RESEARCH ARTICLE

Betty's (Re)Search Engine

A client-based search engine for research software stored in repositories.

Vasiliy Seibert [1]
Andreas Rausch [1]
Stefan Wittek [1]

1. Institute for Software and Systems Engineering, TU Clausthal, Clausthal-Zellerfeld.

Date Received:
2023-02-15

Licenses:
This article is licensed under:

Keywords:
Inggrid, Data, research data management, research software

Data availability:
Data can be found here: https://github.com/VasiliySeibert/cascading_search_ecology_eval

Software availability:
Software can be found here: https://github.com/VasiliySeibert/BettysReSearchEngine

Abstract. Promoting research, without providing the source code that was used to conduct the research, means a greater effort for every researcher down the line. Existing solutions that aim to make research software FAIR [1], fail to provide a wholesome solution, for they do not sufficiently consider already existing research software stored on platforms like GitHub or organizational GitLabs. We therefore present Betty's research engine, a client-based implementation of a cascading search process, that first finds research software stored on platforms like GitHub and then links them to corresponding publications or entries in third party databases. We evaluated 400 random search results from the domain of ecology and found that 345 out of 400 repositories made a reference to a corresponding publication / entry in third party database and therefore clearly indicating the potential of the cascading search. Betty's research engine is live and openly available under this URL: http://nfdi4ing.rz-housing.tu-clausthal.de/

1 Introduction

Findability and accessibility of research related software is crucial for every researcher who aims to (i) fully understand and (ii) reproduce software related publications as well as (iii) benchmark the own software related research to other solutions. However, finding fitting research software is not a trivial task. It is not obligatory for most authors of software related publications to provide a link to corresponding software repositories where the research software would then be stored.

[2] annotated a statified sample of a collection of software mentions, extracted from the CORD 19 Dataset [3] of open access publications. Their goal was to understand the status of software citation practices. Out of 295.609 software mentions in the Softcite-CORD-19 Dataset 50 % are solely names of the software without further detailes, 35 % provide a version, 21 % mention their publisher, and 9 % have a URL given in the Text. The authors of [4] reviewed the state of code availability in ecology using a random sample of 346 nonmolucolar articles published between 2015 and 2019 under mandatory or encouraged code sharing policies. Their results show, that only 27 % of eligible articles were accompanied by code, although the percentage of ecological journals with mandatory or encouraged code sharing policies increased from 15 %

(2015) to 75% (2020).

There are existing approaches, that aim to make research software FAIR. They can be broadly categorized as:

- Community driven approaches (e.g. Papers with Code [5])
- Technology driven approaches (e.g. Zenodo [6], Citation File [7])
- Standards and policies (e.g. FAIR4RS [8])

Papers with Code [5] is an example of a community driven approach to make research software more findable and accessible. It offers references to papers from the field of mathematics, physics, machine learning, astronomy and computer science as well as their corresponding data- and software repositories. Generation of new content as well as quality checking is done by a user community.

Zenodo [6] and Citation File [7] are examples of technology driven approaches. Zenodo is a platform that allows users to upload research related digital artifacts (such as research software) and generate a DOI (Digital Object Identifier) for them (therefore making them citable). If the digital artifact is also stored on GitHub, Zenodo offers a Zenodo-Badge, a small image file that can be integrated into the README to make the Zenodo DOI more visible. The citation file format [7] is a file format that is human and machine readable. If a citation file is placed in a Github Repository it unlocks the ”cite this repository” function on GitHub, which further increases visibility and the probability of getting cited.

Metadata standards (e.g. FAIR4RS [8]) and consequently policies (because someone has to enforce these standards) are designed to uniform the way research is documented and archived, with the objective of making it FAIR. Findability according to [8] can be achieved if:

1. the software is assigned a globally unique and persistant identifier,
2. the software is described with rich metadata,
3. the metadata explicitly includes the identifier of the software it describes
4. the Metadata is searchable and indexable

All of the described approaches rely on researchers to explicitly add additional information to their publications and therefore none of the approaches can be regarded as a wholesome solution for the problems described in [4] and [2] because of the following reasons:

- Finding existing research software is more important then the chance of finding (a fraction) of future research software.
- The effort for researchers to adapt their existing software repositories to a new database, technology, metadata standard is considerable and therefore a further obstacle.

For the above reasons there is a need for searching **already existing research software repositories** (stored on GitHub [9] or GitLab [10]) automatically and then linking them to corresponding publications or databases.

