Universidade Estadual de Campinas
Instituto de Computação

João Luís Villar de Oliveira

Improved Knowledge Sharing in Ticket Repositories: a Genetic Algorithm Approach for Feature Selection

Melhorando o Compartilhamento de Conhecimento em repositórios de Tickets: uma abordagem de Algoritmo Genético para Seleção de Características

CAMPINAS
2020

João Luís Villar de Oliveira

Improved Knowledge Sharing in Ticket Repositories: a Genetic Algorithm Approach for Feature Selection

Melhorando o Compartilhamento de Conhecimento em repositórios de Tickets: uma abordagem de Algoritmo Genético para Seleção de Características

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Mario Lúcio Côrtes
Co-supervisor/Coorientador: Prof. Dr. Ricardo da Silva Torres

Este exemplar corresponde à versão final da Dissertação defendida por João Luís Villar de Oliveira e orientada pelo Prof. Dr. Mario Lúcio Côrtes.

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

OL4i Oliveira, João Luís Villar de, 1988-
Improved knowledge sharing in ticket repositories : a genetic algorithm approach for feature selection / João Luís Villar de Oliveira. – Campinas, SP : [s.n.], 2020.

Orientador: Mário Lucio Côrtes.
Coorientador: Ricardo da Silva Torres.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Recuperação da informação. 2. Seleção de características. 3. Algoritmos genéticos. 4. Sistemas de recomendação (Filtragem da informação). 5. Mineração de repositórios de software. I. Côrtes, Mário Lucio, 1950-. II. Torres, Ricardo da Silva, 1977-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Melhorando o compartilhamento de conhecimento em repositórios de tickets : uma abordagem de algoritmo genético para seleção de características
Palavras-chave em inglês:
Information retrieval
Feature selection
Genetic algorithms
Recommender systems (Information filtering)
Mining software repositories
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Mário Lucio Côrtes [Orientador]
Adler Diniz de Souza
Islene Calciolari Garcia
Data de defesa: 30-04-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-0958-2225
- Currículo Lattes do autor: http://lattes.cnpq.br/1760760660946934

Universidade Estadual de Campinas
Instituto de Computação

João Luís Villar de Oliveira

Improved Knowledge Sharing in Ticket Repositories: a Genetic Algorithm Approach for Feature Selection

Melhorando o Compartilhamento de Conhecimento em repositórios de Tickets: uma abordagem de Algoritmo Genético para Seleção de Características

Campinas, 30 de abril de 2020

Agradecimentos

Essa dissertação de mestrado não poderia ser concluída sem o apoio de várias pessoas.

Em primeiro lugar, não posso deixar de agradecer aos meus orientadores, Professor Doutor Mario Lúcio Côrtes e Professor Doutor Ricardo da Silva Torres, por toda a paciência, empenho e orientação pautada por um elevado nível científico em todas as etapas subjacentes ao trabalho realizado, cujos esforços e auxílio tornaram possível a concretização deste projeto.

Desejo igualmente agradecer aos meus pais e minhas avós pelo apoio, motivação e conselhos preciosos. A minha esposa, meus irmãos e aos meus tios por desejarem sempre o melhor para mim. A vocês, minha família, sou grato por todo apoio, carinho, atenção e felicidade que eu tenho.

Por último, o agradecimento mais importante: Agradeço a Deus por me abençoar, me iluminar, me guiar; sem Ti eu nada poderia fazer.

Obrigado a todos!

Resumo

Gestão do conhecimento é essencial para qualquer organização. As empresas precisam ser capazes de produzir, organizar, salvar, recuperar e compartilhar conhecimento com eficiência. As empresas de software não são diferentes de outras empresas, produzindo conhecimento continuamente em larga escala. Essas empresas geralmente geram centenas de milhares de tickets por dia por meio de ferramentas de gerenciamento de projetos, como Jira, Bugzilla e Trello. Neste contexto, o compartilhamento apropriado de conhecimento sem automação é extremamente difícil ou até impossível. A chance de perder informações valiosas ao longo do tempo é alta. Para solucionar este problema, desenvolvemos um sistema de recomendação em que, para cada novo ticket, o sistema analisa e classifica os tickets relevantes no repositório para ajudar os usuários compartilhando conhecimentos e lições aprendidas em casos semelhantes anteriores. O sistema de recomendação é baseado no *Vector Space Model (VSM)* com TF-IDF e medida de similaridade cosseno. Os experimentos revelaram resultados positivos, mostrando que o VSM é uma técnica adequada para classificar tickets relevantes. Em combinação com o sistema de recomendação, foi desenvolvida uma técnica de seleção de características baseada no algoritmo genético e uma nova abordagem que o aprimora heuristicamente. O sistema de recomendação com seleção de características obteve resultados positivos (todos os resultados foram validados estatisticamente) em comparação ao método que se baseia no modelo tradicional de espaço vetorial, sem nenhum procedimento de seleção de características. A precisão aumentou em quase 23%, o tempo médio de pesquisa por consulta reduziu em 44% e o tamanho do vetor reduziu em 64%.

Abstract

Knowledge management is essential for any organization. Corporations need to be capable of producing, organizing, saving, retrieving, and sharing knowledge efficiently. Software corporations are not different from any other businesses, producing knowledge continuously on a large scale. Such companies often generate hundreds of thousands of tickets per day through project management tools, such as Jira, Bugzilla, and Trello. In this context, appropriate knowledge sharing without automation is extremely difficult or even impossible. The chance of losing valuable information through time is high. To address this issue, we have developed a recommendation system where, for each new ticket, the system analyses and ranks relevant tickets on the repository to help users to share knowledge and lessons learned from previous similar cases. The recommendation system is based on the Vector Space Model (VSM) with TF-IDF and cosine similarity. The experiments revealed positive results showing that the VSM is a suitable technique to rank relevant tickets. In combination with the VSM recommendation system, a feature selection technique based on Genetic Algorithm, and a novel approach that heuristically improves the well-known Genetic Algorithm were developed. The recommendation system with feature selection had positive results, where all results were statistically validated, compared with the method that relies on the traditional vector space model without any feature selection procedure. The precision increased by almost 23%, the average search time by query reduced by almost 44%, and the vector size reduced by 64%.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Knowledge management is essential for any organization. Nonaka and Takeuchi [44], for example, analyzed how Japanese corporations became competitive globally by creating environments that promoted knowledge sharing. Peter Senge [60], in turn, pointed out that organizations without proper knowledge management tend to focus on treating symptoms instead of fixing problems. Menolli et al. [38] claimed that companies should build conditions that motivate information sharing and knowledge dissemination; otherwise, valuable information could be internalized only on the mind of the employees. All these studies show that disseminating knowledge is a fundamental competence for an organization in general, particularly software companies [21, 58].

Software companies are composed of multiple knowledge-intensive activities as defining requirements, testing, analyzing, learning the applied technology, and others [4]. Software development teams need valuable project knowledge to deliver the work with quality [20]. Furthermore, knowledge shared needs to be precise, easy, and fast to find, primarily due to fierce competition and work quality.

Software corporations strive to stay competitive due to intense competition and pressure on the market [42, 50], forcing them to have a more efficient development process for all costs [18]. This is especially true in large organizations and open source projects that have software development spread on teams all over the world. One of the most efficient and flexible processes applied nowadays is the Agile Software Development (ASD) methods [16]. However, it comes with a high cost of prioritizing Knowledge Sharing (KS) by face-to-face or chat communication than through written documentation, significantly reducing the quality and the number of written documentation leading to losing essential project knowledge over time [3, 15, 46]. Further, most of the large software organizations have distributed teams all over the world, making KS even harder [11, 20]. Consequently, be capable of balancing KS between human interaction and written documentation without impact the flexibility of ASD is a requirement.

ASD relies on team collaboration, communication, and creativity instead of extensive documentation to guide the development [15], mostly because of the Agile Principals [22] that focus on human interaction than written documentation [9]. This change of paradigm from traditional software development brought new challenges for managing knowledge. The most crucial obstacles are the lack of time to write down the knowledge acquired through the interactions and time to find the relevant knowledge when it is needed, as most knowledge does not exist in writing [20, 45, 54].

These challenges became more relevant in large organizations and open source projects, as knowledge is not centralized and spread in teams around the world [20]. Distributed teams have difficulty in sharing knowledge due to communication barriers between team members in different locations [20], and for successful software development, precise communication is crucial [49]. Effective knowledge sharing on these projects is essential to improve productivity and be competitive on the market [49].

It is known that conventional knowledge management techniques are challenging to apply to ASD. Relevant information is always unshared and belongs to the team members, even though that could be useful for other projects. Agile methods stimulate significant knowledge sharing through face-to-face interactions [11]. Agile teams carry out the software development activities through effective communication and customer collaboration. Team members frequently share sophisticated and context-specific knowledge that is essential to deliver business value to the client [6, 36]. From this perspective, KS in ASD teams is imperative for the success of Agile Projects [37].

Large software organizations, such as Google, IBM, Facebook, and Amazon, are using ticket repositories, such as Jira, Bugzilla, and Trello to manage all their projects with written knowledge (e.g., requirements, defects, and user stories) [20]. These corporations can generate hundreds of thousands of tickets per day; the risk of losing relevant knowledge is highly likely. Be capable of creating, acquiring, integrating, implementing, and disseminating knowledge in this environment without automation is extremely difficult.

Any automation solution could cover the full KM process or focus on just one or more steps. By analyzing possible techniques for automation, we found that applying some Information Retrieval (IR) methods could deliver a possible automated solution. IR deals with the representation, storage, organization, and how to access information [5], this is most of the full KM process. A technique that we identify that is highly applicable in our context is a recommendation framework. A reliable recommendation system is accountable to integrate, implementing, and disseminating knowledge automatically and with high precision [5]. In our context, a recommendation system is a software component that rate documents on a dataset and find a subset of documents that is interesting for a specific user [47]. A recommendation system assists a user in finding a set of items in a collection that is of interest [74].

This dissertation suggests a recommendation system to address the demand for automation to support the knowledge management of written project information described above. The proposed recommendation system disseminates relevant ticket information when it is needed, helping the users to discover helpful information on other tickets related to their assigned ticket. The ticket recommender is a text-based recommendation system, in which to suggest tickets, the system analyzes and ranks relevant tickets on the repository automatically, helping to share the knowledge among the users effortlessly.

A typical method for building a text recommendation system is implementing the widely-used Vector Space Model (VSM) using TF-IDF weights with cosine similarity [55, 56], which is a well-known technique for text classification and retrieval. However, this solution is inherently costly as it deals with high-dimensional feature vectors (proportional to the number of terms). In this work, we also propose solutions that use a VSM recommendation system in combination with a feature selection technique based on Genetic Algorithm (GA), a potentially less costly alternative to the first solution.

GA is a well-known non-deterministic algorithm that has been used for feature selection in different text classification tasks [19]. It has been proved that GA improves the effectiveness of some text classifiers [1, 8, 19, 23], as well as other areas of studies, as content-based image classifiers [64], neural network design [71], the discovery of relevant genes on microarray datasets related to one cancer disease [59], and others. The use of GA in our work is motivated by those outstanding results.

This work also investigates which set of attributes of a ticket are more effective to recommend relevant tickets, validates if GA is an effective approach to select suitable features to support the recommendation of change requests, and analyzes the impact of reducing the feature vector size on the overall performance of the recommendation system. Also, to increase the search area between the first individuals to reduce the probability of the GA resulting on a local maximum (the highest fitness function value), this dissertation presents a novel approach as an improvement built on GA by applying a heuristic algorithm based on Hamming distance to generate the first generation of the feature vector population. Furthermore, we investigate the proposed GA with the heuristic approach for the first generation is stable and robust for different datasets.

Research questions (RQ) were established to guide the development of the proposed solutions and investigations. The defined RQs are:

- RQ1: Which attributes are most effective to recommend relevant tickets?
- RQ2: Is Genetic Algorithm a suitable approach for selecting appropriate terms and then improving the effectiveness of ticket recommendation system?
- RQ3: Does the GA approach based on the Hamming distance heuristic algorithm for the first generation for selecting appropriate terms improves the effectiveness and the efficiency of the ticket recommendation system?
- RQ4: How stable is the heuristic GA approach for selecting appropriate terms?
- RQ5: Is the heuristic GA-based feature selection approach robust for different datasets?

Experiments were designed, executed, and analyzed to address each of these RQs. This document is organized as follows. Chapter 2 presents background concepts of knowledge management, information retrieval, Genetic Algorithm, and introduces relevant related work. Chapter 3 describes each of the proposed solutions, the recommendation system with and without feature selection, the Genetic Algorithm experiment design, and the Hamming distance heuristic algorithm for the GA first generation. Chapter 4 describes the adopted experimental protocol, the dataset, the conventional ticket recommendation system, the GA parameters, evaluation metrics, and the applied statistical tests. Chapter 5 presents the experimental results and analyzes obtained results. Chapter 6 presents our conclusions and points out possible future work.

Chapter 2

Basic Concepts

This chapter presents the background concepts related to techniques used to develop the proposed recommendation system, as well as discusses related work.

2.1 Knowledge Management

Knowledge management is the process of capturing, developing, and sharing knowledge [17]. Different frameworks categorize the types of knowledge in different ways. A well-known framework is the Knowledge Spiral created by Nonaka & Takeuchi [44] that classifies the types of knowledge in two categories, tacit and explicit knowledge. The framework process is divided into four steps and is based on the interaction and relationship between these two types of knowledge, as shown in Figure 2.1.

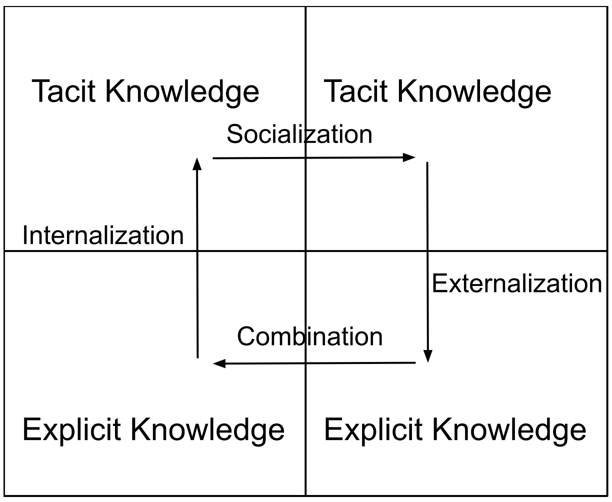


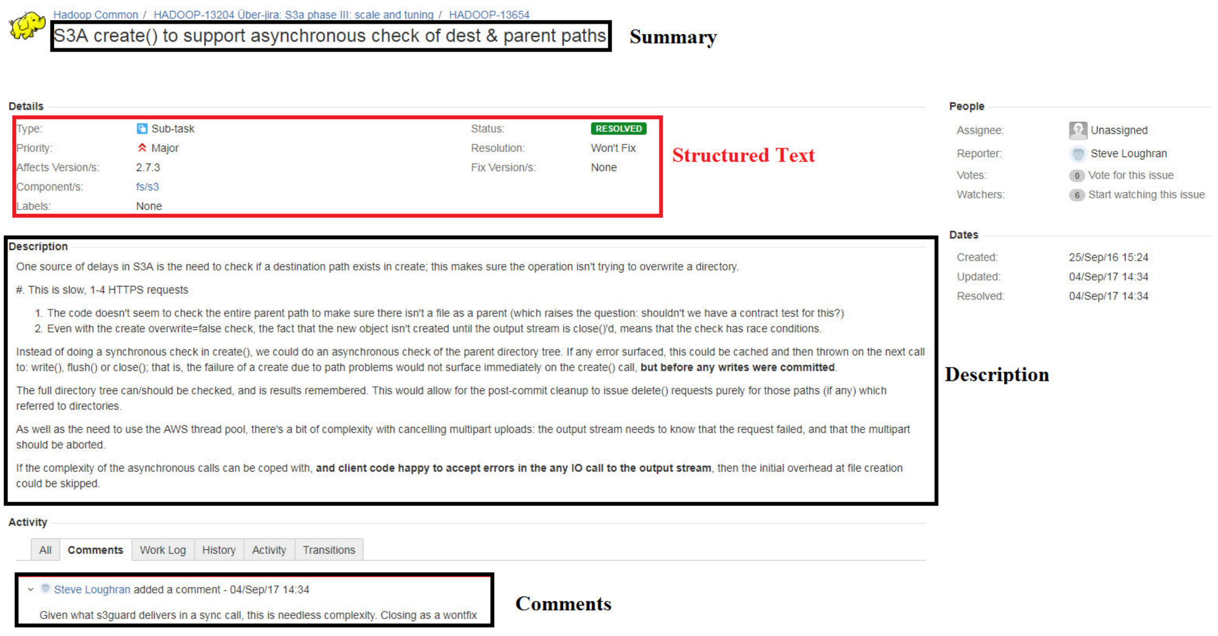
Figure 2.1: Knowledge Spiral Framework.

Tacit knowledge is an internalized knowledge that a person may not be consciously aware of, such singular tasks that the individual does without notice [12]. A good example, given by Peter Senge [60], is when a baker student is trying to learn how to prepare a dough with the head baker. The head baker will show how to stretch and twist the dough, but the precise knowledge about how much strength should be applied during the process for the dough be perfect is consciously unknown by the head. However, unconsciously the head knows the necessary strength, and because of that, the head's dough is better. As explained in the example, tacit knowledge is typically acquired by the experience and not through written information. That is why it is so hard to be shared among others. On the other hand, explicit knowledge the individual holds consciously in a form that can be easily communicated to others, and the knowledge can be transferred in many ways [12], such as written documents, teaching, or speaking. That is why explicit knowledge is more natural to acquire and share than tacit knowledge.

The focus of the Knowledge Spiral Framework is to make the company generate knowledge unconsciously through the process of sharing and transforming knowledge. The first step is called *Socialization*, where the organization incentives individuals to share tacit knowledge through imitation and observation. When the experience is all shared, the second step is to *Externalize*, to make the unconscious knowledge consciously, and the most important to document knowledge. The next step is to perform the *Combination* of the different explicit knowledge to identify and create new knowledge. The final step is to *Internalize* the new knowledge created in the previous step. In this process, new knowledge should be so ordinary on the company that the knowledge became tacit, where individuals know unconsciously [44].

In software projects, the Knowledge Spiral framework is applied every day, especially on Agile projects [46]. Dorairaj [20] and Santos [58], for example, investigated how teams collect, store, and share knowledge. Both studies share the same view: software development is a suitable environment for tacit knowledge sharing among teams, mainly because of the focus on face-to-face team interaction and pair programming. These activities identified in the studies are examples of *Socialization*. Tools as ticket repositories, forums, wiki pages, and chats are used to *Externalize* the knowledge. The *Combination* happens when developers search for knowledge on the repositories or through training. By the end, *Internalization* occurs when insights naturally happen when the developer is searching for relevant knowledge to help. However, knowledge sharing in a software project is not well understood, and uncertainty is a standard. Uncertainty is broadly defined as the absence of complete information [43] and linked to the inability of accurate predictions [40]. Uncertainties have a significant impact on project performance because of incomplete information that can lead to costly delays, redundant work, and other inefficiencies [21]. Uncertainty is one of the main problems in the software industry. Some uncertainty that exists in a software development environment is the limited knowledge of the application domain distributed across the actors involved in the project [63].

Usually, software organizations use ticket repositories tools like Jira, Bugzilla, Trello, and others to save and share all their project written knowledge (e.g., requirements, defects, and user stories) [20]. Figure 2.2 shows an example of a Jira ticket that, by definition, is a collection of texts organized in attributes that should contain the necessary information to execute, understand, and organize an activity in a project. The attributes are text fields that could contain predefined values to be selected or are free text input. All the attributes showing in Figure 2.2 are not required to exist in all projects and tools, and even if it exists, it is not required to be fulfilled. All attributes are configurable, depending on the requirements and the necessity of each project.



Hadoop Common / HADOOP-13204 Über-jira: S3a phase III: scale and tuning / HADOOP-13654

S3A create() to support asynchronous check of dest & parent paths

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Won't Fix
Affects Version/s:	2.7.3	Fix Version/s:	None
Component/s:	fs/s3		
Labels:	None		

People

Assignee:	Unassigned
Reporter:	Steve Loughran
Votes:	0 Vote for this issue
Watchers:	6 Start watching this issue

Dates

Created:	25/Sep/16 15:24
Updated:	04/Sep/17 14:34
Resolved:	04/Sep/17 14:34

Description

One source of delays in S3A is the need to check if a destination path exists in create; this makes sure the operation isn't trying to overwrite a directory.

#. This is slow, 1-4 HTTPS requests

1. The code doesn't seem to check the entire parent path to make sure there isn't a file as a parent (which raises the question: shouldn't we have a contract test for this?)
2. Even with the create overwrite=false check, the fact that the new object isn't created until the output stream is close()'d, means that the check has race conditions.

Instead of doing a synchronous check in create(), we could do an asynchronous check of the parent directory tree. If any error surfaced, this could be cached and then thrown on the next call to: write(), flush() or close(); that is, the failure of a create due to path problems would not surface immediately on the create() call, **but before any writes were committed**.

The full directory tree can/should be checked, and is results remembered. This would allow for the post-commit cleanup to issue delete() requests purely for those paths (if any) which referred to directories.

As well as the need to use the AWS thread pool, there's a bit of complexity with cancelling multipart uploads: the output stream needs to know that the request failed, and that the multipart should be aborted.

If the complexity of the asynchronous calls can be coped with, **and client code happy to accept errors in the any IO call to the output stream**, then the initial overhead at file creation could be skipped.

Activity

Steve Loughran added a comment - 04/Sep/17 14:34

Given what s3guard delivers in a sync call, this is needless complexity. Closing as a wontfix

Figure 2.2: Example of a Jira ticket.

In Figure 2.2, there are three attributes highlighted in black. They are all free text fields that on Information Retrieval (IR) are called unstructured attributes. These fields are standard in almost all tools and projects, and they represent the title (Summary) of the ticket, a description of the activity (Description), and the comments field could contain better explanations of the ticket as well as issues about the activity progress (Comments). The red highlighted part shows a group of predefined texts that on IR are called structured attributes.

Ticket management tools are essential nowadays to help Software organizations to manage project knowledge. Add automation solutions, as the proposed ticket recommendation system, helps improve the KM effectiveness. The next section explains the concepts related to IR techniques that were applied to the project.

2.2 Recommendation System & Information Retrieval

The creation of Recommender Systems (RS) has attracted the interests of both industry and academy since the 90's [2]. Analyzing a large volume of information, like a ticket repository on a big organization, is nearly impracticable without a RS [28]. A recommendation system is a service responsible for finding relevant items on a collection for a specific user [74]. For this to be possible, a RS employs techniques to find relevant items and rank them according to user's past behaviors [29].

Recommendation system techniques are classified into three filtering groups: collaborative [25, 72], content-based [10, 69], and hybrid [29]. Collaborative filtering is based on the assumption that users with the same interest in the past perhaps will have the same interest in the future [51]. So, for a specific user, an RS based on collaborative filtering identifies other users with the same interest and recommends the items that these users preferred [47]. Content-based filtering determines similarities between items contents [14]. The idea is that if the user had an interest on an item in the past, he or she would presumably have an interest in similar items in the future [47].

Baeza-Yates and Ribeiro-Neto [5] defined Information Retrieval as a technique that "deals with the representation, storage, organization, and access to information items such as documents, structured and semi-structured records, multimedia objects. The representation and organization of information items should provide users with easy access to information of their interests". The goal of an IR system is to retrieve all relevant documents to a user while retrieving a few non-relevant documents. The difficulty is how to decide the document relevance, essentially because relevance is a personal assessment that depends on the context [5]. Figure 2.3 illustrated the process of retrieving (collection, index, and query) and ranking.

The first process of retrieval and ranking method is to index documents. This procedure starts before any user interaction. The first step is to define the dataset, choose the documents that will be used to compose it, and the attributes that each document will contain. With the dataset prepared, the next step is to apply some text operations, such as removing stopwords, stemming, and selecting relevant terms. After the terms have been selected, the final step is to use them to index documents, as showed in Figure 2.3.

With the documents indexed, users may define queries that trigger the search for relevant documents. The retrieval process starts with a query provided by the user. The query is the user input used to find the relevant items or documents. The query can be a single word, phrase, or document and could be structured or unstructured. With the user's input query, the IR system should be capable of retrieving relevant documents. Before going to the Retrieval step, the query goes through a text transformation similar to the indexing process, removing stopwords, stemming, and selecting terms. On the Retrieval step, the system uses the selected terms of the query to obtain the indexed documents.

The Ranking Process ranks the retrieved documents according to the similarity between them and the query. The Ranking Process utilizes Text Classification Algorithms to classify the documents according to their similarities. The algorithms are divided into two groups unsupervised and supervised [5]. An unsupervised algorithm is suitable when there is no information on training examples available, especially to large collections [5]. On the other hand, the supervised algorithm uses some training data to improve the output [5].

A traditional Information Retrieval approach for an unsupervised algorithm is the Vector Space Model (VSM) [56]. Where a vector representation is constructed for each document based on the frequency of terms defined in terms of the traditional TF-IDF weighting scheme [5], as explained in the next paragraphs.

