Dutta, B. and Patel, J. (2022). AMV: A Representational Model and Metadata Vocabulary for Describing and Maintaining Algorithms. *Journal of Information Science*. https://doi.org/10.1177%2F01655515221116557

AMV: A Representational Model and Metadata Vocabulary for Describing and Maintaining Algorithms

Biswanath Dutta*
DRTC, Indian Statistical Institute, Bengaluru -560059, India
bisu@drtc.isibang.ac.in

Jyotima Patel
DRTC, Indian Statistical Institute, Bengaluru -560059, India
Department of Library and Information Science, Calcutta University, Kolkata, India
jyotima@drtc.isibang.ac.in

Abstract. Metadata vocabularies are used in various domains of study. It provides an in-depth description of the resources. In this work, we develop Algorithm Metadata Vocabulary (AMV), a vocabulary for capturing and storing the metadata about the algorithms (a procedure or a set of rules that is followed step-by-step to solve a problem, especially by a computer). The snag faced by the researchers in the current time is the failure of getting relevant results when searching for algorithms in any search engine. The designed vocabulary can be used by the algorithm repository developers, managers, and application developers. Besides, AMV is represented as a semantic model and produced OWL file, it can be directly used by anyone interested to create and publish algorithm metadata as a knowledge graph, or to provide metadata service through the SPARQL endpoint. To design the vocabulary, we propose a well-defined methodology, which considers factual issues faced by the algorithm users and the practitioners. The evaluation shows promising results.

Keywords: Algorithm Metadata Vocabulary; Algorithm Search and Retrieval; Algorithm Repository; Ontology; Semantic Application; Linked Data Vocabulary

1 Introduction

An algorithm is a step-by-step strategy to tackle any issue. The very purpose of it is to solve a problem by stating precisely the steps or rules to follow. It is used to solve a complex problem by breaking it down into steps [1]. An algorithm can be expressed in flowcharts, pseudo-codes, or some programming language. Algorithms are widely used in the current digital era. For instance, top hashtags on Twitter, trending videos on YouTube, etc. are automatically detected and displayed to the users by various complex algorithms [2,3]. Considering the importance of algorithms, they can be called plans for crucial computational processes. Algorithms are the core idea that links between a problem and the implementations (aka software programme) as shown in Figure 1.

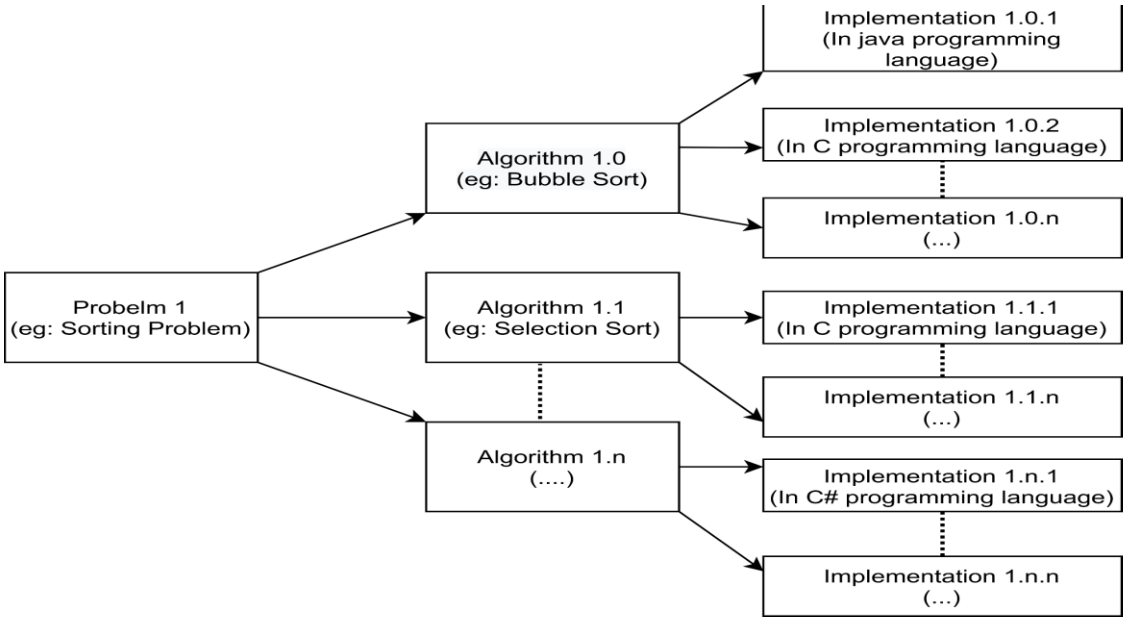


Figure 1. Algorithms as a link between a problem and the implementations

Figure 1 shows how the algorithm acts as a link between a problem and the various implementations. As clear in Figure 1 An algorithm must address a problem that is well-defined. A problem is defined here by specifying the entire set of inputs it must work with, as well as the output it produces after executing on one of these inputs [4]. For example, the sorting problem is defined as, Input: a sequence of n items $b_1,...,b_n$; Output: the rearrangement of the input sequence such that $b_1 < b_2 \ldots\ldots < b_{n-1} < b_n$. An input of sorting can be an array of names or a list of numbers. An algorithm is a method for transforming any of the potential input instances into the desired output. The problem of sorting can be solved using a variety of algorithms. Insertion sort, for example, is a sorting approach that starts with a single element and slowly inserts the remaining components to keep the list sorted. Examples of other sorting algorithms are merge sort, quick sort, bubble sort, and heap sort. These algorithms provide various ways to solve the sorting problem. The implementations are the actual realization of the algorithms, transformed as software programmes using some programming language (e.g., Java, C, Python), they are concrete and executable [4]. More information about algorithm as a domain of study is provided in section 2.1.

In this work, we propose a metadata vocabulary to facilitate the description of algorithms. Every discipline (e.g., Computer Science, Mathematics) has uncountable research algorithms, making it difficult for specialists, academicians, application engineers, and others to find, distinguish, select, and reuse them. In the current situation, we search the algorithms in search engines like Google, Bing, and Yahoo. Furthermore, it is exceptionally difficult to find and select the necessary algorithm. Because mostly the algorithms are published as part of the scientific literature. Hence, finding them on the web involves two steps: first, retrieve the relevant literature and then manually process them to find the relevant algorithm. Also, in the present scenario, we usually search them by theme. This restricts the search, making the search process a complex and time-consuming task. We cannot search the algorithms by their properties, for example by the data structure, problem, time complexity, etc. For example, if we want to find the shortest route between two cities, various literature dealing with path optimization problems need to be studied, and following that the best-suited algorithm is selected. If the metadata vocabulary is used, the search process becomes easier and can be done by executing some queries. We can also compare various path optimization algorithms and select the best suited for our needs. Our proposition is, to make the algorithm search more effective and personalized, they should be treated as an independent digital object, similar to the research article, research data, and in recent times the ontologies [5]. Given the significance of the algorithms as digital assets, they should be well described and managed. There should be a dedicated repository for maintaining algorithms like the repositories available for the above-mentioned objects, for example, ontology repositories like BioPortal[1], data repositories[2], etc. In respect to the algorithms, we found three well-appreciated initiatives, such as The Stony Brook Algorithm Repository[3], AlgoWiki,[4] and knowledge base wikidata[5]. The repositories provide information for the various existing algorithms including their sources. The major issue with them is the lack of a proper search facility. In the Stony Brook repository, we can browse the algorithm by problems and by language. In AlgoWiki, the popular algorithms are enlisted. Wikidata models the algorithms and interlinks them. More insights about these repositories are provided in section 2.2. There is less work towards algorithms while the main emphasis is laid towards software. In the current work, we focus on the description of algorithms.

One of the essential prerequisites to work with the item search is the metadata. We need a metadata vocabulary to enable the object description and administration. Further, we need dedicated and specific metadata vocabulary for this purpose. The metadata vocabularies (e.g., Dublin Core (DC), DCAT, LOM) for various digital assets are not enough to describe algorithms. The algorithms possess some characteristics which are unique and cannot be captured by any such vocabulary. They may be utilized to capture the most generic information, such as title, author, date, identifier, etc. In our study, we could not find any existing initiatives towards creating a vocabulary, particularly for the description of the algorithm. A dedicated vocabulary for algorithms will help in the identification, selection, and retrieval of the desired algorithm. A dedicated vocabulary will also solve the problem of different repositories and scientific domains denoting the information about algorithms in diverse ways, which currently makes it difficult to work across these diverse sources and valuable information is lost sometimes [6].

Using a well-defined vocabulary, we can create a repository that will empower seamless comparison between algorithms and will make understanding them easy. Algorithm Metadata Vocabulary (AMV) is the first of its kind. It is an effort toward making algorithm metadata FAIR data (Findable, Accessible, Interoperable, and Reusable) [7]. AMV can be used by the algorithm repository developers and managers.

Motivation

Research relies heavily on algorithms. A vast and growing number of scholars are inventing algorithms, as part of their research. Existing procedures for archiving, discovering, sharing, and citing algorithm contributions are inconsistent among disciplines and archives, and they rarely follow the best standards [8].

The main contributions of the current work are:

- Provides a brand-new yet harmonized metadata vocabulary AMV specifically designed for describing and maintaining information about algorithms;
- The vocabulary is developed as an ontology using semantic web technologies (RDF and OWL) to make data machines processable and also to convey to the community easily;
- Provides a well-defined lifecycle methodology with a set of guiding principles for creating and maintaining a metadata vocabulary.

Anyone interested in creating and publishing algorithm metadata as a knowledge graph [9] and/or providing metadata service through the SPARQL endpoint [10,11] can use the ontology directly.

