Formal Model of Problems, Methods, Algorithms and Implementations in the Advancing AlgoWiki Open Encyclopedia*

Andrey Popov[1], Dmitry Nikitenko[2][0000-0002-2864-7995], Alexander Antonov[2][0000-0003-2820-7196], and Vladimir Voevodin[2][0000-0001-6036-5106]

[1] Lomonosov Moscow State University, Moscow, Russia
dr10x@mail.ru
[2] Research Computing Center of Lomonosov Moscow State University, Moscow, Russia
{dan,asa,voevodin}@parallel.ru

Abstract. Any automation, modeling or computing is based on algorithms. The choice of methods, algorithms and efficient implementations is a very important question. AlgoWiki Open Encyclopedia is an online resource which was developed to describe all existing algorithms as fully as possible, provide information about its properties, give example of its implementations and completely classify all algorithms. The first version of the AlgoWiki Open Encyclopedia did not have a formal model, describing relationship between the four key entities, which are problems, methods, algorithms and implementations. The main idea of the advanced approach is to create the formal graph model of algorithm classification, which could be used by AlgoWiki website itself or any trusted web service for many purposes, and first of all, organizing existing descriptions of algorithms into the proposed core structure preserving the data integrity during all possible routines.

Keywords: Algorithms Classification · Relationship Graph Model

1 Introduction

Any type of computing is based on some algorithms. The choice of methods, algorithms and proper implementations is a very important question [1]. The AlgoWiki is an online encyclopedia of algorithms, created with purpose to describe structure and key features of various algorithms and provide complete information about different implementations of these algorithms on different platforms and describe its properties such as parallel complexity and parallel structure, efficiency, scalability and etc. The service also classifies all the algorithms, making navigation easier for user [2].

* The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University. The results described in section III were obtained in Lomonosov Moscow State University with the financial support of the Russian Science Foundation (agreement N 14-11-00190).

There are four levels of entities in the AlgoWiki: Problem, Method, Algorithm and Implementation. Problem level describes the problems which need to be solved using methods or algorithms. Method and Algorithm levels may seem similar, but they are actually not. The main difference between them is that method is slightly more abstract description of way or technique to solve problem, while algorithm descriptions are more extended and more particular. For example methods can be described with just simple words, but description of algorithm has to be supported with flow chart or pseudocode statements for example, because algorithms are more certain about the sequence of actions, exact amount of data operated, its type and its inputs, outputs and etc. And Implementation level shows the certain examples of implementing the algorithm, i.e. example codes on most popular programming languages. This is the least abstract description stage, because algorithm implementations are written on specific programming language and is oriented to be executed on specific platform. According to the idea of AlgoWiki algorithm classification, one can look at the problem which has to be solved, and find the list of methods which can be used as a solution. Within the method, one or several algorithms can be chosen, depending on whether there are different versions of applying the method. Finally, on the algorithm page user can find its Implementations. But sometimes it will not be a simple Problem – Method – Algorithm – Implementation path for the user, because of some unusual relationships. Some problems can be solved with several methods, but in the same way one method can be used to solve several different problems. Sometimes it is also possible to go directly from problem to algorithm, skipping the method stage. Some problems can have several subtypes or versions of itself, and each of that subtype will be a part of Problem level. Or problems can be grouped together into categories of problems. The same principles can be applied to methods and algorithms. Just like problems, they also may have subtypes.

So as it turns out, algorithm classification is more complicated than it could have been imagined at the first stage of the project and there's a strong need in a special core description and management.