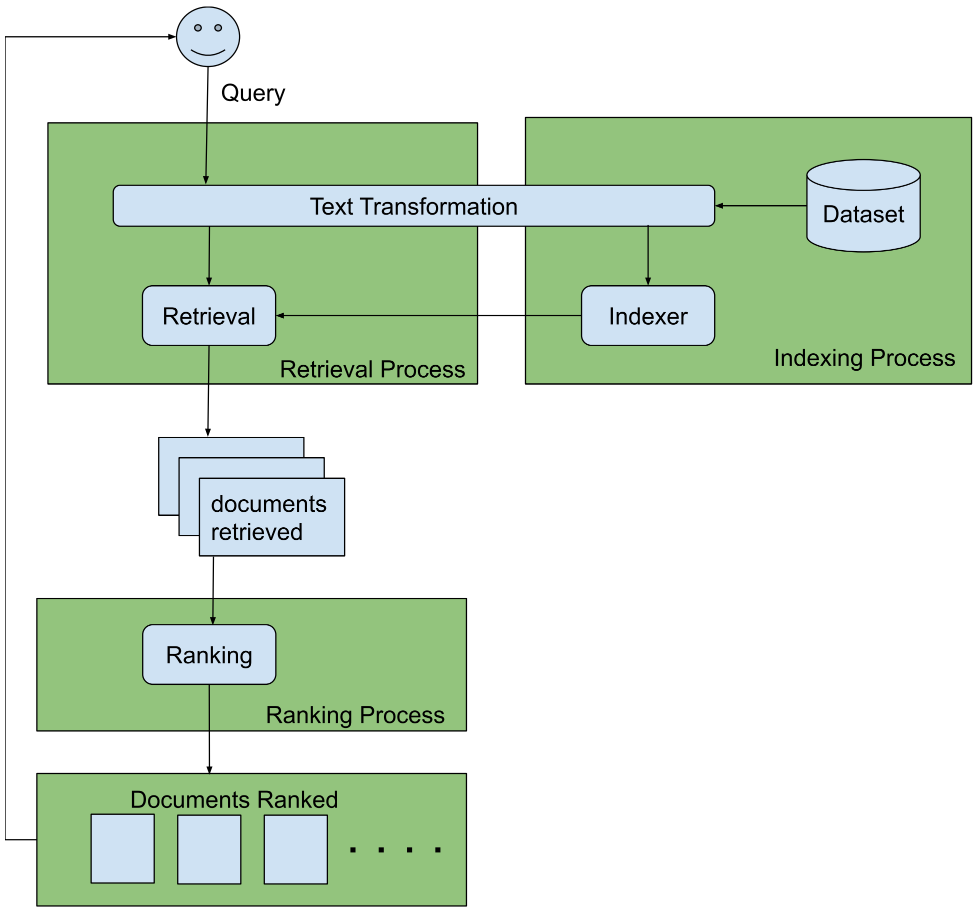


Figure 2.3: Information Retrieval and Ranking process.

The definition of feature vectors related to textual document works as follow:

1. The first step concerns the identification of all terms of the collection. A term is a word or a string of characters. However, some terms, known as stopwords, are not included in the vector. Stopwords are common words in a language that do not convey meaning to the text, such as prepositions and conjunctions. A typical list of English Stopwords are: “a”, “an”, “and”, “are”, “as”, “at”, “be”, “but”, “by”, “for”, “if”, “in”, “into”, “is”, “it”, “=”, “no”,“not”, “of”, “on”, “or”, “such”, “that”, “the”, “their”, “then”, “there”, “these”, “they”, “this”, “to”, “was”, “will”, and “with”. The vector size is the number of terms previously defined, i.e., each position of the array is associated with a distinct term.

2. The weight of each term in the document is computed based on the TF-IDF weighting scheme. TF stands for Term Frequency, while IDF, Inverse Document Frequency [5]. TF encodes the frequency of a term in a document. The more frequent a term is in the document, the higher its TF is [34]. IDF, in turn, encodes how rare a term is in the collection [30]. In this case, the less frequent a term is in the collection, the higher its IDF is [52].

   A popular definition uses the log normalization TF-IDF proposed by Salton and Yang [56]. Let n_i be the document frequency of the term k_i, and $w_{i,j}$ be the term weight associated with the pair $(k_i,\ d_j)$, where d_j is the document j. Then, the TF-IDF weight is defined as [5]:

$$TF = (1 + \log_2 f_{i,j}), \tag{2.1}$$

where $f_{i,j}$ is the frequency of term i in document j.

$$IDF = \log_2 \frac{N}{n_i}, \tag{2.2}$$

where N is the number of documents and n_i is the number of documents that contain term i.

$$w_{i,j} = \begin{cases} (1 + \log_2 f_{i,j}) \times \log_2 \frac{N}{n_i} & iff_{i,j} > 0 \\ 0 & otherwise \end{cases} \tag{2.3}$$

where $f_{i,j}$ is the frequency of the term i in document j, and N is the number of documents in the collection. Figure 2.4, Table 2.1, Table 2.2, and Table 2.3 show an example of how the TF-IDF is calculated.

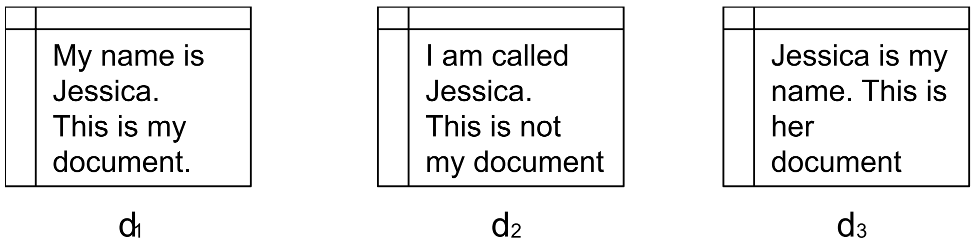

My name is Jessica. This is my document.
I am called Jessica. This is not my document
Jessica is my name. This is her document
d1
d2
d3

Figure 2.4: Documents example for TF-IDF calculation.

Table 2.1: TF Computation.

Term i	Term	$f_{i,1}$	$f_{i,2}$	$f_{i,3}$	$TF_{i,1}$	$TF_{i,2}$	$TF_{i,3}$
01	My	2	1	1	2	1	1
02	name	1	-	1	1	-	1
03	is	2	1	2	2	1	2
04	Jessica	1	1	1	1	1	1
05	This	1	1	1	1	1	1
06	document	1	1	1	1	1	1
07	I	-	1	-	-	1	-
08	am	-	1	-	-	1	-
09	called	-	1	-	-	1	-
10	not	-	1	-	-	1	-
11	her	-	-	1	-	-	1

Table 2.2: IDF Computation.

i	Term	n_i	IDF_i
Term i 01	My	3	0
02	name	2	0.58
03	is	3	0
04	Jessica	3	0
05	This	3	0
06	document	3	0
07	I	1	1.58
08	am	1	1.58
09	called	1	1.58
10	not	1	1.58
11	her	1	1.58

Table 2.3: TF-IDF Computation.

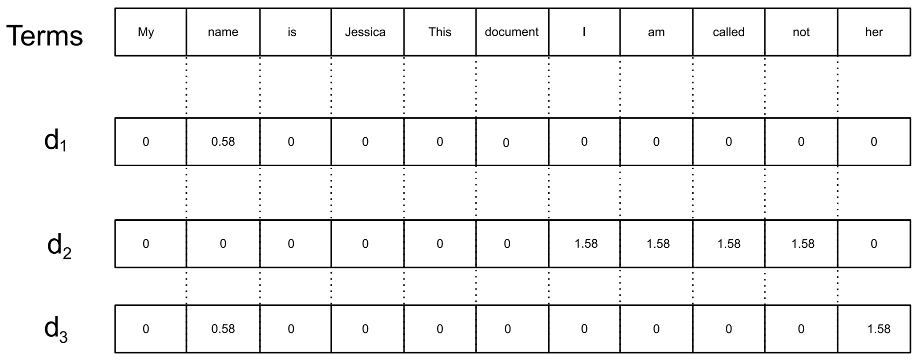
Term i	Term	$TF-IDF_{i,1}$	$TF-IDF_{i,2}$	$TF-IDF_{i,3}$
01	My	0	0	0
02	name	0.58	0	0.58
03	is	0	0	0
04	Jessica	0	0	0
05	This	0	0	0
06	document	0	0	0
07	I	0	1.58	0
08	am	0	1.58	0
09	called	0	1.58	0
10	not	0	1.58	0
11	her	0	0	1.58

The column Term that is present on all tables shows the terms extracted from all documents. The columns $f_{i,j}$ in Table 2.1 show the frequency of the term i on document j. The columns $TF_{i,j}$ on the same table is the result of the TF computation for every term per document. Column n_i in Table 2.2 shows the number of documents that the term is present, and column IDF_i is the IDF calculation of each term. Table 2.3 shows the TF-IDF computation for every term on each document.

Given two documents, their similarity is computed based on the cosine similarity. Recall that weight $w_{i,j}$ of the term i of document j is defined by the TF-IDF scheme. This weight is a non-negative and non-binary real number. The terms are assumed to be all mutually independent and represent units of a t-dimension vector, in which t is the total number of terms. The documents d_j and query q are represented as t-dimensional vectors given by $\overrightarrow{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$ and $\overrightarrow{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$. To evaluate the degree of similarity of document d_j and the query q, the cosine of the angle connecting these two vectors is calculated as [5]:

$$sim(d_j, q) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{|\overrightarrow{d_j}| \times |\overrightarrow{q}|} \tag{2.4}$$

where $\bullet$ is the dot product of two vectors and $\times$ is the algebraic product of their scalar norms.



Terms	My	name	is	Jessica	This	document	I	am	called	not	her
d_1	0	0.58	0	0	0	0	0	0	0	0	0
d_2	0	0	0	0	0	0	1.58	1.58	1.58	1.58	0
d_3	0	0.58	0	0	0	0	0	0	0	0	1.58

Figure 2.5: Document vector representation with TF-IDF weight example.

Table 2.4: Computation of the Cosine Similarity with TF-IDF weight.

Similarity	$\overrightarrow{d_j} \bullet \overrightarrow{q}$	$\lvert\overrightarrow{d_j}\rvert$	$\lvert\overrightarrow{q}\rvert$	$\lvert\overrightarrow{d_j}\rvert \times \lvert\overrightarrow{q}\rvert$	Result	Rank
sim(d_1, d_3)	0.34	0.34	2.84	0.97	0.35	1
sim(d_2, d_3)	0	9.99	2.84	28.37	0	2

Figure 2.5 and Table 2.4 show a practical example of cosine similarity with TF-IDF weights. Figure 2.5 gives the vector representation of each Figure 2.4 document. Table 2.4 illustrates the computation of the cosine similarity using document d_3 as the query $\overrightarrow{q}$.

2.3 Feature Selection

Large term-dimension vectors might slow down the performance of document classifiers, making the classification impractical. A classic solution is to reduce the number of terms, known as features, by selecting a subset of all terms to represent the document [1,8,19,71]. This solution is known as *feature selection.* Feature selection techniques decrease the issue mentioned previously as well as the curse of dimensionality.

Bellman [7] introduced the concept curse of dimensionality in 1961 and still a contemporary issue. For example, on VSM, each feature is a dimension on the document vector representation. Therefore, intuitively we conclude that as many features we can use more precisely will be the classification. However, Bellman [7] discovered that there is an optimal amount of features to be used. Figure 2.6 shows that more dimensions (features) data points have, the less effective classifiers that exploit such features are.

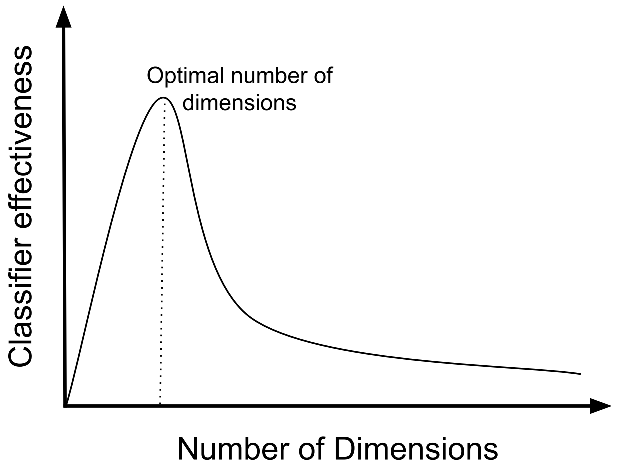


Figure 2.6: Curse of Dimensionality.

Given the curse issue described above, choosing an appropriate feature selection method to select a proper feature subset is mandatory. The selected feature selection approach should be capable of handling some of the following challenges [64]:

1. **The high number of combinations:** Given a problem with N features, testing all possible feature combinations to find the best subset is virtually impossible. The number of possible feature sets is equal to $2^N - 1$ (exponential). It is impracticable for a computer to measure all combinations in a satisfying time.

2. **Remove irrelevant features:** Several features are not relevant, as stopwords, for example. However, stopwords are language-specific and not context-specific. Therefore, some features are relevant only to some contexts. Identify and remove these no-relevant context features is a challenge.

3. **Remove redundancy:** The stemming process removes redundant words by reducing inflected words to the root word. However, finding all redundant words on a context-specific project is hard. Identifying synonyms, typos, and word stems is a complicated process, and a challenge for all feature selection approaches.

4. **Keep integration features:** Even if a given feature is considered irrelevant, it may be relevant when combined with others. Feature selection approaches should not evaluate features individually and should keep associated features together.

Feature selection techniques handle the above challenges in different ways. The main known groups of methods to handler theses difficulties are filter, wrapper, and hybrid methods. These groups are defined according to how the methods evaluate the feature subset [64]:

1. **Filter:** The filter methods evaluate individual features regardless of the model, by essentially filtering the least relevant features. The relevance calculation is usually based on the intrinsic properties of the feature, such as consistency, information measurement, and correlation. These methods are usually of low computational

cost, making scalable even in high dimensional datasets. However, these approaches tend to select redundant features due not to consider the relationships between them [64].

The most popular filter methods are the Correlation-based Feature Selection (CFS) [24], the Fast Correlation Based-Filter (FCBF) [73], RelieffF [53], and the minimal Relevance Maximal Redundance (mRMR) [48]. CFS [24] uses Pearson's correlation to evaluates the features. FCBF [73] measures the features based on the symmetrical uncertainty calculation, using the Markov blanket [73] concept. ReleifF [53] evaluates features quality by verifying how well their instances discern from distinct classes. The mRMR [48] selects the features most correlated with the classes and most different to the others, based on the criteria of maximum dependence, maximum relevance, and minimum redundancy defined by their authors.

2. **Wrapper:** The wrapper methods evaluate a feature subset based on the performance of a predetermined mining algorithm, which allows the possibility to detect relationships between features [64]. This mining algorithm could be supervised as k-Nearest Neighbors (k-NN) [31] and unsupervised as k-Means [27]. However, challenges as the risk of overfitting [32] and increase of computational time when the number of features is large are constant present on this approach [64]. Some studies applied a combination of Genetic Algorithm and these techniques to mitigate these issues [31, 64].

3. **Hybrid:** The hybrid methods evaluate the features by applying the filter and wrapper methods together, exploring their symbiosis in the search for a better selection of features [64]. Typically, filtering evaluation is used to improve the efficiency of a wrapper method. The selection methods of hybrid characteristics are quite popular today, due to the fact that they allow an increase in the efficiency of the wrapper methods, preserving their effectiveness [64]. Most successful hybrid methods employ a global search such as GA search, refined through local search operations [64,75,76]. The concepts of GA and global and local search are presented in the next section.

2.4 Genetic Algorithm

There are different proposed approaches in the literature for automatically selecting features that handle all the challenges. A well-known non-deterministic algorithm that has been used in different text-classification studies for feature selection is the Genetic Algorithm (GA) [8, 19, 71]. John Holland [26] introduced GA in 1975, where he noticed that natural evolution is like a learning method. The natural evolution concepts defined nature as a process that selects the best individual, according to a given fitness function. Given a particular group of people, the individuals more prepared to survive stay alive, and through inheritance pass the best features to the next generation. GA is an evolutionary algorithm, which mimics evolution principles found in nature to solve optimization and search problems [64].

GA is an iterative process that maintains a population of candidates that resolve the problem. Through each iteration, called generation, the current population is evaluated by a performance measure that indicates how close an individual is to solving the problem. Based on the previous evaluations, a new population is formed using three genetic operators: selection, crossing, and mutation. This process is explained in Figure 2.7.

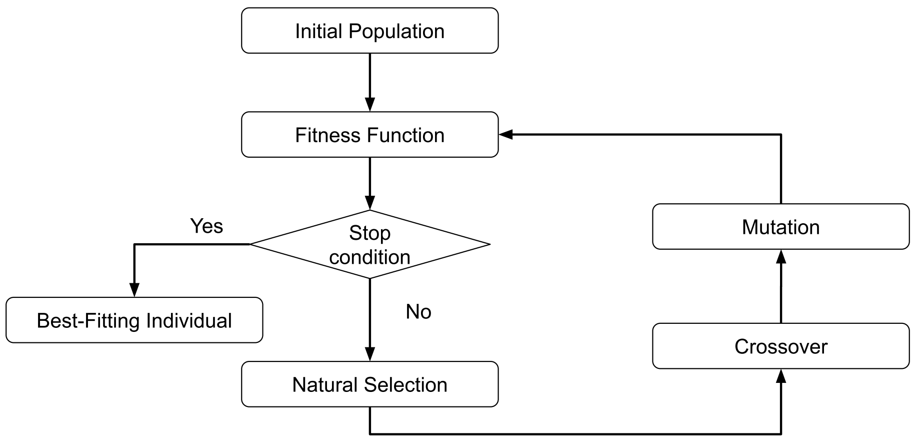


Figure 2.7: The Genetic Algorithm pipeline.

Before starting the GA process, the first step is to define the individual. A typical representation of an individual is a chromosome. A chromosome is a sequence of genes, where each gene represents a unique feature, and the value (0 or 1) of the gene denotes the presence of that feature. Figure 2.8 shows an example of a chromosome. With the chromosome defined, the creation of the Initial Population is straightforward. The idea of this step is to establish an initial number of chromosomes and how they will be created.

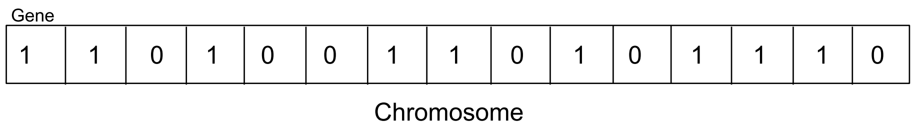


Figure 2.8: Chromosome example.

The number of chromosomes is usually defined through empirical tests. Initially, it is set a random amount of individuals. Then, through observation of the fitness results of the population on each generation, it is verified the necessity to increase, keep, or decrease the number of chromosomes. The creation of each chromosome can be randomly or through a heuristic.

The Fitness Function is used for the performance evaluation of each individual. This step measures how good the chromosomes are by giving a score to each of them. The score and how to calculate are context-specific for the problem that the GA is being applied. The next phase is the starting point of the GA iteration. Here, a stop condition should be provided. Usually, the stop criterion adopted is a fixed number of generations. That is the total number of iterations the loop will run. The number of generations, in a similar way of the number of chromosomes, is often defined through empirical tests.

Another traditional strategy for the stop condition is to verify the best individual for each generation. If the best individual does not change for several generations, it is an indication that will not find a better individual. The Natural Selection step removes the most unfit chromosomes. The most common methods for the natural selection step are [64]:

1. **Stochastic with replacement:** Develop by Holland [26]; it is the standard method. The first step calculates the probability of a chromosome to be selected using the fitness evaluation to normalize. This way, the sum of the chromosome probabilities is equal to 100%. So, the probability P_i of a chromosome C_i with fitness evaluation F_i is:

$$P_i = \frac{F_i}{\sum_{i=1}^{C} F_i} \tag{2.5}$$

   With all probabilities calculated, the next step is to use them to create a roulette with all chromosomes. Therefore, each time the roulette runs, a chromosome is selected until a desired number of chromosomes are chosen.

2. **Simple tournament:** The idea is to divide the chromosomes into N groups randomly. The method then chooses a group randomly and select the chromosome with the higher fitness evaluation of the chosen group. The previous step is repeated until the desired number of chromosomes is elected.

3. **Stochastic tournament:** Use the same principle as the Simple Tournament. However, instead of choosing the chromosome with the highest fitness evaluation, the roulette principle of the Stochastic with Replacement is employed.

4. **Truncation:** This refers to the selection of a subset of the highest fitness evaluation chromosomes.

5. **Order:** Sort the chromosomes by the fitness evaluation and calculate a probability of each chromosome to be chosen. The most common calculation to define the probability P_i of a chromosome C_i on the position $P(C_i)$ are:

$$Linear : P_i = a \times P(C_i)\,;\; where\; a > 0 \tag{2.6}$$

$$Exponential : P_i = a^{b \times P(C_i)}\,;\; where\; a > 0\, and\, b > 0 \tag{2.7}$$

6. **Elitism:** As the name suggests, this method chooses the top-N chromosomes according to the fitness evaluation.

In the Crossover step, two chromosomes exchange genes for creating a child chromosome. The expectation is that for each generation, the most adapted chromosomes generate best descendants. The well-known Crossover methods are [64]:

1. **Simple Crossover:** Figure 2.9 shows the Simple Crossover process. The first step is to decide a Crossover Point, where the parent chromosomes will be cut. Child A will be composed of the genes from the first part of Parent A, and the second part of Parent B. Child B will be composed of the genes from the first part of Parent B and the second part of Parent A.

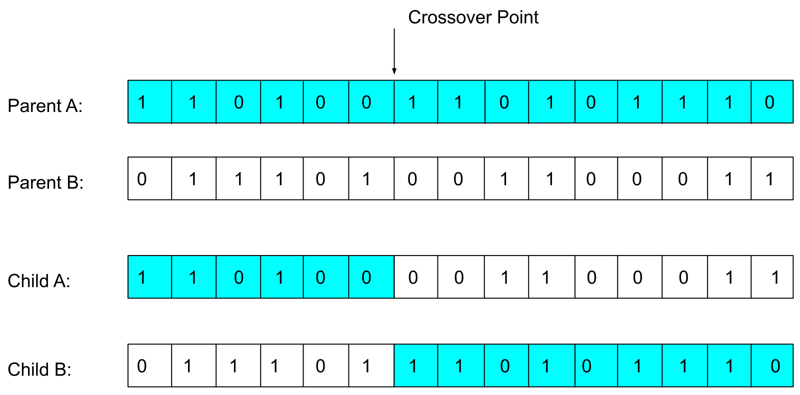

Crossover Point
Parent A: 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0
Parent B: 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1
Child A: 1 1 0 1 0 0 0 0 1 1 0 0 0 1 1
Child B: 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0

Figure 2.9: Example of a simple crossover.

2. **Multiple Crossover:** The process is similar to the Simple Crossover. However, it should have two or more Crossover Points, as showing in Figure 2.10.

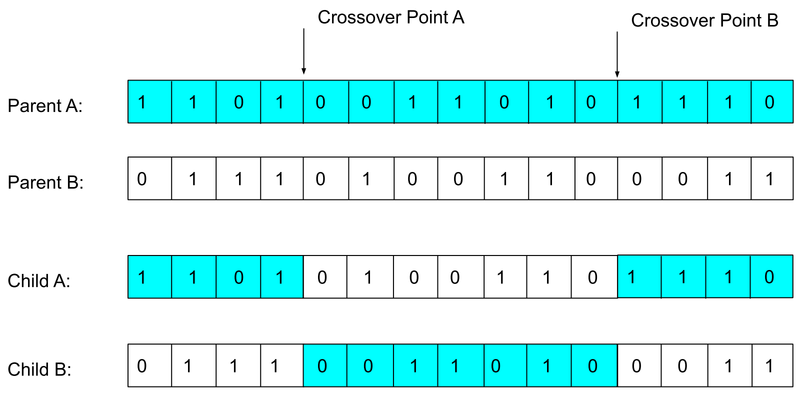

Crossover Point A
Crossover Point B
Parent A: 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0
Parent B: 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1
Child A: 1 1 0 1 0 1 0 0 1 1 0 1 1 1 0
Child B: 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1

Figure 2.10: Example of a multiple crossover.

3. **Uniform Crossover:** The process is similar to the Simple and Multiple Crossover. However, the Crossover Points should be decided in a way that the Children have 50% genes from the Parent A and 50% genes from Parent B.

The Mutation is an operation that randomly modifies the gene(s) of a chromosome. If the gene value is 0, it becomes 1, and if it is 1, it becomes 0. This process is essential because it introduces genetic diversity, fostering the possibility of individuals to be found in different locations of the search space. To modify the genetic code of the chromosome, the GA first decides the probability for that to happen. Furthermore, it decides if the probability of the mutation will be applied by chromosome or gene. The most common ways to apply the mutation are replacing one gene by another randomly or applying a permutation of the genes present in the chromosome.

The evolution process through generation produces the best individuals. In the end, the GA framework output is the best individual found when the Stop Condition is reached. An accurate GA approach should be capable of getting the best individual from the maxima or minima global and not local. Maxima and minima of a function are known to be the extremes points. Local maxima and minima are points that are extremes by the neighbor points. However, they are not extremes by all points. Global maxima or minima is the goal for all GA approaches.

