Document clustering with evolved search queries

HIRSCH, Laurence <http://orcid.org/0000-0002-3589-9816> and DI NUOVO, Alessandro <http://orcid.org/0000-0003-2677-2650>

Document Clustering with Evolved Search Queries

Laurence Hirsch and Alessandro Di Nuovo
Department of Computing
Sheffield Hallam University
Sheffield, United Kingdom
l.hirsh@shu.ac.uk; a.dinuovo@shu.ac.uk

***Abstract*— Search queries define a set of documents located in a collection and can be used to rank the documents by assigning each document a score according to their closeness to the query in the multidimensional space of weighted terms. In this paper, we describe a system whereby an island model genetic algorithm (GA) creates individuals which can generate a set of Apache Lucene search queries for the purpose of text document clustering. A cluster is specified by the documents returned by a single query in the set. Each document that is included in only one of the clusters adds to the fitness of the individual and each document that is included in more than one cluster will reduce the fitness. The method can be refined by using the ranking score of each document in the fitness test. The system has a number of advantages; in particular, the final search queries are easily understood and offer a simple explanation of the clusters, meaning that an extra cluster labelling stage is not required. We describe how the GA can be used to build queries and show results for clustering on various data sets and with different query sizes. Results are also compared with clusters built using the widely used k-means algorithm.**

Keywords— text clustering, genetic algorithm, text mining

I. INTRODUCTION

Text clustering is widely used to organise, summarise and visualise large volumes of text [15], [20]. Clusters in a collection can be used to identify topics, to support user navigation, to improve information retrieval performance and accuracy and to optimise search engines. Ideally, documents are automatically clustered such that those within a cluster have a high degree of similarity in comparison with documents in other clusters. Although supervised learning is shown to have achieved high accuracy, clustering is a good alternative where labelled training documents are not available or are expensive to obtain, as is commonly the case, especially where the collection is dynamic.

For automated clustering, documents are traditionally represented by a multi-dimensional feature vector where each dimension corresponds to a weighted value of a term within the document collection [19]. Various similarity or distance measures have been proposed e.g. [1] and are a central component of most text clustering algorithms. Work has been done on alternative models which recognise word order such as using lexical chains to preserve the semantic relationships between words, for example by using WordNet [23].

In his commonly quoted paper of 1948 Alan Turing made an observation which seems quite pertinent to the key subjects covered here:

"It may be of interest to mention two other kinds of search in this connection. There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being survival value. The remarkable success of this search confirms to some extent the idea that intellectual activity consists mainly of various kinds of search" [22]

In this paper, we describe a novel system whereby an island model Genetic Algorithm (GA) is used to partition a collection of documents into clusters by generating a set of search queries. This is achieved integrating the GA with Apache Lucene, which is a widely used open source information retrieval library. We suggest that search queries are a natural and intuitive way for humans to define a set of text documents and this will improve the applicability of the document clustering.

The rest of the paper is organised as follows: background to the proposed system is discussed in section II. Section III describes the proposed system, the datasets used and the experimental setup. Section IV presents the results and section V and VI give the conclusions and suggestions for future work.

II. BACKGROUND

A. GA for text clustering

GA is a stochastic global optimisation technique which mimics the process of Darwinian evolution whereby the search for solutions is guided by the principles of selection and heredity [10]. GAs have proved to be an effective computational method, especially in situations where the search space is un-characterized (mathematically), not fully understood, or/and highly dimensional [6], such as a clustering problem [11]. Clustering a large volume of text is an example of unsupervised problems that fits all these characteristics, thus, GAs are a perfect tool to investigate in this area.

978-1-5090-4601-0/17/$31.00 ©2017 IEEE

Genetic algorithms have been used in text clustering [2]. For example, Song et al [21] have used a GA in combination with techniques based on swarm intelligence to optimise text clustering. In this case GA is used to find an optimal set of centres for text clusters. Each chromosome represents a combination of centres which represent the candidate solution to the clustering problem. Once the GA is complete the clusters with the highest fitness in the final generation are used to initialize a quantum behaved particle swarm optimization (QPSO).

B. GA for text classification

Genetic methods have been in use for some time in the area of document classification [5], [17]. We have previously described a system whereby Apache Lucene search queries were evolved from a set of training documents in order to classify documents in a collection [9]. The system uses a GA to generate a set of human understandable search queries where each query acts as a binary classifier for a single category in the collection. The search queries are evolved independently for each category.