2 Background

At the moment AlgoWiki uses wiki engine, meaning that it is a set of webpages which are connected with direct links. This approach is simple, but it has some shortcomings. There was no formal model describing relationships and dependencies at the first project stage, so it was not represented by some kind of data structure. Furthermore there's no system which would stick problems, methods, algorithm and implementations together, nothing takes care of the key data when it is getting changed. If webpages are only connected with direct links, when accidental changing mistakes with links can lead to disorders. On long term basis mistakes are inevitable, just because of human factor. This is already enough to call the approach not sustainable, but things become even more confusing and easy to mess up with, if the website wants to support several languages and, in fact, it is planned to cover as many languages as possible. AlgoWiki is also planned to be reliable scientific source of information, so other people can refer to it in their work. But referring by just leaving the link is not reliable enough, because there is no guarantee that after the site was edited the original referred page would be accessible. The other problem of this approach is that it becomes less effective as the website grows in size. The lack of systematic strategy becomes more critical as more and more webpages are added to the website, because it starts getting more difficult to operate.

Most of these issues can be tackled with advanced wiki engines, but there's a key reason to follow another way.

As soon as the idea to generate a Top500 like dynamical performance rankings regarding any available object model slice (problem, method or a group of methods, algorithm or a set of algorithms) together or in separate with hardware details appeared, it became obvious that AlgoWiki needs to change its engine to make efficient access to the object descriptions [3].

3 Proposed Approach Principles and the Graph Model of Algorithm Classification

The structure of original algorithm classification model could be imagined as a forest, where every single tree represents problem level, branches represent methods and optionally their subtypes, then thinner branches are algorithms and possibly algorithm subtypes. Finally, the leaves of these trees stand for implementations and there can be more than one leaf at the end of each last branch. In fact the woods are going to be very leafy, because most algorithms have a lot of implementations and it is planned to include in the model as many of them as possible.

But as it was mentioned earlier, it appeared that a same method can be used to solve different problems. For example Breadth First Traversal can be used for different purposes like detecting cycles in undirected graph or finding if it is possible to travel between two nodes in a graph. So many to many relationships are also possible in the system, so the forest should be tropical, with lianas hanging around and connecting trees together.

Hopefully this analogy helps understanding the structure, but it is not correct to call this model a tree, because some vertices are allowed to have more than one parent node, so the graph may contain loops. But apart from that the graph shares some common features with the tree, because it has a root, a node which exists only for technical purpose to be a parent for problems, starting vertex of the problem level objects, connected unrelated parts together and making a single connected graph.

On the problem level of classification, the nodes can represent single problems, their subtypes, categories or even a subject of problems, for example "Algebra Problems". Root vertex is formally a part of the problem level and can be called as "All problems". Methods and Algorithms can have subtypes as well as problems. Implementations due to the fact that they are leaves, cannot have any subtypes or any kind of further extensions.

In this, we can built a formal model of objects relationship (Fig. 1).

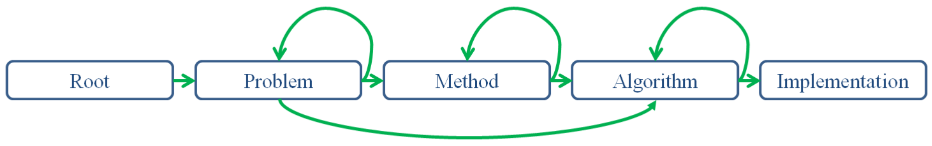

Fig. 1. Acceptable parent-child relationships

The model is quite flexible and anything is possible, except the following restrictions applied to all parent-child relationships in the model:

– The child node must have the same level as the parent node or lower. Problem level is meant to be the highest and, obviously, implementation level is meant to be the lowest.
– The parent for the Implementation level vertex must be Algorithm level vertex.
– The child node of the root must be a Problem level vertex

The graph model solves the problem of the lack of system of the old approach, because now the all relationships between key entities of the website exist not only in the heads, but also in a formal model, which can be synchronized with the AlgoWiki website and avoid accidental confusion with working with it in the future.

This model is the main scientific contribution of the paper. There can be various implementations and data structures developed, but they should all cover the proposed schema.

4 Implementation

4.1 Choosing Data Structure to Represent the Model

This graph model was decided to be represented with relational database techniques because of one simple key reason – all the benchmark data for all presented in AlgoWiki implementations are best managed with these techniques, and it would be easy and reasonable to use the same approach to objects description when building object or hardware related rankings. At the same time the relational database describes many-to-many relationships good, and in many ways.