Figure 2.11 shows a graphic representation of all possible fitness evaluation value of an individual. In this figure, there are four local maxima one global maximum. It is normal to a GA find out that the best individual is one of the local maxima because, during the generations, the children are close to the parents. It is the developer's responsibility to add some heuristics, boost mutation probability, and increase the number of individuals to overcome this issue.

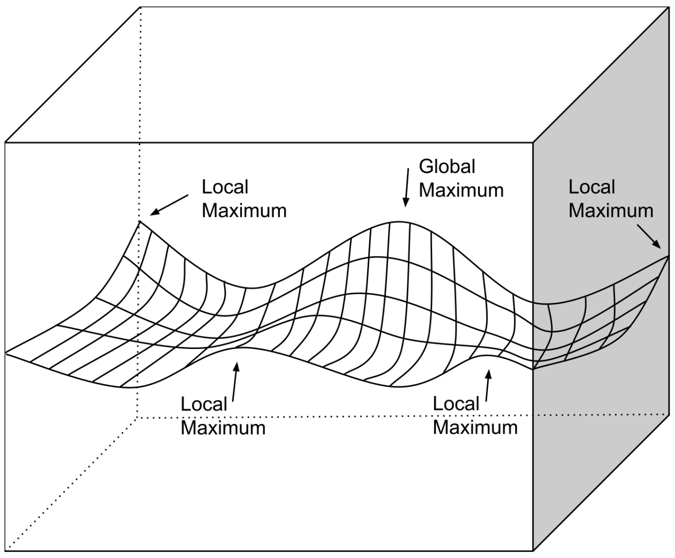


Figure 2.11: Examples of local and global maxima in a typical GA search space.

2.5 Related Work

Some recent studies show that automatically sharing knowledge is essential in software projects. Liu et al. [33] developed a recommendation system called EXPSOL to recommend Github and Stack Overflow threads to exception-related bug reports to help developers find solutions for their problems fast. An exception is an unusual event that occurs during the execution of a software system, and the presence of exception often indicates a bug to be fixed [35]. Most high-level programming languages, like Java, Python, and C++, have mechanisms that handle the exceptions [13].

During the software development process, exception-related bugs are reported through ticket systems, like Jira, Trello, and others. When developers work on a ticket that addresses an exception-related bug, and the developer is unfamiliar with the exception bug reported, they often search online forums (e.g., StackOverflow) to understand it. Although there are some search engines and research tools available, they are unable to properly recommend forum threads for exception-related bugs, especially for large-scale online resources [33]. To address that, Liu et al. [33] proposed the EXPSOL, to recommend the forum threads and possible solutions for newly exception-related bug tickets.

The recommended system build an Exception Tree using the exception-related online resources and software language taxonomy. With this structure, they extract features from exception-related bug tickets from GitHub and forum threads from StackOverflow to build Space Vector Models representations. This way, when a new ticket is created, the EXPSOL is capable of finding the similarity between the ticket and the forum threads to recommend possible threads that contain the solution. They analyzed the efficiency of the system and compared it with others. To evaluate the proposed recommendation system, they used the Mean Average Precision, the Mean Reciprocal Rank, and the Recall of the first 10, 20, and 30 items recommended. They compared their solution with the Google and StackOverflow search engine, where EXPSOL showed significant improvement on the recommendation of exception-related bug threads.

Santos [57] investigated an approach that links project tickets with StackOverflow threads that have previously been curated and used task similarities to investigate the possibility to recommend curated StackOverflow threads for similar tickets. For any software development, the developers' knowledge and expertise are essential for a successful project [67]. To acquire knowledge, the developers usually use search engine websites [39]. In this process, they need to use precise terms on the searches and verify if the results are reliable and relevant [39].

A popular website where developers find knowledge to support their development is StackOverflow [33]. However, there is not an explicit link between the tickets and the StackOverflow threads that were useful to solve them. This lack of integration between ticket repositories and StackOverflow threads is an open issue identified by researchers [68]. To improve this issue, Santos [57] proposed a recommendation system using RapidMiner to recommend StackOverflow threads for tickets that are similar to previous tickets linked to StackOverflow threads. The proposed system obtained higher precision when compared to other studies.

Sun et al. [65] conducted an empirical study with multiple recommendation techniques in the literature to understand if historical commits are always useful to recommend tickets to developers. Tickets, such as bugs fix and new feature requests, are regularly proposed [66]. These tickets need to be allocated to the developer that is most capable of working on them [66]. All information available in the source-code repositories, ticket repositories, and historical commits are used to facilitate this task [70].

The research hypothesis to recommend tickets to developers is based on the developer's experience on similar tickets. So the developer that has more experience than the others should be the one recommended for the ticket [61]. Based on this hypothesis, historical commits could be useful for the recommender. A common way to do that is to use text classification techniques to find similarity between the source-code files in the historical commits and the tickets [62], and other methods using historical commits are also proposed.

The studies of Sun et al. [65] aimed to understand which of these methods is better to recommend tickets. For that, an empirical study was applied, and critical research questions were proposed to guide the investigation. They discovered that the commit description with more meaningful words increases the precision of the recommendation, and the description is more useful than the files related to the commit [65].

At the same time, studies aiming at improving the techniques used in most of the recommendation systems, as text classification, feature selection, machine learning, and Genetic Algorithm, have been conducted recently. Mirończuk et al. [41] delivered a recent overview of the state-of-the-art elements of text classification. The study mentions that text classification is a sophisticated process that involves not only the training models but also other procedures, like data processing and dimensionality reduction. Figure 2.12 displays the most utilized baseline process for text classification presented in the research, showing six essential elements and their respective results.

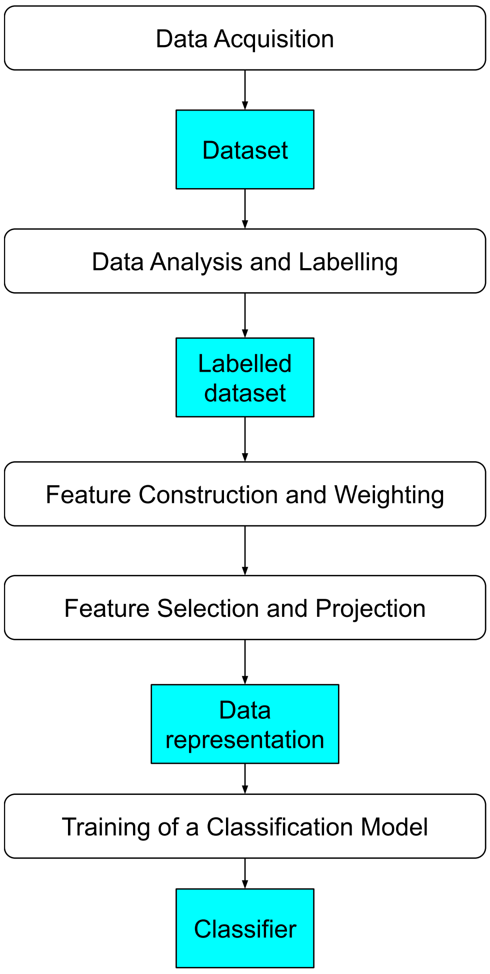

Data Acquisition
Dataset
Data Analysis and Labelling
Labelled dataset
Feature Construction and Weighting
Feature Selection and Projection
Data representation
Training of a Classification Model
Classifier

Figure 2.12: Text classification framework.

The first step is data acquisition, where the required data is acquired from multiple text sources to solve a research objective related to a classification task. The purpose of this phase is to build the research dataset. Next, the dataset goes through the data analysis and labeling phase, where two main strategies could be applied to generate the Labelled dataset: labeling groups of text or indexing. On the next step, the Labelled dataset is processed to generate the data representation required by the selected learning method. In this phase, there are two well-known data representations: Vector Space

Model and graph. Both representations are based on features and their weights.

With the data representation ready, the feature selection step is started aiming to reduce the dimensionality. This process is mainly divided into two phases: the selection of the most relevant features and feature transformation. Following the feature selection, the training model phase begins to generate the classifier. The training model algorithm could be divided into the following groups: supervised, semi-supervised, ensemble, active, transfer, and multi-view learning. All the steps described before form the text classifier. Finally, to complete the baseline process, an evaluation method should be employed.

The evaluation stage is divided into three steps. First, an established indicator, as Precision, Recall, Accuracy, F-score, error rate should be selected. Second, an evaluation protocol, like the k-fold cross-validation, should be defined. The last phase is to compare the results with other approaches or different datasets. All of the above steps form the framework for text classification.

Deng et al. [19] provided an overview of state-of-the-art of methods that have been used for feature selection on text classification. Genetic Algorithm with term-weighting schemes was one of the most popular methods. Bidi et al. [8] provide an empirical study of a feature selection method based on genetic algorithms on different text classification methods. The performance evaluation of the proposed feature selection was done on three text classifiers: Naive Bayes, Nearest Neighbors, and Support Vector Machines.

As an extension of these studies, this dissertation presents a change request recommendation system using a different Genetic Algorithm implementation with TF-IDF and cosine-similarity. In particular, we introduce a new heuristic to guide the selection of the GA initial population, which is demonstrated to yield effective results. More details are provided next.

Chapter 3

GA-based Ticket Recommendation System

This chapter introduces the proposed approaches to recommend tickets, based on the document representations defined through GA-based feature selection procedures. Section 3.1 presents the architecture and design of the feature selection based on GA. Section 3.2 explains the heuristic approach based on Hamming distance to improve the feature selection based on GA.

3.1 Feature Selection Based on Genetic Algorithm (FSGA)

As mentioned in Section 2.3, feature selection is the process of selecting a subset of features. In our problem, those features are expected to improve the effectiveness and efficiency performance of the ticket recommendation system.

Figure 3.1 illustrates the feature selection overall process for text classification using GA. The first step is to define and collect all the features. In the step Extract Terms, every single term (word) from each document is considered a feature. These terms are collected, filtered by removing the stop words mentioned in Section 2.2, grouped on a set, and then saved to be used on the following steps. With all features selected, the process to find the best subset using the Genetic Algorithm method can start. Firstly, the chromosome representation is built according to the Feature Set. The chromosome array has the size of the number of features present on the set, and each position on the array corresponds to a unique feature. An example of the chromosome is present in Figure 3.1 on the Initial Population step.

Furthermore, the Initial Population is built. Forty distinct chromosomes are built with a random distribution of the values 1 and 0 through the array. The value 1 represents that the feature corresponded on that array position is present on the subset, and 0 the feature is not present. The population size was defined by empirical tests, via analyses of the performance of the population over the generations. Another determinant limitation of the population size was the computational costs involved, as the computation of the Fitness Function of each chromosome consumes significant processing time.

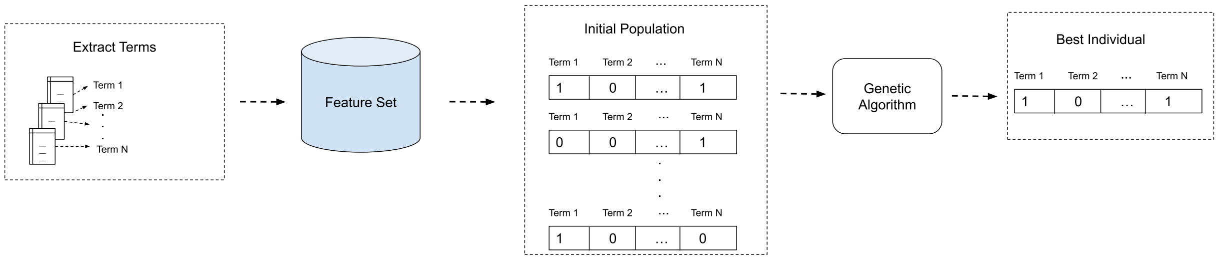


Figure 3.1: Feature selection using Genetic Algorithm.

With the First Population built, the following steps are to perform the GA process of Figure 2.7. Firstly, the Fitness Function used to assess how best an individual relies on the quality of rankings produced. The evaluation used is the average Precision at 10 (see Section 4.5) of all queries of the conventional ticket recommendation system presented in Section 4.4. Where the document vector representation, similar to Figure 2.5 example, will contain only the terms that have value 1 on the individual been evaluated. With all individuals evaluated, the next step is the Natural Selection. The method applied is the stochastic with replacement and always keeping the individual with the highest evaluation. The stop condition used is the number of generations. The Best Individual found contains the most suitable subset of terms that is expected to improve the effectiveness of the ticket recommendation system.

3.2 Heuristic Approach for Feature Selection Based on Genetic Algorithm (H-FSGA)

The number of possible different individuals on text classification is significantly high. More precisely, there are 2^{N_T} possible different individuals, where N_T is the number of terms. For example, in this dissertation using the Hadoop collection, there are around 50,000 different terms, so $2^{50{,}000}$ possible individuals. In such a high-dimensional search space, the chance of GA finding only a local minimum is very high, especially for problems when the size of the population is small due to the long processing time of the Fitness Function.

To illustrate this issue, Figure 3.2 shows an example of a small three-dimension search space, where there are eight possible individuals. Let's suppose the GA for this example uses two individuals for the first population. Unless the pair of the first individuals has the highest Hamming distance, the individuals generated through crossover will not explore all dimensions without a mutation. Even with a small dimensionality example, the probability of that happens is high. Building the first population comprised of 25% (2 out of 8 individuals) of the size of all possible individuals, there is an 85.7% (24 out of 28 pairs of individuals) chance of GA still relies on the mutation operator to be able to explore all dimensions of the cube. The risk of getting confined within a sub-space grows exponentially with the number of dimensions.

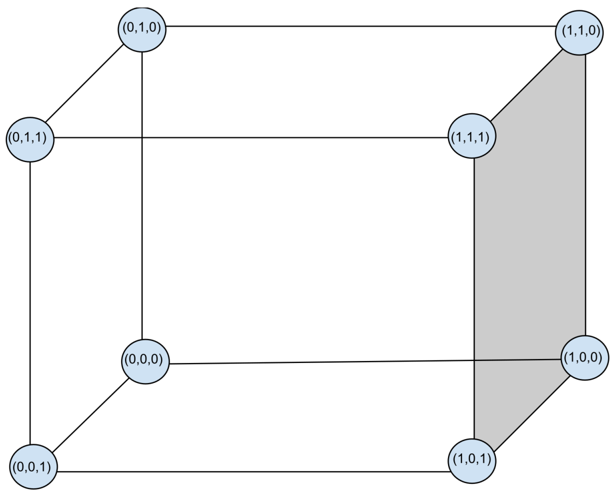


Figure 3.2: Example of a three-dimension search space.

This dissertation addresses this issue by proposing a new heuristic to generate the initial population. The goal is to spread the first individuals into the search space in a smarter way so that the search for the best individual does not become trapped in local minimum regions. The strategy relies on the First Population step creating individuals as far as possible from each other in the search space. Using Figure 3.2 search space scenario as an example and the first population with two individuals, the strategy is that pairs of the first individuals should be from a diagonal of the cube. For the heuristic to do that, it uses the Hamming distance to guide the definition of suitable chromosomes in the initial population. Thus, two individuals are as far apart as possible from each one when the genes of one individual are entirely different from the genes of the other individual. Using the individuals from Figure 3.2 as an example, the possible pairs with the highest Hamming distance are: (0,0,0 and 1,1,1), (0,0,1 and 1,1,0), (0,1,0 and 1,0,1), and (0,1,1 and 1,0,0).

Algorithm 1 outlines the main steps of the proposed heuristic to compute the initial population. The strategy is to produce chromosomes with a sequence of genes 1 (segment) on different positions, as illustrated in Figure 3.3. The segment size is the ratio of the number of chromosomes on the first population with the number of genes. So, three chromosomes with fifteen genes generate a segment that contains five genes 1, as shown in Figure 3.3. Thereby, the Hamming Distance of the chromosomes will be as far as possible from each. However, this strategy will always generate the same individuals, and this is not desired for GA approaches. So, to incorporate a certain degree of randomness, the genes inside the segment will be defined randomly. Consequently, the size of the segment is not more fixed, the number of genes 1 is higher than the previous ratio, and overlaps between the segment of other chromosomes are permitted, as shown in Figure 3.4.

To address this strategy, Algorithm 1 needs the following inputs: the number of chromosomes in the first population N_C, the number of genes for each chromosome N_G, and a constant k that is used to control the size of the segments in a way that does not let the segment overlap being so significant. The constant k value was defined through experimentation and found that the value 2 led to better results. Furthermore, Lines 1 to 3 defined the population variable P that is initially empty and is the output of Algorithm 1, an auxiliary variable S to guide the starting position of the segments on each chromosome, and a constant N_{max} that is the desired number of genes (1) on each segment.

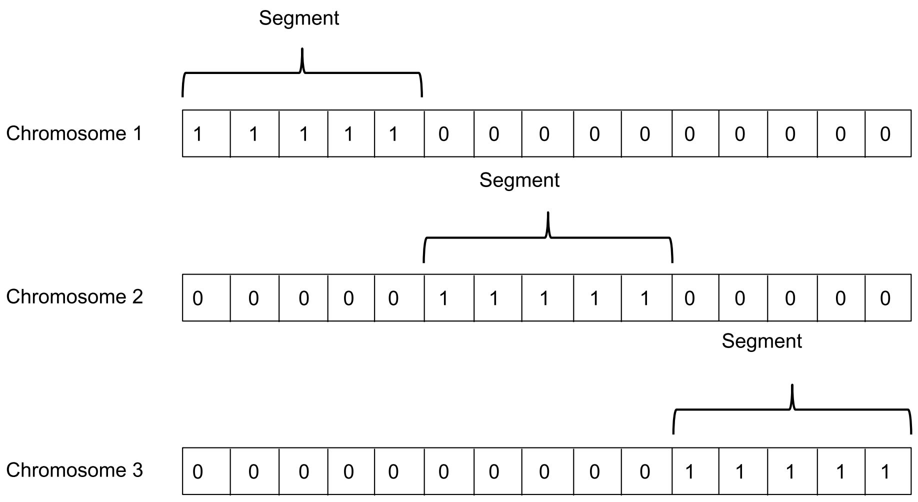


Figure 3.3: Example of chromosomes as far as possible from each by Hamming Distance.

Algorithm 1 Heuristic Approach for Selecting the GA Initial Population.

Input: Number of chromosomes n_C, number of genes n_G, and k that is a constant that defines the number of genes 1s in the chromosome

Output: Population of chromosomes P

```
▷ Let $N_{max}$ be the maximum number of genes 1s in the chromosome.
▷ Let $N_C$ be the number of genes 1s in chromosome $C$.
▷ Let $S$ be the index that start the segment of genes 1s.
 1: $P \leftarrow \emptyset$
 2: $S \leftarrow 0$
 3: $N_{max} \leftarrow k \times (\frac{n_G}{n_C})$;
 4: for each chromosome $C \in P$ do
 5:     $N_C \leftarrow 0$
 6:     for each gene $G \in C$ do
 7:         $G \leftarrow$ getRandomGene();
 8:         if $((G_{position} >= S)$ AND $(N_C < N_{max})$ AND $(G = 1))$ then
 9:             $C[G] \leftarrow 1$;
10:             $N_C{++}$;
11:         else
12:             $C[G] \leftarrow 0$;
13:         end if
14:     end for
15:     $S = S + (N_G/N_C)$;
16:     $P = P \cup \{C\}$;
17: end for
18: return $P$
```

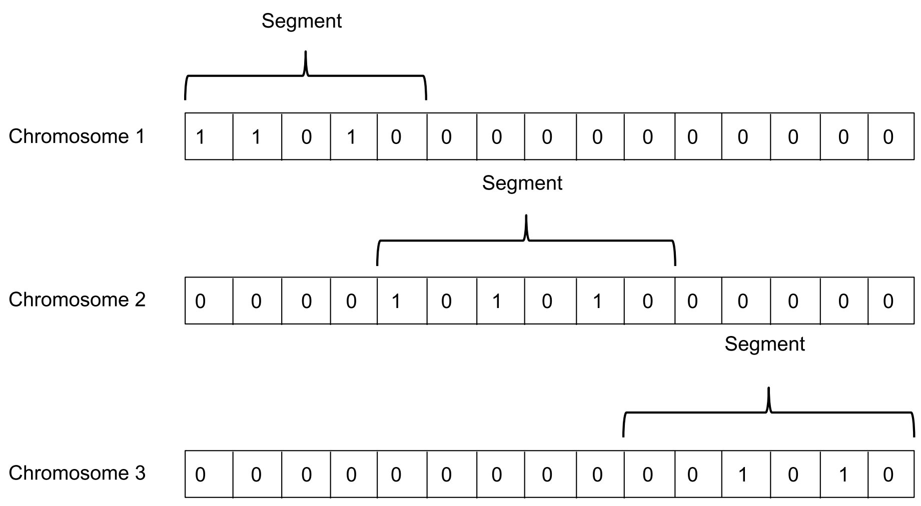


Figure 3.4: Illustration of the creation of a population using the proposed heuristic.

N_{max} is calculated using the input constant k, where the value is the ratio of the number of chromosomes by the number of genes multiplied by k. Therefore, the number of genes 1 on the segment will always be k times the ratio. N_{max} controls the strategy explained previously. Lines 7 to 13 are the core of Algorithm 1 , where each chromosome C is created and added to the population P. Firstly, it is verified where the segment will start using the variable S. The index S increases according to the ratio of the number of chromosomes by the number of genes.

Knowing where the segment starts the Algorithm 1 can start to defined the genes G. From the beginning of the segment, each gene has a 50% chance to be 1 or 0. This process will continue until the number of genes 1 created hits the N_{max} value, so the segment is closed. Furthermore, all genes before the beginning and after the ending of the segment will have the value 0. This method is repeated for each chromosome. In the end, the output will be a group of chromosomes similar to Figure 3.4.

The next chapter explains the designed protocol for the experiments that address each of the research questions and the proposed systems. Describe the dataset, the conventional ticket recommendation system, the applied evaluation metrics, and the statistical tests.

Chapter 4

Experimental Protocol

This chapter describes how the experiment was designed, conducted, and analyzed. Each section provides details about how the used dataset was chosen and organized, the experimental goals, and how the results were evaluated and validated.

Section 4.1 introduces the experiment's goals by presenting the research questions and showing how they will be addressed. Section 4.2 gives details of the chosen Apache Jira datasets and how it was organized. Section 4.3 explain what a relevant ticket is and how it was defined. Section 4.4 presents the conventional ticket recommendation system considered in our experiments. Section 4.5 discusses the evaluation metrics and statistical tests adopted.

4.1 Research Questions

A set of research questions were defined to guide the experiments. Achieved results, in turn, provide us insights regarding the main research goal of our dissertation, related to the investigation if the use of appropriate feature selection methods leads to more effective ticket recommendation systems. In the following, we present each research question considered in our study, providing its respective context.

RQ1: Which attributes are most effective to recommend relevant tickets?

The datasets used in our study are composed of *tickets*. Each ticket contains attributes that are text fields that could be structured or not. The attributes include all sorts of information about the ticket, as a short description, comments, or labels to group the tickets according to some relevance criterion.

Companies use different types of systems to create, track and organize tickets. Most of them share the same ticket structure but with different attributes. A group of the most common attributes should be selected; otherwise, the project will be relevant only for a specific ticket system.

These selected attributes need to be tested to identify which of them contribute to recommend relevant tickets. To address this question an experiment was performed to determinate which attributes are most suitable to improve the effectiveness of the ticket recommendation system.

RQ2: Is Genetic Algorithm a suitable approach for selecting appropriate terms and then improving the effectiveness of ticket recommendation systems?

A common approach for recommending tickets rely on performing searches based on vector representations. The objective is to rank relevant tickets, i.e., those who are the most similar to an input pattern (e.g., input ticket). The vector representation is often composed of weights associated with words defined in terms of their frequency in a specific ticket and the whole collection. Choosing a better (discriminative) set of terms to generate suitable vector representations is expected to improve the effectiveness of ticket recommendation systems.

In this work, we address the term selection problem (also known in the literature as feature selection) by investigating the use of a Genetic Algorithm apparatus for selecting suitable terms for the problem of ticket recommendation.

RQ3: Does the GA approach based on the Hamming distance heuristic algorithm for the first generation for selecting appropriate terms improve the effectiveness and the efficiency of the ticket recommendation system?

Genetic Algorithm is composed of essential steps to deliver the best result, like mutation, natural selection, and others that are presented in Chapter 2; one of these steps is the creation of the first generation. In this phase, the GA creates the first individuals to be used through the following steps of each generation.

This question aims to understand and produce insights to ascertain if a different approach to generate the first generation enhance the feature selection results by improving the effectiveness and the efficiency of ticket recommendation syste.

RQ4: How stable is the GA approach for selecting appropriate terms?

As pointed out in Chapter 3, our feature selection problem is exponential $O(2^n)$ in terms of time complexity, where n is the number of features. As mentioned in Chapter 2, the Genetic Algorithm is a possible solution for this problem. In this project, the GA approach seems a compatible solution in terms of processing time to generate consistent results.

How this solution is non-deterministic; in other words, for the same data input, the GA could generate different results. This question was required to conclude how stable the results generated by the GA approach are.

RQ5: Is the GA-based feature selection approach robust for different datasets?

Different applications are used to create and manage tickets. Each project and application have its business domain. The goal of this investigation is to confirm if the proposed GA-based feature selection approach is not restricted to a single business domain, being therefore suitable for different applications and datasets.

