Active Inductive Logic Programming for Code Search

Aishwarya Sivaraman, Tianyi Zhang, Guy Van den Broeck, Miryung Kim

University of California, Los Angeles

Tool and dataset: https://github.com/AishwaryaSivaraman/ALICE-ILP-for-Code-Search
Developers Often Search For Similar Code

- Bug fix [Kim et al., 2006]
- API-related refactoring [Dig and Johnson, 2006]
- Optimization [Ahmad and Cheung, 2018]
Existing Code Search

- Internet code search engines [Krugle, S6, CodeGenie]
  - Lacks expressiveness and query refinement is tedious
- Clone detection techniques [CCFinder, Deckard]
  - Threshold metric insufficient to capture the abstract search intent
- Interactive template based code search [Critics]
  - Interaction is tedious
ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: One code example
ALICE: Generates a query (a search pattern)
Output: Set of method locations that match the query
ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: More labels
ALICE: Interactive Code Search via Active Inductive Logic Programming

**Input:** More labels

**ALICE:** Refines the initial query (search pattern)

**Output:** A smaller set of method locations that match the new query
ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: More labels

Code Search Results (Iteration 2)

Query∧ a
ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: More labels
ALICE: Keep refining the query
Output: A smaller set of method locations that match the new query
Active Learning

• Obtaining labels is time consuming and expensive

Inductive Logic Programming

• Data as feature vectors cannot easily express the structure of code
• ILP: Positive examples + negative examples + background knowledge as rules
Represent Code as Logic Facts

<table>
<thead>
<tr>
<th>Fact Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (ID, CONDITION)</td>
</tr>
<tr>
<td>loop (ID, CONDITION)</td>
</tr>
<tr>
<td>parent (ID, ID)</td>
</tr>
<tr>
<td>next (ID, ID)</td>
</tr>
<tr>
<td>methodCall (ID, NAME)</td>
</tr>
<tr>
<td>type (ID, NAME)</td>
</tr>
<tr>
<td>exception (ID, NAME)</td>
</tr>
<tr>
<td>methodDec (ID, NAME)</td>
</tr>
</tbody>
</table>
Represent Code as Logic Facts

<table>
<thead>
<tr>
<th>Fact Predicate</th>
<th>Extracted Logic Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (ID, CONDITION)</td>
<td></td>
</tr>
<tr>
<td>loop (ID, CONDITION)</td>
<td></td>
</tr>
<tr>
<td>parent (ID, ID)</td>
<td></td>
</tr>
<tr>
<td>next (ID, ID)</td>
<td></td>
</tr>
<tr>
<td>methodCall (ID, NAME)</td>
<td></td>
</tr>
<tr>
<td>type (ID, NAME)</td>
<td></td>
</tr>
<tr>
<td>exception (ID, NAME)</td>
<td></td>
</tr>
<tr>
<td>methodDec (ID, NAME)</td>
<td></td>
</tr>
</tbody>
</table>

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```
Represent Code as Logic Facts

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```

**Extracted Logic Facts**

- `methodDec (0, queryDB), type (1, Connection), parent (0, 1)`
- `connection (ID, CONDITION)`
- `Statement stmt (ID, CONDITION)`
- `ResultSet rs (ID, CONDITION)`
- `while (ID, CONDITION)`
- `System.out.println (ID, CONDITION)`
- `catch (ID, CONDITION)`
- `DriverManager.getConnection (ID, CONDITION)`
- `con.createStatement (ID, CONDITION)`
- `con.close (ID, CONDITION)`
- `try (ID, CONDITION)`
- `try {`  
  - `Connection con (ID, CONDITION)`  
    - `DriverManager.getConnection (ID, CONDITION)`  
      - "jdbc:mysql://localhost:3306/db","root","root";  
  - `Statement stmt = con.createStatement();`  
  - `ResultSet rs = stmt.executeQuery("select * from emp");`  
  - `while (rs.next()) {`  
    - `System.out.println(rs.getInt(1));`  
  - `con.close();`  
  - `} catch (SQLException e) {`  
    - `System.out.println(e);`  
  - `}`
## Represent Code as Logic Facts

