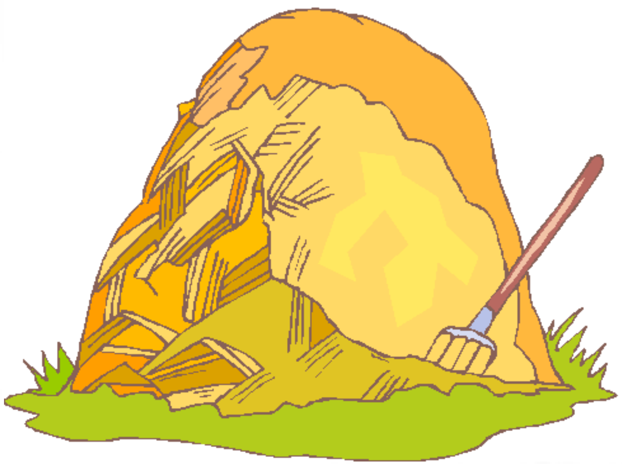
Applying Flow Graph Mining to the Performance Analysis of Flat Profile Applications

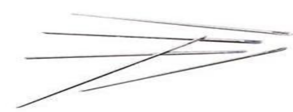
Carolina Simões Gomes and Jose Nelson Amaral

Department of Computing Science, University of Alberta, Canada.

Li Ding, Arie Tal and Joran Siu

IBM Toronto Lab, Canada.





Motivation

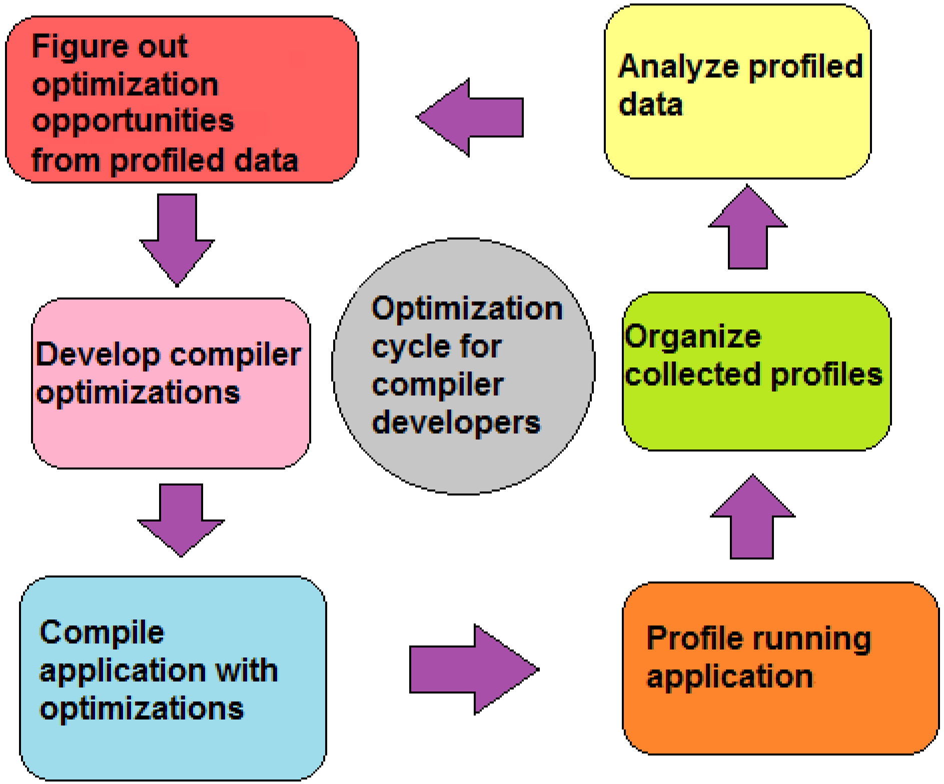


Figure out optimization opportunities from profiled data
Analyze profiled data
Develop compiler optimizations
Optimization cycle for compiler developers
Organize collected profiles
Compile application with optimizations
Profile running application

