Application of Information Retrieval Techniques for Source Code Authorship Attribution

Steven Burrows, Alexandra L. Uitdenbogerd, and Andrew Turpin

School of Computer Science and Information Technology
RMIT University
GPO Box 2476V, Melbourne 3001, Australia
{steven.burrows,alexandra.uitdenbogerd,andrew.turpin}@rmit.edu.au

Abstract. Authorship attribution assigns works of contentious authorship to their rightful owners solving cases of theft, plagiarism and authorship disputes in academia and industry. In this paper we investigate the application of information retrieval techniques to attribution of authorship of C source code. In particular, we explore novel methods for converting C code into documents suitable for retrieval systems, experimenting with 1,597 student programming assignments. We investigate several possible program derivations, partition attribution results by original program length to measure effectiveness of modest and lengthy programs separately, and evaluate three different methods for interpreting document rankings as authorship attribution. The best of our methods achieves an average of 76.78% classification accuracy for a one-in-ten classification problem which is competitive against six existing baselines. The techniques that we present can be the basis of practical software to support source code authorship investigations.

Keywords: Adversarial information retrieval, authorship attribution, source code.

1 Introduction

Automatically detecting the author of a document is useful in plagiarism detection, copyright infringement, computer crime and authorship disputes. To assign authorship, a profile of each author is constructed from known sources, and then documents of unknown or uncertain authorship can be matched against these profiles manually, or with computer-based statistical analysis, machine learning or similarity calculation methods [1].

Authorship attribution techniques can be used to solve real-world natural language (plain text) problems in literature, papers, articles, essays and reports; but it can also solve problems in the domain of *structured text* such as source code. For example, will further analysis of the malicious WANK (Worms Against Nuclear Killers) and OILZ worms confirm that three authors were indeed involved [2]? Moreover, what can be learned about the author of the unnamed Internet worm of 1989 [3]? And finally, how can a lecturer know if programming assignments have indeed been written by the students themselves, obtained from a source code search engine such as Codase,[1] or outsourced to an anonymous programmer at Rent A Coder?[2]

In contrast to the large body of work on natural language authorship attribution (see for example Zhao and Zobel [4] for a comparison of many methods), we focus on authorship attribution for structured text; specifically C source code. Various source code authorship attribution techniques have been proposed, but they all disagree on what should comprise an author profile; use limited collections; and lack comparison to multiple baselines. We address all of these shortcomings by thoroughly investigating a wide range of features, employing a large data set, and comparing our work to the published baselines of six other research groups.

We use a collection of 1,597 C programs written by the 100 most prolific authors of C programming assignments in the School of Computer Science and Information Technology at RMIT University over an eight year period. Anonymous versions are used which are then tokenised and converted to n-gram representations which form document surrogates that are indexed and queried using standard information retrieval techniques [5,6]. We treat our experiments as being a 10-class authorship attribution problem. That is, we index a random subset of ten authors from the collection, and then attempt to attribute authorship for all documents in the subset to one of the ten authors.

In Section 2 we review methods for solving authorship attribution problems, and discuss approaches that have been applied to source code. Section 3 describes the baseline work that forms the starting point for this paper. In Section 4 we outline our methodology and present results in Section 5. We conclude in Section 6 and offer avenues for future work.

2 Previous Work

In this section we describe popular authorship attribution methods which fall within four broad categories. We then summarise our previous work, and the baselines to which we compare our new work.

2.1 Authorship Attribution Methods

Frantzeskou et al. [1] describe a taxonomy of four authorship analysis methodology categories: manual inspection, statistical analysis, machine learning and similarity measurement which we discuss in turn.

Manual inspection involves analysis by an expert to draw conclusions about coding skill or motive. Such evaluations can be used for authorship disputes in courts of law. Authorship disputes can arise because "programmers tend to feel a sense of ownership of their programs" [7] which can lead to code reproduction in successive organisations. This approach is not scalable for large problems.

[1] `http://www.codase.com`
[2] `http://www.rentacoder.com`

Statistical analysis techniques can be used to find effective feature sets for authorship analysis by eliminating features that make insignificant contributions towards identifying authorship compared to others. Discriminant analysis is one such statistical analysis technique which Ding and Samadzadeh [8] use for authorship attribution feature selection to create Java code fingerprints.

Machine learning involves finding patterns in samples of work belonging to one or more authors, and then classifying unseen works as belonging to the most likely author based upon the patterns in the training data. Many forms of machine learning algorithms have been applied to authorship attribution including support vector machines, decision trees, case-based reasoning and neural networks [9,10].

Similarity measurement involves measuring the distance between query documents and documents belonging to candidate authors by means of a similarity function. Nearest-neighbour methods [11] have been applied here to calculate the distance between feature lists of documents. One implementation is the Euclidean distance metric which computes the distance between two vectors in n-dimensional space.

