Revista Innovación y Software
Vol. 1, No. 1, Mes Marzo-Agosto, 2020
ISSN: 2708-0935
Pág. 11-25
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos originales
Temática: Desarrollo de aplicaciones informáticas
Recibido: 03/12/2019 | Aceptado: 20/12/2019 | Publicado: 30/03/2020

QuantityEr: Una solución simple y extensible para obtener la cantidad de resultados de consultas complejas a GitHub

QuantityEr: An extensible and simple solution to obtain the amount of results of complex queries to GitHub

Ernesto Soto Gómez[1][0000-0001-6521-2221]*

[1]Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, Km. 2 ½. Torrens, La Lisa, La Habana, Cuba. esoto@uci.cu

*Autor para correspondencia: esoto@uci.cu

Resumen
GitHub es una plataforma que proporciona alojamiento para el control de versiones de desarrollo de software utilizando Git. Cuenta con una interfaz de programación de aplicaciones para permitir que el software interactúe con la plataforma. La enorme cantidad de información alojada en GitHub puede ser útil para realizar estudios sobre la presencia actual de herramientas de desarrollo en la comunidad de desarrollo de software de código abierto. Sin embargo, el motor de búsqueda posee restricciones que hacen imposible emitir consultas complejas a la plataforma. En este informe, se describe una solución extensible y orientada a objetos, llamada QuantityEr, para obtener la cantidad de resultados de búsqueda de consultas complejas a GitHub utilizando el principio de inclusión- exclusión. Se presentan las definiciones matemáticas y los conceptos relacionados. Se discute el modelo matemático. Se presentan el diseño general de la aplicación y las herramientas de desarrollo utilizadas. Además, son mostrados resultados de ejemplos de ejecución. Se concluye que el problema tratado ha sido resuelto, aunque se puede trabajar para mejorar la solución.

Palabras claves: cantidad de resultados de búsqueda, GitHub, principio de inclusión-exclusión, programación orientada a objetos, Python

Abstract
GitHub is a platform that provides hosting for software development version control using Git. It features an application programming interface to allow the software to interact with the platform. The enormous quantity of information Hosted in GitHub may be useful to make studies about the current presence of development tools in the open-source software development community. However, the search engine has restrictions that make it impossible to issue complex queries to the platform. In this report, it is described as an object-oriented and extensible solution, named QuantityEr, to obtain the number of search results of complex queries to GitHub by using the inclusion-exclusion principle. The mathematical definitions, as well as related concepts, are presented. The mathematical model is discussed. The application of general design and used development tools are presented. Also, the results of the execution examples are showed. It is concluded that the treated problem has been solved although more work may be done to improve the solution.

Keywords: *search results amount, GitHub, inclusion-exclusion principle, object-oriented programming, Python*

Introduction

GitHub[1] is a platform that provides hosting for software development version control using Git[2]. It provides several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. It also features an application programming interface (API) to allow software to interact with the platform[3] [1]. Through this API a search engine can be accessed. The search engine allows users to find almost every single aspect across several projects, source codes and other areas and features of the platform[4] [2]. A web page that serves as an interface to the search API is also available[5].

As of August 2019, GitHub reports having over 40 million users and more than 100 million repositories[6]. This enormous quantity of information may be useful, among other things, to obtain the number of projects, source codes, issues, etc, that mention a set of technologies, tools, development libraries, etc, in order to make studies about the current presence of these tools in the open source software development community. Other kind of quantitative studies may be done as well [3]. Examples of those kinds of research are [4–7].

However, the search engine has some restrictions[4] that make impossible to issue complex queries to the platform. According to the GitHub Developer Guide[4], the restrictions are the following:

- The Search API does not support queries that:
  - are longer than 256 characters (not including operators or qualifiers).
  - have more than five AND, OR, or NOT operators.
- For authenticated requests can be made up to 30 requests per minute. For unauthenticated requests, the rate limit allows making up to 10 requests per minute.

Furthermore, if the search is over source code files, especial restrictions apply[7].

