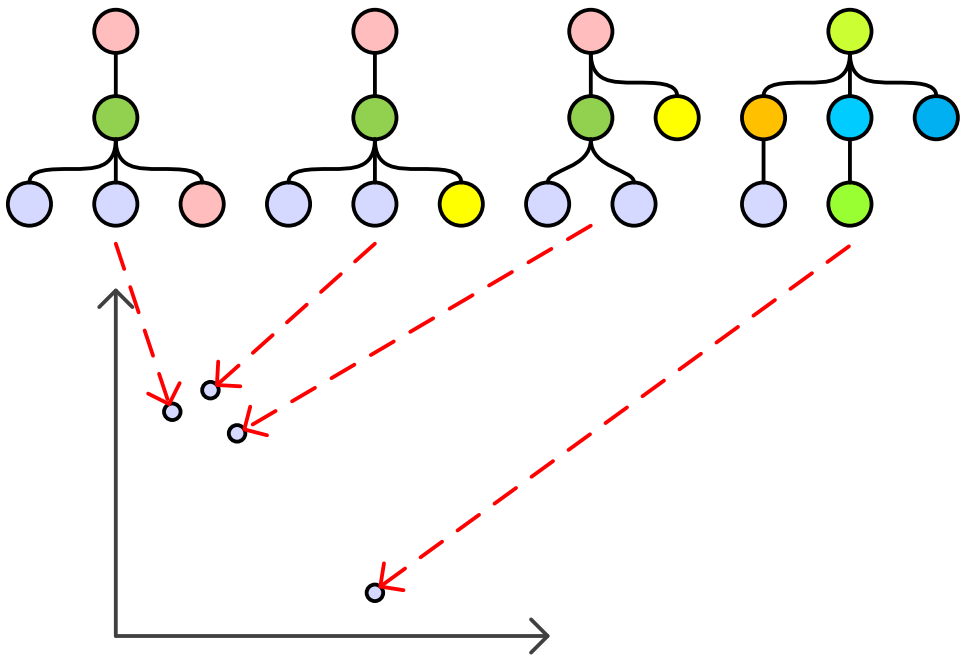
- Use a distance function $D(a, b)$
  - Symmetric bag distance $D_{\text{bag}}(a, b)$ in case of p,q-grams
- For an unmapped subtree a from A, map the subtree b from B with smallest $D(a, b)$
- Problem: Even if computing $D(a, b)$ is fast — say $O(1)$ — we still have to compare all unmapped subtrees in A with all in B
  $\Rightarrow$ $O(n^2)$ complexity ☹
  $\Rightarrow$ Solution: Avoid comparing all pairs! ☺

Feature Vectors

- For each unmatched subtree a, compute a d-dimensional feature vector v_a
- Desired property: $D(a, b) \approx D(v_a, v_b) = ||v_a - v_b||$

$\Rightarrow$ Euclidean distance is approximation of dissimilarity

$\Rightarrow$ Similar subtrees have close feature vectors in euclidean space!



- Choose d meaningfully!
  - Too high $\Rightarrow$ curse of dimensionality
  - Too small $\Rightarrow$ too much loss of information

Matching with Feature Vectors

- For each unmapped subtree a in A, compute v_a

Matching with Feature Vectors

- For each unmapped subtree a in A, compute v_a
- Insert each v_a into an index structure
  - k-D tree, hierarchical k-means clustering, or k-means locality sensitive hashing

Matching with Feature Vectors

- For each unmapped subtree a in A, compute v_a
- Insert each v_a into an index structure
  - k-D tree, hierarchical k-means clustering, or k-means locality sensitive hashing
- For each unmatched subtree b in B find nearest neighbor a in index and match nodes if similar
  - $D(v_a, v_b)$ only approximation

  $\Rightarrow$ False positives possible!

