Social Dynamics Used for Verification of Large Proof Trees.

Jianchi Chen
California Institute of Technology
1200 E. California Blvd.
Pasadena, United States
jchen2@caltech.edu

Ying-Yu Ho
California Institute of Technology
1200 E. California Blvd.
Pasadena, United States
yingyu@caltech.edu

Tim Holland
California Institute of Technology
1200 E. California Blvd.
Pasadena, United States
th0114nd@gmail.com

Kexin Rong
California Institute of Technology
1200 E. California Blvd.
Pasadena, United States
krong@caltech.edu

Abstract

A mathematical proof has the structure of a DAG. However, it is typically presented in a linear fashion, and cannot practically expose the depth of its roots. We propose a web service to provide both of these by exposing part of the graph to the user and allowing them to explore down into the subproofs. Naturally it would be prohibitive for the authors to write even a useful amount of the tree in, so the service allows the introduction of new proofs to expand the reach of the proof trees. Specification of proofs could either be very formal, in which case a program could verify them or more relaxed and reviewed by other users in a social effort to find mathematical truth.

1. INTRODUCTION

We propose a web application that allows the inspection of, addition to, and potentially input of verification of collection of large proof DAGs. Interfacing with a node of the tree will present a statement of the theorem associated with that node, edges to the children and potentially to some of its parents. Text accompanies the node, and explains how the children are combined together to prove the theorem. Ideally the accompanying text is as small as possible and the nodes have a small fan out to break the proof down into digestible chunks.

We have two possible choices to proceed on how to keep the DAG logically sound. The first is more of a technical problem: require highly precise statements of additions to the tree to be parsed, the logical statements can then be verified (or rejected) automatically. Alternatively, with a more social emphasis proofs could be stated in natural mathematical language. Under this scheme, the correctness would have to be verified by users, through a combination of moderation, reputation, and democracy.

2. TECHNICAL HURDLES

At the very basic level we have to deal with all the usual problems in building a webapp: deciding upon tools (languages, databases, frameworks) and an API to communicate between client and server. Further features needed would be dynamic loading of the graph as the user scrolls through, search to look for other proofs to minimize redundancy, automated theorem verification or a voting mechanism, ability to mention other theorems to include them is lemmas in a particular proof, text entry for a theorem, inclusion of images for proofs that necessitate it, and collections of axioms for the various fields of mathematics that could be present as a starting point. Hard problems would involve the choice of data structure for the proofs, intuitive presentation of the DAG to the user, an effective mechanism of rejecting poorly worded or incorrect proofs.

3. POTENTIAL ORDER OF IMPLEMENTATION

1. Persistent storage of a proof in the database.
2. Communication of a proof over the network.
3. User interface.
4. Adding a new theorem from the user interface.
5. Persistence of theorems in UI across page loads.
6. Verification system (social or formal).

4. BACKGROUND

I'm not aware of any similar projects.

5. TIMELINE

5.1 Initial Timeline

Below is the initial project timeline we proposed at the project proposal presentation. The plan was to build a working prototype before midterm, and to add as many features as time allows afterwards.

- Week 2 - 5: Basics
  - Client-server API
  - Database API
  - Implement views and templates for browsing, editing, and adding theorems
  - Front end visualization
- Week 6: Milestone:
  Should have a simple working prototype, which allows adding, deleting and viewing nodes from the proof DAG
- Week 7 - 9: Useful advanced features
  - Search by keywords
  - User profile
  - private graph
  - Auto-keyword-extraction
  - Suggested theorems

5.2 Actual Timeline

Below is a summary of our actual weekly progress based on the blogs. The front end progress was highly non-linear because we either underestimated the amount of work or overestimated the ability to stick to schedule of the person in charge. Thus many features were first implemented on server-side and enhanced with Ajax later. Nevertheless, we were able to implement the three most useful advanced features: keyword search, user profile and private graph. A detail description of the end product is given in the next section.

1. *Week 1: Changing Gears*

   Due to the discovery of a fairly comprehensive open source traffic simulator project, we decided to abandon the original project proposal and to work, instead, on a web app to accommodate the visualization of mathematical proof tree structures. We discussed the basic application architecture: the typical LAMP stack with a REST API and CRUD functionality. In our case the P stands for Python with Django REST framework, the M stands for SQLite, and the A could either be Apache or Amazon Web Service.

   The first steps we took were similar to any team making a CRUD app: setting up the database and a method of communication between the client and server. Specifically, we determined a schema for the database and fleshed out data structures for the DAG. We also started to familiarized ourselves with the frameworks that we will be working with for the rest of the term.