A system named GHTorrent have been already developed to ease the interaction with the large quantity of

[1] https://github.com/
[2] https://git-scm.com/
[3] https://developer.github.com/v3
[4] https://developer.github.com/v3/search/
[5] https://github.com/search
[6] https://github.com/about
[7] https://developer.github.com/v3/search/#search-code

information hosted in GitHub[8] [8]. This solution is mainly conceived to mirror the data hosted in GitHub in order to facilitate parallel access and studies on snapshots of the data, but does not provide an alternative to making complex queries to GitHub. In fact, this system has its own restrictions on the quantity of data that can be accessed at any time[9] [10]. Also, the system only provides snapshots for a reduced set of projects[11] [12]. Moreover, its design is centered only on the interaction with the repositories of GitHub. This means, for example, that search on source code is not allowed. Furthermore, the objective of the system is to interact with GitHub, which means that a future interaction with other platforms is not currently conceived.

A different kind of alternative is GH Archive[13] which records events form GitHub[14]. The recorded data can be accessed through BigQuery[15] which allows any kind of SQL-like queries. GH Archive, although a powerful and flexible solution, does not constitute an alternative to explore the data stored in GitHub but a tool to explore the data that represents the interaction with GitHub. This means that, for example, searching inside public source code cannot be done with GH Archive.

Moreover, both of these systems are server like development tools and not client applications ready to use for making queries.

In the context of this article, complex queries are those that have many logical connectives and sub-expressions –for example: `A OR (C AND (D OR E))`– especially those that exceed the allowed number of logical operators. By getting the results number of queries of this kind, analysis of the current presence of technologies might be done. Although many reporting tools has been developed none of them are capable of getting the number amount of complex queries directly to GitHub. Some of these tools are listed in https://www.gharchive.org/. Another example not listed in previous URL is https://www.programcreek.com/. In that case the reports are just for statically-selected libraries from statically-selected languages.

In this report, it is described a simple solution, named QuantityEr[16], to obtain the search results number of complex queries directly to GitHub. The proposed design was conceived with the aim of extension in mind, in such a way that it would be possible to incorporate the ability to interact with other similar platforms besides GitHub as well as other queries languages and algorithms for obtaining the amount of search results.

[8] http://ghtorrent.org/
[9] http://ghtorrent.org/raw.html
[10] http://ghtorrent.org/mysql.html
[11] http://ghtorrent.org/mongo.html
[12] http://ghtorrent.org/relational.html
[13] https://www.gharchive.org/
[14] https://developer.github.com/v3/activity/events/types/
[15] https://developers.google.com/apps-script/advanced/bigquery
[16] Source code accessible from https://github.com/EStog/QuantityEr/tree/v0.1

The current document is structured in the following manner. Section exposes some mathematical definitions and concepts necessary to understand the proposed solution. Section describes the proposed solution as well as some usage examples. Section makes the final remarks and conclude.

Mathematical background

In order to understand the proposed solution, some mathematical background is necessary. To archive a self-contained report, in this section is mentioned the principal mathematical concepts used in the design of the solution. The following definitions (or equivalent ones) as well of other complementary concepts and profs can be found in the cited references [9–17].

The following notations will be used in this report.

- $\wp(A)$ denotes the power set of a set A, that is the set of all subsets of A.
- $|A|$ denotes the cardinality of a set A, that is the number of elements in A.
- $\varnothing$ denotes the empty set.

Boolean algebras

The first essential concept important to the design of the proposed solution is that of Boolean algebra.

Definition 1. A Boolean algebra is a tuple $(S, +, \cdot, ', \bot, \top)$ where S is a set containing distinct elements $\bot$ and $\top$, $+$ and $\cdot$ are binary operators on S and $'$ is a unary operator on S. Every Boolean algebra satisfies the following laws for all $x, y, z \in S$.

Commutative laws:	$x + y = y + x$	$x \cdot y = y \cdot x$
Distributive laws:	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	$x + (y \cdot z) = (x + y) \cdot (x + z)$
Identity laws:	$x + \bot = x$	$x \cdot \top = x$
Complement laws:	$x + x' = \top$	$x \cdot x' = \bot$

Associative and idempotent laws, as well as other laws can be also considered since they follow from the definition laws. Furthermore, other useful operators can be derived from the previous ones [12, 14, 16].

