A NOVEL METHODOLOGY FOR CREATING AUTO GENERATED SPRING-BASED TRUSS PROBLEMS THROUGH MECHANIX

A Thesis

by

SABYASACHI CHAKRABORTY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	J. Maurice Rojas
Co-Chair of Committee,	Tracy Hammond
Committee Member,	Peter Howard
Head of Department,	Emil Straube

December 2018

Major Subject: Mathematics

ABSTRACT

Software that provides automated teaching assistance and instantaneous feedback for students has revolutionized the modern classroom. In addition to helping instructors manage large classes, the interactive experience can also benefit students. For instance, several existing systems incorporate recognition of student's hand-drawn solutions to problems. In these cases, the instructor sketches the solution to the problem and the student's sketches are expected to match this template. While this framework provides immediate feedback to students, it is still a constraint on instructors' time; additionally, it can be difficult to test conceptual understanding through only a small number of problems. There remains a strong need to generate questions automatically based on templates drawn by instructors so as to promote greater customization and variation in problems for students.

The focus of our research is to develop a novel method that can automatically generate new valid problems from a given reference problem. We have chosen linear spring-based truss systems as our domain. Another outcome of our research is to develop a method for recognizing a spring network sketched naturally by the user with commonly used symbols. We also generate different types of questions and boundary conditions from the recognized and auto-generated truss structures using the finite element method (FEM) in a novel way.

Our system has been integrated with Mechanix, a tool developed at Texas A&M University which supports free body diagrams (FBDs) and the creative design of truss structures. Mechanix supports engineering learning by providing intelligent and immediate feedback on hand-drawn sketches, and it has already been actively deployed in a number of university classrooms. We build a problem generator on top of Mechanix to leverage its capabilities for instantaneous, personalized feedback while enabling more thorough testing of student abilities and providing them a limitless pool of practice problems.

ACKNOWLEDGMENTS

I am extremely thankful to the members of the Sketch Recognition Lab for their continued support and help in the research work covered in this thesis. This thesis would not have been possible without their support. In particular, I truly want to thank my advisors Dr. Tracy Hammond (Co-Chair), Dr. J. Maurice Rojas (Chair), and Dr. Peter Howard (member) for their valuable guidance. I learned a lot while working with Dr. Hammond and am very appreciative to her for giving me the opportunity to work on this problem. She has indeed been a great inspiration.

Last but not the least, I would like to thank Shreyas Vinayakumar, Seth Polsley, Pallab Barai for helping me develop this application and finish this research. Also, I am really thankful to my wife for her moral support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis (or) dissertation committee consisting of Professors J. Maurice Rojas (Chair) and Peter Howard of the Department of Mathematics and Professor Tracy Hammond (Co-Chair) of the Department of Computer Science and Engineering.

Much of the research performed as part of this thesis is centered around the Mechanix project from the Sketch Recognition Lab at Texas A&M University. Stephanie Valentine and Seth Polsley were important contributors to the original development of this tool. Shreyas Vinayakumar contributed to the recognition model described in Chapter 3.

This material is based in part upon work supported by the National Science Foundation under Grant Number 1726306 and by Microsoft (Surface Hub Grant). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

All other work conducted for the thesis (or) dissertation was completed by the student independently.

Funding Sources

No outside funding was received for the research and compilation of this document.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

In the current economic situation it is hard for a college or university to maintain a preferable teacher-to-student ratio in large Science, Technology, Engineering, and Math (STEM) classrooms. State expenditures have increased significantly for enrollment in colleges of science and engineering; concurrently, tenure-track faculty have declined. A record from Colorado State University shows that the average Computer Engineering undergraduate teacher-to-student ratio worsened from 1:4.6 to 1:5.9 since 1995[1]. In 2014, the Education Commission of the States released measurements of teacher-to-student ratios in several jurisdictions, which shows even lower ratios[2]. Notably, the minimum ratio is 1:15 in New Mexico and New York, and the maximum is 1:30 in South Carolina, with several other states providing ratios that include teaching assistants with values ranging from 1:10 to 1:15.

Nizamettin and Bekir [2] studied the correlation between number of students to teachers and student achievement in Turkish high schools. They found a negative correlation of -0.561, supporting the conclusion that student achievement rates drop as there are fewer instructors available. They suggested hiring more teachers as the most direct solution to the problem. Unfortunately, this can be expensive, and it runs counter to the goal of minimizing budget expenditures.