   On the front end side, several JavaScript frameworks quickly came to our mind: AngularJS and/or jQuery for DOM manipulation, Bootstrap for style and UI components, and D3.js for data visualization. These frameworks have some overlapping functions so it was important to use them in a way that avoided conflicts and, preferably, reflected separation of concerns. Since we had little experience in web development (actually the main front-end architect had none), it would take significant time to research on JavaScript and its frameworks before making design decisions.

2. *Week 2: Getting Started*

   This week the first draft of the API was written. It's a fairly standard one that supports the collection of all entries in a paginated manner, getting a single entry for a close up view, and the CRUD operations expected to have.

   On the server side, we've decided to go for AWS instead of Apache. As a result, we switched to using virtualenv, which gave us consistent versioning across our local machines as well as allowing us to use versions required by AWS without having conflicts of the development computers. We also migrated from SQLite to MySQL, since AWS had the best support for the latter. Although setting up MySQL was slightly frustrating.

   On the backend side, we implemented serializers to help form JSON responses, and view methods for creating and reading theorems/proofs.

   On the front end side, we started by finding data visualization examples to understand the power of available tools. One of our particular interest was **d3 process map** (https://github.com/nylen/d3-process-map), which demonstrated a hierarchical layout of graph that we would eventually mimic. Meanwhile, we realized that AngularJS, jQuery, and D3.js all provided DOM manipulation methods, which had the most potential of conflicts. Hence we decided to structure front end applications on MVC (model-viewer-controller) framework of AngularJS, avoid jQuery as possible, and use D3.js mainly as a graph layout engine.

   No JavaScript codes were written yet, but we were able to write some Django view templates to provide text-based interaction with DAG.

3. *Week 3: Backend Milestone*

   After adding templates and view methods for edit and deletion, we had a fully working set of CRUD functionalities implemented this week. We also did substantial unit tests on database models and api calls to make sure that our backend was robust so far.

   At this point, the backend has already reached the original midterm milestone. In the front end, we succeeded in driving AngularJS templates with D3 graph layout engine. However, time was wasted on studying hierarchical force layout algorithm of d3 process map. While force layout was provided by D3.js, it turned out that the hierarchical part was manually specified in the dataset, so we needed to invent our own algorithm. A minor issue was that we formatted graphs as

adjacency lists while D3 Force Layout used nodes and edges as native objects, which required some conversion.

4. *Week 4: Moving on*

   This week, the backend side has moved on to advanced features, starting with the keyword related ones. We implemented keyword creation, edit and search. Users can add an arbitrary number of keywords for theorems/proofs, and a search bar was added on the index page to allow search by keywords.

   As we moved one person from the backend to the front end, the front end side has made substantial progress this week. We wrote a stylesheet for the website to make it more user friendly. We also set up MathJax, a free online LaTeX compiling tool, to support adding and displaying LaTeX code on the website.

   Finally, we built a client-side knowledge graph editor that made it possible to implement and test a working graph layout algorithm by the end of week. This graph editor also served as an exercise on Bootstrap UI components and a reference of our final user interface. Our next mission was turning the graph editor into an Ajax application that was able to communicate with the server.

5. *Week 5: MVP online*

   This week, we have been focusing on putting a simple working version of our product online. On one hand, we tried to connect the knowledge graph visualization with backend database. On the other hand, we worked on setting up Amazon EC2 (Amazon Elastic Compute Cloud) to host the website. We were able to seed the database with number theory theorems and deploy the latest backend code. Time was spent on fixing API for Ajax calls, delaying the integration of knowledge graph visualization.

   Meanwhile, we have also been extending the website's functionalities. We started to implement user authentication system in the hope of enabling users to set up private knowledge graphs for classroom settings.

6. *Week 6:*

   We made a single-page application that loaded latest submissions and performed searches via Ajax calls. In addition, we were able to display the whole graph in database with our graph layout algorithm. Then we needed to make the graph interactive.

7. *Week 7:*

8. *Week 8:*

6. END PRODUCT

The end product would hopefully be a useful web application that would allow publishing of proofs in a graphical format subject to immediate review of other users, and a repository for a large collection of proofs to be seen in a graphical format upon request.

6.1 UI and Visualization

6.2 Web Framework

The server middleware is based on the Django web-framework. It provides a full set of functionalities for socialmath users to conveniently contribute to the mathematical knowledge library and to build their own graph:

- *Nodes and Dependencies*
  The most fundamental functionality in the socialmath web framework is the node-dependency system. In the framework, the nodes contain information about its theme, content, type, publication and modification dates. We used relationship structures to store node-node relationship, as well as node-to-other-object relationships.