2.2 Authorship Attribution Implementations

This paper continues from our previous work [12] where we investigated how n-gram representations [13] using modest numbers for n can be used to detect small repeating patterns of source code to indicate style. Scalability considerations were explored next by increasing the 10-class authorship attribution problem in increments up to 100-class to observe how severely the effectiveness of our approach degrades for larger problems. Next we further explored scalability considerations by reducing the number of submissions per author to simulate a scenario of having limited training data. Finally, we compared our work to a baseline implementation [14] and found our work to be more effective by 39% when measured with mean reciprocal rank.

We have identified works by six other research groups for source code authorship attribution which we treat as baseline implementations for our new contributions. These are the works by Ding and Samadzadeh [8] (canonical discriminant analysis), Elenbogen and Seliya [15] (C4.5 decision trees), Frantzeskou et al. [16] (nearest neighbour measurement), Krsul and Spafford [17] (discriminant analysis), Lange and Mancoridis [18] (nearest neighbour measurement) and MacDonell et al. [10] (case-based reasoning, discriminant analysis and neural networks). This list conveniently includes representative classification methods from all automated authorship attribution categories as described in Section 2.1 [1]. In addition to varying classification methods, these approaches also vary in terms of number of authors and work samples in each experiment, average program length, programming language, and level of programming experience of the authors whom contributed the work samples. We reserve a detailed comparison of the baseline systems to our own work in Section 5.3 where we explain the impact of these factors on reported classification accuracy scores.

3 Baseline

In this section, we describe the C program collection used as data in our experiments, and summarise some of our previous work [12] which forms the baseline for our new contributions.

3.1 Collection Creation

Our collection is comprised of 1,597 C programming assignments from the School of Computer Science and Information Technology at RMIT University. We refer to these assignments as Collection-A. These assignments represent works from the 100 most prolific authors of C programming assignments in our school from 1999 to 2006 with fourteen to twenty-six submissions per author. No discrimination is made between undergraduate assignments, postgraduate assignments and year level — there is a mixture of assignments from all levels of ability. Collection-A contains near-empty submissions of one line up to 10,789 lines of code, with the mean length being 830 lines of code. We note that the distribution of submission lengths is skewed towards shorter submissions, as we have a median of 650 lines of code.

When constructing Collection-A, all submissions were made anonymous to comply with the ethical requirements of our university's Human Research Ethics Committee. This meant renaming files and removing all source code comments and output strings. We additionally deleted byte-identical copies resulting from duplicate submissions.

3.2 N-Gram Results

In our previous work, we used standard information retrieval methods to conduct 10-class authorship attribution experiments. Each C program was reduced to a "document" that contained n-grams comprised of operators and keywords only. In Section 4.1 we consider other types of features. We experimented with 1-gram to 90-gram representations [13] of these features to see if small frequent patterns are good markers of authorship.

In each 10-class experiment, documents belonging to ten random authors comprised a "collection" (about 160 documents) and each document in the collection was used as a "query" against this collection in turn. The Zettair search engine,[3] with its various ranking schemes, returned a ranked list for each query. The reciprocal rank (RR) of the highest ranked document by the same author as the query was used as the main outcome metric. For example, if the highest ranked document by the query author is 3, then RR is 0.33; or if the rank is 5, the RR is 0.2. We also used the Average Precision (AP) of the ranked list as an outcome metric, where AP is defined as "taking the set of ranks at which the relevant documents occur, calculating the precision at those depths in the

[3] `http://www.seg.rmit.edu.au/zettair`

ranking, and then averaging the set of precision values so obtained" [19]. As is typical in information retrieval, we report the mean of these two measures over many queries: mean reciprocal rank (MRR) and mean average precision (MAP). The random sampling of ten authors was repeated 100 times and we tested five different ranking schemes: Cosine [6], Pivoted Cosine [20], Dirichlet [21], Okapi BM25 [22,23], and our own scheme called Author1 [12].

The above process allowed us to compare several n-gram sizes with each of the five similarity measures and results are presented in Table 1. Note that the results differ to those reported previously [12] as byte-identical duplicates have now been removed from Collection-A. We found that the Okapi similarity measure combined with 8-grams was most effective when measured in MRR (76.39%) and Okapi with 6-grams was most effective when measured in MAP (26.70%). These results are highlighted in Table 1.

We also examined the statistical significance of differences in MRR and MAP at the 95% confidence level using a permutation test (the distribution of RR scores is very skewed towards 1.00). We chose the null hypothesis to be "no difference from the most effective MRR or MAP result", and tested 2-gram up to 14-gram results for all five similarity measures. Two MRR results were found to be not significant from Okapi with 8-grams — Pivoted Cosine with 8-grams (75.89%; $p = 0.24$) and Okapi with 6-grams (75.59%; $p = 0.06$). These results are also marked on Table 1. Also, the most effective MAP result (Okapi with 6-grams) was found to be statistically significant compared to all other tested MAP results ($p < 0.05$). We carried forward Okapi with 6-grams over 8-grams for all future experiments for this reason and also because these representations take up less space.

