Collaboration in Design Optimization

by

Eunice Lin

B.S., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2017

© Massachusetts Institute of Technology 2017. All rights reserved.

Author

Department of Electrical Engineering and Computer Science

February 3, 2017

Certified by

Professor Maria Yang
Director, MIT Ideation Lab
Associate Professor, Department of Mechanical Engineering
Thesis Supervisor

Accepted by

Professor Christopher Terman
Chairman, Masters of Engineering Thesis Committee
Collaboration in Design Optimization

by

Eunice Lin

Submitted to the Department of Electrical Engineering and Computer Science
on February 3, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Tacit is a software tool for designing and solving 2D truss structures. The Tacit web application expands these features to allow users to design truss structures with integrated analysis and optimization tools. Tacit was intended for single user interaction, but much of engineering and design work is team-based and collaborative. To examine the effects of collaboration on design optimization problems, it is crucial for users to be able to easily share and view the structure designs of their teammate. This thesis presents a new version of Tacit with these collaboration features along with a new method for assessing the similarity of pairs of structures created by collaborators, and the results and analysis of a user study conducted where participants collaborated on design optimization problems with these new functionalities. Results showed that collaboration and the collaborative software tools were both effective in improving performance. The new software functionalities improved the efficiency of collaboration but unintentionally reduced the amount of physical collaboration.

Thesis Supervisor: Professor Maria Yang
Title: Director, MIT Ideation Lab
Associate Professor, Department of Mechanical Engineering
Acknowledgments

It has been a great experience working on this project, and it would not have been possible without the help of Professor Maria Yang, Edward Burnell, and MIT Ideation Lab.

Professor Maria Yang has been incredibly helpful throughout the process, from supporting my research interest in design, to helping me with the direction of this project. I am incredibly lucky to have such a kind and dedicated supervisor for my thesis.

I would like to thank Edward Burnell for all his previous work on Tacit, answering technical questions about the codebase, help with brainstorming ideas and running user studies for this phase of the project. I am always inspired and encouraged by his enthusiasm about the project. I would also like to thank all the other developers who have contributed to the software.

I would like to thank MIT Ideation Lab for supporting this project and participating in the pilot study.

I would like to thank Professor Caitlin Mueller for her advice on the truss similarity algorithm.

I would like to thank all the MIT students who took time out of their busy schedules and participated in the user study.

Finally, I would like to thank my friends and family who have supported and believed in me throughout this journey.
## Contents

### 1 Introduction

1.1 Motivation ........................................... 17
1.2 Goals .................................................. 18
1.3 Hypothesis ............................................ 18
  1.3.1 Effect of Collaboration on Performance ............... 18
  1.3.2 Design Tools for Collaboration ....................... 18
  1.3.3 Common Patterns in Collaboration ................... 19
1.4 Thesis Summary ....................................... 19

### 2 Literature Review

2.1 Collaboration in System Design ....................... 21
2.2 Effects of Team Structure on Performance ............. 22

### 3 Tacit Background

3.1 Related Design Software ............................. 23
3.2 Overview .............................................. 24
3.3 Data Models ........................................... 25
  3.3.1 Structures ......................................... 25
  3.3.2 Events ............................................ 25
  3.3.3 Nodes ............................................. 25
  3.3.4 Beams ............................................. 26
3.4 Features ............................................... 26
  3.4.1 Current Score and Time ............................ 26
List of Figures

3-1  This depicts the user’s workspace while designing a truss structure using the previous implementation of the optimized analysis tool of Tacit without the integrated collaboration features. This version supports only individual users. ........................................ 24

3-2  The goal, current score, and time remaining displayed in the center of the top panel. .............................................................. 26

3-3  Example of a user’s history of saved designs. .......................... 27

3-4  Tools selection panel for users to switch between the move, draw, and erase tools. .............................................................. 27

3-5  Undo, redo, and finish button at the top right of the window. ..... 28

3-6  User performance on the road sign problem and the bridge sign problem, using each tool. The colored line indicates the median and the black line indicates the threshold for additional reward. The darker section indicates the 25th to 75th percentile[2]. ............................. 29

4-1  Current implementation of Tacit with collaboration features as seen in the panel at the bottom right of the window. ............... 32

5-1  The road sign problem. The red arrows are the forces acting on the road sign, the dotted brown line is an unsupported beam, and the blue triangles are the supports where the forces should be grounded. .... 36

5-2  A step from the tutorial that participants completed before the problem. 38
6-1 Example structures components for calculating beam similarity metric. In this example, Structure Component 2 and 3 are both replacements of Structure Component 1. Structure Component 2 is an exact replacement and Structure Component 3 is an inexact replacement. 

6-2 Example structures for calculating similarity metrics for additional nodes and beams. 

7-1 Cumulative distribution function that shows the comparison of the performance of individual user scores on the road sign problem using the optimized tool. 

7-2 Individual best scores versus the number of times users loaded their teammate’s structure for users who used the collaboration software tool. The individual best score is the lowest cost for each individual user. 

7-3 Team best scores versus the number of times teammates loaded each other’s for teams who used the collaboration software tool. The team’s best score is the lower cost of two individual costs from each team member. The number of times users loaded a teammate’s structure is the sum of the times each user loaded their partner’s structure. 

7-4 Survey responses from users on how the collaborative load tool changed their approach to the problem. 

7-5 Survey responses from users on which part of the collaborative features was the most influential to their design approach. 

7-6 Team best score vs. structural diversity. 

7-7 Example of similarity metrics between teammates’ structures throughout the problem for the teams with the cheapest structure (best score). 

7-8 Example of similarity metrics between teammates’ structures throughout the problem for the teams with the most expensive structure (worst score). 

12
7-9  Survey results from users on how talking to their teammate changed their approach to the problem. ........................................... 60
7-10 Survey results from users on how often they talked to their teammate. 60
7-11 Survey results from users on when they looked at their partner’s design. 62
7-12 Survey results from users on when they talked to their teammate. . . 62
## List of Tables

6.1 Similarity metrics between example structures. . . . . . . . . . . . . . 47

7.1 Participant demographics . . . . . . . . . . . . . . . . . . . . . . . 51
Chapter 1

Introduction

1.1 Motivation

Historically, the complexity of designs for products have been limited because of limitations in manufacturing technology. However, with the advancements in additive manufacturing in particular, we are now able to manufacture products of high complexity and are instead limited by current design and analysis tools[2]. Thus, it is important for software design tools to incorporate analysis and optimization features to help users fully optimize the design, though such tools currently were created for traditional manufacturing methods, not additive[2].

At the same time, collaboration is becoming ubiquitous in both education and industry, as software technologies now allow us to collaborate and develop in distributed teams[4]. Though there have been studies conducted on the impact of types collaboration and the effectiveness of software collaboration tools, an interesting area that has not been explored is the influences of collaboration on performance specifically in design optimization[4][6].

In this study, we examined the effects of collaboration through software tools for design optimization. Our approach was to conduct a controlled study to compare users who completed design optimization problems without collaboration, with collaboration in person via software collaboration tools, and with collaboration in person without collaboration tools.
1.2 Goals

This study strived to better understand the role and effects of collaboration in design optimization. Design optimization differs from other engineering problems because of its clear design goal and objective method of evaluation. By providing teams with live feedback and updates of team progress, we hope to observe the effect on their performance in design optimization problems. In particular, we hope to investigate the following ideas:

- How does collaboration influence performance on design optimization problems?
- How can we make design tools better for collaboration?
- What are some common patterns in collaboration, if any?

1.3 Hypothesis

1.3.1 Effect of Collaboration on Performance

We anticipated that users who collaborate with a teammate will perform better than those who work alone, since they are able to explore different designs in parallel and share ideas. We also expected that users who collaborate with built-in collaboration tools will perform even better since it allows them to more efficiently view, switch between, and build off of each others designs, and thus are able to rapidly explore many prototypes in the early stages before fully implementing one design. As studies have shown, users who explore a larger number of low-fidelity prototypes typically perform better and are more satisfied with their final design[5]. In addition, we believe that the ability for teams to view their teammate’s progress in real time will create an element of friendly competition, which may motivate better team performance.

