

# An Extensible Perceptron Framework for Revision RTL Debug Automation

John Adler

Ryan Berryhill

Andreas Veneris



University of Toronto

# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion

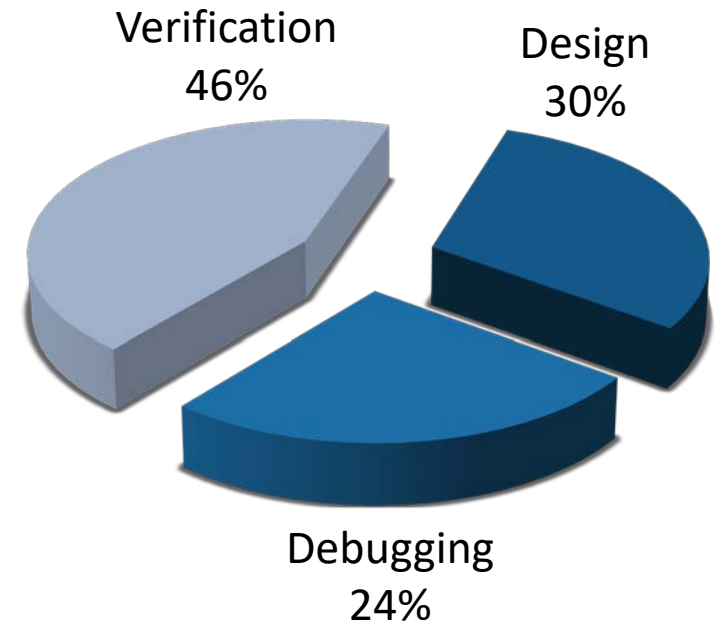
# Outline

- **Motivation**
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion

# Motivation

- Up to 70% of total design effort and cost is consumed by functional verification<sup>1</sup>
  - 37% of cost is spent on debugging: localizing and correcting design errors
- SAT-based automated debugging techniques can reduce time spent on this task

**Typical Design Cycle**



1: H.D. Foster, Trends in Functional Verification: A 2014 Industry Study

# Motivation

- **On-line verification**
  - Simulation, model checking, formal
  - Narrow subset of design functionality
- **Off-line verification**
  - Regression verification
  - Extensive test suites
  - Majority of design functionality
- **Problem**
  - On-line debugging: automated
  - Off-line debugging: largely manual process
    - Compounded by multiple simultaneous failures
    - Current automation techniques have limitations

# Motivation

- Software Configuration Management (SCM) stores information about the evolution of a design
  - Version Control Systems (VCSs) store individual changes to a design
  - Issue Tracking Systems (ITSs) store relationships between changes

## **Objective:**

Can we expedite analysis of debugging results using SCM data?

- **Contribution:** an extensible debug framework that uses SCM information to shorten the verification/debug cycle

# Outline

- Motivation
- **Preliminaries**
  - Software Configuration Management (SCM)
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion

# Outline

- Motivation
- Preliminaries
  - **Software Configuration Management (SCM)**
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion

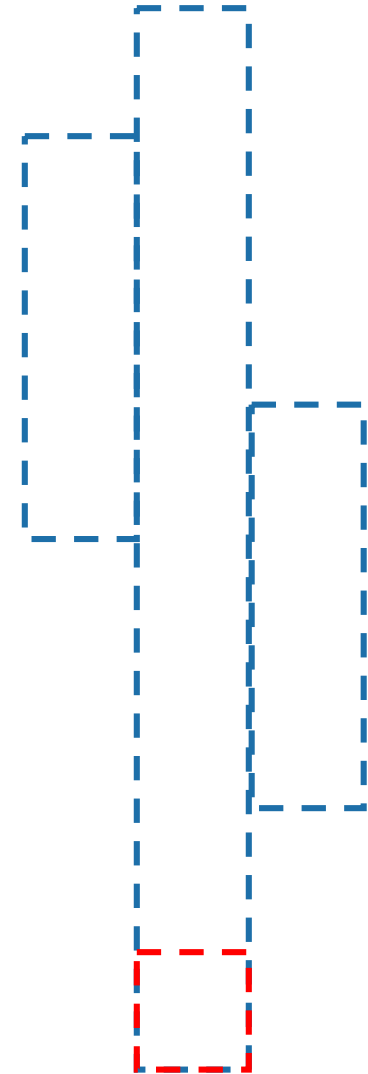


# Version Control Systems (VCS)

- **Version Control Systems (VCS)** are an important facet of an SCM flow
- Organize and track changes to design files
  - Eases sharing overhead
  - Allows multiple designers to work on a single project in tandem
- Examples: Git, Subversion, Mercurial, etc.
- Successive changes: **revisions** (commits)
  - Unique revision ID
  - Branch ID
  - Changes (diff) to files
  - Commit message, time, user

# Version Control Systems (VCS)

- Branches are used to isolate development on a single feature or bugfix
- Once development on a branch is complete, it is merged onto the mainline (master branch)
- Mainline
  - Unmerged branch
  - Merged branch
  - Head



# Issue Tracking Systems (ITS)

- **Issue Tracking Systems (ITS)** are another important facet of a modern SCM flow
- Allows branches to be associated with issues
- Issues can be tagged with additional human-oriented information
  - Is the branch a bugfix or a feature addition?
  - Related issues
  - Issue hierarchy (parent/child)
  - Resolved (closed), in progress (open)

# Outline

- Motivation
- Preliminaries
- **Clustering-based Revision Debug**
  - Suspect Clustering
  - Revision Ranking
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion

# Clustering-based Revision Debug

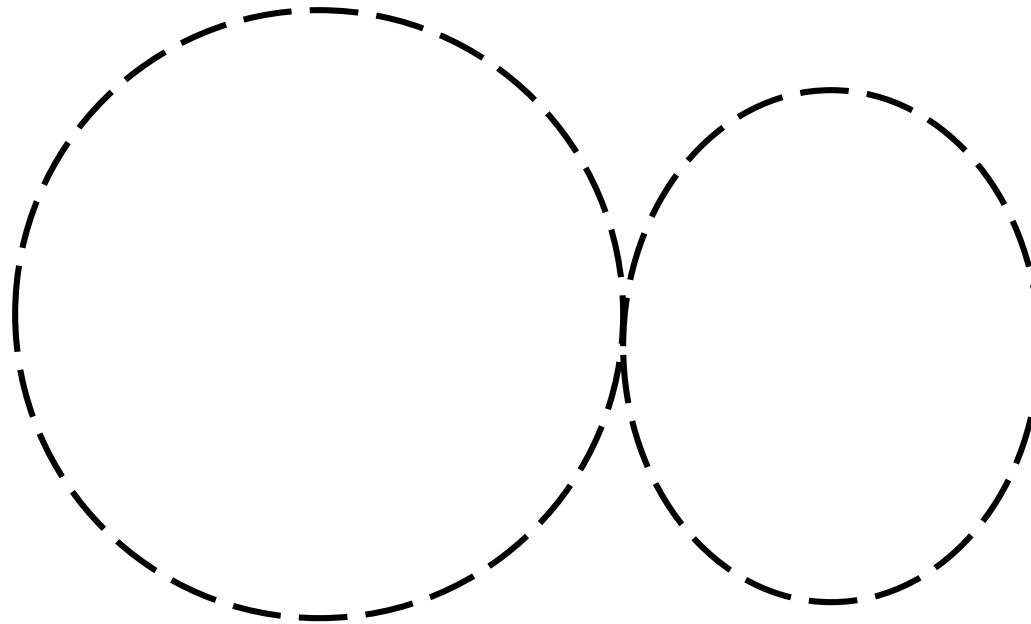
- Clustering-based Revision Debug in Regression Verification [Maksimovic ICCD '15]
  - Ranks revisions based on their likelihood of having introduced an error into the design
  - Decreases the expected number of results that need to be analyzed before locating the error
  - Can be extended to handle branches as well
- Two primary steps:
  1. Suspect clustering
  2. Revision ranking

# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
  - **Suspect Clustering**
    - Revision Ranking
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion

# Suspect Clustering

- **Goal:** estimate number of errors in design and how suspects relate to errors
- Affinity propagation clustering is used to automatically locate exemplars (cluster centers)