Motivation

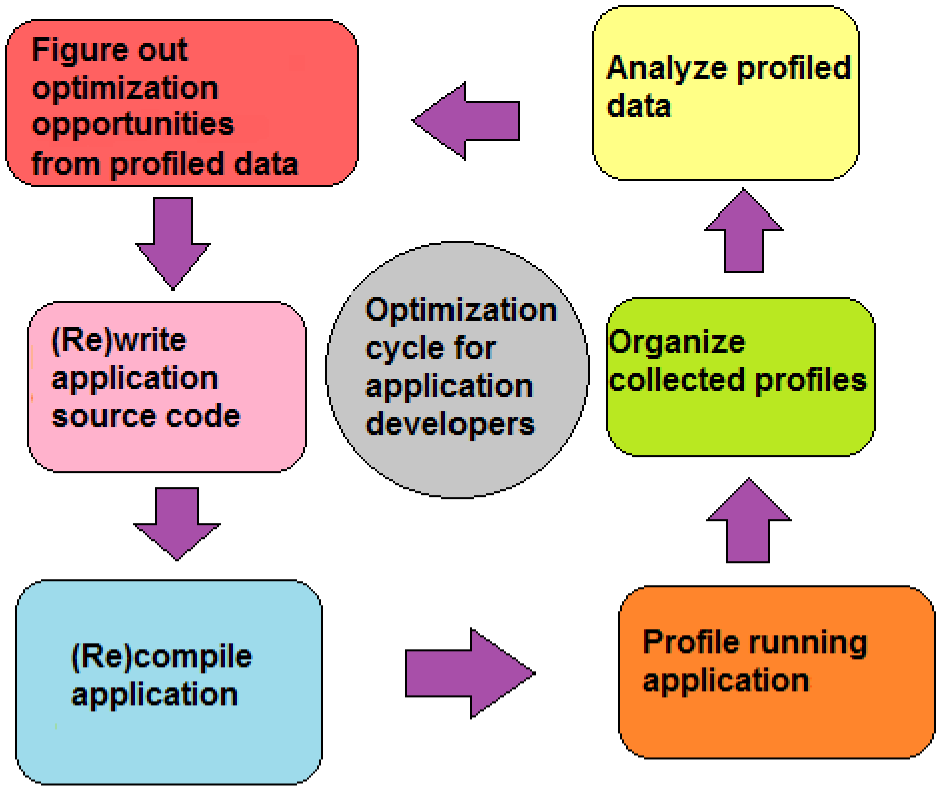
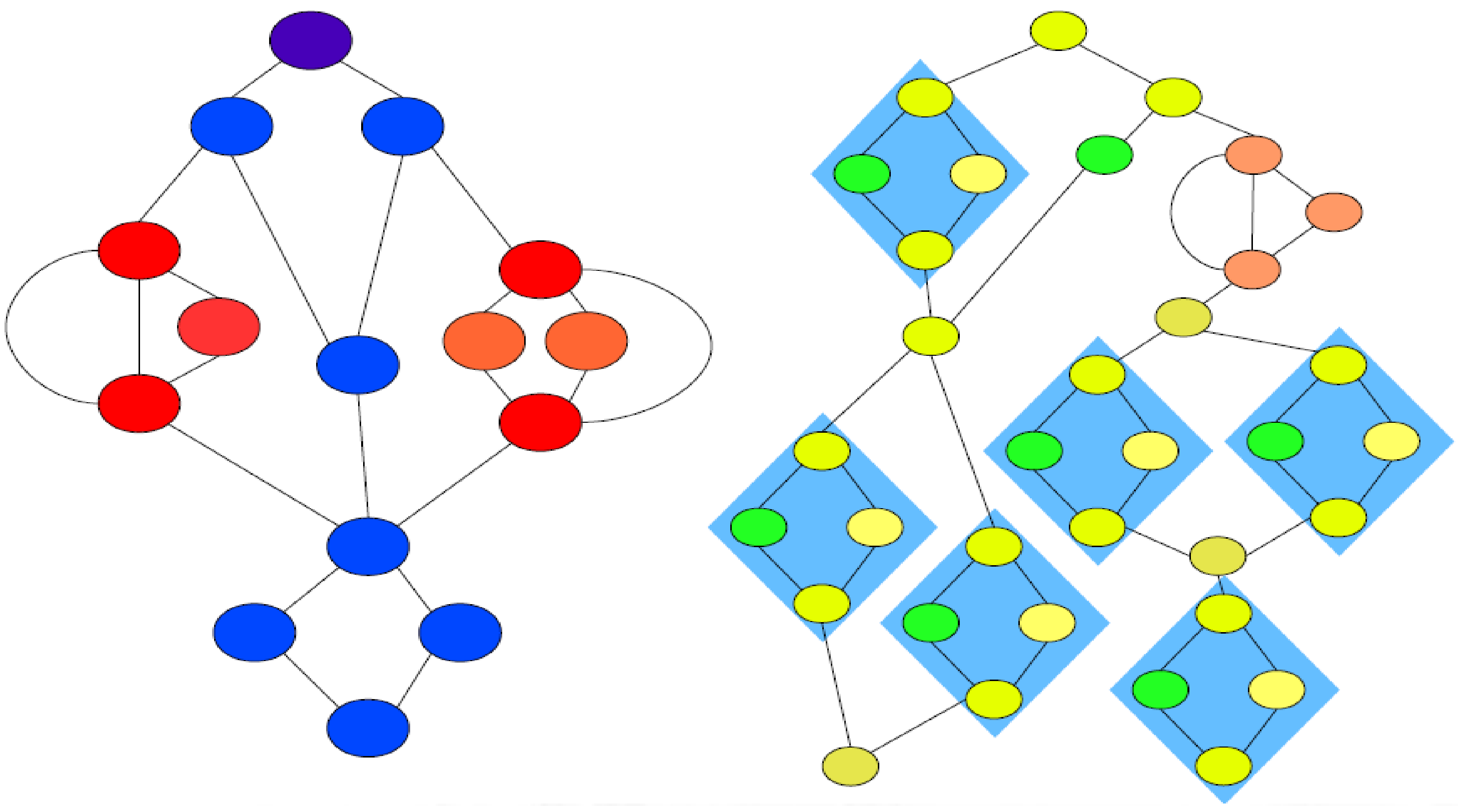


Figure out optimization opportunities from profiled data
Analyze profiled data
(Re)write application source code
Optimization cycle for application developers
Organize collected profiles
(Re)compile application
Profile running application

