Technische Universität Berlin

IN-PLACE UPDATES IN TREE-ENCODED BITMAPS

Marcellus Prama Saputra, Eleni Tzirita Zacharatou, Serafeim Papadias, Volker Markl

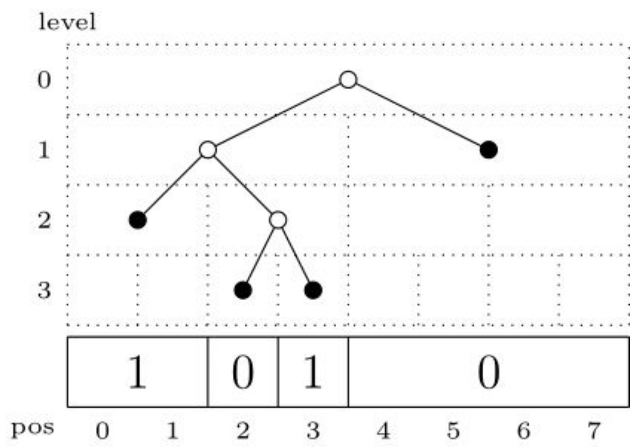
IT-UNIVERSITETET I KØBENHAVN

Agenda

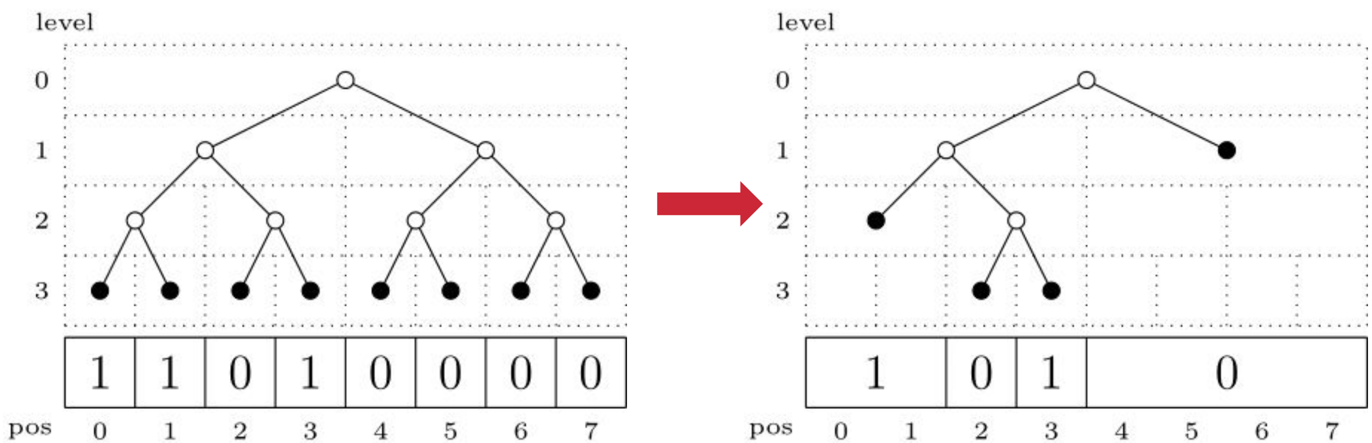
- Tree-Encoded Bitmaps
  - Idea
  - Construction
  - Differential Updates
- Solution Approach
  - Run-Forming Updates
  - Run-Breaking Updates
  - Hybrid Updates
- Experimental Results
- Conclusion

Tree-Encoded Bitmaps

- Bitmap index compression scheme.
- Represent bitmaps as binary trees.
  - Leaf nodes represent runs.
  - A label is assigned to every leaf node to indicate type of run.
  - Length of run is indicated by distance between leaf node and root.