4.2 Dataset

Software Project Tickets are the basis for the construction of the datasets considered in our study. Those repositories comprise tickets organized as semi-structured text documents that are expected to contain the necessary information to execute a particular action, as the implementation of a new feature, a bug fix, or even a simple code investigation.

Figure 2.2 shows a generic ticket design that is a collection of texts that are organized in attributes. There are three sections highlighted in black in the figure that are examples of unstructured text attributes: *Description*, *Summary*, and *Comments*. The red highlighted part contains a group of attributes that are predefined texts that represent the structured fields.

The decision to use tickets as the base of the datasets was straightforward because it is the most common software project activity representation in the market nowadays. As mention on Chapter 2, tickets contain all the knowledge, historical and current, of all software projects in a company.

In our project, it was decided to use tickets from Open Source projects, where we have free access to a historical knowledge base. Furthermore, the project can contribute to the community. We chose to use tickets from the most active projects of the Apache Foundation:

1. **Hadoop** (`issues.apache.org/jira/issues/?jql=project=HADOOP` – As of March 2020).
2. **Solr** (`issues.apache.org/jira/issues/?jql=project=SOLR` – As of March 2020).
3. **MapReduce** (`issues.apache.org/jira/issues/?jql=project=MAPREDUCE` – As of March 2020).
4. **Hive** (`issues.apache.org/jira/issues/?jql=project=HIVE` – As of March 2020).

Hadoop tickets were collected in November of 2017, while *MapReduce*, *Solr*, and *Hive* tickets were collected in March of 2019. All of them were read from `https://issues.apache.org/jira` (As of March 2020) website through a REST Entry-point provided by Jira. Table 4.1 shows the number of tickets for each project dataset.

Table 4.1: Datasets considered in our study.

Project Name	# of tickets in Dataset
Hadoop	1197
MapReduce	378
Solr	301
Hive	2762

4.3 Ground Truth

In our formulation, the recommendation system is implemented by means of a search system. The evaluation of different search systems rely on the definition of queries (in our case, search tickets) and a ground truth dataset, which contains a list of collection objects (in our case, other tickets) that are relevant to the query.

An essential task for the project is to define what a relevant tickey is. The relevant ticket definition will guide the evaluation metrics and the definition of suitable fitness functions for training GA. In this project, we used *siblings* tickets as relevant tickets. Therefore, to evaluate how many relevant tickets are delivered by the ticket recommendation system, we count the number of *siblings* tickets, where *siblings* tickets are subtask tickets that share the same *parent* ticket.

In a typical software project, it is normal to have tickets whose handling is complex and significant. Often, in such cases, developers break such complex tickets into smaller ones called subtasks. *Siblings* tickets (subtasks) are tickets that, by definition, are relevant to each other. They are not the same task, but they are related, and their description is relevant to its *siblings*.

Figure 4.1 illustrates how we organize tickets of a particular dataset. In the example, Dataset (D) contains a subset called Search Tickets (ST), where $ST \subset D$. D is a set that includes all tickets that have at least one *sibling*, and ST is a subset that includes all tickets that have at least ten *siblings*.

Table 4.2 shows the number of Search Tickets per Dataset considered in our evaluation protocol. Table 4.3 shows the average number of relevant tickets per search ticket and the maximum and the minimum number of relevant tickets.

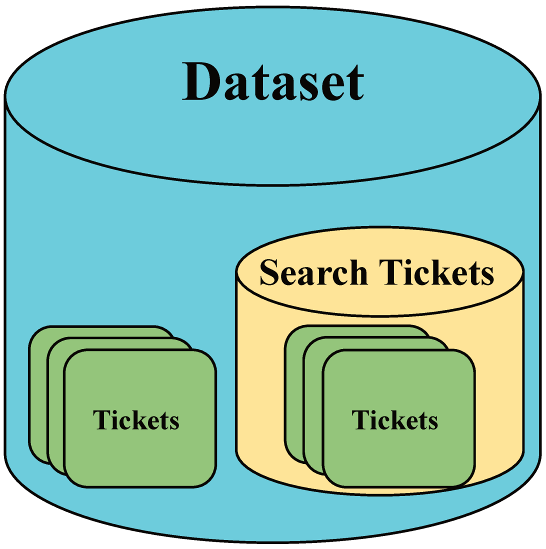


Figure 4.1: Dataset organization.

Table 4.2: Distribution of search tickets per dataset.

Project Name	# of tickets in Dataset	# of tickets in Search Tickets
Hadoop	1197	829
MapReduce	378	158
Solr	301	86
Hive	2762	2040

Table 4.3: Distribution of relevant (sibling) tickets for different datasets, including the average, the maximum, and the minimum number of relevant tickets per search ticket.

Project Name	Average	Maximum	Minimum
Hadoop	51.80	103	13
MapReduce	28.00	38	15
Solr	22.53	29	12
Hive	131.08	341	13

4.4 Conventional Ticket Recommendation System

In our study, the conventional ticket baseline recommendation system is based on ranking a ticket collection according to their similarity to an input ticket, taking advantage of the vector space model (VSM), that is a traditional information retrieval approach. In our conventional ticket recommendation system, a vector representation is constructed of each ticket document based on the frequency of terms defined in terms of the traditional TF-IDF weighting scheme. The similarity between two ticket documents is computed based on the cosine similarity function applied to their vector representation.

Figure 4.2 provides a schematic view of the conventional ticket recommendation system. The left side shows each step of the process. The right side demonstrates examples of the results that each respective step produces. Each step is detailed below:

1. Dataset:

   The first step is the ticket collection itself. As shown on the right side, the Dataset is composed of *sibling* ticket documents as described in Section 4.2.

2. Define Terms:

   Define Terms is responsible for obtaining the list of terms from all ticket documents, removing Stopwords, and indexing the tickets using the traditional Inverted Index.

   In the work, the following list of Stopwords were considered: “a”, “an”, “and”, “are”, “as”, “at”, “be”, “but”, “by”, “for”, “if”, “in”, “into”, “is”, “it”, “=”, “no”,“not”, “of”, “on”, “or”, “such”, “that”, “the”, “their”, “then”, “there”, “these”, “they”, “this”, “to”, “was”, “will”, “with”.

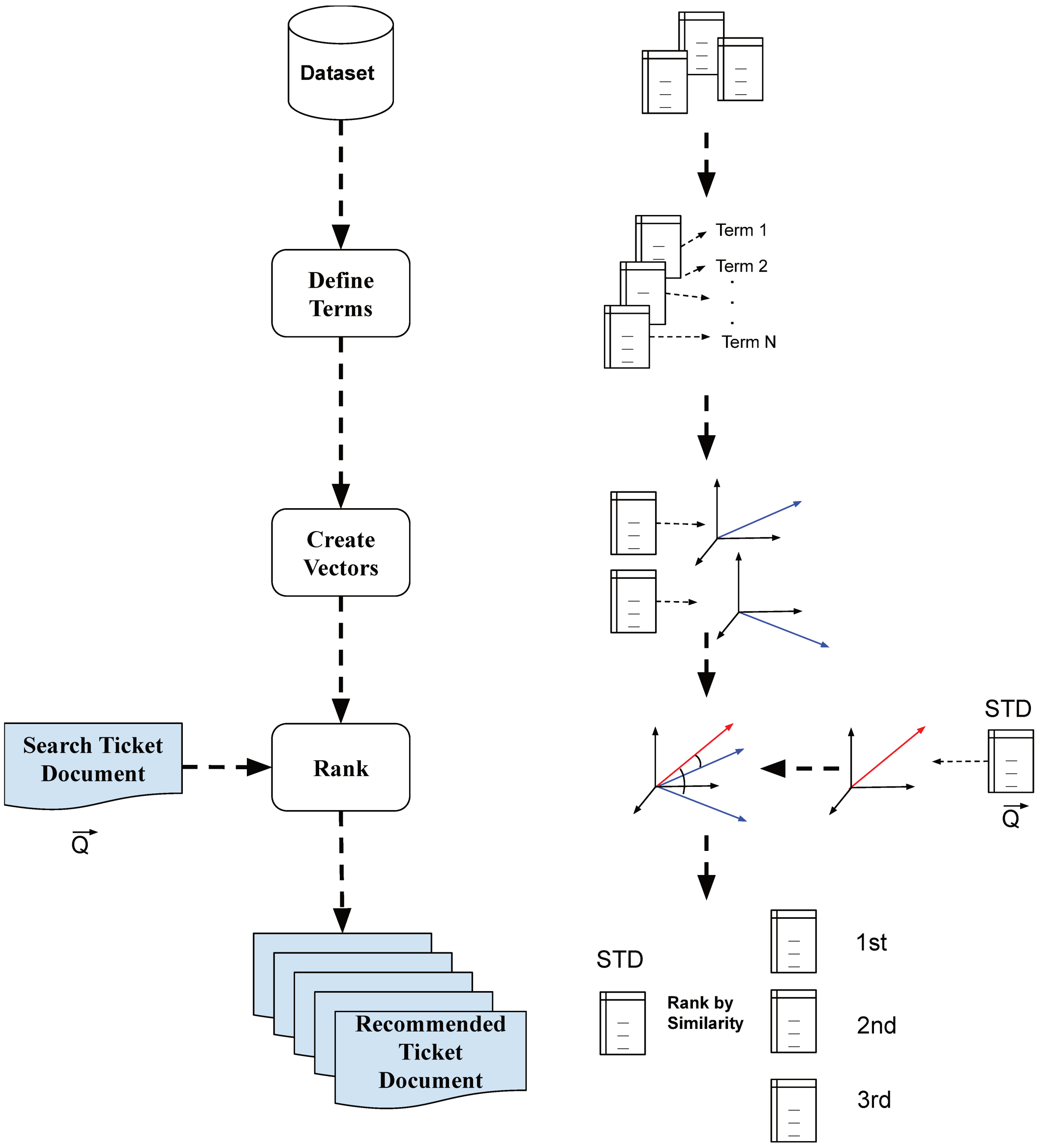


Figure 4.2: Conventional ticket recommendation system.

3. Create Vectors:

   This is the most important step in the conventional ticket recommendation system, where a mathematical representation of the ticket document is created. As shown on the right side on Figure 4.2, we create one vector for each ticket document, in such a way that the number of dimensions is the same as the number of terms defined in the previous step. Each dimension position on the vector is related to one specific term.

   Next, we calculate a weight for each term and inserts the result on the respective position on the vector. The weight is calculated using the TF-IDF weighting scheme, where the weight is related to the term frequency on the document and the collection.

4. Search Ticket Document:

   The Search Ticket Document (STD) is a single ticket document that belongs to the Search Ticket subset mentioned in Section 4.2. How $ST \subset Dataset$ a vector representation of the STD was created in the previous steps, therefore this vector will be select as the query for the Rank step.

5. Rank:

   In this step, the vectors are compared and ranked with the query vector using the cosine-similarity technique. This rank method consists in to calculate, for each ticket document, the cosine of the angle between the query vector and the document vector. The result for each comparison is a value that ranges from 0 to 1. This value is a representation of how similar one ticket is when compared to others, where 0 means being completely non-similar and 1, highly similar. A detailed description of the cosine-similarity technique can be found in Chapter 2.

6. Recommended Ticket Documents:

   The last step is the conventional ticket recommendation system result, is a list of ticket documents ranked according to relevance to the ticket query document.

4.5 Evaluation Metrics and Statistical Tests

All experiment results should be evaluated and validated, not only to confirm that a result is better than another but also to provide insights on its statistical significance. In our project, we applied standard evaluation metrics and statistical tests for text retrieval. Recall that we have modeled the ticket recommendation problem as a search task.

To evaluate the results, we compute Precision (P) metrics. Precision is the portion of relevant documents retrieved in the top-N ranked objects. Therefore, when $N = 10$ the P@10 is the percentage of relevant documents in the first ten positions of the recommended list. The full definition of Precision can be found in Chapter 2.

In our project, the Precision is used to evaluate the percentage of relevant tickets that are retrieved in the first five (P@5) and ten (P@10) positions. The Precision is calculated from the List of Recommended Tickets (Figure 4.2) for all queries, referred to as Search Tickets. The effectiveness metric used for comparison is the mean of P@5 and P@10 considering all searching tickets.

To validate the results statistical tests were applied. Statistical hypothesis test is a method where a data set obtained is compared against a data set from an idealized model. In this project we applied k-fold cross-validation and Wilcoxon signed-rank with $p > 0.05$ test to validate the results for some experiments. (As the data distribution does not have a normal distribution, the Wilcoxon is the recommended statistical hypothesis test.) The definition of k-fold cross-validation and Wilcoxon signed-rank test were presented in Chapter 2.

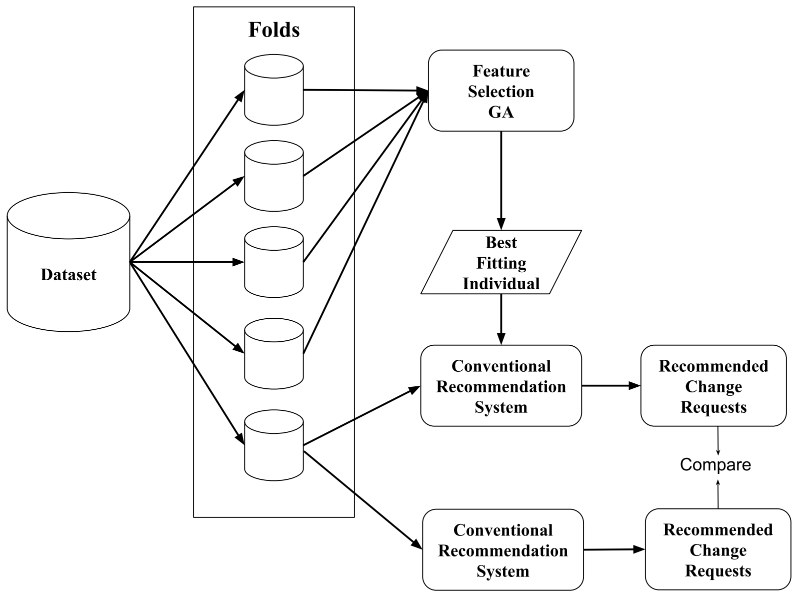

Folds
Dataset
Feature Selection GA
Best Fitting Individual
Conventional Recommendation System
Recommended Change Requests
Compare
Conventional Recommendation System
Recommended Change Requests

Figure 4.3: 5-Fold Cross-Validation

Figure 4.3 shows the 5-fold cross-validation protocol applied to the feature selection experiment. The Dataset is organized into five random small groups (*Folds*) with the same size. For each interaction, four *Folds* are selected to be used to train the Feature Selection algorithm. The result of the training is the best-fitting individual that will be used with the conventional ticket recommendation system. The validation is performed using the non-used *Fold* as input.

The designed validation protocol has the objective to compare the effectiveness of the conventional ticket recommendation system with and without the best-fitting individual. The effectiveness comparison is made using the evaluations metrics aforementioned on the Recommended Tickets.

4.6 Genetic Algorithm Parameters

The Genetic Algorithm is configurable through some parameters like the size of the population or the mutation rate. The parameters used in the experiment are displayed in Table 4.4. The number of chromosomes and the number of generations were limited due to hardware limitations. The calculation of the Fitness fuction of each individual was time-consuming. The mutation rate and the crossover approach values are the default value of the jgap library used on the implementation.

Table 4.4: Genetic Algorithm parameters.

Parameter	Value
Number of Chromosomes	40
Number of Generations	60
Mutation rate	1/12
Crossover approach	Uniform with 35% rate

Chapter 5

Experimental Results and Analysis

This chapter presents the results related to the use of the conventional ticket recommendation system, as well as the use of the proposed approaches that rely on Genetic Algorithm for feature selection. This section also provides a discussion of achieved results, considering each raised research question. The chapter is divided into two sections: Section 5.1 introduces the experiments and presents the results, while Section 5.2 discusses the obtained results.

5.1 Experiment Results

This section describes the executed experiments for each research question and shows the obtained results.

5.1.1 RQ1: Which attributes are most effective to recommend relevant tickets?

The first Research Question aims to identify which set of ticket fields is more effective in recommending tickets. To answer this question, an experiment that evaluates the use of different sets of ticket fields was created. In this experiment, we focus on using only unstructured attributes that are common in different types of ticket systems, like Jira, Trello, Bugzilla, and others.

Using fields that are commonly found in different ticket systems makes the project more valuable for researchers, developers, and practitioners interested in developing ticket recommendation systems. The experiment considered three unstructured attributes Summary, Description, Comments, and their combinations were also evaluated. The field evaluation experiment was based on the use of the conventional ticket recommendation system present in Section 4.4.

The conventional ticket recommendation system presented in Figure 4.2 was applied to this experiment. The dataset was composed of 1197 Hadoop tickets collected from the Apache Jira System, which 829 Hadoop tickets were used as search (Query Tickets), as explained in Figure 4.1 and Table 4.1. The steps described in Section 4.4 were executed seven times, each time using a different set of fields.

Each execution performed the rank step for the 829 Query Tickets, calculating the $P@5$ and $P@10$ for each query, followed by the mean for each field set. As we opted to assess the effectiveness of the recommendation systems based on the quality of the ranking up to the tenth position $P@10$, only tickets with more than ten siblings were taken as Search Ticket, as mention on Section 4.2.

Table 5.1 shows the evaluation results of each ticket field set. In this table, we report mean values for $P@5$ and $P@10$ for different configurations of the ticket field set. This table also presents the number of searches with non-relevant tickets found within the first ten positions (column *# of queries with* $P@10 = 0$).

Table 5.1: Effectiveness performance considering different ticket attributes.

Attribute Set	$P@5$	$P@10$	# of queries with $P@10 = 0$
Summary	30.90	26.43	153
Description	29.89	24.08	137
Comments	52.62	45.91	96
Summary + Description	36.60	30.77	89
Summary + Comments	54.33	47.68	76
Description + Comments	55.22	48.01	75
Summary + Description + Comments	55.95	48.77	66

5.1.2 RQ2: Is Genetic Algorithm a suitable approach for selecting appropriate terms and then improving the effectiveness of ticket recommendation systems?

The effectiveness of the proposed VSM technique based on TF-IDF with the cosine-similarity for the ticket system recommendation is highly related to the selected terms. The stopwords are an excellent example of how a set of terms could impact the results. However, discover the best combination of terms is computationally hard, e.g., the Hadoop dataset, used in our project, has around 2^{50000} possibles combinations of terms. Test all these combinations to find the best one could take thousands of years on a personal computer.

Optimization problems are well-known in the literature because of the difficulty of developing a deterministic algorithm to solve it. Different types of solutions have already been studied and proposed, and one of these solutions is the Genetic Algorithm. GA is a heuristic method that has been proved that generates high-quality results for the optimization problem. The use of this technique to find the best set of terms could improve the precision, the performance, or both of the ticket recommendation system.

By applying the GA in the project, an experiment is needed to verify if this approach is suitable for selecting appropriate terms and then improve the effectiveness of the ticket recommendation system. The experiment follows the 5-fold cross-validation model described in Section 4.5.

The validation approach described in Figure 4.3 divides the dataset into five folds with the same size, where four folds are used to training the GA and the remaining fold to validate it. The dataset used was the Hadoop described in Section 4.2, and the experiment steps are:

1. Fold creation:

   The creation of the folds consists of dividing all tickets of the dataset randomly into five folds. However, the sibling's tickets should be uniformly divided into the folds, e.g., ten sibling's tickets should be randomly divided into five groups of two. So, each fold will contain two.

2. Feature Selection:

   The Feature Selection is where the GA is applied to find the best set of terms. The process consists of using four folds as the input data for the GA presented in Chapter 3.

3. Best Fitting Individual:

   The best-fitting individual is the best combination of terms found by the GA.

4. Conventional ticket recommendation system:

   The goal is to execute the ticket recommendation system with and without the best set of terms using the remaining fold as input data. Figure 4.3 shows the ticket recommendation system two times, where one uses the best-fitting individual and the other does not. The ticket recommendation system implementation is described in Section 4.4.

5. Recommended Tickets:

   The Recommended Tickets are the results of the ticket recommendation system. The results are the mean $P@5$ and $P@10$ for each ticket query. The precision of both executions, with and without the best-fitting individual, is compared.

The previous steps are executed five times, on each time the remaining fold is a different one until all folds are the remaining at least one time. The mean $P@5$ and $P@10$ of each of the five iterations are showed in Table 5.2. Each line represents an iteration of the experiment, and the last line is the mean of the results. The Fold column shows the fold that was the remaining one used for the validation. The $P@5$ and $P@10$ Conventional Approach columns refer to the mean effectiveness results of the conventional ticket recommendation system without using the best-fitting individual. The $P@5$ and $P@10$ GA are the results using the best-fitting individual, and the % value in the parentheses is the difference between the Conventional Approach and the GA results.

Figure 5.1 shows the best-fitting individual performance by each generation on the GA. The x-axis shows the number of generations, and the y-axis shows the fitness result of the best individual. As explained in Chapter 3, the fitness algorithm is the mean $P@10$ of the ticket recommendation system using that particular set of terms, or knows as an individual. Each line represents an iteration of the training set, e.g., if the validation used the fold 1, the training set was composed of folds 2, 3, 4, and 5. The analysis and insights about the results of the experiment are presented in Section 5.2.

Table 5.2: 5-fold Experiment Result on traditional GA.

Fold	P@5 Conventional Approach	P@5 GA	P@10 Conventional Approach	P@10 GA
1	53.43%	56.42%(+5.60%)	41.19%	44.78%(+8.72%)
2	50.15%	50.75%(+1.20%)	36.72%	41.64%(+13.40%)
3	45.63%	57.81%(+26.69%)	36.09%	49.53%(+37.24%)
4	52.50%	53.13%(+1.20%)	39.84%	45.16%(+13.35%)
5	58.13%	59.06%(+1.60%)	44.84%	51.88%(+15.70%)
Mean	51.97%	55.43%(+6.66%)	39.74%	46.60%(+17.26%)

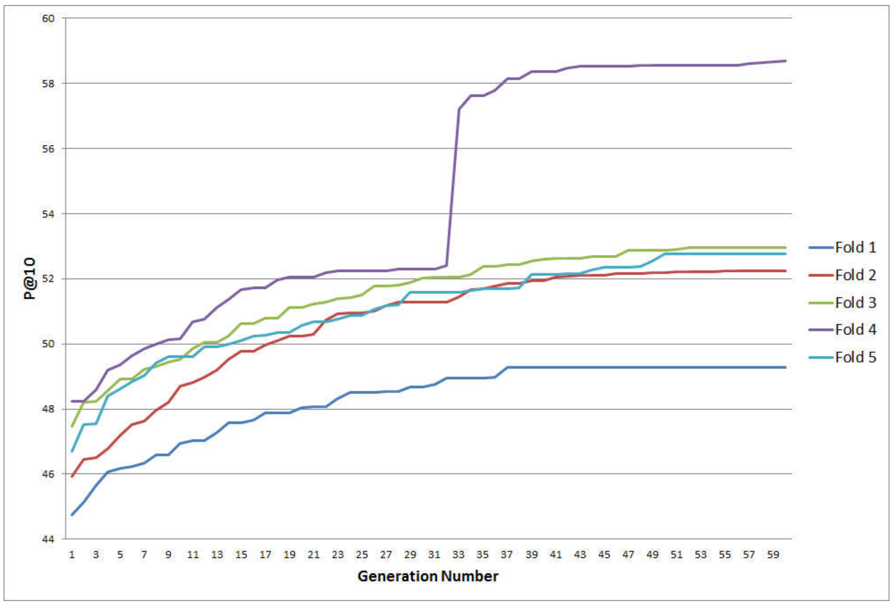


Figure 5.1: Best-fitting individual performance by number of generation.

5.1.3 RQ3: Does the GA approach based on the Hamming distance heuristic algorithm for the first generation for selecting appropriate terms improves the effectiveness and the efficiency of the ticket recommendation system?

Genetic Algorithm is a heuristic method that can be configured. The population size, the number of generations, how to create the first generation, how to perform the crossover, and the mutation rate are all configurable variables or methods in GA. There are analysis techniques that provide insights about what could be changed to improve the GA results. The performance of the best-fitting individual by generation is one of these analysis techniques. This analysis was performed on RQ2, and the insight is discussed in Section 5.2.2. Two main approaches were presented to improve the GA: use a different crossover method or add a heuristic on the creation of the first generation.

In order to address this research question, a heuristic was added to create the first generation. The goal is to demonstrate that the proposed heuristic could improve the best-fitting individual delivered by GA, therefore improving the effectiveness of the ticket recommendation system. The experiment follows the same protocol and steps of Section 5.1.2, the 5-fold cross-validation, but with a different GA implementation. The difference of the previous GA is the implementation of a heuristic based on Hamming Distance to create the first generation. The heuristic, mainly, spread the individuals of the first population on the search space, making each individual as far as possible from each one using Hamming Distance to measure the distance. The heuristic is detailed in Section 3.2.

Table 5.3 presents the mean $P@5$ for each testing fold, considering the 5-fold cross-validation protocol. We compare the conventional approach with the GA-based recommendation systems that exploit the proposed feature selection approaches (*Without* and *With*). Recall that *Without* refers to the method that does not use the proposed heuristic to guide the process of defining the GA initial population. Table 5.4, in turn, compares the results of the recommendation systems in terms of $P@10$.