3.3 Scalability Considerations

The previous experiment was for a 10-class problem, so we next tested larger problems up to the 100-class problem. With the problem size ten times bigger than our previous experiment, we were pleased to observe that MRR using the Okapi 6-gram system only fell from 75.59% to 64.97% (10.62%), which shows promise for dealing with even larger problems.

We also explored reducing the number of submissions per author. Beginning with our average of sixteen submissions per author, we randomly removed submissions in fixed amounts until we had only two submissions per author. In this case we found that MRR for the Okapi 6-gram system fell from 75.59% to 31.53% (44.06%). These results indicate that an information retrieval approach to attributing authorship for C code may not work well when there is a limited number of samples per author. However, we speculate that removing submissions chronologically instead of randomly may have less impact on the MRR score, as this simulates a more real-life scenario. We were not able to do this in Collection-A as the school submission archive is messy in parts, and we don't have confidence in the timestamps. Experiments using reliable timestamps is an avenue for future work.

Table 1. Effect of varying the n-gram size and similarity measure for a 10-class authorship attribution problem using sixteen work samples per author on average, with authors randomly selected from COLLECTION-A. Results are averaged using all documents as queries (about 160 queries) in 100 random subsets of ten authors; a total of approximately 16,000 runs. The most effective results are highlighted — Okapi with 8-grams for MRR and Okapi with 6-grams for MAP. Two further MRR results are highlighted that were not statistically significant from Okapi with 8-grams at the 95% confidence level.

Grm	Similarity Measure MRR%					Similarity Measure MAP%				
Sze	Au1	Cos	Dir	Oka	P.Co	Au1	Cos	Dir	Oka	P.Co
1	51.53	59.41	23.36	42.66	28.49	17.05	19.85	12.22	17.72	13.31
2	65.80	68.44	28.33	67.34	53.33	20.73	22.50	12.98	23.24	18.25
4	72.00	74.10	53.43	75.52	72.79	23.91	25.10	19.33	26.00	23.70
6	73.85	74.42	59.42	**75.59**	74.70	25.71	25.82	20.44	**26.70**	25.52
8	75.49	74.94	61.17	**76.39**	**75.89**	24.96	24.65	19.58	25.61	25.00
10	73.72	73.35	60.44	74.95	74.69	22.78	22.73	17.76	23.47	23.25
12	74.17	73.45	61.39	74.95	74.55	21.57	21.42	16.95	22.01	21.75
14	72.77	72.20	62.41	73.51	73.32	19.09	18.93	15.74	19.38	19.28
16	71.63	71.12	63.55	72.20	72.25	16.81	16.71	14.73	16.98	16.97
18	70.41	70.14	64.21	70.81	70.86	14.59	14.65	13.31	14.79	14.81
20	67.96	67.92	64.48	68.27	68.33	13.36	13.41	12.66	13.53	13.54

4 Methodology

In this section, we outline the methodology for the new contributions. We first describe our approach for evaluating effective feature sets. Next we explore the effect of varying program length and the implications of particularly small queries, where a query is a program of undetermined authorship. Following that we describe an experiment to simulate a real-life authorship classification problem.

4.1 Feature Selection

Much work is available that defines good programming style. For example, Cannon et al. [24] define coding style guidelines such as commenting, white space, declarations, naming conventions and file organisation. Oman and Cook [25] collate programming style guidelines from many sources and organise them in a taxonomy which we use as a basis for categorising style features in our work. The taxonomy contains three main categories. *Typographic style* refers to all aspects of code that do not affect program execution such as commenting, naming characteristics and layout (spaces, tabs and new lines); next, *control structure style* refers to flow control tokens that affect algorithm implementation decisions such as operators, keywords and standard library functions; finally, *information structure style* refers to the organisation of program memory, input and output such as data structures, and input/output functions such as `printf` and `scanf`.

Based upon the above taxonomy, we create six classes of features for experimentation. White space, operators, literals, keywords, I/O words (from the

Table 2. Number of unique features in each feature class, and the distribution of tokens in Collection-A

	W. Space	Operator	Literal	Keywd	I/O	Func	Total
Features	4	39	5	32	60	185	325
Percent	1.23	12.00	1.54	9.85	18.46	56.92	100.00
Tokens	8,109,257	1,495,730	1,409,749	384,718	143,691	76,355	11,619,500
Percent	69.79	12.87	12.13	3.31	1.24	0.66	100.00

`stdio.h` ANSI C89 header file) and function words (all standard library words not in `stdio.h`). We omit comments for de-identification reasons. We provide a summary of the number of tokens in each class and the total volume of tokens in Collection-A in Table 2.