1.3.2 Design Tools for Collaboration

Collaboration allows teams to explore solutions and prototypes in parallel. We believed that design tools can help users take advantage of this efficiency and experiment
with more designs faster. We also expected design tools to facilitate collaboration by allowing team members to seamlessly integrate their work environments. Users can easily switch between their own designs and their partner’s design. Design tools can also encourage collaboration by providing real-time information about their teammates’ work progress.

1.3.3 Common Patterns in Collaboration

We anticipated to potentially see pairs of users who work together to fall into roles of leader and follower. We believed that in an environment where users collaborating on a design optimization problem do not know each other, people will tend to follow their teammate’s design if it performs well.

1.4 Thesis Summary

Chapter 2 covers a brief literature review of other research that has been conducted on team collaboration in design.

Chapter 3 provides background information on the Tacit software and presents relevant findings from previous user study results.

Chapter 4 discusses the new collaboration features that have been developed for this version of Tacit.

Chapter 5 explains the experimental design of the user study conducted for this research.

Chapter 6 presents the motivation and procedure of a structure similarity algorithm that quantifies the similarity between two truss structures, which was developed to analyze the structures created by participants of the user study.

Chapter 7 presents and analyzes the results from the user study.

Chapter 8 discusses future work that can be done based on this research to further improve and examine the effects of collaboration on design optimization problems.

Chapter 9 outlines the contributions and conclusion of this thesis.
Chapter 2

Literature Review

2.1 Collaboration in System Design

System design is the process of creating modules and components for a system subject to specifications. In one study, teams of college graduates and engineers currently do not perform well in system design projects. They were not well prepared for large-scale engineering projects that require teams to balance a range of design and decision making skills, even when equipped with common modern collaboration tools, such as shared spreadsheets, Google Documents, and Skype instant messaging[1]. The results showed that teams tend to focus on the optimization of individual parts and subsystems rather than the overall design, and do not consider the effects that these modules have at a system level[1]. Collaborative optimization software may not effectively improve performance on complex design problems without the augmentation of more structure or additional feedback to further facilitate the process of making design decisions.

In this study, we elected to focus on simple design optimization problems. By keeping the problem relatively simple, we eliminate the potential problems with collaboration in system design as described in this study, because designs will not be intricate enough to become modular and not dependent on subsystems optimizations.
2.2 Effects of Team Structure on Performance

Social dynamics and team structure are integral components of a team’s performance. According to another study, co-located teams tend to adopt a social oriented approach to team projects whereas distributed teams are more likely to follow a task oriented approach. This research study found that, based on self-evaluation, co-located teams ranked higher in trust and evaluation[4].

If a team member is perceived to be more knowledgeable or experienced at a particular task, their ideas may be weighted more heavily and more easily accepted by the rest of the team. The study observed a likely example of the Abilene Paradox where some ideas are not opposed because the team member was believed to have more knowledge at the particular subject[1].

According to informal interviews conducted during pilot studies of this experiment, participants expressed that knowing their teammate better makes them more comfortable and more likely to communicate and share ideas with team members without fear of judgement. Thus, knowledge of team members’ backgrounds can have a significant impact on team dynamic.

Although team structure can be highly influential on the performance of a team, design optimization problems differ from regular design problems in that there is a clear goal and method of evaluating the performance of each design. Therefore, while qualities like trust and evaluation are extremely important, teams may function well without these traits when working on a design optimization problem. For example, a distributed-located team who has never worked together before may not have a lot of trust their teammates, but if a design scores well, team members may not have a problem trusting the design of their teammates.

In this study, we focused on the collaboration between a pair of participants, who are task-oriented instead of socially-oriented. The participants who worked together were co-located and were randomly paired together based on their availability.
Chapter 3

Tacit Background

A custom collaborative software was created for this study, which built off of an earlier piece of research software called Tacit. This chapter describes the original version of the software.

3.1 Related Design Software

Similar software that aids users in structural design includes traditional single-user CAD and cloud-based Onshape. Both CAD and Onshape are full featured computer aided design software aimed at professionals to support 3D designs. CAD is a local software and to collaborate, users must share the CAD files with each other. Onshape is a cloud-based CAD system that supports live collaboration. Though these types of software are powerful, they are not appropriate for the type of study we want to conduct.

Tacit was built for research purposes to study design behavior. It is a lot simpler than the available commercial products and has features that integrate analysis and optimization into design and allow real-time collaboration between users.
3.2 Overview

Tacit is a 2D software tool that allows individual users to design and evaluate truss structures. It is a web application built primarily with Coffeescript and is hosted and supported by a Firebase backend.

Users can view the score (cost of their structure) as they update it, and can save and load versions of the structures they have built. There are currently three different versions of the tool - manual, automatic, and optimized - designed to investigate how people perform on optimization challenges using the different tools. The manual tool requires the user to click an “analyze” button to see the feedback analysis in a separate panel, the automatic tool provides real-time analysis on the users structure, and the optimized tool automatically adjusts the sizes of the beams in the structure to the optimal size. Figure 3-1 is a screenshot from the single-user version of Tacit.

Figure 3-1: This depicts the user’s workspace while designing a truss structure using the previous implementation of the optimized analysis tool of Tacit without the integrated collaboration features. This version supports only individual users.
3.3 Data Models

Tacit is built primarily with four data models - structures, events, nodes, and beams. These objects are stored in the Firebase database, and this section will describe these data models in detail.

3.3.1 Structures

Structures are created by users throughout the design problem. Every time a structure is created, it is automatically recorded into the database. When the user clicks the “save” button, the structure is saved into the database and added to the user’s history of designs, where it can be loaded later.

Each structure object contains a list of beams and a list of nodes that compose the structure, as well as the count of the number of beams and number of nodes in the structure. It also contains a timestamp of when the structure is created, and the type of tool that it was created with. Structures can be created with the draw, erase, move, or load tool. Each structure also has a weight attribute, which is the cost of the structure.

3.3.2 Events

Events are user actions that are logged throughout the problem. Each event contains a timestamp at which the event occurred and an event type. The type of the event can either be a switch to a new tool (draw, erase, or move), load from saved designs, undo, redo, or save. For the “save” event type, the saved structure is recorded as part of the event along with the event type.

3.3.3 Nodes

Each node has an x, y, and z coordinate that specify the position of the node. Each node also keeps track of a fixed attribute that specifies whether the node is fixed in each dimension, as well as an immovable attribute that records whether the user can
move this node. Since this tool is currently 2D, all nodes are fixed in the z dimension but not fixed in the x and y dimensions, with the exception of the nodes from the problem, which are fixed in all dimensions. Nodes also contain a force attribute, which specifies the amount of force acting on the node in each dimension.

3.3.4 Beams

Each beam has a starting x, y, and z coordinate and an ending x, y, and z coordinate. However, the direction of the beam does not matter in the structure. Each beam also has a size, which is the thickness of the beam. In the optimized tool, the tool used in this study, the size is set automatically to the optimal size, which is the minimum size that still creates a stable structure. However, the other tools allow users to adjust the size of the beams. Beams also have an “immovable” attribute that specifies whether the user can move the beam. All beams are movable, except the beams from the problem.

3.4 Features

3.4.1 Current Score and Time

In the top center of the window, the goal cost, the score of the user’s current design, and the time remaining for the problem are displayed, as seen in Figure 3-2. The goal of the problem is to minimize the cost of the structure, and the goal cost is a maximum threshold, such that all users who reach the goal will receive an additional reward. The user’s current score is the cost of their current structure. This is updated every time the user updates their structure.

Figure 3-2: The goal, current score, and time remaining displayed in the center of the top panel.
3.4.2 History

Figure 3-3 depicts the panel at the top left that shows all of the user’s previously saved designs, and their respective scores. The user can click on any of the designs to load it into their current workspace and then continue to edit it.

![Figure 3-3: Example of a user’s history of saved designs.](image)

3.4.3 Tools

On the left panel, shown in Figure 3-4, users can switch between the move, draw, and erase tools. There is also a zoom button that allows the user to center their screen on the structure if they accidentally zoom in or out too far.

![Figure 3-4: Tools selection panel for users to switch between the move, draw, and erase tools.](image)
3.4.4 Undo, Redo, and Finish

Users can undo or redo changes to their structure using the buttons in Figure 3-5. To finish the problem before the time is up, users can click the finish button at the top right of the window, which will end the problem and take them to the post problem survey.

