

# Source Code Authorship Attribution using $n$ -grams

Steven Burrows      S.M.M. Tahaghoghi

School of Computer Science and Information Technology

RMIT University

GPO Box 2476V, Melbourne 3001, Australia

{*steven.burrows,seyed.tahaghoghi*}@rmit.edu.au

## Abstract

*Plagiarism and copyright infringement are major problems in academic and corporate environments. Existing solutions for detecting infringements in structured text such as source code are restricted to textual similarity comparisons of two pieces of work. In this paper, we examine authorship attribution as a means for tackling plagiarism detection. Given several samples of work from several authors, we attempt to correctly identify the author of work presented as a query. On a collection of 1640 documents written by 100 authors, we show that we can attribute authorship in up to 67% of cases. This work can be a valuable additional indicator for the more difficult plagiarism investigations.*

**Keywords:** Authorship Attribution, Plagiarism Detection, Co-derivative Documents

## 1 Introduction

Educational institutions and industry frequently encounter plagiarism and copyright infringement. Plagiarism is the presentation by one person of the ideas or work of another person as their own; this presentation may be in the form of plain text — as present in essays and reports — or structured text, such as equations and computer programming languages. Copyright provides exclusive publishing rights to authors to protect their ideas and information. Authors may licence their copyrighted works for a fee, but unauthorised reproductions by other parties are considered copyright infringement.

Searching for plagiarism or copyright infringement by human inspection is time consuming and not practical for large collections of work. Plagiarism detection tools, such as Turnitin [16] for plain text, are often used to detect infringed work. However, approaches for detecting plagiarism in plain text are not suitable for detecting plagiarism in structured text, as they ignore important aspects of structured text documents such as programming syntax in source

**Proceedings of the 12th Australasian Document Computing Symposium, Melbourne, Australia, December 10, 2007. Copyright for this article remains with the authors.**

code. For example, changing literal names or rewriting comments does not affect the function of a program.

Source code plagiarism detection tools for detecting anomalies within a set of files — such as assignment submissions by a student cohort — are available [4, 8, 11, 17, 26, 33]. However, these do not consider historical data on each author, and cannot detect whether work has been plagiarised from a source outside the set.

*Authorship attribution* aims to detect plagiarism by establishing a profile of an author's style from several sample documents. Work diverging significantly from the profile can be flagged for investigation. If the work fits the profile of a different known author, then the case for plagiarism is strengthened.

Authorship attribution has been studied for plain text [36], but has not been explored for structured text such as source code. We investigate several techniques for attributing authorship of source code, building on techniques used in text information retrieval. On a collection of student assignments containing 1640 documents from 100 authors, the best of our techniques correctly attributes authorship 67% of the time.

The remainder of this paper is organised as follows. In the next section we report on the magnitude of unauthorised code reuse. In Section 3, we review existing approaches to source code similarity detection and plain-text authorship attribution. In Section 4 we describe our approach to the problem, and introduce the data we use to evaluate our solutions. We present results of our experiments and discuss our findings in Section 5, and give conclusions in Section 6.

## 2 Motivation

Unauthorised reuse of code is a major problem in both academia and industry. Marsden [25] describe the results of a comprehensive study of 954 students from four Australian universities where 81% admitted to having engaged in some form of plagiarism. Moreover, Alemozafar [1] describes the increasing trend of the problem at Stanford University, where the Office of Judicial Affairs witnessed violations of the honour code increase by 126% between 1998 and 2001.

In interviews with academic staff, Bull et al. [5] found that 50% of the 321 respondents agreed that there “has been an increase of plagiarism in recent years”;

only 15% disagreed and 35% didn't know. Culwin et al. [7] describe another study in which representatives from 55 higher education computing schools completed a questionnaire. They found that 89% of institutions surveyed felt that source code plagiarism was either a "minor nuisance", a "routine headache" or "bad and getting worse". Only 9% felt that it was either "not a problem" or "under control".

Several cases of plagiarism have attracted media attention or prompted special actions by academic staff. Ketchell [21] describes a case at RMIT University where staff were alerted to suspicious pin-up advertisements offering tutoring and "extra help". The investigation found many examples of work produced by the advertiser, with some solutions concurrently sold to multiple students. Zobel [38] notes that if fresh solutions were provided for each client, profiling student work is likely to be key to automated detection.

Some web sites act as software development marketplaces by allowing users to post work specifications and invite competitive bids from independent contractors. D'Souza et al. [9] report two cases where students had finalised deals on the RentACoder web site<sup>1</sup> for assignment solutions valued at US\$200 and AU\$35; these prices are affordable to many students. To demonstrate the severity of this problem, D'Souza et al. presented a list of over twenty such web sites that function as software development marketplaces.