Table 5.3: P@5 effectiveness performance of the Conventional approach and the GA-based recommendation systems (with and without the use of the proposed heuristic in the definition of the initial GA population).

Fold	Conventional	GA-based	GA-based
	Approach	Without	With
1	53.43%	56.42%(+5.60%)	58.21%(+8.95%)(+3.17%)
2	50.15%	50.75%(+1.20%)	50.75%(+1.20%)(+0.00%)
3	45.63%	57.81%(+26.69%)	50.31%(+10.26%)(-12.97%)
4	52.50%	53.13%(+1.20%)	59.38%(+13.10%)(+11.76%)
5	58.13%	59.06%(+1.60%)	63.75%(+9.67%)(+7.94%)
Mean	51.97%	**55.43%(+6.66%)**	**56.48%(+8.68%)(+1.89%)**

Table 5.4: P@10 effectiveness performance of the Conventional approach and the GA-based recommendation systems (with and without the use of the proposed heuristic in the definition of the initial GA population).

Fold	Conventional	GA-based	GA-based
	Approach	Without	With
1	41.19%	44.78%(+8.72%)	51.19%(+24.28%)(+14.31%)
2	36.72%	41.64%(+13.40%)	43.43%(+18.27%)(+4.30%)
3	36.09%	49.53%(+37.24%)	45.00%(+24.69%)(-9.15%)
4	39.84%	45.16%(+13.35%)	48.75%(+22.36%)(+7.95%)
5	44.84%	51.88%(+15.70%)	55.16%(+23.01%)(+6.32%)
Mean	39.74%	**46.60%(+17.26%)**	**48.71%(+22.57%)(+4.53%)**

We also evaluated the recommendation systems in terms of efficiency aspects. Table 5.5 presents the mean search time for all systems. Also this dissertation compare the recommendation systems by taking into account the size of the feature vectors (number of terms). Table 5.6 presents the results.

Table 5.5: Mean search time by query (ms) of the Conventional approach and the GA-based recommendation systems (with and without the use of the proposed heuristic in the definition of the initial GA population).

Run	Conventional Approach	GA-based Without	GA-based With
1	33.59	19.31(-42.51%)	18.84(-43.91%)(-2.43%)
2	33.76	19.37(-42.62%)	19.34(-42.71%)(-0.15%)
3	33.21	18.79(-43.42%)	18.41(-44.56%)(-2.02%)
4	33.40	21.22(-36.47%)	19.08(-42.87%)(-10.08%)
5	32.66	22.67(-30.59%)	17.82(-45.44%)(-21.39%)
Mean	33.32	**20.27(-39.17%)**	**18.70(-43.88%)(-7.74%)**

Table 5.6: Number of terms used on the vector (vector size) of the Conventional approach and the GA-based recommendation systems (with and without the use of the proposed heuristic in the definition of the initial GA population).

Run	Conventional Approach	GA-based Without	GA-based With
1,2,3 and 4	43,066	21,475(-50.13%)	15,979(-62.90%)(-25.59%)
1,2,3 and 5	40,919	20,388(-50.17%)	13,687(-66.55%)(-32.87%)
1,2,4 and 5	42,674	21,060(-50.65%)	16,490(-61.36%)(-21.70%)
1,3,4 and 5	43,564	21,555(-50.52%)	15,033(-65.49%)(-30.26%)
2,3,4 and 5	42,900	21,367(-50.19%)	13,915(-67.56%)(-39.54%)
Mean	42,624	**21,169(-50.33%)**	**15,020(-64.74%)(-29.05%)**

Figures 5.2 and 5.3 provide a summary of all results. They compare all recommendation systems, using $P@10$, and $P@5$ effectiveness measures, respectively. In both figures, the x-axis represents the average search time, considering the queries in the testing folds. The y-axis, in turn, refers to the observed effectiveness scores. The sizes of the circles encode the average size of the feature vectors (number of terms) for the different solutions. The smaller the circle, the smaller the feature vector is, i.e., the less costly a recommendation system is in terms of storage requirements. Best solutions are located in the top left quadrant, with higher precision and lower search time.

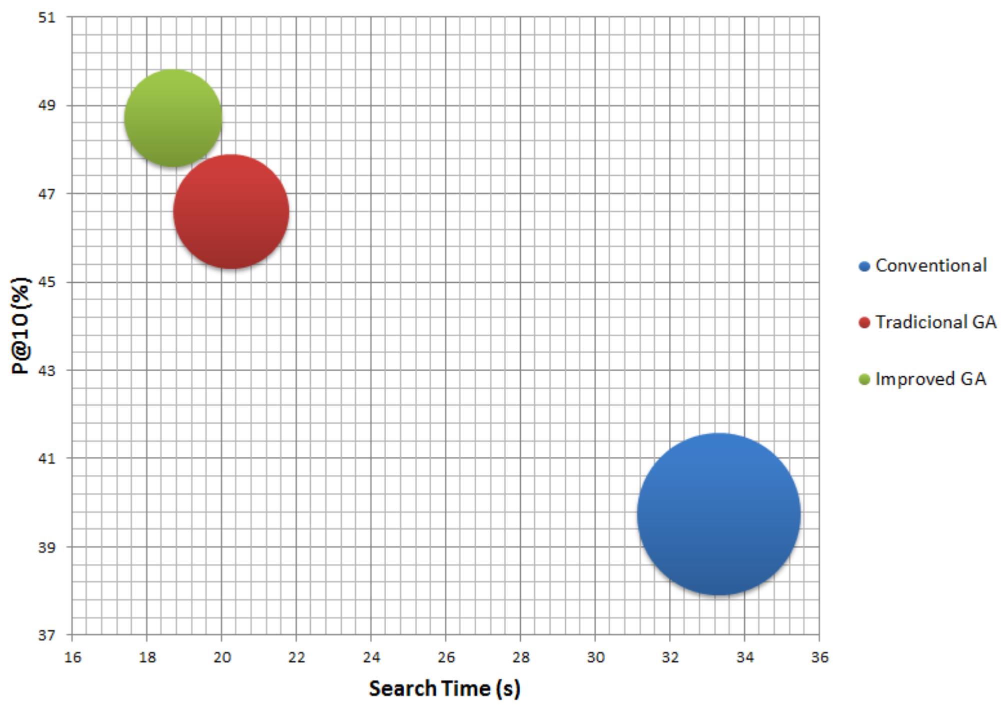


Figure 5.2: Search Time vs P@10 vs Vector Size

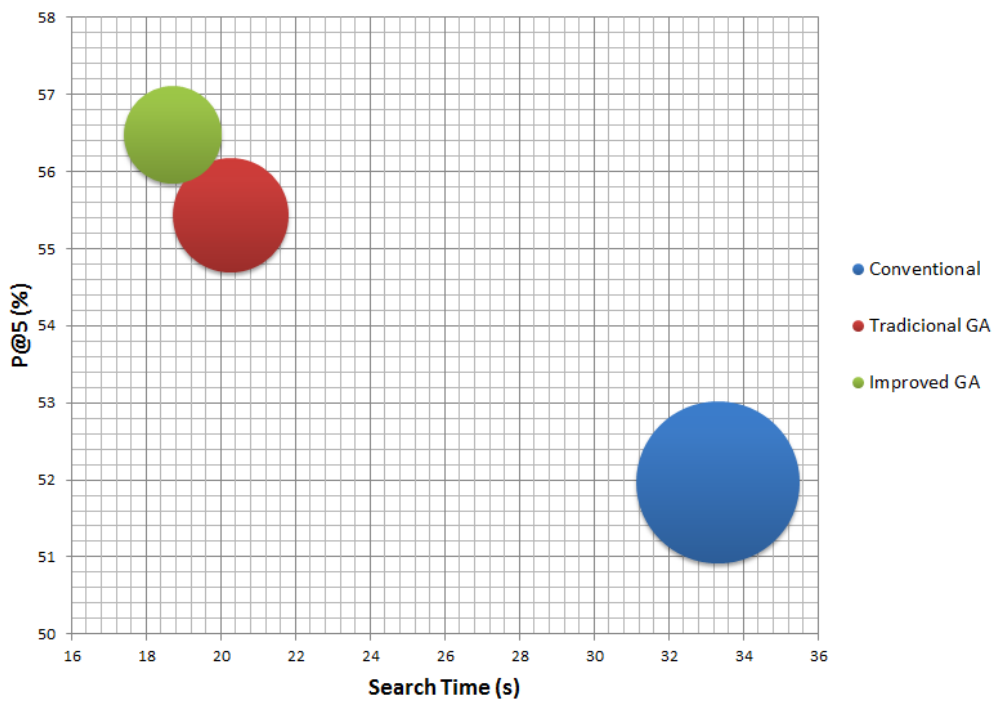


Figure 5.3: Search Time vs P@5 vs Vector Size

5.1.4 RQ4: How stable is the GA approach for selecting appropriate terms?

How GA is a non-deterministic algorithm, the necessity to validate if always generates good results is a requirement. Verifying that the proposed GA is stable endorses that the heuristic is admissible. In this research question, an experiment to assess if the GA with the heuristic is stable was developed. The designed experiment validated if the results generated by the GA are acceptable, regardless of the seed (the first generation created). For that, the 5-fold cross-validation protocol, the same implemented on RQ2 and RQ3, was applied to this research question. However, the protocol was performed three times, each time with a different random seed.

The best-fitting individual of each generation and the iteration results of the 5-fold cross-validation were collected. Figures 5.4, 5.5, 5.6, 5.7, and 5.8 show the performance of the best-fitting individual through the generations. The x-axis shows the number of generations, the y-axis displays the Fitness result (the mean $P@10$) of the best-fitting individual, and the lines illustrate an iteration of the training set. Tables 5.7, 5.8, 5.9, 5.10, and 5.11 present the results of the iterations, comparing with the conventional ticket recommendation system . The Iteration column shows the iteration number of the experiment. The $P@5$ and $P@10$ Conventional Approach columns are the average $P@5$ and $P@10$ results of the conventional ticket recommendation system without using the best-fitting individual. The $P@5$ and $P@10$ GA are the results of the recommender using the best-fitting individual, and the % value in the parentheses is the difference between the conventional ticket recommendation system and the GA results.

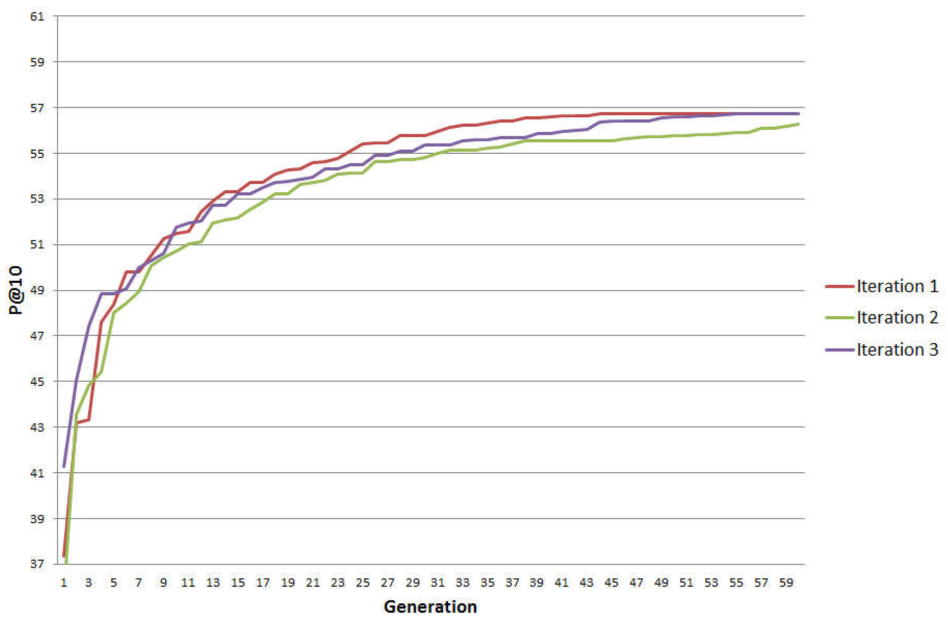


Figure 5.4: Stability test for fold 1.

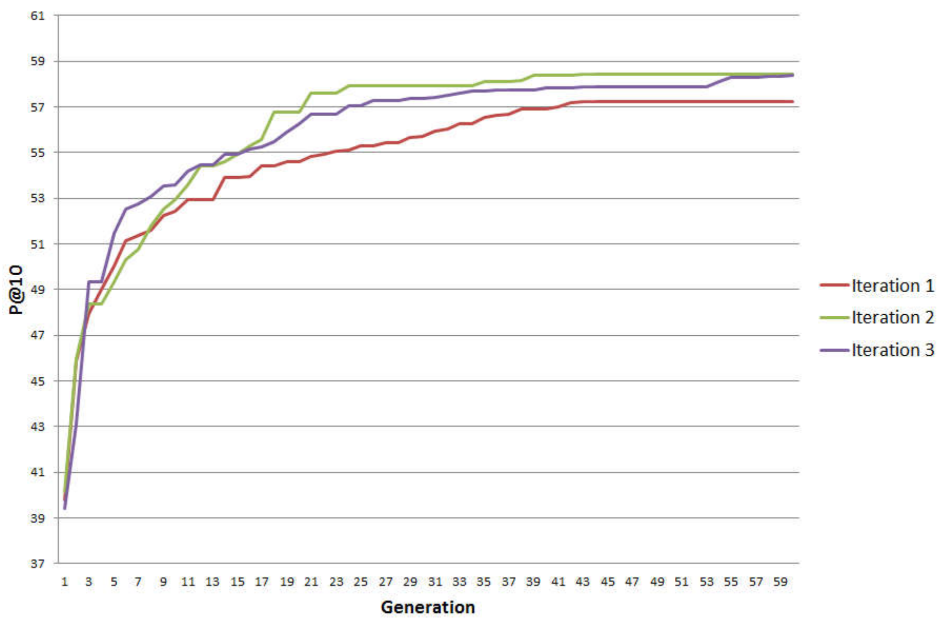


Figure 5.5: Stability test for fold 2.

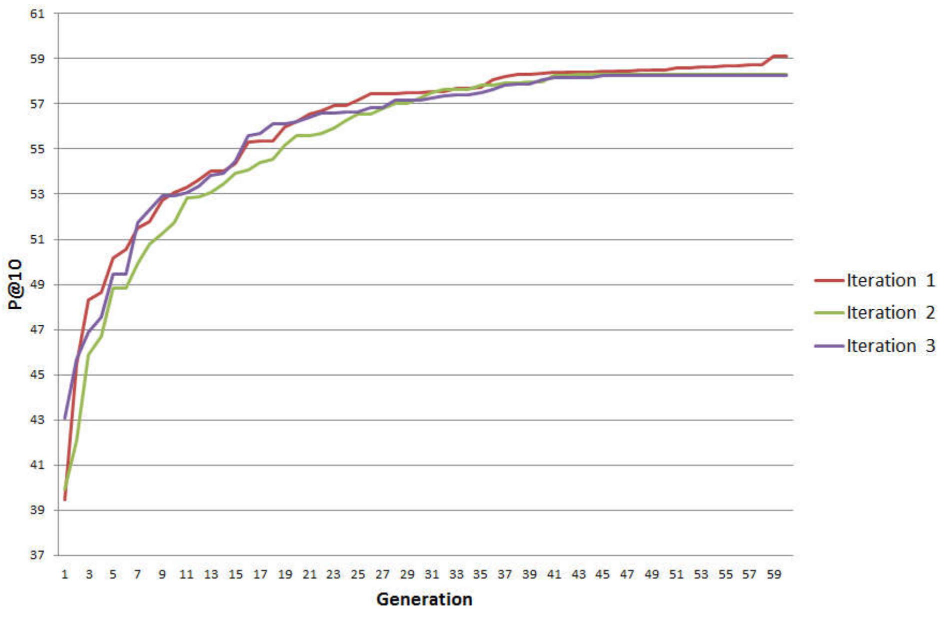


Figure 5.6: Stability test for fold 3.

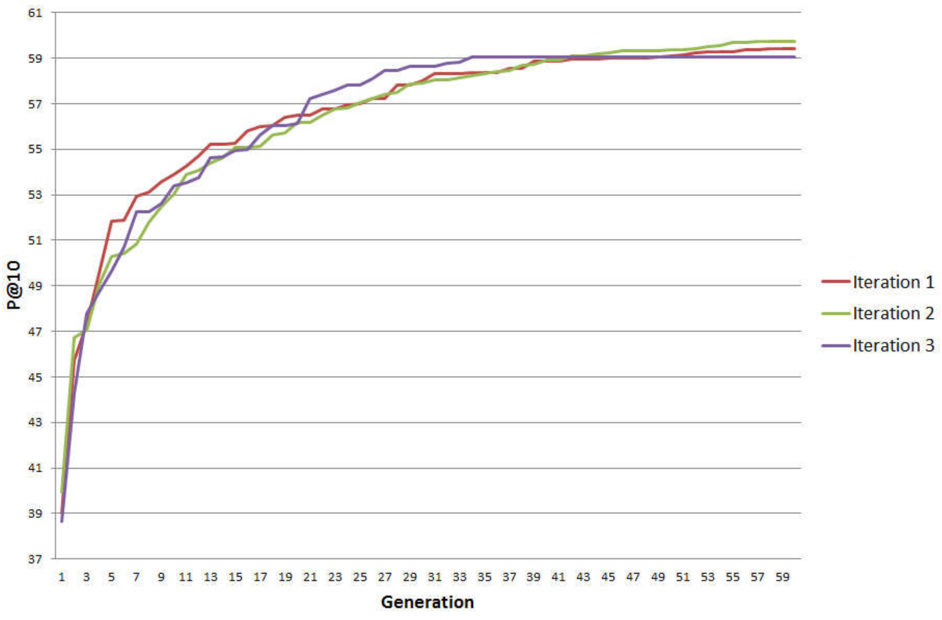


Figure 5.7: Stability test for fold 4.

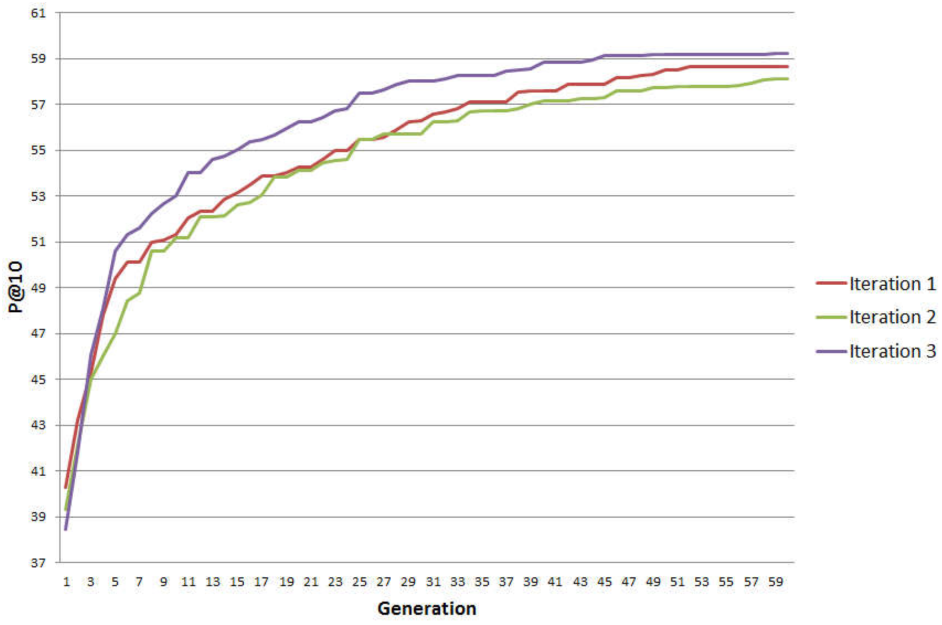


Figure 5.8: Stability test for fold 5.

Table 5.7: Stability result for fold 1.

Iteration	P@5 Conventional Approach	P@5 GA	P@10 Conventional Approach	P@10 GA
Iteration 1	53.43%	58.21%(+4.78%)	41.19%	51.19%(+10.00%)
Iteration 2	53.43%	62.39%(+8.96%)	41.19%	52.69%(+11.50%)
Iteration 3	53.43%	59.70%(+6.27%)	41.19%	47.01%(+5.82%)

Table 5.8: Stability result for fold 2.

Iteration	P@5 Conventional Approach	P@5 GA	P@10 Conventional Approach	P@10 GA
Iteration 1	50.15%	50.75%(+0.60%)	36.72%	43.43%(+6.71%)
Iteration 2	50.15%	52.54%(+2.39%)	36.72%	45.67%(+8.95%)
Iteration 3	50.15%	49.85%(-0.30%)	36.72%	42.99%(+6.27%)

Table 5.9: Stability result for fold 3.

Iteration	P@5 Conventional Approach	P@5 GA	P@10 Conventional Approach	P@10 GA
Iteration 1	45.63%	50.31%(+4.68%)	36.09%	45.00%(+8.91%)
Iteration 2	45.63%	51.25%(+5.62%)	36.09%	45.78%(+9.69%)
Iteration 3	45.63%	53.13%(+7.50%)	36.09%	47.34%(+11.25%)

Table 5.10: Stability result for fold 4.

Iteration	P@5 Conventional Approach	P@5 GA	P@10 Conventional Approach	P@10 GA
Iteration 1	52.50%	59.38%(+6.88%)	39.84%	48.75%(+8.91%)
Iteration 2	52.50%	55.94%(+3.44%)	39.84%	46.56%(+6.72%)
Iteration 3	52.50%	52.81%(+0.31%)	39.84%	45.00%(+5.16%)

Table 5.11: Stability result for fold 5.

Iteration	P@5 Conventional Approach	P@5 GA	P@10 Conventional Approach	P@10 GA
Iteration 1	58.13%	63.75%(+5.62%)	44.84%	55.16%(+10.32%)
Iteration 2	58.13%	62.50%(+4.37%)	44.84%	54.22%(+9.38%)
Iteration 3	58.13%	63.13%(+5.00%)	44.84%	55.78%(+10.94%)

5.1.5 RQ5: Is the GA-based feature selection approach robust for different datasets?

The goal of this research question is to verify if the results of the GA with the proposed heuristic is not biased. The idea is to run the RQ 3 experiment with different datasets. The datasets used were Solr, Hive, and MapReduce, presented in Section 2. Table 5.12 shows the presents the mean $P@5$ and $P@10$ of each dataset. We compare the conventional ticket recommendation system with the GA-based recommendation system with the heuristic. Table 5.13 shows the size of the feature vectors (number of terms) used.

Table 5.12: Precision results for different datasets.

Dataset	$P@5$ Conventional Approach	$P@5$ GA	$P@10$ Conventional Approach	$P@10$ GA
Hive	49.68%	64.16%(+29.15%)	43.76%	54.27%(+24.02%)
Solr	69.07%	79.53%(+15.14%)	53.37%	73.26%(+37.27%)
MapReduce	74.94%	86.58%(+15.53%)	67.72%	80.38%(+18.69%)

Table 5.13: Vector sizes for different datasets

Dataset	Conventional Vector Size	GA Vector Size
Hive	97,331	35,453(-63.57%)
Solr	16,915	3,911(-76.88%)
MapReduce	16,992	4,809(-71.70%)

5.2 Discussion

This section discusses achieved results from each experiment and provides an analysis of these results with regard to the raised Research Questions. Each of the following sections is related to a Research Question.

5.2.1 RQ1: Which attributes are most effective to recommend relevant tickets?

Table 5.1 presents the results concerning the use of attributes. As we can observe, the attribute Comments yielded the best results when compared to the use of the other ones in isolation. Also, the more attributes are used, the better the effectiveness scores observed. Other insights are:

1. The use of the Summary attribute led to better precision scores than the use of attribute Description.

   The Description is an attribute that contains all the work requirements related to the ticket and should be the most informative attribute. The Summary attribute is a small title that gives the subject of the ticket. However, even with more information, the Description precision score was slightly worse than the Summary.

The lack of information in software requirements is one of the main causes of rework or delays on a software project. The constant missing information on Description attributes could impact on finding relevant tickets, as the results show. In contrast, even with not enough amount of information but with a precise definition of the subject of the ticket, using the Summary instead of the Description is more effective to recommend relevant tickets.

2. The use of the Comment attribute yielded better precision scores than the use of Summary and Description.

   Comment attribute was expected to provide complementary information to the Description attribute, but what happened is that, frequently, comments had more relevant information than the description. This behavior is not suprising when we analyze software projects nowadays. Software companies and Open Source project environments are trying to increase human interaction instead of using extensive documentation; and to support learning by interaction instead of relying on long planning sessions [15, 20, 46, 58]. All these environments' innovations reflect on how the documents, like tickets, are created.

   Comments contain daily descriptions of what was done, providing information about the current status of the ticket. There are discussions regarding the ticket requirements, possible solutions, and implementation explanation. Because of this new era of software development [20], the Comment attribute is becoming more relevant than any other attribute on the ticket, and the results presented in Table 5.1 confirm this behavior.