![Figure 3-5: Undo, redo, and finish button at the top right of the window.]

3.5 Initial Individual User Study Results

A previous study conducted with Tacit experimented with combinations of different versions of the tools with two different structural design problems - a road sign problem and a bridge problem[2]. Participants solved each problem using either the manual tool, the automatic tool, or the optimized tool. The manual tool performs the analysis of the structure after the user completes the design. The automatic combines analysis with design, leaving the optimization to the user. The optimized tool integrated analysis and optimization into the design tool. The results showed that users using the optimized tool were able to achieve higher scores (lower cost structures) in less time, in comparison to users who used the automatic tool, who performed better than users who used the manual tool[2].

For this collaboration study, we chose to have users complete the road sign problem using the optimized tool. As we can see from Figure 3-6, the top 75th percentile of users completing the road sign problem span a much larger region, whereas the range for the top 75th percentile of users completing the bridge problem spanned a much smaller range of scores. Thus, we chose the road sign problem for this user study
because it is more effective in differentiating the differences in performances among users.

Figure 3-6: User performance on the road sign problem and the bridge sign problem, using each tool. The colored line indicates the median and the black line indicates the threshold for additional reward. The darker section indicates the 25th to 75th percentile[2].
Chapter 4

Tacit Collaboration Features

4.1 Goals

We added a few new collaboration features to Tacit that allow users more easily design with a teammate in real time, and this new software is called “Tacit Collaboration.” A screenshot of the software including these new functionalities is shown in Figure 4-1. These new features aimed to:

- Allow users to seamlessly collaborate closely with their teammate regardless of their physical location
- Allow teams to explore a larger solution space more efficiently within the limited timeframe

4.2 Database Design

Tacit uses Firebase which stores all necessary data in JSON format. Firebase allows the application to synchronize data across multiple clients accessible via a web API, so that team members can have access to each other’s data in real time. Previously, Tacit was a front-end only application, since it was not necessary for the data collected to be readily accessible by a different client, so the data was simply downloaded at
the end of the session. However, in order for multiple users to work together, we need to be able to store and retrieve data in real time between applications.

In the Firebase database, the application creates a new entry for each study, which contains an entry for each of the two user. Currently, we support two users per session, since each user will be working with exactly one other team member. However, this database structure can easily be scaled to support more than two clients per session by adding another client with a unique ID.

Each client has a separate JSON entry for each “problem” that they complete. For this experiment, each participant completes the tutorial and then the road sign problem, so each user has two entries. This is also easily scalable for any number of problems that the user completes.

For each problem, we store three entries - metadata, events, and structures. The metadata entry contains information about which problem the user is designing for (road sign or bridge), and the tool that they used (manual, automatic, or optimized). In this study, all participants designed for the road sign problem using the optimized tool. The events entry contains a list of user action events that are recorded during each problem session. The structures entry contains a list of structures that were created during the problem session.


4.3 Live Preview and Score

The live preview allows users to see their teammate’s design progress and score updates in real time. As seen in Figure 4-1, there is a panel on the bottom right of the user’s workspace. At the top of the panel, it displays the current score (cost) of the teammate’s structure. In the middle, the preview window shows the easel that contains the current state of the structure in the teammate’s workspace, and any updates will be immediately visible to the user.

To provide a live preview of a user’s teammate’s design, we attach an event listener to the structures JSON object entry of the teammate’s entry in Firebase. This event listener will be triggered every time a new structure is added to this list. Since every action will create a structure, we update the preview window by pulling and redrawing the latest structure in the list every time the event listener is triggered. Thus, the preview window will always show the current design in the teammate’s workspace.

Since each structure has a “weight” attribute that is the cost of the structure, the current cost of a user’s teammate’s current structure can easily be updated by accessing this attribute of the displayed structure.

4.4 Load Teammate Designs

Users can load their teammate’s current design or any previously saved designs and can modify and build off of those designs. This allows teams to rapidly try various prototypes in parallel. As seen in Figure 4-1, under the live preview window in the bottom right panel, users can see all the previously saved versions of their teammate’s design.

In order to support loading designs from teammates, we need to store the entire structure on every save. Thus, every time a user clicks the “save” button, the current structure is also stored into the database so that the user’s teammate can use this data to update the preview.

When users save their designs, in addition to appending it to their own history
of designs, we also update the teammate’s preview of the user’s history. This allows a user’s teammate to see that the user has saved a new design. We do this by also calling method to append a new easel to the history of versions for the teammate. In order to distinguish between the two applications, each user has an ID - either 0 or 1, and every time an update is made, we can make the appropriate updates to the client with the other ID.

4.5 Instant Messaging

We had considered adding a instant messaging feature to the application to allow users to talk to their teammate during the problem. However, since the study was only conducted on co-located teams, we decided that this was not a necessary feature right now. Users will be able to talk to their teammate in person as they will be physically in the same room throughout the experiment, and it may be unnatural for people to communicate through instant message when they are located in the same physical space. Incorporating the chatbox into the user interface will take up additional space on the screen, which reduces the available workspace.
Chapter 5

Experimental Design

In this experiment, we looked at the results of users completing the road sign design problem using the optimized design tool. Based on previous user studies on Tacit, we chose to conduct the study using the road sign design problem. Figure 5-1 shows the road sign problem that participants were asked to complete.

We compared three conditions: users who did not collaborate with a teammate, users who collaborated with a co-located teammate in person without any built-in collaboration tools, and users who collaborate with a co-located teammate using the new version of Tacit with built-in collaboration tools.

5.1 Group 1: No Collaboration

As the control for this study, we used results of users who completed the road sign problem using the optimized tool from the previous user study conducted with the previous implementation of Tacit. The previous implementation of Tacit does not have any built-in collaboration features. Users in this experiment group completed the design problem on their own, without any collaboration. All other experimental variables, such as the tutorial, time limit for the problem, and prize structure, were kept constant for the other groups of this experiment.
Figure 5-1: The road sign problem. The red arrows are the forces acting on the road sign, the dotted brown line is an unsupported beam, and the blue triangles are the supports where the forces should be grounded.

5.2 Group 2: Collaboration with Collaboration Software

The second group of users completed the design optimization problem with a version of Tacit with collaboration functionalities, and worked with a co-located teammate. The participants were encouraged to talk to each other and share ideas with their teammate as if they were someone they have been working with for a while. This group of users used the version of Tacit described in the next section of this paper.

5.3 Group 3: Collaboration without Collaboration Software

The third group of users completed the design project using the previous implementation of Tacit, which does not include any built-in collaboration features. However,
they worked together in the same space and collaborated in-person with a teammate, and were also encouraged to communicate with their teammate. Team members are allowed to look at each other’s computers. The goal of this group of users was for us to see whether people performed better simply by working with a partner, or whether the software tools provided were actually able to further enhance performance.

5.4 Compensation and Prize Structure

All participants received a 15 dollar Amazon gift card for participating in the study. Users who achieve the goal score for the problem received an additional 5 dollars. The team with the highest score each received a prize of an additional 50 dollars. This prize structure was designed to motivate participants to build structure with the lowest score possible. They are first presented with a reachable goal that motivates them to create a slightly optimized stable structure. Once they have a baseline structure, they have the prize as a motivation to optimize their structure to be as cheap as possible. The prize structure also encourages participants to work with their partner to further optimize the cost of their structure.

5.5 Experimental Method

We recruited 44 undergraduate students from MIT for this user study. We did not require any previous experience with structural design or engineering, because the previous user study showed no correlation between relevant coursework and performance on these design problems. Participants completed the design problems in pairs. 9 pairs collaborated with the previous implementation of Tacit that did not have built-in collaboration features. 13 pairs collaborated with the current implementation of Tacit, which included the collaboration tools.

In both groups, pairs of participants worked in a room where they were each set up with a desk with a computer, keyboard, and mouse. The pairs who did not have the collaboration software sat next to each other, such that they could see each other’s
computer screens. The pairs who did have the collaboration software sat facing each other, such that they could not see each other’s computer screens. Both groups were sat close enough that they can easily talk to each other and see each other.