We experiment with all sixty-three ($2^6 - 1$) possible combinations of these feature classes, with each combination forming a *feature set*, using our experimental methodology from our previous work [12]. That is, we create 6-gram program representations with each feature set in turn and query each document against all works from different combinations of ten randomly selected authors.

4.2 Query Length Investigation

We next explore the effects of varying query length towards the effectiveness of source code authorship attribution; recording the query length in number of tokens against the MRR and MAP scores of each run. We still continue to run all feature class combinations described above to observe if tokens generated from some feature classes are more resilient to shorter queries than others.

4.3 Classification

Finally, we simulate a real-life 10-class authorship classification problem, attempting to correctly classify work samples as belonging to their authors using the best methods from previous sections. In the previous experiments we simply used the MRR and MAP scores as indications that a system would return a ranked list of documents with works from the correct author ranked highly in the list. In this experiment, we test three metrics for measuring the strength of style for *all* candidate authors, and we classify each work sample as belonging to the author with the highest score.

The *single best result* metric attributes authorship to the author of the top ranked document. For the *average scaled score* metric, we divide all scores returned by the ranking algorithm by the score of the top ranked document. The scores for each candidate author are then averaged and authorship is assigned to the highest average score. Finally for the *average precision* metric, we calculate the average precision for documents of each author in turn, and classify the query program as belonging to the author with the highest average precision score. For all metrics, we take care to omit the query document from all result lists. To identify the most effective metric, we average the results of 100 runs.

Table 3. Effectiveness of twelve of sixty-three tested feature sets sorted by MRR score. The six feature classes are operators, keywords, I/O tokens, function tokens, white space tokens and literal tokens respectively. We omit the remaining fifty-one results for brevity. These results demonstrate that operators, keywords and white space features together are strong markers of authorship. Note that the "feature set" column is a numeric identifier which we refer to in the text.

F. Set	Oper	Keywd	I/O	Func	W. Spc	Lit	MRR	MAP
50	Yes	Yes			Yes		82.28	41.33
58	Yes	Yes	Yes		Yes		82.20	40.36
55	Yes	Yes		Yes	Yes	Yes	81.98	39.22
51	Yes	Yes			Yes	Yes	81.74	39.74
54	Yes	Yes		Yes	Yes		81.68	41.13
62	Yes	Yes	Yes	Yes	Yes		81.61	39.90
..	...	...	...	...	...	...	.....	.....
32	Yes						74.78	26.19
..	...	...	...	...	...	...	.....	.....
16		Yes					69.40	20.33
..	...	...	...	...	...	...	.....	.....
01						Yes	65.73	23.10
08			Yes				62.07	16.79
04				Yes			56.98	9.91
02					Yes		43.26	22.15

5 Results and Analysis

In this section we provide results and analysis of our feature selection, query length investigation and classification experiments. Our classification results are then benchmarked against implementations by six other research groups.

5.1 Feature Selection Results

The top six rows of Table 3 provide a summary of the top performing feature sets (all of which have a statistically insignificant MRR from Feature Set 50 (the top row) using a permutation test at the $p < 0.05$ level). The bottom six rows show the performance of the six feature classes in isolation. As can be seen, using feature sets that contain a single feature class leads to a poor ranking of documents by the same author as the query, whereas selecting white space, operator and keyword features leads to a high ranking of similarly authored documents.

Koppel et al. [26] discuss that for text categorisation, "frequent but unstable features are especially useful", and so we would expect these to particularly effective. White space features were the most prevalent as shown in Table 2 representing 69.79% of all tokens. We believe they are especially useful as white space placement is a strong marker of programming style. For example, the following two for-loop declarations are functionally equivalent, but the use of white space can be helpful in distinguishing authorship:

```
for(i=0; i<limit; i++);          for (i = 0; i < limit; i++);
```

Our results showed that the most effective feature set without white space features had a MRR score of 4.49% less than the most effective feature set. This is a statistically significant result as shown by a permutation test at the 95% confidence interval.

Key observations were made when inspecting the volumes of each individual token. For example, the ratio of new lines to carriage returns was found to be 16:1.[4] This suggests that at least one out of sixteen submissions was not developed in our predominantly Unix-based environment which gives a useful authorship marker concerning choice of operating system for programming assignment development. The white space and literal tokens were all found in large quantities — the nine tokens that make up these categories were all within the top fifteen when totalling the volume of each token. Of the remaining token classes, the parenthesis was the most prevalent operator token, `int` was the most prevalent keyword, `NULL` was the most prevalent I/O token, and `strlen` was the most prevalent function token.