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

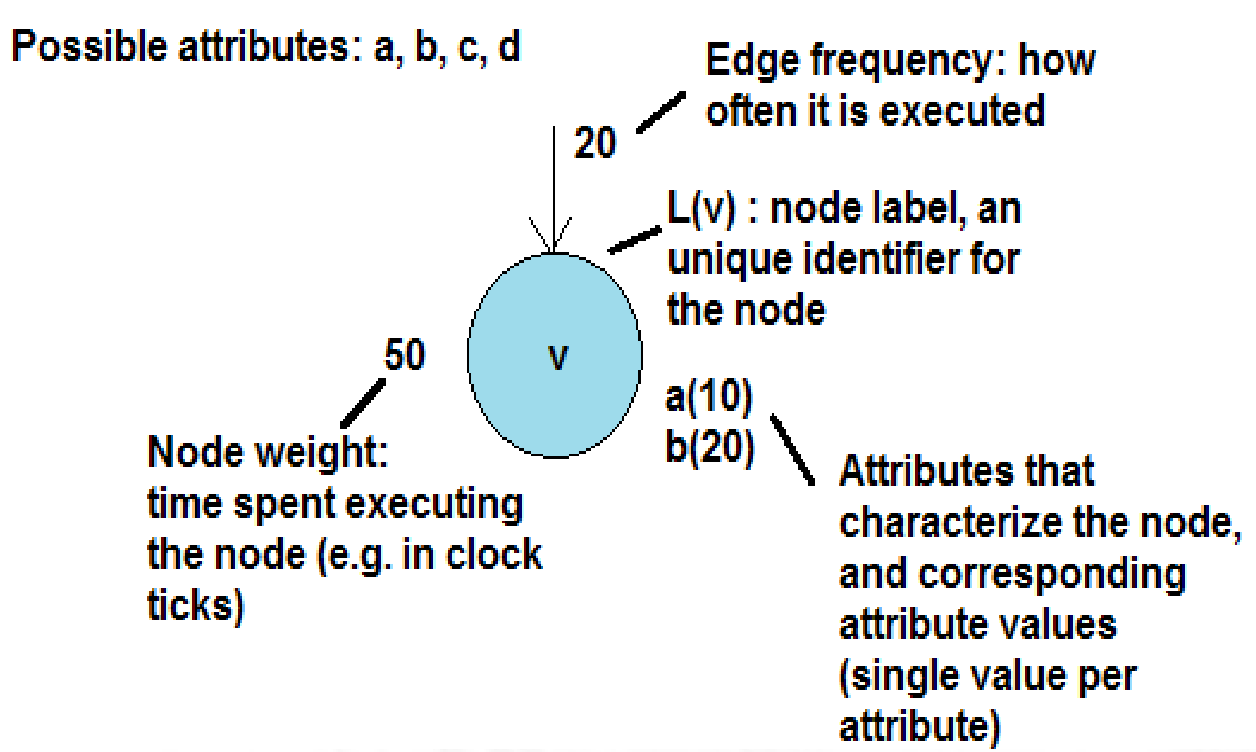
Fundamental Concepts

- **What makes a pattern interesting?**

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

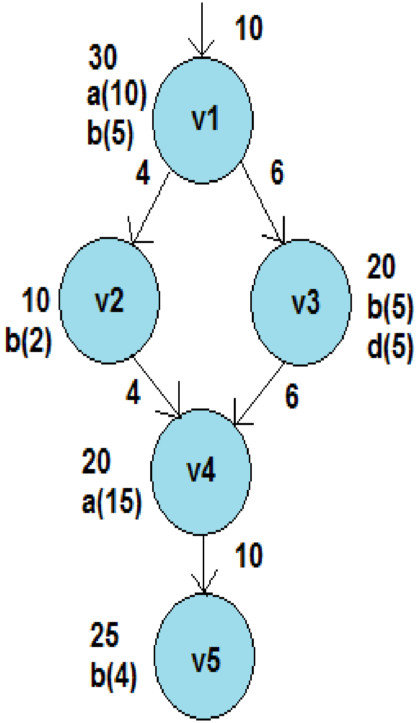
- **Frequent execution pattern**: a pattern that has a support value higher than a threshold. The support value of a pattern is calculated from all its instances

Execution Flow Graph



Execution Flow Graph

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



Solution: FlowGSpan

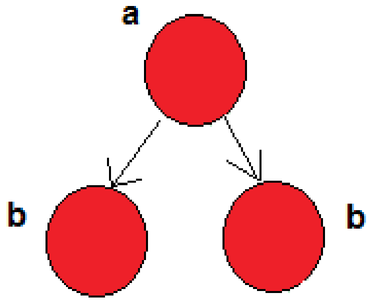
- Based on gSpan (*Yan and Han, 2002*) and FlowGSP (*Jocksch et al., 2010*)
- Mines for sequential execution patterns (sub-paths) and execution patterns with branches (sub-graphs)
- Maps frequent patterns to pattern instances
- Uses support criteria based on attributed, weighted nodes and weighted edges

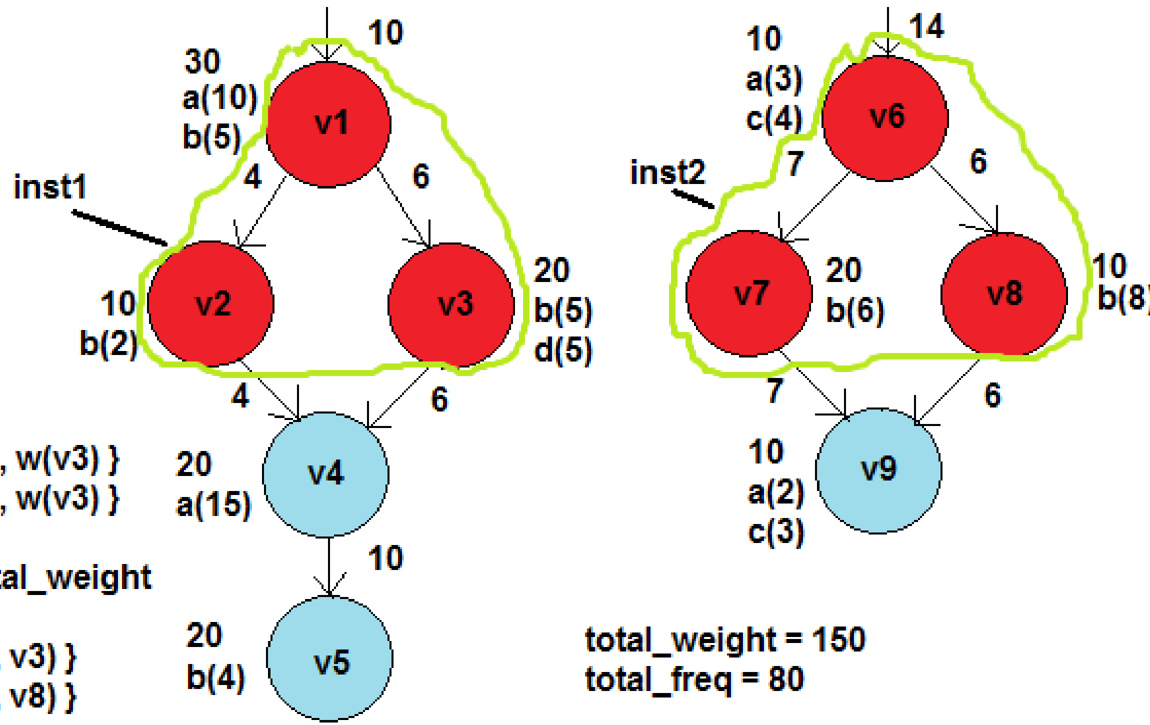
Support Criteria

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

Support Criteria

Pattern





Sw(inst1) = min{ w(v1), w(v2), w(v3) }
Sw(inst2) = min{ w(v1), w(v2), w(v3) }
Sw(Pattern) =
(Sw(inst1) + Sw(inst2))/total_weight