The pairs of collaborators did not know each other before the experiment, and were randomly paired based on the availability they provided. In a few cases, participants have coincidentally been matched with someone they have seen before, but none of the participants knew each other well or have worked together before.

Each session lasted approximately 30 to 40 minutes, beginning with a brief introduction explaining the compensation structure and set up of their work station. Each participant individually completed a short quiz designed to provide background knowledge on truss structures, and then went through a computer-aided tutorial of the software, as shown in Figure 5-2. The tutorial explained the basics of the tool and the participants had a chance to work through some examples.

![Figure 5-2: A step from the tutorial that participants completed before the problem.](image)

After both participants have completed the tutorial, they began the problem together. They are encouraged to work together and talk to each other throughout the problem. There were no restrictions on how they could collaborate or communicate with each other. Following the problem, the participants each completed a survey individually. The survey asked about their experience during the problem and their thoughts on the tool they used. There were also questions that asked about how and when they collaborated with their partner, and how collaborating influenced the de-
sign approach. It also covered basic demographics information and their background knowledge in structural design prior to the problem.
Chapter 6

Structure Similarity Algorithm

6.1 Motivation

To measure the effect that collaboration has on the performance of teams, it is essential to quantify the type and amount of collaboration between teammates. Originally, we wanted to measure the amount of collaboration via software tools by logging the events when users would load designs from their teammate. However, during a pilot study conducted with five pairs of students, we found that this is does not accurately portray the collaboration we observed.

We observed that participants would pick up certain aspects from their teammate’s design by looking at the live preview window and then create and try out those features in their own workspace without actually clicking to load their teammate’s design. We also observed that often times users would end up with similar structures at approximately the same time without actually loading each other’s structures. This kind of collaboration would not be captured by the logs in the software. When we talked to the participants at the end of the pilot study about this observation, participants sometimes said they were trying out certain features of their teammate’s design by looking at the preview window. However, sometimes the participants do not realize they are copying their teammate’s design and so we deduce that the live preview may be unconsciously influencing the participant’s design.

Thus, we believe it is crucial to be able to measure the similarity between two
truss designs. With a metric to measure the similarity between two trusses, we can compare team members’ designs at any given time throughout the problem and track how the two designs converge or diverge throughout the problem. We believe that this may reveal interesting patterns and trends in collaboration.

6.2 Specifications

This algorithm will take two truss structures as input, and return a metric of how similar the two designs are. The more similar the two structures are, the lower their metric will be. Two identical structures will have a metric of 0.

Each structure has a list of nodes and a list of beams that compose of the structure. The two structures will be designs for the same problem, so they are guaranteed to share nodes and beams from the problem. The two structure may be topologically different (differing number of nodes and beams) and geometrically different (the nodes and beams may have different coordinates).

6.3 Algorithm Design

This algorithm quantifies the similarity between two trusses by returning a metric that represents the similarity between two structures. We calculate this metric through three main steps, as follows:

1. **Node matching**: Match as many node as possible between the structures, such that the sum of the Euclidean distances between each pair of mapped nodes is minimized. The metric is the sum of the distances between each pair of mapped nodes.

2. **Beam matching**: Consider all the nodes that are mapped in the node matching phase and calculate the graph edit distance by summing the lengths of beams that would need to be added or removed from one structure to achieve the other structure. The sum of these lengths is the metric for this phase.
3. **Additional nodes and beams:** Consider the nodes that were not matched in the node matching phase and all the beams that connect it to the structure. To calculate the metric, we examine each possible pairing of beams that connected it to the structure for each node. If the pair of beams is a replacement for a single beam in the original structure, we calculate the difference between the two beams and subtract the original length, and if not, we add the length of the additional beams.

The final metric that quantifies the similarity between the two structures is the sum of the metrics from each step.

### 6.3.1 Node Matching

In the node matching phase, we want to create a mapping between the nodes of each structure and calculate a metric that represents the similarity between matched nodes. We are given two structures $s_1$ and $s_2$ with $m$ and $n$ nodes, respectively, where $m$ is less than or equal to $n$. Since structure $s_1$ has fewer than or equal number of nodes as $s_2$, we want to map every node of $s_1$ to a unique node in $s_2$, such that the sum of the Euclidean distances between each pair of mapped nodes is minimized.

**Step 1: Fix Problem Nodes**

Each problem starts with certain nodes and beams that cannot be moved. First, we map each node from the problem in $s_1$ to the corresponding node from the problem in $s_2$. Since $s_1$ and $s_2$ are structure designs for the same problem, they are guaranteed to have the same nodes from the problem. The mapping of the nodes from the problem are fixed regardless of whether a different mapping would minimize the overall metric.

**Step 2: Node Mapping and Metric Initialization**

After all the nodes from the problem are mapped, we sort the list of remaining nodes for each structure by the $x$ coordinate and initialize the mapping to map each node in the node list of $s_1$ to the node in the corresponding index in the node list of $s_2$ in the sorted order. Then, we initialize the node similarity metric by computing the Euclidean distance between each pair of mapped nodes. We sort the list of nodes by
their $x$ coordinates first in order to reduce the number of modifications we need to make to the mapping.

**Step 3: Node Mapping and Metric Optimization**

To find the optimal node mapping, we consider each of the possible pairings between a node in $s_1$ and a node in $s_2$. If switching the pairings reduces the overall metric, then we switch the mapping and update the overall metric accordingly. Thus, after we have examined each pairing, we will have arrived at a mapping between the nodes of $s_1$ and the nodes of $s_2$ such that the sum of the Euclidean distances between every pair of mapped node is minimized and every node in $s_1$ is mapped to a different node in $s_2$.

**6.3.2 Beam Matching**

For the beam matching phase, we have access to the node mapping from the node matching phase and we want to calculate similarity between the beams of the structure. We consider the $m$ pairs of nodes that are matched in the previous phase and calculate the beam similarity metric for these nodes only. Thus, we will be disregarding all the nodes from $s_2$ that are not mapped and all the beams that start or end at those unmapped nodes. The beam similarity metric is the graph edit distance between the two structures, where the two possible operations are to add or remove beams, and the cost of each operation is the length of the beam that is added or removed[3].

**Step 4: Beam Metric Calculation**

We calculate beam similarity metric by calculating the graph edit distance between $s_1$ and $s_2$. We check each of the possible pairings of the $m$ nodes in $s_1$ and the corresponding pair in $s_2$ according to the node mappings made in the previous phase of the algorithm. If a pair of node is connected by a beam in $s_1$ but not $s_2$ or vice versa, then we need to add or remove the beam. The cost to add or remove that beam the length of that beam in $s_1$. The sum of the costs of all operations to make two structures identical is the beam similarity metric. Thus, a structure that needs an addition or removal of a longer beam to transform to the other structure will have
a higher metric than a structure that needs a shorter beam.

### 6.3.3 Additional Nodes and Beams

In the final phase of this algorithm, we take into account the additional nodes and beam from $s_2$ that were not mapped in the node matching phase. Therefore, if $s_1$ and $s_2$ have the same number of nodes, then all the beams would also have been considered in the beam matching phase, so the additional metric here would be 0.

![Figure 6-1: Example structures components for calculating beam similarity metric.](image)

The idea is that if a beam $b_1$ in $s_1$ can be replaced with two beams $b_2$ and $b_3$ in $s_2$ that compose of $b_1$, we want quantify the difference between the $b_2$ and $b_3$ together versus $b_1$. That difference would be added to the similarity metric, instead of the graph edit distance cost, which would add the removal of $b_1$ and the addition of $b_2$ and $b_3$ to the similarity metric. Thus, two beams in $s_2$ that exactly make up a beam in $s_1$ would be considered identical and have a metric of 0, even though essentially a beam in $s_1$ is split by a node.
Step 5: Determining Replacement Beams

To calculate the additional similarity metric, we want to determine which beams are replacement beams. A pair of beams in $s_2$ “replaces” a beam in $s_1$, if the two nodes connected to the unmapped node is directly connected by one beam in $s_1$ but not in $s_2$. As seen in Figure 6-1, Beam 1 and Beam 2 in Structure Component 2 is an exact replacement for Beam 1 in Structure Component 1. In Structure Component 3, Beam 1 and Beam 2 is an inexact replacement for Beam 1 in Structure Component 1.

