Applying Flow Graph Mining to the Performance Analysis of Flat Profile Applications

Carolina Simões Gomes and Jose Nelson Amaral
Department of Computing Science, University of Alberta, Canada.

Li Ding, Arie Tal and Joran Siu
IBM Toronto Lab, Canada.
Motivation

- Figure out optimization opportunities from profile data
- Optimize cycle for compiler developers
- Analyze profiled data
- Organize collected profiles
- Compile application with optimizations
- Profile running application
Motivation

- Figure out optimization opportunities from profiled data
- (Re)write application source code
- (Re)compile application
- Analyze profiled data
- Organize collected profiles
- Profile running application
Motivation
Problem Statement

- How to facilitate the performance analysis of flat-profile applications?
Problem Statement

- How to facilitate the performance analysis of flat-profile applications?

- More specifically: how to automate the search for execution patterns in flat-profile applications, that may indicate the need for optimization?
Problem Statement

• How to facilitate the performance analysis of flat-profile applications?

• More specifically: how to automate the search for execution patterns in flat-profile applications, that may indicate the need for optimization?

• Optimization may be at different levels, e.g. hardware architecture, code generation, application source-code
Idea

Problem:

Mine for frequent patterns of execution in a program
Idea

Problem:
Mine for frequent patterns of execution in a program

Possible Solution:
Mine for frequent sub-graphs in a flow graph
Fundamental Concepts

- **Execution pattern**: set of attributes that characterize distinct executed regions of the program

- Program regions that map to a pattern are called **pattern instances**

- Two program regions that contain the same attributes are two instances of the same pattern
Fundamental Concepts

- What makes a pattern interesting?

- **Support value**: measure of how interesting the pattern is.

- **Frequent execution pattern**: a pattern that has a support value higher than a threshold. The support value of a pattern is calculated from all its instances.
Execution Flow Graph

Possible attributes: a, b, c, d

Edge frequency: how often it is executed

L(v): node label, an unique identifier for the node

Node weight: time spent executing the node (e.g. in clock ticks)

a(10) b(20)

Attributes that characterize the node, and corresponding attribute values (single value per attribute)
Execution Flow Graph

- Generic representation that places together static and dynamic data
- Can be adapted to different mining granularities
Solution: FlowGSpan

- Based on gSpan (Yan and Han, 2002) and FlowGSP (Jocksch et al., 2010)
- Mines for sequential execution patterns (sub-paths) and execution patterns with branches (sub-graphs)
- Maps frequent patterns to pattern instances
- Uses support criteria based on attributed, weighted nodes and weighted edges
Support Criteria

- Weight support \((Sw)\)
- Frequency support \((Sf)\)
- Support value \((Sm = \max\{Sw, Sf\})\)
- Anti-monotonicity property
Support Criteria

\[ Sw(\text{inst1}) = \min\{ w(v1), w(v2), w(v3) \} \]
\[ Sw(\text{inst2}) = \min\{ w(v1), w(v2), w(v3) \} \]
\[ Sw(\text{Pattern}) = \frac{(Sw(\text{inst1}) + Sw(\text{inst2}))}{\text{total\_weight}} \]

\[ Sf(\text{inst1}) = \min\{ f(v1, v2), f(v1, v3) \} \]
\[ Sf(\text{inst2}) = \min\{ f(v6, v7), f(v6, v8) \} \]
\[ Sf(\text{Pattern}) = \frac{(Sf(\text{inst1}) + Sf(\text{inst2}))}{\text{total\_freq}} \]

\[ \text{total\_weight} = 150 \]
\[ \text{total\_freq} = 80 \]

Dataset
FlowGSpan Example

• Procedure:
  - generation of candidate sub-graph $g$ of size $k$ by combining possible attributes
  - matching of $g$ on dataset
  - support value calculation of matches of $g$
  - comparison of support value of $g$ against threshold
  - if $g$ is not frequent, discard it
  - else extend $g$ by adding an edge to it, that can either be connected to a new node or to a node already in $g$
FlowGSpan Example

- Support threshold (minSup): 0.1
- Possible attributes: a, b, c, d
- Dataset size: 2 (in number of EFGs)
FlowGSpan Example

0-edge sub-graphs

- **a**
  - $Sw = (10 + 15 + 3 + 2)/150 = 0.2$
  - $Sf = 0$
  - $Sm = \text{max}(0.2, 0) = 0.2$

- **b**
  - $Sw = (5 + 2 + 5 + 4 + 6)/150 = 0.15$
  - $Sf = 0$
  - $Sm = 0.15$

- **c**
  - $Sw = 0.04$
  - $Sf = 0$
  - $Sm = 0.04$

- **d**
  - $Sw = 0.09$
  - $Sf = 0$
  - $Sm = 0.09$

- **v1**
  - 30
  - 4
  - 10

- **v2**
  - 10
  - 20
  - 4
  - 6

- **v3**
  - 20
  - 6

- **v4**
  - 20
  - 20
  - 10
  - 4

- **v5**
  - 20
  - 6

- **v6**
  - 10
  - 14

- **v7**
  - 20
  - 7

- **v8**
  - 10
  - 6

- **v9**
  - 10
  - 7

**total_weight = 150**
**total_freq = 80**
FlowGSpan Example

0-edge sub-graphs

**Example:**

- **a:**
  - $Sw = (10+15+3+2)/150 = 0.2$
  - $Sf = 0$
  - $Sm = \max(0.2, 0) = 0.2$