Sf(inst1) = min{ f(v1, v2), f(v1, v3) }
Sf(inst2) = min{ f(v6, v7), f(v6, v8) }
Sf(Pattern) =
(Sf(inst1) + Sf(inst2))/total_freq

total_weight = 150
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 = max{0.2, 0} = 0.2

b

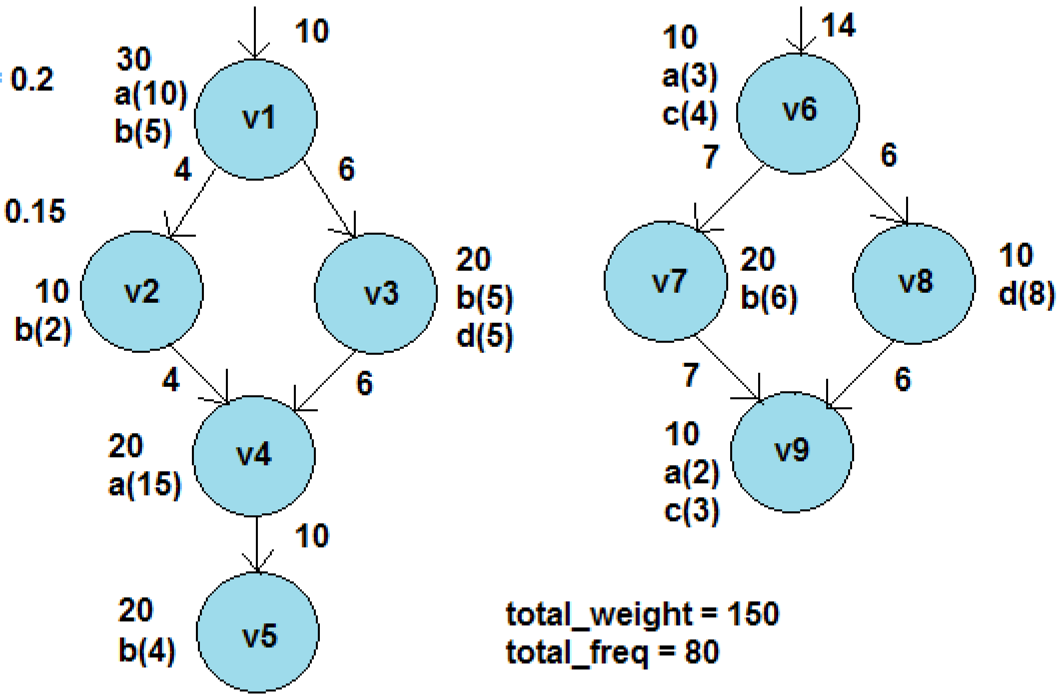
Sw = (5+2+5+4+6)/150 = 0.15
Sf = 0
Sm = 0.15