  $\Rightarrow$ Pick k (const) nearest neighbours, choose best or none

Random Walk Similarity

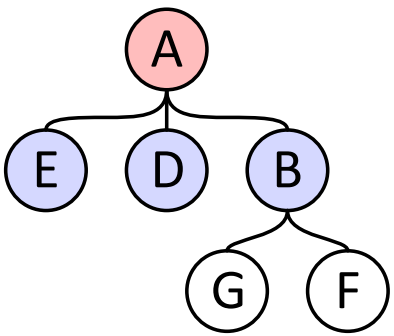
- Open challenge: How to compute feature vector v_a?
  - Computation of all v_as must be in $O(n \log n)$

  $\Rightarrow$ single v_a computation must be in $O(\log n)$
  - Vectors must possess similarity condition: $D(a, b) \approx ||v_a - v_b||$

Random Walk Similarity

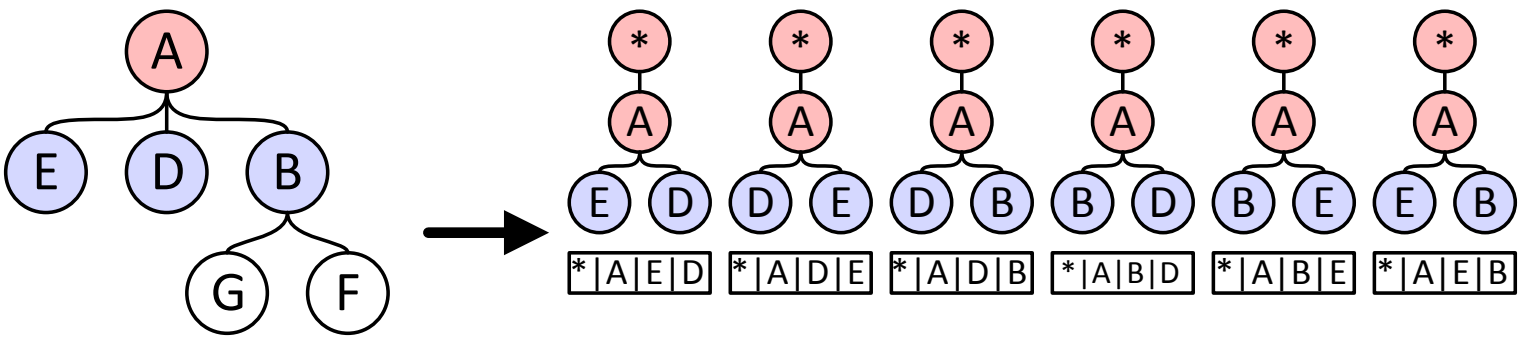
- Open challenge: How to compute feature vector v_a?
  - Computation of all v_as must be in $O(n \log n)$
  - $\Rightarrow$ single v_a computation must be in $O(\log n)$
  - Vectors must possess similarity condition: $D(a, b) \approx ||v_a - v_b||$
- Solution: Random Walk Similarity (RWS)
  - v_a is the endpoint of a special pseudo-random walk in d-dimensional space
  - RWS's properties make it a very good choice for feature vectors