The reason why graph or other database approaches [4] are not attempted is because AlgoWiki will have its Algorithms Performance rankings, lists of algorithms, based on their linear and parallel properties, which is the fastest, which is more efficient and etc. It is easier to implement it with relational database model. In addition, AlgoWiki algorithm classification model is too simple and the graph itself is relatively small. There is no need in features of graph database model, such as being able to deal with millions of vertices and edges and extreme parallelism [5].

The PostgreSQL will be used in production mode probably, which is used to store and manage the results of implementation runs for a number of algorithms, based on different parallel technologies and run options.

Actually, the database implementation of AlgoWiki core was pushed by evolving need for linking the implementation run results with the AlgoWiki structure, allowing creating various ranking slices[3]. Authors find two real ways of future problem solution.

The first is based on development of MediaWiki interface, in this case every time when an on-the-fly ranking is built, the ranking side query has to connect to the Wiki first, extracting the needed data, and run a query in its own database then. Another option is to have a duplicate object description in the database of the ranking, refreshed by schedule.

The second way has an idea of centralized AlgoWiki core management tool, based on database object descriptions. In this case, both Wiki and the ranking get the structure information out of that utility. The paper discusses the possible way of following this second way.

4.2 Relational Database Possible Model

Problem, method, algorithm and implementation are the four different levels of elements existing in the AlgoWiki algorithm classification (Fig. 3). Inside the proposed database model they all are treated as the same entity, which is called Object.

The Object table is the heart of the database schema and it lists all the problems, methods, algorithms and implementations, giving each of them unique identifier, Object_ID. This identifier plays important role as a primary key for Object and foreign key for several other tables, but it can be used for external purposes. For example, trusted apps or resources can reference a certain problem, method, algorithm or implementation by using its unique identifier in this relational database.

The fact that Object instances must be one of four different types, i.e. problem, method, algorithm or implementation is represented with the aid of Level table, which lists all the four levels of algorithm classification and Object table refers Level table with a foreign key, Level_ID.

Object entity has many-to-many relationship with itself, so there is a separate table Object_has_Object which describes all the parent-child relations. This table stores a pair of identifiers for parent and child objects, which are essential attributes, but it may be useful for technical purpose to store who and when created the connection as well and some tables would have similar attributes with similar technical purpose.

The table Object_Fixing is used to store the identifiers of those objects, that are being edited at the moment, and will not be available for editing.

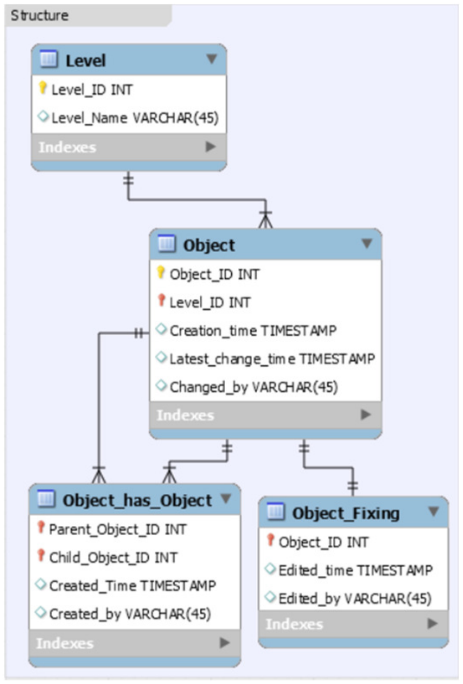

Fig. 2. Relationships between Object, Level, Object_has_Object and Object_Fixing entities

This is necessary to prevent collisions, occurring when two people simultaneously edit the same object or one started editing the object before the other finished editing and saved his changes. The alternate way of achieving it was having a boolean flag attribute in the Object table, but this option was denied, because among hundreds of objects only few will be edited simultaneously, so there is no point in spending extra memory capacity on storing lots of "false" values. It is more sensible to store the list of those few objects.