Such a *cascading search* would (i) initiate the search process by retrieving a list of references and metadata that refer to software repositories. If a given software repository contains any identifiers that can be connected to a corresponding publication or the listing of that software repository in a third-party database, then (ii) the search engine would access and retrieve that information. The result (iii) would be a list of software repositories, enriched with additional metadata and therefore sortable by the user's preferences.

From a technical perspective this cascading search engine faces a variety of challenges in terms of privacy, networking, individualization, reliability, scaleability and the complexity that arise from the use of multiple APIs. A user must be able to:

- understand the search process and the logic behind it,
- perform the described cascading search fully automatically,
- perform the described cascading search in an acceptable amount of time,
- be assured that his/her search is private and not being tracked,
- access sources of information that are commercial or limited to members of an organisation,
- extend the cascading search with further databases.

In this paper we reiterate the problem of not findable, accessible research software and show that existing approaches are unable to provide a wholesome solution. We motivate the need for searching existing research software and linking them to publications / database entries afterwards (this is what we call cascading search). We formulate top level requirements and elaborate how we meet them through our architectural design and our choice of technology. We evaluate our results by analyzing 400 repositories and reflect these results in a discussion. We then conclude our work.

2 Cascading Search

[4] and [2] focused on finding research software by searching for a reference in the publications. We introduce the cascading search, a search process according to which, we (i) initiate the search by retrieving a list of references and metadata that refer to software repositories (from a platform like GitHub). If a given software repository contains any identifiers that can be connected to a corresponding publication or the listing of that software repository in a third-party database, then (ii) the search engine will access and retrieve that information. The result (iii) of such a cascading search is a list of software repositories, enriched with additional metadata and therefore sortable by the user's preferences.

We find (i) and retrieve references to software repositories from GitHub by using the GitHub REST API. For a given search query, GitHub will find repositories that can be associated with that search query (e. g. the occurrence of that term in the name of the repository). GitHub also allows the user of the REST API to modify the search query in a way that makes adjustments to GitHubs internal search process. For example, the search query "seismology doi in:readme" would return a list of references to repositories, that can be associated with the term "seismology"

and that contain the word ”doi” in their README files. After retrieving a list of references, we iterate through that list to collect more information on every single repository. Among the collected information is the README file as well as the name of every other file in the main directory. We then search (ii) for any identifiers to a corresponding publication or the entry of that repository in a third party database. Searching for unique identifiers is a quality determining factor for the cascading search, since this information is most reliable for performing further search on that repository. As the results from the evaluation will show, references turn out to be highly heterogenous. We developed rules for identifying a *target reference* (a reference that refers to a corresponding publication or entry in a third party database, the counterpart of a target reference would be a *normal reference* that refers to any literature that was used to conduct or support the research).

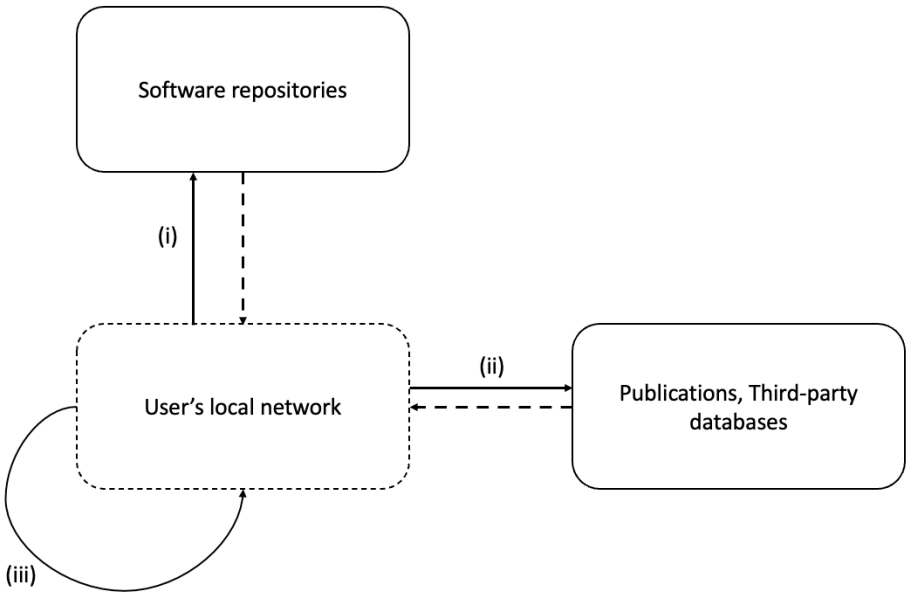

Figure 1: The cascading search (i) initiates the search process by retrieving a list of references and metadata that refer to software repositories. If a given software repository contains any identifiers that can be connected to a corresponding publication or the listing of that software repository in a third-party database, then (ii) the search engine will access and retrieve that information. The result (iii) is a list of software repositories, enriched with additional metadata and therefore sortable by the user’s preferences.