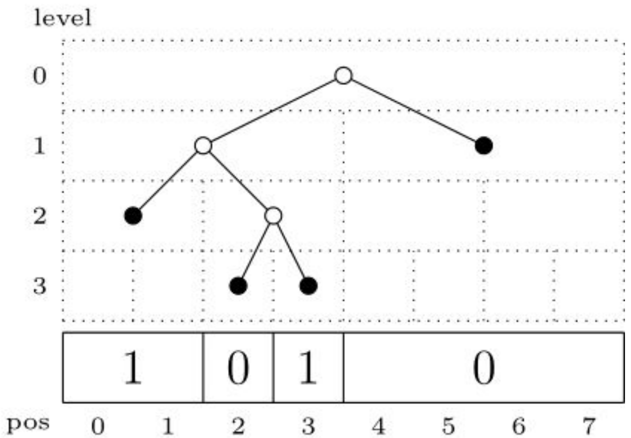
Tree-Encoded Bitmaps: Construction



1. Construct perfect binary tree on top of original bitmap.
2. Prune sibling nodes that have the same label bottom up.

Tree-Encoded Bitmaps: Encoding

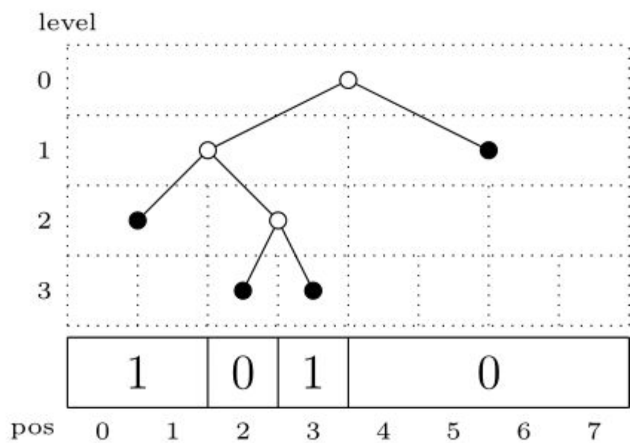
- After pruning, the binary tree is encoded into 2 bitmaps, *T* and *L*.
- The binary tree is traversed left to right in level order, and bits are appended to *T* and *L* during the traversal.
- *T* represents the structure of the tree and *L* contains the labels of every leaf node.
- 0 is appended to *T* when encountering a leaf node, 1 otherwise.



IT-UNIVERSITETET I KØBENHAVN

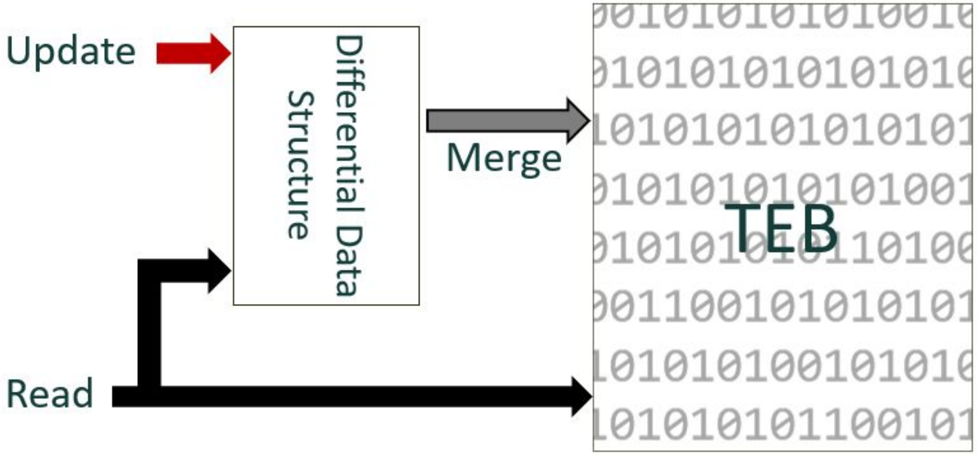
Tree-Encoded Bitmaps: Encoding

- After pruning, the binary tree is encoded into 2 bitmaps, *T* and *L*.
- The binary tree is traversed left to right in level order, and bits are appended to *T* and *L* during the traversal.
- *T* represents the structure of the tree and *L* contains the labels of every leaf node.
- 0 is appended to *T* when encountering a leaf node, 1 otherwise.




```
T=1100100
L=0101
```