5.2 Query Length Results

In Figure 1 we summarise the results for our query length investigation. We compare the best of our feature sets (Feature Set 50) to the average of all feature sets. The goal is to show how the selection of one of our strongest feature sets can increase effectiveness particularly for the smallest queries. Our results are partitioned into program lengths of intervals of 100 tokens initially (0–99, 100–199, 200-299, and so on). Programs are then partitioned into intervals of 1,000 tokens for programs with 1,000 tokens or more given that we have less of these. The final partition is marked as 20,000 which represents all programs with at least 20,000 tokens up to the maximum in our collection (95,889 tokens).

The trends in Figure 1 show that shorter queries are markedly less effective in attributing authorship when averaged across all feature sets when compared to Feature Set 50 alone. There is a reasonable correlation between MRR/MAP and query length for Feature Set 50, however this trend drops off for queries less than 5,000 tokens for all feature sets averaged.

We conducted permutation tests on both the MRR and MAP results taking all results from Feature Set 50 and the first run for all other results. Results were found to be statistically significant ($p < 10^{-15}$). We also fitted linear models to each of the four lines and found that the gradient of the line for all feature sets compared to Feature Set 50 was 1.89 times greater for MRR and 2.86 times greater for MAP which confirms the strength of this feature set for smaller queries. Additionally, all our linear models were found to be statistically significant ($p < 10^{-13}$). The implication of these results is that we can largely retain authorship attribution effectiveness for short queries provided that an appropriate feature set is selected.

[4] New lines and carriage returns were treated as separate tokens.

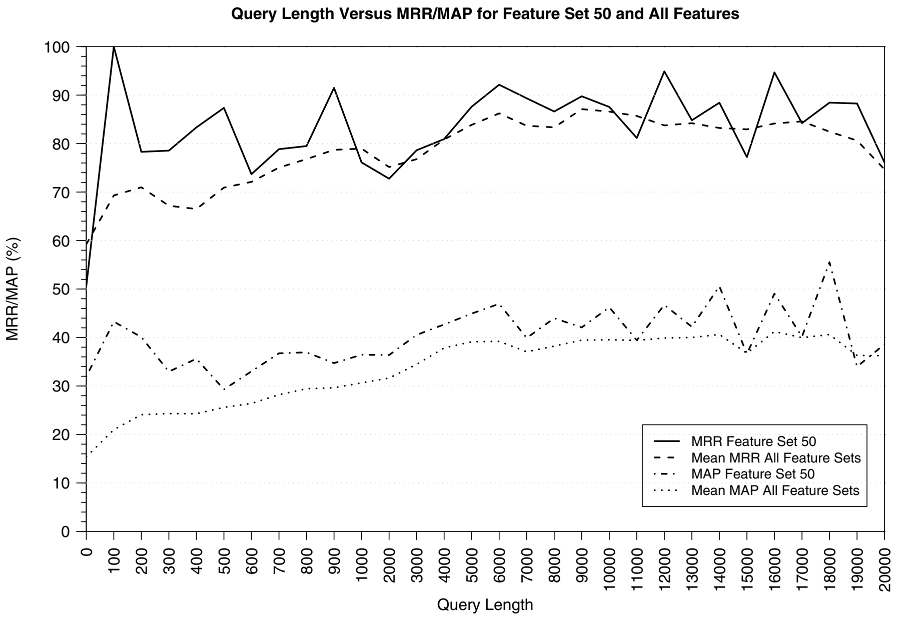

Fig. 1. Effectiveness of Feature Set 50 compared to the average of all feature sets measured in MRR and MAP. Results are partitioned based upon thirty-one query length intervals on the x-axis. These results show a downward trend for the lines representing all feature sets alone for queries less than 5,000 tokens, but results are much more consistent for Feature Set 50.

5.3 Classification Results

When using the "single best score" classification method we correctly classified work in 76.78% of cases for the 10-class problem. This is the best of our methods compared to "average scaled score" (76.47%) and "average precision" (74.68%), but this difference was not found to be statistically significant ($p > 0.05$).

In Table 4, we present our "single best score" method results for eight problem sizes ranging from 7-class to 46-class. We found that the "single best score" method was again most effective for the harder authorship attribution problems (12-class up to 46-class), however the "average scaled score" results were marginally higher for the 7-class and 8-class problems by 0.20% and 0.21% respectfully. The "single best score" results are reported here only.