3. The use of all attributes led to the best effectiveness results.

   The use of all attributes did not significantly improve the figures obtained when using the Comment attribute combined with Summary or Description, around 0.7% improvement for the $P@5$ and $P@10$. However, the number of no relevant tickets in the first ten positions was reduced by 12 percentual points. As the execution time is not impacted when using all attributes, and the use of all of them reduce the number of no relevant tickets, we conclude that using all attributes is more effective to recommend relevant tickets.

5.2.2 RQ2: Is Genetic Algorithm a suitable approach for selecting appropriate terms and then improving the effectiveness of ticket recommendation systems?

The $P@5$ and $P@10$ improvement in Table 5.2 helps to understand the effectiveness of the GA on the ticket recommendation system. The increase of 6.66% and 17.26% on $P@5$ and $P@10$ respectively, shows that the GA is a suitable feature selection approach to select a better set of terms. However, GA is a configurable method and the use of a different configuration could improve even more the selected terms.

Figure 5.1 gives us more insights into what could be changed to improve the GA. We can see that after the fiftieth generation, the best-fitting individual performance of each iteration becomes stable. This behavior on the graph indicates that the GA was capable of discovering an "optimal" solution. Therefore, even increasing the number of generations or the number of individuals, the result will be the same or close. However, the "optimal" solution could be a local minimum, as explained in Chapter 2.

As we can see, the Fold 4 iteration had a strange behavior around the thirty-second generation. There is a big jump on the performance of the best individual. This performance change may happen because of the mutation factor on the GA, which makes the algorithm explore distant individuals. The GA goal is to avoid a minimum local result. However, this jump on Fold 4 result could indicate that the GA was on local maxima and not on the global maximum.

5.2.3 RQ3: Does the GA approach based on the Hamming distance heuristic algorithm for the first generation for selecting appropriate terms improves the effectiveness and the efficiency of the ticket recommendation system?

As we can observe in Table 5.3, the use of feature selection led to very effective results for all runs, with gains of 6.66% and 8.68%, for the mean $P@5$ of the GA-based without and with heuristic, respectively. Also, we can observe that the use of the heuristic method led to results that are 1.89% better in terms of mean $P@5$ when compared to the other GA feature selection approach (*Without*). Similarly to what was observed for $P@5$, the use of GA-based feature selection procedures led to better results when compared with the conventional ticket recommendation system (17.26% and 22.57% better for the GA-based without and with heuristics, respectively). Again, the use of heuristics led to the best effectiveness results.

As we can observe in Table 5.5, the use of GA reduced the search time up to almost 45% in the case of the method, which exploits heuristics in the definition of the initial population. Also, the recommendation with the proposed heuristic was 7.74% faster than the one which does not rely on a conventional strategy for creating the initial population. Table 5.6 shows that the GA-based methods were less costly than the conventional ticket recommendation system, requiring 50.34% and 64.74% fewer terms for the the GA-based *without* and *with* heuristics, respectively. We can also observe that the use of heuristics was the less expensive method in terms of storage requirements.

Figures 5.3 and 5.2 show that the GA-based solution which considers the proposed heuristic was capable of selecting suitable terms (using only 36% of all terms), which led to improvements in terms of both effectiveness and efficiency: the precision was improved by almost 23% and the search time reduced 44% when compared with the conventional ticket recommendation system. Also, the feature vector size reduced considerably (about 65%).

5.2.4 RQ4: How stable is the GA approach for selecting appropriate terms?

The analysis focus on verify the stability of the proposed G.A with the first generation heuristics. Figures 5.4 to 5.8 and Tables 5.7 to 5.11 show positive results that assist in confirming it. During the analysis, we come with the following key points:

1. The first generations' performance improves faster than the following generations.

   Figures 5.4 to 5.8 show that the P@10 of the best-fitting individual had an average improvement of 33.11% during the first ten generations. However, in the next twenty generations, the average improvement was 7.23%. Comparing the GA without the heuristic in Figure 5.1, the first ten generations, the average P@10 was 5.09%, and in the next twenty generations was 3.99%. These results endorse that the heuristic is finding a global minimum more frequently. With the first population spread through the search space, during the first generations, the local minimal are evaluated faster, lending to the global minimum.

2. The last generations are stable.

   Figures 5.4 to 5.8 show the performance stability of the best-fitting Individual in the last twenty generations. In some iteration and fold, the best-fitting individual did not improve during some generations.

5.2.5 RQ5: Is the GA-based feature selection approach robust for different datasets?

Tables 5.12 and 5.13 show that even with different datasets, the GA with the heuristic improved the effectiveness of the ticket recommendation system, and reduced the term vector size in more than 60%. An interesting result in Table 5.13 is that the GA is capable of selecting fewer terms when the list of terms is smaller. The Hive with 97k terms was reduced around 64%, and the Solr and MapReduce with 16k terms were reduced around 70%.

The Curse of Dimensionality is evident in these results. The improvement on datasets with a more significant number of terms, Hive and Hadoop, are higher than datasets with a smaller number of terms, Solr and MapReduce. The datasets, with a large number of dimensions, are in the most right position in Figure 2.6. Therefore, with each fewer dimensions, the shift to the left of the curve leads to more significant improvement. However, when the number dimensions is small, the position on the curve is close to the optimal, making the growth smaller.

5.3 Typical Usage Scenario

This section provides an overview of a typical usage scenario concerning the use of recommendation systems implemented used some of the evaluated search systems. In the examples tickets used in this section are from the HADOOP dataset and prints are presented on Appendix A.

The ticket **HADOOP-12753**: *S3A JUnit tests failing if using HTTP proxy* is taken as the search tickets. This ticket has 81 possible relevant (sibling) tickets and concerns a task of fixing a failing test related to the use of a HTTP proxy to access external file systems, such as Amazon S3A. The provided description has enough information about the problem, even including some system logs. The comments focus on the approval of changes rather than a discussion about the target problem itself. Furthermore, the comments have some automatically generated messages as detailed in Appendix A.

By using this ticket as input query, we expect to receive from the recommendation systems tickets that, for example, worked on the same test, implemented the test, or performed changes on the access of the file systems S3A.

Table 5.14: Top-5 ranked tickets considering **HADOOP-12753**: *S3A JUnit tests failing if using HTTP proxy* as search ticket. Tickets highlighted in bold are relevant (siblings).

Rank	Conventional Approach	Heuristic GA
1	**HADOOP-12804:** ***Read Proxy Password from Credential Providers in S3 FileSystem***	**HADOOP-12804:** ***Read Proxy Password from Credential Providers in S3 FileSystem***
2	**HADOOP-13130:** ***s3a failures can surface as RTEs, not IOEs***	**HADOOP-13130:** ***s3a failures can surface as RTEs, not IOEs***
3	**HADOOP-13131:** ***Add tests to verify that S3A supports SSE-S3 encryption***	**HADOOP-13599:** ***s3a close() to be non-synchronized, so avoid risk of deadlock on shutdown***
4	HADOOP-13110: *add a streaming subcommand to mapred*	**HADOOP-12444:** ***Support lazy seek in S3AInputStream***
5	HADOOP-13019: *Implement Erasure-Codec for HitchHiker XOR coding*	**HADOOP-13131:** ***Add tests to verify that S3A supports SSE-S3 encryption***

Table 5.14 shows the top-five tickets returned as relevant tickets by the conventional recommendation system and by the recommendation system which relies on the use of the heuristic GA. Appendix A shows a screenshot of all listed tickets.

Looking at Table 5.14, when we use the conventional ticket recommendation system without the GA, only the top-three tickets are relevant (siblings) to the input search ticket. The first three tickets (**HADOOP-12804:** *Read Proxy Password from Credential Providers in S3 FileSystem*; **HADOOP-13130:** *s3a failures can surface as RTEs, not IOEs*; and **HADOOP-13131:** *Add tests to verify that S3A supports SSE-S3 encryption*) concern the use of the S3A file system. However, the last two recommended tickets (**HADOOP-13110:** *add a streaming subcommand to mapred* and **HADOOP-13019:** *Implement ErasureCodec for HitchHiker XOR coding*), do not have any similarity to the search ticket, handling non-related issues.

The ticket recommendation system that uses the heuristic GA, in turn, was capable of exploiting more meaningful terms (due the selection process), being therefore able to rank more relevant tickets in the top positions, as shown in Table 5.14. All five tickets placed in top-5 positions are relevant (siblings). All of them concern the use of the S3A file system.

Chapter 6

Conclusions

Most software development environments of large applications use ticket management systems. Especially in Open Source projects, where numerous developers are engaged in the development, the number of tickets is in the range of tens of thousand items. Retrieving valuable information buried in such an immense database without the help of automation is virtually impossible. The typical way to go about automating this task is via recommendation systems based on the widely-used Vector Space Model (VSM) using TF-IDF weights with cosine similarity. This solution is inherently costly as it deals with high-dimensional feature vectors, that are proportional to the number of terms found in the ticket repository. In this work, we address this problem by proposing and testing two solutions based on the Genetic Algorithm.

The implemented solutions use the VSM in combination with a feature selection technique based on GA, which made possible working with a smaller dimension feature vector, thus reducing complexity and processing time, without compromising the ranking precision. This work shows that GA is an effective approach to select suitable features to support the recommendation of change requests. Furthermore, we also present a novel approach as an improvement built on GA algorithm by applying a heuristic algorithm based on Hamming distance to generate the first generation of the feature vector population. As a result, the measured precision increased by almost 23%, the average search time by query was reduced by almost 44%, and the vector size was reduced by 64%, showing that this GA-based approach for building recommendation systems for change request repositories is a promising approach.

Possible improvements were identified during the development of the research such as:

1. Use different information retrieval techniques (e.g., BM25, BM25F, and BM25+) in the implementation of the fitness function. A different technique could improve the effectiveness of the recommendation by ranking more relevant tickets or reducing the processing time of the GA.

2. Improve the TF-IDF evaluation by removing terms with a lower value. Terms with lower TF-IDF tend to be less relevant, removing them before the feature selection phase could improve the GA effectiveness and generate smaller vectors.

3. Extend the heuristic algorithm to be used in the implementation of genetic opera-

tors (e.g., crossover and mutation). It was showed in this dissertation that genetic diversity improves the result of the GA. Validate if this diversity is beneficial in other phases of the GA is a promising research venue.

4. Improve the processing time of the GA or use better hardware to be capable of increasing the number of chromosomes on each generation and the number of generations. Test the proposed heuristic with a faster GA with more chromosomes and generation may increase the likelihood that is not resulting only on local minimum.

5. Investigate the use of different GA parameter values. Other configurations could increase the effectiveness and efficiency of ticket recommendation systems.

6. Use structured fields to increase the precision of the ticket recommendation system. Tickets usually have fields that created relevance links between them. These links are created by users that already notice relevance between these tickets.

7. Extend the recommendation system to recommend relevant code, users, teams, forum threads together with the recommended tickets. Sometimes, even when the ticket is relevant, there is not enough information to help the user. Adding more information as the developer or team that worked on the recommended ticket could be helpful.

8. Add a heuristic on the crossover phase that reduces and increases the number of selected features on the children. This heuristic may be defined in terms of the proximity of the optional number of dimensions of the Curse of Dimensionality.

9. Add user feedback to identify false positives of relevant tickets. The goal would be to include the users' background knowledge in the identification of the most suitable features, improving the effectiveness of recommendation systems.

10. Explore different steps, such as the "Training of a Classification Model" in Figure 2.12, to increase the effectiveness of the ticket recommendation system.

11. Investigate the ideal number of features for text classifiers regardless context. By analyzing the results of the ticket recommendation system with and without the GA, it was observed that the use of less terms results in better precision. Despite the dataset, using or not GA, and the total number of features, the precision was higher when the number of terms was lower.

12. Investigate if the proposed heuristic improves the results in those different applications, such as financial and medical studies, is another promising research venue.

13. Develop the theoretical foundations related to the use of the proposed heuristic. Investigate theoretically if and how the proposed heuristic increases the probability of any Genetic Algorithm approach to find the global minimum/maximum could be investigated in future work.

14. Investigate the use of the proposed recommendation system for assisting the development of traceability matrices. A traceability matrix is often used by companies to track tickets that could impact other tickets. For example, ticket A could impact tickets B, C, and D. Find what ticket could be impacted is often a time-consuming and subjective manual task.

The ticket recommendation system is an exciting tool to improve knowledge sharing. The proposed approaches have been demonstrated to lead to effective and efficient recommenders. Therefore, developed methods are promising solutions to help developers in their daily tasks related to the proper maintenance of software projects.

Bibliography

[1] O. Abayomi-Alli, V. O. Matthews S. Misra, M. Odusami, A. Abayomi-Alli, R. Ahuja, and R. Maskeliunas. An Improved Feature Selection Method for Short Text Classification. *Journal of Physics: Conference Series*, 1235, 2019.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005.

[3] M. C. Annosi, M. Magnusson, A. Martini, and F. P. Appio. Social Conduct, Learning and Innovation: An Abductive Study of the Dark Side of Agile Software Development. *Creativity and Innovation Management*, 25:515–535, 2016.

[4] A. Aurum, F. Daneshgar, and J. Ward. Investigating Knowledge Management practices in software development organisations - An Australian experience. *Information and Software Technology*, 50:511–533, 2008.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval the concepts and technology behind search.* Pearson, 1999.

[6] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, 1999.

[7] R. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[8] N. Bidi and Z. Elberrichi. Feature selection for text classification using genetic algorithms. *8th International Conference on Modelling, Identification and Control (ICMIC)*, 2016.

[9] F. O. Bjørnson and T. Dingsøyr. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50:1055–1068, 2008.

[10] Y. Bo and Q. Luo. Personalized web information recommendation algorithm based on support vector machine. *The 2007 International Conference on Intelligent Pervasive Computing*, page 487–490, 2007.

[11] A. Boden, G. Avram, L. Bannon, and V. Wulf. Knowledge Management in Distributed Software Development Teams - Does Culture Matter? *Fourth IEEE International Conference on Global Software Engineering*, 2009.

[12] D. A. Bray. Literature Review - Knowledge Management Research at the Organizational Level. *Harvard Business Review*, 2007.

[13] B. Cabral and P. Marques. Exception Handling: A Field Study in Java and .NET. *ECOOP 2007 – Object-Oriented Programming*, 4609:151–175, 2007.

[14] R. M. Chao, J.-T. Huang, and C.-W. Yang. The study of knowledge service-oriented recommendation mechanism - a case of e-learning platform. *International Conference on Machine Learning and Cybernetics*, 4:2228–2233, 2005.

[15] A. Cockburn and J. Highsmith. Agile software development: The people factor. *Computer*, 34:131–133, 2001.

[16] K. Conboy. Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research*, 20:317–480, 2009.

[17] T. H. Davenport. Saving IT's Soul: Human Centered Information Management. *Harvard Business Review*, pages 119–131, 1994.

[18] T. H. Davenport, L. Prusak, and L. Prusak. *Working Knowledge: How Organizations Manage What They Know.* Harvard Business School Press, 1997.

[19] X. Deng, Y. Li, J. Weng, and J. Zhang. Feature selection for text classification: A review. *Multimedia Tools and Applications*, 78:3797–3816, 2019.

[20] S. Dorairaj, J. Noble, and P. Malik. Knowledge Management in Distributed Agile Software Development. *Agile Conference*, 2012.

[21] D. Dönmez and G. Grote. The Practice of Not Knowing for Sure: How Agile Teams Manage Uncertainties. *XP: International Conference on Agile Software Development*, 149:61–75, 2013.

[22] K. Beck et al. Manifesto for agile software development. `http://agilemanifesto.org/`. Accessed: 2020-02-29.

[23] A. S. Ghareb, A. A. Bakar, and A. R. Hamdan. Hybrid feature selection based on enhanced genetic algorithm for text categorization. *Expert Systems with Applications*, 49:31–47, 2016.

[24] M. A. Hall. Correlation-based feature selection of discrete and numeric class machine learning. *17th Conference on Machine Learning*, page 359–366, 200.

[25] A. S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. *31st annual international ACM SIGIR conference on Research and development in information retrieval*, page 91–98, 2008.

[26] J. H. Holland. *Adaptation in natural and artificial systems.* MIT Press, 1975.

[27] E. R. Hruschka and T. F. Covoes. Feature Selection for Cluster Analysis: an Approach Based on the Simplified Silhouette Criterion. *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, pages 28–30, 2005.

[28] Z. Huang, H. Chen, and D. Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems*, 22:116–142, 2004.

[29] Z. Huang, W. Chung, T.-H. Ong, and H. Chen. A graph-based recommender system for digital library. *2nd ACM/IEEE-CS joint conference on Digital libraries*, page 65–73, 2002.

[30] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60:493–502, 2004.

[31] L. Li, C. R. Weinberg, T. A. Darden, and L. G. Pedersen. Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the GA/KNN method. *Bioinformatics*, 17:1131–1142, 2001.

[32] H. Liu, E .R. Dougherty, J. G. Dy, K. Torkkola, E. Tuv, H. Peng, C. Ding, F. Long, M. Berens, L. Parsons, L. Yu, Z. Zhao, and G Forman. Evolving feature selection. *IEEE Intelligent Systems*, 20:64–76, 2005.

[33] X. Liu, B. Shen, H. Zhong, and J. Zhu. EXPSOL: Recommending Online Threads for Exception-Related Bug Reports. *23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016.

[34] H. P. Luhn. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, 1:309–317, 1957.

[35] M. V. Mantyla, J. Vanhanen, and C. Lassenius. Bad smells - humans as code critics. *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, 2004.

[36] A. Martin, R. Biddle, and J. Noble. The XP Customer Team: A Grounded Theory. *Agile Conference*, 2009.

[37] G. Melnik and F. Maurer. Direct verbal communication as a catalyst of agile knowledge sharing. *Agile Development Conference*, 2004.

[38] A. L. Menolli, A. Malucelli, and S. Reinehr. Towards a Semantic Social Collaborative Environment for Organization Learning. *The 7th International Conference on Information Technology and Applications*, 2011.

[39] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The Work Life of Developers: Activities, Switches and Perceived Productivity. *IEEE Transactions on Software Engineering*, 43:1178–1193, 2017.

[40] F. J. Milliken. Three Types of Perceived Uncertainty About the Environment: State, Effect, and Response Uncertainty. *Academy of Management Review*, 12:133–143, 1987.

[41] M. M. Mirończuk and J. Protasiewicz. A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, 106:36–54, 2018.

[42] H. Munir, P. Runeson, and K. Wnuk. A theory of openness for software engineering tools in software organizations. *Information and Software Technology*, 97:26–45, 2018.

[43] S. R. Nidumolu. Standardization, requirements uncertainty and software project performance. *Information & Management*, 31:135–150, 1996.

[44] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Inovation.* Oxford University Press, 1995.

[45] K. Olmos-Sánchez and J. Rodas-Osollo. Requirements Engineering Based on Knowledge Management: Theoretical Aspects and a Practical Proposal. *International Journal of Software Engineering and Knowledge Engineering*, 27:1199–1233, 2017.

[46] R. A. B. Ouriques, K. Wnuk, T. Gorschek, and R. B. Svensson. Knowledge Management Strategies and Processes in Agile Software Development: A Systematic Literature Review. *International Journal of Software Engineering and Knowledge Engineering*, 29:345–380, 2019.

[47] R. L. Z. Panaggio. Arcabouço genérico baseado em técnicas de agrupamento para sistemas de recomendação. Master's thesis, Instituto de Computação, Universidade Estadual de Campinas, 2010.

[48] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1226–1238, 2005.

[49] M. A. Razzak and R. Ahmed. Knowledge sharing in distributed agile projects: Techniques, strategies and challenges. *Federated Conference on Computer Science and Information Systems*, 2:1431–1440, 2014.

[50] B. Regnell and S. Brinkkemper. Market-Driven Requirements Engineering for Software Products. *Engineering and Managing Software Requirements*, pages 287–308, 2005.

[51] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. *1994 ACM conference on Computer supported cooperative work*, page 175–186, 1994.

[52] S. Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60:503–520, 2004.

[53] M. Robnik-Šikonja and I. Kononenko. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning*, 53:23–69, 2003.

[54] I. Rus, M. Lindvall, and S. S. Sinha. A State of the Art Report: Knowledge Management in Software Engineering. *IEEE Software*, 19:26–38, 2002.

[55] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18:613–620, 1975.

[56] G. Salton and C. S. Yang. On the specification of term values in automatic indexing. *Journal of Documentation*, 29:351–372, 1973.

[57] G. M. Santos. Retrieving curated stack overflow of similar project tasks. Mphil thesis, Instituto de Matemática, COPPE Universidade Federal do Rio de Janeiro, 2018.

[58] V. Santos, A. Goldman, H. Filho, D. Martins, and M. Cortés. The Influence of Organizational Factor on InterTeam Knowledge Sharing Effectiveness in Agile Environments. *47th Hawaii International Conference on System Science*, 2014.

[59] S. Sayed, M. Nassef, A. Badr, and I. Farag. A Nested Genetic Algorithm for feature selection in high-dimensional cancer Microarray datasets. *Expert Systems with Applications*, 121:2033–243, 2019.

[60] P. M. Senge. *The Fifth Discipline*. Currency, 1990.

[61] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani. Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation. *10th Working Conference on Mining Software Repositories (MSR)*, 2013.

[62] R .Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani. A time-based approach to automatic bug report assignment. *Journal of Systems and Software*, 102:109–122, 2015.

[63] A. Sillitti, M. Ceschi, B. Russo, and G. Succi. Managing uncertainty in requirements: a survey in documentation-driven and agile companies. *11th IEEE International Software Metrics Symposium (METRICS'05)*, 2005.

[64] S. F. Silva. *Seleção de características por meio de algoritmos genéticos para aprimoramento de ranking e de modelos de classificação*. PhD thesis, Instituto de Ciências Matemáticas e de Computação, Universidade Estadual de São Paulo, 2011.

[65] X. Sun, H. Yang, H. Leung, B. Li, H. J. Li, and L. Liao. Effectiveness of exploring historical commits for developer recommendation: an empirical study. *Frontiers of Computer Science*, 12:528–544, 2018.

[66] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li. An Empirical Study on Real Bugs for Machine Learning Programs. *24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017.

[67] S. Vasanthapriyan, J. Tian, and J. Xiang. A Survey on Knowledge Management in Software Engineering. *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*, 2015.

[68] T. Wang, G. Yin H. Wang, C. Yang, and P. Zou. Linking stack overflow to issue tracker for issue resolution. *6th Asia-Pacific Symposium on Internetware*, page 11–14, 2014.

[69] B. Wu, L. Qi, and X. Feng. Personalized recommendation algorithm based on SVM. *International Conference on Communications, Circuits and Systems*, page 951–953, 2007.

[70] H. Yang, X. Sun, B. Li, and J. Hu. Recommending developers with supplementary information for issue request resolution. *38th International Conference on Software Engineering Companion*, page 707–709, 2016.

[71] J. Yang and V. Honavar. Feature Subset Selection Using a Genetic Algorithm. *The Springer International Series in Engineering and Computer Science*, 453, 1998.

[72] K. Yu, A. Schwaighofer, V. Tresp, X. Xu, and H. P. Kriegel. Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering*, 16:56–69, 2004.

[73] L. Yu and H. Liu. Efficient Feature Selection via Analysis of Relevance and Redundancy. *Machine Learning Research*, 5:1205–1224, 2004.

[74] W. Chung Z. Huang and H. Chen. A graph model for e-commerce recommender systems. *American Society for Information Science and Technology*, 55:259–274, 2004.

[75] Z. Zhu, Y. Ong, and M. Dash. Wrapper–Filter Feature Selection Algorithm Using a Memetic Framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37:70–76, 2007.

[76] Z. Zhu, Y. Onga, and M. Dash. Markov blanket-embedded genetic algorithm for gene selection. *Pattern Recognition*, 40:3236–3248, 2007.