Tree-Encoded Bitmaps: Differential Updates



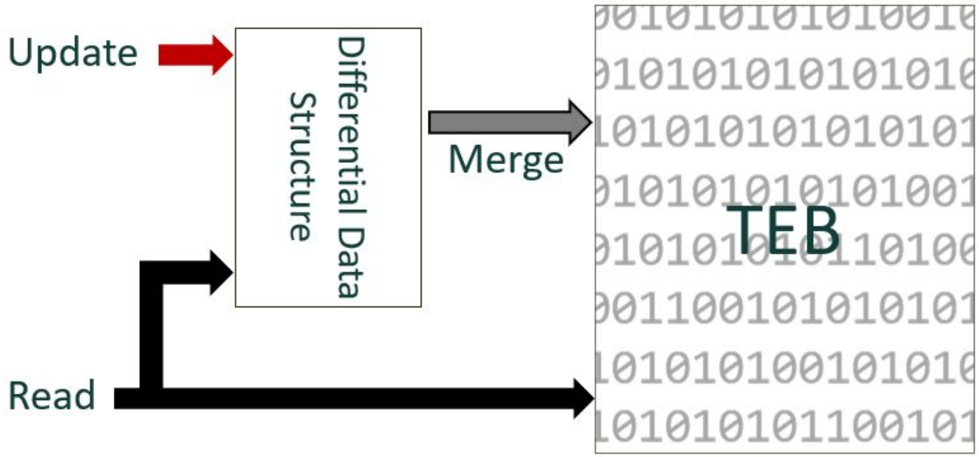
Store updates in a *differential data structure*.

After a number of updates have been stored, the differential data structure can be merged with the TEB by decompressing and reconstructing the TEB.

Tree-Encoded Bitmaps: Differential Updates

Drawbacks:

- Increased space overhead from maintaining an auxiliary data structure.
- Increased read latency as reads must consult the differential data structure as well.

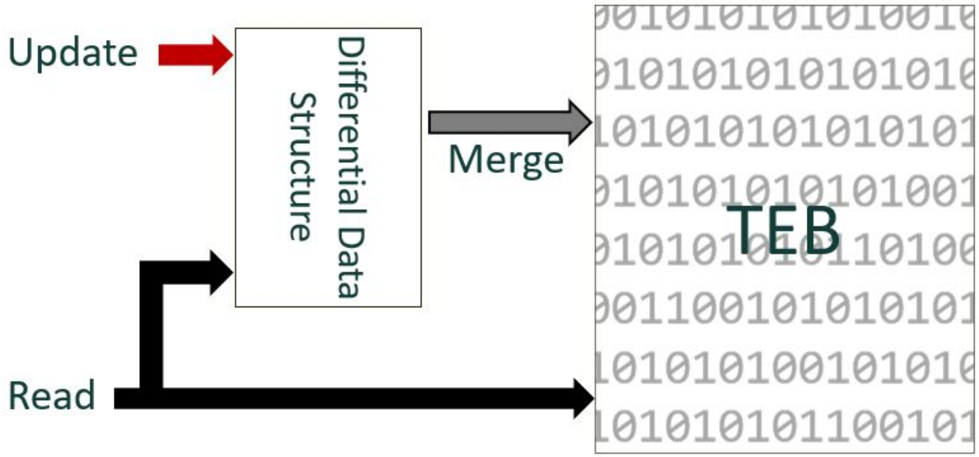


IT-UNIVERSITETET I KØBENHAVN

Tree-Encoded Bitmaps: Differential Updates

Drawbacks:

- Increased space overhead from maintaining an auxiliary data structure.
- Increased read latency as reads must consult the differential data structure as well.



Objective: Reduce space overhead and read overhead.