There is a need for whole organisations to protect themselves against plagiarism and copyright infringement. For example, Edward Waters College in Jacksonville, Florida had its accreditation revoked in 2004 after plagiarised content was found in documentation sent to its accreditation agency [3]. Accreditation was regained six months later after legal proceedings [23]. This incident resulted in reduced enrolments and threatened current students with loss of funding.

Unauthorised code reuse is also a concern in the corporate sector. For example, SCO Group sued IBM for more than one billion dollars for allegedly incorporating Unix code in its Unix-like AIX operating system in March 2003 [27]. Similarly, the Software Freedom Law Center, which investigates possible violations of software licences such as the GNU General Public License, has identified the need for automated approaches to determine whether one project is the derivative of another [22].

### 3 Background

Tools to detect authorship issues in source code identify matches using metric-based or structure-based approaches [32]. Jones [17] describes an unnamed metric-based system with a hybrid of physical metrics (line, word, and character counts) and Halstead metrics [13] (token occurrences, unique tokens, and Halstead

volume) to characterise code. Jones uses the Euclidean distance measure on normalised program profiles to score program closeness. Other metric-based systems reported in the literature employ between four and twenty-four metrics [8, 10, 12]. These systems are among the first that implemented electronic plagiarism detection and many measure archaic features of source code no longer present in modern-day programming languages. Limitations exist in the work by Faidhi and Robinson [10] and Grier [12] who measure features in Pascal code as some of the features chosen are among the easiest to modify by plagiarists such as comments and indentation. Donaldson et al. [8] present a plagiarism detection system for Fortran code, however the summation metrics used to compare programs are sensitive to program length.

Prechelt et al. [26] describe the JPlag structure-based system that works in two phases. First, program source code files are parsed and converted into token streams. Second, the token streams are compared in an exhaustive pairwise fashion using the Greedy String Tiling algorithm as used in the YAP plagiarism detection system [33]. Collections of maximally overlapping token streams above a threshold length are stored and given a similarity score. Program pairs with a similarity score above a threshold percentage are made available to the user for manual side-by-side inspection. Systems such as this are unsuited to verify authorship as they aim to identify functionally equivalent — rather than stylistically consistent — code.

In previous work [6] we describe a scalable code similarity solution that uses the Zettair search engine<sup>2</sup> to index  $n$ -grams [34] of tokens extracted from the parsed program source code. At search time, the index is queried using the  $n$ -gram representations of each program to identify candidate results. These are then post-filtered using the local alignment approximate string matching technique [30].

Plain text infringement detection tools [5, 28] have been used on academic and corporate text-based work such as essays and reports. Hoad and Zobel [14] describe two methods for detecting co-derivative text documents. *Ranking* involves presenting the user with a list of candidate answers sorted by similarity to a query; this approach is commonly used by search engines to retrieve multiple answers to a query where there is not necessarily a single correct answer. *Fingerprinting* computes compact descriptions of documents using a hash function. It is critical that the generated integers are as uniformly distributed as possible to reduce the number of duplicates — this is a property of any good hash function. Other issues for consideration are the substring length used for the hash function (fingerprint granularity) and the number of fingerprints used to represent a document (fingerprint resolution). Manber [24] describe the *sif* plagiarism detection

<sup>1</sup><http://www.rentacoder.com>

<sup>2</sup><http://www.seg.rmit.edu.au/zettair>

system that uses fingerprinting. Fingerprinting is a lossy approach, so we use ranking in our work to retain all stylistic markers.

Authorship attribution techniques for plain text apply text retrieval, statistical, or machine-learning approaches on a set of document features to characterise an author’s *style* — Zhao et al. [36, 37] compare many of these. Zhao and Zobel [36] describe three classes of authorship attribution. In one-class authorship attribution, a pool of documents is gathered where some are known to belong to a single author and others are of unknown authorship; the task is to determine whether a new document belongs to the known author. In binary authorship attribution, a collection contains documents belonging to one of two authors, and we must determine to which author a new document belongs. Binary authorship attribution is a special case of multi-class authorship attribution, where more than two authors are involved. We focus on the multi-class attribution problem for source code as it best simulates a real-life scenario involving authorship verification of work samples from large classes.

Turnitin<sup>3</sup> and the related iThenticate<sup>4</sup> system are well-known text plagiarism detection tools. These compare submitted work to documents from the Web and other repositories [15, 16]. However, details of the system implementation of Turnitin are difficult to obtain due to commercial considerations.

## 4 Methodology

We use an information retrieval model for attributing authorship; to determine the author of a document, we use it as a search engine query — the *query document*, written by the *query author* — on a corpus with known authors; our approach produces a list of all documents in the corpus ranked by estimated similarity to the query document. Ideally, every other corpus document by the query author would appear at the top of the ranked list, followed by all other documents. We now detail each component of this process.