To identify replacement beams, we perform the following steps for each unmapped node in $s_2$:

- If it is connected to exactly one beam, it is not a “replacement” beam.
- If it is connected to more than one beam, check all possible pairings of the
Table 6.1: Similarity metrics between example structures.

<table>
<thead>
<tr>
<th>Structure</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>34.816</td>
<td>12.475</td>
<td>54.378</td>
<td>124.283</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>34.816</td>
<td>12.5</td>
<td>47.316</td>
<td>125.825</td>
</tr>
<tr>
<td>3</td>
<td>34.816</td>
<td>34.816</td>
<td>0</td>
<td>47.316</td>
<td>12.5</td>
<td>91.009</td>
</tr>
<tr>
<td>4</td>
<td>12.475</td>
<td>12.5</td>
<td>47.316</td>
<td>0</td>
<td>41.903</td>
<td>120.413</td>
</tr>
<tr>
<td>5</td>
<td>54.378</td>
<td>47.316</td>
<td>12.5</td>
<td>41.903</td>
<td>0</td>
<td>78.509</td>
</tr>
<tr>
<td>6</td>
<td>124.283</td>
<td>125.825</td>
<td>91.009</td>
<td>120.413</td>
<td>78.509</td>
<td>0</td>
</tr>
</tbody>
</table>

beams.

- For each pairing, if the pair replaces a beam in $s_1$, then the pair of beams in $s_2$ are both replacement beams. The beam in $s_1$ that is replaced is also returned for the additional similarity metric calculation in the next step.

- If the pair of beams in $s_2$ is not replacing a beam in $s_1$, we mark both beams of the pair in $s_2$ as not “replacement” beams.

- If a beam is marked as replacement for one unmapped node but not for another, “replacement” takes priority.

**Step 6: Calculating Additional Metric**

After categorizing each additional beam as either a replacement or non-replacement beam, we want to calculate the additional similarity metric. If a pair of beams is a replacement, then we subtract the length of the beam in $s_1$ that it replaces, since it would have been added to the beam similarity metric during the beam matching phase. By subtracting the length now, we “undo” it and then add the difference between the beam in $s_1$ and the pair of beams in $s_2$. The “difference” is defined as the height of the triangle formed by the start and end node of the beam in $s_1$ and the unmapped node, with the unmapped node as the apex.

In Figure 6-1, the additional similarity metric of Structure Component 1 and 2 is the negative length of Beam 1 in Structure Component 1, so that the overall similarity metric is 0. The additional similarity metric of Structure Component 1 and 3 is the
negative length of Beam 1 in Structure Component 1 plus the height of the triangle formed by Nodes 1, 2, and 3 in Structure Component 3, with Node 3 as the apex, as shown with the red dotted line. Thus, the overall similarity metric is just the length of the red dotted line.

In Figure 6-2, Structure 1 and Structure 2 would be considered identical, even though Structure 2 has an extra node, Node 6. In the beam matching phase, using the graph edit distance method, we would incur a metric of the length of Beam 6 in Structure 1 since we would have to “add” a beam of that length from Structure 2 to obtain Structure 1, when we do not consider the unmapped node, Node 6. However, Beam 6 and Beam 7 in Structure 2 replaces of Beam 6 in Structure 1, so we would subtract the length of Beam 6 in Structure 1 so the overall metric would be 0.

**Step 7: Final Similarity Metric**
The final similarity metric between two truss structure is the sum of the node similarity metric, beam similarity metric, and the additional similarity metric.

### 6.4 Alternative Solutions
We have also considered alternative solutions, such as image recognition and visual ratings. An example of a simple image recognition solution is to recreate a black and white image of each structure and then compare the pixel differences between the two images. However, this is not very effective since two beams that differ by a small angle would only share the pixels near their intersection point, resulting in a low similarity metric, even though visually we would consider the two beams to be similar. Since we have all the geometric coordinate information of the nodes and beams of the structure, we believe that we can utilize that information to calculate a better metric using a graph problem approach.

An example of a visual rating solution is to have people rate the similarity of two structures sampled at fixed time intervals based on their visual perception of the structures. However, this solution is not efficient, because users typically produce approximately 100 structures throughout the problem. It is not scalable if we want
to run the experiment with larger groups of users. Also, this solution would be affected by the rater’s subjective opinion of what looks similar to them. A graphical approach would be scalable for larger datasets and can quantify similarity metrics across different structures objectively.
Chapter 7

User Study Results and Analysis

7.1 Effects of Collaboration on Performance

7.1.1 Result Comparisons

Results from this user study showed the general trends as expected from the hypothesis. Users who collaborated with another participant were able to reach lower costs (better score) than users who completed the problem individually. Users who used the collaboration software tools performed better than users who did not have the collaboration software.

Table 7.1 shows the basic demographics of the users who participated in the study. Figure 7-1 shows the performances of the three groups of users on the road sign problem using the optimized tool. Focusing on the 25th to 50th percentile, we can see that the average user performs the best using the software with collaboration tools, followed by users who collaborated without the collaboration tools, and then the users

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Freshman</td>
<td>24</td>
</tr>
<tr>
<td>Sophomore</td>
<td>9</td>
</tr>
<tr>
<td>Junior</td>
<td>4</td>
</tr>
<tr>
<td>Senior</td>
<td>3</td>
</tr>
<tr>
<td>Masters</td>
<td>4</td>
</tr>
<tr>
<td>Female</td>
<td>26</td>
</tr>
<tr>
<td>Male</td>
<td>17</td>
</tr>
<tr>
<td>Other or prefer not to answer</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 7-1: Cumulative distribution function that shows the comparison of the performance of individual user scores on the road sign problem using the optimized tool. who did not collaborate. 20 individuals completed the problem without collaboration, and their performance is depicted by the red cumulative distribution function in Figure 7-1. 9 pairs completed the problem together without the collaboration software, shown in green, and 13 pairs worked together with the collaboration software, shown in blue.

The goal for this problem was to reach a cost of 360 dollars. Participants who reached the goal received an additional 5 dollars. As we can see from 7-1, all the teams who collaborated were able to reach this goal. Teams who used the collaboration software were able to achieve a cost cheaper than 350 dollars.

7.1.2 Collaboration Tool Utilization Analysis

We anticipated that users who collaborated with another user with the collaboration software would perform better because they can explore multiple designs in parallel
and thus efficiently explore a larger solution space.

We found that users who utilized the collaboration tools did perform better. In particular, we looked at the the load feature that allowed users to directly load their partner’s design into their workspace. Based on the results from the user study, users who loaded their teammate’s structure more frequently generally performed better. However, there were also many teams who performed just as well without utilizing the collaboration software.

![Figure 7-2: Individual best scores versus the number of times users loaded their teammate’s structure for users who used the collaboration software tool. The individual best score is the lowest cost for each individual user.](image)

Figure 7-2 shows how each individual user performed versus the number of times they loaded from their teammate’s structure. As we can see, approximately half of the users loaded their teammate’s structure at some point during the problem, and users who loaded from their partner at least four times throughout the problem scored in the upper 50th percentile.

Looking at each team as a whole, we can see that some teams had one teammate who loaded from their partner more than the other, while some team members loaded from their partner more evenly. Despite these different situations, we still see the same
Figure 7-3: Team best scores versus the number of times teammates loaded each other’s for teams who used the collaboration software tool. The team’s best score is the lower cost of two individual costs from each team member. The number of times users loaded a teammate’s structure is the sum of the times each user loaded their partner’s structure.

general trend as in the individual scores. The teams who used the load feature more performed better, but there are also teams who performed well without using the load feature at all. As we can see in Figure 7-3, teams who together loaded each other’s designs over five times throughout the problem scored in the upper 50th percentile.

Although many users did not load their teammate’s structure more than a few times throughout the problem, a majority of participants reported in the post problem survey that they felt like loading their partner’s design changed their approach to the problem, which we can see in Figure 7-4. Figure 7-5 shows that teams who had the collaborative software felt that loading their partner’s design and playing around with was more effective in influencing their design approach than viewing the design in the preview window.