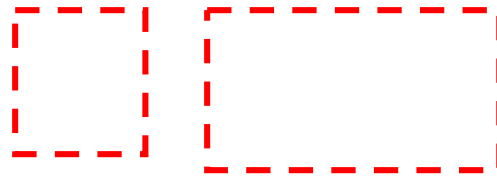
# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
  - Suspect Clustering
  - **Revision Ranking**
- Perceptron-based Revision Debug
- Experimental Results
- Conclusion



# Revision Ranking

- **Goal:** rank revisions based on likelihood of having introduced an error into the design
- Combine information from suspect clustering and revision classification into weight



- **Intuition:**
  - Revisions that are past bugfixes have higher weight
  - Revisions matching suspects closer to an exemplar have smaller weight

# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- **Perceptron-based Revision Debug**
  - Flattening Design History
  - Features and Training
  - Extensions
- Experimental Results
- Conclusion

# Perceptron-based Revision Debug

- **Goal:** predict probability that a revision has inserted an error
- Train a perceptron (binary classifier, supervised learning)
- Single-layered neural network can be implemented using Logistic Regression or Support Vector Machine

# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
  - **Flattening Design History**
    - Features and Training
    - Extensions
- Experimental Results
- Conclusion

# Flattening Design History

- **Goal:** transform complex branching revisions into a list suitable for input to a perceptron
- Model revisions and branches as a Directed Acyclic Graph (DAG)
  - Graph search can be used to identify branches
- Two flattening methods:
  1. Revision-to-revision
  2. Revision-to-head

# Revision-to-revision

- Revision-to-revision flattening
  - Diff for each revision
  - Extract changed lines



```
always @ (posedge clk)
begin
  if(rst)
    q <= 0;
  else
    q <= q;
end
```



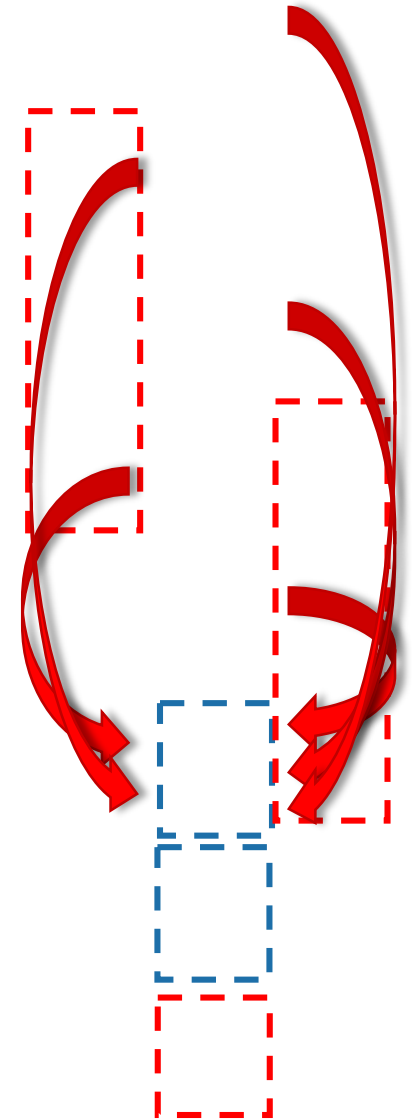
```
always @ (posedge clk)
begin
  if(rst)
    q <= 0;
  else
    q <= d;
end
```



```
always @ (posedge clk)
begin
  if(rst)
    q <= 0;
  else if(set)
    q <= 1;
  else
    q <= 1;
end
```

# Revision-to-head

- Revision-to-head flattening
    - **Intuition:** how a revision's changes affects the head
1. Prune redundant revisions
  2. Flatten
  3. For each revision, generate revision-to-head diff



# Head Selection

- *Heads* are failing revisions
  - The current failing revision is a head
  - Historically failing revisions can be used as heads for training
- Information on previous failures and their fixes is available in the VCS and ITS
  - As bugfixes are committed, new heads are opened to be selected



# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
  - Flattening Design History
  - **Features and Training**
  - Extensions
- Experimental Results
- Conclusion

