Tree Similarity and Code Clones

- Well defined tree similarity problem
  - Tree editing distance $\delta(T_1, T_2)$:
    - $T_1$ and $T_2$ are $\sigma$-similar if $\delta(T_1, T_2) < \sigma$

- Code clones based on $\delta$

```
for (int i = 0; i < n; i++)
    x[i] = 0;
```

Clone pair, if $\delta(T_1, T_2) < \sigma$

```
for (int i = 0; i < n; i++)
    y[i] = "";
```

---

Minimal # of edit operations
(insert/delete a node or change labels)
Not Scalable

A clone detection directly based on the definition would not scale:

- Computing tree editing distance is expensive
  
  "Computing tree editing distance is expensive where \( d_i \) is the minimum of the depth of \( T_i \) and the number of leaves of \( T_i \)

- Need to compare many pairs of subtrees

```
for (int i = 0; i < n; i++)
  x[i] = 0;
```

Quadratic # of Tree Comparison

```
for (int i = 0; i < n; i++)
  x[i] = 0;
```

```
for (int i = 0; i < n; i++)
  x[i] = 0;
```

```
for (int i = 0; i < n; i++)
  x[i] = 0;
```

Minimal # of edit operations

\[
O (|T_1| \times |T_2| \times d_1 \times d_2) \quad (Zhang \ et \ al. \ 1989)
\]
The Idea – Numerical Vectors

- Numerical vectors are much easier to compare than trees
  - Suppose \( \mathbf{v}_1 = \langle x_1, \ldots, x_n \rangle \) and \( \mathbf{v}_2 = \langle y_1, \ldots, y_n \rangle \)
  - Hamming distance (or the \( l_1 \) norm): \( H(\mathbf{v}_1, \mathbf{v}_2) = \sum_{i=1}^{n} |x_i - y_i| \)
  - Euclidean distance (or the \( l_2 \) norm): \( D(\mathbf{v}_1, \mathbf{v}_2) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \)

- Characterize parse trees as numerical vectors
  - Cluster vectors based on numerical distances
  - Code corresponding to vectors within a same cluster are clones
Deckard’s High Level Architecture
Characteristic Vectors for Trees

Definitions

- **q-level atomic patterns**
  - Labeled complete binary trees of height q
  - E.g., 2-level
    - At most $|L|^{2^q-1}$ patterns, where L is the set of labels

- **q-level vector** for a tree or tree-forest T
  - $<b_1, \ldots, b_k>$, where $k \leq |L|^{2^q-1}$
  - Each $b_i$ counts # of i-th pattern in T
An Sample Characteristic Vector

for ( int i = 0; i < n; i++ )
    x[i] = 0;

1-level patterns:
<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
An Sample Characteristic Vector

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;
```

1-level patterns:

```
<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```
Map Trees to Numerical Vectors (1)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:
Map Trees to Numerical Vectors (1)

```plaintext
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```

- Subtree vectors:
Map Trees to Numerical Vectors (1)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- **Subtree vectors:**
Map Trees to Numerical Vectors (1)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```

- **Subtree vectors:**

```
for s
  decl 1,1,0,0,0,0,0,1,0
  ;
  cond_e 2,0,0,0,1,0,0,0
  ;
  incr_e
  ;
  expr_s
  int id = primary_e
  ;
  primary_e > primary_e
  primary_e ++
  array_e 2,0,0,0,1,0,0,0
  ;
  primary_e
  [ primary_e ]
  lit
  id
  id
```
Map Trees to Numerical Vectors (1)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```

- **Subtree vectors:**

![Subtree vectors diagram]

19
Map Trees to Numerical Vectors (1)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;
```

=id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

**Subtree vectors:**

![Diagram showing subtree vectors for a for loop expression.](image)
Map Trees to Numerical Vectors (1)

```plaintext
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:
```
Map Trees to Numerical Vectors (1)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;

/id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s/
```

- Subtree vectors:
Map Trees to Numerical Vectors (2)

- Forest vectors
  - For sequences of program elements
  - Why needed?
    - $S1; S2; S3; S4$
    - $S0; S5; S2; S3; S6$
Map Trees to Numerical Vectors (2)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```

- Forest vectors:
Map Trees to Numerical Vectors (2)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```

- Forest vectors:
Map Trees to Numerical Vectors (2)

```c
for ( int i = 0; i < n; i++ )
    x[i] = 0;

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>
```

- All vectors:

![Diagram of tree with numerical values]
Vector Clustering

- Have millions of generated vectors
- Need to efficiently cluster “close” vectors together
  - Hashing

- Locality-sensitive hashing (LSH)
  - A family of hash functions, s.t., with high probability,
    - Similar vectors hashed to a same hash value
    - Distant vectors hashed to different hash values
LSH-based Clone Detection (1)

- Rely on LSH to construct hash tables for vectors.

\[ v_1, v_i, \ldots, v_n \] → \[
\begin{array}{c}
\text{LSH} \\
\hline
h(v_1) \\
h(v_i) \\
h(\ldots) \\
h(v_n)
\end{array}
\]
LSH-based Clone Detection (2)

- Query LSH for vector clusters

```latex
\v_1, \v_k, \v_j, \ldots
\v_i, \v_n, \v_m, \ldots, \v_y
```

Close neighbors of \( \v_1 \)
LSH-based Clone Detection (2)

- Query LSH for vector clusters

![Diagram of LSH-based clone detection]

- Close neighbors of \( v_i \)
- \( h(v_i) \)
- \( h(v_{1}) \)
- \( h(\ldots) \)
- \( h(v_n) \)
LSH-based Clone Detection (2)

- Query LSH for vector clusters

- Post-process to reduce “false” clones
  - Small clones
  - Overlapping/duplicate clones
Test Programs

<table>
<thead>
<tr>
<th>Application</th>
<th>Language</th>
<th># files</th>
<th># lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel 2.6.16</td>
<td>C</td>
<td>7,988</td>
<td>5,287,090</td>
</tr>
<tr>
<td>JDK 1.4.2</td>
<td>Java</td>
<td>8,534</td>
<td>2,418,767</td>
</tr>
<tr>
<td>GCC 3.3.6</td>
<td>C</td>
<td>7,828</td>
<td>1,246,322</td>
</tr>
<tr>
<td>PostgreSQL 8.1.0</td>
<td>C</td>
<td>725</td>
<td>606,015</td>
</tr>
<tr>
<td>Derby 10.0.2.1</td>
<td>Java</td>
<td>1,418</td>
<td>523,939</td>
</tr>
<tr>
<td>Apache 2.2.0</td>
<td>C</td>
<td>537</td>
<td>269,287</td>
</tr>
</tbody>
</table>
Clone Quality (1)

- Examined 100 clone clusters
  - Randomly selected from JDK 1.4.2
  - Used parameters
    - Similarity: 1.0
    - Minimum clone size in tokens: 50
    - Stride: 4

- Among the 100 clusters
  - 93 are clearly real clones
  - The remaining 7 are difficult to assess
    - But they are structurally similar
Clone Quality (2)

```java
if ......

......
else if (option.equalsIgnoreCase(``basic'')) {
    bBasicTraceOn = true;
} else if (option.equalsIgnoreCase(``net'')) {
    bNetTraceOn = true;
} else if (option.equalsIgnoreCase(``security'')) {
    bSecurityTraceOn = true;
} else ......

------------------------------

if ......

......
else if (opt.equals(``-nohelp'')) {
    nohelp = true;
} else if (opt.equals(``-splitindex'')) {
    splitindex = true;
} else if (opt.equals(``-noindex'')) {
    createindex = false;
} else ......
```

41