Another option to diminish the impact of high student-to-teacher ratios on student achievement is to reduce the instructor workload. Lowering teachers' workload can enable mastery of goals and objectives in a class because they will have more time to interact

[1] http://www.engr.colostate.edu/dean/admin/files/04C_sutdent_faculty_ratios.pdf

[2] http://ecs.force.com/mbdata/mbquestRT?rep=Kq1411

with students[3, 4, 5]. For this reason, higher education has seen a continual growth in the adoption of educational technology [6, 7, 8]. Such tools allow instructors to handle larger classes as well as provide distance and online learning options [9, 10, 11].

The focus of our research is to use educational software to help reduce teacher workload and simultaneously help students to actively learn class materials, thereby increasing motivation and interest, promoting critical thinking skills, and stimulating independent learning. Research has shown that one technique that can improve learning and engagement in STEM courses is "deliberate practice." This technique involves introducing a concept but devoting a majority of time to interactive practice through problems and challenges during class time, a period termed "thinking scientifically" [12]. Our solution is built on top of Mechanix, an educational tool which tests students' knowledge of truss and free body diagram problems through interactive sketching. This framework supports "deliberate practice" teaching, and our system uses automatic, individualized generation of problems to encourage student learning.

Another critical aspect of our research is to enlarge the scope of usability of Mechanix so as to support institutions in their undergraduate/graduate programs in Math, Mechanical, Aerospace, or Civil Engineering Departments. To fulfill this goal, we have considered the situation that a teacher is covering the implementation of direct stiffness method of Finite Element Method (FEM) to a spring-based truss system in a STEM classroom. We want to provide an interface where students and their instructors can draw any truss systems, and Mechanix recognizes the spring-based truss system and generates different types of questions and boundary conditions for this system.

1.1 Existing Systems

Most of the popular Course Management Systems (CMS) have very limited support for automatic question generation. Blackboard[3] and WebCT[4] are some popular CMS, and they only support test banks and question editors. Respondus[5] provides a question editor to make matching questions.

Some projects have explored auto-problem generation using natural language processing (NLP), such as generation of single or multiple problems from text-based prompts [13]. AlgoTutor provides a system that aids instructors in creating question templates and in generating equivalent questions for an online tutoring system [14]. AlgoTutor uses a GUI for creating question templates, and the system creates equivalent multiple-choice questions for Computer Science I (CS-I) and Computer Science II (CS-II) level courses.

For STEM classes, there are some tools which can support tutors in creating questions. PhysicsBook is one important learning interface which uses sketch-based interaction [15]. This program enables users to solve physics problems and then animates any diagram used in solving the problem and provides feedback on the correctness. McGraw-HillConnect is a web-based digital teaching and learning environment[6]. It provides interactive tools for instructors and students to work on assignments with built-in assessment. WinTruss allows students to draw trusses using a set of pallet tools and it solves for forces in the members and shows truss deformation under a load. It is a non-sketch-based recognition system for solving truss diagrams [16]. SketchIT is an another tool which can read a sketch of a mechanical device and produce multiple families of designs from a single sketch [17].

Most of these tools still require instructors to provide solutions for all generated problems; this runs counter to our goal of reducing instructor workload. Furthermore, no such

[3] `www.blackboard.com`
[4] `www.webct.com`
[5] `www.respondus.com`
[6] `http://connect.mheducation.com`

solution could be found specifically in our domain, a tool supporting auto-generation of truss-based questions with the capability of creating new truss structures. Therefore, there is a significant need for new technologies which assist instructors to create question templates for truss structures with minimal effort, from which new questions can be generated and automatically solved. Motivated by this, we explore how an instructor can create a structural problem template (with a combination of springs and beams) and generate new problems by using this template.

To achieve our first goal to create an initial template, we ask instructors to draw their structure as they do in the classroom and assign initial information for the structure. To get a structure and its initial information, we use the recognition system Mechanix. We add another truss recognition model in Mechanix that can recognize spring-beam based truss structures. Regarding the second goal to generate questions from initial structural problems, we have implemented several algorithms in Mechanix that use certain constraints and methods of structural optimization processes.