Generating RWS Feature Vectors



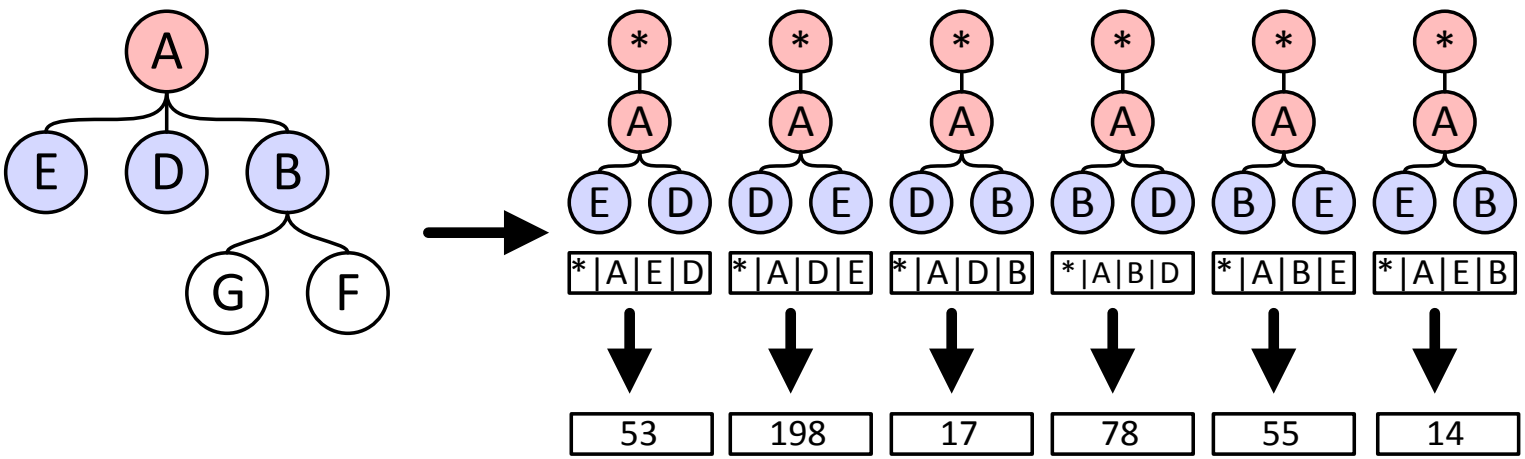
Generating RWS Feature Vectors

1. Generate p,q-grams



Generating RWS Feature Vectors

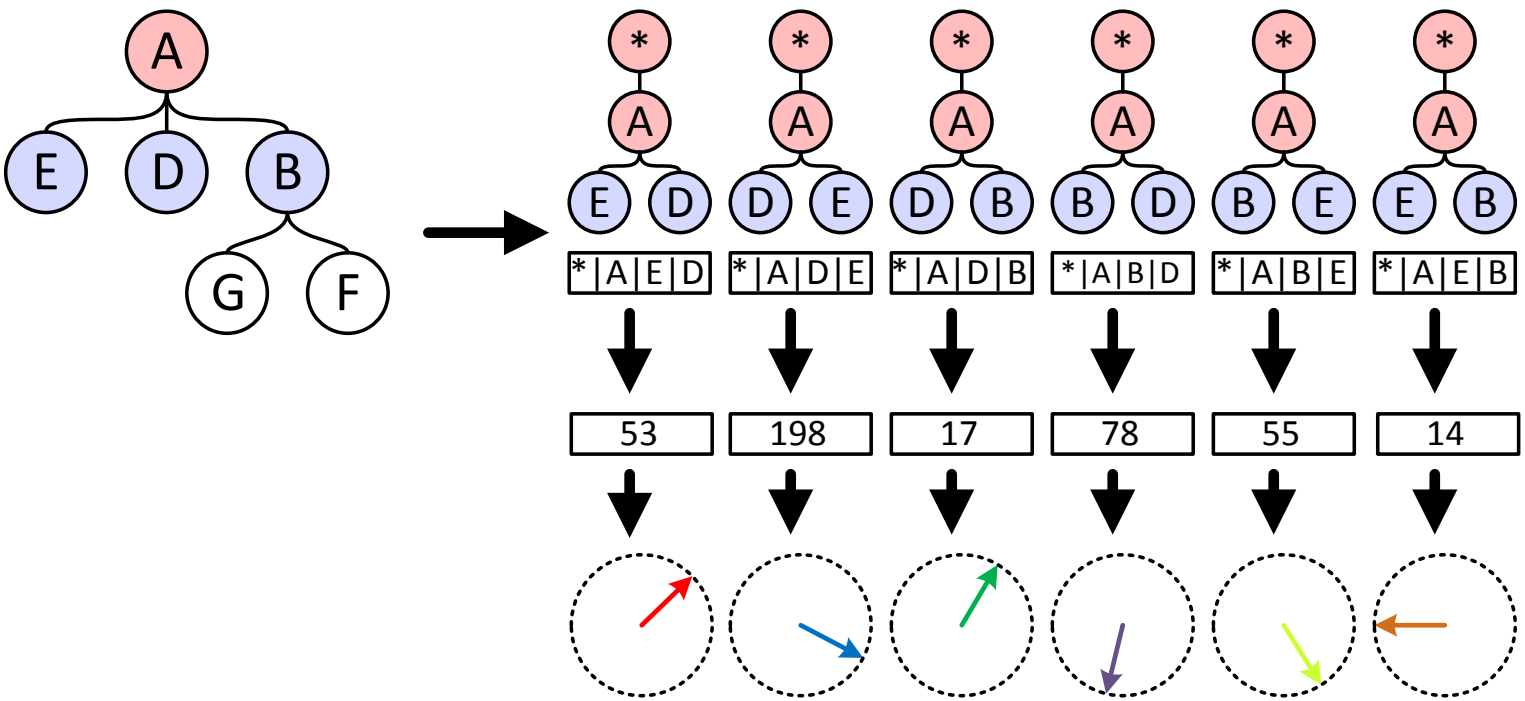
1. Generate p,q-grams
2. Hash each p,q-gram p_i to h_i