<table>
<thead>
<tr>
<th>Fact Predicate</th>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (ID, CONDITION)</td>
<td>if</td>
<td>Logical condition</td>
</tr>
<tr>
<td>loop (ID, CONDITION)</td>
<td>loop</td>
<td>Loop condition</td>
</tr>
<tr>
<td>parent (ID, ID)</td>
<td>parent</td>
<td>Parent relationship</td>
</tr>
<tr>
<td>next (ID, ID)</td>
<td>next</td>
<td>Next relationship</td>
</tr>
<tr>
<td>methodCall (ID, NAME)</td>
<td>methodCall</td>
<td>Method call</td>
</tr>
<tr>
<td>type (ID, NAME)</td>
<td>type</td>
<td>Type declaration</td>
</tr>
<tr>
<td>exception (ID, NAME)</td>
<td>exception</td>
<td>Exception declaration</td>
</tr>
<tr>
<td>methodDec (ID, NAME)</td>
<td>methodDec</td>
<td>Method declaration</td>
</tr>
</tbody>
</table>

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(  
            "jdbc:mysql://localhost:3306/db","root","root");  
        Statement stmt = con.createStatement();  
        ResultSet rs = stmt.executeQuery("select * from emp");  
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```

**Extracted Logic Facts**

- `methodDec (0, queryDB), type (1, Connection), parent (0, 1), methodCall(2, getConnection), parent (0, 2), next (2, 1)`
Represent Code as Logic Facts

public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
Formulate a Search Query

• A user selects a code example and annotate important features.

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```

A code example with user annotations
Logic-based Code Search

Search Query

\[ \text{methodDec}(i_0, m) \land \text{type}(i_1, \text{ResultSet}) \land \text{contains}(i_0, i_1) \land \text{methodCall}(i_2, \text{executeQuery}) \land \text{contains}(i_0, i_2) \land \text{looplike}(i_3, \text{".next()"}) \land \text{contains}(i_0, i_3) \]

Fact Base


Rules

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>iflike ((\text{ID}, \text{regex})) :- \text{if}(\text{ID}, \text{cond}), \text{match}(\text{cond}, \text{regex})</td>
</tr>
<tr>
<td>loollike ((\text{ID}, \text{regex})) :- \text{loop}(\text{ID}, \text{cond}), \text{match}(\text{cond}, \text{regex})</td>
</tr>
<tr>
<td>contains ((\text{ID}_1, \text{ID}_2)) :- \text{parent}(\text{ID}_1, \text{ID}_2)</td>
</tr>
<tr>
<td>contains ((\text{ID}_1, \text{ID}_3)) :- \text{parent}(\text{ID}_1, \text{ID}_2), \text{contains}(\text{ID}_2, \text{ID}_3)</td>
</tr>
<tr>
<td>before((\text{ID}_3, \text{ID}_2)) :- next(\text{ID}_2, \text{ID}_1)</td>
</tr>
<tr>
<td>before((\text{ID}_3, \text{ID}_1)) :- next(\text{ID}_2, \text{ID}_1), before(\text{ID}_2, \text{ID}_3).</td>
</tr>
</tbody>
</table>
Logic-based Code Search

Search Query

methodDec (i₀, m) \land
\text{type}(i₁, \text{ResultSet}) \land
\text{contains}(i₀, i₁) \land
\text{methodCall}(i₂, \text{executeQuery}) \land
\text{contains}(i₀, i₂) \land
\text{loopy}(i₃, "\text{"next()"}") \land
\text{contains}(i₀, i₃)

Fact Base

Fact Rules

Matched Code

public void getUserName(String id) {
    try {
        ResultSet set = db.executeQuery(
            "select name from users where id=" + id);
        while (set.next()) { … }
    } catch (SQLException e) { … }
}

public void queryDatabase() {
    try {
        ResultSet result = s.executeQuery("select * from customers");
        while (result.next()) { … }
    } catch (SQLException e) { … }
}

public List get() {
    ResultSet set = stmt.executeQuery("select * from t");
    List l = new List();
    while (set.next()) { … }
    return l;
}

and 32 other matched locations
public void getUserName(String id) {
    try {
        ResultSet set = db.executeQuery(
            "select name from users where id=" + id);
        while (set.next()) {...}
    } catch (SQLException e) {...}
}

public void queryDatabase() {
    try {
        ResultSet result = s.executeQuery("select *
            from customers");
        while (result.next()) {...}
    } catch (SQLException e) {...}
}

public List get() {
    ResultSet set = stmt.executeQuery("select *
            from t");
    List l = new List();
    while (set.next()) {...}
    return l;
}
Query Refinement via Active Learning

```java
public void getUserName(String id) {
    try {
        ResultSet set = db.executeQuery("select name from users where id=\"+ id);
        while (set.next()) { … }
    } catch (SQLException e) { … }
}

public void queryDatabase() {
    try {
        ResultSet result = s.executeQuery("select * from customers");
        while (result.next()) { … }
    } catch (SQLException e) { … }
}

public List get() {
    ResultSet set = stmt.executeQuery("select * from t");
    List l = new List();
    while (set.next()) { … }
    return l;
}
```