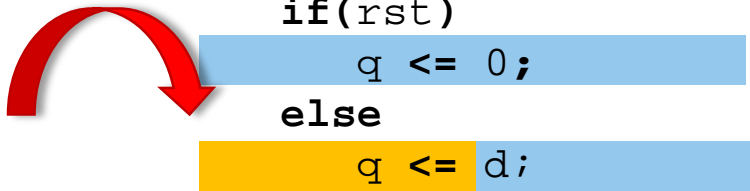
# Features

- For each revision used as a training sample:
  - Unique revision ID
  - Branch ID
  - Bugfix/feature flag
  - Set of matching values

# Features – Matching Values

- **Intuition:** match suspects with revisions
- For each head, run SAT-based debugging tool
- *Matching values* represent how much a given changed line matches with suspects

```
always @ (posedge clk)  always @ (posedge clk)
begin                   begin
    if(rst)              if(rst)
        q <= 0;          q <= 0;
    else                  else
        q <= q;          q <= d;
end                       end
```





# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
  - Flattening Design History
  - Features and Training
  - **Extensions**
- Experimental Results
- Conclusion

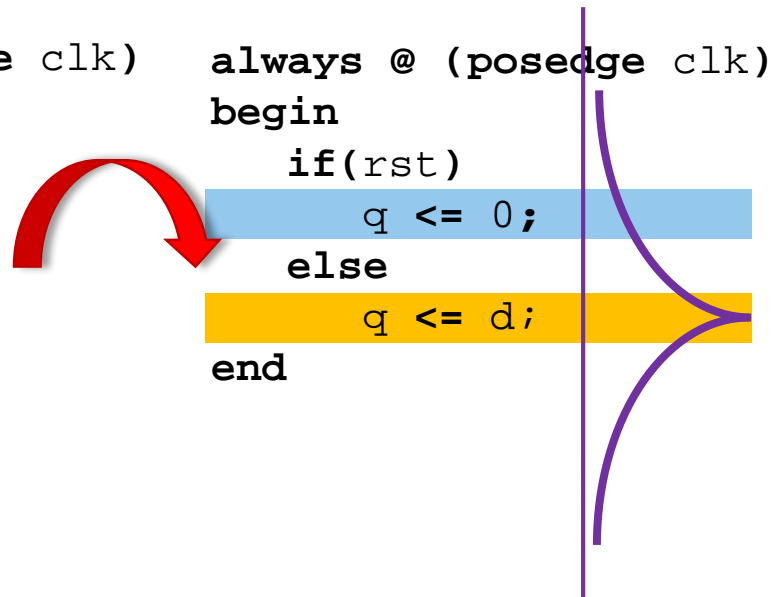
# Extensions – SVM

- Logistic Regression performs well when features are linearly separable
- A Support Vector Machine (SVM) can be used instead of Logistic Regression
  - Allows classification of features that are non-linearly separable
  - Requires additional tuning of hyperparameters when compared to Logistic Regression, but can be trained with the same feature set

# Extension – Weighted Distance

- Rather than exact matching, use exponentially decaying matching function between changed line and suspects

```
always @ (posedge clk)
begin
  if(rst)
    q <= 0;
  else
    q <= q;
end
```



# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
- **Experimental Results**
- Conclusion



# Experimental Results

- Run on an **Intel Core i5-3570K** workstation
  - Clocked at 3.40 GHz, 16 GB memory limit
- Backend SAT solver: **Minisat 2.2.0**
- Testcases include **OpenCores** designs and in-house industrial designs
- Perceptrons coded in Python

# Experimental Results

Test	Clustering	LR, r2r	LR, r2h	SVM, r2h
	Rank	Rank	Rank	Rank
ethernet	1	26	12	4
ethernet	1	10	11	7
ethernet	1	19	16	3
tate pair	4	12	8	8
SD card	4	11	15	9
SDRAM CTRL	1	16	10	25
6507 CPU	41	14	9	5
VGA	12	23	19	16
packet fwd	8	23	18	13

