Similarity Queries over Hierarchical Data

PhD Defense - December 9th 2021

Thomas Hütter

Supervisor: Univ.-Prof. Dipl.-Ing. Nikolaus Augsten, Ph.D.
Co-Supervisor: Dr. Mateusz Pawlik
Introduction
Trees, Tree Edit Distance, JSON
Trees
Trees - Sibling Order

- **Ordered trees**: sibling order matters.

\[ T_1 \quad \neq \quad T_2 \]
Trees in Computer Science

- **XML**: 

```xml
<article mdate="2017-01-11" key="AugstenBG10">
  <author>Nikolaus Augsten</author>
  <author>Michael H. Boehlen</author>
  <author>Johann Gamper</author>
  <title>The pq-gram distance between ordered labeled trees.</title>
  <year>2010</year>
  <volume>35</volume>
  <journal>ACM Trans. Database Syst.</journal>
  <number>1</number>
  <ee>http://doi.acm.org/10.1145/1670243.1670247</ee>
  <url>db/journals/.../AugstenBG10</url>
  <pages>4:1-4:36</pages>
</article>
```
Trees in Biology

- RNA secondary structure\(^2\):
Trees in Linguistics

- Sentimental analysis of movie ratings[3]:

Sean Penn, you owe Nicolas Cage an apology.
Tree Edit Distance (TED)

- **Definition:** the minimum number of edit operations that transform one tree into another.

- **Edit operations:**

![Diagram of a tree with nodes a, b, c, d, and f.](image-url)
Tree Edit Distance (TED)

- **Definition:** the minimum number of edit operations that transform one tree into another.

- **Edit operations:**
Tree Edit Distance (TED)

- **Definition:** the minimum number of edit operations that transform one tree into another.

- **Edit operations:**
Tree Edit Distance (TED)

- **Definition:** the minimum number of edit operations that transform one tree into another.

- **Edit operations:**

  ![Diagram of tree edit distance](image)
Tree Edit Distance (TED)

- **Example:** trees $T_1$ and $T_2$ with TED($T_1$, $T_2$) = 3.

- **Complexity:**
  - $O(n^3)$ time for *ordered* trees\cite{4}.
  - NP-hard for *unordered* trees\cite{5}.
JSON Data Format

- **Definition[^6]:**
  - **Objects**: unordered collection of key-value pairs.
  - **Arrays**: ordered list of values.
  - **Values**: literals (e.g., string), arrays, and objects.

```json
{
    "cast": [
        "Ford",
        "Fisher"
    ],
    "running time": 125,
    "name": "Star Wars - A New Hope"
}
```

[^6]: JSON document 1
Cumulative Thesis

• “Effective filters and linear time verification for tree similarity joins”
  Thomas Hütter, Mateusz Pawlik, Robert Löschinger, and Nikolaus Augsten,
  IEEE 35th International Conference on Data Engineering (ICDE), 2019.

• “JEDI: These aren't the JSON documents you're looking for...”
  Thomas Hütter, Nikolaus Augsten, Christoph M Kirsch, Michael J Carey, and Li Chen,
  Submitted to the International Conference on Management of Data (ACM SIGMOD), 2022.

• “DeSignate: detecting signature characters in gene sequence alignments for taxon diagnoses”
  Thomas Hütter, Maximilian H Ganser, Manuel Kocher, Merima Halkic, Sabine Agatha, and Nikolaus Augsten,

• “A Link is not Enough – Reproducibility of Data”
  Mateusz Pawlik, Thomas Hütter, Daniel Kocher, Willi Mann, and Nikolaus Augsten,
**Definition:**
- Given: distance function $\delta$, user-defined threshold $\tau$, and a query JSON document $d_q$.
- Goal: retrieve all JSON documents $d_i$ from a database $D$ that are similar to $d_q$, i.e., $\delta(d_q,d_i) \leq \tau$.