However, we must also note that the sample size from this user study is not large enough to be statistically significant.
7.1.3 Structural Diversity Analysis

As we observed in the pilot study, the utilization of the collaborative load feature does not accurately portray the amount of collaboration we observed. Since the structures for this problem were simple enough, sometimes users would copy their partner’s structure by redrawing it themselves without using the load tool, and sometimes users seem to be unconsciously affected by the preview of their partner’s design. Thus, we are using the structural similarity algorithm discussed in Chapter 6 to further examine the collaboration between team members.

According to our hypothesis, teams who collaborate would perform better because they can explore different design ideas in parallel, and teams who used the collaboration tool would perform even better since they can explore different prototypes more efficiently. Thus, we expected to see a correlation between more diverse structures and lower cost designs.
Figure 7-5: Survey responses from users on which part of the collaborative features was the most influential to their design approach.

Figure 7-6: Team best score vs. structural diversity.
To measure structural diversity, we calculated the similarity metrics using the structural similarity algorithm between each structure and the best structure, and took the average similarity metric for all the structures recorded throughout the problem. Thus, a team that explored structurally different designs would have a higher structural diversity than a team that explored structurally similar designs. A team that explored different designs for a longer period of time will also have a higher structural diversity than a team that explored different designs for a shorter period of time.

From the results of this user study, we can see in Figure 7-6 that teams that explored structures with greater diversity did indeed perform better. Although there are also some teams who performed well despite not exploring diverse structures. We can also see in Figure 7-6 that teams who collaborated using the collaboration tool were able to create more diverse designs.

This is consistent with our hypothesis as teams who explored a larger solution space are more likely to find the optimal solution, but teams who find a good design early on in the process may not feel the need to explore other designs and may instead focus their time on optimizing the current design they have. Thus, the collaboration tools are helpful for exploring more designs efficiently, but are not necessarily useful for teams that want to focus on optimizing a good design.

In addition to comparing the structural similarity between each structure with the best structure, we also looked at the structural similarity between the designs of each pair of teammates throughout the problem. We found that teams that performed well showed several peaks where teammates were exploring very different designs, whereas teams that did not perform well had much more similar designs throughout the problem. For example, Figure 7-7 shows the progress of one team that achieved one of the highest score. This team was able to achieve a stable structure very early on. The teammates explored two very different designs near the beginning of the problem, and then converged to a similar structure near the middle of the problem. From there, the teammates then explored two different designs, building off from the same design, before eventually converging to their final design. In Figure 7-8, we can
Figure 7-7: Example of similarity metrics between teammates' structures throughout the problem for the teams with the cheapest structure (best score).

Figure 7-8: Example of similarity metrics between teammates' structures throughout the problem for the teams with the most expensive structure (worst score).
see that this team took longer to achieve a stable structure and both teammates had relatively similar structures throughout the problem, in comparison to the team in Figure 7-7.

This shows illustrates the significance for teammates to explore structurally different designs to create a more optimized structure. Collaboration allows teams to explore these different designs in parallel, and the collaboration software features allows teams to do this efficiently.

7.2 Design Tools for Collaboration

One of the goals of this study was to improve design tools for collaboration. We anticipated that the addition of the feature that allows teammates to load each other’s designs will increase the efficiency of collaboration, and thus improve performance. We also thought that having a live preview of their teammate’s progress would facilitate more collaboration between team members.

However, in this user study, teams who collaborated without the collaboration software demonstrated more physical collaboration through discussion. Teams that did not have the collaboration tool discussed talked to each other more throughout the problem. By providing the live preview to the users, we unintentionally reduced the physical interaction between teammates.

Figure 7-9 depicts how users felt talking to their partner influenced their approach to the problem. Most users either did not feel strongly or agreed that talking to their partner influenced their design approach. In addition, most users felt that they did not talk to their partner frequently throughout the problem, as seen in Figure 7-10. However, this may be an inaccurate portrayal as “frequent” is a subjective description. For example, one of the most talkative pair throughout the study, did not feel like they talked much with their partner at all.

Some interactions were simple questions like “Can I look at your last [saved] structure?” or “I think your structure has an extra beam here.” In these cases, we assume that teams with the collaboration software tool can simply click on their
Figure 7-9: Survey results from users on how talking to their teammate changed their approach to the problem.

Figure 7-10: Survey results from users on how often they talked to their teammate.

teammate’s last structure or re-load their teammate’s structure to achieve the same structure. Thus, the collaboration software successfully increase the efficiency of these interactions by eliminating them. However, these simple interactions also made team members more comfortable with talking to each other, which is helpful for discussing more complex ideas and strategies.
Teams sometimes discussed strategies to optimized their structure. For example, a participant mentioned to his partner that “your structure mirrored on the other side would be cheaper,” and another participant told his teammate that “I think it is cheaper to make a lot of smaller beams.” Talking about these strategies with their teammates allowed the teams to come up with an approach to the problem together, which is faster than the trial and error method where teammates would load their partner’s structure and play around with it to learn from it. This type of interaction was more common in teams without the collaboration tools, perhaps because they interact more throughout the problem so they are more comfortable talking to each other.

We anticipate that in an actual team environment, team members would already know each other and be comfortable discussing ideas with each other, so that this type of interaction would be more common regardless of the type of tool that they were using. To make design tools better for collaboration, we need the tools to facilitate discussion. We can encourage more discussion and collaboration by including an instant messaging feature for team members to talk to each other. This would be especially useful for teams who are collaborating remotely.

7.3 Common Patterns in Collaboration

We had originally anticipated to see some patterns in the ways team collaborated where team members falls into the roles of leader and follower. However, we were unable to find any common patterns in collaboration. Different teams varied in the amount and times teammates loaded from each other’s structure, and the similarity of team members’ structures to the final design and to each other were varied across teams.

We also looked at when users interacted the most with their teammate. The results showed that most users communicated with their partner evenly throughout the problem, while the teams without the collaboration tool talked to their partner either evenly or more towards the end of the problem. Most teams with the col-
Figure 7-11: Survey results from users on when they looked at their partner’s design.

Figure 7-12: Survey results from users on when they talked to their teammate.

A collaboration tool looked at their partner’s structure throughout the problem, while the teams without the collaboration tool was split between the beginning, end, and evenly throughout the problem.

Conducting further studies with more complex problems and team members who are more comfortable working with each other may reveal more interesting patterns in collaboration.
Chapter 8

Future Work

8.1 Tacit Software

8.1.1 Concurrent Editing

The next step in creating an even more integrated collaboration environment is to allow users to work on the same design. This improves the efficiency of collaboration by eliminating the need for users to load each other’s structures since teammates will be modifying the same structure. Concurrent editing will allow users to collaborate even more closely than the current version of Tacit. It is also more scalable to accommodate teams with more than two users, since it will be able to support a large number of users without too many preview windows cluttering the window. However, this means that teams cannot explore different designs in parallel and removes the motivation that stems from friendly competition among team members. It would be an interesting user study to compare the results of teams using these two versions of Tacit.

8.1.2 Communication Support

Tacit does not currently support communication between teammates to facilitate discussion of ideas and strategies. However, the responses from the post problem survey in this user study indicated that this is an influential component of the design
process. Thus, a possible next step is to include features that encourage communication in teams. For example, incorporating instant messaging or video chat into the software may help teammates talk to each other more frequently throughout the problem.

8.2 Experimental Method

8.2.1 Teammate Familiarity

In this user study, many participants did not collaborate as much with their teammate as much because they did not know their teammate before and so they were more reserved in their communication. A potential modification to the experimental method would be to have teams complete the tutorial collaboratively so they can get to know each other better before tackling the problem together. Another possibility is to have participants sign up for the user study in pairs so they can work with someone they comfortable with.

8.2.2 Increasing Problem Complexity and Time Limit

To keep the experiments consistent with the control group who did not collaborate, we limited users to solving the road sign problem in 12 minutes in this user study. However, we can increase the complexity of the problem and the time limit to better observe patterns in collaboration, since it takes more time for teams to discuss and experiment with their partner’s designs. Some participants expressed in the comments in the post problem survey that they wished they had more time to work on the problem, and that having a time limit affected their design approach since they felt more inclined to just load their partner’s design if it had a lower cost instead of fully exploring their own design.
8.2.3 Exploring Distributed vs. Co-Located Teams

Another interesting area for exploration is the differences between distributed and co-located teams. In this study, all teams were co-located, but with the addition of communication features, such as instant messaging and video chat, teams can work together and communicate regardless of physical location. It would be interesting to see how teams perform with these collaboration software functionalities while being physically in the same location or not.