Fact 1. *In a Boolean algebra* $(S, +, \cdot, ', \bot, \top)$ *the following laws are satisfied for all* $x, y, z \in S$*:*

Associative laws:	$x + (y + z) = (x + y) + z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Idempotent laws:	$x + x = x$	$x \cdot x = x$

Boolean algebras are used to model operations over the elements of a set that relates two elements with the maximum ($+$ operation) or the minimum ($\cdot$ operation) of both elements in a partial order where the minimum and the maximum are $\bot$ and $\top$, respectively. In other words, a partial order $\leq$ can be defined over S where

$$\forall a, b \in S \left(a \leq b \iff a + b = b\right)$$

or equivalently

$$\forall a, b \in S \left(a \leq b \iff a \cdot b = a\right)$$

and

$$\forall a \in S \left(\bot \leq a \leq \top\right)$$

[14].

Also, intuitively speaking, all the elements have an associated complement counterpart that together from the maximum but apart form the minimum as stated in the complement laws.

Fact 2. *The tuple* $\left(\{0, 1\}, \vee, \wedge, \neg, 0, 1\right)$ *is a Boolean algebra with the operations of disjunction* ($\vee$)*, conjunction* ($\wedge$) *and negation* ($\neg$) *defined as follow.*

$\vee$	*0*	*1*
0	*0*	*1*
1	*1*	*1*

$\wedge$	*0*	*1*
0	*0*	*0*
1	*0*	*1*

x	$\neg x$
0	*1*
1	*0*

This is the most elemental Boolean algebra and is the one found in classical binary logic that has applications in several areas of computer sciences [10, 12–14].

Fact 3. *The tuple* $\left(\wp(U), \cup, \cap, {}^{\mathrm{C}}, \varnothing, U\right)$ *is a Boolean algebra with the operation of union* ($\cup$)*, intersection* ($\cap$) *and complement* (${}^{\mathrm{C}}$) *defined as follows for all* $X, Y \in \wp(U)$.

$$X \cup Y = \left\{x \mid x \in X \vee x \in Y\right\} \qquad X \cap Y = \left\{x \mid x \in X \wedge x \in Y\right\} \qquad X^{\mathrm{C}} = \left\{x \mid x \notin X\right\}$$

This specific Boolean algebra is of great interest in science since mathematics in general are founded in set theory [11–14].

In this specific work, the last two described Boolean algebras are crucial because the current problem is to find the number of objects that makes true a logical sentence. In this context, the logical sentence is the query to be issue to the platform. The proposed solution takes advantage of the equivalences between classical logic and set theory in the context of Boolean algebras to solve this problem.

Boolean functions

In some contexts, the combination of operations in the set $\{0,1\}$ are called Boolean functions. The following definitions relate to this subject.

Definition 2 (Boolean function)**.** A Boolean function of degree n is a function $f\colon \{0,1\}^n \to \{0,1\}$ where f is an atom (a single variable or value) or a composition of the operations $\wedge$, $\vee$ and $\neg$ of the Boolean algebra $\left(\{0,1\}, \vee, \wedge, \neg, 0, 1\right)$. This composition is called a Boolean expression, and the variables of the Boolean expression are called Boolean variables.

This concept has wide application in logic gates circuits design. In this topic one of the main problems is the simplification of Boolean expressions [9, 12, 14, 16].

In the case of this work, these are of great importance because, as we will see, each query has an associated Boolean expression. The objective is to simplify it in order to obtain an expression that involves less computation.

The simplification of a Boolean expressions may be done symbolically by applying the laws of a Boolean algebra (definition 1) but also by applying specific methods that simplify an equivalent form of the expression.

Definition 3. Two Boolean expressions $A(x_1, x_2 \ldots, x_n)$ and $B(x_1, x_2, \ldots, x_n)$ are equivalent if $\forall x_1, x_2, \ldots, x_n \in \{0,1\}\left(A(x_1, x_2, \ldots, x_n) = B(x_1, x_2, \ldots, x_n)\right)$.