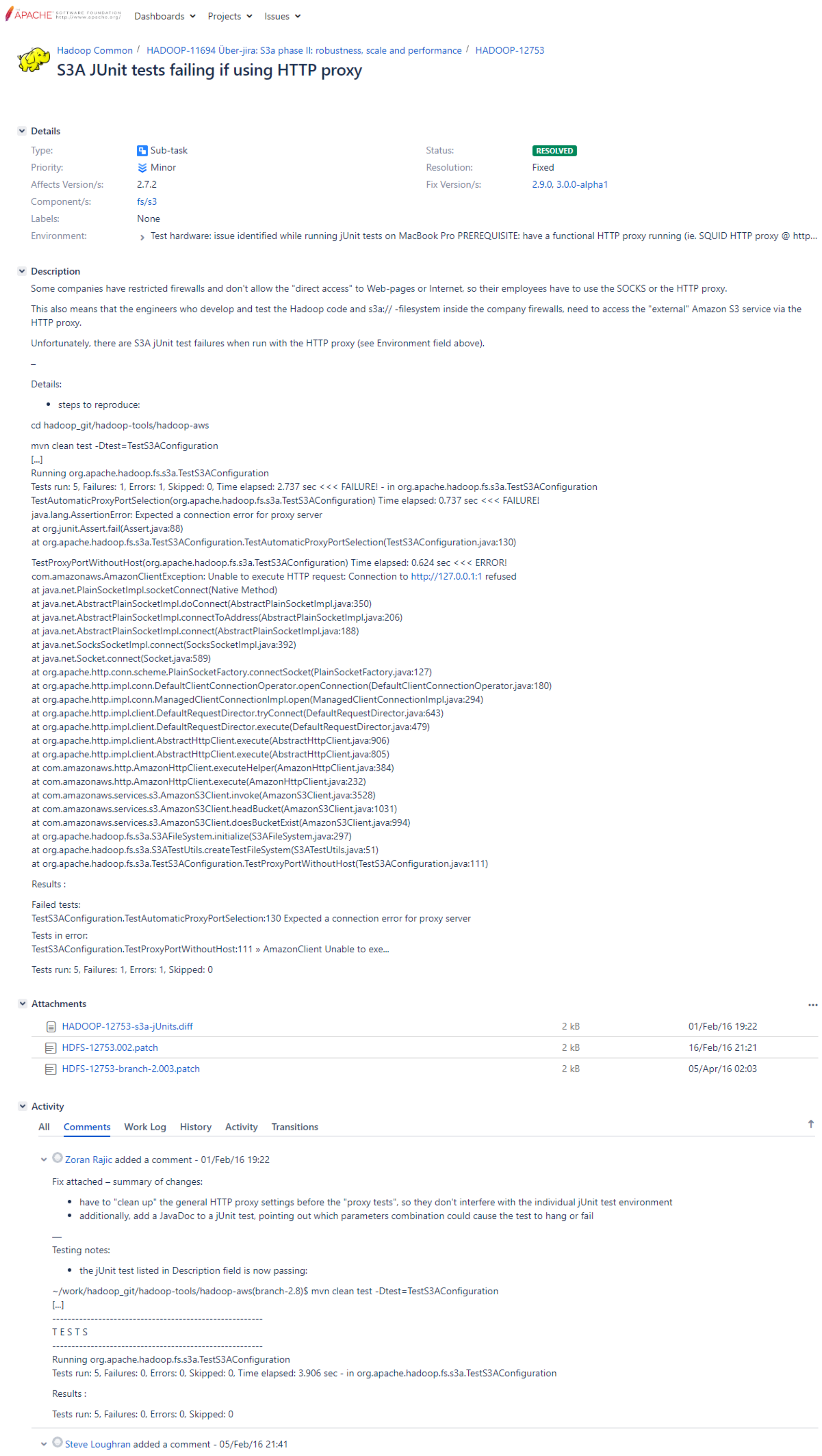
Appendix A

Typical examples of HADOOP tickets

This Appendix presents examples of HADOOP Jira tickets. These prints are not the full content of the tickets. To see all the ticket content, the reader may refer to the provided link. The tickets presented in this Appendix are as follows:

Table A.1: Search Ticket: HADOOP-12753.

Returned Tickets	by Conventional Approach	by Heuristic GA
HADOOP-12804	✓	✓
HADOOP-13130	✓	✓
HADOOP-13131	✓	✓
HADOOP-13599		✓
HADOOP-12244		✓
HADOOP-13110	✓	
HADOOP-13019	✓	



Hadoop Common / HADOOP-11694 Über-jira: S3a phase II: robustness, scale and performance / HADOOP-12753

S3A JUnit tests failing if using HTTP proxy

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Minor	Resolution:	Fixed
Affects Version/s:	2.7.2	Fix Version/s:	2.9.0, 3.0.0-alpha1
Component/s:	fs/s3		
Labels:	None		
Environment:	› Test hardware: issue identified while running jUnit tests on MacBook Pro PREREQUISITE: have a functional HTTP proxy running (ie. SQUID HTTP proxy @ http...		

Description

Some companies have restricted firewalls and don't allow the "direct access" to Web-pages or Internet, so their employees have to use the SOCKS or the HTTP proxy.

This also means that the engineers who develop and test the Hadoop code and s3a:// -filesystem inside the company firewalls, need to access the "external" Amazon S3 service via the HTTP proxy.

Unfortunately, there are S3A jUnit test failures when run with the HTTP proxy (see Environment field above).

–

Details:

- steps to reproduce:

```
cd hadoop_git/hadoop-tools/hadoop-aws

mvn clean test -Dtest=TestS3AConfiguration
[...]
Running org.apache.hadoop.fs.s3a.TestS3AConfiguration
Tests run: 5, Failures: 1, Errors: 1, Skipped: 0, Time elapsed: 2.737 sec <<< FAILURE! - in org.apache.hadoop.fs.s3a.TestS3AConfiguration
TestAutomaticProxyPortSelection(org.apache.hadoop.fs.s3a.TestS3AConfiguration) Time elapsed: 0.737 sec <<< FAILURE!
java.lang.AssertionError: Expected a connection error for proxy server
at org.junit.Assert.fail(Assert.java:88)
at org.apache.hadoop.fs.s3a.TestS3AConfiguration.TestAutomaticProxyPortSelection(TestS3AConfiguration.java:130)

TestProxyPortWithoutHost(org.apache.hadoop.fs.s3a.TestS3AConfiguration) Time elapsed: 0.624 sec <<< ERROR!
com.amazonaws.AmazonClientException: Unable to execute HTTP request: Connection to http://127.0.0.1:1 refused
at java.net.PlainSocketImpl.socketConnect(Native Method)
at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
at java.net.Socket.connect(Socket.java:589)
at org.apache.http.conn.scheme.PlainSocketFactory.connectSocket(PlainSocketFactory.java:127)
at org.apache.http.impl.conn.DefaultClientConnectionOperator.openConnection(DefaultClientConnectionOperator.java:180)
at org.apache.http.impl.conn.ManagedClientConnectionImpl.open(ManagedClientConnectionImpl.java:294)
at org.apache.http.impl.client.DefaultRequestDirector.tryConnect(DefaultRequestDirector.java:643)
at org.apache.http.impl.client.DefaultRequestDirector.execute(DefaultRequestDirector.java:479)
at org.apache.http.impl.client.AbstractHttpClient.execute(AbstractHttpClient.java:906)
at org.apache.http.impl.client.AbstractHttpClient.execute(AbstractHttpClient.java:805)
at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.java:384)
at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:232)
at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:3528)
at com.amazonaws.services.s3.AmazonS3Client.headBucket(AmazonS3Client.java:1031)
at com.amazonaws.services.s3.AmazonS3Client.doesBucketExist(AmazonS3Client.java:994)
at org.apache.hadoop.fs.s3a.S3AFileSystem.initialize(S3AFileSystem.java:297)
at org.apache.hadoop.fs.s3a.S3ATestUtils.createTestFileSystem(S3ATestUtils.java:51)
at org.apache.hadoop.fs.s3a.TestS3AConfiguration.TestProxyPortWithoutHost(TestS3AConfiguration.java:111)

Results :

Failed tests:
TestS3AConfiguration.TestAutomaticProxyPortSelection:130 Expected a connection error for proxy server

Tests in error:
TestS3AConfiguration.TestProxyPortWithoutHost:111 » AmazonClient Unable to exe...

Tests run: 5, Failures: 1, Errors: 1, Skipped: 0
```

Attachments

HADOOP-12753-s3a-jUnits.diff	2 kB	01/Feb/16 19:22
HDFS-12753.002.patch	2 kB	16/Feb/16 21:21
HDFS-12753-branch-2.003.patch	2 kB	05/Apr/16 02:03

Activity

Zoran Rajic added a comment - 01/Feb/16 19:22

Fix attached – summary of changes:

- have to "clean up" the general HTTP proxy settings before the "proxy tests", so they don't interfere with the individual jUnit test environment
- additionally, add a JavaDoc to a jUnit test, pointing out which parameters combination could cause the test to hang or fail

—

Testing notes:

- the jUnit test listed in Description field is now passing:

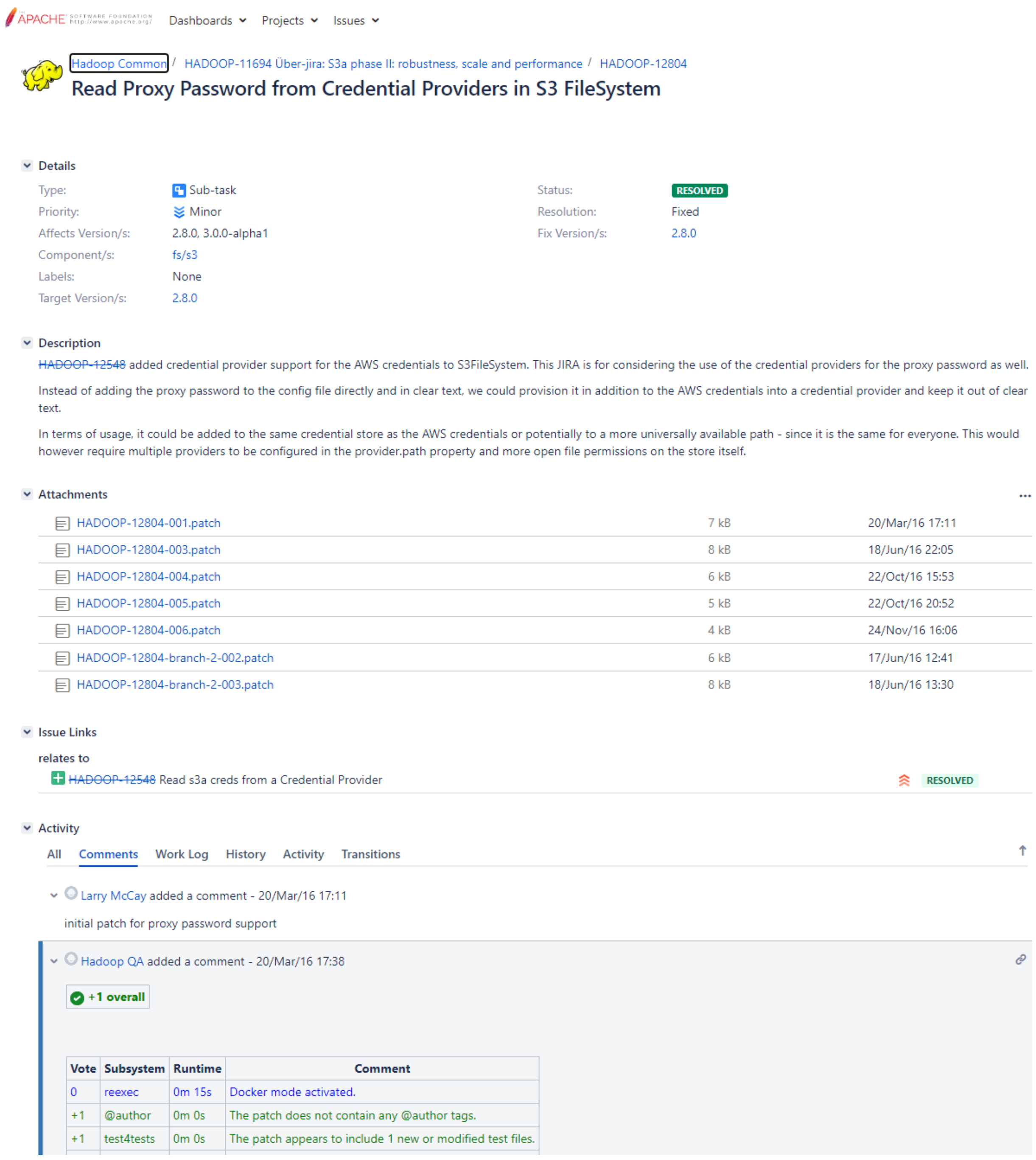
```
~/work/hadoop_git/hadoop-tools/hadoop-aws(branch-2.8)$ mvn clean test -Dtest=TestS3AConfiguration
[...]
-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running org.apache.hadoop.fs.s3a.TestS3AConfiguration
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.906 sec - in org.apache.hadoop.fs.s3a.TestS3AConfiguration

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
```

Steve Loughran added a comment - 05/Feb/16 21:41

Figure A.1: HADOOP-12753 (`https://issues.apache.org/jira/browse/HADOOP-12753` – As of June 2020).



Dashboards Projects Issues

Hadoop Common / HADOOP-11694 Über-jira: S3a phase II: robustness, scale and performance / HADOOP-12804

Read Proxy Password from Credential Providers in S3 FileSystem

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Minor	Resolution:	Fixed
Affects Version/s:	2.8.0, 3.0.0-alpha1	Fix Version/s:	2.8.0
Component/s:	fs/s3		
Labels:	None		
Target Version/s:	2.8.0		

Description

~~HADOOP-12548~~ added credential provider support for the AWS credentials to S3FileSystem. This JIRA is for considering the use of the credential providers for the proxy password as well.

Instead of adding the proxy password to the config file directly and in clear text, we could provision it in addition to the AWS credentials into a credential provider and keep it out of clear text.

In terms of usage, it could be added to the same credential store as the AWS credentials or potentially to a more universally available path - since it is the same for everyone. This would however require multiple providers to be configured in the provider.path property and more open file permissions on the store itself.

Attachments

HADOOP-12804-001.patch	7 kB	20/Mar/16 17:11
HADOOP-12804-003.patch	8 kB	18/Jun/16 22:05
HADOOP-12804-004.patch	6 kB	22/Oct/16 15:53
HADOOP-12804-005.patch	5 kB	22/Oct/16 20:52
HADOOP-12804-006.patch	4 kB	24/Nov/16 16:06
HADOOP-12804-branch-2-002.patch	6 kB	17/Jun/16 12:41
HADOOP-12804-branch-2-003.patch	8 kB	18/Jun/16 13:30

Issue Links

relates to

~~HADOOP-12548~~ Read s3a creds from a Credential Provider RESOLVED

Activity

All Comments Work Log History Activity Transitions

Larry McCay added a comment - 20/Mar/16 17:11

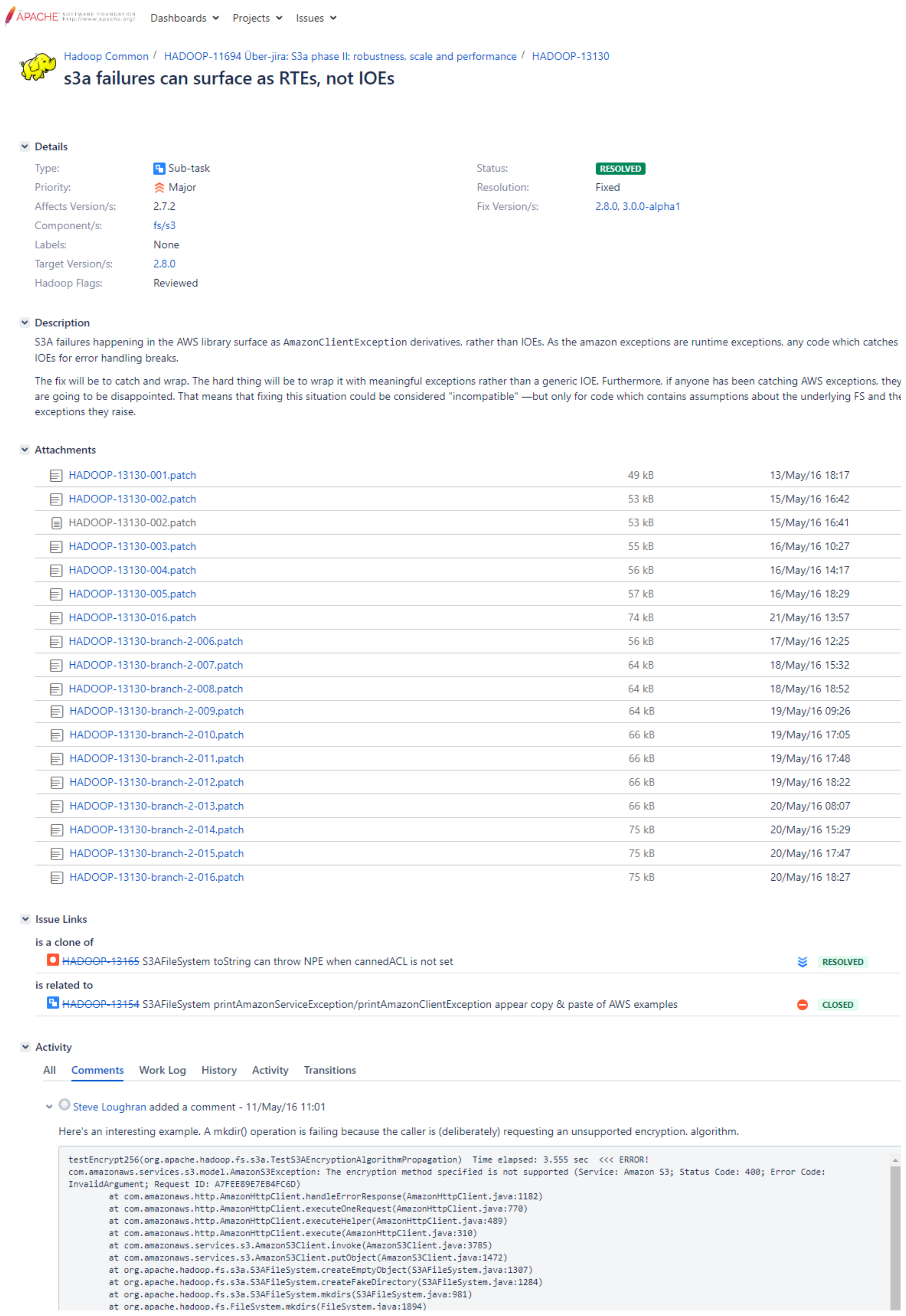
initial patch for proxy password support

Hadoop QA added a comment - 20/Mar/16 17:38

+1 overall

Vote	Subsystem	Runtime	Comment
0	reexec	0m 15s	Docker mode activated.
+1	@author	0m 0s	The patch does not contain any @author tags.
+1	test4tests	0m 0s	The patch appears to include 1 new or modified test files.

Figure A.2: HADOOP-12804 (`https://issues.apache.org/jira/browse/HADOOP-12804` – As of June 2020).



Hadoop Common / HADOOP-11694 Über-jira: S3a phase II: robustness, scale and performance / HADOOP-13130

s3a failures can surface as RTEs, not IOEs

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.7.2	Fix Version/s:	2.8.0, 3.0.0-alpha1
Component/s:	fs/s3		
Labels:	None		
Target Version/s:	2.8.0		
Hadoop Flags:	Reviewed		

Description

S3A failures happening in the AWS library surface as `AmazonClientException` derivatives, rather than IOEs. As the amazon exceptions are runtime exceptions, any code which catches IOEs for error handling breaks.

The fix will be to catch and wrap. The hard thing will be to wrap it with meaningful exceptions rather than a generic IOE. Furthermore, if anyone has been catching AWS exceptions, they are going to be disappointed. That means that fixing this situation could be considered "incompatible" —but only for code which contains assumptions about the underlying FS and the exceptions they raise.

Attachments

HADOOP-13130-001.patch	49 kB	13/May/16 18:17
HADOOP-13130-002.patch	53 kB	15/May/16 16:42
HADOOP-13130-002.patch	53 kB	15/May/16 16:41
HADOOP-13130-003.patch	55 kB	16/May/16 10:27
HADOOP-13130-004.patch	56 kB	16/May/16 14:17
HADOOP-13130-005.patch	57 kB	16/May/16 18:29
HADOOP-13130-016.patch	74 kB	21/May/16 13:57
HADOOP-13130-branch-2-006.patch	56 kB	17/May/16 12:25
HADOOP-13130-branch-2-007.patch	64 kB	18/May/16 15:32
HADOOP-13130-branch-2-008.patch	64 kB	18/May/16 18:52
HADOOP-13130-branch-2-009.patch	64 kB	19/May/16 09:26
HADOOP-13130-branch-2-010.patch	66 kB	19/May/16 17:05
HADOOP-13130-branch-2-011.patch	66 kB	19/May/16 17:48
HADOOP-13130-branch-2-012.patch	66 kB	19/May/16 18:22
HADOOP-13130-branch-2-013.patch	66 kB	20/May/16 08:07
HADOOP-13130-branch-2-014.patch	75 kB	20/May/16 15:29
HADOOP-13130-branch-2-015.patch	75 kB	20/May/16 17:47
HADOOP-13130-branch-2-016.patch	75 kB	20/May/16 18:27

Issue Links

is a clone of

~~HADOOP-13165~~ S3AFileSystem toString can throw NPE when cannedACL is not set RESOLVED

is related to

~~HADOOP-13154~~ S3AFileSystem printAmazonServiceException/printAmazonClientException appear copy & paste of AWS examples CLOSED

Activity

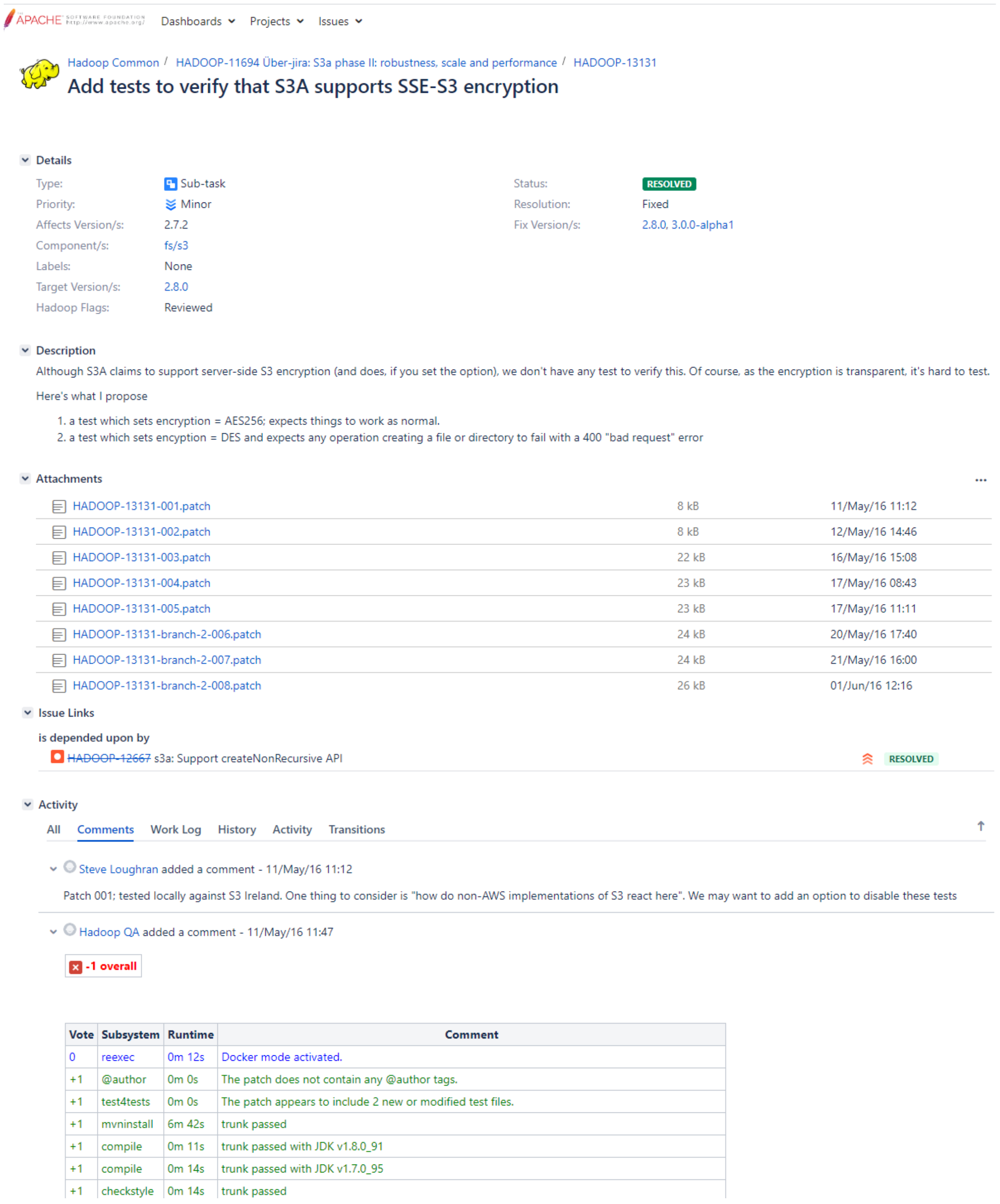
All Comments Work Log History Activity Transitions

Steve Loughran added a comment - 11/May/16 11:01

Here's an interesting example. A mkdir() operation is failing because the caller is (deliberately) requesting an unsupported encryption. algorithm.

```
testEncrypt256(org.apache.hadoop.fs.s3a.TestS3AEncryptionAlgorithmPropagation)  Time elapsed: 3.555 sec  <<< ERROR!
com.amazonaws.services.s3.model.AmazonS3Exception: The encryption method specified is not supported (Service: Amazon S3; Status Code: 400; Error Code:
InvalidArgument; Request ID: A7FEE89E7EB4FC6D)
	at com.amazonaws.http.AmazonHttpClient.handleErrorResponse(AmazonHttpClient.java:1182)
	at com.amazonaws.http.AmazonHttpClient.executeOneRequest(AmazonHttpClient.java:770)
	at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.java:489)
	at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:310)
	at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:3785)
	at com.amazonaws.services.s3.AmazonS3Client.putObject(AmazonS3Client.java:1472)
	at org.apache.hadoop.fs.s3a.S3AFileSystem.createEmptyObject(S3AFileSystem.java:1307)
	at org.apache.hadoop.fs.s3a.S3AFileSystem.createFakeDirectory(S3AFileSystem.java:1284)
	at org.apache.hadoop.fs.s3a.S3AFileSystem.mkdirs(S3AFileSystem.java:981)
	at org.apache.hadoop.fs.FileSystem.mkdirs(FileSystem.java:1894)
```

Figure A.3: HADOOP-13130 (`https://issues.apache.org/jira/browse/HADOOP-13130` – As of June 2020).