1.2 Background of Mechanix

Mechanix is educational software developed by the Sketch Recognition Lab at Texas A&M University that provides a novel interface where students and instructors can interact. It can recognize, correct, and provide real-time feedback for a student's hand-drawn truss diagram that is checked against a sketch which was drawn by an instructor [18, 19, 20, 21]. It includes two modes for users, one for the instructors and another for the students. Both modes provide an interface where users can draw diagrams and insert meta-data related to the defined system.

Mechanix relies on geometric recognition to achieve an accurate results [22]. Geometric recognition has been explored and researched in various distinct domains. This recognition system uses a bottom-up approach and after preprocessing, a lower level rec-

ognizer can identify primitive shapes such as line segments, circles, arcs, polylines, triangles, spirals, helices, and field dots. On top of primitive recognition there is a high-level recognizer that uses a set of constraints to determine if the basic shapes and relationships between them can generate a complex shape. Mechanix relies on Paleo (also called PaleoSketch) which is a powerful low-level recognizer and has a reported accuracy of more than 98% [23].

Our study enlarges the scope of usability of Mechanix to support institutions in their undergraduate/graduate studies for STEM classes. We want to provide an interface where students and their instructors can draw any truss structure that can be a combination of springs and beams, requiring the development of spring-based truss recognition algorithms and inclusion of algorithms for generating different types of questions and boundary conditions for such system.

1.3 Research Goals

Our research is directed towards the following outcomes:

1. Reduce teaching load for instructors by implementing problem generation in educational software used in STEM classrooms

2. Development of a method to recognize and generate stable structures of interconnected beams/springs of arbitrary complexity from simple templates

3. Generation of new problems automatically to help create a large database of questions that can be used as individualized problems for students or problem banks across classes or schools

4. Finally, develop a solver to solve the equilibrium equations taking into consideration the geometric linearity associated with large deformations of the spring-based truss network.

2. BACKGROUND AND METHODOLOGY

In our study, we use some predefined techniques and methods that have already been used for different purposes. We improve these methods and develop new approaches that can serve our purpose through Mechanix. In this chapter, we provide an overview of our research.

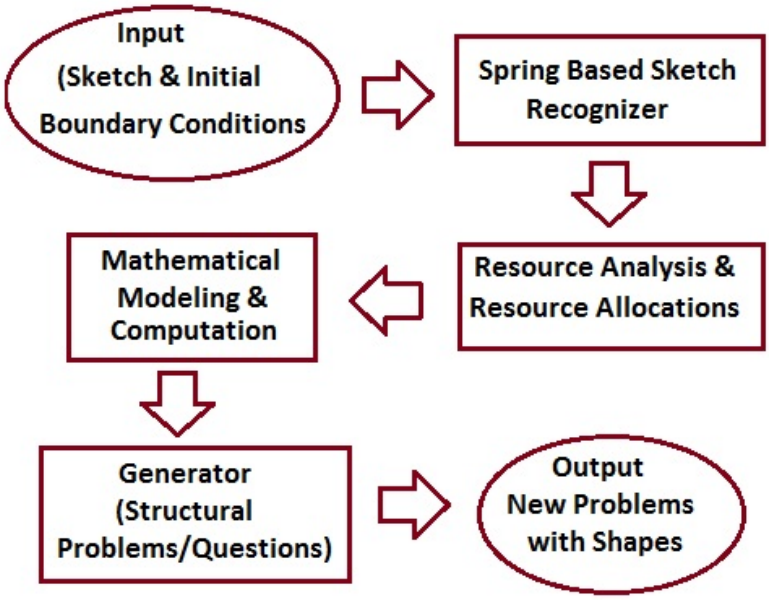


Figure 2.1: Full process of dynamic question generation.

2.1 Spring-Based Sketch Recognizer

In the current version of Mechanix, it can recognize only certain types of linear truss systems, and there is no scope for instructors to introduce spring systems or related problems using Mechanix. It fails to recognize certain linear structure based on a spring (or a combination of a spring and beam). Thus, there is a need to improve the current version of Mechanix.