c

Sw = 0.04
Sf = 0
Sm = 0.04

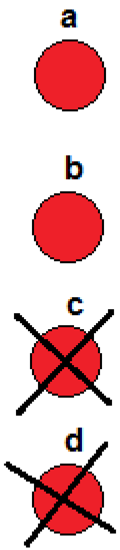
d

Sw = 0.09
Sf = 0
Sm = 0.09



FlowGSpan Example

0-edge sub-graphs

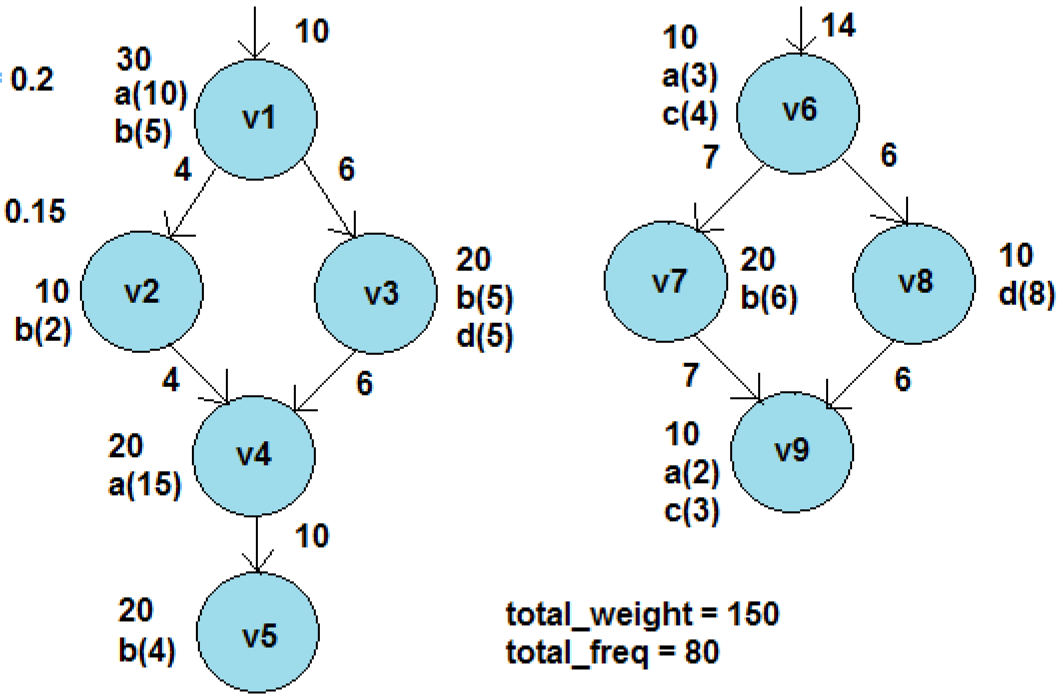


Sw = (10+15+3+2)/150 = 0.2
Sf = 0
Sm = max{0.2, 0} = 0.2

Sw = (5+2+5+4+6)/150 = 0.15
Sf = 0
Sm = 0.15

Sw = 0.04
Sf = 0
Sm = 0.04

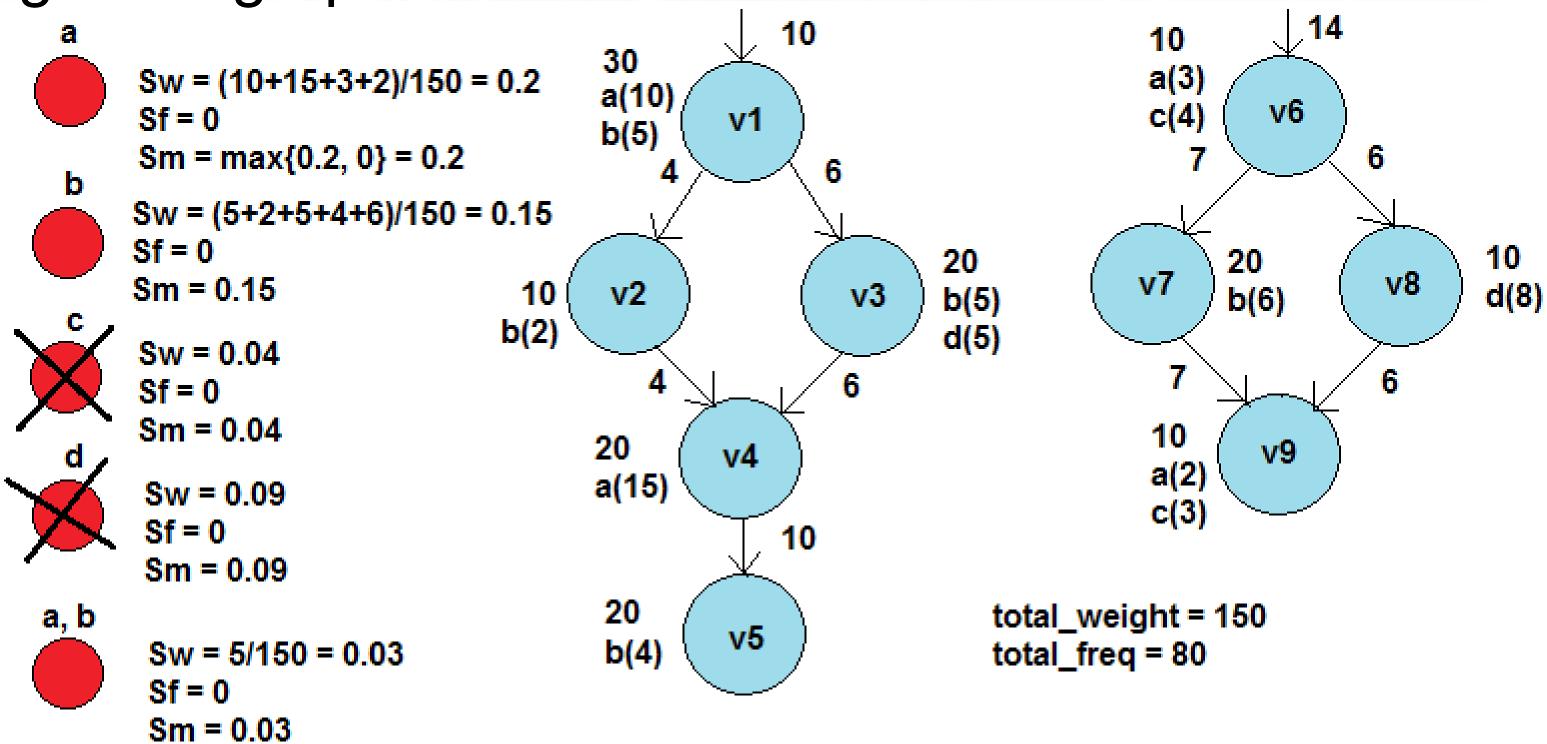
Sw = 0.09
Sf = 0
