TESTING A PLAGIARISM DETECTION TOOL

Konrad Mikalauskas
Alex Edan Garcia Sachez
Alina Hohensinger
Luca Thoms
Christine Zhao
Pankaj Singla

Sohum Bhatt

QHelp 2022, Leuven

INTRO TO PLAGIARISM

Existing approaches:

- Textual
- Lexical: token-based
- Syntactic: Abstract syntax trees
- Semantic: Program Dependence Graph

Table 1: Plagiarism detection tools for plain texts and program codes

System Name	Domain	Base Method	Usage	Cost
Plagiarism.org	Plain text	Fingerprint	On-line	Free
IntegriGuard	Plain text	Unknown	On-line	$4.95/Month
EVE2	Plain text	Unknown	Stand-alone	$19.99
CopyCatch	Student reports	Lexical matching	Stand-alone	Free
MatchDetectReveal	Plain text	Suffix Tree matching	DB Service	Free
SCAM	Digital library	Vector-space model	DB Service	Free
YAP3	Software	Greedy-String-Tiling	Stand-alone	Free
Clonechecker	Software	Unknown	Stand-alone	Commercial
MOSS	Software	Winnowing	Web service	Free
JPlag	Software	Greedy-String-Tiling	Web service	Free
SID	Software	Data Compression	Web service	Free
SIM	Software	Local Alignment	Stand-alone	Free
CodeMatch	Software	Fingerprint	Stand-alone	Commercial

(Lim et al., 2011)

BACKGROUND

- Increase in the usage of R
- 5% to 20% is plagiarized code
- More opportunities to copy existing assignments/codes
  - Without proper understanding

(Ullah et al., 2020)

METHODS

Materials:

- Plagiarism detection program in R
  - 'SimilaR' package
- Typical 'Intro to Statistics in R' assignment script
  - means and SDs, lm(), aov(), normality checks, etc.

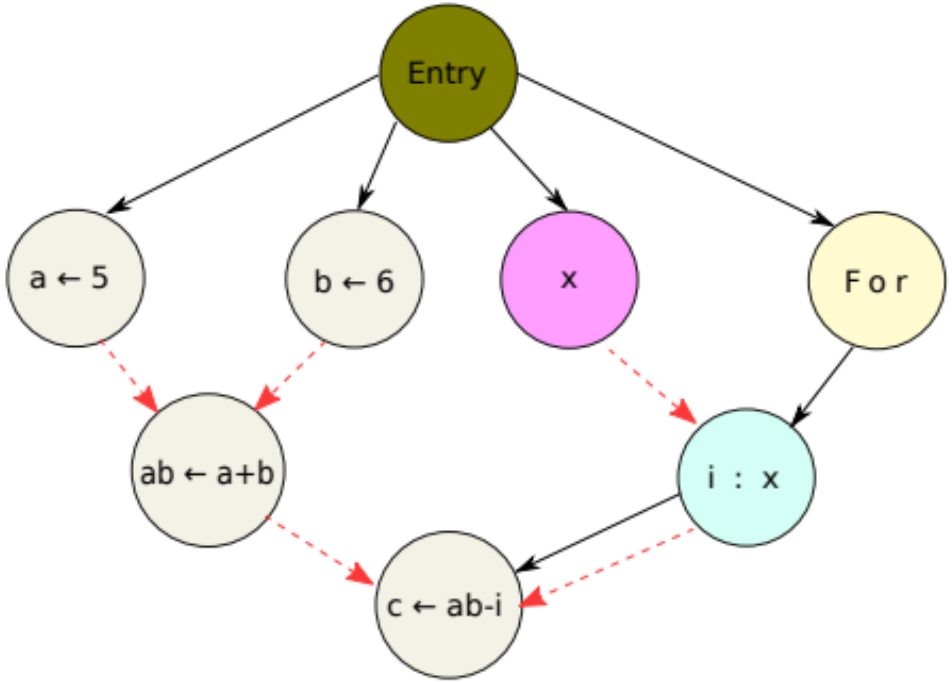
Goal:

- Plagiarize the assignment without getting caught
- Change as little as possible

PROGRAM DEPENDENCE GRAPH

Directed graph representing the relations between individual expressions in a chunk of source code.

```
sum <- function(x)
{
   a <- 5
   b <- 6
   for(i in x)
   {
     c <- a + b - i
   }
}
```



(Bartoszuk & Gagolewski, 2020)