For example, in an undergraduate mechanics class or in an introductory finite element to analyze truss class in a Civil Engineering Department, an instructor wants to introduce

spring problems. The instructor wants to discuss the problem shown in Figure 2.2. On the basis of this problem, the instructor would like to explore similar questions with different boundary conditions and put them in homeworks or quizzes to enhance the learning process. Sometimes, it is difficult to do such elaborate discussions on related problems within the time limit of a class. Also if students feel they needs to practice more to ensure a clear understanding of the topic, they may be unable to due to lack of resources. Often class textbooks do not cover all material in depth, and because of lack of practice questions, a set of students in the class are not able to learn the materials taught. They tend to lose their confidence and skill to solve problems on the topic. Research shows that this is one of the reasons why students sometimes lose interest in a particular course [24]. Our current research focuses on this issue in a mechanics class and tries to improve the current form of Mechanix software to facilitate the learning process by generating more questions on a particular topic taught in class.

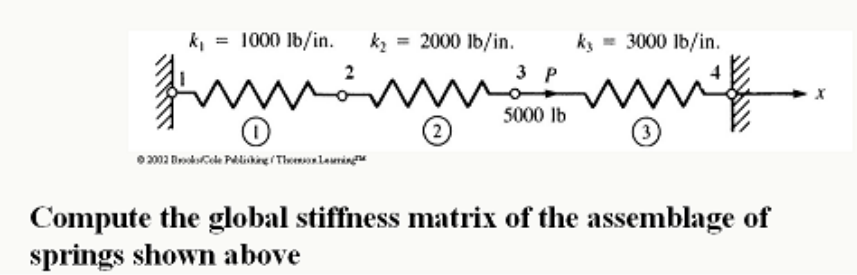


Figure 2.2: Example introductory problem that an instructor may wish to use as the basis for homework or test questions.

We collected a problem from University of Memphis [1]. We tried to create the problem in Figure 2.2 using the current version of Mechanix, and it failed to recognize the spring structure. Figure 2.3 shows this result in Mechanix.

To address this limitation in Mechanix, we develop another recognizer which we call a spring network recognizer that is able to identify any truss system that contains springs,

[1] www.ce.memphis.edu/7117/notes/presentations/chapter_02.pdf

beams, and loads drawn intuitively using the symbols found in most mechanics textbooks. We use the same bottom-up technique used in Mechanix for linear truss body diagrams [25], probabilistically combining stroke interpretations provided by the low-level geometric recognizer to build a composite view of the entire sketch. The spring recognizer updates its interpretation of the sketch every time the user draws or erases components on the screen.

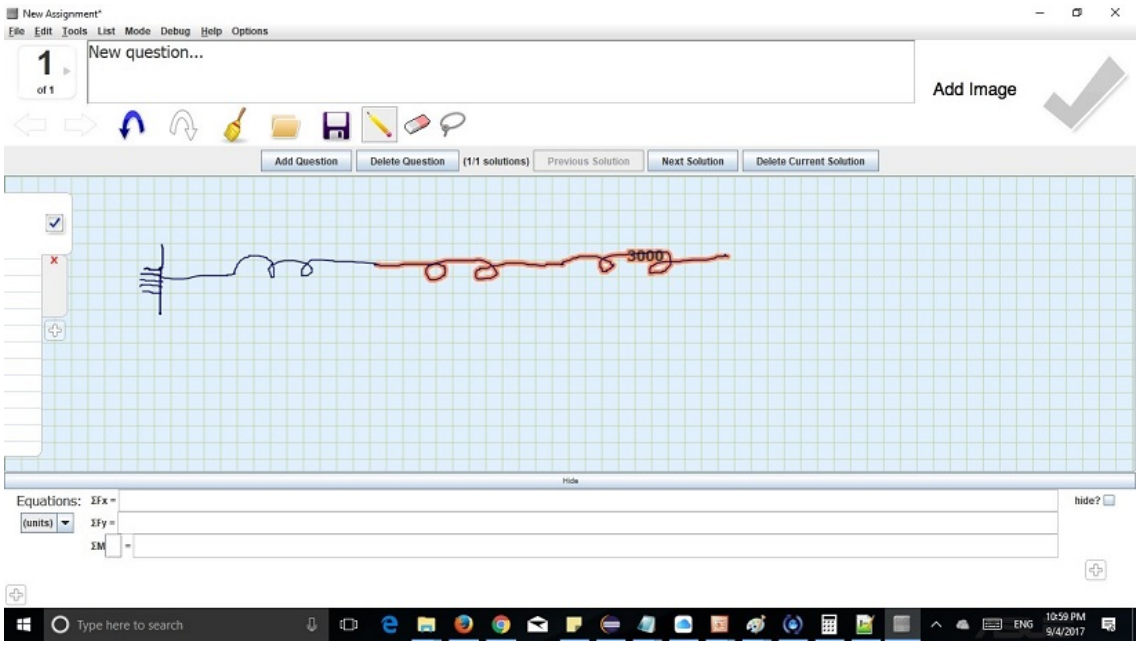


Figure 2.3: The sample spring-based truss problem is not recognized by the current Mechanix.