Here, we describe an extension the GA classifier such that each chromosome generates a complete set of search queries for a collection and the system can run in an unsupervised manner. Each individual represents a set of queries and the set of documents retrieved by each query represents a cluster in the collection. Each document that is included in one of the clusters adds to the fitness of the individual and each document that is included in more than one cluster will reduce the fitness.

Finally, a recent trend in document classification is multi-label learning [7], which consists in associating different labels to the same document.

C. K-means

Several algorithms have been described for clustering data when the number of clusters is known in advance. K-means is the most widely used and attempts to solve the clustering problem into a fixed number of clusters K known in advance. It is an iterative hill-climbing algorithm and solution. K-means provides a set of clusters but does not give labels to the clusters or any human understandable indication of how the clusters are formed.

D. Cluster Labels

Once clustering is complete, it is often very helpful to provide a comprehensible label for the clusters. This can be useful for various purposes, for example in allowing a user to navigate a collection or in providing parametric search capability. As mentioned above, a clustering algorithm such as k-means defines clusters in a way which is quite opaque to humans, and so labelling is often performed once the clustering is complete. This approach has some drawbacks and a number of attempts have been made to create clusters defined in a way which is clear to a human. Frequent term-based text clustering is an attempt to provide an understandable description of the discovered clusters by their frequent term sets. Frequent term sets are sets of terms co-occurring in more than a threshold percentage of all documents of a database. An important advantage is that each cluster can be labelled by the obtained frequent term set shared by documents in the same cluster [3]. An alternate approach was taken by Rafi et al. [18] who introduced a new document representation model based on the compact topic maps that are present in a document.

E. Apache Lucene

Apache Lucene is a high-performance full text indexing and searching software library written in Java. Lucene provides a wide variety of query representations, as well as several query parsers and a query parsing framework to assist developers in writing their own query parser.

Evolutionary computation is notoriously resource intensive and we find that using Lucene to store and query the document collections is a significant boost to the performance of the system. In our work, we use the Lucene scoring system to rank documents as part of the fitness function calculation; we give a brief description below. The interested reader should refer to the Lucene documentation for a full explanation[1].

For documents scoring, Lucene combines the "Boolean model (BM) of Information Retrieval" with the "Vector Space Model (VSM) of Information Retrieval" [14] so that documents "approved" by BM are then scored by VSM. Lucene refines the VSM score for both search quality and usability applying some transformations, such as a document length normalization factor that normalizes to a vector equal to or larger than the unit vector. A document may match a multi term query without containing all the terms of that query.

For efficient score computation, some scoring components are computed and aggregated in advance. Lucene's Practical Scoring Function is:

$$score(q,d) = coord(q,d) \cdot \sum_{t\ in\ q} (tf(t\ in\ d) \cdot idf(t)^2 \cdot norm(t,d))$$

where q is the query to score, d is the document under consideration and t are the terms in the query.

1. ***coord(q,d)*** is a score factor based on how many of the query terms are found in the specified document. Typically, a document that contains more of the query's terms will receive a higher score than another document with fewer query terms.
2. ***queryNorm(q)*** is a normalizing factor used to make scores between queries comparable. This factor does not affect document ranking
3. ***tf(t in d)*** is defined as the number of times the term *t* appears in the currently scored document *d*. Documents that have more occurrences of a given term receive a higher score.
4. ***idf(t)*** stands for Inverse Document Frequency. This value is defined as the number of documents in which the term *t* appears). This means rarer terms give higher contribution to the total score.

[1] http://lucene.apache.org/core/6_3_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

5. ***norm(t,d)*** is another normalizing factor that encapsulates boost and length factors: **Field boost** is not used in this work as only one field containing the document text is available to the GA; **lengthNorm** - normalizes by the number of words in the document.

III. Material and Methods

A. Data Sets

We run our algorithm on three labelled collections to evaluate the effectiveness of the clustering. Of course, the labels are not used in the clustering process but only to assess the results. We also provide results for a standard implementation of the k-means algorithm for comparison purposes. The type of data in the three sets is quite varied coming from an academic source, a newsgroup and twitter data.