Generating RWS Feature Vectors

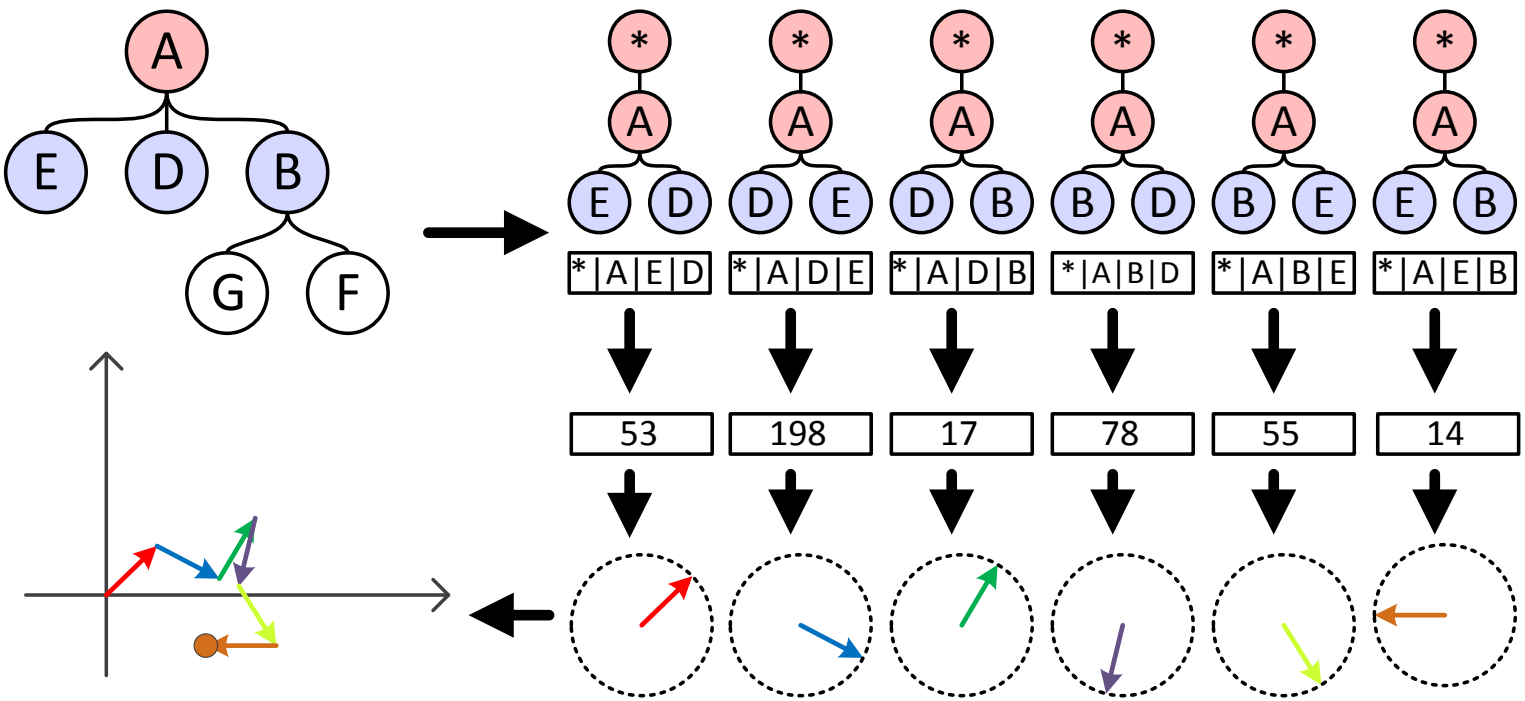
1. Generate p,q-grams
2. Hash each p,q-gram p_i to h_i
3. From each hash h_i, generate a point v_i on the d-dimensional unit sphere

   $\Rightarrow$ v_i is step of length 1 into "random" direction



Generating RWS Feature Vectors

1. Generate p,q-grams
2. Hash each p,q-gram p_i to h_i
3. From each hash h_i, generate a point v_i on the d-dimensional unit sphere
   $\Rightarrow$ v_i is step of length 1 into "random" direction
4. Add up all v_i to get v_a
   $\Rightarrow$ v_a corresponds to a d-dimensional random walk