### 4.1 Document collection construction

Our collection contains 1 640 assignment submissions from the 100 most prolific authors of C programming assignment submissions submitted to our school over the period 1999 to 2006. We removed byte-identical submissions from the same authors, and removed all personal details from the source code in adherence to ethics guidelines.

We expect our collection to have some bias towards weaker students. For example, our collection has many submissions from our first semester C programming course — Programming Techniques. We found that approximately 12% of submissions over nine consecutive offerings of Programming Techniques

were from repeat students who submitted to more than one offering. However we expect some of the strongest students who pursue further coursework programs to also submit more assignments than average.

Each author in our collection is represented by at least fourteen documents; the number of documents per author is shown in Figure 1a. The collection contains source code of varying complexity and size. The distribution of submission lengths is shown in Figure 1b, with the exception of twenty-nine large outliers (less than 2% of the total number of files in the collection) that we omit from the graph for brevity. Twenty-six of these submissions were between 5 000 and 9 000 tokens, with the remaining cases having lengths of 10 951, 14 998 and 22 598 tokens respectively.

While some documents in this corpus may be plagiarised from other sources, we consider our results to be valid; much research in information retrieval and authorship attribution relies on imperfect ground truth. For example, Amitay et al. [2] describe collections obtained from online sources where manual verification of data was infeasible, but where some authors may have created work under multiple aliases.

Recent plain-text authorship attribution work has demonstrated that “function words” — such as “and”, “or”, and “but” — are strong markers of individual style [37]. In keeping with this approach, we discard all comments and literals from the program source code in our collection, and retain only operators and keywords; these are listed in Table 1 [20]. Note that the reduction was done at a lexical level, rather than with reference to the C language grammar, and so overloaded tokens are not treated differently in this work. For example, all ampersands are treated as bitwise-and operators, even if they were used to generate a pointer from a variable.

### 4.2 Evaluation

To preserve some local information on token sequences, we extract windows of  $n$  tokens overlapping by one token; a program of  $t$  tokens is represented by  $t - n + 1$   $n$ -grams. We ignore submissions smaller than  $n$ . This approach is similar to that of our previous work [6]. To determine the best value of  $n$ , we tested average attribution effectiveness over 100 runs, with each run using random samples of work from our collection.

To determine the best choice of document similarity measure, we evaluate four common text similarity measures built into the Zettair search engine that we use: Okapi BM25 [18, 19], Cosine [34], Pivoted Cosine with a pivot value of 0.2 [29], and language modeling with Dirichlet smoothing with  $\mu$  set to 2000 [35]. We also propose a fifth metric that we call Author1 designed specifically for source code authorship attribution. This measure is based upon the idea that the term frequency in a query and a document should be similar when the query is a document. We define this measure as:

<sup>3</sup><http://www.turnitin.com>

<sup>4</sup><http://www.ithenticate.com>

Operators				
(	parenthesis	!=	not equals	
[	bracket	&	bitwise and	
->	i. member access	^	bitwise xor	
.	member access		bitwise or	
++	increment	&&	boolean and	
--	decrement		boolean or	
!	not	?	conditional	
~	complement	=	equals	
*	multiply	+=	plus equals	
/	divide	-=	minus equals	
%	modulo	*=	multiply equals	
+	plus	/=	divide equals	
-	minus	%=	modulo equals	
<<	left shift	<<=	left shift equals	
>>	rightshift	>>=	right shift equals	
<	less than	&=	bitwise and equals	
>	greater than	^=	bitwise xor equals	
<=	less than equals	=	bitwise or equals	
>=	great. than equals	,	comma	
==	equality			

Keywords				
auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	

Table 1: Operators and keywords preserved in documents in our collection [20].

$$\text{Author1}(Q, D_d) = \sum_{t \in Q \cup D_d} \frac{1}{\min(|f_{q,t} - f_{d,t}|, 0.5)}$$

where  $t$  is a term,  $Q$  is a query,  $D_d$  is a document in collection  $D$ ,  $f_{q,t}$  is the query-term frequency, and  $f_{d,t}$  is the document-term frequency.

Our choices for a baseline comparison system were limited — there is much previous work on source code plagiarism detection, but none on source code authorship attribution. We chose the most recent published metric-based plagiarism detection approach — that of Jones [17] — as our baseline. We believe that structure-based plagiarism detection systems such as JPlag [26] are less suitable for this task as they match contiguous plagiarised chunks of source code best, but this remains to be proven in future work.