1) Classic 4 Collection (name: classic4)

The dataset contains 7095 documents, where each document belongs to one of the four distinct collections: CACM (titles and abstracts from the journal Communications of the Association of Computing Machinery), CISI (information retrieval papers), CRANFIELD (aeronautical system papers), MEDLINE (medical journals). We take the commonly used approach (e.g. [4]) and randomly select 500 documents from each category for our experimental analysis.

2) 20 Newsgroup (name: 20NG5)

The 20 Newsgroups collection, set up by Lang [12] The documents are messages posted to Usenet newsgroups, and the categories are the newsgroups themselves. The data on this set is considered particularly noisy and as might be expected does include complications such as duplicate entries and cross postings. We construct a 500 document subset of the 20 Newsgroup dataset in the same way as Song et al. [21] by randomly taking 100 documents from five categories (comp.os.ms-windows.misc, misc.forsale, rec.sport.hockey, sci.space, soc.religion.christian).

3) Crisis Data (name: Crisis3)

CrisisLex.org is a repository of crisis-related social media data and tools [16]. The 'CrisisLexT6' collection[2] contains tweets collected in 2012-13 in different crisis situations. We use 1000 of the tweets from each of the three of the sets: Colorado wildfires, Boston bombings and Queensland floods.

B. Method

The GA clustering system presented in this paper works by implementing the steps shown below. We include a small example based on the Classic4 dataset to support the explanation.

1) Evolution

Step 1: Pre-processing

Before we start the evolution of classification queries a few pre-processing steps are made.

1. All the text is placed in lower case.
2. A small stop set is used to remove common words with little semantic weight.
3. For each dataset, a Lucene index is constructed and each document labelled (using Lucene fields) according to its category (only used to evaluate the effectiveness of the clustering).
4. Set Parameters. For this example we may set Clusters: 4, Query length: 2, Chromosome Length: 8, Word Set Size: 8, Data Type: int in range -1 : 7 (-1 indicates empty). Longer chromosomes will allow more words to be used in the queries.

Step 2: Create an ordered list of words with integer indices.

The list is used by the GA for building queries. The *tf*idf* value for each term in the collection is calculated by summing the value in each document. Words are then sorted by their overall *tf*idf* value and the top N words are selected where N is a parameter set by the user of the system; 8 in this example but 100 in the results described below. The integer index is simply the words place in the *tf*idf* ordering.

TABLE I. EXAMPLE WORD LIST (CLASIC4)

Index	Value
0	flow
1	information
2	library
3	pressure
4	Cell
5	boundry
6	patients
7	algorithm

Step 3: Create generation 0.

Table II shows an example chromosome from the population of generation 0 and the way in which the representation forms clusters, each cluster being defined by a search query formed from up to two words.

TABLE II. EXAMPLE CHROMOSOME TO CREATE 4 SEARCH QUERIES (SQ)

Cluster	SQ0		SQ1		SQ2		SQ3	
Chromosome	6	3	0	5	4	-1	2	3

Step 4: Create a set of search queries from each individual.

In this case, we need 4 queries so we break the chromosome into 4 groups (elements 0:1, 2:3, 4:5, 5:6). The `int` value from the chromosome is used as the index to word

[2] available from http://cri sislex.org/data-collections.html

map (-1 indicates no word). We can then create a Lucene Boolean search query by joining the words in each cluster with a Boolean OR.

Example search queries (SQ) for the Classic4 dataset:

SQ0: patients OR pressure
SQ1: flow OR boundary
SQ2: cell
SQ3: library OR pressure

SQ0 will return all documents in the dataset which contain either the word 'patients' OR the word 'pressure'. This is likely to produce poor clustering as measured by F1 since it will retrieve document in both the aeronautics (cran) and medical (medline) topic areas. On the other hand, the next two queries could be useful for building clusters since SQ1 is likely to retrieve documents mainly from the cran set and SQ2 is likely to retrieve documents mainly from the medline set. SQ3 is likely to retrieve documents from both cran and cisi.

Step 5

For each member of the population fire the four queries produced and determine its fitness by examining the set of documents returned by the queries.

Step 6

Apply genetic operators to create a new generation.

Step 7

Repeat steps 4-6 until the termination criteria are reached (1000 generations).

A GA contains many random elements. We therefore repeat each run 21 times and select the individual with the highest fitness.