The following rules were created with the the computational cost and number of required API calls in mind.

- If the repository contains a citation file [7], then the reference in that file is considered a target reference
- If the repository contains a Zenodo image file [6] then the DOI is considered a target reference
- If the repository contains a Zenodo DOI in plain text and the data on Zenodo is similar to the name of the repository, we consider that reference as the target reference (only

checking the first Zenodo DOI)

After retrieving the target reference of a repository, we search that reference on further platforms. Current platforms include: Zenodo [6], Open Alex [11], DataCite [12] and Open citations [13].

The information that we retrieve from the cascading search (iii) is added to our data model. With the cascading search, we enriched the repositories with additional metadata that would not be available through a search on GitHub alone. The user can now, sort the repositories according to personal preferences and interests (like the number of citations a repository holds). The repository also holds information about what reference was considered the target reference. With that, we meet the requirement of making the cascading search understandable.

3 Client-based search engine

The cascading search faces a variety of challenges. A consequence of using multiple APIs is that multiple rate limitations have to be considered (one or more limitation rule per API). Since rate limitations are in many cases tracked through the IP Address that makes the request, a centralized backend is not possible. A centralized backend would also mean that the user could not call services and databases with restricted access (commercial or organizational).

To face the described challenges, Betty's research engine is written solely in languages that can run in the browser. We send the user everything needed to perform the cascading search from the local machine. For that reason, the user is required to provide own credentials, since it is the user that utilizes the different APIs. No communication is being recorded, the credentials are stored on the local machine, so the user enjoys the maximum amount of privacy that is possible.

The architectural design follows a MVC (Model View Controller) pattern (See Fig 2.). This means that the user interface (View) is strictly divided from the logic (Controller) and the database (Model) of the system. The user initiates the cascading search from the view. Inside the controller a *manager* holds knowledge of all available *agents*. An agent is a concrete implementation of how an API is utilized. This includes making the API call, receiving the returned information, processing it and passing that information through a defined interface to the model. By thinking in terms of agents, the complexity that arises from using multiple APIs is capsuled in a defined container. This way, the manager can orchestrate the agents by a generalized logic (and doesn't need to implement individual exceptions and error handlings). The retrieved information is stored in the database according to a defined data model. If new information is added to the model, the manager gets notified by an update mechanism. So every time a data instance is added or modified, the manager gets notified and can instantiate new agents that use that information.

Therefore the cascading search can be seen as a circular flow of data. The manager instantiates a GitHub agent, who performs a call to the GitHub REST API, retrieves that information and processes it before calling a model interface and adding the repositories to the database. The manager gets notified about new repositories being added, and instantiates Zenodo, DataCite, OpenCitations etc. agents that use the newly added information to perform searches on their own, before calling another model interface and adding the information.