We measure authorship attribution effectiveness using the reciprocal rank of the first correct author match for each query. We also measure authorship attribution effectiveness of whole result sets using average precision. Approaches are compared using the mean reciprocal rank percentage (MRR%) and mean average precision percentage (MAP%) over random samples of documents in our collection, where each document is treated in turn as the query document.

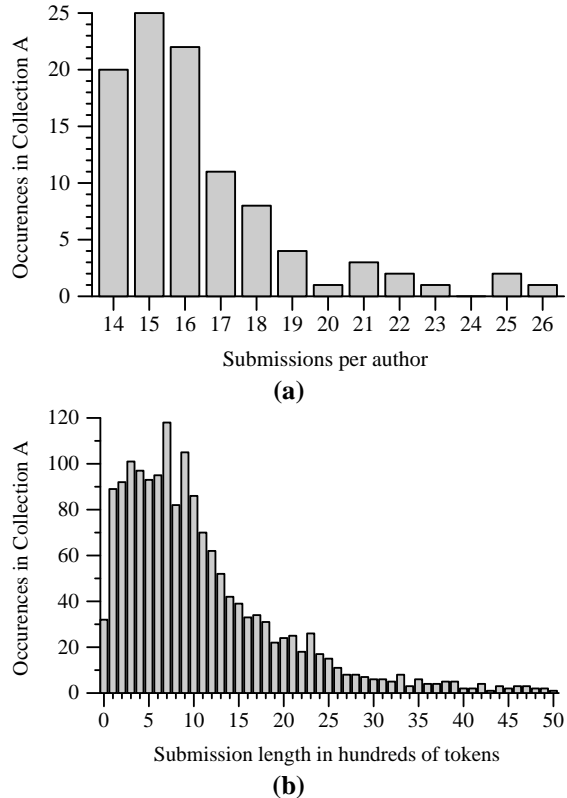


Figure 1: **(a)** Number of submissions for each author in COLLECTION A. **(b)** Distribution of submission lengths in COLLECTION A in hundreds of tokens, rounded down to the nearest 100 tokens.

## 5 Results

In this section, we first discuss results describing the most appropriate  $n$ -gram size for our problem domain. Second, we explore the degradation of reciprocal rank and average precision scores as the number of authors is increased and the number of submissions per author is decreased. Finally, we present our results in comparison to a baseline system before offering discussion.

### 5.1 $n$ -gram results

Table 2 illustrates the performance of the five similarity measures using sixteen different values of  $n$ . For each similarity measure and  $n$ -gram size combination, we use all work samples from ten authors selected at random from COLLECTION A and calculate the mean reciprocal rank and mean average precision percentages of 100 runs. With ten authors, we would expect random behaviour to produce a mean reciprocal rank and mean average precision of around 10%.

In the same table we highlight the most effective gram size for each similarity measure. Unigrams are clearly inferior, and the best choice of  $n$ -gram size is six or eight for mean reciprocal rank and twenty for mean average precision. Effectiveness degrades for the largest  $n$ -gram sizes, as the number of large  $n$ -grams shared between program representations falls quickly. Such large  $n$ -gram sizes are suitable only for

Gram Size	Similarity Measure MRR%				
	Au1	Cos	Dir	Oka	P.Co
1	54.28	61.75	23.22	44.56	28.72
2	67.15	70.24	29.10	68.60	54.51
4	73.17	75.06	56.17	76.30	74.11
6	75.65	<b>76.51</b>	61.50	<b>77.77</b>	<b>76.81</b>
8	<b>76.63</b>	75.90	<b>62.59</b>	77.43	76.80
10	75.63	74.97	61.64	76.49	76.25
12	75.91	75.00	63.96	76.64	76.31
14	73.76	73.07	63.92	74.46	74.23
16	73.16	72.50	64.80	73.50	73.56
18	72.05	71.97	66.34	72.51	72.44
20	70.03	70.03	66.16	70.36	70.39
30	62.21	62.17	61.88	62.29	62.22
40	56.64	56.53	56.51	56.59	56.62
50	52.96	53.08	52.93	53.07	53.10
70	48.22	48.26	48.13	48.28	48.29
90	44.19	44.21	44.13	44.23	44.22

Gram Size	Similarity Measure MAP%				
	Au1	Cos	Dir	Oka	P.Co
1	18.71	21.83	12.92	19.39	14.27
2	22.94	24.95	13.95	25.53	19.91
4	26.69	27.73	21.35	28.84	26.28
6	29.25	29.52	23.41	30.60	29.33
8	32.46	32.00	25.85	33.34	32.56
10	37.56	37.25	30.36	38.55	38.07
12	44.00	43.56	36.91	44.77	44.36
14	53.34	53.11	46.19	54.12	53.77
16	58.82	58.58	52.26	59.43	59.32
18	63.25	63.28	58.43	63.90	63.74
20	<b>64.33</b>	<b>64.48</b>	<b>60.91</b>	<b>64.95</b>	<b>64.86</b>
30	60.89	60.91	60.47	61.08	60.96
40	56.08	56.11	56.16	56.16	56.15
50	52.75	52.90	52.76	52.90	52.90
70	48.10	48.18	48.09	48.19	48.17
90	44.17	44.19	44.12	44.21	44.20

Table 2: Effect of increasing the  $n$ -gram size using collections of ten authors and sixteen samples of work per author on average.

near-duplicate cases of plagiarism which are rare — for example iParadigms [16] detail that 29% of papers received exhibit “significant plagiarism” whereas only 1% are entire copies. The remaining 70% have no plagiarism.