2) Fitness calculation

Text clustering aims to return sets of documents which are related to each other but not to documents in other clusters. In its simplest form the fitness (f) of an individual could be:

$$f = \frac{count\ docs\ returned\ by\ only\ one\ query}{count\ docs\ returned\ by\ more\ than\ one\ query}$$

However, we have found it useful to include the ranking of results provided by Lucene in the fitness function both in directing the evolution and in improving the final accuracy of the results. Ranking is achieved in Lucene by assigning each document returned by a query (known as a 'hit') a score and then ordering the documents returned by their score.

We have also found it useful to introduce penalties into the fitness test for certain conditions:

1. An empty query (i.e. where all the genes were -1)
2. A 'core cluster penalty' occurs when a document is returned in the first 20 hits that is also returned by another query in the set.

The basic fitness is calculated by running each of the queries from an individual and calculating the sum of the scores of documents returned by only one query minus the sum of the scores of documents returned by more than one query. This number is then divided by the total negative indicators (such as a core cluster penalty).

The Groovy style pseudo code for the fitness test is shown below:

```
//assuming k = 4 each chromosome will
//generate 4 queries
def queries = [q0, q1, q2, q3]

def totalScore=0, penalty=1

queries.each{ q ->
  if (q.isEmpty()) penalty++

//hits is the collection of documents
//returned by the query, ordered by
//relevance
  def hits=searcher.search(q)

//iterate hits in order of the
//documents (d) score
//the score is provided by Lucene
//based on the closeness of the query
//to the document

  hits.each { d ->
    if (d.isReturnedByAnotherQuery()){
      if (hitnumber < 20) {
        penalty++
      }
      totalScore=totalScore - d.score
    }
    else {
      totalScore=totalScore + d.score
    }
}

fitness=totalScore / penalty
```

3) GA Parameters

We used a fixed set of standard GA parameters in all our experiments which are summarised in Table III. An individual is selected according to fitness and can be simply copied into the then next generation (reproduction) or part of the chromosome may be randomly changed (mutation) or most commonly parts of the chromosome are exchanged with another selected individual to create two new individuals (crossover). The probabilities of these 3 possibilities are determined by the parameters in Table III.

Subpopulations can be used as a method of increasing diversity in the population and it is possible to maintain more than one best individual [12]. Only limited communication

(immigration/emigration) is allowed between subpopulations. In our case, we implemented emigration exchanging 3 individuals between the subpopulations every 50 generations.

TABLE III. GA PARAMETERS

WordList Size	100
Chromosome int values	-1 : 100 (-1 indicates no query term)
Selection type	tournament
Subpopulations	3
Population size	512
Generations	1000
Crossover probability	0.8
Mutation probability	0.1
Engine	ECJ 23 [13]

C. Effectiveness Measures

In our analysis, we consider the three widely adopted measures of clustering effectiveness. These include *precision (p)* (also known as *positive predictive value*), which is defined as the portion of relevant instances that are retrieved, and *recall (r)* (also labelled as *sensitivity*), which is the share of relevant instances retrieved by the algorithm. Finally, the main measure we use to determine cluster effectiveness is the F1 score [2], which computes both the precision and the recall to measure the overall accuracy. The F1 measure has the advantage of giving equal weight to precision and recall and is given by:

$$F1 = \frac{2pr}{p+r}$$

IV. RESULTS

In this section, we first present the results for the GA Clustering system together with those obtained by the Bisecting K-Means algorithm in order to compare the effectiveness of the proposed query based approach against a standard clustering algorithm used in information retrieval.

In our experiments, for each dataset, we run 21 times the GA with different random initialization and the parameters in Table III. On average, one run of the GA required 183.19 seconds on an Intel i7 processor at 3.0GHz. We provide an analysis of the different runs in subsection B, which show that they usually converge to similar results with a low standard deviation.

A. Effectiveness Analysis

In Tables from IV to IX, *p* indicates precision and *r* indicates recall. Tables IV, VI and VIII reports the queries of the best solution found by the GA for each dataset.

The results tables indicate that the GA evolved queries perform well on the datasets and that accuracy is constantly higher when compared to bisecting k-means. This is because the higher precision of the evolved queries with respect to the k-means clustering whilst the recall rate is comparable between the two approaches. We should note that some configurations of k-means can improve upon the results reported here, e.g. applying [4], but also that cluster accuracy is not the only goal for the GA search query method. In fact, our approach focuses on the readability aspect of the clustering which allows a human analyst to interpret the results and provides the possibility of further refinement.