Apart from Level and Object, the database model has two more entities. Those are Language and NameType (Fig. 4). The idea is that any object has different instances of the same attribute. For example an algorithm has a name, but it may have a full name or a short name, and these names will be different on different languages. All the names of different types and descriptions on different languages are stored in separate tables Object_has_Name and Object_has_Description respectively. If necessary, more tables like can be added, if there will be a necessity in storing information about other object attributes. The primary key of each attribute instance is a composite key of Object_ID and

Language_ID foreign keys. Object_has_Name has NameType_ID in addition, because there are different types of names. Many algorithms or methods may have many names on many languages, but the combination three identifiers is unique.

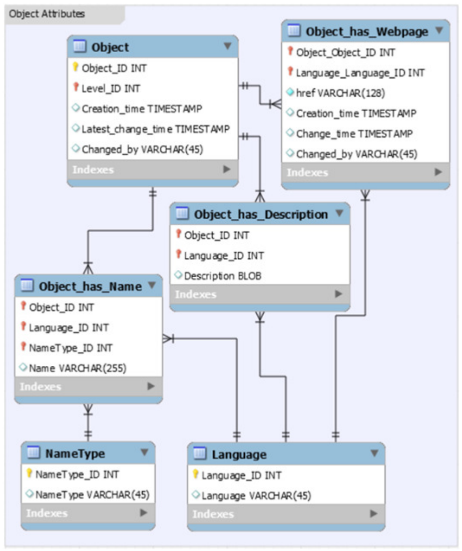

Fig. 3. Object, Language and NameType entities and three more object-owned attribute entities which depend on their combinations

The table, Object_has_Webpage allows making objects related with existing webpages. That can be useful to solve reference problem. As was earlier mentioned, AlgoWiki is planned to be used for scientific purposes, so additional attention should be paid to make a better way to reference the resource information. Referring to an algorithm with a direct hyperlink may not be not sustainable enough, because the algorithm is not attached to its webpage firmly and after a series of website changes and fixes, the referred algorithm may be deleted or moved to different page or something else can happen. But if instead the unique identifier of the algorithm was used to refer to it, there would not be such problem. In this case, if the algorithm webpage changed its URL address, this change will be reflected in the corresponding row in Object_has_Webpage table. Making things work as it was described may eliminate the chance that someone would end up referring to an empty page which used to be the algorithm old page for an instance.

As a result, the proposed relational database schema stores all the necessary information about the formal graph model of AlgoWiki algorithm classification. The schema is capable of scaling and adding new things, for example new language or object-describing attributes. In addition, it creates new opportunities for trusted websites or apps, which need using AlgoWiki structure data.

4.3 Database Handling Algorithms

When the first version of the schema was created, it was uncomfortable and confusing to insert some test data into the database by hand and it was very easy to mess up with it, because the database is normalized. However, it is not a big problem to automate the process with programmed tools, and normalization is very helpful in increasing the queries running speed. There is another important reason to create algorithms which is to make it easier to work with the database in the way that would be safe for it. It is vital to preserve the data integrity, when inserting or changing values in the database and prevent the data being orphaned due to the object deleting in improper way. There are seven main algorithms, which make all the essential validity checks for better creating, updating and deleting data:

- Creating new object
- Creating a relationship between the object and another object
  - Stating parent (adding new parent connection)
  - Stating child (adding new child connection)
- Replacing the object's parent
- Deleting the relationship between the object and its parent
- Relationships integrity check
- Erasing the object

The algorithms listed above ensure that the three restrictions of the graph model of algorithm classification are followed, i.e. it eliminates the chance of creating illegal relationships, such as implementation being a parent of a problem, for example.

The person who edits the graph model dose not actually edit the entire graph. He only works with just one object, creates new one and sets its type and relatives or picks an existing object and updates or removes it. When the object is picked to be edited, a new row with its Object_ID, current timestamp and the username of the author is added to the Object_Fixing table, which makes the object blocked for anyone else to edit it, so there is no way two people make changes for the same object at the same moment.