We conducted Tukey HSD (Honestly Significant Difference) tests on a subset of results in Table 2. Testing the 6-gram, 8-gram and 10-gram rows, we found that Okapi was statistically most effective when comparing mean reciprocal rank. Okapi with 6-grams is our best result, but this is not statistically significant from six out of fourteen other combinations of similarity measure and  $n$ -gram size tested. Concerning mean average precision, Okapi was only statistically more significant than Dirichlet and the Author1 metric when testing the 18-gram, 20-gram and 30-gram rows. Okapi with 20-grams is our most effective combination; this is statistically better than 18-grams

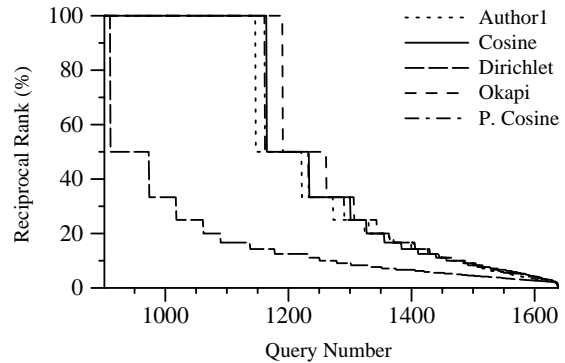


Figure 2: Reciprocal rank of five similarity measures tested using using 6-gram program representations. The combined results of ten runs are shown using collections of ten authors and sixteen samples of work per author on average. The first 900 results are omitted as these all achieved a perfect reciprocal rank.

and 30-grams, but is only statistically better than Dirichlet when considering other combinations in the same row.

Three of the five similarity measures shared the highest mean reciprocal rank score for the 6-gram program representations; this is illustrated in Figure 2, with results sorted by decreasing reciprocal rank. The best 900 query results for each measure had a perfect reciprocal rank score, and we omit these for brevity. With the exception of the Dirichlet measure, the similarity measures performed with similar effectiveness, with perfect reciprocal rank scores in at least 1 146 queries out of 1 640. The poor performance of the Dirichlet measure may possibly be addressed through empirical tuning of the  $\mu$  parameter for our problem domain.

We select 6-grams as our choice of gram size for experiments described in Sections 5.2 and 5.3.

## 5.2 $n$ -gram scalability

In Table 3 we show the degradation of mean reciprocal rank and mean average precision scores over five collection sizes for all five similarity measures evaluated. Unsurprisingly, the scores drop as the collection size increases, but we still achieve a mean reciprocal rank of 67% using the Okapi measure on our full collection. We anticipate that the scores will degrade to an unacceptable level when the collection size reaches thousands of authors — this is a real-life collection size as evidenced by our school assignment archives dating back to 1999. We intend to verify authorship in future work by comparing query documents to other documents by the same author and a subset of other authors instead of exhaustive comparison. The intention is to be satisfied that submitted works belong to the author, and not necessarily find the source of plagiarism offences.

To verify statistical significance of these results, we considered rows with ten and twenty-five submissions per author and we found that results were more conclu-

Num Auth	Similarity Measure MRR%				
	Au1	Cos	Dir	Oka	P.Co
10	<b>75.65</b>	<b>76.51</b>	<b>61.50</b>	<b>77.77</b>	<b>76.81</b>
25	69.68	70.88	58.47	72.52	71.32
50	66.21	67.32	57.44	69.61	68.30
75	64.43	65.41	56.63	68.06	66.58
100	63.26	64.14	56.31	67.01	65.48

Num Auth	Similarity Measure MAP%				
	Au1	Cos	Dir	Oka	P.Co
10	<b>29.25</b>	<b>29.52</b>	<b>23.41</b>	<b>30.60</b>	<b>29.33</b>
25	20.63	20.74	16.52	22.06	20.89
50	16.75	16.84	13.86	18.23	17.17
75	15.05	15.12	12.73	16.54	15.53
100	14.09	14.16	12.16	15.55	14.58

Table 3: Effect of increasing the number of authors with 6-gram program representations and collections of sixteen submissions per author on average.

sive for this experiment. Okapi was statistically most effective for both mean reciprocal rank and mean average precision. Okapi with 10-grams was always the statistically most effective combination except for Pivoted Cosine with 10-grams when measuring mean reciprocal rank.