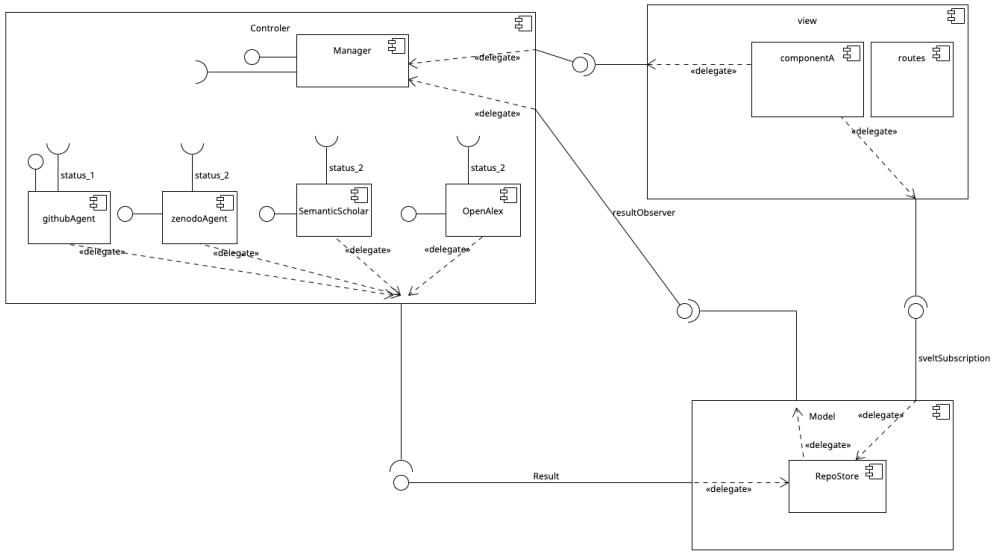

Figure 2: Betty's research engine is an implementation of the cascading search that faces the challenges of individual rate limitations and restricted access by a client based design. By handling the complexity that arises from the use of several APIs in agents, the cascading search can be performed in parallel processes while maintaining a sound, coherent structure.

Through the presented architectural design, the cascading search can now be conducted fully automatically and since there is no direct dependency between the agents, they can run in parallel.

4 Evaluation

[4] analyzed 346 publications, published in ecological journals. They found that between 2015 and 2019 under mandatory or encouraged code sharing policies only 27% of eligible articles were accompanied by code. To evaluate Betty's re search engine, we performed a cascading search for ecology related research software. Out of 2513 software repositories that were available on GitHub, we randomly choose 400 for detailed analysis (all data and code that was used to perform this analysis can be found at `https://github.com/VasiliySeibert/cascading_search_ecology_eval`). We found that 345 out of 400 (86,25%) repositories referred to a corresponding publication or the listing of the repository in a third party database.

With the introduced rules, we were able to correctly identify 89 (22,25%) target references (true positives). 256 (64%) repositories had a target reference, but were not identified (false negatives). 52 (13%) repositories did not refer to corresponding research and were correctly disregarded (true negatives). 3 (0.75%) repositories were falsely used for the cascading search (false positives).

Of further interest is how a repository refers to corresponding publications / entries in a third party database. We found that the majority of references (252, 63%) were mentions in plain text. 74 (18,5%) repositories used a Zenodo image file, 5 (1,25%) repositories had a citation file placed in their main directory. 10 (2,5%) repositories had a BibTEX text element in their README.

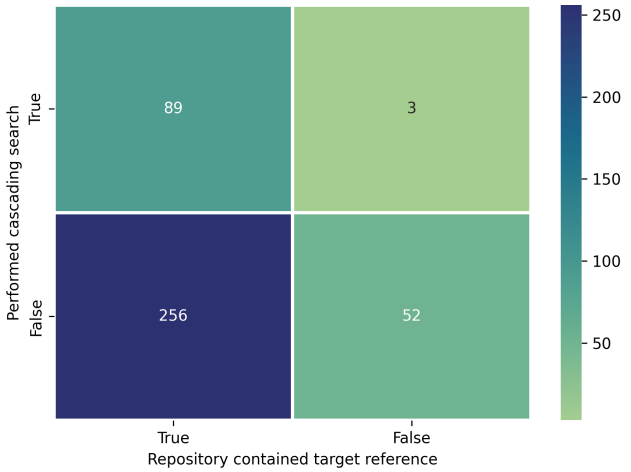

Figure 3: The introduced rules yield an accuracy of 35,25%. (all data and code that was used to perform this analysis can be found at `https://github.com/VasiliySeibert/cascading_search_ecology_eval`)

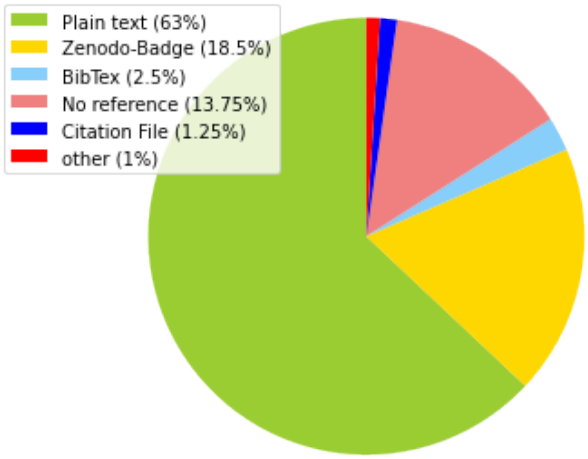

Figure 4: We found that the majority of references (252, 63%) were mentions in plain text. 74 (18,5%) repositories used a Zenodo image file, 5 (1,25%) repositories had a citation file placed in their main directory. 10 (2,5%) repositories had a BibTEX text element in their README.