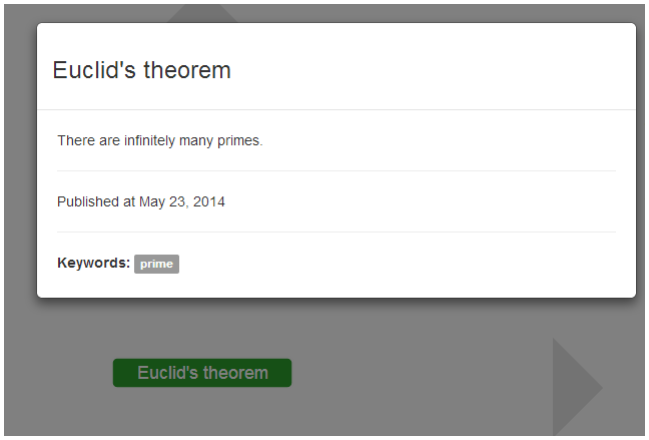

Figure 1: One node in social math knowledge graph

Dependencies are classified into three types:

1. *Proof-to-Theorem*: Lemma dependency. This relationship exists when a proof of one theorem cites another theorem in the proving process.
2. *Theorem-to-Proof*: Prove dependency. This relationship exists when a proof proves a theorem.

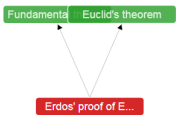

Figure 2: Erdos' prove of Euclid's theorem exhibits prove relationship to Euclid's theorem, and exhibits lemma relationship to Fundamental Theorem of Arithmetic

3. *Theorem-to-Theorem*: High-Relevancy dependency. This relationship exists when a theorem is the special case or a broader case of another, or they have strong logical relationship with each other. This also includes the relationship between theorems and definitions, which we also model as theorems in our system (only that defs can't have proofs).

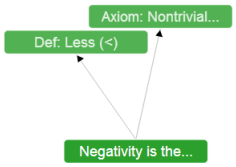

Figure 3: Theorem-theorem and theorem-definition relationship

- *Keywords and Search*
  Keywords are words that generalize a node's content and branch of study. They can be added upon adding the node, or can be added when editing an existing node. Node-keyword relationships are stored in a separate object named KWMap.

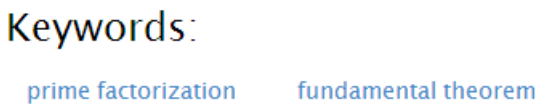

Figure 4: Keywords in the detail-view of a node

The search functionality is supported by keyword mapping. Search can be used in two ways: you can either enter text in the search bar displayed at the index page, or you can click on the link of an existing keyword.

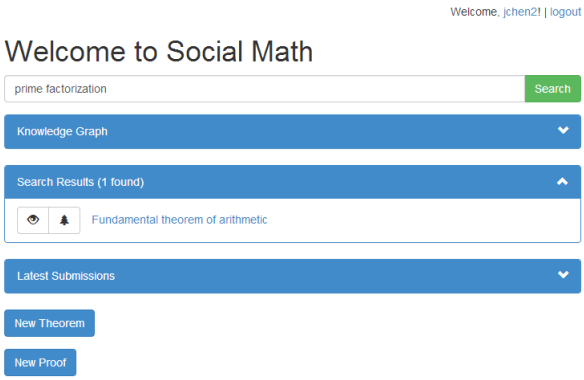

Figure 5: Search result for the keyword "prime factorization"

- *Submission and Editing*
  Adding a new theorem or proof can be done in two ways: Click on the button at the bottom of the index page,

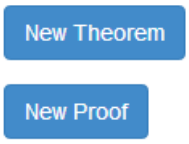

Figure 6: Add theorem/proof button at bottom of index page

or use the "add child" tool on the graph.

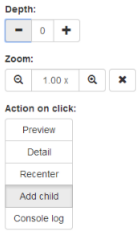

Figure 7: The "add child" tool

To edit an existing theorem, the user would have to go to the detail page and click on the "edit" button, where the user will see an interface similar to the add-node page, only that existing information has been filled in.

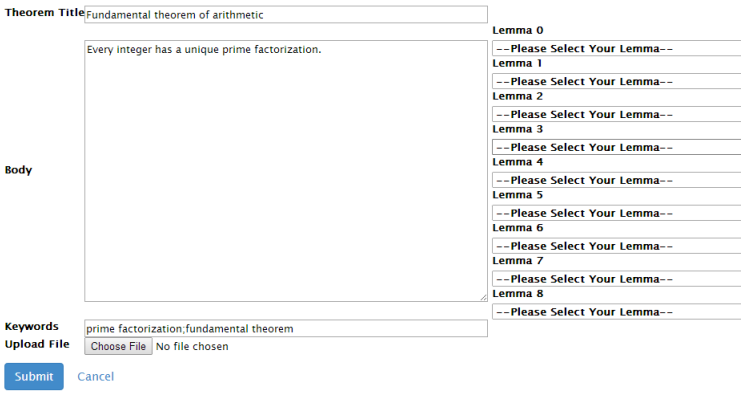

Figure 8: Editing an existing node

- *Users and Profile*
  Socialmath doesn't require a user to login to view its content, but there are additional functionalities for logged in users.