8.3 Structural Similarity Algorithm

8.3.1 Mirrored Structures

In many cases, a mirrored structure may be considered very similar to the structure itself. For example, in the context of the road sign problem, a structure mirrored across the $y$ axis would be considered similar to the structure itself. It is not exactly similar in this case since the forces in the problem are only from one direction. However, sometimes teams explored both sides of the structure. Thus, the structural similarity algorithm should take into account this fact. One way would be to always calculate the similarity metric between both the structures and their mirrored images, and then scale by an appropriate factor. For example, if users were doing a problem where the forces are symmetric, it may not necessary to scale the metric.

8.3.2 Multiple Beam Replacement

In the additional nodes and beams phase of the structural similarity algorithm, we take into account the fact that two additional beam may “replace” an existing beam in the other structure. However, any number of additional beams may “replace” an existing beam in the original structure. Currently we only consider two additional beams “similar” enough to be treated differently. However, there may be cases where a set of beams can be very similar to a single beam in the original structure and should be treated as a “replacement” in the metric calculation instead of using the
edit distance.

In the case of exact replacements, currently we only consider exact replacements of two beams that replaces one beam. We should modify the algorithm to take into account cases when multiple exactly replaces a single beam. However, exact beam replacement occurs extremely rarely, and is also covered in the tutorial that users should replace a set of exact replacement beams with just a single beam. Thus, this is not currently problematic.
Chapter 9

Contributions and Conclusion

9.1 Contributions

The contributions of this thesis include building a new version of Tacit with collaboration features, a user study that examined the performance of users who collaborated on design optimization problems with and without these new features, and an algorithm that quantifies the similarity between two truss structures in this context.

Tacit now allows users to collaborate with a partner in real time through their web browsers regardless of physical location. Team members using the same Tacit session can preview their partner’s design in real time and see all of their teammate’s saved structures and costs. Clicking on any of these previews will load the teammate’s structure into the user’s workspace where the user can modify and build off of the design.

The user study showed that collaboration improves the performance on design optimization problems and demonstrated the effectiveness of the collaboration software. It also revealed the significance of structural diversity on performance and how the collaboration software tools affected physical collaboration.

The structural similarity algorithm takes into account the geometry and topology of two truss structures and returns a metric that quantifies the similarity between the two designs.

These new features and results from the user study can be used to improve the
live collaboration features in Tacit. It can also be used to further explore the effects of collaboration in teams in design and how to improve software tools for collaboration.

9.2 Conclusion

In this research, we examined the influence of collaboration on performance in design optimization problems, ways to improve design tools for collaboration, and common patterns in collaboration. We found that, as expected from our hypothesis, participants who collaborated performed better than those who worked individually, and the collaborative software further enhanced performance. We created new features that aimed to improve design tools for collaboration by allowing teams to collaborate regardless of physical location and increasing the efficiency in which they can explore different solutions. This successfully improved performance, but unintentionally decrease the amount of physical interaction between the participants. We originally expected pairs to fall into the role of leader and follower, but we did not observe any strong evidence or patterns exhibited by the teams we studied.

The new functionalities presented in this thesis now allow users to design truss structures collaboratively in real time using Tacit. The user study results and structural similarity algorithm provided insight into how to better understand and improve design collaboration tools. It is the hope that these new software features and user study analysis can be used in further research to better understand and improve the effectiveness of collaboration in more complex design problems.
Appendix A

Structure Similarity Algorithm

```python
import operator

def match_nodes(structure1, structure2, problem):
    if len(structure1['nodeList']) > len(structure2['nodeList'])
        temp = structure1
        structure1 = structure2
        structure2 = temp
    mapping1 = []
    mapping2 = []

    s1_nodes = []
    s2_nodes = []
    for node in structure1['nodeList']:
        n = {'x': node['x'], 'y': node['y'], 'z': node['z']}
        s1_nodes.append(n)
    for node in structure2['nodeList']:
        s2_nodes.append(node)
```

n = {"x": node["x"], "y": node["y"], "z": node["z"]}
s2_nodes.append(n)

# nodes from the problem must match up
for node in problem:
    n = {"x": node["x"], "y": node_["y"], "z": node["z"]}
    problem_node_s1 = approximately_in(n, s1_nodes, 0.01)
    problem_node_s2 = approximately_in(n, s2_nodes, 0.01)
    if problem_node_s1 == None or problem_node_s2 == None:
        raise Exception("structure does not contain nodes from
                        problem")
    mapping1.append(problem_node_s1)
    mapping2.append(problem_node_s2)

# sort nodes by x coordinates first
s1_nodes.sort(key=operator.itemgetter("x"))
s2_nodes.sort(key=operator.itemgetter("x"))

for node in s1_nodes:
    if not node in problem and not node in mapping1:
        mapping1.append(node)
for node in s2_nodes:
    if not node in problem and not node in mapping2:
        mapping2.append(node)

metric = get_node_similarity_metric(mapping1, mapping2)
for i in range(len(mapping1)):
    for j in range(len(mapping2)):
        if i < len(problem) or j < len(problem) or i == j:
            continue
        # switch a pair of node patching
        mapping2[i], mapping2[j] = mapping2[j], mapping2[i]
        new_metric = get_node_similarity_metric(mapping1, mapping2)
        if new_metric < metric:
# update metric if new mapping improves it
metric = new_metric
continue
else:
    # switch back if metric doesn’t improve
    mapping2[i], mapping2[j] = mapping2[j], mapping2[i]
    return match_beams(mapping1, structure1, mapping2, structure2, metric)

Given two lists of nodes such that the nodes at the same
index are matched, return a similarity metric for beams,
and a list of beams of the two structures.

We do this by calculating length of beams needed to be
added/removed from structure1 to obtain structure2, averaged
with the length of beams needed to be added/removed from
structure2 to obtain structure1.

def match_beams(node_mapping1, structure1, node_mapping2, structure2, node_metric):
    beams1 = structure1["beamList"]
    beams2 = structure2["beamList"]

    # maps node indices that are connected by a beam
    beam_mapping1 = []
    beam_mapping2 = []
    for beam in beams1:
        # get node indices from structure1
        i = node_mapping1.index({
            "x": beam["start_x"],
        }
        },
"y": beam["start_y"],
"z": beam["start_z"]
})

j = node_mapping1.index({
  "x": beam["end_x"],
  "y": beam["end_y"],
  "z": beam["end_z"]
})

beam_mapping1.append((i,j))

for beam in beams2:
    # get node indices from structure1
    i = node_mapping2.index({
        "x": beam["start_x"],
        "y": beam["start_y"],
        "z": beam["start_z"]
    })

    j = node_mapping2.index({
        "x": beam["end_x"],
        "y": beam["end_y"],
        "z": beam["end_z"]
    })

    beam_mapping2.append((i,j))

beam_metric = 0
num_nodes = len(node_mapping1)
for i in range(num_nodes-1):
    for j in range(i+1, num_nodes):
        mapping1_has_beam = ((i, j) in beam_mapping1) or ((j, i) in beam_mapping1)
        mapping2_has_beam = ((i, j) in beam_mapping2) or ((j, i) in beam_mapping2)
if (mapping1_has_beam and not mapping2_has_beam) or (not mapping2_has_beam and not mapping1_has_beam):
    beam1 = get_euclidean_distance(node_mapping1[i], node_mapping1[j])
    beam_metric += beam1
return unmatched_nodes_beams(node_mapping1, beam_mapping1, beams1, structure1, node_mapping2, beam_mapping2, beams2, structure2, beam_metric + node_metric)

'''
Take into account nodes and beams that were not mapped.

If beams add up to original beams, remove them from metric calculation.

For each unmapped node:
- if connected by one unmapped beam, add beam to metric
- if more than one beam, check each pair of beams
  - if pair of beams is replacing a corresponding beam, add difference to metric
  - if pair of beams is not replacing a corresponding beam, add both beams to metric
  - each beam is added at most once, replacement takes priority
- define replace as if two nodes connected to extra node are connected in structure1 and not connected in structure2.