Definition 4. A normal form of a Boolean expression $f(x_1, x_2, \ldots, x_n)$ is an equivalent Boolean expression in the form $g(x_1, x_2, \ldots, x_n) = t_1 * t_2 *, \ldots, *t_m$ where each $t_{1\leq i\leq m}$ is in the form $y_1 \star y_2 \star, \ldots, \star y_{k\leq n}$ and each $y_{1\leq j\leq k}$ is in the form x_k or $\neg x_k$ where $1 \leq k \leq n$.

When $*$ is $\wedge$ and $\star$ is $\vee$ the normal form is called conjunctive (CNF). Similarly, when $*$ is $\vee$ and $\star$ is $\wedge$ the normal form is called disjunctive (DNF). Additionally, when the normal form is conjunctive each $t_{1\leq i\leq m}$ is called a maxterm. Similarly, when the normal form is disjunctive each $t_{1\leq i\leq m}$ is called a minterm.

The Quine-McCluskey algorithm is one of such methods that uses the normal form of a Boolean expression, specifically DNF, to obtain an equivalent minimal expression. The algorithm, in essence, test combinations of the minterms in order to find those that are essential to represent the value of the expression. It is known that it does not performance well when the size of the input, in this case the expression to simplify, is big. In fact, the problem of simplification of Boolean expressions is considered NP-hard [12, 14, 16].

However, the simplification of a Boolean expression is steel of great importance to this work, because small queries are preferable to big ones.

Definition 5. Let $X_1, X_2, \ldots, X_n$ be given sets. A predicate is a function $P\colon X_1 \times X_2 \times, \ldots, X_n \to \{0, 1\}$ [10, 13].

It obvious that a predicate has an associated Boolean expression if each atom is replaced by a Boolean variable.

Definition 6. The expression $S = \{x \mid P(x)\}$ is equivalent to $x \in S \iff P(x)$ [11].

The following theorem will be useful in the modeling of the solution.

Theorem 1. *The following relations are satisfied for any* $A = \{x \mid P(x)\}$ *and* $B = \{x \mid Q(x)\}$*:*

(a) $A \cup B = \{x \mid P(x) \vee Q(x)\}$

(b) $A \cap B = \{x \mid P(x) \wedge Q(x)\}$

(c) $A^{\mathrm{C}} = \{x \mid \neg P(x)\}$

Demonstration 1. *Proof follows directly from fact 3 and definition 6.*

This relations may be easily understood, since if A contains all the elements x such that $P(x) = 1$ and B is all the elements x such that $Q(x) = 1$ then it follows –from the definition 6 and the definition of union in the fact 3– that $A \cup B$ will have the elements x such that $P(x) \vee Q(x) = 1$. The same analysis can be done for the intersection and complement cases.

Inclusion-exclusion principle

First let consider the cardinality of the power set. This will be useful later in the description of the proposed solution.

Fact 4. *The cardinality of the power set of U is*

$$\left|\wp(U)\right| = 2^{|U|}$$

[13].

The inclusion-exclusion principle (IEP) is a mathematical formula that can be used to obtain the cardinality of the union of finite sets taking into account the cardinality of all possible intersections of the given sets.

Fact 5 (Inclusion-exclusion principle)**.** *The cardinality of the union of sets $S_{i\in\{1,2,\ldots,n\}}$ is*

$$\left|\bigcup_{i=1}^{n} S_i\right| = \sum_{\varnothing\neq J\in\{1,2,\ldots,n\}} (-1)^{|J|+1} \left|\bigcap_{j\in J} A_j\right|$$

The number of every possible intersection of n sets is the same that the number of subsets of a set of n elements without counting the empty set. This leads to the following fact taking into account fact 4.

Fact 6. *There are*

$$2^n - 1$$

terms in the inclusion-exclusion principle formula for n sets.

This means that an algorithm that calculates the cardinality of the union of n sets by directly using the IEP have an exponential complexity [15, 17].