[1] https://bioportal.bioontology.org/
[2] https://datadryad.org/
[3] https://algorist.com/algorist.html
[4] https://wiki.algo.is/
[5] https://www.wikidata.org/wiki/Wikidata:Main_Page

The rest of the paper is organized as follows: Section 2 briefly discusses the algorithm as a domain of study followed by the discussion on the current state of the algorithm repositories; section 3 provides the guiding principles and methodology to build AMV; section 4 deals with the AMV metadata model describing the various classes and properties and also the existing vocabularies that are (re)used in this study; section 5 discusses the evaluation of the model conducted from various aspects; section 6 discusses the relevant and related state-of-the-art works. Finally, section 7 concludes the paper and provides future research directions.

2 State of the Art Study

2.1 Algorithm

An algorithm is a finite series of well-defined, rigorous instructions that is often used to solve a computational problem. According to the encyclopedia of mathematics [12], "a computational algorithm has two parts: (1) an abstract (or a proper) computational algorithm applicable to mathematical objects which is independent of the particular computer used and which may be written down in conventional mathematical terms or in some algorithmic language; (2) a program, i.e. a collection of machine commands describing the computational algorithms, organizing the realization of the computational process in the given computer."

Algorithms are not new inception, ancient mathematicians and scientists used them. For example, the method that is still taught in schools for calculating the Greatest Common Divisor (GCD) is the Euclidean algorithm which was developed in 300 BC [13]. The year 1842 marks a milestone in the history of algorithms, as Ada Lovelace published the first algorithm for a computing engine [13]. Since then, there is continual advancement in the field and algorithms are being published in various disciplines like Mathematics, Physics, Computer science, and so on. With advancements in the field of algorithms, there are improvizations and extensions made to the existing algorithms. For example, the YOLO algorithm [14] for object detection has 4 improvised versions. Each improved version of the YOLO algorithm [15,16] has an improved accuracy along with optimized runtime [17]. Similarly, there exist extensions of algorithms, for example, the ant colony optimization algorithm [18] has multiple extensions like max-min ant system [19], ant colony system [20], continuous orthogonal ant colony system [21], etc. There are various types of algorithms being created in diverse domains and these algorithms are at the core of most technologies used in modern computers.

The algorithms can be classified in various ways some of them are, by design technique (e.g. divide and conquer, brute force, backtracking, etc), by implementation (e.g. recursive or iterative, exact or approximate, etc), by optimization technique (e.g. greedy, dynamic, heuristic method, etc), by field of study (e.g. graph theory, string theory, etc) and so on [22]. For example, A* path-finding algorithm follows both greedy algorithm and dynamic programming for optimization. It uses the greedy approach to optimize the best cases, and the dynamic programming approach to optimize the worst cases [23]. Apart from this, the most recursive algorithms can be expressed iteratively as well. As clear from the above examples some of the classifications available for algorithms are not mutually exclusive. The classification of the algorithms by design technique is the most popular classification available in various literature. One of the key aspects of an algorithm design technique is selecting an appropriate data structure (the way of storing and accessing the data). Selecting the correct data structure for algorithm design provides efficiency, reusability, and abstraction. It also plays an important role in enhancing performance when implementing an algorithm [24].

As illustrated in section 1 an algorithm acts as a link between a problem and the various implementations. There are various types of problems that can be solved by an algorithm like combinatorial problems, polynomial-time problems, set problems, string problems, numerical problems, etc. Sometimes for solving a single problem there can be multiple algorithms. For instance, sorting problem has approximately 15 different algorithms to sort a set of items like merge sort, bubble sort, insertion sort, etc. This often leads us to the challenge of finding and selecting the Mr. Right algorithm. As mentioned in [25] by Akhter et al., different sorting algorithms perform differently for disparate input cases. There are various parameters on which algorithms solving the same problem can be compared like usage of computational resources, correctness, etc., and accordingly, a choice can be made. Besides, there are cases when a complex problem is solved by using a combination of algorithms. For example, the encoder-decoder model in deep learning combines two algorithms to solve problems like machine translation and image captioning. Nal Kalchbrenner and P. Blunsom in [26] utilized CNN (Convolutional Neural Network) to encode the source language and then used RNN (Recurrent Neural Network) as the decoder to transform into the target language thus solving a complex problem of machine translation. Another example of a complex problem is [27] where Sen et al. provide a solution to a polynomial time problem of query processing. To solve this problem 3 algorithms were developed, one main algorithm followed by two sub-ordinate algorithms.

The focus of algorithms as a robust conceptual object lies in their correctness and efficiency. Correctness of an algorithm refers to the ability of the algorithm to successfully produce the correct output. It can be checked by running the algorithm for all the possible input cases i.e. the algorithm to solve problem P is correct if and only if for all the problem instances, it terminates and produces the correct output [28,29]. The efficiency of an algorithm can be analyzed by computing the number of computational resources used by an algorithm [30]. The computational resources comprise the time and space complexities of an algorithm [31]. The space complexity is the memory required to solve an instance of the computational problem. Space complexity includes both auxiliary space (temporary space used by the algorithm) and space used by input. The time complexity of an algorithm does not depend on factors like a programming language, processing power, etc. Time complexity represents the number of times a statement is executed [32]. The execution time of the algorithm depends only on the algorithm itself and its input. Some other important- attributes of an algorithm are, e.g., the form of expression (e.g., flowcharts, pseudo-codes), input, output, independence, and constraints.

2.2 Current State of the Algorithm Repositories

As mentioned above, there are three algorithm repositories available on the Web and they are AlgoWiki, Stony Brook Algorithm Repository, and wikidata which model the algorithm.

The AlgoWiki is an online encyclopedia of algorithms. It contains a list of algorithms on a wiki page arranged in alphabetical order [33]. Once we select our desired algorithm and click on it, the site redirects to a page where the algorithm is explained and the pseudo-code is given. The redirected site is either a Wikipedia page or any tech site. AlgoWiki consists of algorithms from the field of mathematics with some features and properties. It is a wiki dedicated to competitive programming. Figure 2 provides a screenshot of the AlgoWiki page. The Stony Brook Algorithm Repository focuses on market research to find which algorithmic problems are used mostly in applications primarily focusing on the field of combinatorial algorithms and algorithm engineering [34]. The study found that the main purpose of the visitors in the repository was to seek implementations of algorithms that solve the problem they are interested in.

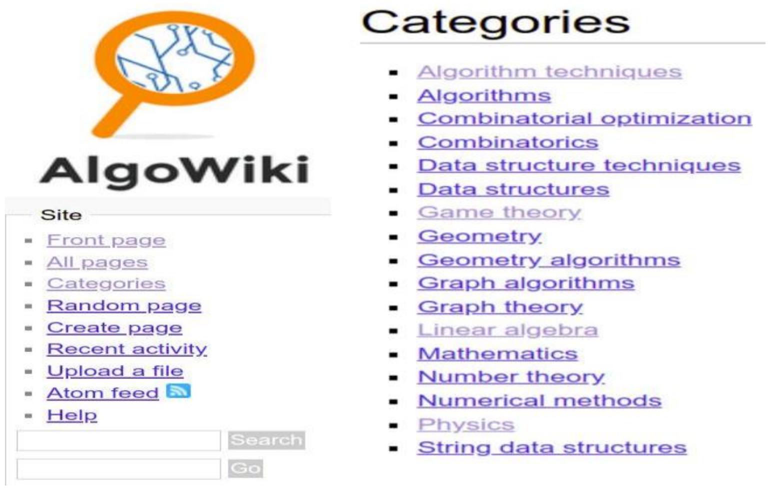
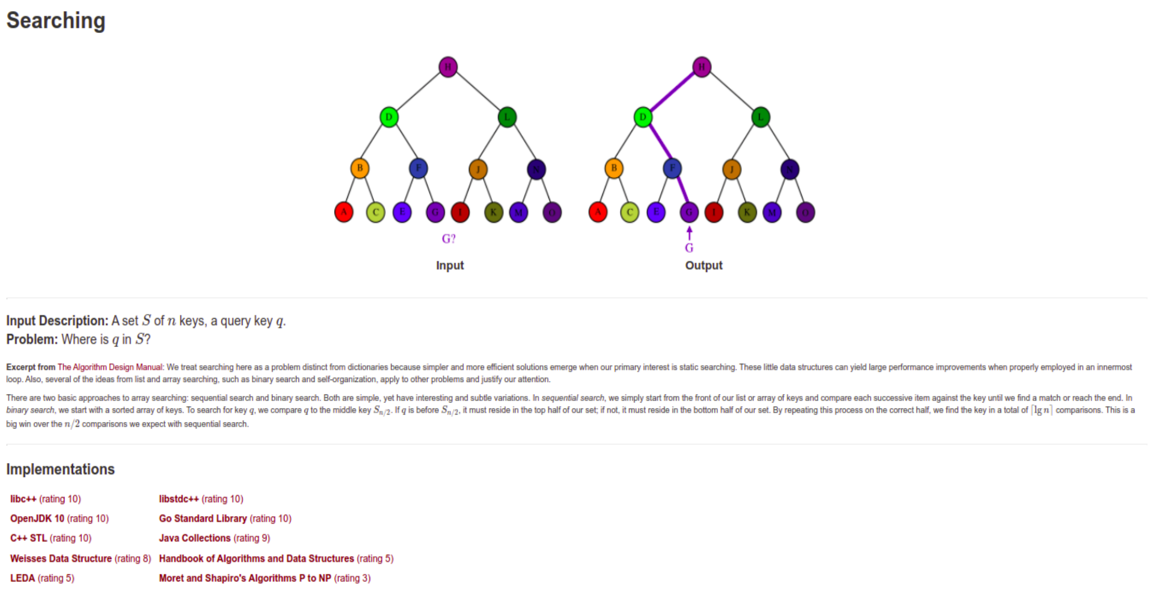


Figure 2. A depiction of AlgoWiki page

The Stony Brook Algorithm Repository consists of 75 fundamental algorithms. The algorithms are grouped into 3 categories: by language, by problem, and by algorithm links. If one clicks on an algorithm, the page provides (as in Figure 3) the description of the problem, the algorithm, the input description, and the various implementations of the same algorithm along with their ratings [35]. The Stony Brook Algorithm Repository has not been updated since 2008.