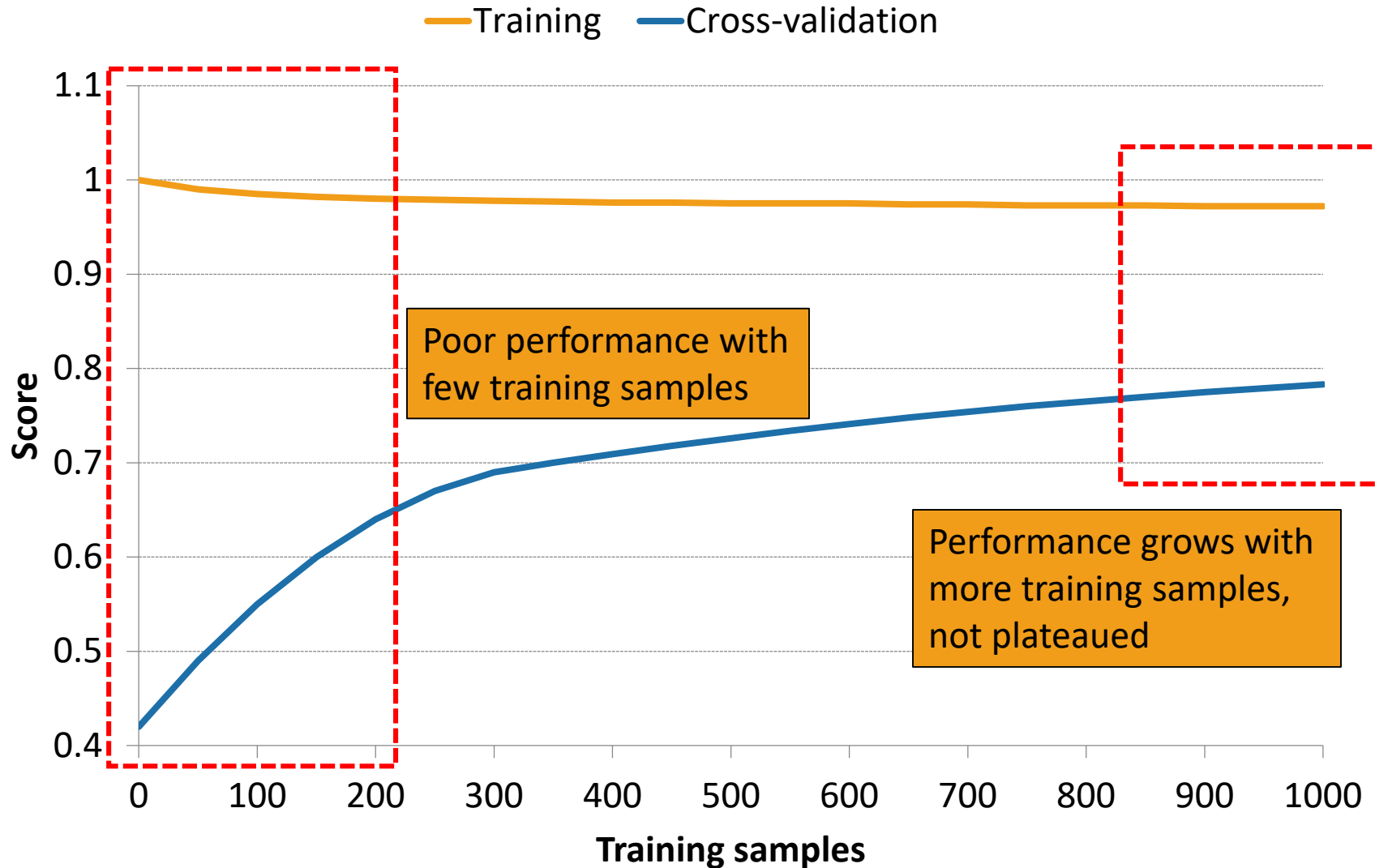
SVM almost universally performs better than LR...

r2h flattening usually gives better performance.