In Table 4 we show the effect of changing the number of submissions per author. We note that the final row of results in these tables are for sixteen submissions per author *on average*; the actual number of varies from fourteen to twenty-six. We observe a steady decline in mean reciprocal rank scores as the number of submissions per author is decreased. Given that reciprocal rank uses the single best result, having more submissions per author gives a higher chance of encountering a document by the same author with a strong style match. We note that our collection is biased towards the most prolific authors in our school, so having a sharp decline like this is undesirable. However, we observe a different trend for mean average precision — these scores are close to constant with a gradual increase as the number of submissions per author is reduced. Given that mean average precision is a measure showing non-degradation of effectiveness using a combination of evidence, we plan to design future experiments with this in mind by implementing similar ideas with voting models.

For this experiment, we tested statistical significance of the rows marked bold in Table 4 and the two nearest rows for each. Okapi is once again the statistically most effective similarity measure. When measuring mean reciprocal rank, Okapi with sixteen submissions per author is statistically better than any other combination except Pivoted Cosine with sixteen submissions per author. However when measuring mean average precision, Okapi with any number of submissions per author is only statistically better than Dirichlet when considering results in the same row.

Subs Auth	Similarity Measure MRR%				
	Au1	Cos	Dir	Oka	P.Co
2	31.90	32.82	25.74	33.49	32.22
4	50.70	50.22	39.17	51.27	49.99
8	64.22	63.84	51.27	65.20	63.70
12	70.62	70.63	56.55	71.95	70.90
16	<b>75.65</b>	<b>76.51</b>	<b>61.50</b>	<b>77.77</b>	<b>76.81</b>

Subs Auth	Similarity Measure MAP%				
	Au1	Cos	Dir	Oka	P.Co
2	<b>31.90</b>	<b>32.82</b>	<b>25.74</b>	<b>33.49</b>	<b>32.21</b>
4	31.46	31.57	24.67	32.34	31.19
8	30.04	30.32	23.89	31.38	30.10
12	29.49	29.87	23.31	30.94	29.62
16	29.25	29.52	23.41	30.60	29.33

Table 4: Effect of increasing the number of submissions per author with 6-gram program representations and collections of ten authors.

Number of Authors	Similarity Measure MRR%	
	Okapi BM25	Euclidean
10	<b>77.77</b>	<b>40.06</b>
25	72.52	28.41
50	69.61	22.62
75	68.06	20.35
100	67.01	18.89

Number of Authors	Similarity Measure MAP%	
	Okapi BM25	Euclidean
10	<b>30.60</b>	<b>15.15</b>
25	22.06	7.37
50	18.23	4.52
75	16.54	3.52
100	15.55	2.97

Table 5: Comparison of our work to a baseline system [17] using the same collection properties described in Table 3.

### 5.3 Baseline comparison

In Table 5 we compare the effectiveness of the best of our approaches with the metric-based system described by Jones [17]. Our approach is statistically more effective than the baseline system, regardless of the similarity function used.

### 5.4 Discussion

Authorship attribution can be a valuable tool in verifying the integrity of documents. We have shown that the combination of  $n$ -grams with text similarity measures can lead to effective authorship attribution for C program source code.

Our results indicate that 6-grams and 8-grams perform well when high mean reciprocal rank is required, while 20-grams are the best choice when aiming for high mean average precision. This differs from results on plain text, where gram lengths of three, four, or five have been shown to be most effective [14, 31]. We believe that this is due to the limitations imposed on

program code by the programming language, requiring more evidence to distinguish between authors.

Our collection contains program code from students of varying competence, and our experiments do not explicitly allow for this. We expect that results of experiments using structured text collections from more mature authors may remedy this problem. Possible resources for such experiments include program code from open source projects and supplementary examples from programming text books.

Other published work relates to plain text authorship attribution. This is a mature field, and Zhao and Zobel [36] compare the effectiveness of several strategies. However, these results cannot be assumed in the structured text domain as our  $n$ -gram experimental results have highlighted that larger  $n$ -gram sizes are needed.

## 6 Conclusion

In this paper, we have presented the case for detecting authorship attribution in program source code, and shown how existing automated tools are not suited to handle this problem.

We have reported experimental results that use a search engine to index and retrieve tokenised representations of C program source codes, with the aim of ranking documents authored by the query author above all other documents in the collection.

