On the Scalability of Computing Triplet and Quartet Distances

Morten Kragelund Holt
Jens Johansen
Gerth Stølting Brodal
Introduction

• Trees are used in many branches of science.
• Phylogenetic trees are especially used in biology and bioinformatics.
• We want to measure how different two such trees are.
Introduction

• Trees are used in many branches of science.
• Phylogenetic trees are especially used in biology and bioinformatics.
• We want to measure how different two such trees are.

Holt, Johansen, Brodal
On the Scalability of Computing Triplet and Quartet Distances
Distances

• Natural in some cases.
• Between trees?
Triplets and Quartets

**Triplets**
- Used in **rooted** trees.
- Sub-trees consisting of **three** leaves.
- \( \binom{n}{3} \) in a tree with \( n \) leaves.
- With 2,000 leaves, 1,331,334,000 triplets.
- Naïve algorithm runs in at least \( \Omega(n^3) \).
- Number of disagreeing triplets.

**Quartets**
- Used in **unrooted** trees.
- Sub-trees consisting of **four** leaves.
- \( \binom{n}{4} \) in a tree with \( n \) leaves.
- With 2,000 leaves, 664,668,499,500 quartets.
- Naïve algorithm runs in at least \( \Omega(n^4) \).
- Number of disagreeing quartets.
Goal

• Comparison of two trees ($T_1$ and $T_2$) with the same set of leaf-labels.
  – Numerical value of the difference of the two trees.
  – Number of different triplets (quartets) in the two input trees.

• A tree has a distance of 0 to itself.
Brodal *et al.* [SODA13]

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolved</td>
<td>Resolved</td>
</tr>
<tr>
<td>$A$: Agree</td>
<td>$C$</td>
</tr>
<tr>
<td>$B$: Disagree</td>
<td>$D$</td>
</tr>
</tbody>
</table>

- For binary trees $C$, $D$ and $E$ are all zero 😊
For binary trees $C$, $D$ and $E$ are all zero 😊
### Brodal et al. [SODA13]

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Arbitrary degree</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Triplets</strong></td>
<td>O(n lg n)</td>
<td>O(n lg n)</td>
</tr>
<tr>
<td></td>
<td>Up to 4d+2 counters in each HDT node</td>
<td>Up to 4d+2 counters in each HDT node</td>
</tr>
<tr>
<td><strong>Quartets</strong></td>
<td>O(n lg n)</td>
<td>O(max(d₁, d₂) n lg n)</td>
</tr>
<tr>
<td></td>
<td>2d² + 79d + 22 counters</td>
<td>2d² + 79d + 22 counters</td>
</tr>
</tbody>
</table>

- A lot of counters ☹️. Is this even feasible?
- Why the $d$ factor on arbitrary degree quartets?
  - $d^2$ counters
Overview

• Basic idea
  – Each triplet (quartet) is anchored somewhere in $T_1$.
  – Run through $T_1$, and for each triplet (quartet), check if they are anchored the same way in $T_2$.

• The algorithm consists of four parts
  1. Coloring
  2. Counting
  3. Hierarchical Decomposition Tree (HDT)
  4. Extraction and contraction
1. Coloring

• Consists of two steps
  1. Leaf-linking \( O(n) \)
  2. Recursive coloring \( O(n \lg n) \)
2. Counting

- Using the coloring of $T_1$ and $T_2$ we count the number of similar triplets (quartets).
- No reason to look at all triplets (would be much too slow) — Instead, look at inner nodes.
- In each inner node, we can keep track of the number of different triplets (quartets), rooted at the given node.
- Using counting and coloring, the triplet distance can be calculated in $O(n^2)$. 
3. Hierarchical Decomposition Tree (HDT)

- **Problem:** $T_2$ is unbalanced.
- **Solution:** Hierarchical Decomposition Trees.

Built in linear time  
Locally balanced  
Triplet distance in $O(n \lg^2 n)$
4. Extraction and Contraction

- Ensuring that the HDT is small, we can cut off that $\lg n$ factor.
- If the HDT is too large, remove the irrelevant parts.

$O(n \lg n)$

Remove $\lg n$ factor
Optimizations

1. [SODA13] hints at constructing HDTs early.
   **Problem:** HDTs take up a lot of memory.
   **Solution:** Postpone HDT construction.
   **Result:** 25-50% reduction in memory usage.
   4-10% reduction in runtime.

2. Utilizing the standard C++ vector data structure.
   **Problem:** Relatively slow (for our needs).
   **Solution:** A purpose-built linked list implementation.
   **Result:** 6-9% reduction in runtime on binary trees.