Searching

Input Description: A set S of n keys, a query key q.
Problem: Where is q in S?

Excerpt from The Algorithm Design Manual: We treat searching here as a problem distinct from dictionaries because simpler and more efficient solutions emerge when our primary interest is static searching. These little data structures can yield large performance improvements when properly employed in an innermost loop. Also, several of the ideas from list and array searching, such as binary search and self-organization, apply to other problems and justify our attention.

There are two basic approaches to array searching: sequential search and binary search. Both are simple, yet have interesting and subtle variations. In *sequential search*, we simply start from the front of our list or array of keys and compare each successive item against the key until we find a match or reach the end. In *binary search*, we start with a sorted array of keys. To search for key q, we compare q to the middle key $S_{n/2}$. If q is before $S_{n/2}$, it must reside in the top half of our set; if not, it must reside in the bottom half of our set. By repeating this process on the correct half, we find the key in a total of $\lceil \lg n \rceil$ comparisons. This is a big win over the $n/2$ comparisons we expect with sequential search.

Implementations

libc++ (rating 10)	**libstdc++** (rating 10)
OpenJDK 10 (rating 10)	**Go Standard Library** (rating 10)
C++ STL (rating 10)	**Java Collections** (rating 9)
Weisses Data Structure (rating 8)	**Handbook of Algorithms and Data Structures** (rating 5)
LEDA (rating 5)	**Moret and Shapiro's Algorithms P to NP** (rating 3)

Figure 3. Description of an algorithm from Stony Brook algorithm repository

Wikidata is a knowledge base that serves as central storage for structured data of its Wikimedia sisters [36]. Initiatives are taken by wikidata to model algorithms where they use generic properties to describe algorithms and then interlink the various other algorithms as sub-classes or instances. Wikidata consists of items, each having a label and description. Items are uniquely identified by "Q" followed by a number. For instance, Q8366 is a unique identifier for *algorithms*. The properties in wikidata are identified by "P" followed by a number, for example, P31 is for *instance of*, P279 is for *subclass of*.

The major shortcomings of the wikidata initiative for modelling algorithms are that the specific characteristics of an algorithm are not considered, like what is the application of an algorithm, what kind of input is required, what are edge cases of an algorithm, and so forth. Wikidata gives us a more generalized description and has broadly modelled the algorithms.

In summary, the currently existing initiatives have various drawbacks, for example, searching is not an easy task as they are HTML based and only facilitate browsing. The minimal annotations act as a bar in keyword searching. Hence, a metadata vocabulary dedicated to algorithms is required for exhaustive description, and proper representation of algorithms, which will facilitate better management and maintenance of the repository. The metadata vocabulary will also enable the individual repository developers to maintain the algorithm-specific knowledge more efficiently. The presence of such vocabulary will facilitate better use and reuse of algorithms, thereby making the various algorithm repositories interoperable as there will be uniformity in terms of the elements used to explicitly describe the algorithms.

Given the above observation, we have studied algorithms and extracted the properties exhibited by them for better classification and to provide a better description of them. We have also studied the various existing metadata vocabularies to build a foundation for our work as detailed in the following section.

3 Methodology

Here, we briefly discuss the design approach of AMV along with the guiding principles. There are various approaches available in the literature for the development of vocabularies (e.g., taxonomy, metadata, thesaurus, ontology). They are, for example, METHONTOLOGY [37], On-To-Knowledge [38], DILIGENT [39], NeOn [40], and YAMO [41], The AMV development is inspired by the METHONTOLOGY, NeOn, and YAMO. AMV started with specifying its purpose and finding the related resources including the search for related existing vocabularies. The AMV development process consists of top-down and bottom-up approaches and each further consists of multiple steps. Before elaborating on the AMV development steps, we first provide a set of guiding principles as follows originally provided in [42,43]. These guiding principles, especially the first three, are applicable when we aim to reuse the existing vocabularies to create a new one.

The guiding principles are:

1. Precedence to the metadata vocabularies particular towards algorithm and its related objects rather than more generic ones.
2. There must not be any conflict (e.g., disjoint classes) if someone describes an algorithm with all the listed properties selected from, for instance, amv: Algorithm and foaf: Document.
3. The vocabulary to be reused should be a recommendation from standard organizations (e.g., World Wide Web Consortium), or community-driven (e.g., Dublin Core, RDA).
4. The vocabulary should have sufficient elements for providing the metadata description. It should contain the elements which will allow the annotators in highlighting the usage and quality of the described resources. The same can be confirmed by engaging the users as shown in section 5.1.
5. The vocabulary should be expressed using the major knowledge representation technologies (e.g., RDFS, OWL, SKOS) currently in use in semantic web applications. The use of these vocabularies in data annotation will facilitate data interoperability across the systems.
6. The metadata elements used should be distinct, with understandable descriptions.

3.1 AMV Development Approach

The AMV vocabulary developed following a two-ways approach top-down (deductive approach) and bottom-up approach (inductive approach) as described below.

Top-down Approach (deductive approach). This approach demands proceeding from a conceptual level to a concrete level, i.e., the broad view of AMV is taken into consideration [44]. We work on the abstract level by mentioning the top-level facets for which we draw up the several features of the resource to be described. In this work, the core resource is an algorithm. After describing the top-level facets, we further analyze them and narrow them down to the different classes and subclasses levels as elaborated in section 4.

Bottom-up Approach (inductive approach). This approach demands proceeding from a concrete level to an abstract level. In this approach, we studied and identified the properties of an algorithm. We tried to analyze how the properties extracted are related to the classes mentioned in the top-down approach. To be systematic in our work, we proceeded step by step as depicted in Figure 4. The steps mentioned below give an idea of how we created the vocabulary from scratch.

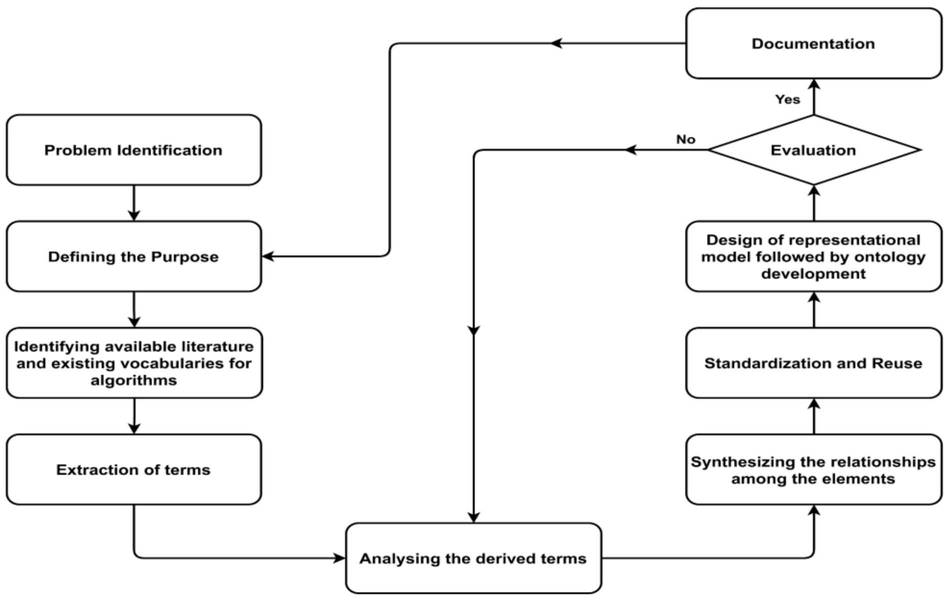

Documentation
Yes
No
Evaluation
Problem Identification
Defining the Purpose
Design of representational model followed by ontology development
Identifying available literature and existing vocabularies for algorithms
Standardization and Reuse
Extraction of terms
Synthesizing the relationships among the elements
Analysing the derived terms

Figure 4. Steps to create AMV

Step 0: Problem identification

In this step, we identify the constraints faced while searching for algorithms. This is done by analyzing the existing algorithm repositories. This step describes how the current repositories lack the proper search and retrieval facility as they are HTML-based and only facilitate browsing. The available repositories also lack suitable filtering criteria for searching. In general, the main constraint faced while searching for a specific algorithm is that many times the algorithms reside in the literature only and are not available on the Web or any repository.

Step 1: Defining the purpose

The step describes the purpose of the designed vocabulary. The main purpose of AMV is to facilitate the description of computational algorithms and their related entities, so that they become findable, and can be identified as independent digital objects. The purpose of a vocabulary can be best illustrated through a set of competency questions (CQ). In the case of AMV, the example CQs are as follows. In the queries, "X" and "Y" represent variables.

CQ1: Give me all the algorithms that compute the solution to a problem X along with their types, input, and data of creation.

CQ2: Which sorting algorithms are derived from the already existing sorting algorithms?

CQ3: Retrieve the implementation details, such as programming language, operating system, and CPU time limit of an algorithm X.