In-Place Updates

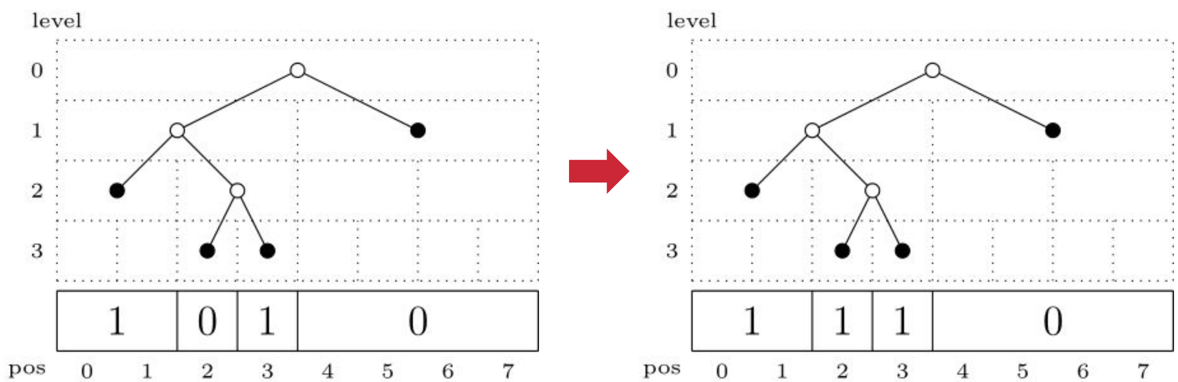
- Directly modifying *T* and *L* instead of storing updates.
- Implementation challenges:
  - Static nature of TEBs.
  - Lack of tools to modify TEBs.
  - Numerous metadata to maintain.

In-Place Updates: Approach

1. Perform point lookup to find leaf node responsible for updated position.
2. Determine type of update depending on the leaf node from point lookup:
   - A run-forming update if the leaf node resides at the lowest possible tree level.
   - A run-breaking update otherwise.
3. Handle the update accordingly.

Run-Forming Updates

- At the lowest possible level of the tree, every leaf node represents an individual bit.
- As a result, performing the update only requires changing the label of one leaf node.
- After the update, a new run may be formed.



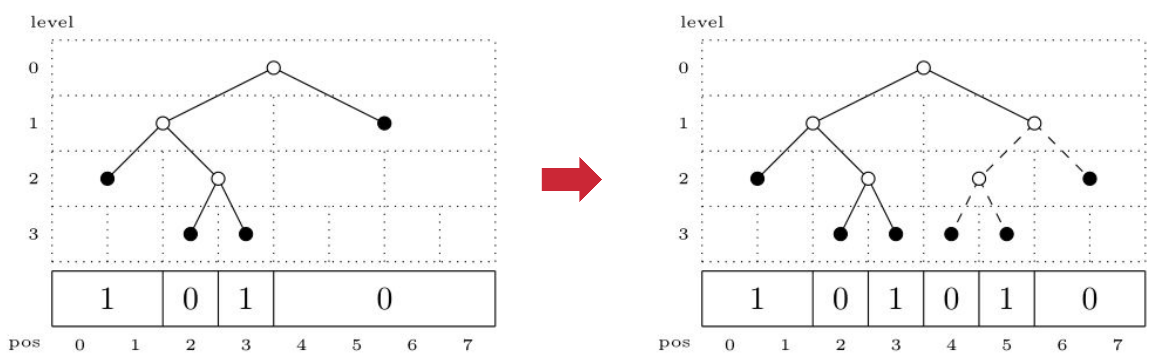
L=0101

L=0111

Technische Universität Berlin

Run-Breaking Updates

- Leaf nodes at upper levels represent runs.
- To apply the update, the leaf node responsible for the updated position is replaced with a subtree.
- This is done by inserting new nodes into the tree, i.e., by inserting bits into *T* and *L*.