PROGRAM DEPENDENCE GRAPH

Any algorithm comparing two Program Dependence Graphs needs to be:

- Flexible
  - Small differences shouldn't have significant impact on degree of estimated similarity
- Fast
  - Pairwise similarity comparisons need to be done quickly

(Bartoszuk & Gagolewski, 2020)

SimilaR ALGORITHM

- After h iterations, a vertex's label depends on the types of vertices whose distance from it is at most h.
- Equivalent labels are assigned to vertices similar to each other.
- Vertex importance – Derived from the number of vertices that depend on a given node.
  Neighbouring vertices are assigned different labels only if the differences cross a certain threshold of importance.
- Final measure of similarity calculated by comparing two vectors (one per graph) of label type counts with each other.

(Bartoszuk & Gagolewski, 2020)

RESULTS

What doesn't work:

- Deleting/adding whitespace, dead code, comments, libraries, etc.
- Changing variable names, chunk order
- Pipes instead of function(function(function(…
- apply() instead of for-loops

RESULTS

What works:

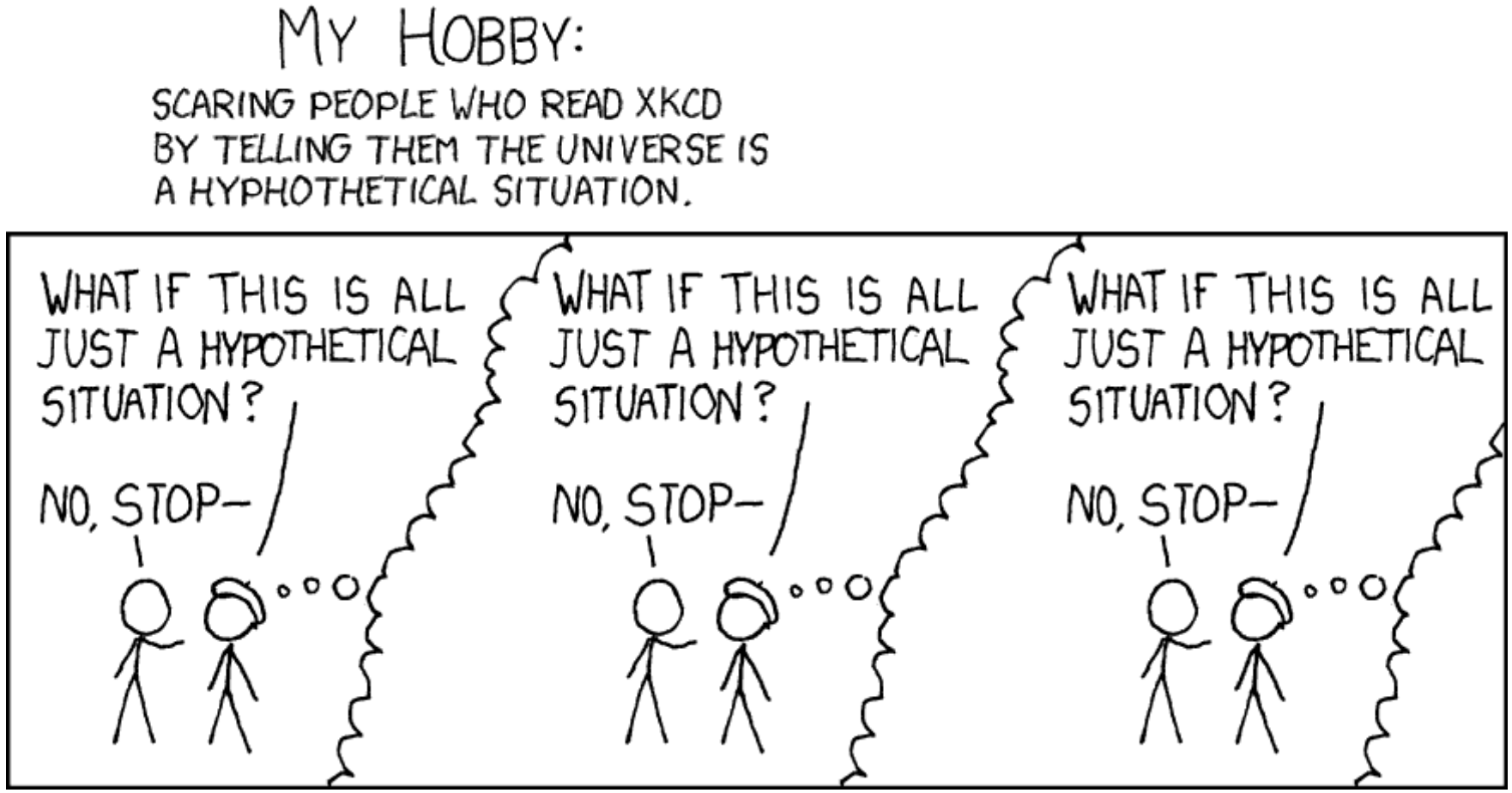
- Wrapping code in useless if-statements (i.e., if(TRUE == TRUE){…})
- Using resid() on 'lm' objects instead of lm$residuals
- Rewriting linear models with interactions in full
- instead of 'y ~ x1*x2', do 'y ~ x1 + x2 + x1:x2'
- Build functions to aggregate code that give one output (e.g: plots)