CQ4: Which data structure is usually used when solving a problem type X?

CQ5: Retrieve the improvised version(s) of algorithm X

CQ6: Which mathematical property is used in an algorithm X and retrieve its defining formula.

CQ7: For a given algorithm X, list all the software that is protected by license Y.

CQ8: Which is the most commonly used method to solve problem X?

CQ9: Retrieve all the available algorithms by their field of study.

CQ10: Retrieve all the algorithms that belong to an algorithm type X.

CQ11: Retrieve all the algorithms that use X data structure.

CQ12: List the literature in which algorithm X appeared.

CQ13: Retrieve the literature in which an algorithm X appeared along with its date of publication.

CQ14. Retrieve all the derived algorithms along with their primary source.

CQ15. Retrieve all the problems and sub-problems along with their corresponding algorithms.

Step 2: Reviewing the available literature and existing vocabularies

In this step, we studied the available literature dealing with the algorithms, characteristics of algorithms, and the metadata vocabularies that could be partially used to describe algorithms. Some of the core literature that we consulted is provided in Table 1. Besides, we also reviewed and reused the existing metadata vocabularies, like DCAT[6], Schema.org[7], Dublin

[6] https://www.w3.org/TR/vocab-dcat-2/

[7] https://schema.org/

Core[8], PROV[9], FOAF[10], SKOS[11], Bibliographic ontology[12], etc. for their potential use in AMV (following the guiding principles 1 and 3). Further details about the existing vocabularies and their usage in AMV are provided in section 6.

Table 1. Lists the crucial literature that we consulted for our work

Ref. no.	**Literature consulted**
[45]	Voevodin, V., Antonov, A., & Dongarra, J. (2016). Why is it Hard to Describe Properties of Algorithms? Procedia Computer Science, 101, 4-7. doi: 10.1016/j.procs.2016.11.002
[46]	Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. Cambridge (Mass.): MIT Press.
[4]	Skiena, S. S. (1998) The algorithm design manual. New York: Springer.

Step 3: Extraction of terms

Identification of the common properties of an algorithm that apply to most algorithms, like the algorithm type, computational complexity, input, output, creator, etc. In extracting the properties, we first focused on the generic terms which usually all digital assets possess, for example, author, year, and publisher. We also referred to the three algorithm repositories as mentioned above, i.e., the Stony Brook Algorithm Repository, wikidata, and AlgoWiki. We have adopted the elements from them, especially from the wikidata and Stony Brook Algorithm Repository. From the Stony Brook Algorithm Repository, we adopted the algorithm classification schema *by Problem*. From wikidata, we have taken various properties, like download link, average, best, and worst space and time complexity, and a few more (as indicated in Figure 5). The in-depth studies of the algorithm helped us to extract properties, like time complexity, edge case, data structure, implementation, etc. In this step, we focused on the properties specific to algorithms. For extracting these properties, we analyzed algorithms of various domains, such as computer science, computational mathematics, information theory and signal processing, bioinformatics, etc.

Step 4: Analysing the derived terms

In this step, we enlisted, defined, and analyzed all the extracted terms. For example, the terms *average case*, *best case*, and *worst-case* are enlisted and defined as follows: *time complexity of an algorithm on average; time complexity of an algorithm at least; and time complexity of an algorithm at most*, respectively. Similarly, the extracted complex terms were broken into their elemental entities. We analyzed the extracted terms based on their characteristics and then grouped them based on similarity.

Step 5: Synthesizing the relationships among the elements

In this step, we arrange and synthesize the terms based on their relationships. For example, following the previous step 4, the terms *average case*, *best case*, and *worst-case* are grouped and placed under a more generic term *time complexity*. Similarly, the terms like *Algorithm* and *Problem* are associated by specifying their relationship *computes solution to*. The same process has been followed for synthesizing the relationships among the terms.

Step 6: Standardization and Reuse

This step is important because integration and (re)use of the existing vocabularies lead to the development and reuse of a vocabulary. We have reused the terms from Dublin Core Terms, Schema.org, FOAF (Friend of a Friend), SKOS, DCAT, and Bibliographic ontology as described in section 6.

Step 7: Design of representational model followed by Ontology development

In this step, we structure and produce the algorithm metadata vocabulary as an ontology based on the domain knowledge produced in the previous step. We start by modelling the domain knowledge, which includes classes, attributes, and their relationships. This is followed by the creation of a formal model using a formal logic language OWL (following the guiding principle 5). We used the Protégé [23] desktop ontology editing environment for designing the ontology. Further details of AMV vocabulary are provided in section 4.

Step 8: Evaluation

This step involves evaluating the designed metadata vocabulary and the formal model. There is no automatic way of evaluating domain knowledge. Hence, to evaluate, we conducted a survey and the responses were in our favor. For the model, the reasoners verified the structure and consistency of the ontology and executed a set of SPARQL queries prepared by transforming the competency questions provided in step 1. We also did the consistency check for our model using OOPS! Ontology Pitfall Scanner [47]. The results are provided in section 5.3. If the evaluation is successful with all the conditions satisfied then we proceed toward the documentation. If the evaluation fails at a certain step, then we redo the process of analyzing the terms to get a better description of the algorithm. For further details on evaluation, see section 6.

8 https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

9 https://www.w3.org/TR/prov-o/

10 http://xmlns.com/foaf/spec/

11 https://www.w3.org/TR/2008/WD-skos-reference-20080829/skos.html

12 https://bibliontology.com/

Step 9: Documentation
Followed by the successful evaluation, we create a vocabulary specification document that can be used by the user for the seamless use of AMV. In the future, for updating/ versioning, we can proceed to step 1 and follow the steps ahead. The AMV documentation may be accessed from[13].

4 AMV Metadata Model

The AMV metadata vocabulary has been expressed as a formal semantic model. We have developed and published it as an OWL ontology. Anyone (e.g., the repository managers, developers, users) interested in creating and publishing algorithm metadata as a knowledge graph can directly download the AMV ontology and populate it with the data. Then the knowledge graph can be either published on Linked Data Cloud or even can be used for providing direct access to the data as a service through the SPARQL endpoint. The present AMV 1.0 ontology consists of 67 classes, 77 object properties, and 52 data properties. Each of these is further described in the following sections.

4.1 Classes

Following the Top-down approach, we deduced seven top-level facets (as shown in Table 2, from left column 1) providing an overview of AMV. The facets are derived by keeping the algorithm at the center of the study and exploring it from the various frames of reference. The seven top-level facets were further explored and narrowed down to the sixty-seven classes (a class is a collective name that is used for a set of individuals sharing common properties). The derived classes are organized by interlinking them through the hierarchical and associative relations (aka object properties). The hierarchical relation *rdfs:subClassOf* is used to organize the classes in their class and subclass relations. It is worth mentioning here that AMV class A*lgorithm* is defined as equivalent to wikidata *Algorithm* class. This follows the guidelines provided in [48] on Five stars of Linked Data vocabulary use. Table 2 enlists the AMV Classes and *sub-classes* (column 3) and their corresponding example instances (column 4). The classes are interlinked using the various object properties as discussed in the following section 5.2. Besides, the classes are also defined using a set of data properties as described in section 5.3.

Table 2. Depicts the AMV top-level facets along with some classes and sub-classes and some examples of class instances. The classes and sub-classes are separated by "," and the sub-classes are expressed in *italics*. Note that here the sub-classes of "Problem" class are indicative. They can be defined and expanded as per the system scope and objective.

Top-level facets	**Description**	**Class name**	**Example of class instances**
General	describes the general features of an algorithm, for instance, the algorithm type, algorithm notation (the form of expression), etc.	Algorithm, AlgorithmType, FormOfExpression, Comment,	Binary Search, Merge Sort Dynamic, recursive Flowchart, Pseudocode comment
Authority	describes the organization and/ or the person responsible for the algorithm.	Agent, *Organization*, *Person*	Organization is related to the algorithm and the person associated with it.
Right	describes the licensing and rights held on algorithms and related objects, such as software programmes.	RightsStatement, *LicenseDocument*	Creative Commons, Copyleft, Apache
Iteration & solution	describes the various concepts used in the algorithm and the iterations used, for instance, mathematical property, loop types, etc.	MathematicalProperty Loop Types	Linear algebra, Theory of sets For, while do-while
Coverage	describes the algorithm problem and the discipline it belongs to	Discipline Problem *OptimizationProblem,* *CombinatorialOptimizationProblem,* *n-bodyProblem,*	Graph, mathematics Nearest neighbor search, Traveling salesman problem, Shortest path problem, Convex Hull,

[13] https://figshare.com/s/b00de52f297591a84ea3

		PolynomialTimeProblem, NumericalProblem, etc.	RangeSearch, Independent Set, Linear Programming
Environment	describes the environment an algorithm needs when converted into a program, for instance, programming languages, operating systems, etc.	Programming language OperatingSystem Implementation DataStructure	C, Fortran, Java Linux, Windows BSD Sort, N Sort Array,Lists,HashTables
Resource	describes the several types of resources, such as time, space, etc. that are required to run an algorithm	AlgorithmAnalysisFunction, Metric, *EvaluationMetric, TimeComplexity, SpaceComplexity*	Constant, logarithmic

4.2 Object Property

An object property relates an entity with another entity. The entities belonging to the same or different classes are connected using the object properties. AMV consists of 77 object properties which include *creator*, *subject*, *problemType*, *source*, *formOfExpression*, *loopConstruct*, *hasImplementation*, and so forth. The domain and range for each object property are defined. For instance, the object property *isATypeOf* has a domain class Algorithm and has a range class AlgorithmicType.