In the proposed solution the IEP is used to decompose a given query in many smaller sub-queries that will be issued to the platform search API. In the next section, will be shown how to manage the problem of the exponential complexity when using this method.

Results and discussion

The problem to solve is: How to get the results number of complex queries to GitHub?

The proposed solution follows a divide and conquer approach as follows:

1. Simplify and decompose complex queries into smaller simple sub-queries.

2. Issue the sub-queries to the server and obtain the results amount of each one.
3. Sum up the results of the sub-queries into one that constitutes the results amount of the initial complex query.

In the next subsection a mathematical model and formalization of the solution is given.

Mathematical model

Mathematically speaking, the problem to solve is as follows.

Let O be the set of all the objects in the platform (projects, source codes, etc). Let $Q\colon O \to \{0,1\}$ be a predicate that represents the query to issue. Then, the set r_Q of all objects that match the query Q is

$$r_Q = \{o \mid Q(o)\}$$

The problem to solve is finding $|r_Q|$ when the associated Boolean expression given by Q has many compositions and logical connectives.

The first step of the proposed solution is to simplify the Boolean expression associated to the query. This may be done by symbolic transformations applying the laws that a Boolean algebra satisfies or also by using the Quine-McCluskey algorithm. It is known that this solution is not effective when the size of the input is too big. For this reason, the resultant expression (simplified or not) must be decomposed into various sub-expressions. For this purpose, the DNF expression is used. By applying theorem 1 it is known that if $Q(o) = Q_1(o) \vee Q_2(o) \vee \ldots \vee Q_n(o)$ is the DNF then

$$r_Q = \Big\{o \mid Q_1(o) \vee Q_2(o) \vee, \ldots, \vee Q_n(o)\Big\} = r_{Q_1} \cup r_{Q_2} \cup \ldots \cup r_{Q_n}$$

where

$$r_{Q_i} = \Big\{o \mid Q_i(o)\Big\}$$

for each $1 \leq i \leq n$.

Each $Q_i(o)$ is in the form $Q_{i_1}(o) \wedge Q_{i_2}(o) \wedge \ldots \wedge Q_{i_m}(o)$. This kind of query can be issued directly to GitHub because it does not have composition and only have conjunctive connectives. The conjunctives connectives (AND in the query language of GitHub) can be stripped of the sub-query since GitHub automatically interprets a tuple of atoms as a conjunction. In this case the is no use of conjunctive or disjunctive connectives in the query. Nevertheless, the case of the negation is a problem that, for now, cannot be avoided. So, in this case,

a query must be designed with care in order not to exceed the restriction that GitHub Search API imposes in the number of operators.

After the sub-queries have been sent, the next step is to find the results amount of the main query by applying IEP (fact 5). The problem with this approach is that the number of terms –according to fact 6– in IEP formula with n sets is $2^n - 1$, which is the number of sub-queries to be issued to the server.

However, each term in IEP is of the form of an intersection. Moreover, the terms in the expression associated to the DNF are also in the form of intersection. Then, by applying fact 1, that it is possible to reduce each term of the IEP formula so that some terms might be repeated afterwards. For this reason, it is proposed to use a cache for storing already issued queries as well as its respective results quantities in order to reduce the number of issued queries. However, work still need to be done to accelerate the computations of the terms in the IEP formula.

Solution design

QuantityEr is designed by using the object-oriented paradigm. Care on extension has been taken from the beginning by assigning a class to each sub-process in the solution. In figure 1 is outlined the class diagram of the most important classes. The classes are given as abstract base classes, so they must be extended for a particular problem. Currently, the extensions for solving the problem in the specific case of GitHub are implemented. Next, it is briefly described each class.

Main: Coordinate the interaction between the Input, Engine and Output classes objects. That is, the main algorithm is implemented inside this class.

Input: Currently, the queries can be presented to QuantityEr from two sources: the command line and files. Several queries can be presented to the application in one single execution. The responsibility of this class is to present these sources as a stream to the Parser. Since the logic of the input is encapsulated in one class, other kind of inputs may be added in the future like, for example, inputs from the network.