RESULTS - CHEATING

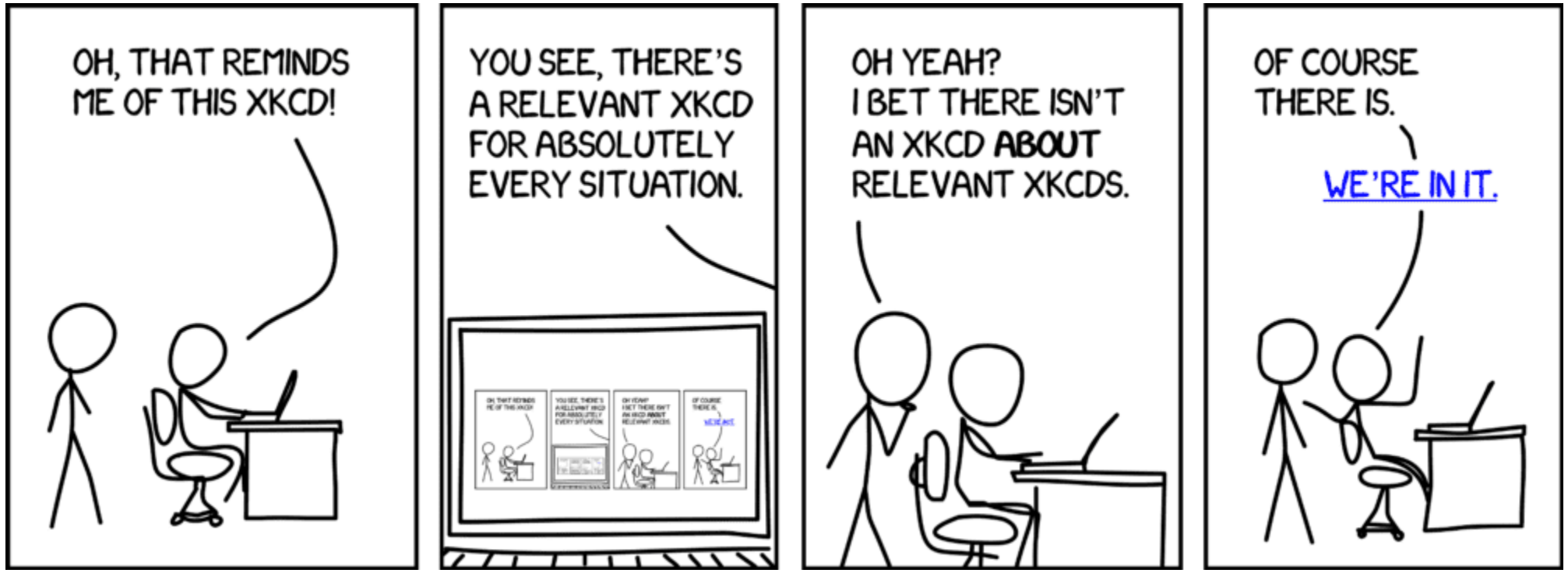
```
x                       1 obs. of 4 variables
    $ name1    : chr "plagiarized.R   plagiarized.R"
    $ name2    : chr "tocopy.R   tocopy.R"
    $ SimilaR : num 0.647
    $ decision: num 0
y                       1 obs. of 4 variables
    $ name1    : chr "plagiarized.R   plagiarized.R"
    $ name2    : chr "tocopy.R   tocopy.R"
    $ SimilaR : num 0.647
    $ decision: num 0
z                       1 obs. of 5 variables
    $ name1     : chr "plagiarized.R   plagiarized.R"
    $ name2     : chr "tocopy.R   tocopy.R"
    $ SimilaR12: num 0.628
    $ SimilaR21: num 0.667
    $ decision : num 0
```

Google “Recursion”.
(Do it!)