3. Allocating memory whenever needed.
   **Problem:** (Relatively) slow to allocate memory.
   **Solution:** Allocation in large blocks.
   **Result:** 18-25% improvement in the runtime.
   10-20% increase in memory usage on large input.
Limitations

Two primary limitations in our implementation:

• **Integer representation**
  – \( \binom{n}{3} \) and \( \binom{n}{4} \) are in the order of \( n^3 \) and \( n^4 \).
  – With signed 64-bit integers, quartet distance of only 55,000 leaves.
  – **Solution**: Signed 128-bit integers for \( n^4 \) counters.
    • Quartet distance of up to 2,000,000 leaves.

• **Recursion depth**
  – OS imposed limitation in recursion stack depth.
  – Input, consisting of a very long chain, will fail.
  – Windows: Height \(~4,000\).
  – Linux: Height \(~48,000\).
  – **Solution**: Purpose built stack implementation*.

*Not done in the implementation
Results: [SODA13]

It works, and it is fast!

<table>
<thead>
<tr>
<th>Leaves</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>.29</td>
</tr>
<tr>
<td>10,000</td>
<td>3.90</td>
</tr>
<tr>
<td>100,000</td>
<td>42.60</td>
</tr>
<tr>
<td>1,000,000</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Improvements

• Why \( \min (d_1, d_2) \)?
  
  – \( d \)-counters given by first input tree
  
  – [SODA13]: Calculates 6 out of 9 cases.
  
  – [SODA13]: \( d_1 = 2, d_2 = 1024 \) is much slower than \( d_1 = d_2 = 2 \).

\[
\begin{array}{c|c|c|c}
\text{T_1} & \alpha & \beta & \gamma \\
\hline
\alpha & \times & \times & \times \\
\beta & \times & \times & \times \\
\gamma & \times & \times & \times \\
\end{array}
\]

Add \( 5d^2 + 18d + 7 \) counters

Total \( 7d^2 + 97d + 29 \) counters

Remove need for swapping

\( O(\min(d_1, d_2) \cdot n \lg n) \)
Results: Improved

Faster in all cases

<table>
<thead>
<tr>
<th>Leaves</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>.02</td>
</tr>
<tr>
<td>10,000</td>
<td>.31</td>
</tr>
<tr>
<td>100,000</td>
<td>4.14</td>
</tr>
<tr>
<td>1,000,000</td>
<td>52.05</td>
</tr>
</tbody>
</table>

Holt, Johansen, Brodal
On the Scalability of Computing Triplet and Quartet Distances
More improvements

A+B is a choice
Count A+E instead
Faster?

Holt, Johansen, Brodal
On the Scalability of Computing Triplet and Quartet Distances
More improvements

To count B

• 14 cases
• 92 sums
• $5d^2 + 48d + 8$ counters
• $O(\min(d_1, d_2) n \lg n)$

To Count E

• 5 cases
• 21 sums
• $1d^2 + 12d + 12$ counters
• $O(\min(d_1, d_2) n \lg n)$
Results: More improvements

Fastest in the field 😊

Leaves | Time (s)
--- | ---
1,000 | 0.01
10,000 | 0.21
100,000 | 3.07
1,000,000 | 40.06

Holt, Johansen, Brodal
On the Scalability of Computing Triplet and Quartet Distances
# Overview

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Arbitrary degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[SODA13]: $O(n \lg n)$</td>
<td>[SODA13]: $O(n \lg n)$</td>
</tr>
<tr>
<td></td>
<td>[SODA13]: ~34 seconds</td>
<td>[SODA13]: ~7 seconds</td>
</tr>
<tr>
<td><strong>Triplets</strong></td>
<td>[SODA13]: $O(n \lg n)$</td>
<td>$d_1 = d_2 = 256$</td>
</tr>
<tr>
<td></td>
<td>[SODA13]: ~125 seconds</td>
<td>[SODA13]: ~139 seconds</td>
</tr>
<tr>
<td><strong>Quartets</strong></td>
<td>[SODA13]: $O(n \lg n)$</td>
<td>[SODA13]: $O(\max(d_1, d_2) \ n \lg n)$</td>
</tr>
<tr>
<td></td>
<td>[SODA13]: ~125 seconds</td>
<td>[SODA13]: ~139 seconds</td>
</tr>
</tbody>
</table>

Balanced tree, 630,000 leaves
Conclusion

• [SODA13] is both practical and implementable.
  • We have
    – Performed a thorough study of the alternative choices not studied in [SODA13].
    – Theoretically, and practically, found good choices for the parameters.
    – Shown that [SODA13], and derivatives, successfully scales up to trees with millions of nodes.
  • Open problem
    – Current algorithm makes heavy use of random accesses, and doesn't scale to external memory.
    – Current algorithm is single-threaded.