Another important notice is that in order to be executed, some algorithms require the chosen object node to satisfy certain conditions. Database editor is not allowed to delete objects if they have children. The reason is to avoid making objects orphaned and creating gaps between nodes. Root is the only vertex which does not have any parents and it would be strange if it did. Other vertices must have at least one parent element. So in order to cut down a big branch of the AlgoWiki "tree", editors should cut off all the smaller branches attached to the big branch first or maybe even start with shearing all the leaves, if the smaller branches have children as well. Note that it is not expected that the formal graph model will be built from scratch so badly, that large changes will be required. The algorithms are designed to make small changes in it.

Another algorithm which has a limitation is Deleting the relationship between the object and its parent. It is allowed to execute the algorithm only if the object has more than one parent. Obviously, all non-root objects must not be left without any parents, so if there is only one parent, it cannot be deleted, but instead, it can be replaced, which is basically deleting the old relationship with parent and creating new one combined into a single transaction.

Creating a relationship between the object and another object algorithm does not need any special conditions. It is always allowed to be executed, but there is an important notice, that parent-child relationship can only be created if both objects exist, i.e. those which exist in the Object table and have an associated identifier. It is illegal to create relationships with non-existing objects. Building graph model starts from a single vertex, the root. Then creating objects on problem level and stating their parents, then adding "younger" vertices and stating their parents and so on. Objects are added to the model one by one. And the case than the child is stated is only possible when those nodes were added earlier. So most commonly the person editing the object will create relationships with parents, rather than with children.

Before looking on the algorithms with more details, it is important to recall, how creating object and creating relationships between two objects are implemented in the database. Creating the relationship is represented by adding a new row with a pair of parent and child identifiers to Object_has_Object table, which is the edge list of the graph model. There are two more attributes: Created_time and Created_by to store secondary technical purpose information. In the same time adding new row to Object table is not enough to create an object, with exception for root vertex. In order to add the object to the graph model it has to be connected with at least one other object, which would be its parent. So stating the parents and children is an important part of creating new object.

Those algorithms were created with three assumptions being made. The first one is that in the Level relationship Level_ID and Level_Name correspond in the following way: 1 – Problem, 2 – Method, 3 – Algorithm, 4 – Implementation. It is most likely to be, but it does not matter that much, just a notice in order to avoid misconception of further algorithm descriptions, where comparison operators were used. The second assumption is that the root object is a part of problem level of classification, so its Level_ID equals 1. And the third assumption is that the root vertex has Object_ID equals 1, which will be used by some algorithms to identify root object among others.

Creating new object algorithm consists of the four major steps. The first one is to state the classification level of the object, i.e. its Level_ID. The second step is to state the parents of the object. Unless the object is the root of the graph model it is forced to have at least one parent node, to ensure that it would not be an orphan. After one stating the first parent, the algorithm suggests adding more parents. The next step is optional and it is adding child nodes relationships. And finally, the last step is adding a new row into Object table and adding rows into Object_has_Object table for each stated relationship, both actions form a single transaction. The object instance must not exist without at least one edge instance. In specific case if the Level_ID of that object equals 4 or, in other words, the object is an implementation, it has a different validity check which makes sure that the parent will be an algorithm level object and does not give a chance to create any kids relationships or stating additional parent for implementation object.

Creating a relationship between the object and another object algorithm is similar to steps of the previous algorithm. Validity checks are identical, parent Level_ID must be lower or equal to the child Level_ID and if the edited object is an implementation (Level_ID=4), its parent must be an algorithm (Level_ID=3). The outcome of the algorithm is a new row added to the edge list table.

Relationships integrity check algorithm traces the Object_has_Object tables and the Object and for each object in the first table it looks at its relationships with parents in the second table and checks whether its relationships follow the restrictions or not, ensuring that all non-root objects have parent node. There is no need in looking at both parents and children, because the relationship can be checked if it is legal either way, so it is looking only at object parents, so root object is skipped. The algorithm may not seem essential, because illegal inputs are prevented by other algorithms, but who knows what can happen. At the end of a day nobody wants to lose data integrity, so being extra safe might be sensible. In addition, this algorithm may be used to indicate the error, if something wrong turned out to happen.