**Query document $d_q$**

```
{
  "cast": [
    "Ford",
    "Fisher"
  ],
  "running time": 125,
  "name": "Star Wars - A New Hope"
}
```

**Database $D$**

```
{  
  "title": "Star Wars - A New Hope",
  "running time": 125,
  "cast": {
    "Han": "Ford",
    "Leia": "Fisher"
  }
}
```
Distance Functions for JSON

• **Goal:** find a distance function $\delta$ for JSON documents.

{  
  "cast" : [  
    "Ford",  
    "Fisher"  
  ],  
  "running time" : 125,  
  "name" : "Star Wars - A New Hope"  
}

JSON document 1

{  
  "title" : "Star Wars - A New Hope",  
  "running time" : 125,  
  "cast" : {  
    "Han" : "Ford",  
    "Leia" : "Fisher"  
  }  
}

JSON document 2

• **Existing solutions:**
  - Ignore the **structure** (e.g., line-based approaches).
  - No quality **guarantees** (e.g., distance not minimal).
• **Goal:** find a *lossless representation* of JSON documents.

• **JSON trees:**

```json
{
    "cast": [
        "Ford",
        "Fisher"
    ],
    "running time": 125,
    "name": "Star Wars - A New Hope"
}
```

**JSON document 1**
Goal: find a distance function $\delta$ for JSON trees.

Existing solutions:

TED is NP-complete for JSON trees.
• **Observation:** TED is too permissive.
**Intuition:** each subtree of a JSON tree is nested document.
**Definition:** the minimum number of node edit operations (insert, delete, and rename) that transform one tree into the other satisfying the document-preserving constraint.

**Baseline algorithm:** runs in $O(n^2 d \log(d))$ time and $O(n^2)$ space where $n$ is the tree size and $d$ the maximum degree of a tree\[11,12\].

---

**JSON tree 1**

- `"cast"`
- `"r.time"`
- `"name"
  - `125`
  - `"Star..."

**JSON tree 2**

- `"title"
  - `"Star..."
- `"r.time"`
- `"cast"
  - `"Han"
  - `"Leia"

---

JEDI = 5
JSON Similarity Lookup

- **Definition:**
  - Given: distance function $\text{JEDI}$, user-defined threshold $\tau$, and a query JSON tree $t_q$.
  - Goal: retrieve all JSON trees $t_i$ from a database $D$ that are similar to $t_q$, i.e., $\text{JEDI}(t_q, t_i) \leq \tau$.

- **Example:** let $\tau = 5$.

Query tree $t_q$  

Database $D$
Challenges

- **Processing large JSON trees:**
  - JEDI verification takes $O(n^2 d \log(d))$ time.

- **Processing large databases:**
  - $|D|$ tree pairs have to be considered.

```
\begin{figure}
\centering
\begin{tikzpicture}
  \node [draw] (query) {Query tree $t_q$};
  \node [draw] (database) at (query -| 3,0) {Database $D$};
  \node [draw] (query_1) at (query -| 1,0) {"cast"};
  \node [draw] (query_2) at (query -| 1,-1) {"r.time"};
  \node [draw] (query_3) at (query -| 1,-2) {"name"};
  \node [draw] (query_4) at (query -| 2,-1) {"Star..."};
  \node [draw] (query_5) at (query_4 -| 1,0) {"Ford"};
  \node [draw] (query_6) at (query_4 -| 1,-1) {"Fisher"};

  \node [draw] (database_1) at (database -| 1,0) {"cast"};
  \node [draw] (database_2) at (database -| 1,-1) {"r.time"};
  \node [draw] (database_3) at (database -| 1,-2) {"name"};
  \node [draw] (database_4) at (database -| 2,-1) {"Star..."};
  \node [draw] (database_5) at (database_4 -| 1,0) {"Ford"};
  \node [draw] (database_6) at (database_4 -| 1,-1) {"Fisher"};

  \draw [->] (query) -- (query_1);
  \draw [->] (query) -- (query_2);
  \draw [->] (query) -- (query_3);
  \draw [->] (query) -- (query_4);
  \draw [->] (query_1) -- (query_5);
  \draw [->] (query_1) -- (query_6);
  \draw [->] (query_2) -- (query_4);
  \draw [->] (query_3) -- (query_4);

  \draw [->] (database) -- (database_1);
  \draw [->] (database) -- (database_2);
  \draw [->] (database) -- (database_3);
  \draw [->] (database) -- (database_4);
  \draw [->] (database_1) -- (database_5);
  \draw [->] (database_1) -- (database_6);
  \draw [->] (database_2) -- (database_4);
  \draw [->] (database_3) -- (database_4);

\end{tikzpicture}
\end{figure}
```
Filter and Verification Framework