We now compare our classification results to those of six other projects [8,10,15,16,17,18]. There are many variables that must be considered when comparing our results to those of other researchers. For example, choice of authorship markers, size of the test collection, level of ability of the authors whom contributed the work samples, and the choice of programming language. Therefore we don't think it is possible to select any approach as being strictly more effective than another. However, we now summarise each approach and discuss

Table 4. Comparison of our work to baseline systems of six external research groups. Codes are given in the first column indicating the first letter of the author's surname and problem size which we refer to in the text: (M07) MacDonell et al. [10] (Case-based reasoning); (F08a/b) Frantzeskou et al. [16] (Nearest neighbour); (B10) Burrows et al. (Okapi BM25); (E12) Elenbogen and Seliya [15] (C4.5 decision tree); (L20) Lange and Mancoridis [18] (Nearest neighbour); (K29) Krsul and Spafford [17] (Discriminant analysis); (F30) Frantzeskou et al. [16] (Nearest neighbour); and (D46) Ding and Samadzadeh [8] (Canonical discriminant analysis). For each baseline the remaining nine columns respectively represent the problem difficulty, range and total number of work samples, range and average lines of code of the collection, level of programming experience of the work owners ("low" for students, "high" for professionals, and "mixed" for a hybrid collection), programming language of the test collection, reported effectiveness of the approach, and our effectiveness score for the same problem size (using the "single best score" metric). To represent incomplete data, we marked non-obtainable data with a dash (—) and data obtained from personal communication with a dagger (†) or double dagger (‡) where estimates were provided.

ID	No Au	Range Work	Total Work	Range LOC	Avg LOC	Experience	Language	Effectiveness	This paper
M07	7	5–114	351	†1–1,179	†148	Mixed	C++	88.0%	78.66%
F08a	8	6–8	‡60	36–258	129	Low	Java	88.5%	78.46%
F08b	8	4–29	107	23–760	145	High	Java	100.0%	78.46%
B10	10	14–26	1,597	1–10,789	830	Low	C	76.8%	76.78%
E12	12	6–7	83	‡50–400	‡100	Low	C++†	74.7%	75.25%
L20	20	3	60	†336–80,131	11,166	High	Java	55.0%	72.47%
K29	29	—	88	—	—	Mixed	C	73.0%	70.67%
F30	30	4–29	333	20–980	172	High	Java	96.9%	70.48%
D46	46	4–10	225	—	—	Mixed	Java	67.2%	68.58%

the strengths and weaknesses of the most effective baselines to account for the differences. The major properties of each work are summarised in Table 4.

When comparing results based upon problem difficulty alone (7-class up to 46-class) using Table 4, our classification rate is well ahead of (L20) Lange and Mancoridis [18], very close to (E12) Elenbogen and Seliya [15], (K29) Krsul and Spafford [17] and (D46) Ding and Samadzadeh [8] (within 3%), and well behind (M07) MacDonell et al. [10] and (F08a/F08b/F30) Frantzeskou et al. [16]. In accounting for the strong result of (M07) MacDonell et al. [10] (9.14% higher classification rate than ours), a large portion of the collection is industry-based which we would expect to be easier to classify given the more mature programming style of the authors compared to our students. Also, the number of work samples per author varies the most compared to all other baselines. For example, the work by one author comprises 114 samples out of the 351 program collection (32.5%), and we would expect these samples to score highly as classification accuracy is 32.5% by random chance alone. Conversely, another author has only five samples which would be particularly difficult to classify correctly, but this poorer result would contribute much less to the overall reported effectiveness.

We summarised three baselines in Table 4 from Frantzeskou et al. [16] which are more effective than our equivalent result by margins of 9.85% (F08a), 21.35% (F08b) and 26.42% (F30) respectfully. The two largest margins were based on industry collections so we don't discuss these further. However, the student collection (F08a) still remained more effective by 9.85%. We observed that the results reporting various n-gram sizes and profile lengths do not agree with each other between collections so we suggest that alternative similarity measures which are less sensitive to the input parameters could be investigated as future work.

In conducting our experiments, we have witnessed varying results between runs which highlight the impact of the collection chosen on classification effectiveness. In Table 4, we reported an average result of 78.65% for our 8-class student collection problem. The eight students were randomly selected from 100 for each of 100 runs with each run attempting to classify every piece of work from eight authors. We could consider our work to have used 100 partly overlapping collections. The accuracy of these 100 results had a standard deviation of 3.92% and ranged from 64.46% up to 88.72%. This result is marginally higher than the 8-class student collection result of (F08a) Frantzeskou et al. [16] (88.5%). When considering that their approach performs poorly when an inappropriate profile length parameter is chosen for the collection in question [27], we believe our standard deviation is quite modest which suggests that our approach is more consistent and reliable.

Our collection is likely to be more difficult for authorship attribution experiments relative to the other researchers discussed above. We obtained work samples from our school's assignment submission archive from an eight-year period and students would often be completing their degrees in non-overlapping periods and have very differing assessments. Moreover, we expect some submissions to be incomplete or very small. Indeed, our smallest submission was only one line of code, and we didn't attempt to manually remove these difficult submissions.