TABLE IV. CLASSIC4 : GA CLUSTERING (CHROMOSOME SIZE: 12, MAX QUERY SIZE:3)

Category	p	r	F1	Query
Cacm	0.97	0.46	0.63	programming OR algorithm
Cisi	0.75	0.66	0.70	library OR information
Cran	0.90	0.56	0.69	pressure OR boundary
Med	0.86	0.49	0.63	cells OR treatment OR patients
Average	0.87	0.54	**0.66**	

TABLE V. CLASSIC4 BISECTING K-MEANS

Category	p	r	F1
Cacm	0.55	0.68	0.61
Cisi	0.68	0.33	0.45
Cran	0.64	0.92	0.76
Med	0.69	0.60	0.64
Average	0.64	0.63	**0.61**

TABLE VI. 20NG5: GA CLUSTERING (CHROMOSOME SIZE: 15, MAX QUERY SIZE:3)

Category	p	r	F1	Query
Windows	0.94	0.66	0.78	windows
Forsale	0.95	0.57	0.71	sale
Hockey	0.97	0.74	0.84	players OR hockey OR nhl[3]
Space	0.94	0.47	0.63	orbit OR station
Christian	0.96	0.74	0.84	jesus OR god OR christ
Average	0.95	0.64	**0.76**	

TABLE VII. 20NG5 BISECTING K-MEANS

Category	p	r	F1
Windows	0.73	0.66	0.69
Forsale	0.42	0.55	0.48
Hockey	0.70	0.82	0.76
Space	0.87	0.69	0.77
Christian	0.78	0.65	0.71
Average	0.7	0.67	**0.68**

TABLE VIII. CRISIS3: GA CLUSTERING (CHROMOSOME SIZE: 15, MAX QUERY SIZE:5

Category	p	r	F1	Query
Queensland floods	0.85	0.81	0.83	floods OR highparkfire OR flooding OR qldfloods OR bigwet
Colorado wildfire	0.98	0.7	0.82	springs OR fire OR colorado
Boston bombing	0.99	0.81	0.9	boston OR bostonmarathon marathon OR suspect OR prayforboston
Average	0.9	0.78	**0.85**	

TABLE IX. CRISIS3 BISECTING K-MEANS

Category	p	r	F1
Queensland floods	0.5	0.62	0.55
Colorado wildfire	0.7	0.7	0.7
Boston bombing	0.52	0.36	0.43
Average	0.6	0.56	**0.56**

The results tables indicate that the GA clustering performs well on the datasets and that precision in particular is generally high when compared to k-Means.

B. Evolution analysis

Fig. 1 – Fig. 4 report the median values for the best fitness score and the F1 score across the 21 runs for each dataset. As we would expect the fitness and F1 values increase rapidly during the early stages of evolution and mostly stabilize toward the end of the run. We should note that individuals in the initial generations normally have their fitness score quite heavily influenced by penalties such as the core cluster penalty described above whilst individuals generating penalties are generally selected out of the population well before the end of a run. Although the standard deviation can be quite large at the start of a run the values tail off and for each database are quite small by the end of the 1000 generations. Standard deviation on Crisis3 would be higher at the end because two runs totally

[3] National hockey league

failed to find a reasonable solution getting an F1 below 0.2. This demonstrates the importance of using multiple runs.