Random Walk Similarity: Properties

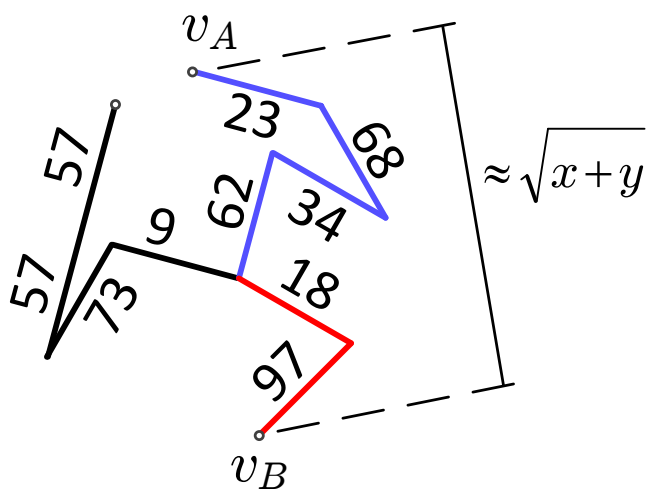
Mathematical Properties of $RWD(a,b) = ||v_a - v_b||^2$:

- $E[RWD(a,b)] = D_{\text{bag}}(a,b) = z$
- $Var[RWD(a,b)] = \frac{2z(z-1)}{d}$
  - $\Rightarrow$ More dimensions $\Rightarrow$ better approximation
  - $\Rightarrow$ More similar points $\Rightarrow$ better approximation
  - $\Rightarrow$ Equal subtrees a and b: $RWD(a,b) = 0$

$\Rightarrow$ RWD is useful approximation for bag distance!

b_A				$x = 4$			
57	57	73	9	62	34	68	23

b_B				$y = 2$	
57	57	73	9	18	97



Approach Overview

Creating an edit script between trees A and B:

Approach Overview

Creating an edit script between trees A and B:

1. Perform simple top down and hash matching
   $\Rightarrow$ Match nodes cheaply if possible

Approach Overview

Creating an edit script between trees A and B:

1. Perform simple top down and hash matching
   $\Rightarrow$ Match nodes cheaply if possible
2. Generate RWS feature vectors for all unmapped subtrees in A and store into index

Approach Overview

Creating an edit script between trees A and B:

1. Perform simple top down and hash matching
   $\Rightarrow$ Match nodes cheaply if possible
2. Generate RWS feature vectors for all unmapped subtrees in A and store into index
3. Probe index for all unmapped subtrees in B, use best candidate

Approach Overview

Creating an edit script between trees A and B:

1. Perform simple top down and hash matching
   $\Rightarrow$ Match nodes cheaply if possible
2. Generate RWS feature vectors for all unmapped subtrees in A and store into index
3. Probe index for all unmapped subtrees in B, use best candidate
4. Generate edit script from mapping

Evaluation

Experiments:

- XML data, randomly altered
- HTML data from news websites, snapshotted every 20 minutes

Baselines:

- XyDiff as best $O(n \log n)$ approach (only simple matching)
- DiffXML as an $O(n^2)$ approach and prominent open source tool

Results:

Evaluation

Experiments:

- XML data, randomly altered
- HTML data from news websites, snapshotted every 20 minutes

Baselines:

- XyDiff as best $O(n \log n)$ approach (only simple matching)
- DiffXML as an $O(n^2)$ approach and prominent open source tool

Results:

- Robustness (1/max number of edit operations) increased by order(s) of magnitude

Evaluation

Experiments:

- XML data, randomly altered
- HTML data from news websites, snapshotted every 20 minutes

Baselines:

- XyDiff as best $O(n \log n)$ approach (only simple matching)
- DiffXML as an $O(n^2)$ approach and prominent open source tool

Results:

- Robustness (1/max number of edit operations) increased by order(s) of magnitude
- Average quality (1/avg number of edit operations) increased by order(s) of magnitude

Evaluation

Experiments:

- XML data, randomly altered
- HTML data from news websites, snapshotted every 20 minutes

