Crosshatch: A Web Application for Crossword Collaboration

Charlie Gunn and Zach Minot

September 23, 2021

1 Motivation and Vision

When doing a crossword puzzle from a physical newspaper, it is easy for a few people to gather around and work on solving it together. This is because all of the clues are visible at once, so each person can scan around for clues they know the answer to. However, this technique for crossword collaboration does not extend well into the digital world. Most major websites that host interactive crosswords only show a small subset of clues (e.g. The LA Times [3]) at a time, forcing all collaborators to focus on the same small portion of the clues. In the authors' personal experience, this is an inferior way to collaborate on a crossword: a lot of the fun of joint crosswords is rapidly going back and forth in different areas of the grid, calling out newly discovered answers.

To solve this problem, this paper proposes Crosshatch: a web application for crossword collaboration. Crosshatch will allow multiple participants to work on the *same* crossword from their own personal devices (in a way similar to document-sharing services, but for crosswords instead of general documents). To make finding and solving crosswords easy, Crosshatch will provide access to multiple free daily crosswords from around the US (LA Times, Wall Street Journal, USA Today, etc.) and feature a convenient user interface for solving.

The ultimate aim of Crosshatch is to provide a crossword collaboration experience that is superior to all other online options.

1.1 Related Work

Crosshatch's core technological design relies on a specific piece of real-time systems technology: collaborative editing. Collaborative real-time editors are applications that enable seemingly instantaneous, simultaneous editing of the same digital document by different users. Possibly the most popular of collaborative editing tools is Google Docs, a collaborative word document tool similar to Microsoft Office Word. The complex problem of creating a network structure to facilitate this type of application has been solved with centralized (Client/Server) and P2P models. Furthermore, different protocols, such as push-based or semi-synchronous, can be implemented based on the type of information being edited, network model constructed, and deadlines for edits to cascade through the network. Thankfully, our project is much simpler than collaboration on an Open Office XML (.docx) document; Crosshatch requires only single character input at designated points with no formatting or large edits at once. This allows experimentation with the methods discussed above and switch designs at relatively low stakes on user-facing application requirements.
`https://dl.acm.org/doi/pdf/10.1145/2933057.2933090`
`https://hal.inria.fr/file/index/docid/345911/filename/main.pdf`
`https://dl.acm.org/doi/pdf/10.1145/1180875.1180916`
`https://link.springer.com/chapter/10.1007/978-94-011-2094-4_15`
`https://www.google.com/docs/about/`

2 Proposed Work

Expanding on the overarching vision expressed in Section 1, the aim of the Crosshatch web application is to provide the following specific features:

- A convenient UI for interacting with and solving a crossword puzzle in a web browser
- The ability for multiple people (at least three) to interact with the same crossword puzzle at the same time from different devices
- Daily crossword ingestion from at least one popular free source (LA Times, Wall Street Journal, USA Today, etc.)

2.1 System Architecture

In order to execute on the goals of this project, there are multiple important architecture and design decisions that must be made. Since Crosshatch is a web application, an immediate crossroads is the choice of web libraries. Vue and FastAPI have been chosen as initial options for frontend and backend libraries respectively, but are subject to change. Sections 2.1.1 and 2.1.2 explain the reasoning behind these choices. More fundamental than these choices though, is the choice of architecture for the collaboration feature. There are two primary options for this, which are explored in Section 2.1.3. Finally, there is a discussion of crossword data ingestion in Section 2.1.4.

2.1.1 Vue

Vue [4] is an approachable, versatile, and performant JavaScript framework that is rapidly growing in popularity (it currently has 188k stars on GitHub [2]). The authors have some experience with other frameworks like React, Django, and Angular, but want to try something new and interesting. Vue is both powerful and simple to get started with, so there are not any expected obstacles relating to it. However, if it does end up being a problem, it is possible to pivot to something slightly more familiar like React.

2.1.2 FastAPI

FastAPI [1] is a fast, easy, intuitive, and robust web framework written in Python. One of the authors has used it before, and found it quite nice.