This new recognition process involves:

1. Recognizing low-level shapes using Paleo - these include line, helix, jagged line, circle [26, 27].

2. Grouping shapes together into appropriate elements (for example, spring, beam, load, support etc.) as well as associating labels with each element. This step involves some heuristics in determining whether the user means to draw several separate elements or a single element since the drawing is allowed to be completely freestyle.

3. Attempting to build a meaningful spring based truss system using the above elements based on physical constraints. For example, a meaningful diagram would have either end of a spring attached to another element and not left dangling. We also need to validate constraints on clamped versus free nodes. This step may again result in backtracking by going to Step 2, in case we picked the wrong interpretation.

We discuss our new recognizer in Chapter 3.

Figure 2.4 shows an example where our new recognizer is able to classify the problem from Figure 2.2.

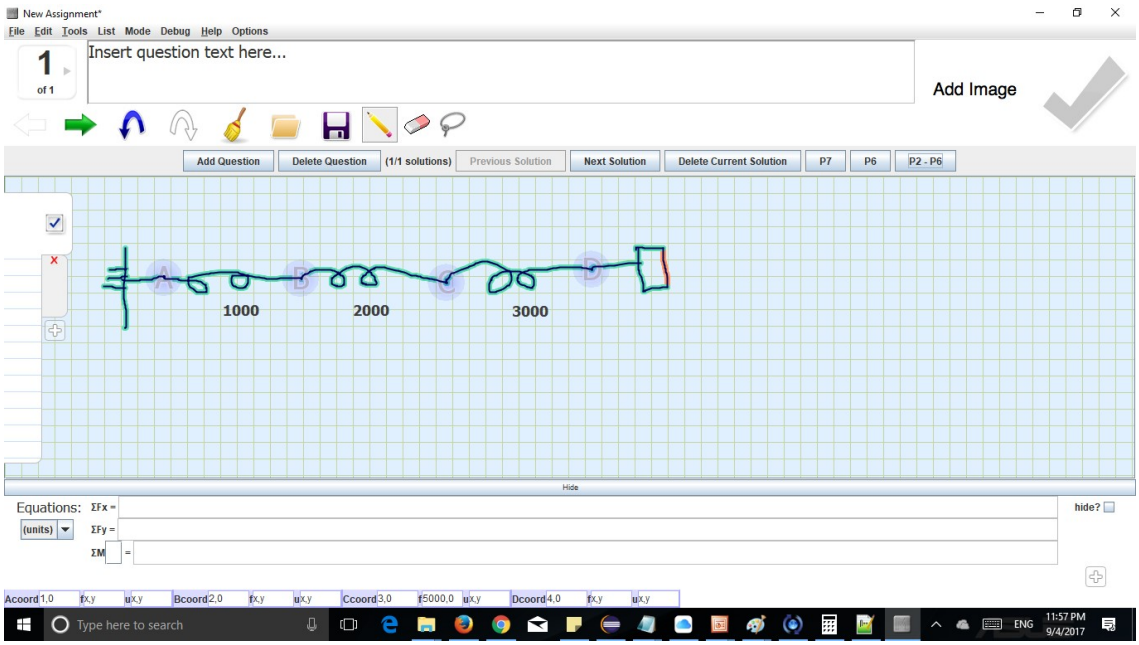


Figure 2.4: The spring-based truss from our previous example is now recognized by the upgraded system.

Let us consider the spring-based truss in 2D as shown in Figure 2.5. We collected this problem from the lecture note of Introduction to the Finite Element Method, conducted by Dr. J. Dean, Department of Material Science & Metallurgy, University of Cambridge [2]. If we create the same module through our recognizer, it is able to identify it correctly as seen in Figure 2.6a. The full system is shown in Figure 2.6b.