Hadoop Common / HADOOP-11694 Über-jira: S3a phase II: robustness, scale and performance / HADOOP-13131

Add tests to verify that S3A supports SSE-S3 encryption

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Minor	Resolution:	Fixed
Affects Version/s:	2.7.2	Fix Version/s:	2.8.0, 3.0.0-alpha1
Component/s:	fs/s3		
Labels:	None		
Target Version/s:	2.8.0		
Hadoop Flags:	Reviewed		

Description

Although S3A claims to support server-side S3 encryption (and does, if you set the option), we don't have any test to verify this. Of course, as the encryption is transparent, it's hard to test.

Here's what I propose

1. a test which sets encryption = AES256; expects things to work as normal.
2. a test which sets encyption = DES and expects any operation creating a file or directory to fail with a 400 "bad request" error

Attachments

HADOOP-13131-001.patch	8 kB	11/May/16 11:12
HADOOP-13131-002.patch	8 kB	12/May/16 14:46
HADOOP-13131-003.patch	22 kB	16/May/16 15:08
HADOOP-13131-004.patch	23 kB	17/May/16 08:43
HADOOP-13131-005.patch	23 kB	17/May/16 11:11
HADOOP-13131-branch-2-006.patch	24 kB	20/May/16 17:40
HADOOP-13131-branch-2-007.patch	24 kB	21/May/16 16:00
HADOOP-13131-branch-2-008.patch	26 kB	01/Jun/16 12:16

Issue Links

is depended upon by

~~HADOOP-12667~~ s3a: Support createNonRecursive API RESOLVED

Activity

Steve Loughran added a comment - 11/May/16 11:12

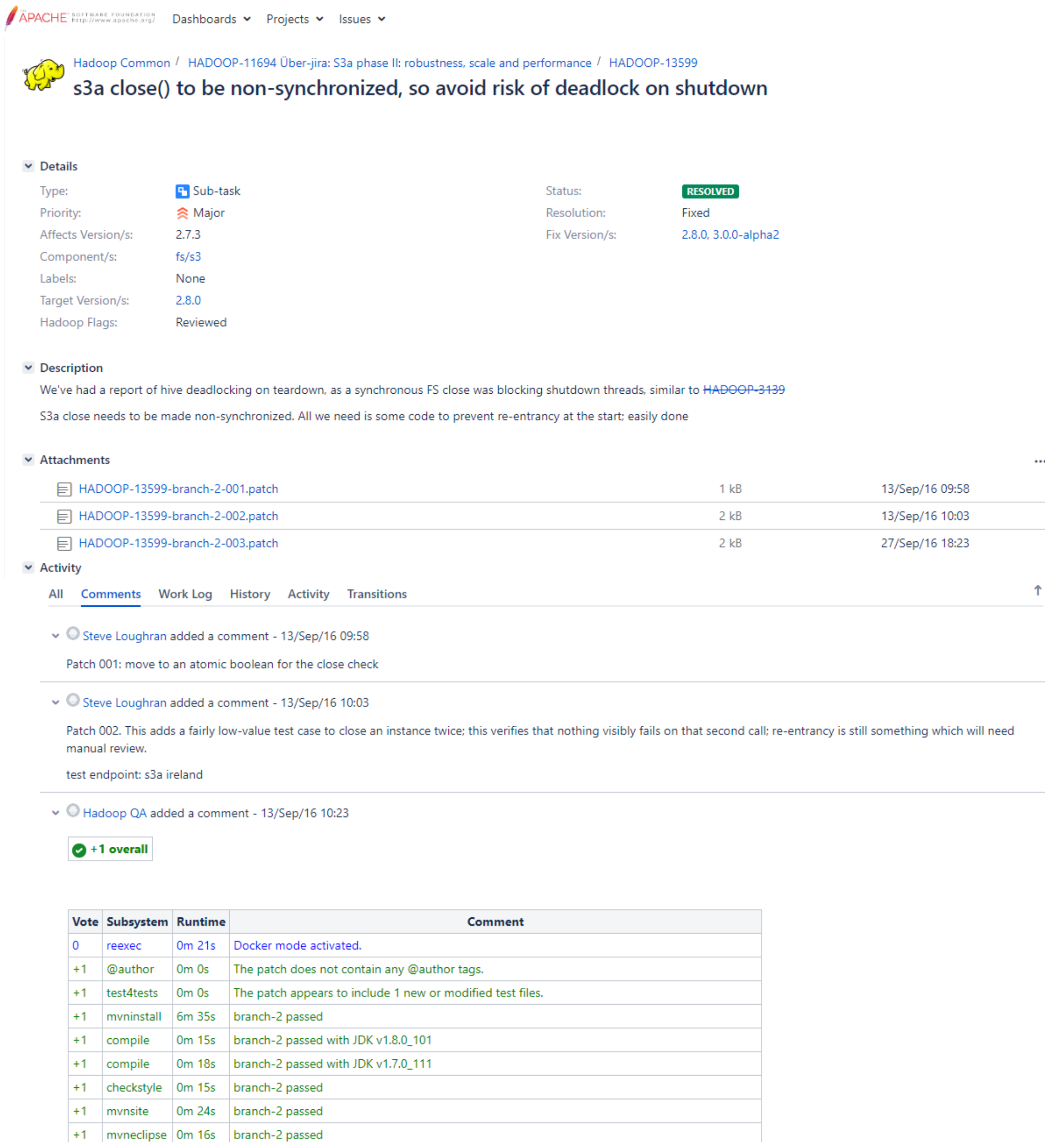
Patch 001; tested locally against S3 Ireland. One thing to consider is "how do non-AWS implementations of S3 react here". We may want to add an option to disable these tests

Hadoop QA added a comment - 11/May/16 11:47

-1 overall

Vote	Subsystem	Runtime	Comment
0	reexec	0m 12s	Docker mode activated.
+1	@author	0m 0s	The patch does not contain any @author tags.
+1	test4tests	0m 0s	The patch appears to include 2 new or modified test files.
+1	mvninstall	6m 42s	trunk passed
+1	compile	0m 11s	trunk passed with JDK v1.8.0_91
+1	compile	0m 14s	trunk passed with JDK v1.7.0_95
+1	checkstyle	0m 14s	trunk passed

Figure A.4: HADOOP-13131 (`https://issues.apache.org/jira/browse/HADOOP-13131` – As of June 2020).



Hadoop Common / HADOOP-11694 Über-jira: S3a phase II: robustness, scale and performance / HADOOP-13599

s3a close() to be non-synchronized, so avoid risk of deadlock on shutdown

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.7.3	Fix Version/s:	2.8.0, 3.0.0-alpha2
Component/s:	fs/s3		
Labels:	None		
Target Version/s:	2.8.0		
Hadoop Flags:	Reviewed		

Description

We've had a report of hive deadlocking on teardown, as a synchronous FS close was blocking shutdown threads, similar to ~~HADOOP-3139~~

S3a close needs to be made non-synchronized. All we need is some code to prevent re-entrancy at the start; easily done

Attachments

HADOOP-13599-branch-2-001.patch	1 kB	13/Sep/16 09:58
HADOOP-13599-branch-2-002.patch	2 kB	13/Sep/16 10:03
HADOOP-13599-branch-2-003.patch	2 kB	27/Sep/16 18:23

Activity

Steve Loughran added a comment - 13/Sep/16 09:58

Patch 001: move to an atomic boolean for the close check

Steve Loughran added a comment - 13/Sep/16 10:03

Patch 002. This adds a fairly low-value test case to close an instance twice; this verifies that nothing visibly fails on that second call; re-entrancy is still something which will need manual review.

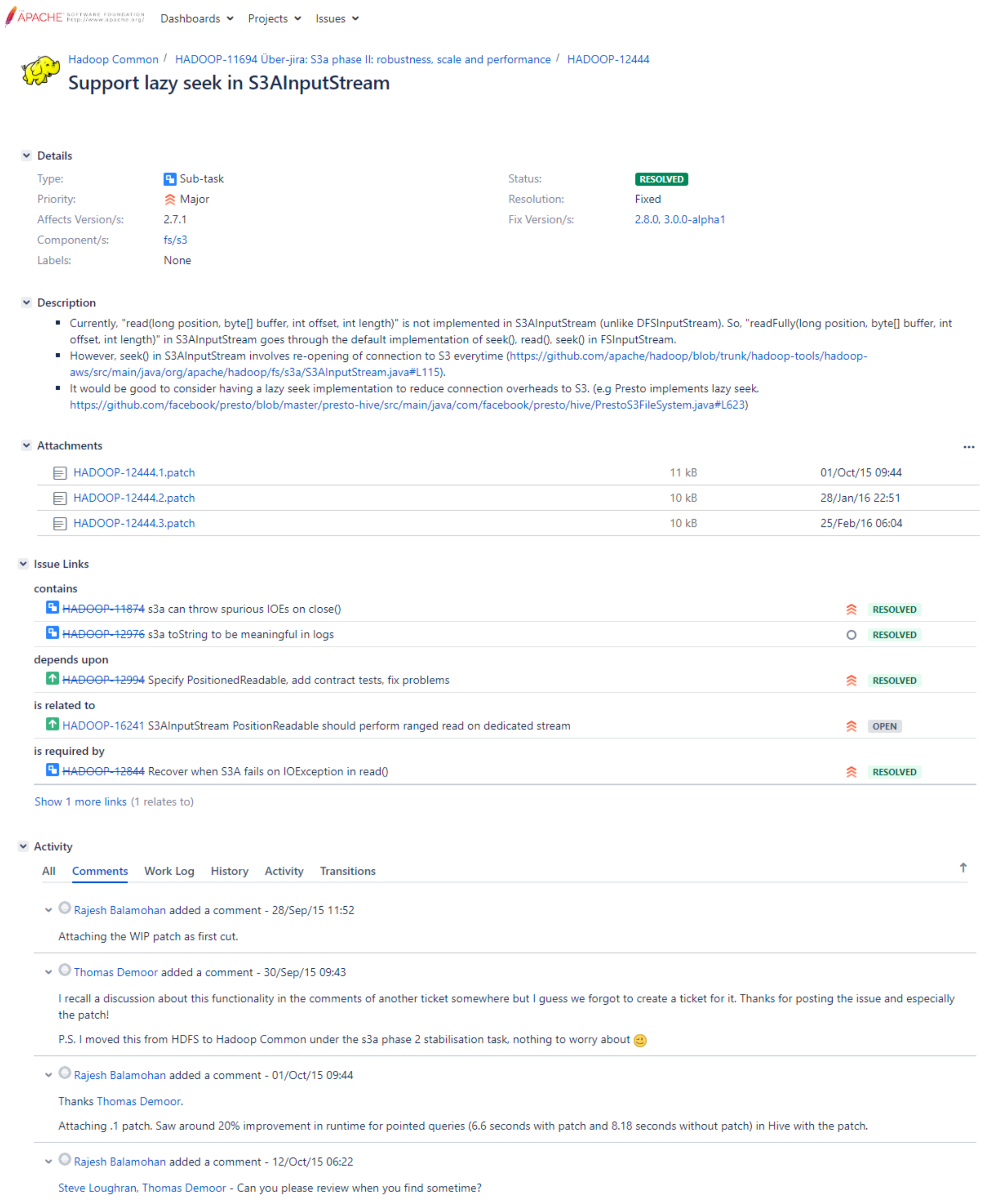
test endpoint: s3a ireland

Hadoop QA added a comment - 13/Sep/16 10:23

+1 overall

Vote	Subsystem	Runtime	Comment
0	reexec	0m 21s	Docker mode activated.
+1	@author	0m 0s	The patch does not contain any @author tags.
+1	test4tests	0m 0s	The patch appears to include 1 new or modified test files.
+1	mvninstall	6m 35s	branch-2 passed
+1	compile	0m 15s	branch-2 passed with JDK v1.8.0_101
+1	compile	0m 18s	branch-2 passed with JDK v1.7.0_111
+1	checkstyle	0m 15s	branch-2 passed
+1	mvnsite	0m 24s	branch-2 passed
+1	mvneclipse	0m 16s	branch-2 passed

Figure A.5: HADOOP-13599 (`https://issues.apache.org/jira/browse/HADOOP-13599` – As of June 2020).



Hadoop Common / HADOOP-11694 Über-jira: S3a phase II: robustness, scale and performance / HADOOP-12444

Support lazy seek in S3AInputStream

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.7.1	Fix Version/s:	2.8.0, 3.0.0-alpha1
Component/s:	fs/s3		
Labels:	None		

Description

- Currently, "read(long position, byte[] buffer, int offset, int length)" is not implemented in S3AInputStream (unlike DFSInputStream). So, "readFully(long position, byte[] buffer, int offset, int length)" in S3AInputStream goes through the default implementation of seek(), read(), seek() in FSInputStream.
- However, seek() in S3AInputStream involves re-opening of connection to S3 everytime (https://github.com/apache/hadoop/blob/trunk/hadoop-tools/hadoop-aws/src/main/java/org/apache/hadoop/fs/s3a/S3AInputStream.java#L115).
- It would be good to consider having a lazy seek implementation to reduce connection overheads to S3. (e.g Presto implements lazy seek. https://github.com/facebook/presto/blob/master/presto-hive/src/main/java/com/facebook/presto/hive/PrestoS3FileSystem.java#L623)

Attachments

HADOOP-12444.1.patch	11 kB	01/Oct/15 09:44
HADOOP-12444.2.patch	10 kB	28/Jan/16 22:51
HADOOP-12444.3.patch	10 kB	25/Feb/16 06:04

Issue Links

contains

- ~~HADOOP-11874~~ s3a can throw spurious IOEs on close() RESOLVED
- ~~HADOOP-12976~~ s3a toString to be meaningful in logs RESOLVED

depends upon

- ~~HADOOP-12994~~ Specify PositionedReadable, add contract tests, fix problems RESOLVED

is related to

- HADOOP-16241 S3AInputStream PositionReadable should perform ranged read on dedicated stream OPEN

is required by

- ~~HADOOP-12844~~ Recover when S3A fails on IOException in read() RESOLVED

Show 1 more links (1 relates to)

Activity

All Comments Work Log History Activity Transitions

Rajesh Balamohan added a comment - 28/Sep/15 11:52

Attaching the WIP patch as first cut.

Thomas Demoor added a comment - 30/Sep/15 09:43

I recall a discussion about this functionality in the comments of another ticket somewhere but I guess we forgot to create a ticket for it. Thanks for posting the issue and especially the patch!

P.S. I moved this from HDFS to Hadoop Common under the s3a phase 2 stabilisation task, nothing to worry about 😊

Rajesh Balamohan added a comment - 01/Oct/15 09:44

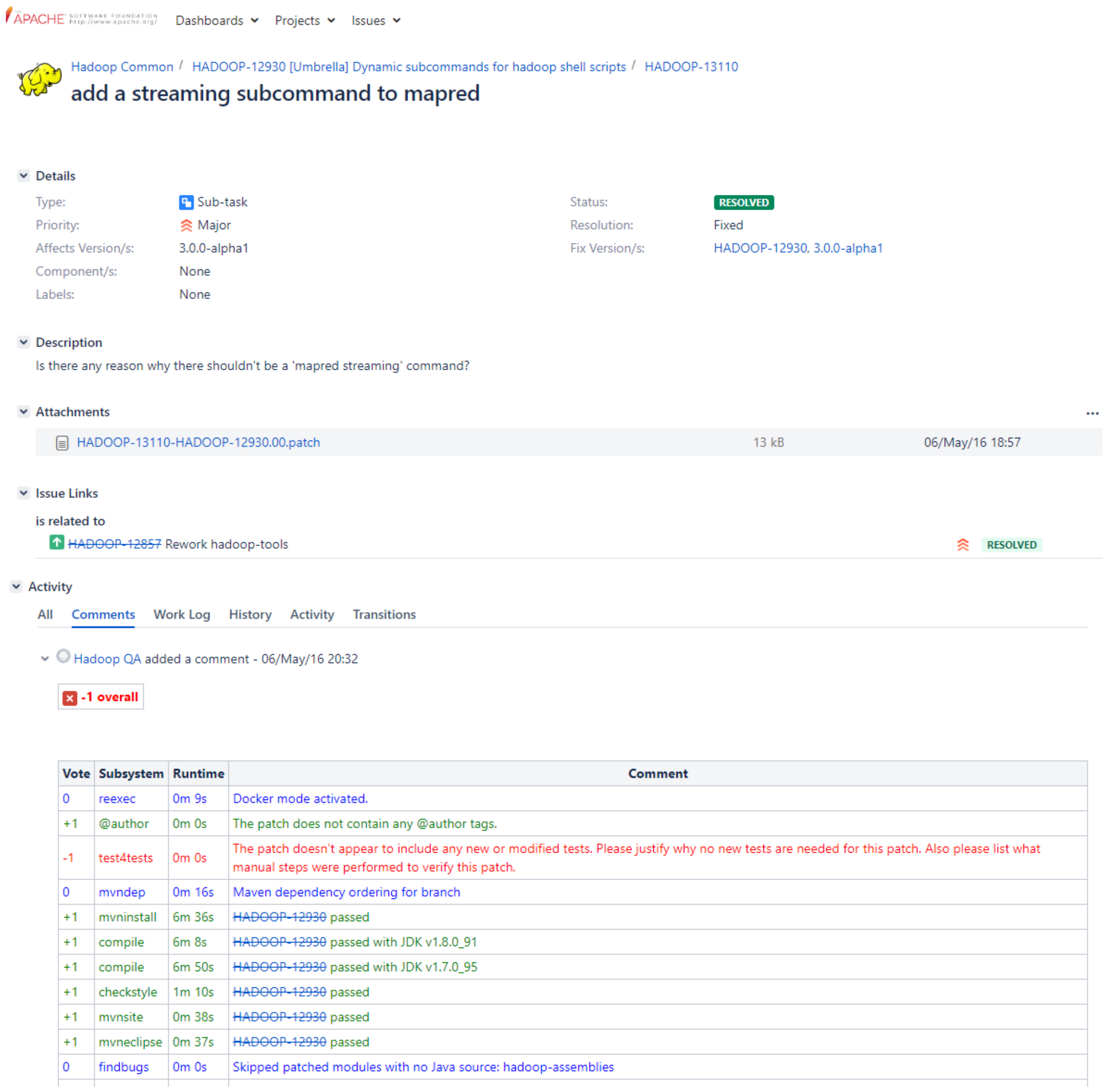
Thanks Thomas Demoor.

Attaching .1 patch. Saw around 20% improvement in runtime for pointed queries (6.6 seconds with patch and 8.18 seconds without patch) in Hive with the patch.

Rajesh Balamohan added a comment - 12/Oct/15 06:22

Steve Loughran, Thomas Demoor - Can you please review when you find sometime?

Figure A.6: HADOOP-12444 (`https://issues.apache.org/jira/browse/HADOOP-12444` – As of June 2020).



Dashboards Projects Issues

Hadoop Common / HADOOP-12930 [Umbrella] Dynamic subcommands for hadoop shell scripts / HADOOP-13110

add a streaming subcommand to mapred

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	3.0.0-alpha1	Fix Version/s:	HADOOP-12930, 3.0.0-alpha1
Component/s:	None		
Labels:	None		

Description

Is there any reason why there shouldn't be a 'mapred streaming' command?

Attachments

HADOOP-13110-HADOOP-12930.00.patch	13 kB	06/May/16 18:57

Issue Links

is related to

~~HADOOP-12857~~ Rework hadoop-tools RESOLVED

Activity

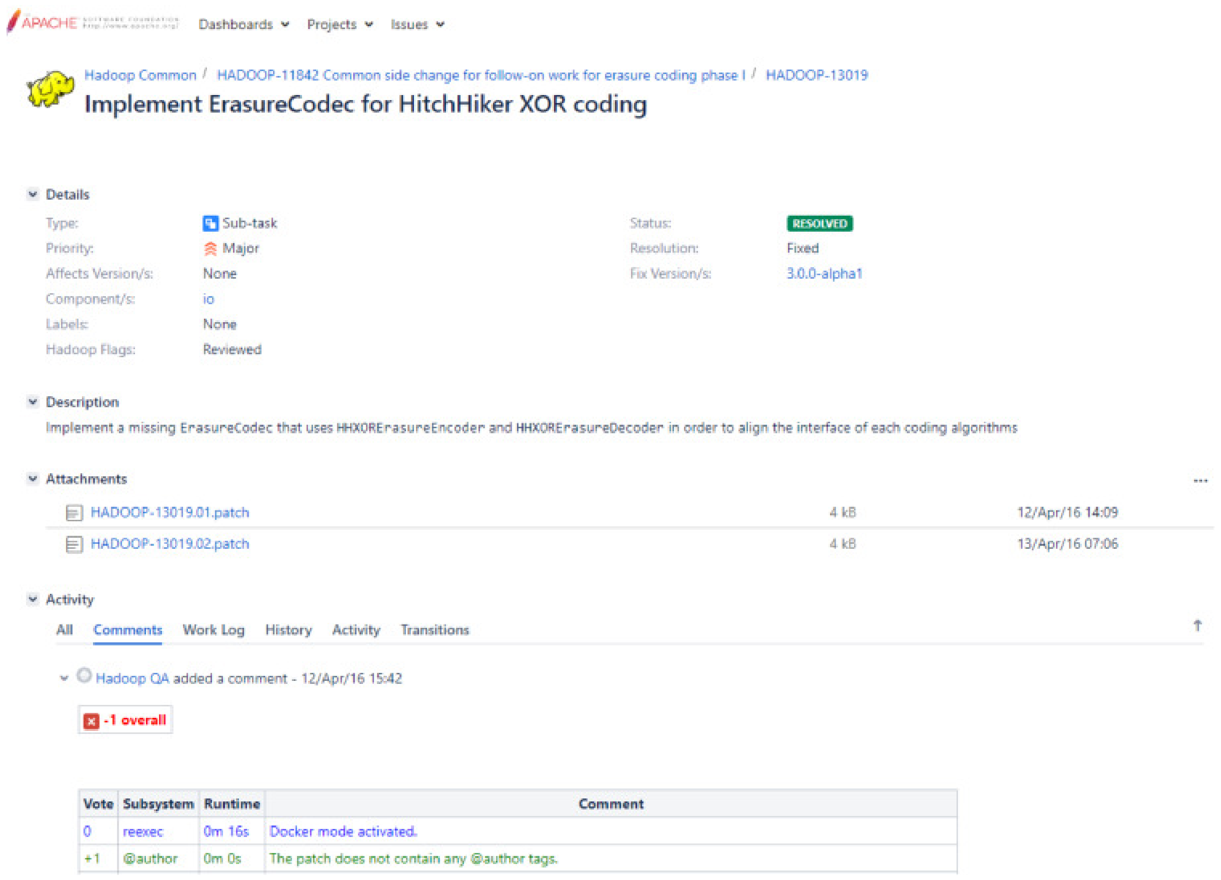
All Comments Work Log History Activity Transitions

Hadoop QA added a comment - 06/May/16 20:32

-1 overall

Vote	Subsystem	Runtime	Comment
0	reexec	0m 9s	Docker mode activated.
+1	@author	0m 0s	The patch does not contain any @author tags.
-1	test4tests	0m 0s	The patch doesn't appear to include any new or modified tests. Please justify why no new tests are needed for this patch. Also please list what manual steps were performed to verify this patch.
0	mvndep	0m 16s	Maven dependency ordering for branch
+1	mvninstall	6m 36s	~~HADOOP-12930~~ passed
+1	compile	6m 8s	~~HADOOP-12930~~ passed with JDK v1.8.0_91
+1	compile	6m 50s	~~HADOOP-12930~~ passed with JDK v1.7.0_95
+1	checkstyle	1m 10s	~~HADOOP-12930~~ passed
+1	mvnsite	0m 38s	~~HADOOP-12930~~ passed
+1	mvneclipse	0m 37s	~~HADOOP-12930~~ passed
0	findbugs	0m 0s	Skipped patched modules with no Java source: hadoop-assemblies

Figure A.7: HADOOP-13110 (`https://issues.apache.org/jira/browse/HADOOP-13110` – As of June 2020).



Hadoop Common / HADOOP-11842 Common side change for follow-on work for erasure coding phase I / HADOOP-13019

Implement ErasureCodec for HitchHiker XOR coding

Details

Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	3.0.0-alpha1
Component/s:	io		
Labels:	None		
Hadoop Flags:	Reviewed		

Description

Implement a missing ErasureCodec that uses HHXORErasureEncoder and HHXORErasureDecoder in order to align the interface of each coding algorithms

Attachments

HADOOP-13019.01.patch	4 kB	12/Apr/16 14:09
HADOOP-13019.02.patch	4 kB	13/Apr/16 07:06

Activity

Hadoop QA added a comment - 12/Apr/16 15:42

-1 overall

Vote	Subsystem	Runtime	Comment
0	reexec	0m 16s	Docker mode activated.
+1	@author	0m 0s	The patch does not contain any @author tags.

Figure A.8: HADOOP-13019 (`https://issues.apache.org/jira/browse/HADOOP-13019` – As of June 2020).