RECURSION


MY HOBBY:
SCARING PEOPLE WHO READ XKCD BY TELLING THEM THE UNIVERSE IS A HYPHOTHETICAL SITUATION.
WHAT IF THIS IS ALL JUST A HYPOTHETICAL SITUATION?
NO, STOP—
WHAT IF THIS IS ALL JUST A HYPOTHETICAL SITUATION?
NO, STOP—
WHAT IF THIS IS ALL JUST A HYPOTHETICAL SITUATION?
NO, STOP—

RECURSION


OH, THAT REMINDS ME OF THIS XKCD!
YOU SEE, THERE'S A RELEVANT XKCD FOR ABSOLUTELY EVERY SITUATION.
OH YEAH? I BET THERE ISN'T AN XKCD ABOUT RELEVANT XKCDS.
OF COURSE THERE IS.
WE'RE IN IT.

ITERATION vs RECURSION

```r
iterative <- function(n) {
  for (x in 1:n) {
    print(paste("Let's print x, which is = ", x))
  }
}
iterative(5)
```

```
> iterative(5)
[1] "Let's print x, which is =  1"
[1] "Let's print x, which is =  2"
[1] "Let's print x, which is =  3"
[1] "Let's print x, which is =  4"
[1] "Let's print x, which is =  5"
```

```r
recursive <- function(n) {
  print(paste("Hey there! I'm a recursive call with n = ", n))
  if (n == 0) {
    return()          # Base case
  }
  recursive(n-1)      # Recursive call
  print(paste("Finally, let's print n, which is = ", n))
}
recursive(5)
```

```
> recursive(5)
[1] "Hey there! I'm a recursive call with n =  5"
[1] "Hey there! I'm a recursive call with n =  4"
[1] "Hey there! I'm a recursive call with n =  3"
[1] "Hey there! I'm a recursive call with n =  2"
[1] "Hey there! I'm a recursive call with n =  1"
[1] "Hey there! I'm a recursive call with n =  0"
[1] "Finally, let's print n, which is =  1"
[1] "Finally, let's print n, which is =  2"
[1] "Finally, let's print n, which is =  3"
[1] "Finally, let's print n, which is =  4"
[1] "Finally, let's print n, which is =  5"
```

ITERATION vs RECURSION

```r
iterative <- function(n) {
  for (x in 1:n) {
    print(x)
  }
}
```

```
> iterative(10)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```r
recursive <- function(n) {
  if (n == 0) {
    return()
  }
  recursive(n-1)
  print(n)
}
```

```
> recursive(10)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
> # SimilaR -> 0.5729167
```

ITERATION vs RECURSION

```r
iterative <- function(n) {
  for (x in 1:n) {
    print(x)
  }
}
```

```
> iterative(10)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```r
recursive <- function(n) {
  if (n == 0) {
    return()
  }
  recursive(n-1)
}
recursive(10)
```

```
> recursive(10)
NULL
```

```
> # SimilaR -> 0.5555556
```

CONCLUSION

When to use it?

- Moderate data analyses (e.g., thesis)
- Not too simple / Not sensible to use with complex scripts
- NOT Intro courses (Beginners likely to stick to known/taught methods)

How should it be used

- Not as a stand-alone checker but as an "assistant" tool

Limitations

- Binary summary

Possible improvements

- Chunk by chunk
- Hybrid checking
- Printing the Program Dependence Graph

REFERENCES

Bartoszuk, M., & Gagolewski, M. (2020). SimilaR: R Code Clone and Plagiarism Detection. The R Journal, 12(1), 367–385.

Lim, J., Ji, J.-H., Cho, H.-G., & Woo, G. (2011). Plagiarism detection among source codes using adaptive local alignment of keywords. Proceedings of the 5th International conference on Ubiquitous Information Management and Communication, 1–10. https://doi.org/https://doi.org/10.1145/1968613.1968643

Ullah, F., Wang, J., Farhan, M., Jabbar, S., Wu, Z., & Khalid, S. (2020). Plagiarism detection in students' programming assignments based on semantics: multimedia e-learning based smart assessment methodology. Multimedia Tools and Applications, 79, 8581–8598. https://doi.org/10.1007/s11042-018-5827-6