Erasing the object algorithm includes, first of all, dropping all the rows from Object_has_Object, where Child_Object_ID equals to the identifier of the object which is going to be deleted. Then the row representing the object is removed from Object table. It is important to do both. Just like it is not right to have relationships with objects not existing yet, it is not appropriate to be related with objects which no longer exist. So all edges connecting with parents are erased first and then the node itself is removed, preventing the edge list table to store the rubbish.

Implementing the following algorithms is beneficial, because the data within the database will be protected and its adding, updating and deleting will be much easier for people. Validity checks prevent the system from being damaged by random errors caused by human factor, so it is no longer an issue. As a result AlgoWiki ends up with a set of helpful algorithms.

5 Conclusion and Future Work

In fact, AlgoWiki just made the first steps towards its full potential. With the aid of implementation of the brand new relational database schema and the set of algorithms managing it AlgoWiki will be switched on the new engine. The seven algorithms described earlier would be the basement for AlgoWiki own Content Managing System (CMS). The new approach will allow everyone making changes in the AlgoWiki Open Encyclopedia. It is obvious, that ordinary website visitors will not be able to change the graph model, only trusted users will, so there are going to be different types of permissions for different users. As a result, the website preserves the good features of the wiki engine being synchronized to the formal graph model makes it significantly less vulnerable to random errors and more capable of expansion without quality losses.

As an extension of AlgoWiki, it is planned to create algorithm performance rankings [3]. The idea is to compare different implementations of different algorithms, based on their various properties, such as time complexity, memory efficiency and etc. Users will be able to filter the rating list, by displaying only the implementations of certain algorithms or written on certain programming languages or written for specific platform, allowing them to analyze and play with data. It is most likely that the renewed Top50 engine [6] will be used as a base for this service development.

In the same time, AlgoWiki is planned to be translated on many languages. Although the knowledge of English is essential for people involved in IT sphere, supporting many languages will make AlgoWiki more accessible for those, who does not speak English well enough.

Summarizing everything, the lack of data organization, possibility of random errors and some other the problems of the old version of AlgoWiki can be solved by creating the formal graph model describing the relationship between the instances of the four levels of algorithm classification. The graph model is implemented in the form of normalized relational database. In addition, seven algorithms helping manage the database have been developed. Although all the accomplishments have importance on the basic level, a lot of work has to be done in future and there are many ways AlgoWiki can be extended and developed.

References

1. Vl. Voevodin, A. Antonov, J. Dongarra: Why is it hard to describe properties of algorithms? In: Procedia Computer Science, Vol. 101 (2016). pp. 4-7. https://doi.org/10.1016/j.procs.2016.11.002
2. AlgoWiki Open Encyclopedia. [Online]. Available: https://algowiki-project.org
3. A. Antonov, J. Dongarra, Vl. Voevodin: AlgoWiki Project as an Extension of the Top500 Methodology. In: Supercomputing Frontiers and Innovations, Vol.5, No.1 (2018). pp.4–10. https://doi.org/10.14529/jsfi180101
4. P. Kostenetskiy and L. Sokolinsky: Analysis of Hierarchical Multiprocessor Database Systems. In: Proceedings of the International Conference on High Performance Computing, Networking and Communication Systems 2007 (HPCNCS 2007),. pp. 245-251, 2007.
5. C. Pan and M. Zymbler: Encapsulation of partitioned parallelism into open-source database management systems. In: Programming and Computer Software, 2015, vol. 41, no. 6, pp. 350-360.
6. D. Nikitenko and A. Zheltkov: The Top50 list vivification in the evolution of HPC rankings. In: Parallel Computational Technologies, vol. 753 of Communications in Computer and Information Science (CCIS), pp. 14–26, Springer International Publishing AG, New York, 2017. https://doi.org/10.1007/978-3-319-67035-5/_2