**Refined Query**

```java
methodDec (i_0, m) \land
\quad type (i_1, ResultSet) \land
\quad contains (i_0, i_1) \land
\quad methodCall (i_2, executeQuery) \land
\quad contains (i_0, i_2) \land
\quad looplike (i_3, ".next()") \land
\quad contains (i_0, i_3)
```

**Query Refinement Optimization**

\[
\text{Specialize}(h_{i-1}, P, N) = \arg \max_{h_i} \sum_{p \in P} [p \models h_i]
\]

such that \( h_i \models h_{i-1} \) and \( \forall n \in N, n \not\models h_i \)
Query Refinement via Active Learning

```java
public void getUserName(String id) {
    try {
        ResultSet set = db.executeQuery("select name from users where id=" + id);
        while (set.next()) { ... }
    } catch (SQLException e) { ... }
}

public void queryDatabase() {
    try {
        ResultSet result = s.executeQuery("select * from customers");
        while (result.next()) { ... }
    } catch (SQLException e) { ... }
}

public List get() {
    ResultSet set = stmt.executeQuery("select * from t");
    List l = new List();
    while (set.next()) { ... }
    return l;
}
```

Refined Query

```java
methodDec (i₀, m) ∧
type (i₁, ResultSet) ∧
contains (i₀, i₁) ∧
methodCall((s, executeQuery) ∧
contains (i₀, i₂) ∧
looplike (i₃, "*.next()") ∧
contains (i₃, i₄) ∧
exception (i₄, SQLException),
contains (i₄, i₄)
```

Query Refinement Optimization

\[ \text{Specialize}(h_{i-1}, P, N) = \arg \max_{h_i} \sum_{n \in P} [p \models h_i] \]

such that \( h_i \models h_{i-1} \) and \( \forall n \in N, n \neq h_i \)
How To Pick a Discriminatory Atom?

A code example with user annotations

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db", "root", "root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```

User annotations

Potential Candidate Features
Inductive Bias

1. **Feature Vector** considers source code has a flat structure

2. **Nested Structure** prioritizes code elements with containment relationship

3. **Sequential Code Order** prioritizes code elements with sequential ordering

A code example with user annotations

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs =stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    }
    catch (SQLException e) {
        System.out.println(e);
    }
}
```
Inductive Bias

1. **Feature Vector** considers source code has a flat structure

2. **Nested Structure** prioritizes code elements with containment relationship

3. **Sequential Code Order** prioritizes code elements with sequential ordering

A code example with user annotations

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection(  
            "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e){
        System.out.println(e);
    }
}
```
Inductive Bias

1. *Feature Vector* considers source code has a flat structure

2. *Nested Structure* prioritizes code elements with containment relationship

3. *Sequential Code Order* prioritizes code elements with sequential ordering

```java
public void queryDB() {
    try {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next()) {
            System.out.println(rs.getInt(1));
        }
        con.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
}
```
public class DefaultCommentMapper {
    
    public int getLeadingCommentNodes() {
        int range = null;
        for (int i = 0; range == null && i < this.leadingPtr; i++) {
            if (this.leadingNodes[i] == node) range = this.leadingIndexes[i];
        }
        return range;
    }

    
    public Comment[] getLeadingComments(ASTNode node) {
        if (this.leadingPtr == 0) {
            int[] range = null;
            for (int i = 0; range == null && i < this.leadingPtr; i++) {
                if (this.leadingNodes[i] == node) range = this.leadingIndexes[i];
            }
            if (range != null) {
                int length = range[1] - range[0] + 1;
                Comment[] leadComments = new Comment[length];
                System.arraycopy(this.comments, range[0], leadComments, 0, length);
                return leadComments;
            }
            return null;
        }
    }

    
    public Comment[] getLeadingCommentsAnomaly2(ASTNode node) {
        if (this.leadingPtr == 0) {
            int[] range = null;
            for (int i = 0; range == null && i < this.leadingPtr; i++) {
                if (this.leadingNodes[i] == node) range = this.leadingIndexes[i];
            }
            if (range != null) {
                int length = range[1] - range[0] + 1;
                Comment[] leadComments = new Comment[length];
                System.arraycopy(this.comments, range[0], leadComments, 0, length);
                return leadComments;
            }
            return null;
        }
    }
}
Evaluation