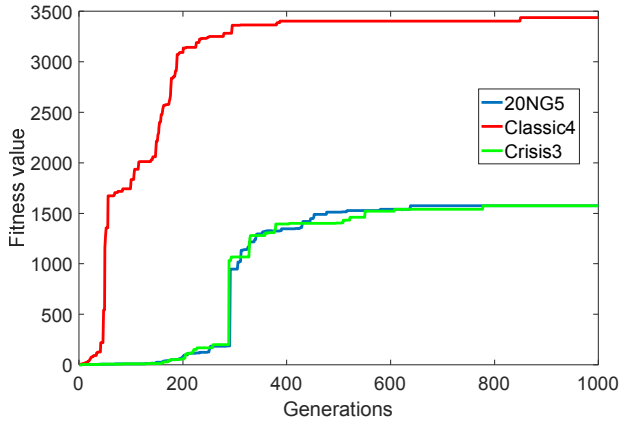

Fig. 1. Median Fitness value by generations

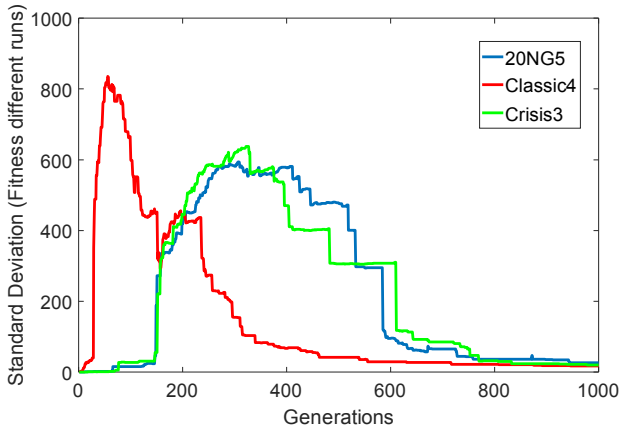

Fig. 2. Standard deviation of total fitness by generation

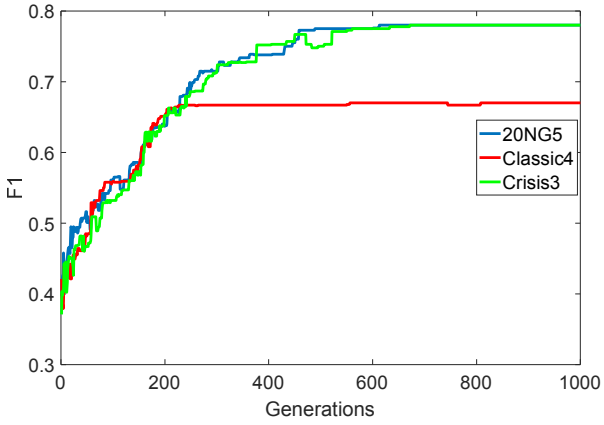

Fig. 3. Median F1 score by generation

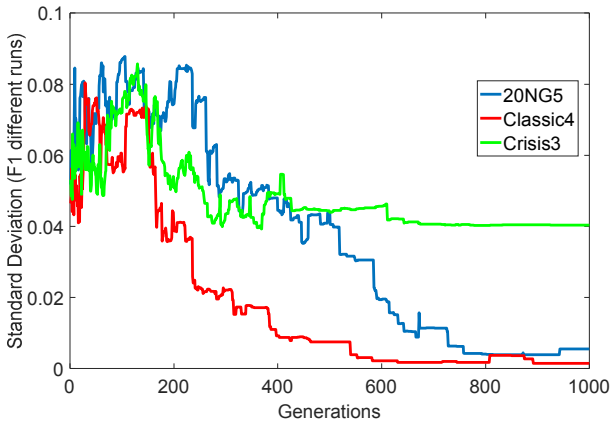

Fig. 4. Standard deviation of F1 by generation

V. Conclusion

We have described the GA clustering algorithm and presented results on three data sets. As far as we are aware this is the first attempt to provide text document clustering using automatically generated search queries. Whilst the results appear to compare favourably to a standard k-means algorithm we do not argue that this method is the most accurate clustering algorithm (see [2]) but rather that the *quality* of the search query format may have significant benefits when compared to previously described methods. A number of advantages to the search query format are listed below:

- Search queries are easily understood by a human and act as labels for each cluster. The labels are a direct representation of the clusters and can be refined by a human if required.
- Search queries explain the clusters and define how to create clusters.
- When created automatically clusters can be generated in unusual ways which are not useful to humans. Search query labels allow for a 'sanity check' of the clusters generated.
- Search queries will scale (e.g. to the size of the web).
- Running the query for a particular cluster returns a ranked list of documents where those near the top of the list are near the centroid of the cluster.
- Clustering can be seen as a search for the cluster centres. Centres can be points in a multi-dimensional space or actual documents in the space. Here we use simple, human understandable search queries as cluster centres.

VI. Future Work

All the search queries generated here use a simple OR between terms. We are experimenting with alternate query formats that can improve accuracy without altering the readability of the queries. Although we believe that simple AND queries are unlikely to be useful AND with OR (i.e.

disjunctive normal form) may be more promising. Including a NOT in the query is also being considered. We will also further examine the effect of chromosome length and other parameters.