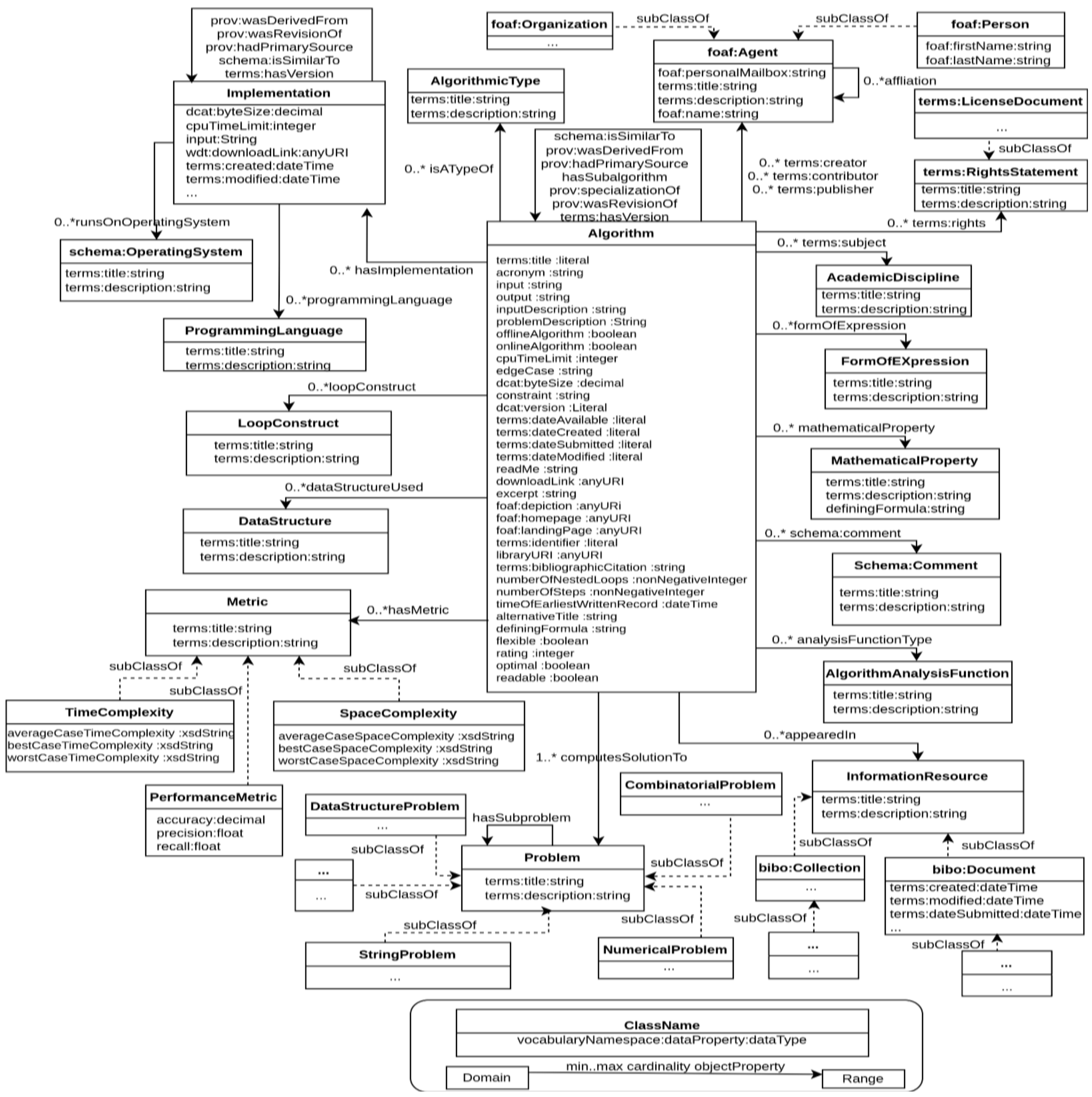


Figure 5. AMV Overview. In the figure the terms without prefixes are, the terms defined in AMV namespace https://w3id.org/amv#. The terms with prefixes indicate the vocabulary namespaces that have been reused in constructing AMV. For example, “foaf” for http://xmlns.com/foaf/0.1/, “dct” for http://purl.org/dc/terms/, and “prov” for http://www.w3.org/ns/prov#. The boxes with dots indicate that some other sub-classes and properties exist, for brevity, we did not show them all. Similarly, the defined inverse relations

are not shown in the figure. For example, in AMV, for an object property *computesSolutionTo* (with a domain class Algorithm and its range class Problem), there is an inverse property *availableAlgorithm* (whose domain class is Problem and range class is Algorithm) exists. In the figure, only the computesSolutionTo is shown. Similarly, properties like *dataStructureUsed, hasImplementation,* and *algorithmType* have inverse properties *dataStructureUsedIn, isImplementationOf* and *isAlgorithmTypeOf* respectively.

4.3 Data Property

The data property is a property that links an entity with its property value. AMV consists of 52 data properties. Each data property is expressed with its respective domain classes and range of data types. Of the total 52 data properties, 12 have the algorithm class as the domain. The rest of the data properties are of the other associated resources like Agent, Implementation, etc. Some of the AMV data properties are *input*, *output*, *bestCasetimeComplexity*, *edgeCase*, *averageCase*, *spaceComplexity*, *cpuTimeLimit*, *constraint*, etc.

Figure 5 provides an overview of AMV ontology. It shows the classes, data properties, object properties, and their cardinality restrictions. It can be seen from the figure that AMV (re)uses the terms from the existing vocabularies, such as FOAF, Dublin Core Terms (DCT), Wikidata, Schema.org, SKOS, and DCAT. For instance, AMV uses some terms from Dublin Core, like Rights statement, License document, creator, title, created, description, identifier, etc. The terms from FOAF are Person, Organization, depiction, page, name, etc. The (re)use of terms from the other existing vocabularies is perfectly aligned with our stated principle in methodology section 3. This approach makes AMV a linked data vocabulary.

5 Evaluation

The evaluation of AMV has been conducted in three diverse ways. The first one is through a user survey, the second one is by executing a set of SPARQL queries and the third is by doing a consistency check using OOPS!

5.1 User survey

The survey was conducted to understand the users' algorithm search pattern, what kind of information they usually seek, how and where they search. The survey questions are not categorized. But logically the questions of the surveys can be grouped into three-wide categories. The survey was circulated among the Professors in Computer Science, Ph.D. students, and software professionals who deal with algorithms on a regular basis. We utilized open-ended questions as a tool, to get additional detail which will assist us with qualifying and clarifying the responses, prompting more accurate information, and furthermore asking some closed questions to get explicit information. After receiving the replies, we examined them and separated the vital requirements in terms of metadata elements.

The three categories of the survey are as follows:

Category 1: In this category, we asked some basic questions from the users intending to know their searching behaviour and the various locations where they conduct their search be it the web or any repository. From these basic questions, we will have an idea about how users conduct searches and where they conduct them.

Category 2: In this category questions where we mentioned some of the basic properties like the number of loops, serial complexity, time complexity, speed, etc. from AMV, and asked whether they consider these properties while searching for an algorithm. In the questions, we only indicated some of the basic properties exhibited by the algorithms and not the complex ones to avoid confusion among the users while answering the survey. From these questions, we will be able to analyse the relevancy of our model when it comes to real-time use.

Category 3: In this category, we had open-ended questions like what keywords the practitioners of the domain use while searching for an algorithm, what filtering criteria they prefer in a repository, what services should an algorithm repository provide, and what implementations of the algorithms they look for. From these questions, we wanted to extract some of the properties from the practitioners, and check whether we have them in our model or not. Note that the above categories 2 and 3 of the user surveys confirm the usage of guiding principle 4 mentioned above.

Survey Analysis. Table 3 provides glimpses of the survey result. This helps us to assess our model and make it more concrete for real-time use. The survey was answered by a total of 10 people, most of them working in the field for at least 5 years.

Table 3: The table shows the questions in the survey and the top responses received

Questions asked		**Top responses to the questions**
Search Location		Internet, Stack Overflow, Standard books, GitHub
Search Keywords		Algorithm name, Specifying problem type
Familiar algorithm repository		Stack Overflow, Github, freecodecamp
Factors considered while searching algorithms		Time and space Complexity, readability, length of the code, efficiency, well documented optimized code
Do you look for the implementation of an algorithm?		100% of respondents said yes
What properties should be mentioned in the description of an algorithm		Sample input, output, time complexity, space complexity, the data structure used, efficiency, popularity, number of parameters and their description

On what basis algorithms should be classified in a repository		Problem types, the data structure used, lesser time-complexity wise, domain, implementation language
Excepted filtering criteria in an algorithm repository		Complexity, scalability, sample data
Services expected from an ideal algorithm repository		Provides search with purpose related keyword, algorithm mapping, link to sample code, link to the research paper, monthly newsletter
Do you consider the number of nested loops as a major factor while searching for an algorithm?		90% yes, 10% no
Do you make a concept-based search for an algorithm?		50% yes, 40% no, 10% not answered
Do you consider the space and time complexity of an algorithm before using it?		100% yes

The category wise survey analysis

In the first category of questions, the user mostly mentioned that they search for algorithms on the Internet and very few of them refer to any repository for searching for an algorithm. 60% of the participants responded that they search on the Internet for algorithms, 10% searched the books and the rest 30% mentioned websites, like Stack Overflow and GitHub repository.

In the second category as we mentioned some properties from our model, like the number of loops, hardware, time complexity, hardware requirements, and applications of an algorithm. 100% of the users mentioned that they consider time and space complexity as one of the major properties of the algorithm. When asked about the number of iterations and nested iterations, 90% of the responses were in the favour that they do consider that in an algorithm.