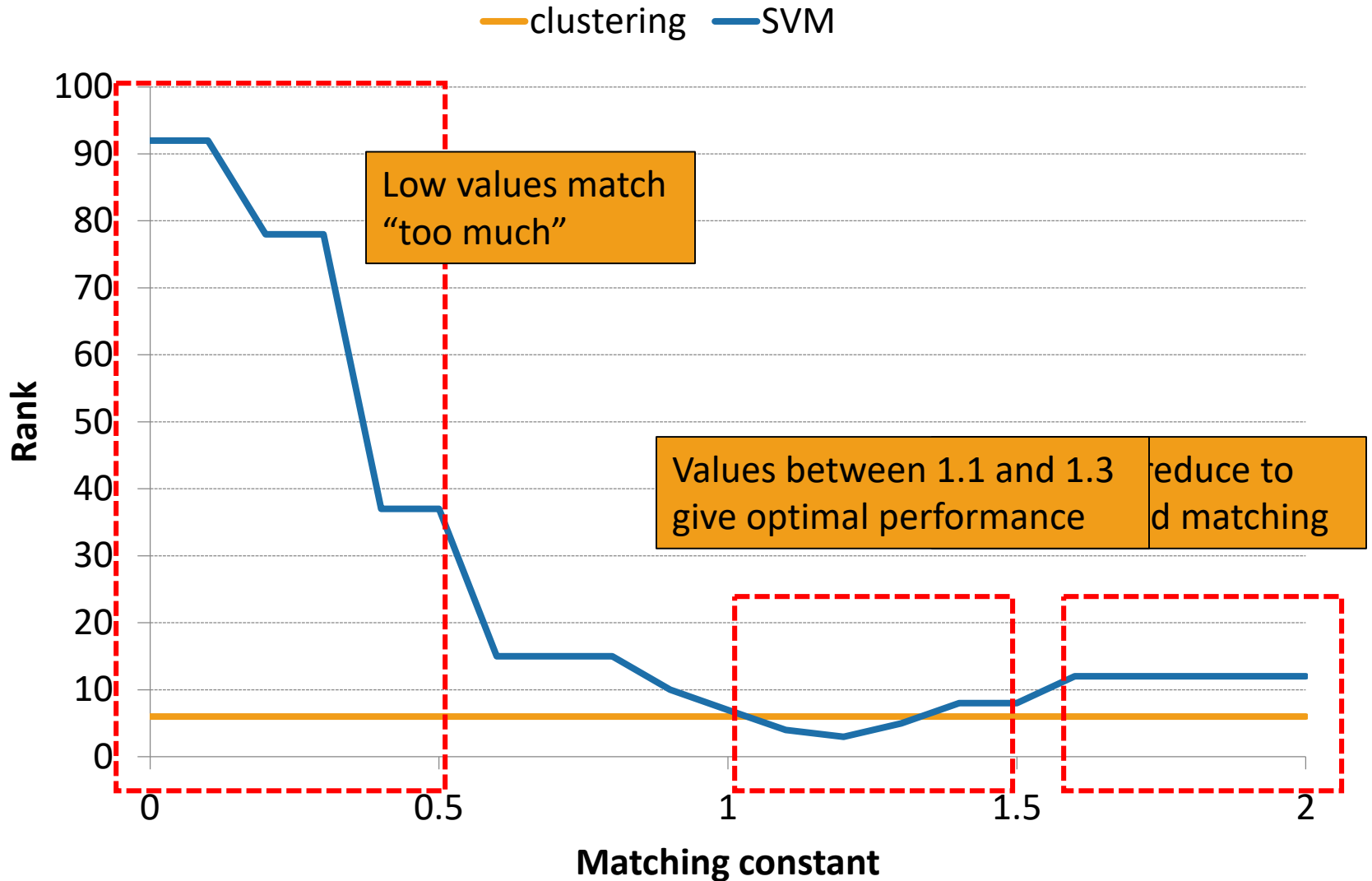
... but performs poorly with very few training samples.

SVM performance approaches clustering, especially with many training samples

# Experimental Results



# Experimental Results



# Outline

- Motivation
- Preliminaries
- Clustering-based Revision Debug
- Perceptron-based Revision Debug
- Experimental Results
- **Conclusion**

# Conclusion

- Perceptron-based revision debug
  - Orthogonal approach to clustering-based revision debug
  - Train perceptron on past failures and fixes
  - Perceptron predicts probability new revisions have inserted an error
  - Performance extensions: SVM, matching
- Future work
  - Train a more complex perceptron: multi-layered neural network
  - Extend features to include additional information available in SCM systems