5 Discussion

Does the cascading search come up for the shortcomings of current approaches? As the results from the evaluation chapter clearly indicate, the cascading search has the potential to compensate for the shortcomings of the approaches which were introduced in the introduction. Existing research software can be found on GitHub and furthermore connected to their respective publications. The cascading search utilize technological driven approaches, in fact these approaches are most reliable when it comes to identifying target references, for their use of structured text elements. Supported by these results, the cascading search is a valid and useful

alternative to community- and technological driven approaches as well as metadata standards and policies.

How effective is Betty's research engine? With an accuracy of merely 35,25%, Betty's re search engine has plenty of room for improvement. However, in relation to the 400 analyzed repositories, only 89 (22,25%) repositories made use of structured text elements Zenodo-Badges, Citation Files, BibTEX. A rule-based approach might not be well suited for this type of textual data. Since 63% of all references are written in plain, unstructured text, a data driven approach might be more promising.

In what use cases, can Betty's re search engine serve best? Betty's research engine can help researcher who aim to (i) fully understand and (ii) reproduce software related publications. Unlike Papers with Code, Betty's research engine does not support any categorisation of the found research. The search results do not have any information about ongoing benchmarks or the state of the art for that matter. Enabling users to (iii) benchmark the own software related research to other solutions would require further work.

6 Conclusion

In this paper we reiterated the problem of not findable, accessible research software and pointed out how current approaches do not provide a satisfying solution, for they are not sufficiently consider already existing research software repositories stored on platforms like GitHub or organizational GitLabs. We proposed the cascading search, a novel approach that has the potential to compensate for the shortcomings of current approaches by searching for software repositories first and then linking them to their corresponding publications. We presented Betty's research engine, an implementation of the cascading search that faces the challenges of individual rate limitations and restricted access by a client based design. We elaborated how the cascading search can be performed in parallel processes while maintaining a sound, coherent structure. We analyzed 400 random search results from the domain of ecology and found that (a) the cascading search is a valid, useful alternative to current approaches, (b) the rule-based approaches presented in this paper yield an accuracy of 35,25% and (c) the majority of references (63%) to corresponding publications / entries in third party databases are written in plain, unstructured text. Reflecting our results we found that a rule-based approach (like the one we use now) might not be well suited for the problem at hand and a data-driven approach might be more promising for identifying references to corresponding publications. Compared to community driven approaches like Papers with Code we reflected on the importance of categorisation and benchmarking of research software and concluded that further work on Betty's research engine is necessary.

7 Acknowledgements

This work would not have been possible without the financial support from the German Research Foundation (DFG) - project number 442146713.

We are grateful to all of those with whom we had the pleasure to work during this and other related projects.

8 Roles and contributions

Vasiliy Seibert: Conceptualization, Writing – original draft

Andreas Rausch: Conceptualization, Writing – original draft

Stefan Wittek: Conceptualization, Writing – original draft

References

[1] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[2] C. Du, J. Cohoon, P. Lopez, and J. Howison, "Understanding progress in software citation: A study of software citation in the cord-19 corpus," *PeerJ Computer Science*, vol. 8, e1022, 2022.

[3] L. L. Wang, K. Lo, Y. Chandrasekhar, *et al.*, "Cord-19: The covid-19 open research dataset," *ArXiv*, 2020.

[4] A. Culina, I. van den Berg, S. Evans, and A. Sánchez-Tójar, "Low availability of code in ecology: A call for urgent action," *PLoS Biology*, vol. 18, no. 7, e3000763, 2020.

[5] *Papers with code*. [Online]. Available: https://paperswithcode.com/.

[6] *Zenodo*. [Online]. Available: https://zenodo.org/.

[7] S. Druskat, N. C. Hong, R. Haines, and J. Baker, "Citation file format (cff)-specifications," *Zenodo. data*, 2018.

[8] M. Barker, N. P. Chue Hong, D. S. Katz, *et al.*, "Introducing the fair principles for research software," *Scientific Data*, vol. 9, no. 1, pp. 1–6, 2022.

[9] *Github*. [Online]. Available: https://github.com.

[10] *Gitlab*. [Online]. Available: https://about.gitlab.com.

[11] *Open alex*. [Online]. Available: https://openalex.org.

[12] *Data cite*. [Online]. Available: https://datacite.org.

[13] *Open citations*. [Online]. Available: https://opencitations.net.