Sm = 0.09



total_weight = 150
total_freq = 80

FlowGSpan Example

0-edge sub-graphs



FlowGSpan Example

0-edge sub-graphs

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

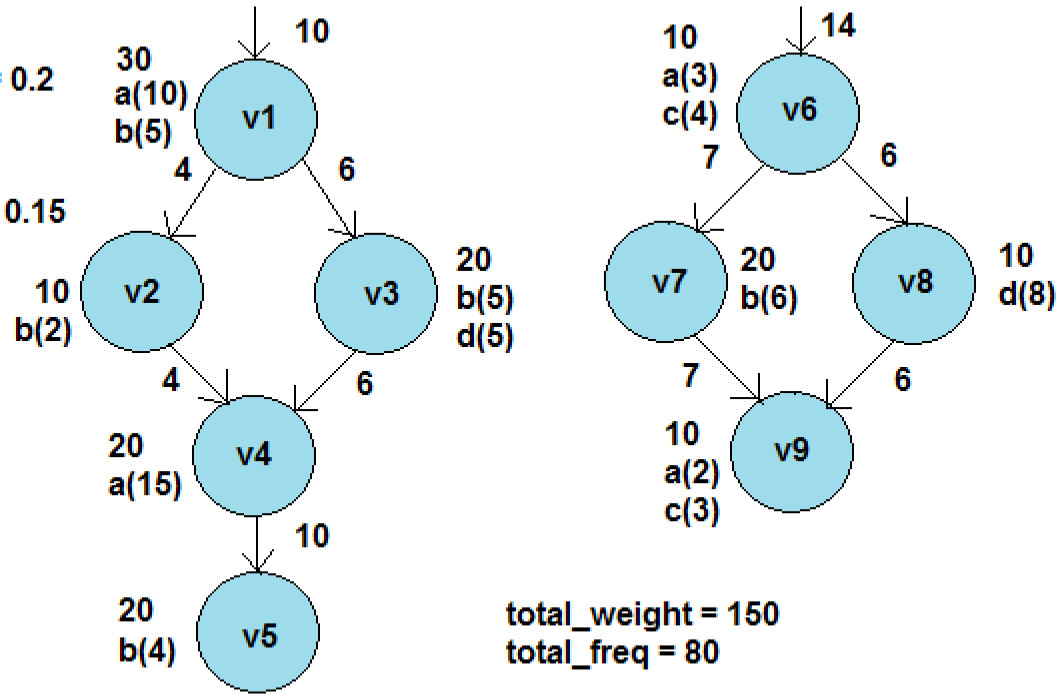
Sw = 0.04
Sf = 0
Sm = 0.04

d

Sw = 0.09
Sf = 0
Sm = 0.09

a, b

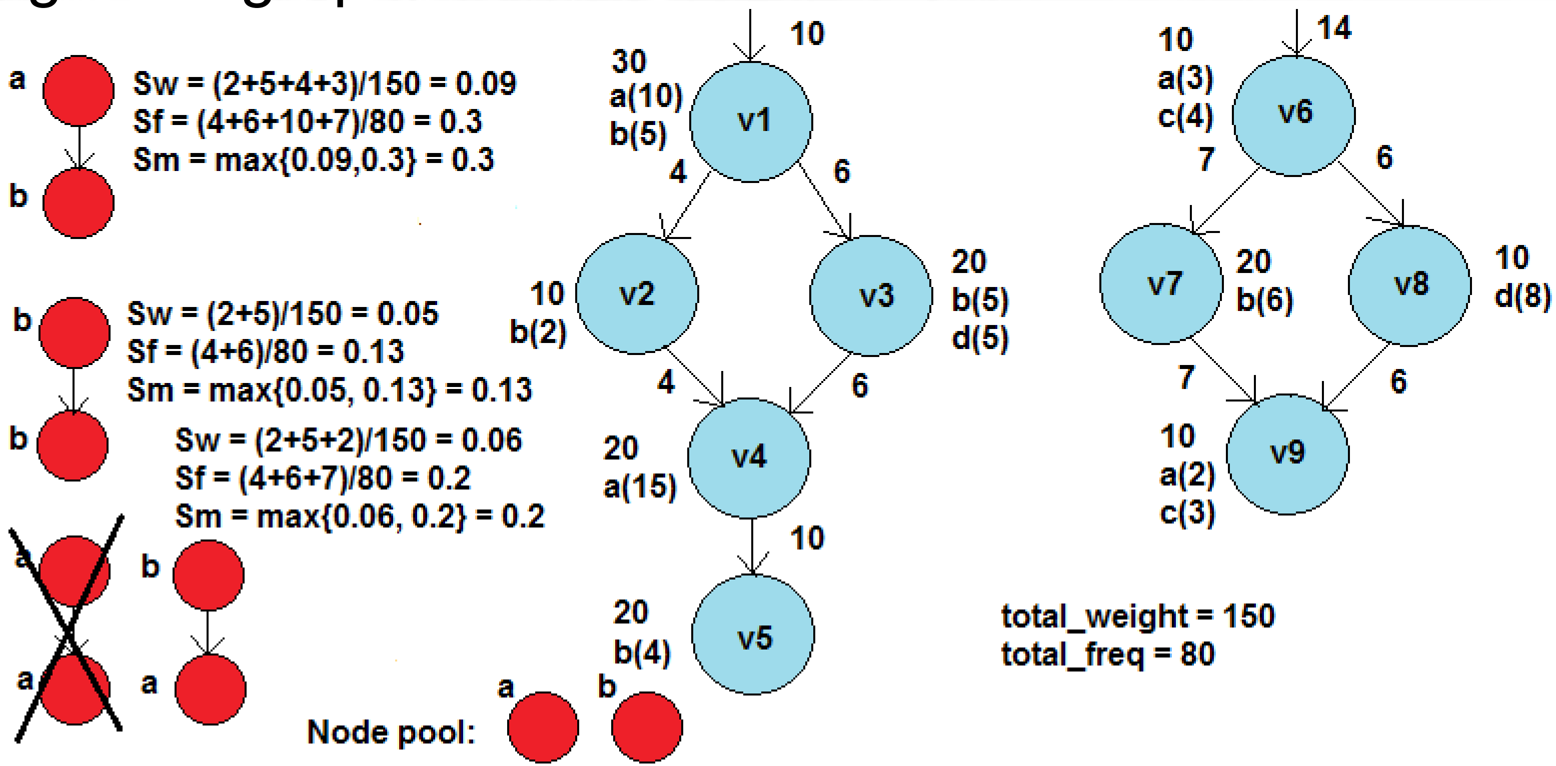
Sw = 5/150 = 0.03
Sf = 0
Sm = 0.03