In the third category of questions, we asked open-ended questions. So, we could extract some properties from the practitioners and match them with our model, the responses included that a user should be able to search in a repository with a purpose related keyword, like pattern matching, data structures used, sample input, time, and space complexity, edge cases, implementation language, description of parameters. According to the users, the above-mentioned properties should be provided in the description of an algorithm. All these responses have not only provided input in making our metadata elements concrete but also provided power to it for real-time usage.

The above-discussed survey and its analysis were one way of standardizing our AMV. The properties like an edge case, implementation, etc. were missing from our Ontology. After the survey analysis, we incorporated them into AMV. Next, we present how our model will work in real-time and how it will answer the various queries of a practitioner.

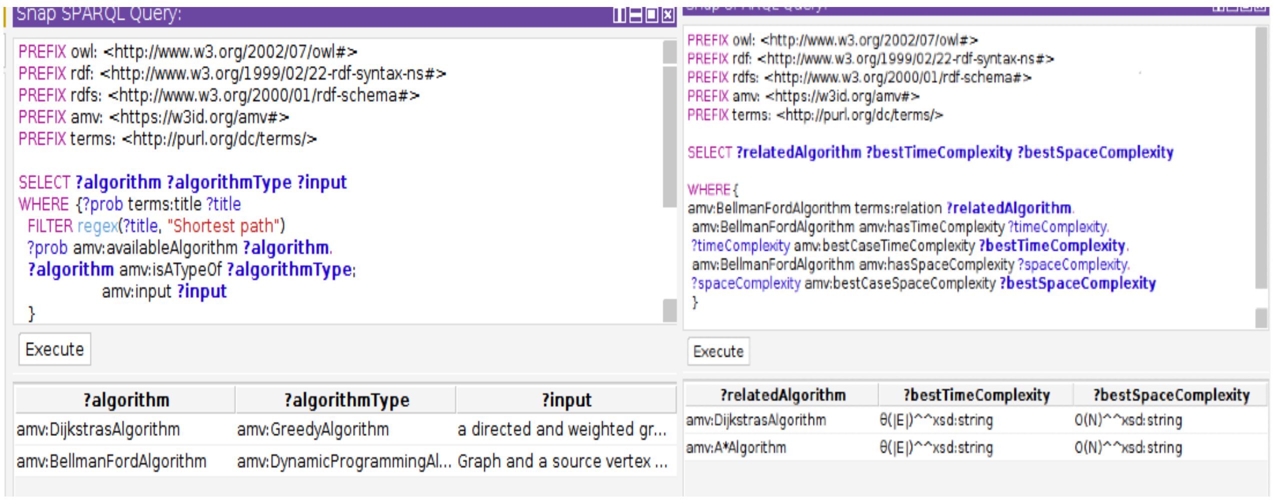
5.2 SPARQL Queries

To evaluate AMV, we execute a set of SPARQL queries. The SPARQL queries are developed by transforming the competency questions enlisted in methodology section 3. The competency queries were developed to indicate the purpose of AMV. Hence, the successful retrieval of desired results will prove the efficacy of AMV and its semantic model. It is worth noting here that before running the queries, we populated the AMV knowledge base with a small dataset consisting of eighteen algorithms and their property values. Table 4 shows one such algorithm and its descriptions in xml/rdf format from the AMV knowledge base.

Table 4: Shows the description of a BellmanFordAlgorithm.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="https://w3id.org/amv#"
    xml:base="https://w3id.org/amv"
    xmlns:wd="http://www.wikidata.org/entity/"
    xmlns:amv="https://w3id.org/amv#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:bibo="http://purl.org/ontology/bibo/"
    xmlns:dcat="http://www.w3.org/ns/dcat#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:prov="http://www.w3.org/ns/prov#"
      …..
 >
   <!-- https://w3id.org/amv#BellmanFordAlgorithm -->

   <owl:NamedIndividual rdf:about="https://w3id.org/amv#BellmanFordAlgorithm">
      <rdf:type rdf:resource="https://w3id.org/amv#Algorithm"/>
      <terms:relation rdf:resource="https://w3id.org/amv#DijkstrasAlgorithm"/>
      <appearedIn rdf:resource="https://w3id.org/amv#NetworkFlowTheory"/>
      <appearedIn rdf:resource="https://w3id.org/amv#OnARoutingProblem"/>
      <computesSolutionTo rdf:resource="https://w3id.org/amv#ShortestPathProblem"/>
      <dataStructureUsed rdf:resource="https://w3id.org/amv#GraphDataStructure"/>
      <hasSpaceComplexity rdf:resource="https://w3id.org/amv#BellmanFordSpaceComplexity"/>
      <hasTimeComplexity rdf:resource="https://w3id.org/amv#BellmanFordTimeComplexity"/>
      <isATypeOf rdf:resource="https://w3id.org/amv#DynamicProgrammingAlgorithm"/>
      <loopConstruct rdf:resource="https://w3id.org/amv#For"/>
      <terms:created rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">1958</terms:created>
      <terms:description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Bellman-Ford
algorithm was first proposed by Alfonso Shimbel (1955), but is instead named after Richard Bellman and Lester
Ford Jr., who published it in 1958 and 1956, respectively.</terms:description>
      <terms:description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Bellmanâ€“Ford
algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a
weighted digraph. It is capable of handling graphs in which some of the edge weights are negative
numbers</terms:description>
      <terms:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Bellman-Ford
Algorithm</terms:title>
      <input rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Graph and a source vertex src.</input>
      <inputDescription rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Works on negative or
positive vertex, in a directed weighted graph.</inputDescription>
      <output rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Shortest distance to all vertices from
src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is
reported</output>
   </owl:NamedIndividual>
```

Figure 6 depicts the SPARQL queries and results for the following questions derived from the competency questions in section 3.1. The first query is Q1: List the algorithms that compute the solution to the shortest path problem along with their types and input. Figure 6(a) shows the query in which the problem i.e shortest path problem is pointed by the user and some properties like the algorithm type and the input are extracted. Similarly, Figure 6(b) depicts the query and the result for the given question Q2: Retrieve an algorithm that solves path optimization problem with its details like best case space complexity, best case time complexity, and the problems related to it. In Figure 6(b), the details about Bellman Ford's algorithm like the related problem and the time and space complexities are shown in the figure. The successful execution of the queries shows the efficiency of the designed AMV vocabulary.



```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX amv: <https://w3id.org/amv#>
PREFIX terms: <http://purl.org/dc/terms/>

SELECT ?algorithm ?algorithmType ?input
WHERE {?prob terms:title ?title
  FILTER regex(?title, "Shortest path")
  ?prob amv:availableAlgorithm ?algorithm.
  ?algorithm amv:isATypeOf ?algorithmType;
            amv:input ?input
 }
```

?algorithm	?algorithmType	?input
amv:DijkstrasAlgorithm	amv:GreedyAlgorithm	a directed and weighted gr...
amv:BellmanFordAlgorithm	amv:DynamicProgrammingAl...	Graph and a source vertex ...

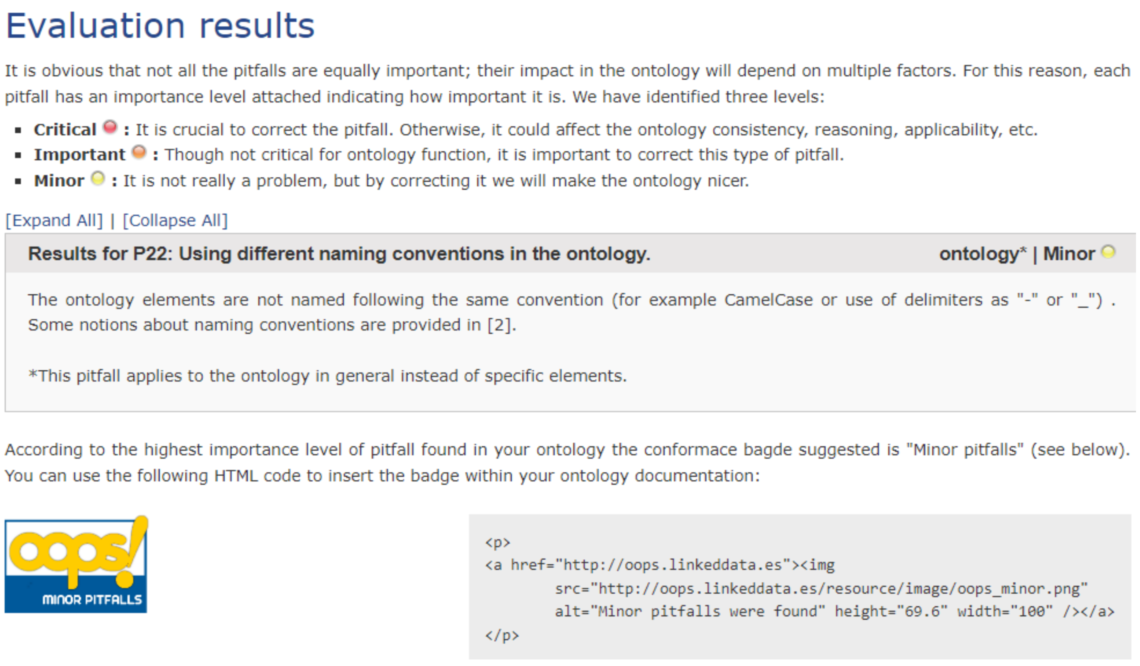
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX amv: <https://w3id.org/amv#>
PREFIX terms: <http://purl.org/dc/terms/>

SELECT ?relatedAlgorithm ?bestTimeComplexity ?bestSpaceComplexity

WHERE {
amv:BellmanFordAlgorithm terms:relation ?relatedAlgorithm.
 amv:BellmanFordAlgorithm amv:hasTimeComplexity ?timeComplexity.
 ?timeComplexity amv:bestCaseTimeComplexity ?bestTimeComplexity.
 amv:BellmanFordAlgorithm amv:hasSpaceComplexity ?spaceComplexity.
 ?spaceComplexity amv:bestCaseSpaceComplexity ?bestSpaceComplexity
 }
```

?relatedAlgorithm	?bestTimeComplexity	?bestSpaceComplexity
amv:DijkstrasAlgorithm	θ(|E|)^^xsd:string	O(N)^^xsd:string
amv:A*Algorithm	θ(|E|)^^xsd:string	O(N)^^xsd:string

Figure 6. Figures 6(a) & 6(b) depicts SPARQL queries and results for Q1 and Q2, respectively.