[2] www.ccg.msm.cam.ac.uk/images/FEMOR_Lecture_1.pdf

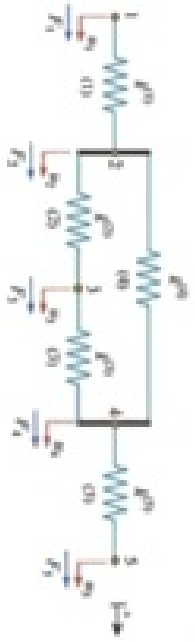
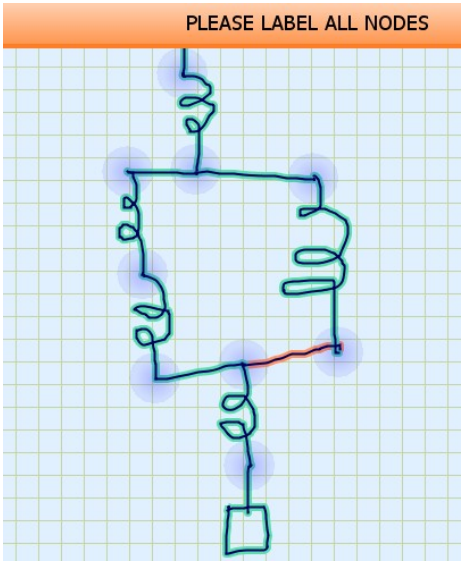
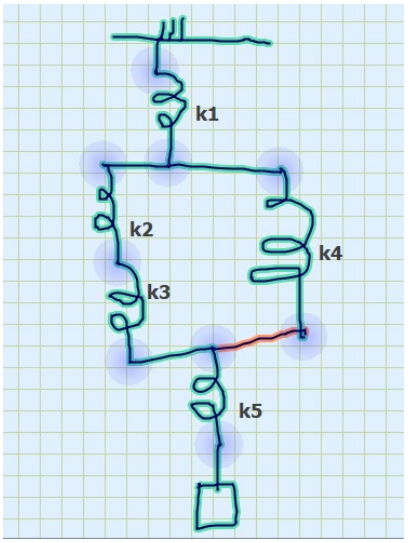


Figure 2.5: A spring-based truss system which contains five springs.



(a) Mechanix is now able to recognize the truss in Figure 4.6



(b) Instructor can create a model like Figure 4.6 through Mechanix.

Figure 2.6: Examples of spring-based truss systems recognized by Mechanix.

During this study, we only consider those structures which are made completely based on springs, though our recognizer is able to distinguish between a spring and beam. It can compute all properties of a beam automatically. Since our main focus is to generate questions from base templates, we do not build a solver which can solve a force-displacement balancing equation for spring- or beam-based truss systems within a limited time frame. Our system can easily implement beams once we have a solver for it. Since all new algorithms in this new system support both springs and beams we must discuss certain features which are created only for beams. All of these features are used in our system during problem formulation and generation.

2.2 Analyze Information

After recognizing spring-based truss systems, our program analyzes resources which are related to the given structure like the nodal information, boundary conditions, and the stiffness of the members of the truss. To get this information from the given structure, Mechanix already has a recognition system and input panel for retrieving values of 2D forces, length of beams, stiffness, etc. We focus on certain problems where the boundary conditions are related to force, displacements of the nodes, and stiffness of the truss elements of the given structure. To get this meta-data for nodes, we add a dynamic input panel where users (teachers/TAs) can provide information about nodes (coordinates, forces, and displacements). Instructors can provide initial boundary conditions through this panel.

The resource analyzing process first compiles information what user provides and checks if it can make a stable structure. If it finds that the given information is not enough to make the structure stable, then it allocates more reliable information and adds more constraints to the given structure. It also informs the user if it fails to build a stable structure. We add these constraints by following the recommendations found in Hultman's well-known approach [1]. To add more constraints to the given problem our system con-

siders that the given structure is made of steel and adds minimum weight according to Eurocode 3 (EC 3) [28, 29]. To distribute minimum weight and generate stiffness, it computes the weight of the structure, bulking stability, the number of free elements, yield strength, compression force, cross-sectional area, effective cross-sectional area, instability resistance, and so on. Depending on the boundary conditions, this recognition system has the ability to identify whether the system is valid (a stable structure) or not. This system always provides a stable structure when it creates questions for students learning.