total_weight = 150
total_freq = 80

FlowGSpan Example

1-edge sub-graphs

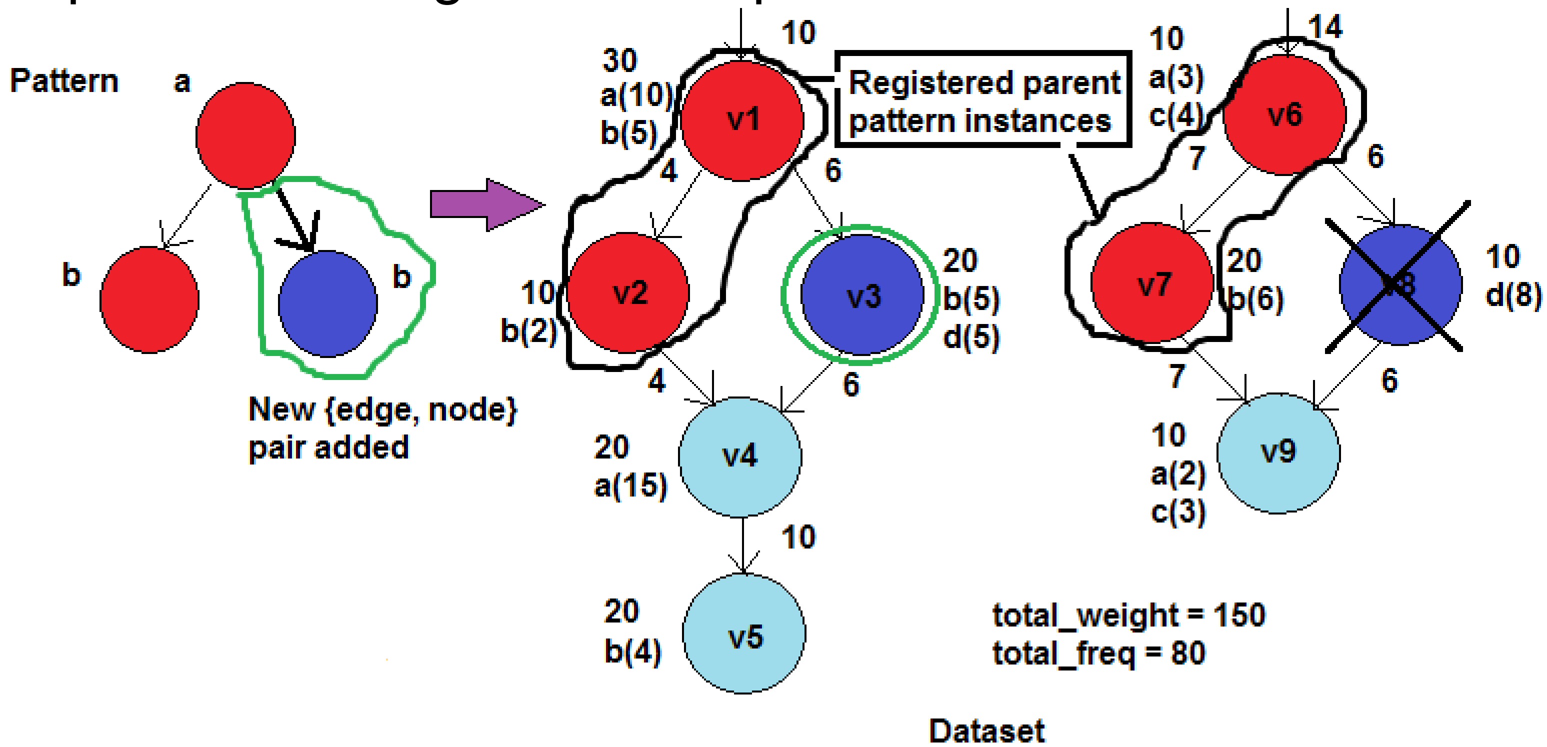


FlowGSpan Example

- 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

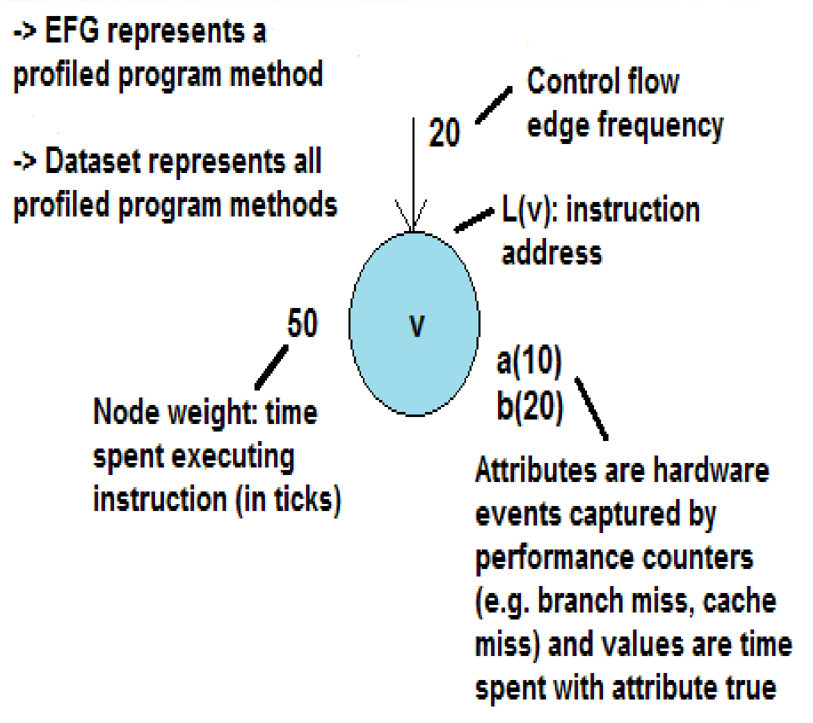
FlowGSpan Example

Core optimization: registration of pattern instances

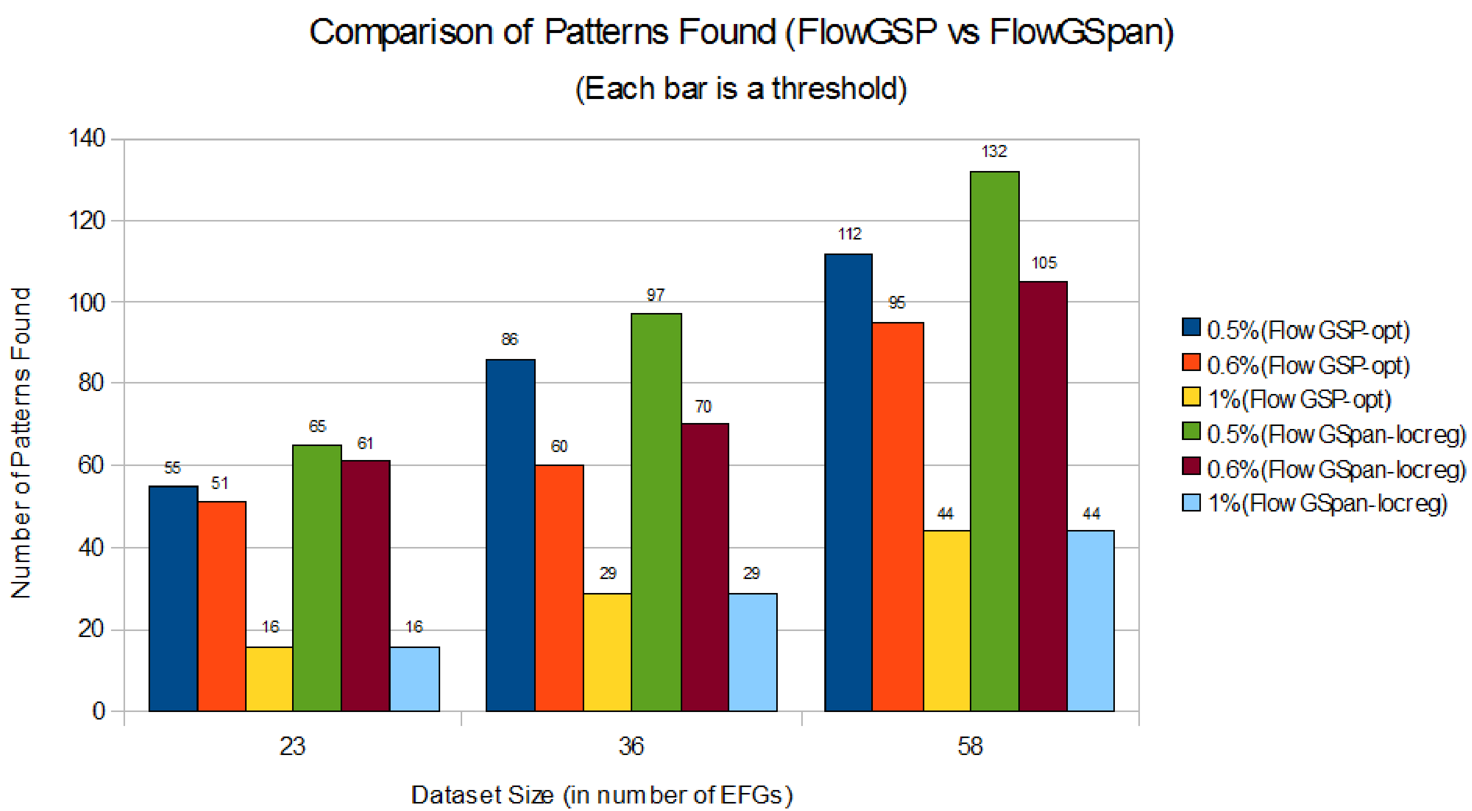


Application: targeting compiler developers