- **b:**
  - $Sw = (5+2+5+4+6)/150 = 0.15$
  - $Sf = 0$
  - $Sm = 0.15$

- **c:**
  - $Sw = 0.04$
  - $Sf = 0$
  - $Sm = 0.04$

- **d:**
  - $Sw = 0.09$
  - $Sf = 0$
  - $Sm = 0.09$

**Graph:**
- **v1:** 10
- **v2:** 4
- **v3:** 6
- **v4:** 20
- **v5:** 20
- **v6:** 14
- **v7:** 10
- **v8:** 20
- **v9:** 10

**Weights:**
- **v1:** 30
- **v2:** a(10)
- **v3:** b(5)
- **v4:** a(15)
- **v5:** b(4)
- **v6:** a(3)
- **v7:** c(4)
- **v8:** b(6)
- **v9:** c(3)

**Total Weight:** 150
**Total Frequency:** 80
FlowGSpan Example

0-edge sub-graphs

- a
  - $Sw = (10 + 15 + 3 + 2) / 150 = 0.2$
  - $S_f = 0$
  - $S_m = \max\{0.2, 0\} = 0.2$

- b
  - $Sw = (5 + 2 + 5 + 4 + 6) / 150 = 0.15$
  - $S_f = 0$
  - $S_m = 0.15$

- c
  - $Sw = 0.04$
  - $S_f = 0$
  - $S_m = 0.04$

- d
  - $Sw = 0.09$
  - $S_f = 0$
  - $S_m = 0.09$

- a, b
  - $Sw = 5 / 150 = 0.03$
  - $S_f = 0$
  - $S_m = 0.03$

- v1
  - $S_f(a(10)) = 4$
  - $S_f(b(5)) = 6$
  - $S_f(c(4)) = 7$
  - $S_f(d(5)) = 6$

- v2
  - $S_f(a(2)) = 10$
  - $S_f(b(2)) = 10$

- v3
  - $S_f(a(15)) = 20$
  - $S_f(b(5)) = 20$

- v4
  - $S_f(a(15)) = 20$
  - $S_f(b(4)) = 20$

- v5
  - $S_f(b(4)) = 20$

- v6
  - $S_f(a(3)) = 10$
  - $S_f(b(6)) = 20$

- v7
  - $S_f(a(2)) = 10$
  - $S_f(c(3)) = 20$

- v8
  - $S_f(b(6)) = 20$

- v9
  - $S_f(d(8)) = 10$

- total_weight = 150
- total_freq = 80
FlowGSpan Example

0-edge sub-graphs

- **a**
  - \( S_w = \frac{10+15+3+2}{150} = 0.2 \)
  - \( S_f = 0 \)
  - \( S_m = \max(0.2, 0) = 0.2 \)

- **b**
  - \( S_w = \frac{5+2+5+4+6}{150} = 0.15 \)
  - \( S_f = 0 \)
  - \( S_m = 0.15 \)

- **c**
  - \( S_w = 0.04 \)
  - \( S_f = 0 \)
  - \( S_m = 0.04 \)

- **d**
  - \( S_w = 0.09 \)
  - \( S_f = 0 \)
  - \( S_m = 0.09 \)

- **a, b**
  - \( S_w = \frac{5}{150} = 0.03 \)
  - \( S_f = 0 \)
  - \( S_m = 0.03 \)

---

```plaintext
v1
---
v2
---
v3
---
v4
---
v5
---
v6
---
v7
---
v8
---
v9
---
```

- Total weight = 150
- Total freq = 80
FlowGSpan Example

1-edge sub-graphs

\[ Sw = \frac{2+5+4+3}{150} = 0.09 \]
\[ Sf = \frac{4+6+10+7}{80} = 0.3 \]
\[ Sm = \max\{0.09, 0.3\} = 0.3 \]

\[ Sw = \frac{2+5}{150} = 0.05 \]
\[ Sf = \frac{4+6}{80} = 0.13 \]
\[ Sm = \max\{0.05, 0.13\} = 0.13 \]

\[ Sw = \frac{2+5+2}{150} = 0.06 \]
\[ Sf = \frac{4+6+7}{80} = 0.2 \]
\[ Sm = \max\{0.06, 0.2\} = 0.2 \]

Node pool:

\[ a \]
\[ b \]
FlowGSpan Example

- For 2-edge sub-graphs onwards...
- Approach based on gSpan: edge-by-edge pattern-growth (extends sub-graph by testing all combinations from frequent node pool)
- Optimized approach: edge combination
- Sub-graph matching issue: restarting search for every candidate sub-graph
FlowGSpan Example

Core optimization: registration of pattern instances
Application: targeting compiler developers

- Implemented FlowGSpan to mine for sets of hardware events
- Matching is exact
- Tested on DayTrader benchmark, which was JITted and profiled on IBM's z196 mainframe architecture
- Compared against optimized FlowGSP (with added pattern instance registration)
Application: targeting compiler developers
Application: targeting compiler developers
Application: targeting application developers

- Implementing FlowGSpan to mine for higher-level patterns (“source-code patterns”)
- Idea: flow graph mining at basic block level
- Challenges:
  - How to define basic block similarity?
  - Approximate matching of patterns
  - How to map from patterns to corresponding source lines?
Conclusion

- **FlowGSpan**: an algorithm that performs attributed sub-graph mining in Execution Flow Graphs
- **FlowGSpan** can be adapted according to the semantics of the dataset of Execution Flow Graphs to be mined
- Efficient implementation is fundamental to achieve acceptable performance when mining real-world, multi-GB datasets
- Large business applications can greatly benefit from automated performance analysis using **FlowGSpan**
Questions?