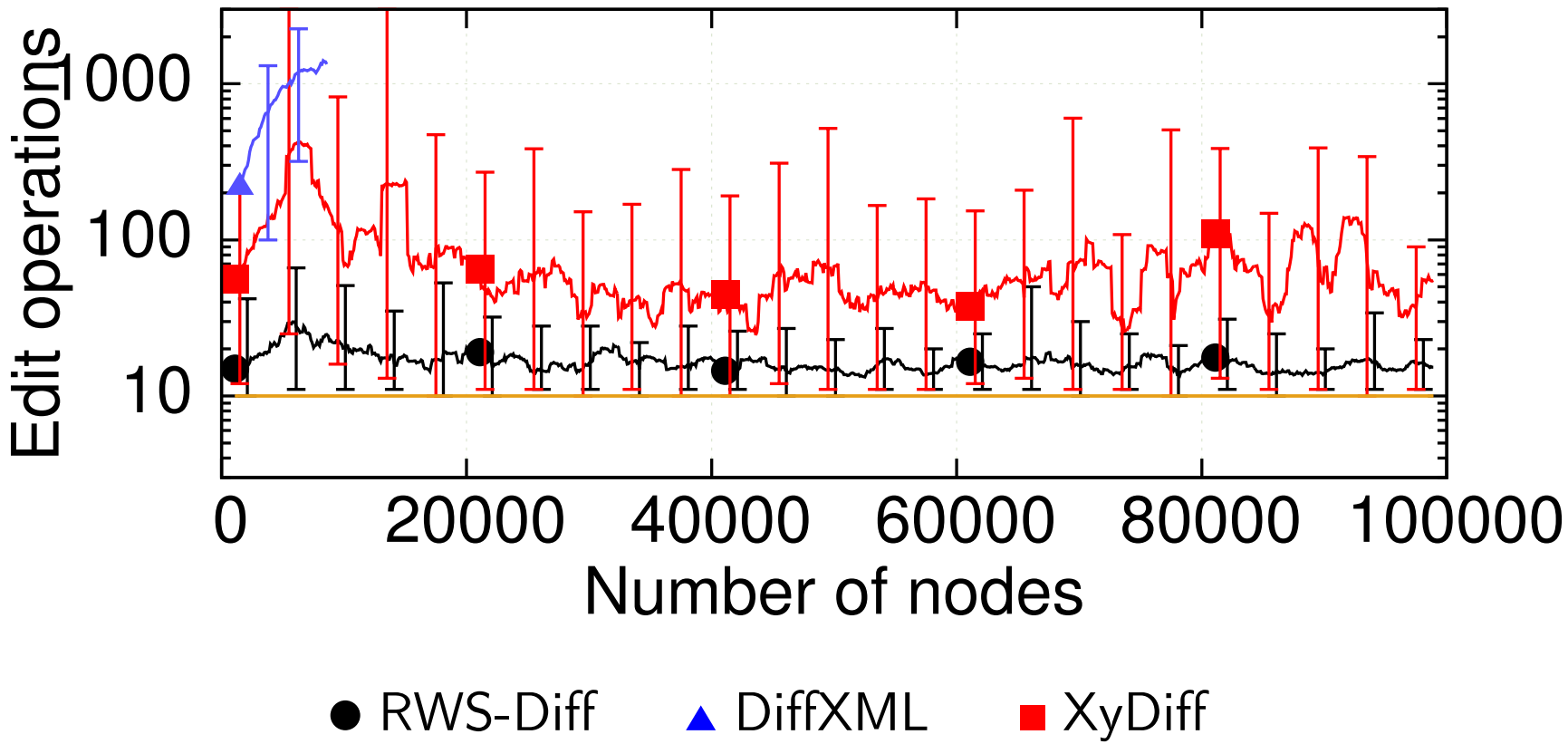
Baselines:

- XyDiff as best $O(n \log n)$ approach (only simple matching)
- DiffXML as an $O(n^2)$ approach and prominent open source tool

Results:

- Robustness (1/max number of edit operations) increased by order(s) of magnitude
- Average quality (1/avg number of edit operations) increased by order(s) of magnitude
- Runtime comparable to simple matching ($\approx$ doubled)

Number of emitted edit operations after performing 10 leaf node changes



Conclusion

- Using similarity for tree differencing increases edit script quality and robustness drastically

Conclusion

- Using similarity for tree differencing increases edit script quality and robustness drastically
- The random walk similarity measure can be used for rapidly finding similar subtrees

Conclusion

- Using similarity for tree differencing increases edit script quality and robustness drastically
- The random walk similarity measure can be used for rapidly finding similar subtrees
- The runtime cost in comparison to simple matchings is bearable

Conclusion

- Using similarity for tree differencing increases edit script quality and robustness drastically
- The random walk similarity measure can be used for rapidly finding similar subtrees
- The runtime cost in comparison to simple matchings is bearable

- Note that random walk similarity is always applicable when the objects to be compared can be decomposed into small excerpts

$\Rightarrow$ approach not limited to trees, various other applications possible!

Thank you for your attention!

Any questions?