A more precise comparison can only be performed if the above methodologies are compared on the same test collections and this remains an avenue for future work. Stein et al. [28] have cited the need to develop "publicly available large corpora" for benchmarking proposed approaches and we agree that this is important for advancing the field. Our contribution goes part way towards solving this problem as we have used more authors and more work samples than any of the benchmarked projects. However there are still privacy issues to be explored which may or may not prohibit the sharing of this student data with others.

6 Discussion and Conclusions

In this paper, we have presented a source code authorship attribution approach based on information retrieval techniques capable of solving real-world authorship attribution problems such as resolving authorship disputes, facilitating easier software project maintenance when authorship data is lacking, and supporting plagiarism detection investigations [29] where authorship evidence is needed in addition to matching content information.

The contributions built upon previous work that used a large collection of university assignments written in the C programming language [12]. The first contribution investigated effective methods for deriving surrogate documents via combining features as 6-grams for authorship attribution tasks. Several feature classes were investigated, and we found that white space, operator and keyword tokens are particularly valuable.

Next we explored the effect of varying length query documents. Smaller programs were indeed more difficult to attribute correctly when averaged across all tested feature sets, but we found that our most effective feature set containing white space, operator and keyword tokens showed greatest authorship attribution improvement for the smallest queries in particular.

Finally, using our "single best result" metric we demonstrated that we can correctly classify 76.78% of all query documents in a 10-class authorship classification experiment. When dealing with the varying collection size, data sources, programming language, underlying methodology and choice of authorship markers of competing approaches, we implemented authorship attribution experiments of equivalent difficulty. We have shown that our information retrieval approach is a competitive alternative to the six existing approaches examined [8,10,15,16,17,18] that have employed a variety of statistical analysis, machine learning and similarity measurement techniques [1].

It is hoped that our contributions will further advance the source code authorship attribution field. However, this paper represents one component of a larger research project and much remains for future work. For example, our collection contains work samples from authors of varying levels of programming ability from first year students up to final year students. We therefore anticipate differences in authorship classification effectiveness between these groups and indeed other groups such as academic staff and industry professionals. We therefore plan to explore how the evolution of programming style between these groups affects authorship classification rates.

Finally, we note that there are an increasing number of contributions towards source code authorship attribution implementations as evidenced by our work and the contributions of the authors of our baselines. Additionally, natural language authorship attribution is a particularly mature field [4,30]. We have demonstrated that with suitable choice of document surrogates, information retrieval techniques work well, so we would expect this to carry over to other domains. For example, we may consider one or more of mathematical proofs, chemical formulae, Z specifications, UML definitions, SQL, XML, HTML, $\LaTeX$ or word processing markup in future experiments.

References

1. Frantzeskou, G., Gritzalis, S., MacDonell, S.: Source code authorship analysis for supporting the cybercrime investigation process. In: Filipe, J., Belo, C., Vasiu, L. (eds.) Proceedings of the First International Conference on E-business and Telecommunication Networks, Setubal, Portugal, pp. 85–92. Kluwer Academic Publishers, Dordrecht (2004)

2. Longstaff, T.A., Schultz, E.E.: Beyond preliminary analysis of the WANK and OILZ worms: A case study of malicious code. Computers and Security 12(1), 61–77 (1993)
3. Spafford, E.H.: The internet worm: Crisis and aftermath. Communications of the ACM 32(6), 678–687 (1989)
4. Zhao, Y., Zobel, J.: Effective and scalable authorship attribution using function words. In: Lee, G.G., Yamada, A., Meng, H., Myaeng, S.-H. (eds.) AIRS 2005. LNCS, vol. 3689, pp. 174–189. Springer, Heidelberg (2005)
5. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, 1st edn. Addison Wesley Longman, Amsterdam (1999)
6. Witten, I., Moffat, A., Bell, T.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann Publishers, San Francisco (1999)
7. Glass, R.L.: Special feature: Software theft. IEEE Software 2(4), 82–85 (1985)
8. Ding, H., Samadzadeh, M.H.: Extraction of Java program fingerprints for software authorship identification. Journal of Systems and Software 72(1), 49–57 (2004)
9. Drucker, H., Wu, D., Vapnik, V.N.: Support vector machines for spam categorization. IEEE Transactions on Neural Networks 10(5), 1048–1054 (1999)
10. MacDonell, S.G., Gray, A.R., MacLennan, G., Sallis, P.J.: Software forensics for discriminating between program authors using case-based reasoning, feed-forward neural networks and multiple discriminant analysis. In: Proceedings of the Sixth International Conference on Neural Information Processing, Perth, Australia, Perth, Australia, pp. 66–71 (November 1999)
11. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Vitter, J. (ed.) Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, ACM Special Interest Group on Algorithms and Computation Theory, Dallas, Texas, pp. 604–613. ACM Press, New York (1998)
12. Burrows, S., Tahaghoghi, S.M.M.: Source code authorship attribution using n-grams. In: Spink, A., Turpin, A., Wu, M. (eds.) Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University, pp. 32–39 (December 2007)
13. Schleimer, S., Wilkerson, D., Aiken, A.: Winnowing: Local algorithms for document fingerprinting. In: Ives, Z., Papakonstantinou, Y., Halevy, A. (eds.) Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM Special Interest Group on Management of Data, San Diego, California, pp. 76–85. ACM Press, New York (2003)
14. Jones, E.: Metrics based plagiarism monitoring. In: Meinke, J.G. (ed.) Proceedings of the Sixth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges, Middlebury, Vermont, Consortium for Computing Sciences in Colleges, pp. 253–261 (April 2001)
15. Elenbogen, B., Seliya, N.: Detecting outsourced student programming assignments. Journal of Computing Sciences in Colleges 23(3), 50–57 (2008)
16. Frantzeskou, G., Stamatatos, E., Gritzalis, S., Katsikas, S.: Effective identification of source code authors using byte-level information. In: Anderson, K. (ed.) Proceedings of the Twenty-Eighth International Conference on Software Engineering, Shanghai, China, ACM Special Interest Group on Software Engineering, pp. 893–896 (May 2006)
17. Krsul, I., Spafford, E.H.: Authorship analysis: Identifying the author of a program. Computers and Security 16(3), 233–257 (1997)