- Overview:

Database → Similarity Index → Candidates → Upper Bound Filter

UB \leq \tau

Upper Bound Filter → Verification

UB > \tau

\delta \leq \tau

Verification → Lookup Result
Filter and Verification Framework

- **Efficient verification:** QuickJedi reduces the runtime by up to one order of magnitude.
JEDI Baseline Algorithm

- **Dynamic programming algorithm:**
  - Process all node pairs in a bottom-up manner.
  - Compute the subtree distance for each node pair.

```
JSON tree 1

- "cast"
  - []
  - "Ford"
  - "Fisher"
  - 125
  - "Star..."

JSON tree 2

- "title"
  - "Star..."
  - 125
  - "cast"
    - "Han"
    - "Leia"
    - "Ford"
    - "Fisher"

Dynamic programming algorithm:
- Process all node pairs in a bottom-up manner.
- Compute the subtree distance for each node pair.
JEDI Baseline Algorithm

- **Subtree distance computation:** consider the edit operation with minimum distance.
JEDI Baseline Algorithm

- **Cost of children matching:**
  - $v$ and $w$ are arrays: sequence edit distance (quadratic runtime).
  - Otherwise: minimum-cost bipartite graph matching (cubic runtime).

\[ \text{cost}(c_i, c'_j) = \text{JEDI}(c_i, c'_j) \]
Skipping the Children Matching

- **Idea:** bound the rename costs. Skip if lower bound exceeds upper bound.

- **Upper bound:** insertion and deletion provide an upper bound.

- **Challenge:** identify a lower bound that is
  - efficient (applied for each node pair) and
  - effective (skip many matchings).
**Aggregate Size Lower Bound**

**Key ideas:**
- The *k*-smallest subtrees of the bigger amount of children are deleted.
- The remaining subtrees are bounded by the size difference.

**Efficiency:**
- Maintain an array for constant time computation.

```
c_1  8
|    |
|    |
|    |  

2

5 + 2 + 2 = 9
```
Filter and Verification Framework

- **Process large databases:** JSIM is the first JSON similarity index.

QuickJedi skips the children matching and reduces the runtime by one order of magnitude.
• **Process large trees:** JOFilter is a linear time and space upper bound filter.

We introduced the first JSON similarity index.

QuickJedi skips the children matching and reduces the runtime by one order of magnitude.
• **Goal:** send candidates to the result set without verification.

• **Key idea:** apply the sequence edit distance on ordered keys.

• **Related work:** fastest algorithm runs in quadratic time[^8].

• **Challenge:** filter applied for each candidate.
Key idea 2: for a node in $T_1$ only $2\tau + 1$ nodes of $T_2$ have to be considered.

Results: reduce the complexity to linear time and space.
Filter and Verification Framework

- **JSON Similarity Lookup:**
  - We introduced the first JSON similarity index.
  - We reduced the runtime of the upper bound from quadratic to linear.
  - QuickJedi skips the children matching and reduces the runtime by one order of magnitude.
Scaling to Large Databases

- **Datasets:** Read (30k trees) and Movies (8.7 million trees).

**Result:** a similarity index is needed to process large databases.
Scaling to Large Trees

- **Datasets:** StanDev (up to 18k nodes) and Schema (up to 48k nodes).

**Result:** QuickJedi and JOFilter reduce the runtime by up to two orders of magnitude.
References