5.3 Consistency Check

To evaluate the AMV OWL model, it is necessary to check the consistency i.e., to find the logical inconsistencies, if any, in the ontology. Consistency check avoids modeling and reasoning problems and improves the maintainability, accessibility, and clarity of the ontology. For checking the consistency of AMV, we used OOPS! Ontology Pitfall Scanner. OOPS! identifies the pitfalls in the ontology semi-automatically. Some of these are namespace hijacking, creating unconnected ontology elements, merging different concepts in the same class, inverse relationships are not explicitly declared, including cycles in the class hierarchy, and many more anomalies [47]. OOPS! categorizes the pitfalls in three levels, namely critical, important, and minor. Critical refers to the crucial pitfalls and should be corrected; important refers to the not critical pitfalls for ontology function but important to correct them, and minor refer to the pitfalls not problematic but correcting them make the ontology better organized and user friendly. Figure 7 depicts the evaluation result of the AMV OWL model produced by OOPS! scanner. As it can be seen from the figure that no critical issues are found by the pitfall scanner. However, one minor error is reported: "Using different naming conventions in the ontology." It is because, in AMV, the classes and individuals are expressed in the form of CaptilizeEachWord, whereas properties are expressed in the form of camelCase.



Evaluation results

It is obvious that not all the pitfalls are equally important; their impact in the ontology will depend on multiple factors. For this reason, each pitfall has an importance level attached indicating how important it is. We have identified three levels:

- **Critical** ● **:** It is crucial to correct the pitfall. Otherwise, it could affect the ontology consistency, reasoning, applicability, etc.
- **Important** ● **:** Though not critical for ontology function, it is important to correct this type of pitfall.
- **Minor** ● **:** It is not really a problem, but by correcting it we will make the ontology nicer.

Results for P22: Using different naming conventions in the ontology. **ontology* | Minor** ●

The ontology elements are not named following the same convention (for example CamelCase or use of delimiters as "-" or "_") . Some notions about naming conventions are provided in [2].

*This pitfall applies to the ontology in general instead of specific elements.

According to the highest importance level of pitfall found in your ontology the conformace bagde suggested is "Minor pitfalls" (see below). You can use the following HTML code to insert the badge within your ontology documentation:

```
<p>
<a href="http://oops.linkeddata.es"><img
        src="http://oops.linkeddata.es/resource/image/oops_minor.png"
        alt="Minor pitfalls were found" height="69.6" width="100" /></a>
</p>
```

Figure 7. Showing AMV OWL ontology evaluation result using OOPS! Ontology Pitfall Scanner

6 Related Work

As stated above, there exists no metadata vocabulary for describing the algorithm. The current work is the first step towards this. However, as we know, there are hundreds of metadata vocabularies available in the literature, starting from generic purpose metadata vocabularies (e.g., DC, DCAT, schema.org, Machine-Readable Cataloging (MARC)) to the more object-specific vocabularies (e.g., Metadata for Ontology Description and publication (MOD), Friend of a Friend (FOAF)). Here, we briefly discuss some of the relevant vocabularies that have been partially (re)used in AMV.

The CodeMeta project is dedicated to the development of a conceptual vocabulary that may be used to standardise the interchange of software metadata between repositories and organisations. The goal is to establish a minimal metadata schema in JSON and XML for scientific software and code [49]. CodeMeta explicitly re-uses schema.org. CodeMeta provides crosswalk tables providing a mapping between the schema.org elements that CodeMeta re-used and the various other vocabularies (e.g. DOAP, Dublin Core, etc.) and repositories (e.g. wikidata) [6]. In AMV, our primary accentuation is to reuse the current vocabularies that are mainstream and are recommended by W3C. CodeMeta also defined some additional elements which are explicit to software, like software development status, software maintainer, issue tracker, etc. The CodeMeta work is partially related to our work as it deals with software depiction. Whereas, the current work AMV is dedicated to algorithms that have a more extensive degree contrasted with CodeMeta. For software, AMV has a class called Implementation which connects the algorithm to the existing software applications of the algorithm. One more such project is OntoSoft. This also focuses on the metadata of software. Its focus is to promote knowledge sharing about the software developed for geosciences [50]. Its focus is on metadata of software whereas in AMV our main center of attention is algorithms.

Dublin Core schema is a general-purpose metadata vocabulary that is used to describe physical and digital resources like books, CDs, web pages, images, etc [51]. Dublin Core has two sets of metadata: the 15 core elements and in 2000 DCMI recommended a set of qualifiers to broaden the scope of the core elements called the qualified Dublin Core [52]. A portion of the terms like license, access rights, title, description, and others are reused in the AMV as they capture the generic details about the algorithms. The Dublin core terms cannot be utilized for depicting the properties explicit to algorithms. DCAT is an RDF vocabulary designed to improve interoperability across web-based data catalogues [53]. The DCAT description of datasets in catalogs increases the findability and enables applications to absorb the metadata from several catalogs [53]. The DCAT terms like *landingPage* and *byteSize* are reused in AMV. Even though DCAT represents datasets expressly yet the vocabulary misses the mark while portraying algorithms specifically. MOD is a metadata vocabulary for describing the ontologies, it uses equivalent terms from the many pre-existing metadata standards (e.g. ontology metadata vocabulary) to ensure standardization [42]. PROV is utilized to portray provenance information generated in different systems and under various contexts. From PROV, we reuse the property, like *wasDerivedFrom*, *hadRevision*, *generalizationOf*, *hadPrimarySource*, *etc*. [54]. Schema.org intends to describe the structured data on the Web for various entity types, such as Organization, Food, Person, Mind products, Software source code, etc. It covers entities, relationships between entities, and actions yet while depicting algorithms just properties like size, comments and similar can be utilized as the rest of the properties of schema.org cannot portray algorithms expressly [55]. DOAP (Description of A Project) is an RDF Schema and XML vocabulary for describing software projects. It was created to fetch semantic information related to open-source software projects [56]. SKOS (Simple Knowledge Organisation System) is a model to demonstrate the basic structure and to create relations between concepts in thesauri, subject heading lists, etc [57]. FOAF (Friend of a Friend), a semantic web ontology for social connections between people, organizations, and groups. It is an RDF vocabulary of classes of things and properties so that machines can process the information [58]. As detailed above, the vocabularies accessible like DCAT, DCT, Schema.org, PROV, etc. are reused in our work. By utilizing these vocabularies, we can only depict an algorithm in a much-generalized way, we will miss out on the algorithm-specific information. The algorithm possesses certain characteristics (e.g., time complexity, implementation, precision) that are unique to it and not possessed by any other digital asset. The existing vocabularies were created with their specific purposes and none of them can be explicitly used to describe algorithms. This makes our present work pertinent which presents a vocabulary that will unequivocally describe algorithms.

7 Conclusion

Metadata is influential in extracting resources be it print materials or digital assets such as webpages, ontologies, algorithms, video, and more. Apart from playing a vital role in finding and selecting the resources, it supports the reusability of the resource. In this context the current work is significant. As stated in this paper, little work has taken place in managing and preserving the algorithms. It is worth mentioning here that the designed algorithm metadata vocabulary has achieved the 4-star rating according to the particulars given in [48]. Eventually, when AMV is published and is utilized it will accomplish the 5-star. We believe that following the current work, there will be more work in the area starting from algorithm metadata vocabulary development to the dedicated algorithm repository development. In our future work, we would like to extend and enhance AMV by analyzing and aligning it with the other relevant existing vocabularies (e.g., CodeMeta), also by emphasizing the various aspects of the algorithm and related areas, such as versioning, implementation, among others.

References

[1] Abiteboul S, Dowek G, Nelson K-R. What Algorithms Do. In: *The Age of Algorithms*. Essay, 2020, pp. 29–35.
[2] Sedgewick R, Flajolet P. *An introduction to the analysis of algorithms*. Addison-Wesley, 2013.
[3] Fill JA, Ward MD. Special Issue on Analysis of Algorithms. *Algorithmica*; 82: 385–385.
[4] Skiena SS. *The algorithm design manual*. London: Springer, 2012.
[5] Naskar, D. and Dutta, B. Ontology and Ontology Libraries: A Study from An Ontofier and An Ontologist Perspective. In: *Proc. of 19th Int. Symp. on Electronic Theses and Dissertations (ETD 2016 "Data and Dissertations")*, 11-13 Jul 2016, pp. 1-12. Lille, France.
[6] Habermann T. Mapping ISO metadata standards to codemeta. 2018. Epub ahead of print 2018. DOI: 10.7287/peerj.preprints.27153v1.
[7] Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al*. The FAIR Guiding Principles for scientific data management and stewardship. In *Sci Data* 3, 160018 (2016).