Parser: Translate the queries presented as input to a standard language that can be managed by the other entities. Since the logic of parsing is encapsulated in one class the syntax of the language used in the input queries do not need to be like the one expected by GitHub. This may ease the input allowing a cleaner syntax.

MiddleCode: Represents the intermediate language that the other classes understand. All the queries inside the application are in this format.

Engine: Coordinate the interaction between the Decomposer, Cache, Translator and QueryIssuer classes objects. That is, the algorithm that give the solution to the problem is implemented inside this class.

Decomposer: Decompose a complex query into several smaller simple queries. Currently, the extension using IEP is implemented.

Cache: Store the results amounts of already issued queries. Currently, an in-memory cache is available as well as a file-based one.

Translator: Translate a given simple sub-query to an issuable one. Currently, only GitHub is supported but more platforms may be added in the future.

QueryIssuer: Emit a simple sub-query to the platform and obtain the results amount or inform of an error if it was the case.

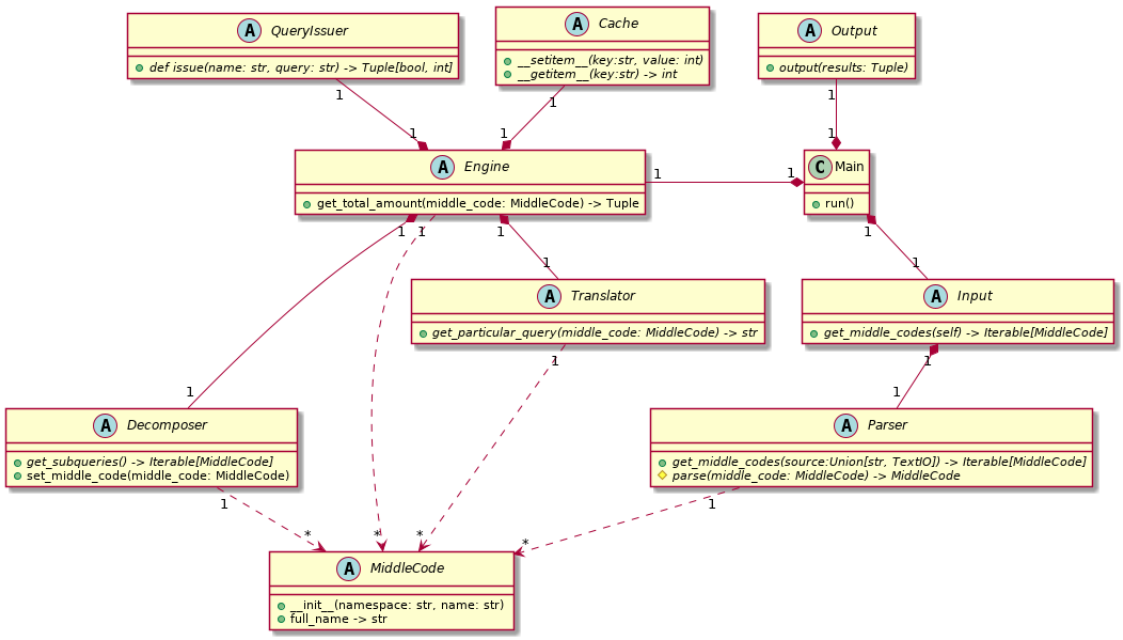

Figure 1. Class diagram of main classes of QuantityEr

Execution example results

In this section we consider a usage example result in order to study the behavior of the application with complex queries.

In this case, the queries ask for the amount of source codes that use the classical synchronization mechanisms defined in the asyncio, multiprocessing and threading Python libraries.

The results are summarized in table 1 and figure 2.

The command lines options to the program, the actual output, the presented queries as well as other execution example can be found in attached document `examples.html`[17].

Table 1. Execution example results summary. # means quantity. % means percent.