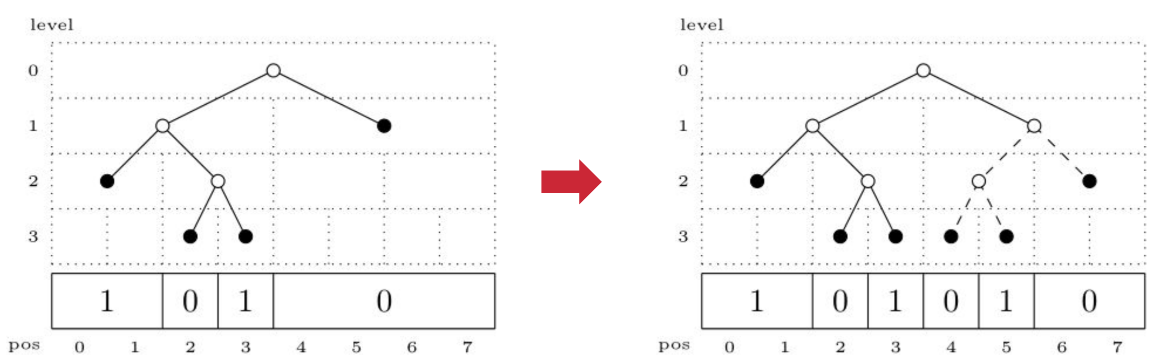
T=1100100
L=0101

T=11101100000
L=_100101

IT-UNIVERSITETET I KØBENHAVN

Run-Breaking Updates

- Problems:
  - Inserting bits into *T* and *L* is expensive.
  - Cannot guarantee extra space required for new bits.

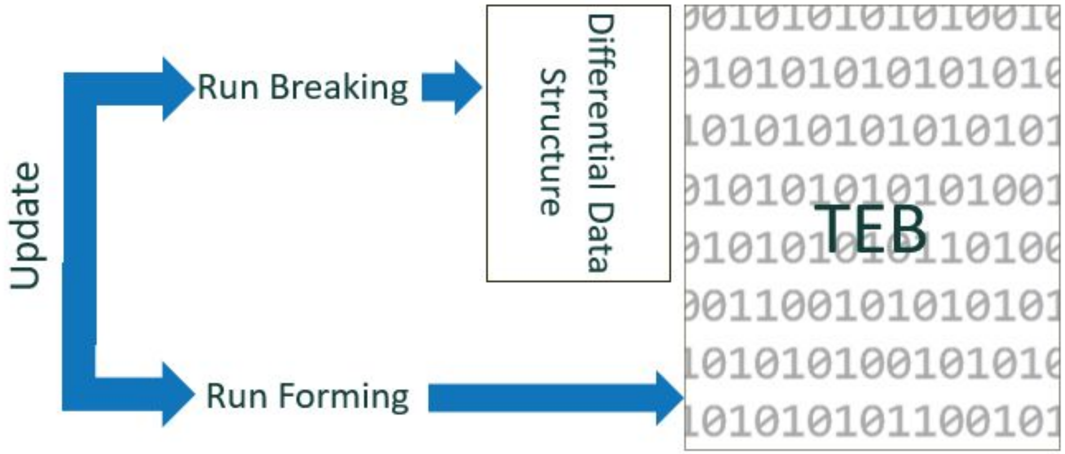


T=1100100
L=0101

T=11101100000
L=_100101

Hybrid Updates

- Run-forming updates are much faster than differential updates.
- Therefore, we devised a hybrid approach:
  - Perform run-forming updates in place.
  - Store run-breaking updates in a differential data structure.