In text clustering problems, it is often the case that the ideal number of clusters is not known in advance. We are experimenting with introducing an extra gene in the chromosome used to specify the number of clusters. In this paper, we have only dealt with the simple case where a document can only belong to one cluster. We will explore the possibility of extending the system to allow for multi-label text clustering by using fuzzy-logic clustering.

REFERENCES

[1] Abraham, A., Das, S., & Konar, A. (2006). Document clustering using differential evolution. In Proceedings of IEEE congress on evolutionary computation (pp. 1784–1791).

[2] Aggarwal, C. C., & Zhai, C. (2012). A survey of text clustering algorithms. In *Mining text data* (pp. 77-128). Springer US.

[3] Beil, F., Ester, M., & Xu, X. (2002, July). Frequent term-based text clustering. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 436-442). ACM.

[4] Bharti, K. K., & Singh, P. K. (2015). Hybrid dimension reduction by integrating feature selection with feature extraction method for text clustering. *Expert Systems with Applications*, *42*(6), 3105-3114.

[5] Clack, C., Farringdon, J., Lidwell, P., & Yu, T. (1997, February). Autonomous document classification for business. In *Proceedings of the first international conference on Autonomous agents* (pp. 201-208). ACM.

[6] Di Nuovo, A. G., & Catania, V. (2008). An evolutionary fuzzy c-means approach for clustering of bio-informatics databases. In Proceedigns of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 2077-2082, IEEE.Press.

[7] Gibaja, E., & Ventura, S. (2015). A tutorial on multilabel learning. ACM Computing Surveys (CSUR), 47(3), 52.

[8] Gong, Y. J., Chen, W. N., Zhan, Z. H., Zhang, J., Li, Y., Zhang, Q., & Li, J. J. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, *34*, 286-300.

[9] Hirsch, Laurence (2010). Evolved Apache Lucene SpanFirst Queries are Good Text Classifiers. In: *IEEE Congress on Evolutionary Computation, 2009. CEC '09*. pp 1-8, IEEE Press.

[10] Holland, J. H. (1992). Genetic algorithms. Scientific american, 267(1), pp 66-72.

[11] Hruschka, E. R., Campello, R. J., & Freitas, A. A. (2009). A survey of evolutionary algorithms for clustering. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 39(2), pp 133-155.

[12] Lang, K. (1995, July). Newsweeder: Learning to filter netnews. In Proceedings of the 12th international conference on machine learning (pp. 331-339).

[13] Luke S, Panait L, Balan G, Paus S, Skolicki Z, Bassett J, Hubley R, Chircop A (2006) ECJ: a Java based evolutionary computation research system. http://cs.gmu.edu/eclab/projects/ecj

[14] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval (Vol. 1, No. 1, p. 496). Cambridge: Cambridge university press.

[15] Muhammad Rafi, M. Shahid Shaikh, Amir Farooq, "Document Clustering based on Topic Maps," International Journal of Computer Applications, Vol. 12– No.1, Dec. 2010

[16] Olteanu, OlteA., Vieweg, S., & Castillo, C. (2015, February). What to expect when the unexpected happens: Social media communications across crises. In Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (pp. 994-1009). ACM.

[17] Pietramala, A., Policicchio, V. L., Rullo, P., & Sidhu, I. (2008, September). A genetic algorithm for text classification rule induction. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 188-203). Springer Berlin Heidelberg.

[18] Rafi, M., Shaikh, M. S., & Farooq, A. (2011). Document clustering based on topic maps. arXiv preprint arXiv:1112.6219.

[19] Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24(5), 513–523.

[20] Shah, N. and Mahajan, S., 2012. Document Clustering: A Detailed Review. *International Journal of Applied Information Systems*, *4*(5), pp.30-38.

[21] Song, W., Qiao, Y., Park, S. C., & Qian, X. (2015). A hybrid evolutionary computation approach with its application for optimizing text document clustering. *Expert Systems with Applications*, *42*(5), 2517-2524.

[22] Turing, A. M. (1948). Intelligent machinery, a heretical theory. The Turing Test: Verbal Behavior as the Hallmark of Intelligence, 105.

[23] Wei, T., Lu, Y., Chang, H., Zhou, Q., & Bao, X. (2015). A semantic approach for text clustering using WordNet and lexical chains. *Expert Systems with Applications*, *42*(4), 2264-2275.