Simulation Experiments

A Comparison with Critics

A Case Study with Real Users
Evaluation

Simulation Experiments

A Comparison with Critics

A Case Study with Real Users
Experiment Benchmarks

• Similar locations to update [Meng et al., 2013]
  • 14 groups of syntactically similar code fragments from Eclipse JDT and SWT

• Code optimization [Ahmad et al., 2018]
  • 6 groups of similar programs that follow the same code pattern
(RQ1) Which inductive bias is effective?

- Nested structure bias is the most effective.

* Averaged over 10 runs.
(RQ2) How much does a user should annotate?

- **Method**: Randomly annotate important code elements in an example

- **Result**: Annotating more features increases precision but not recall.

<table>
<thead>
<tr>
<th></th>
<th>1 Feature</th>
<th>2 Features</th>
<th>3 Features</th>
<th>4 Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.16</td>
<td>0.47</td>
<td>0.68</td>
<td>0.80</td>
</tr>
<tr>
<td>Recall</td>
<td>0.91</td>
<td>0.86</td>
<td>0.80</td>
<td>0.78</td>
</tr>
</tbody>
</table>

* Averaged over 10 runs.
(RQ3) How many labels should a user provide?

• **Method:** Label randomly selected search results w.r.t. the ground truth.

• **Results:** Labeling three examples is optimal.

<table>
<thead>
<tr>
<th></th>
<th>2 Labels</th>
<th>3 Labels</th>
<th>4 Labels</th>
<th>5 Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>1.0</td>
<td>0.88</td>
<td>0.81</td>
<td>0.75</td>
</tr>
<tr>
<td><strong># Iterations</strong></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong># Total Labels</strong></td>
<td>14</td>
<td>18</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

* Averaged over 10 runs.
(RQ4) What if a user makes mistakes?

- **Method**: Flip a label (e.g., positive -> negative) with a probability.

- **Result**: Report contradictory labels immediately and behave robustly when no inconsistencies are found.

<table>
<thead>
<tr>
<th>Error Rate</th>
<th>10%</th>
<th>20%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Recall</td>
<td>0.95</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td>% of Inconsistency feedback</td>
<td>33%</td>
<td>60%</td>
<td>54%</td>
</tr>
</tbody>
</table>

* Averaged over 10 runs.
Overall Performance

• Simulate user behavior
  • Randomly select a code fragment in each group as a seed example
  • Randomly tag two important features
  • Randomly label three examples w.r.t. the ground truth

• 93% precision and 96% recall in 3 search iterations

* Averaged over 10 runs.
Evaluation

Simulation Experiment

Comparison with Critics

Case Study with Real Users
Comparison with Critics [Zhang et al., ICSE 2015]

• Critics supports interactive code search via template refinement.

try {
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from emp");
    while (rs.next()) {
        System.out.println(rs.getInt(1));
    }
    con.close();
} catch (SQLException e) {
    System.out.println(e);
}

A concrete code example

try {
    $EXCLUDE
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from emp");
    while (rs.next()) {
        System.out.println(rs.getInt(1));
    }
    $v0.close();
} catch (SQLException e) {
    System.out.println(e);
}

A search template
Comparison with Critics [Zhang et al., ICSE 2015]

• Critics supports interactive code search via template refinement.

```
try {
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/db","root","root");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from emp");
    while (rs.next()) {
        System.out.println(rs.getInt(1));
    }
} catch (SQLException e) {
    System.out.println(e);
}
```

A concrete code example

```
try {
    $EXCLUDE
    $t1 $v1 = $v0.$m1();
    ResultSet rs = $v1.executeQuery("select * from emp");
    while (rs.next()) {
        System.out.println(rs.getInt(1));
    }
    $v0.close();
} catch (SQLException e) {
    System.out.println(e);
}
```