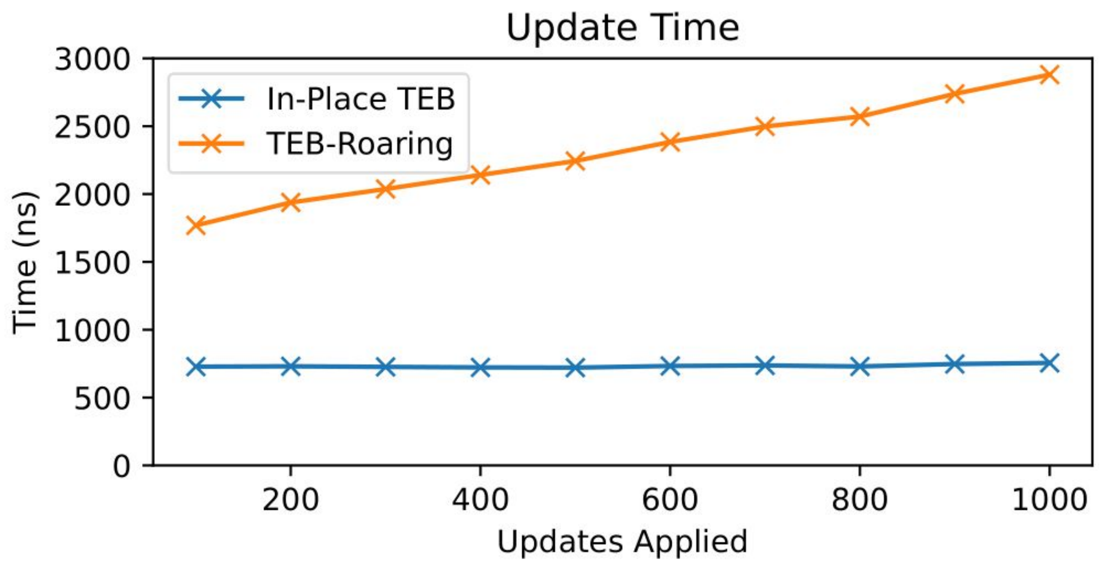


Hybrid Updates: Remarks

- “Best of both worlds” approach; combining in-place updates and differential updates.
- In worst case performs as fast as differential updates, i.e., every update is a differential update.
- Degree of speedup is determined by proportion of run-forming updates in the workload.
- Reduced space overhead from differential data structure as fewer updates are stored.
  - With fewer updates stored, there is less merging in the long run.
- Read latency is the same as with differential updates.

Technische Universität Berlin

Experimental Results: Run-Forming Update Performance



Setup:

- Roaring as differential data structure
- 1 million bits long bitmaps
- Randomly generated bitmaps and updates

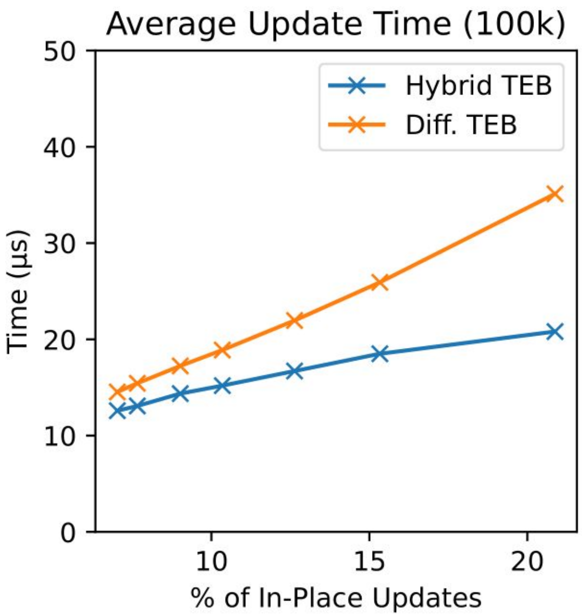
In-place run-forming updates are 3 times faster than differential updates.

As the differential data structure grows, differential updates perform progressively worse.

IT-UNIVERSITETET I KØBENHAVN

Technische Universität Berlin

Experimental Results: Hybrid Update Performance



Setup:

- Roaring as differential data structure
- 1 million bits long bitmaps
- Same bitmap and update generation as in the original paper

Hybrid updates 15% faster than differential updates when 7% of all updates are run forming,

~60% faster than differential updates when 20% of all updates are run forming.

IT-UNIVERSITETET I KØBENHAVN

Conclusion

- Two types of in-place updates:
  - Run-forming updates: change label of target leaf node.
  - Run-breaking updates: expand target leaf node into a subtree.
- Hybrid Approach:
  - Perform run-forming updates in-place, store run-breaking updates.
  - At least as fast as differential updates.
  - Improvement over differential updates increases with more run-forming updates.

References

1. Harald Lang, Alexander Beischl, Viktor Leis, Peter Boncz, Thomas Neumann, and Alfons Kemper. 2020. Tree-Encoded Bitmaps. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 937–967. https://doi.org/10.1145/3318464.3380588