I. Implementation

Following the idea of our approach, we have implemented STCD. The source code is available at https://github.com/CSCC5704/SourceCodewithUI.

A. Bigram Similarity

After catching all the tokens and accumulating their frequencies, in the pretreatment of variable names, we introduced bigram similarity. After actual testing, the threshold is set to 0.7 in our project. If the similarity of two variable names is higher than 0.7, then they are considered the same and their frequencies will be compared.

B. Similarity Algorithm

For each category of tokens, to calculate the similarity, we used am algorithm similar to Levenshtein Distance:

$$\text{Sim}(\text{List}_x, \text{List}_y) = 1 - \frac{\sum \text{Diff}(\text{Elem}_{x_i}, \text{Elem}_{y_i})}{\sum (\text{Freq}_{x_i} + \text{Freq}_{y_i})} \quad (1)$$

where the difference of two elements is defined by:

$$\text{Diff}(\text{Elem}_{x_a}, \text{Elem}_{y_a}) = \text{Abs}(\text{Freq}_{x_a} - \text{Freq}_{y_a}) \quad (2)$$

This is a well-defined calculation, because no difference in the lists gives a similarity of 1, while totally different lists give similarity of 0.

Here are two examples showing the resonablity of this algorithm:

$$\begin{aligned} \text{List}_x &= <a,3>, <b,3>, <c,3>, <d,2> \\ \text{List}_y &= <a,3>, <b,3>, <c,3>, <d,5> \\ \text{Sim}(\text{List}_x, \text{List}_y) &= 1 - \frac{3}{11+14} = 0.88 \end{aligned} \quad (3)$$

$$\begin{aligned} \text{List}_x &= <a,3>, <b,1>, <c,3>, <d,1> \\ \text{List}_y &= <a,0>, <b,5>, <c,2>, <d,5> \\ \text{Sim}(\text{List}_x, \text{List}_y) &= 1 - \frac{3+4+1+4}{8+12} = 0.4 \end{aligned} \quad (4)$$

here $<a,3>$ means the frequency for a is 3.

This result matches our intuition: the lists in the first example are more similar than in the second example, and we are likely to take the first example as clone, which has a similarity of 0.88.

C. Machine Learning

Before using machine learning to set the weights, we set the weights arbitrarily, which may not provide the best result, but it works in finding clones. That being said, STCD can also run without training, however the training process will provide a better result.

To get training data, we use manual selection to find “ground truth”. However it is not so easy to manually find “ground truth”: if a Java file has 100 methods, then we need to compare $\frac{100\times 99}{2} = 4950$ pairs. In these pairs, most of them are not clone, so we applied our tool to the Java file, with a low threshold of 0.65 to rule out most of the pairs, and manually checked the rest. This reduced our work from thousands of comparasions to less than a hundred. This process may introduce false negatives but we consider it to be small.

We selected training files from SWT–the tool to develop our UI. The advantage to use data from the same project is that machine learning can learn the developer’s behavior. We chose 10 Java files, namely: *Combo.java, DragSource.java, DropTarget.java, FormData.java, Label.java, Printer.java, Program.java, Shell.java, Text.java, WebKit.java*. The number of methods ranges from 12 to 125.

In the training process, MLP studies 50 clone pairs and 150 non-clone pairs to decide the best weights. There are many more non-clone pairs but studying more of them does not give a better result.