			Sub-queries				
				Cached		Issued	
No.	**Queries libraries**	**Amount**	**Total**	#	%	#	%
01	asyncio	69 053	15	0	0	15	100
02	multiprocessing	159 515	31	0	0	31	100
03	threading	1 451 344	31	0	0	31	100
04	asyncio ∩ multiprocessing	3 095	16 383	16 353	99.82	30	0.18
05	asyncio ∩ threading	29 658	16 383	16 353	99.82	30	0.18
06	multiprocessing ∩ threading	124 327	31	0	0	31	100
07	asyncio ∩ multiprocessing ∩ threading	1 947	16 383	16 353	99.82	30	0.18
08	asyncio ∪ multiprocessing	228 130	511	435	85.13	76	14.87
09	asyncio ∪ threading	1 494 420	511	435	85.13	76	14.87
10	multiprocessing ∪ threading	1 489 850	1 023	930	90.91	93	9.09
11	asyncio ∪ multiprocessing ∪ threading	1 528 155	16 383	16 185	98.79	198	1.21

[17]Downloadable from https://github.com/EStog/QuantityEr/blob/v0.1/running/jupyterlab/examples.html

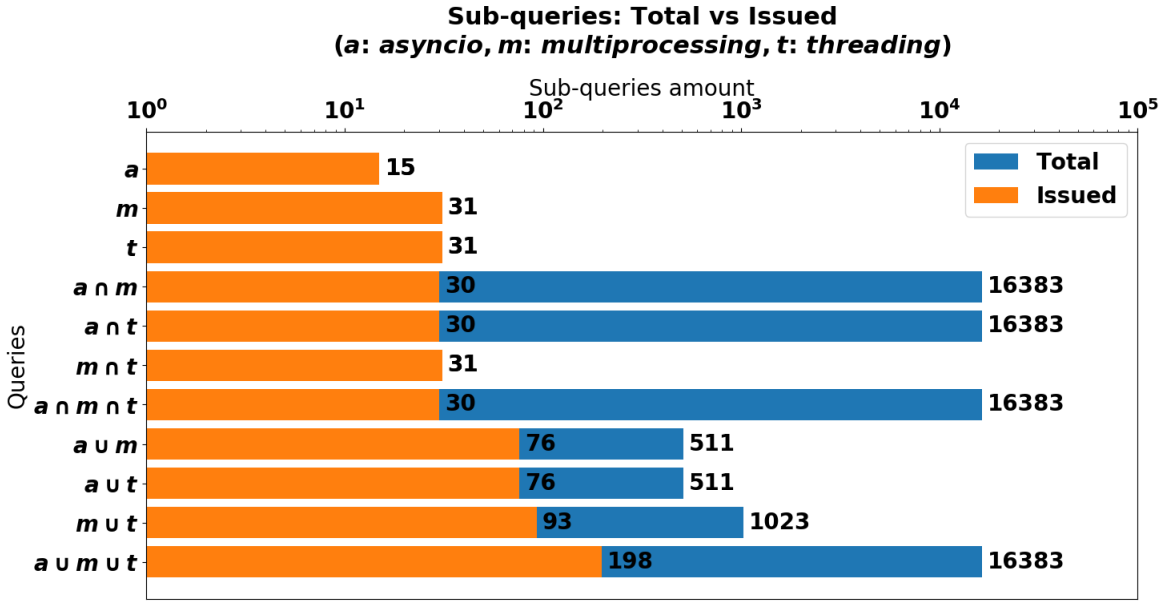

Figure 2. Sub-queries amount. Total vs Issued

In table 1 and figure 2 can be seen that the number of sub-queries depend on the ability of the Python's[18] [19] [18] Sympy[20] [21] [19] library to simplify the given expression. Also, in this case, the presence of the cache effects a great reduction on the number of issued queries, especially when the number of sub-queries is big.

Conclusions

In this report a tool, named QuantityEr, to obtain the results number of complex queries to GitHub search API has been described. The application uses the inclusion-exclusion principle and other mathematical abstractions to decompose the query in several simple sub-queries. The application uses a cache in order to reduce the number of sub-queries issued to the server. Even though it is considered that the use of the cache improves the solution and makes it viable, more work may to be done in order to accelerate the computations of the IEP formula terms. Moreover, the application may be extended to resolve other restrictions problems in GitHub and other platforms.