[8] Howison J, Bullard J. Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology*; 67: 2137–2155.

[9] Le-Phuoc D, Quoc HNM, Quoc HN, et al. The Graph of Things: A Step Towards the Live Knowledge Graph of Connected Things. *SSRN Electronic Journal* 2016. Epub ahead of print 2016. DOI: 10.2139/ssrn.3199225.

[10] Franconi E, Tessaris S. The logic of RDF and SPARQL. *Proceedings of the twenty-fifth ACM Sigmoid-Sigact-Sigart symposium on Principles of database systems - PODS 06* 2006. Epub ahead of print 2006. DOI: 10.1145/1142351.1142402.

[11] Lincoln M. Using SPARQL to access Linked Open Data. *The Programming Historian* 2015. Epub ahead of print 2015. DOI: 10.46430/phen0047.

[12] Computational algorithm - Encyclopedia of Mathematics [Internet]. [cited 2022 Jun 2]. Available from: https://encyclopediaofmath.org/wiki/Computational_algorithm

[13] Algorithm. In: Wikipedia [Internet]. 2022 [cited 2022 Jun 7]. Available from: https://en.wikipedia.org/w/index.php?title=Algorithm&oldid=1091046319

[14] Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection [Internet]. arXiv; 2016 May [cited 2022 Jun 7]. Report No.: arXiv:1506.02640. Available from: http://arxiv.org/abs/1506.02640

[15] Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger [Internet]. arXiv; 2016 Dec [cited 2022 Jun 7]. Report No.: arXiv:1612.08242. Available from: http://arxiv.org/abs/1612.08242

[16] Redmon J, Farhadi A. YOLOv3: An Incremental Improvement [Internet]. arXiv; 2018 Apr [cited 2022 Jun 7]. Report No.: arXiv:1804.02767. Available from: http://arxiv.org/abs/1804.02767

[17] GitHub - ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite [Internet]. [cited 2022 Jun 7]. Available from: https://github.com/ultralytics/yolov5

[18] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Computat. 1997 Apr;1(1):53–66.

[19] Stützle T, Hoos HH. MAX–MIN Ant System. Future Generation Computer Systems. 2000 Jun 1;16(8):889–914.

[20] Dorigo M, Maniezzo V, Colorni A. The Ant System: Optimization by a colony of cooperating agents. :26.

[21] Hu X, Zhang J, Li Y. Orthogonal methods based ant colony search for solving continuous optimization problems. Journal of Computer Science and Technology. 2008 Jan;23(1):2–18.

[22] Classification of Algorithms with Examples [Internet]. GeeksforGeeks. 2021 [cited 2022 Jun 2]. Available from: https://www.geeksforgeeks.org/classification-of-algorithms-with-examples/

[23] Hart P, Nilsson N, Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans Syst Sci Cyber. 1968;4(2):100–7.

[24] R V. What is Data Structure? Learn more today! [Internet]. GreatLearning Blog: Free Resources what Matters to shape your Career! 2021 [cited 2022 Jun 7]. Available from: https://www.mygreatlearning.com/blog/data-structure-tutorial-for-beginners/

[25] Akhter N, Idrees M, Furqan-ur-Rehman. Sorting Algorithms – A Comparative Study. International Journal of Computer Science and Information Security. 2016 Dec 1;14:930–6.1.

[26] Kalchbrenner N, Blunsom P. Recurrent Convolutional Neural Networks for Discourse Compositionality. Workshop on CVSC. 2013 Jun 15;

[27] Sen S, Katoriya D, Dutta A, Dutta B. RDFM: An alternative approach for representing, storing, and maintaining meta-knowledge in web of data. Expert Systems with Applications. 2021 Oct 1;179:115043.

[28] codecrucks. Correctness of Algorithm - Concept and Proof [Internet]. CodeCrucks. 2021 [cited 2022 Jun 2]. Available from: https://codecrucks.com/correctness-of-algorithm-concept-and-proof/

[29] Lecture Notes | Introduction to Algorithms | Electrical Engineering and Computer Science | MIT OpenCourseWare [Internet]. [cited 2022 Jun 2]. Available from: https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/pages/lecture-notes/

[30] Measuring an algorithm's efficiency | AP CSP (article) | Khan Academy [Internet]. [cited 2022 Jun 2]. Available from: https://www.khanacademy.org/_render

[31] Jungnickel, D.: Algorithms and Complexity. *In Graphs, Networks and Algorithms*. pp. 35-63, Springer, Heidelberg (2013)

[32] Yse DL. Essential Programming | Time Complexity. Medium. 2020 [cited 2022 Jun 7]. Available from: https://towardsdatascience.com/essential-programming- time-complexity-a 95bb2608cac

[33] Antonov AS, Maier RV. A New Representation of Algorithmic Approaches in the AlgoWiki Encyclopedia. *Lobachevskii Journal of Mathematics*; 42: 1483–1491.

[34] Skiena SS. Who is interested in algorithms and why? *ACM Sigact News*; 30: 65–74.

[35] The Stony Brook Algorithm Repository. *Algorithm and Data Structure Repository Https://www3.cs.stonybrook.edu/~algorithm/implement/takaoka/implement.shtml* (accessed June 5, 2021).

[36] Main page. *Wikidata*https://www.wikidata.org/wiki/Wikidata:Main_Page (accessed June 13, 2022).

[37] Fernández-López M, Gómez-Pérez A, Juristo N. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. undefined [Internet]. 1997 [cited 2022 May 26]; Available from: https://www.semanticscholar.org/paper/METHONTOLOGY3A-From-Ontological-Art-Towards-FernCA1ndez-L

[38] On-to-knowledge methodology (OTKM) | Request PDF [Internet]. ResearchGate. [cited 2022 May 26]. Available from: https://www.researchgate.net/publication/2472439_On-to-knowledge_methodology_OTKM

[39] DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolving Engineering of oNTologies. [Internet]. ResearchGate. [cited 2022 May 26]. Available from: https://www.researchgate.net/publication/220836704_DILIGENT_Towards_a_fine-grained_methodology_for_Distributed_Loosely-controlled_and_evolving_Engineering_of_oNTologies
[40] NeOn Methodology for Building Ontology Networks: a Scenario-based Methodology [Internet]. ResearchGate. [cited 2022 May 26]. Available from: https://www.researchgate.net/publication/ 49911337_NeOn Methodology_for_Building_Ontology_Networks_a_Scenario-based_Methodology
[41] Dutta B, Chatterjee U, Madalli D. YAMO: Yet Another Methodology for large-scale faceted Ontology construction. Journal of Knowledge Management. 2015 Feb 9;19:6–24.
[42] Dutta, B., Nandini, D., & Shahi, G. K. MOD: metadata for ontology description and publication. In: *International Conference on Dublin Core and Metadata Applications,* São Paulo, Brazil, September, 2015, pp. 1-9.
[43] Dutta B, Toulet A, Emonet V, et al. New Generation Metadata Vocabulary for Ontology Description and Publication. *Metadata and Semantic Research Communications in Computer and Information Science*; 173–185.
[44] Dutta, B., DeBellis, M. CODO: an ontology for collection and analysis of COVID-19 data. In *Proceedings of 12th International Conference. on Knowledge Engineering and Ontology Development (KEOD)*, Lisboa, Portugal, 2-4 November 2020, vol. 2, pp. 76-85.
[45] Voevodin V, Antonov A, Dongarra J. Why is it Hard to Describe Properties of Algorithms? *Procedia Computer Science*; 101: 4–7.
[46] Cormen TH, Leiserson CE, Rivest RL, et al. *Introduction to algorithms*.Cambridge, MIT Press, 2009.
[47] Poveda-Villalón, María, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. "OOPS!(Ontology Pitfall Scanner!): An on-line tool for ontology evaluation." International Journal on Semantic Web and Information Systems (IJSWIS) 10.2 (2014): 7-34.
[48] Janowicz K, Hitzler P, Adams B, et al. Five stars of Linked Data vocabulary use. *Semantic Web*; 5: 173–176.
[49] Codemeta. The codemeta project. *The CodeMeta Project*https://codemeta.github.io/ (accessed June 13, 2022).
[50] Gil Y, Garijo D, Mishra S, et al. OntoSoft: A distributed semantic registry for scientific software. In: *2016 IEEE 12th International Conference on e-Science (e-Science)*, Baltimore, MD, USA, 23-27 October 2016, pp. 331-336. DOI: 10.1109/eScience.2016.7870916.
[51] Weibel S. Metadata: The Foundations of Resource Description. *D-Lib Magazine*, 1995.
[52] Weibel SL, Koch T. The Dublin Core Metadata Initiative. *D-Lib Magazine* 2000; 6.
[53] w3c. w3c/dxwg: Data Catalog Vocabulary (DCAT). *GitHub*https://github.com/w3c/dxwg/ (2021, accessed 1 May 2021).
[54] Prov-O: The prov ontology. *O*https://www.w3.org/TR/prov-o/ (accessed June 13, 2022).
[55] Product. *Product - Schema.org Type*https://schema.org/Product (accessed August 20, 2021).
[56] Ewilderj. ewilderj/doap: RDF schema for describing software projects. *GitHub*https://github.com/ewilderj/doap (accessed August 12, 2021).
[57] Miles A, Matthews B, Brickley D, et al. SKOS Core: Simple Knowledge Organisation for the Web. In: Proceedings of the International Conference on Dublin Core and Metadata Applications.2005.
[58] Yu L. FOAF: Friend of a Friend. *A Developer's Guide to the Semantic Web*; 357–382.