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

Code Search Results (Iteration 2)
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: 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

```
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)`
- `methodCall (ID, NAME)`
- `parent (ID, ID)`
- `next (ID, ID)`
- `type (ID, NAME)`
- `exception (ID, NAME)`
- `if (ID, CONDITION)`
- `loop (ID, CONDITION)`
- `methodDec (ID, NAME)`

Fact Predicate

- `public`
- `void`
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, 0)`
- `Fact Predicate`
  - `if (ID, CONDITION)`
  - `loop (ID, CONDITION)`
  - `parent (ID, ID)`
  - `next (ID, ID)`
  - `methodCall (ID, NAME)`
  - `type (ID, NAME)`
  - `exception (ID, NAME)`
  - `methodDec (ID, NAME)`
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>methodDec (0, queryDB),</td>
</tr>
<tr>
<td>loop (ID, CONDITION)</td>
<td>type (1, Connection),</td>
</tr>
<tr>
<td>parent (ID, ID)</td>
<td>parent (0, 1),</td>
</tr>
<tr>
<td>next (ID, ID)</td>
<td>methodCall(2, getConnection),</td>
</tr>
<tr>
<td>methodCall (ID, NAME)</td>
<td>parent (0, 2),</td>
</tr>
<tr>
<td>type (ID, NAME)</td>
<td>next (2, 1)</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

```
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),
  ...
  loop (7, "rs.next()"),
  methodCall (8, getInt),
  parent (7, 8),
  ...
  exception (10, SQLException),
  parent (0, 10),
  ..."
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

search query
Logic-based Code Search

Logic-based Code Search

**Search Query**

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

**Fact Base**

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

**Matched Code**

```
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) { ... }
}

class Database {
    public List get() {
        ResultSet set = stmt.executeQuery("select * from t");
        List l = new List();
        while (set.next()) { ... }
        return l;
    }
    
    public void queryDatabase() {
        try {
            ResultSet result = s.executeQuery("select * from customers");
            while (result.next()) { ... }
        } catch (SQLException e) { ... }
    }
    
    public void queryUsers() {
        try {
            ResultSet users = db.executeQuery("select name from users");
            while (users.next()) { ... }
        } catch (SQLException e) { ... }
    }
}

Partial Feedback

Search Query

methodDec (i_0, m) \land
  type (i_2, ResultSet) \land
  contains (i_0, i_1) \land
  methodCall(i_2, executeQuery) \land
  contains (i_0, i_2) \land
  looplike (i_3, "*.next()") \land
  contains (i_0, i_3)
Query Refinement via Active Learning

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

Query Refinement Optimization

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

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

Refined Query Refinement Optimization

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 $\forall n \in N, n \not\models h_i$
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

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);
    }
}
```
```java
    } return index;

    Comment[] getLeadingComments(ASTNode node) {
        if (this.leadingPtr >= 0) {
            int[] range = null;
            for (int i=0; range == null && i <= this.leadingPtr; i++) {
                if (this.leadNodes[i] == node) range += this.leadIndexes[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;
    }

    Comment[] getLeadingCommentsAnomaly2(ASTNode node) {
        // Implementation...
    }
```
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>Precision</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Recall</td>
<td>1.0</td>
<td>0.88</td>
<td>0.81</td>
<td>0.75</td>
</tr>
<tr>
<td># Iterations</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td># Total Labels</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></th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10%</td>
</tr>
<tr>
<td>Precision</td>
<td>1.0</td>
</tr>
<tr>
<td>Recall</td>
<td>0.95</td>
</tr>
<tr>
<td>% of Inconsistency feedback</td>
<td>33%</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.

A concrete code example

```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 search template

```java
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);
}
```
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 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 ($v2.next()) {
        System.out.println($v2.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));
    }
    con.close();
} catch (SQLException e) {
    System.out.println(e);
}
```
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><strong>Average</strong></td>
<td><strong>0.86</strong></td>
<td><strong>1.0</strong></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