2.1.3 Collaboration

The main design decision that goes into executing the collaboration feature of Crosshatch is *where to store the ground truth crossword*. It can either be stored on the client-side of whichever user is hosting the shared session, or it can be stored on the server-side in a simple database.

In the first option (client-side storage), the backend implementation would be simpler since much less information would be flowing through it. However, the second option (server-side storage) leaves more room for stretch features, like the ability to "save and quit" a crossword to resume it later. Both options will be explored and researched in the first two weeks of the project timeline (see Section 3), and a definitive choice will be provided in the first follow-up.

2.1.4 Data Ingestion

Ideally, users would be able to cooperatively work on crosswords from their favorite newspapers. Unfortunately, the ingestion of crosswords from popular newspapers is likely the task that has the most complexity. Initial research did not uncover any simple means, such as an API, that could be utilized to get the information in a universally digestable manner. Newspaper websites often have a specific embedded 'online game' that makes collecting the crossword data difficult or impossible. There are off-brand sites that have the solutions and clues in a textual form, but the graph data is likely not accessible and scraping the data would be the only option. Further options for data ingestion will be explored in the first two weeks of the project timeline (see Section 3).

Another design decision that comes into play is when, if, and how to fetch and store the crosswords from the popular newspapers. An option is to store all crosswords once and persist them into a central database. This would ensure that crosswords are only retrieved once from each source and would provide backwards compatability and continued usage of the application in the event that crossword retrieval breaks. Another option is to retrieve the crosswords on a request basis. The benefit of this is that there would be no central storage overhead on the server-side; however, it would also lead to creating a request for crossword ingestion every time someone wants to load a crossword. Each of these options might also lead to a different 'save and quit' implementations if a server-side storage architecture is chosen.

If necessary, a final resort would be to generate custom crosswords or have a set list instead of grabbing current ones from the newspapers. This is an non-ideal scenario, but it is a fallback plan just in case the data ingestion completely fails.

2.2 Deliverables

There are three concrete deliverables for the proposed project:

- The entire source code for Crosshatch
- A final report explaining the design and implementation of Crosshatch
- A comprehensive demo of the final working web application

3 Timeline

Week	Task
09-26	Research and experimentation with methods to connect clients into a shared crossword session
10-03	Further research of above and the ability to ingest daily crosswords from popular sources
10-10	Establish foundational source code and UI
10-17	Continue to establish foundational source code and UI
10-24	Set up crossword ingestion into the application
10-31	Create working connections between clients and any necessary application entities (e.g., the ground truth crossword)
11-07	Test and evaluate different methods of rectifying conflicting edits
11-14	Investigate scalability of collaboration (i.e. 5+ people working on one crossword)
11-21	Finalize source code and perform bugfixes
11-28	Create presentation, demonstration, and final report

3.1 Stretch Goals and Future Work

The timeline leaves room for design switches and experimentation, disastrous bugfixes, and sudden life events that may prevent ample progess. However, in the case that there is extra time at the end of the semester, here are some stretch goals that would be interesting to complete or try.

- Reinforcing scalability and testing extreme cases of collaborative editing (50+ users)
- Additional settings and features, such as optional incorrect character detection or accessability settings
- Complete logging and system-wide evaluation of useful statistics such as response time over user load
- Extending the application to other single input entry games, such as Sudoku or Picross.

Furthermore, the project will be documented and released as open-source at the end of the semester so others may utilize its techniques and findings.

References

[1] FastAPI. FastAPI. https://fastapi.tiangolo.com/. Accessed: 2021-09-23.

[2] GitHub. vuejs/vue. https://github.com/vuejs/vue. Accessed: 2021-09-23.

[3] LA Times. Daily Crossword. https://www.latimes.com/games/daily-crossword. Accessed: 2021-09-23.

[4] Vue. The Progressive Javascript Framework. https://v3.vuejs.org. Accessed: 2021-09-23.