Our results show that larger  $n$ -gram sizes are needed than that of the plain text author attribution domain. We have shown that our work is clearly more effective than that of a reference baseline solution. We have also investigated scalability implications that are critical considerations before a structured text authorship attribution system is put into production.

Surprisingly, Okapi BM25 was modestly more effective than our Author1 metric custom designed for our problem domain. In future work we plan to explore the authorship attribution metric further as our results confirm that a small difference in term frequency is important.

Program source code is one example of structured text; other examples include mathematical equations, XML,  $\LaTeX$ , word processing markup, and we intend to investigate authorship attribution on such data.

We intend to use collections outside the academic domain. Our university assignment collection exhibits high variance in coding quality, so we plan to secure new collections from industry for further experiments.

We also plan to more vigorously investigate our choice of features. Our work uses more features than that of metric-based plagiarism detection systems and we use these features with  $n$ -grams to look for short patterns to evaluate of author style. Other combinations of features will be investigated. We note that this work examines features at the lexical level and more work is needed to extend tokenisation to the grammatical level so that overloaded tokens are treated differently.

The experiments described in this paper are limited to 100 authors; we expect that attribution effectiveness will continue to degrade as the number of authors is increased. We conjecture that a practical system might not perform exhaustive comparisons to all other authors; but instead compare the query document to other samples of work by the same author and samples from other authors obtained randomly or determined by some form of initial filtering. We plan to explore voting schemes that use multiple indicators to determine authorship of query documents.

## Acknowledgements

We thank Andrew Turpin for his valuable ideas and feedback.

## References

- [1] A. Alemozar. Online software battles plagiarism at Stanford. *The Stanford Daily*, February 2003.
- [2] E. Amitay, S. Yogev and E. Yom-Tov. Serial sharers: Detecting split identities of web authors. In B. Stein, M. Koppel and E. Stamatatos (editors), *Proceedings of the SIGIR'07 International Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection*, pages 11–18, Amsterdam, Netherlands, July 2007. CEUR Workshop Proceedings.
- [3] B. Bollag. Edward waters college loses accreditation following plagiarism scandal. *The Chronicle of Higher Education*, December 2004. URL: <http://chronicle.com/prm/daily/2004/12/2004120904n.htm> [Accessed 8 October 2007].
- [4] K. Bowyer and L. Hall. Experience using MOSS to detect cheating on programming assignments. In *Proceedings of the Twenty-Ninth ASEE/IEEE Frontiers in Education Conference*, pages 18–22, San Juan, Puerto Rico, November 1999. IEEE Computer Society Press.
- [5] J. Bull, C. Collins, E. Coughlin and D. Sharp. Technical review of plagiarism detection software report. Technical Report LU1 3JU, University of Luton Computer Assisted Assessment Centre, Bedfordshire, England, July 2002.
- [6] S. Burrows, S. M. M. Tahaghoghi and J. Zobel. Efficient plagiarism detection for large code repositories. *Software: Practice and Experience*, Volume 37, Number 2, pages 151–175, February 2007.
- [7] F. Culwin, A. MacLeod and T. Lancaster. Source code plagiarism in UK HE computing schools: Issues, attitudes and tools. Technical Report SBU-CISM-01-01, South Bank University School of Computing, Information Systems and Mathematics, September 2001.
- [8] J. Donaldson, A. Lancaster and P. Sposato. A plagiarism detection system. In *Proceedings of the Twelfth SIGCSE Technical Symposium on Computer Science Education*, pages 21–25. ACM Press, 1981.
- [9] D. D'Souza, M. Hamilton and M. Harris. Software development marketplaces—implications for plagiarism. In S. Mann and Simon (editors), *Proceedings of the Ninth Australasian Computing Education Conference*, pages 27–33, Ballarat, Australia, January 2007. Australian Computer Society.
- [10] J. A. W. Faidhi and S. K. Robinson. An empirical approach for detecting program similarity and plagiarism within a university programming environment. *Computers and Education*, Volume 11, Number 1, pages 11–19, 1987.