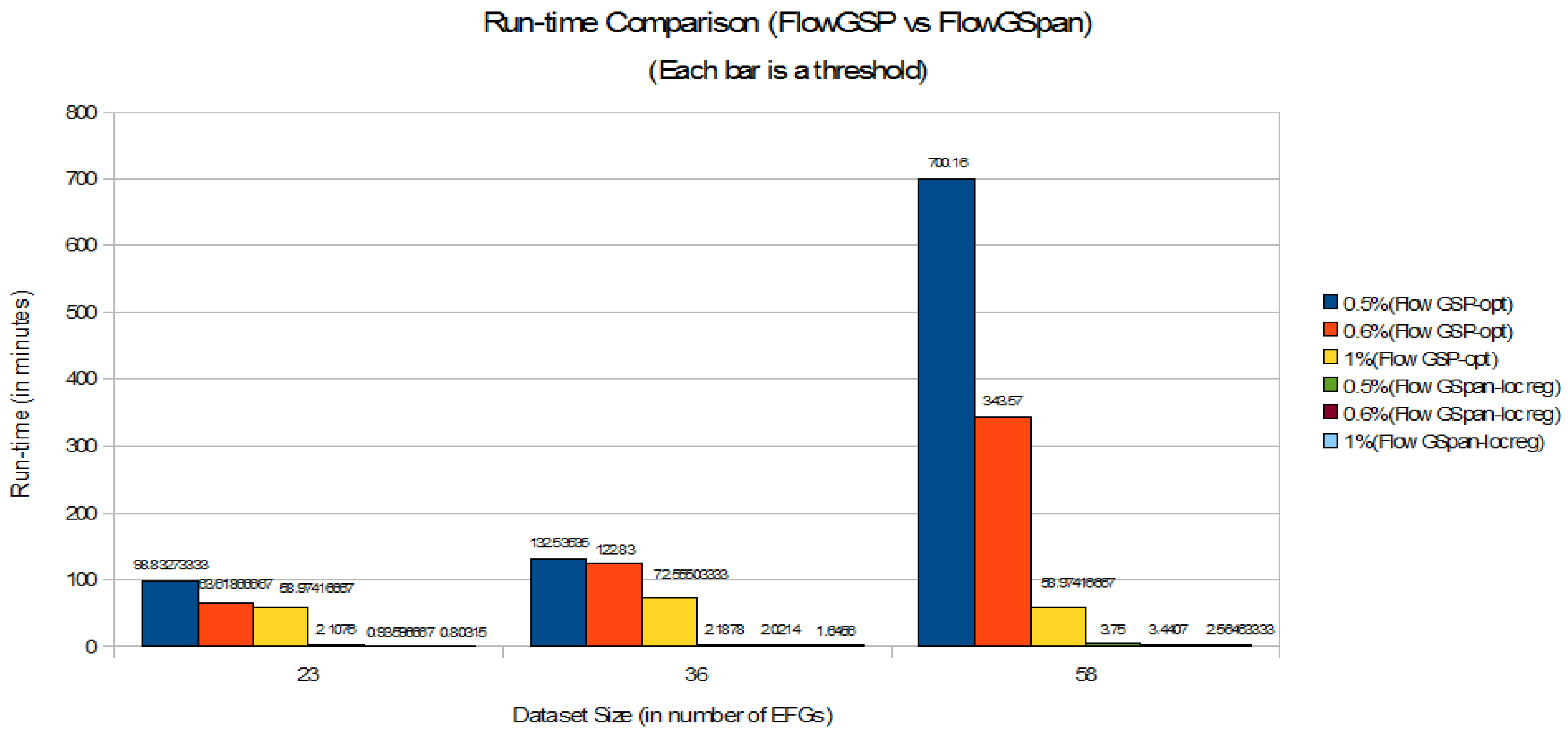
- Implemented FlowGSpan to mine for sets of hardware events
- Matching is exact
- Tested on DayTrader bechmark, which was JITted and profiled on IBM's z196 mainframe architecture
- Compared against optimized FlowGSP (with added pattern instance registration)



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?