18. Lange, R.C., Mancoridis, S.: Using code metric histograms and genetic algorithms to perform author identification for software forensics. In: Thierens, D. (ed.) Proceedings of the Ninth Annual Conference on Genetic and Evolutionary Computation, London, England, ACM Special Interest Group on Genetic and Evolutionary Computation, pp. 2082–2089. ACM Press, New York (2007)
19. Moffat, A., Zobel, J.: Rank-biased precision for measurement of retrieval effectiveness. ACM Transactions on Information Systems 27(1), 1–27 (2008)
20. Singhal, A., Buckley, C., Mitra, M.: Pivoted document length normalization. In: Frei, H.-P., Harman, D., Schaubie, P., Wilkinson, R. (eds.) Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Zurich, Switzerland, pp. 21–29. ACM Press, New York (1996)
21. Zhai, C., Lafferty, J.: A study of smoothing methods for language models applied to information retrieval. ACM Transactions on Information Systems 22(2), 179–214 (2004)
22. Jones, K.S., Walker, S., Robertson, S.E.: A probabilistic model of information retrieval: Development and comparative experiments part 1. Information Processing and Management 36(6), 779–808 (2000)
23. Jones, K.S., Walker, S., Robertson, S.E.: A probabilistic model of information retrieval: Development and comparative experiments part 2. Information Processing and Management 36(6), 809–840 (2000)
24. Cannon, L.W., Elliott, R.A., Kirchhoff, L.W., Miller, J.H., Miller, J.M., Mitze, R.W., Schan, E.P., Whittington, N.O., Spencer, H., Keppel, D., Brader, M.: Recommended C style and coding standards. Technical report, Bell Labs, University of Toronto, University of Washington and SoftQuad Incorporated (February 1997) (accessed September 24, 2008), `http://vlsi.cornell.edu/courses/eecs314/tutorials/cstyle.pdf`
25. Oman, P.W., Cook, C.R.: A taxonomy for programming style. In: Sood, A. (ed.) Proceedings of the 1990 ACM Annual Conference on Cooperation, Association for Computing Machinery, pp. 244–250. ACM Press, New York (1990)
26. Koppel, M., Akiva, N., Dagan, I.: A corpus-independent feature set for style-based text categorization. In: Proceedings of the IJCAI 2003 Workshop on Computational Approaches to Style Analysis and Synthesis, Acapulco, Mexico (2003)
27. Frantzeskou, G., Stamatatos, E., Gritzalis, S., Katsikas, S.: Source code author identification based on n-gram author profiles. In: Maglogiannis, I., Karpouzis, K., Bramer, M. (eds.) Artificial Intelligence Applications and Innovations, vol. 204, pp. 508–515. Springer, New York (2006)
28. Stein, B., zu Eissen, S.M., Potthast, M.: Strategies for retrieving plagiarized documents. In: Kraaij, W., de Vries, A.P., Clarke, C.L.A., Fuhr, N., Kando, N. (eds.) Proceedings of the Thirtieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 825–826. ACM Press, Amsterdam (2007)
29. Burrows, S., Tahaghoghi, S.M.M., Zobel, J.: Efficient plagiarism detection for large code repositories. Software: Practice and Experience 37(2), 151–175 (2006)
30. Zhao, Y., Zobel, J., Vines, P.: Using relative entropy for authorship attribution. In: Ng, H.T., Leong, M.-K., Kan, M.-Y., Ji, D. (eds.) AIRS 2006. LNCS, vol. 4182, pp. 92–105. Springer, Heidelberg (2006)