- [11] D. Gitchell and N. Tran. Sim: A utility for detecting similarity in computer programs. In J. Prey and R. E. Noonan (editors), *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, pages 266–270, March 1999.
- [12] S. Grier. A tool that detects plagiarism in Pascal programs. In *Proceedings of the Twelfth SIGCSE Technical Symposium on Computer Science Education*, pages 15–20. ACM Press, 1981.
- [13] M. H. Halstead. Natural laws controlling algorithm structure? *ACM SIGPLAN Notices*, Volume 7, Number 2, pages 19–26, February 1972.
- [14] T. Hoad and J. Zobel. Methods for identifying versioned and plagiarised documents. *Journal of the American Society for Information Science and Technology*, Volume 54, Number 3, pages 203–215, February 2002.
- [15] iParadigms. iThenticate questions and answers. *iThenticate Web Site*, October 2007. URL: <http://www.ithenticate.com/static/training.html> [Accessed 4 October 2007].
- [16] iParadigms. Turnitin plagiarism prevention. *Turnitin Web Site*, October 2007. URL: <http://www.turnitin.com/static/plagiarism.html> [Accessed 11 November 2007].
- [17] E. Jones. Metrics based plagiarism monitoring. In *Proceedings of the Sixth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, pages 253–261, Middlebury, Vermont, April 2001. Consortium for Computing Sciences in Colleges.
- [18] K. Sparck Jones, S. Walker and S. E. Robertson. A probabilistic model of information retrieval: Development and comparative experiments part 1. *Information Processing and Management*, Volume 36, Number 6, pages 779–808, November 2000.
- [19] K. Sparck Jones, S. Walker and S. E. Robertson. A probabilistic model of information retrieval: Development and comparative experiments part 2. *Information Processing and Management*, Volume 36, Number 6, pages 809–840, November 2000.
- [20] A. Kelly and I. Pohl. *A Book on C*. Addison Wesley Longman, Reading, Massachusetts, fourth edition, December 1997. pp. 77, 705.
- [21] M. Ketchell. The third degree. *The Age*, May 2003. URL: <http://www.theage.com.au/articles/2003/05/26/1053801319696.html> [Accessed 4 October 2007].
- [22] B. M. Kuhn. Personal communication, 2007.
- [23] D. Lederman. Edward waters college regains accreditation. *Inside Higher Ed*, June 2005. URL: <http://www.insidehighered.com/news/2005/06/24/waters> [Accessed 8 October 2007].
- [24] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Francisco, California, January 1994.
- [25] H. Marsden. Who cheats at university? A self-report study of dishonest academic behaviours in a sample of Australian university students. *Australian Journal of Psychology*, Volume 57, Number 1, pages 1–10, 2005.
- [26] L. Prechelt, G. Malpohl and M. Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, Volume 8, Number 11, pages 1016–1038, November 2002.
- [27] S. Shankland. SCO sues Big Blue over Unix, Linux. *CNET News.com*, March 2003. URL: <http://news.com.com/2100-1016-991464.html> [Accessed 4 October 2007].
- [28] A. Si, H. V. Leong and R. W. H. Lau. CHECK: A document plagiarism detection system. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, pages 70–77, San Jose, California, February 1997. ACM Press.
- [29] A. Singhal, C. Buckley and M. Mitra. Pivoted document length normalization. In *Proceedings of the Nineteenth annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–29, Zurich, Switzerland, August 1996. ACM Press.
- [30] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, Volume 147, Number 1, pages 195–197, March 1981.
- [31] B. Stein, S. Meyer zu Eissen and M. Potthast. Strategies for retrieving plagiarized documents. In *Proceedings of the Thirtieth annual international ACM SIGIR conference on Research and development in information retrieval*, pages 825–826, Amsterdam, Netherlands, July 2007. ACM Press.
- [32] K. Verco and M. Wise. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. In J. Rosenberg (editor), *Proceedings of the First Australian Conference on Computer Science Education*, pages 81–88, Sydney, Australia, July 1996.
- [33] M. Wise. YAP3: Improved detection of similarities in computer program and other texts. In J. Impagliazzo, E. S. Adams and K. J. Klee (editors), *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, pages 130–134, Philadelphia, Pennsylvania, February 1996. SIGCSE: ACM Special Interest Group on Computer Science Education, ACM Press.
- [34] I. Witten, A. Moffat and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, second edition, 1999.
- [35] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, Volume 22, Number 2, pages 179–214, April 2004.
- [36] Y. Zhao and J. Zobel. Effective and scalable authorship attribution using function words. In G. G. Lee, A. Yamada, H. Meng and S. H. Myaeng (editors), *Proceedings of the Second AIRS Asian Information Retrieval Symposium*, pages 174–189, Jeju Island, Korea, October 2005. Springer.
- [37] Y. Zhao, J. Zobel and P. Vines. Using relative entropy for authorship attribution. In H. T. Ng, M.-K. Leong, M.-Y. Kan and D. Ji (editors), *Proceedings of the Third AIRS Asian Information Retrieval Symposium*, Volume 4182 of *Lecture Notes in Computer Science*, pages 92–105, Singapore, Singapore, October 2006. Springer.
- [38] J. Zobel. “Uni cheats racket”: A case study in plagiarism investigation. In R. Lister and A. Young (editors), *Proceedings of the Sixth Australasian Computing Education Conference*, Volume 30 of *Conferences in Research and Practice in Information Technology (CRPIT)*, pages 357–365, Dunedin, New Zealand, January 2004. Australian Computer Society.