In Chapter 4 we discuss in detail how our system analyzes information to build a structural problem. We use most of the constraints in Chapter 4 for both springs and beams. In this section, we only focus on those constraints which are used in our system, but we do not have to show their importance since we are not using beams now. The following constraint, derived from Hultman's work [1], is implemented into the structure through resource analysis:

Cross-section classifications: Different cross-sections have different local buckling resistance, depending on the inner width-to-thickness ratio. In this work, square, hot finished hollow profiles are used, and in that case, the different cross-sectional classes are calculated as (with c and t as in Figure 2.7):

$Class\,1\,if\,\frac{c}{t} \leq\ 33\varepsilon$

$Class\,2\,if\,\frac{c}{t} \leq\ 38\varepsilon$

$Class\,3\,if\,\frac{c}{t} \leq\ 42\varepsilon$

$and\,Class\,4\,if\,it\,fails\,to\,satisfy\,the\,limit\,for\,Class\,3$

$where\,\varepsilon\,is\,equal\,to\,\sqrt{\frac{235}{f_x}}\,with\,yield\,strength\,f_y\,in\,\frac{N}{mm^2}$

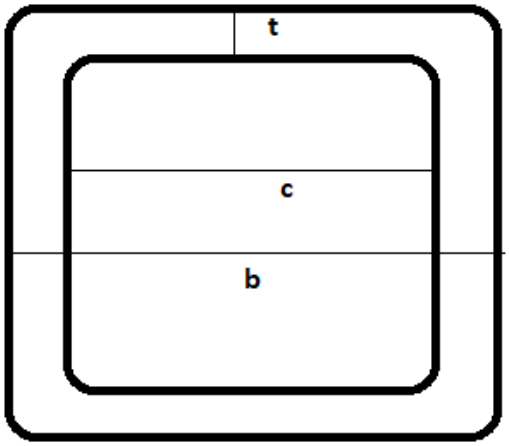


Figure 2.7: Designation of a square hollow profile [1].

2.3 Mathematical Modeling and Computation

To check whether the given system with initial boundary conditions is valid (stable) or not, the system creates a module which computes the stiffness matrix K of the given system after analyzing element resources. Here we follow the same numerical procedure which all textbooks suggested [30]. It immediately notifies the user if the given boundary conditions (nodal coordinate, force, and displacement) are valid for creating a unique solution of this problem or not. By comparing the rank of force and the augmented matrix created by stiffness and force, it informs the user that the given boundary condition has many solutions or no solution.

Our system has an ability to balance the stiffness of each member (Chapter 4.4.2.1) so that it produces a non-singular stiffness matrix, unless if a user added stiffness for a member that is many times greater than other user-provided stiffness values (we use a threshold of 10^9). Therefore the uniqueness of the equation $\{U\}$ = $[K]^{-}1$ $\{F\}$ depends on boundary conditions. The system allows those boundary conditions for which $\{U\}$ = $[K]^{-}1$ $\{F\}$ has a unique solution, and it helps users to add proper boundary conditions

for a given structure. For example, it can remove unnecessary nodal forces from the force displacement panel and consider proper nodal forces (Constraints 6). It gives continuous feedback during this process until proper boundary conditions are defined; then Mechanix runs the solver on the above equation in order to formulate the different types of questions based on the original structure.

We consider the momentum balance equation (see Equation 2.1) and solve it by using FEM. We fit this model in our system using nodal information and stiffness of the elements as input for the solver. It also generates a problem set $S = \{U, F, M\}$ where

$U \in \{u_i, v_i\}, F \in \{f_{xi}, f_{y_i}\}$

$and\, M \in \{k_i : k_j\, is\, the\, stiffness\, of\, the\, truss\, element\, j\}$

$for\, i = 1, ..., n; j = 1, .., m$

The moment balance equation:

$$\vec{\nabla}.\sigma + \vec{b} = 0 \tag{2.1}$$

or in a simplified form:

$$\frac{\partial \sigma_{i_j}}{\partial x_j} + \vec{b} = 0 \tag{2.2}$$

Once the system formulates K and finds that the above equation has a unique solution, then it finds out the solution for U. We used the solver for FEM implemented in Jama Matrix[3].