Figure 9: User registration page

Once a user is logged in, every edition/addition activity will be recorded as an "event" in the database. They will show up both on the detail page of the node being edited, and on your own profile page as well.

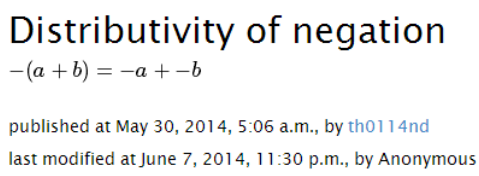

Figure 10: Edition history of a node showing the initial author and the last modifier

Follow is a functionality provided for registered users. The user may click on the "follow" button of any given node, and the server will record it as an event of "follow" type.

Figure 11: The follow button

To view a user's own profile, the user can always click on his own username on the top navigation bar. Right now, there is now search-for-user functionality, so the only entrance for viewing other users' profiles is in the edition history of nodes.

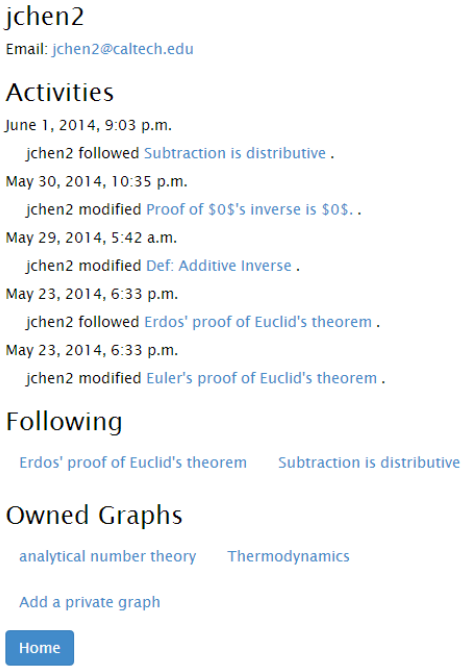

Figure 12: The profile page of Jianchi Chen, consist of basic info, activities, following-nodes, and private graphs he owns

- *Private Graphs*
  In addition to one public graph visible to everyone, socialmath also implemented a functionality called *private graph* so that each registered user can create and edit his own graph. Upon creation, the user may choose to make the graph public or only disclosed to authorized users.

Figure 13: Creating a private graph

The created private graph will have a individual index page, and every node added to the private graph will not show up on the public graph.

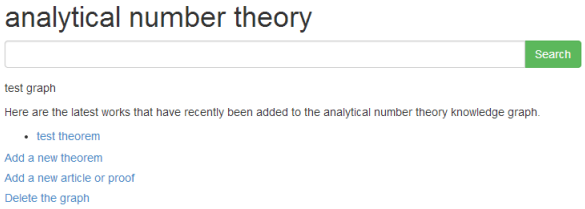

Figure 14: The index page of a private graph, with a node called "test theorem" added.

All contents associated with a private graph will be visible to the owner and invited user only. An attempt to view private graph contents when you're not invited will result in a 401 unauthorized error.

7. FUTURE PLANS

Although we think that the current socialmath is ready for public usage, there are several additional goals that we plan to achieve with socialmath in the future in order to make it a better product:

1. *Protection against malicious activity*
   Right now, socialmath has no protection against a user who simply wants to delete everything on the knowledge graph, or change everything to spam. We plan to learn from how wikipedia prevents users from malicious editions, by having a more comprehensive edition history and revert-edition functionality, as well as user-penalty system.

2. *Correctness system*
   Also, the website currently has no way of detecting correctness of a proof or a theorem. Multiple alternatives can be used in this regard: an user upvote system, or a user "questioning" system where users can request verification on statements, etc.

3. *A more complete private graph system*
   An ideal model for a private graph will be a github repository, where users can collaborate and utilize multiple edition tools. Also, we will work to implement full visualization of the private graph so that it is the same as the global graph.

4. *File uploading*
   Right now, socialmath only supports in-box writing. Although LaTeX formulas are supported, it would be more desirable if we can add pdf-uploading and displaying functionality.

5. *Lemma and keyword detection*
   A more advanced version of the socialmath node submission system plans to have lemma and keyword autodetection, where the system scans for heuristics in the content.

6. *Better user interaction system*
   We plan to enhance the "social" side of socialmath. Search-for-user, follow-user, invite/messaging systems are all planned.