[18] https://www.python.org/
[19] https://docs.python.org/3.7/
[20] https://www.sympy.org/en/
[21] https://docs.sympy.org/latest/index.html

References

[1] C. Dawson and B. Straub, *Building Tools with GitHub: Customize Your Workflow*, 1st ed. O'Reilly Media, Inc., 2016.

[2] ——, "Python and the Search API," in *Building Tools with GitHub: Customize Your Workflow*, 1st ed. O'Reilly Media, Inc., 2016, pp. 53–80.

[3] S. Amann, S. Beyer, K. Kevic, and H. Gall, "Software Mining Studies: Goals, Approaches, Artifacts, and Replicability," in *Software Engineering: International Summer Schools, LASER 2013-2014, Elba, Italy, Revised Tutorial Lectures*, ser. Lecture Notes in Computer Science, B. Meyer and M. Nordio, Eds. Cham: Springer International Publishing, 2015, pp. 121–158. [Online]. Available: https://doi.org/10.1007/978-3-319-28406-4_5

[4] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, Mar. 2016, pp. 470–481.

[5] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "Investigating Social Media in GitHub's Pull-requests: A Case Study on Ruby on Rails," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, ser. CrowdSoft 2014. New York, NY, USA: ACM, 2014, pp. 37–41, event-place: Hong Kong, China. [Online]. Available: http://doi.acm.org/10.1145/2666539.2666572

[6] ——, "A Exploratory Study of @-Mention in GitHub's Pull-Requests," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1, Dec. 2014, pp. 343–350.

[7] A. A. Sawant and A. Bacchelli, "fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1348–1371, Jun. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9444-6

[8] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean GHTorrent: GitHub Data on Demand," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 384–387, event-place: Hyderabad, India.

[9] J. W. Grossman, "Functions," in *Handbook of Discrete and Combinatorial Mathematics*, 2nd ed., K. H. Rosen, Ed. Chapman & Hall/CRC, 2018, pp. 32–42.

[10] ——, "Propositional and Predicate Logic," in *Handbook of Discrete and Combinatorial Mathematics*, 2nd ed., K. H. Rosen, Ed. Chapman & Hall/CRC, 2018, pp. 12–22.

[11] ——, "Set Theory," in *Handbook of Discrete and Combinatorial Mathematics*, 2nd ed., K. H. Rosen, Ed. Chapman & Hall/CRC, 2018, pp. 22–32.

[12] R. Johnsonbaugh, "Boolean Algebras and Combinatorial Circuits," in *Discrete Mathematics*, 8th ed. New York, NY: Pearson, 2017, pp. 532–567.

[13] ——, "Sets and logic," in *Discrete Mathematics*, 8th ed. New York, NY: Pearson, 2017, pp. 1–61.

[14] J. G. Michaels, "Boolean Algebras," in *Handbook of Discrete and Combinatorial Mathematics*, 2nd ed., K. H. Rosen, Ed. Chapman & Hall/CRC, 2018, pp. 269–379.

[15] R. G. Rieper, "Inclusion/Exclusion," in *Handbook of Discrete and Combinatorial Mathematics*, 2nd ed., K. H. Rosen, Ed. Chapman & Hall/CRC, 2018, pp. 110–116.

[16] K. H. Rosen, "Boolean Algebra," in *Discrete Mathematics and Its Applications*, 8th ed. New York, NY: McGraw-Hill, 2019, pp. 847–883.

[17] ——, "Inclusion–Exclusion," in *Discrete Mathematics and Its Applications*, 8th ed. New York, NY: McGraw-Hill, 2019, pp. 579–585.

[18] S. Kapil, *Clean Python: Elegant Coding in Python.* Apress, 2019.

[19] J. M. Stewart, "SymPy: A Computer Algebra System," in *Python for Scientists*, 2nd ed. New York, NY: Cambridge University Press, 2017, pp. 128–149.