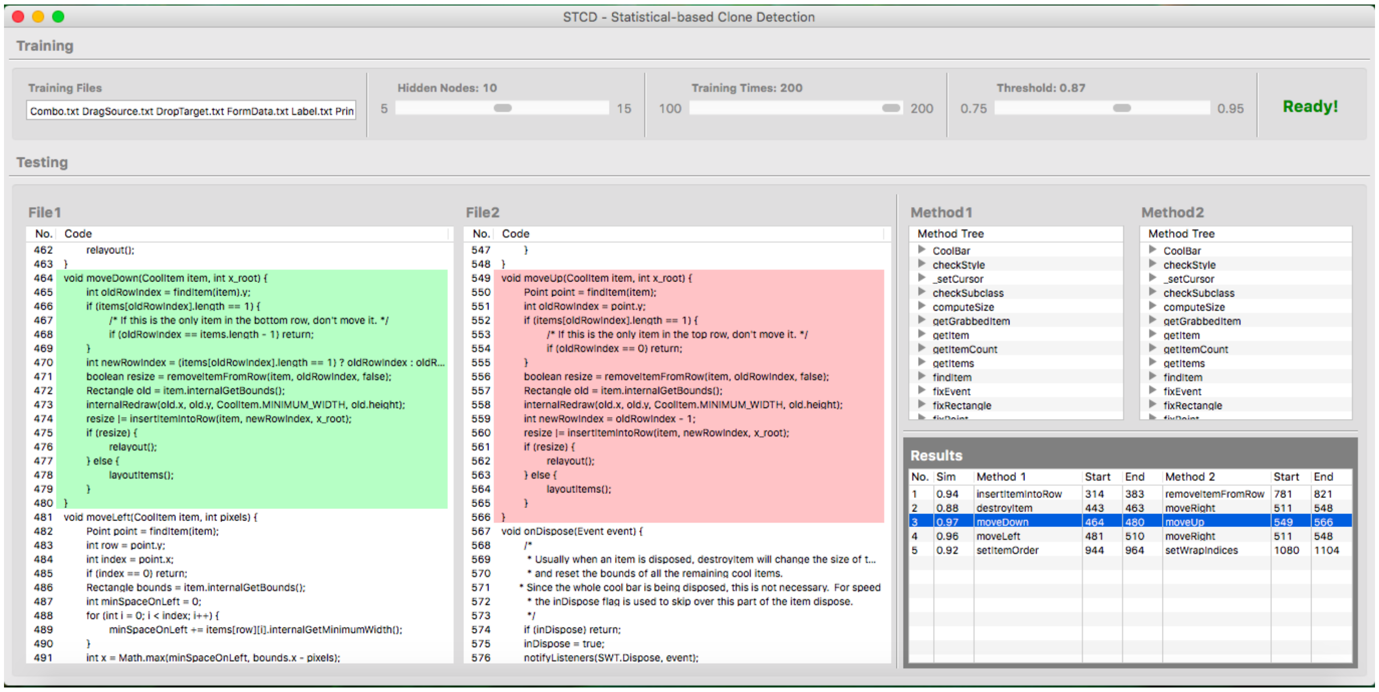
D. Application

To make STCD more user friendly, we choose SWT as a toolkit. SWT is a graphical widget toolkit developed by IBM. It uses native elements when possible and always shows native behaviors. With SWT we developed an application, as shown in Fig:1.

E. Instruction of STCD

The step-by-step instruction to use STCD:

1) From the training menu users are able to select training files. STCD can also run the testing without training, then starts from Step 4.
2) After the selection of training files, users can adjust Hidden Nodes and Training Times in the MLP training process, as well as the threshold.
3) Run the training until it’s ready, that takes less than seconds.
4) From the test menu users can choose the Java file to test. A second file can be selected, if not, the test will be within the same file.
5) Run the test until the result is shown.
6) From the Results window, click on the clone and the clone pairs will be shown in green and red color. The Method 1 and Method 2 windows show the token frequencies in each method.



STCD - Statistical-based Clone Detection

Training

Training Files: Combo.txt DragSource.txt DropTarget.txt FormData.txt Label.txt Prin

Hidden Nodes: 10 (5 – 15)

Training Times: 200 (100 – 200)

Threshold: 0.87 (0.75 – 0.95)

Ready!

Testing

File1

```
No.  Code
462          relayout();
463  }
464  void moveDown(CoolItem item, int x_root) {
465          int oldRowIndex = findItem(item).y;
466          if (items[oldRowIndex].length == 1) {
467                  /* If this is the only item in the bottom row, don't move it. */
468                  if (oldRowIndex == items.length - 1) return;
469          }
470          int newRowIndex = (items[oldRowIndex].length == 1) ? oldRowIndex : oldR...
471          boolean resize = removeItemFromRow(item, oldRowIndex, false);
472          Rectangle old = item.internalGetBounds();
473          internalRedraw(old.x, old.y, CoolItem.MINIMUM_WIDTH, old.height);
474          resize |= insertItemIntoRow(item, newRowIndex, x_root);
475          if (resize) {
476                  relayout();
477          } else {
478                  layoutItems();
479          }
480  }
481  void moveLeft(CoolItem item, int pixels) {
482          Point point = findItem(item);
483          int row = point.y;
484          int index = point.x;
485          if (index == 0) return;
486          Rectangle bounds = item.internalGetBounds();
487          int minSpaceOnLeft = 0;
488          for (int i = 0; i < index; i++) {
489                  minSpaceOnLeft += items[row][i].internalGetMinimumWidth();
490          }
491          int x = Math.max(minSpaceOnLeft, bounds.x - pixels);
```

File2

```
No.  Code
547          }
548  }
549  void moveUp(CoolItem item, int x_root) {
550          Point point = findItem(item);
551          int oldRowIndex = point.y;
552          if (items[oldRowIndex].length == 1) {
553                  /* If this is the only item in the top row, don't move it. */
554                  if (oldRowIndex == 0) return;
555          }
556          boolean resize = removeItemFromRow(item, oldRowIndex, false);
557          Rectangle old = item.internalGetBounds();
558          internalRedraw(old.x, old.y, CoolItem.MINIMUM_WIDTH, old.height);
559          int newRowIndex = oldRowIndex - 1;
560          resize |= insertItemIntoRow(item, newRowIndex, x_root);
561          if (resize) {
562                  relayout();
563          } else {
564                  layoutItems();
565          }
566  }
567  void onDispose(Event event) {
568          /*
569           * Usually when an item is disposed, destroyItem will change the size of t...
570           * and reset the bounds of all the remaining cool items.
571         * Since the whole cool bar is being disposed, this is not necessary.  For speed
572           * the inDispose flag is used to skip over this part of the item dispose.
573           */
574          if (inDispose) return;
575          inDispose = true;
576          notifyListeners(SWT.Dispose, event);
```

Method1

Method Tree
▶ CoolBar
▶ checkStyle
▶ _setCursor
▶ checkSubclass
▶ computeSize
▶ getGrabbedItem
▶ getItem
▶ getItemCount
▶ getItems
▶ findItem
▶ fixEvent
▶ fixRectangle

Method2

Method Tree
▶ CoolBar
▶ checkStyle
▶ _setCursor
▶ checkSubclass
▶ computeSize
▶ getGrabbedItem
▶ getItem
▶ getItemCount
▶ getItems
▶ findItem
▶ fixEvent
▶ fixRectangle

Results

No.	Sim	Method 1	Start	End	Method 2	Start	End
1	0.94	insertItemIntoRow	314	383	removeItemFromRow	781	821
2	0.88	destroyItem	443	463	moveRight	511	548
3	0.97	moveDown	464	480	moveUp	549	566
4	0.96	moveLeft	481	510	moveRight	511	548
5	0.92	setItemOrder	944	964	setWrapIndices	1080	1104

Fig. 1. User Interface