A search template
Comparison with Critics [Zhang et al., ICSE 2015]

- Critics supports interactive code search via template refinement.

```java
try {
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db", "root", "root");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from emp");
    while (rs.next()) {
        System.out.println(rs.getInt(1));
    }
    con.close();
} catch (SQLException e) {
    System.out.println(e);
}
```

A concrete code example

```java
try {
    $EXCLUDE
    \$t1 \$v1 = \$v0.$m1();
    ResultSet \$v2 = \$v1.executeQuery($v3);
    while ($v2.next()) {
        System.out.println(rs.getInt(1));
    }
    $v0.close();
} catch (SQLException e) {
    System.out.println(e);
}
```

A search template
Comparison with Critics [Zhang et al., ICSE 2015]

- Critics supports interactive code search via template refinement.

```java
try {
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from emp");
    while (rs.next()) {
        System.out.println(rs.getInt(1));
    }
    con.close();
} catch (SQLException e) {
    System.out.println(e);
}
```

A concrete code example

```java
try {
    $EXCLUDE
    $t1 $v1 = $v0.$m1();
    ResultSet $v2 = $v1.executeQuery($v3);
    while ($v2.next()) {
        $EXCLUDE
    }
    $v0.close();
} catch (SQLException $v4) {
    $EXCLUDE
}
```

A search template
Comparison with Critics

- ALICE achieves comparable or better accuracy with fewer iterations.

<table>
<thead>
<tr>
<th>Group ID</th>
<th>ALICE</th>
<th>Critics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>6</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>7</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Average</td>
<td>0.86</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Evaluation

Simulation Experiment

Comparison with Critics

Case Study with Real Users
Case Study: Eclipse SWT Revision 16379

- Recruit three graduate students to perform a code search task
- Participants can
  - easily recognize important features to annotate
  - distinguish positive and negative examples without much effort

<table>
<thead>
<tr>
<th>Participant</th>
<th>#Examples</th>
<th>#Positives</th>
<th>#Negatives</th>
<th>Time Taken(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>P2</td>
<td>437</td>
<td>0</td>
<td>2</td>
<td>55</td>
</tr>
<tr>
<td>P3</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>
Summary

• A novel learning based paradigm that lets users to express search intent via annotation and labelling.

• Our inductive bias eliminates tedious labelling effort by requiring a user to label a partial dataset.

• Our active learning engine enables an easy query refinement by leverage both positive and negative examples.

• A comprehensive simulation and a case study with real users indicate that interactivity pays off.

Tool and dataset: https://github.com/AishwaryaSivaraman/ALICE-ILP-for-Code-Search
Q & A
Backup Slide
Future Work

• Extend fact predicates to capture more program syntax and semantics
  • e.g., switch statements, def-use relationships, etc

• Support logical negations in the query language

• Support backtracking
Research Questions

• (RQ1) How much does a user need to annotate?
• (RQ2) How many labels should a user provide?
• (RQ3) Which inductive bias is effective?
• (RQ4) What if a user makes a mistake?
(RQ1) Which inductive bias is effective?

- Nested structure bias is the most effective.

* Averaged over 10 runs.
(RQ1) Which inductive bias is effective?

- Nested structure bias is the most effective.

* Averaged over 10 runs.
(RQ5) Should a user provide positive and negative examples?

• **Method:** Labelling both positive & negative, only positive and only negative

• **Results:** Labelling both positive and negative is optimal

* Averaged over 10 runs.
(RQ5) Should a user provide positive and negative examples?

- **Method:** Labelling both positive & negative, only positive and only negative

- **Results:** Labelling both positive and negative is optimal

* Averaged over 10 runs.
(RQ2) How much does a user should annotate?

• **Method:** Randomly annotate important code elements in an example

• **Result:** Annotating more features increases precision but not recall.

<table>
<thead>
<tr>
<th></th>
<th>1 Feature</th>
<th>2 Features</th>
<th>3 Features</th>
<th>4 Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.16</td>
<td>0.47</td>
<td>0.68</td>
<td>0.80</td>
</tr>
<tr>
<td>Recall</td>
<td>0.91</td>
<td>0.86</td>
<td>0.80</td>
<td>0.78</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.27</td>
<td>0.61</td>
<td>0.74</td>
<td>0.79</td>
</tr>
</tbody>
</table>

* Averaged over 10 runs.