'''
def unmatched_nodes_beams(node_mapping1, beam_mapping1, beams1, structure1, node_mapping2, beam_mapping2, beams2, structure2, beam_metric + node_metric):
    unmapped_nodes = []

73
unmatched_metric = 0

# no unmatched nodes
if len(node_mapping1) == len(node_mapping2):
    return metric

# look at each unmapped nodes
first_unmapped = len(node_mapping1)
for i in range(first_unmapped, len(node_mapping2)):
    unmapped_node = node_mapping2[i]
    unmapped_nodes.append(node_mapping2[i])

# array containing all unmapped beams
non_replacement_beams = []
replacement_beams = []

# make a 2D array of lists of unmapped beam for each unmapped node
unmapped_beams = [[] for i in range(len(unmapped_nodes))]
for i in range(len(unmapped_nodes)):
    node = unmapped_nodes[i]
    for beam in beams2:
        if (beam["start_x"] == node["x"]) and (beam["start_y"] == node["y"]):
            non_replacement_beams.append(beam)
            unmapped_beams[i].append(beam)
        if (beam["end_x"] == node["x"]) and (beam["end_y"] == node["y"]):
            non_replacement_beams.append(beam)
            unmapped_beams[i].append(beam)

# remove beams that replaces a beam in structure 1
for beam_set in unmapped_beams:
    if len(beam_set) > 1:
        for i in range(len(beam_set)-1):
            for j in range(i+1, len(beam_set)):
                b1 = beam_set[i]
b2 = beam_set[j]
midpoint, beam = is_replacement(b1, b2, structure1, structure2)

if beam != None:
    if b1 in non_replacement_beams:
        non_replacement_beams.remove(b1)
        replacement_beams.append(b1)
    if b2 in non_replacement_beams:
        non_replacement_beams.remove(b2)
        replacement_beams.append(b2)

n1 = {"x": beam["start_x"], "y": beam["start_y"], "z": beam["start_z"]}
n2 = {"x": beam["end_x"], "y": beam["end_y"], "z": beam["end_z"]}

difference = get_difference(midpoint, n1, n2)
unmatched_metric += difference
unmatched_metric -= get_beam_length(beam)

else:
    continue

# add up beams to metric
for beam in non_replacement_beams:
    unmatched_metric += get_beam_length(beam)
return unmatched_metric + metric

# Helper Methods#

Given a beam, return length of the beam.
def get_beam_length(beam):

Given two nodes, calculates the Euclidean distance between them.

Given lists of nodes ordered by mapping, return the sum of the Euclidean distance between each pair of nodes.
Given a node and a list of beams, returns the beam it splits if it splits a beam, otherwise return None.

def splits_beam(node, beams):
    intersecting_beams = []
    for beam in beams:
        dx_node = node["x"] - beam["start_x"]
        dy_node = node["y"] - beam["start_y"]

        dx_beam = beam["end_x"] - beam["start_x"]
        dy_beam = beam["end_y"] - beam["start_y"]

        cross_product = dx_node * dy_beam - dx_beam * dy_node
        if abs(cross_product) <= 2e-13: # take into account errors from floats
            # don’t count node as splitting beam if it’s start/end node of beam
            if (node["x"] == beam["start_x"]) and (node["y"] == beam["start_y"]):
                continue
            if (node["x"] == beam["end_x"]) and (node["y"] == beam["end_y"]):
                continue
        intersecting_beams.append(beam)
    return intersecting_beams

Given two beams and a third beam, return whether the two beams makes up the third beam.

def is_beam_component(beam1, beam2, beams):
beam1_start = {"x": beam1["start_x"], "y": beam1["start_y"],
"z": beam1["start_z"]}
beam1_end = {"x": beam1["end_x"], "y": beam1["end_y"], "z":
beam1["end_z"]}
beam2_start = {"x": beam2["start_x"], "y": beam2["start_y"],
"z": beam2["end_x"]}
beam2_end = {"x": beam2["end_x"], "y": beam2["end_y"], "z":
beam2["end_z"]}
beam_start = {"x": beam["start_x"], "y": beam["start_y"], "z":
beam["start_z"]}
beam_end = {"x": beam["end_x"], "y": beam["end_y"], "z":
beam["end_z"]}

if beam1_start == beam2_start:
    if (beam1_end == beam_start and beam2_end == beam_end) or
    (beam1_end == beam_end and beam2_end == beam_start):
        return True
if beam1_start == beam2_end:
    if (beam1_end == beam_start and beam2_start == beam_end)
    or (beam1_end == beam_end and beam2_start ==
    beam_start):
        return True
if beam1_end == beam2_start:
    if (beam1_start == beam_start and beam2_end == beam_end)
    or (beam1_start == beam_end and beam2_end ==
    beam_start):
        return True
if beam1_end == beam2_end:
    if (beam1_start == beam_start and beam2_start == beam_end)
    or (beam1_start == beam_end and beam2_start ==
    beam_start):
        return True
Given a three nodes, return the height of the triangle with the three nodes as vertices, and first node as apex of the triangle.

```python
def get_difference(node1, node2, node3):
    area = abs(node1["x"]*(node2["y"]-node3["y"])+node2["x"]*(
            node3["y"]-node1["y"])+node3["x"]*(node1["y"]-node2["y"]))/2
    base = get_euclidean_distance(node2, node3)
    height = 2 * area / base
    return height
```

Given two beams and a structure, return the beam in the structure that is replaced if the two beams replaces a beam in the structure, or None otherwise, and the node where the two beam components join.

"Replace" defined as if two nodes connected to the extra node is connected in structure 1 and not connected in structure 2.

```python
def is_replacement(beam1, beam2, structure1, structure2):
    beam1_start = {"x": beam1["start_x"], "y": beam1["start_y"],
               "z": beam1["start_z"]}
    beam1_end = {"x": beam1["end_x"], "y": beam1["end_y"], "z":
                  beam1["end_z"]}
    beam2_start = {"x": beam2["start_x"], "y": beam2["start_y"],
               "z": beam2["start_z"]}
    beam2_end = {"x": beam2["end_x"], "y": beam2["end_y"], "z":
```
if beam1_start == beam2_start:
    node1 = beam1_end
    node2 = beam2_end
    midpoint = beam1_start
elif beam1_start == beam2_end:
    node1 = beam1_end
    node2 = beam2_start
    midpoint = beam1_start
elif beam1_end == beam2_start:
    node1 = beam1_start
    node2 = beam2_end
    midpoint = beam1_end
elif beam1_end == beam2_end:
    node1 = beam1_start
    node2 = beam2_start
    midpoint = beam1_end
else:
    raise Exception("beams from same beam_set should share a node")
if (is_connected(node1, node2, structure1) != None) and not (is_connected(node1, node2, structure2) != None):
    return (midpoint, is_connected(node1, node2, structure1))
else:
    return (midpoint, None)

''
Given two nodes and a structure, return whether the beam that connects the two nodes if the two nodes are connected, return None otherwise.

def is_connected(node1, node2, structure):
    beam1 = {
        "start_x": node1["x"],
        "start_y": node1["y"],
        "start_z": node1["z"],
        "end_x": node2["x"],
        "end_y": node2["y"],
        "end_z": node2["z"]
    }
    beam2 = {
        "start_x": node2["x"],
        "start_y": node2["y"],
        "start_z": node2["z"],
        "end_x": node1["x"],
        "end_y": node1["y"],
        "end_z": node1["z"]
    }
    if beam1 in structure["beamList"]: return beam1
    elif beam2 in structure["beamList"]: return beam2
    else: return None

Given a node and a node list and an epsilon, return true if the node is approximately in the node list. Approximately being if each of the node's coordinates differs from the nodes in the node list by at most epsilon.
If it is approximately in the list, then return
it otherwise return None.

def approximately_in(node, nodeList, epsilon):
    if node in nodeList:
        return node
    found = False
    for n in nodeList:
        if (abs(float(node['x']) - float(n['x'])) <= epsilon) and
            (abs(float(node['y']) - float(n['y'])) <= epsilon)
            and (abs(float(node['z']) - float(n['z'])) <= epsilon):
            return n
    return None
Bibliography