2.4 Question Generation

To generate questions from an initial structure, we have created a simulation which considers all possible aspects about how an instructor can create different questions on the

[3] https://math.nist.gov/javanumerics/jama/

basis of a given template. In this research, we have limited the scope of question generation to an undergraduate mechanics or mathematics class where a teacher introduces the Finite Element Method (FEM) using spring-based truss structure. This system has five different methods to generate new questions from a given sample, and it considers how to improve a question by adding difficulty levels, although this feature is not currently activated. In Chapter 5 we discuss all of our new algorithms in Mechanix in detail.

2.4.1 Change Boundary Conditions and Stiffness

In this case, the structure remains the same, meaning the given sketch, the number of truss elements, and the nodes will stay the same as in the initial problem. However, this method will change one of the vectors between displacement, force, and stiffness in a way so that the equation will have a solution. Therefore, it will create a new solution set, and the system will be able to ask a few specific questions from this new solution set. Depending on the computation level this new problem will be assigned a difficulty level dependent on each newly created question. Since the structure remains the same, the system initially sets the same difficulty level as the base level problem for this newly formed problem set P. It will compute a new difficulty level for each question of P while solving the Equation 2.1. This process first selects one parameter randomly from displacement, force, and stiffness and then uses a specific method to change values of this vector so that Equation 2.1 has an exact solution. We follow the same idea when we add constraints for the initial problem; details for each vector are discussed below.

2.4.1.1 Displacements at the nodes

This method randomly selects a node from n nodes. Initially, it randomly considers a node where displacement is unknown for students in the initial reference problem. After selecting the node it can change the displacement value in x-direction or y-direction or both, as well as validating the movement. In Chapter 5.1.1, we discuss how we add a new

displacement at the selected node.

After changing the value it rechecks the existence of the solution for Equation 2.1. Our research shows that the failure of this method to create a stable structure has probability zero since the stiffness matrix remains same, and it is non-singular (from the initial problem). Now we have a new displacement vector. Therefore, an instructor can generate a significant number of questions by changing the value of the displacement vector from

$F = KU$ where K is given non-singular stiffness matrix,

F is a set of force vectors and U is set of the displacement vector.

2.4.1.2 Stiffness of the truss members

The system would prefer to create a new equation for the given truss with the same number of nodes and truss elements, making the task of adjusting stiffness non-trivial. In general, in an introductory mechanics classroom a teacher wants to formulate all truss problems which are stable. Therefore, we create all truss structures which have non-singular stiffness matrices. We follow the same technique to assign a stiffness along with a truss element which we describe in Chapter 4.4.2.1. In this method, we first check if there is any missing stiffness from the given truss elements input from the instructor. If it finds any element for which the instructor did not define stiffness, then it assigns a stiffness by the method described in Chapter 4.4.2.1 to that element. It makes sure that the structure has a non-singular stiffness matrix.

2.4.2 Create Different Truss Topology

In Chapter 5.2 we describe how our system can create different truss structures from a base template. It achieves this by adding new members to structures or removing members and nodes from structures. It is able to generate a new template through this process. Adding a member increases computational work for students, thereby increasing the difficulty level. Conversely, removing members and nodes reduces the difficulty level.

3. SPRING-BASED TRUSS SYSTEM RECOGNIZER

To introduce spring-based truss system in Mechanix we define three new data structures. We first discuss their properties and a few important methods which are used in our recognizer to a generate question model. We use all the pre-existing data structures that were used in Mechanix, for example, Point, Shape, CoincidentConstraint, ConstrainableLine, ConstrainablePoint, ConstrainableShape, BoundingBox, and so on [22, 31].

3.1 Important Data Structures

Mechanix has a few important data structures to recognize a spring-based truss system. They also play an important role in designing the truss for generating problems. We currently use a solver that can solve a force-displacement equilibrium equation through stiffness just for a linear truss whose members are all springs. Although our new recognizer can distinguish between completely spring-based truss or a combination of spring- and beam-based structure. A design method is introduced to assign different properties to a member of the truss. This method can assign different properties to a spring and beam. Since we don't have the solver ready for the structure which has both a spring and beam, we only concentrate on spring-based structures.

3.1.1 Spring Node

$SpringNodes$ refers to the vertices of an undirected graph (spring system) whose edges are a spring truss. It is the discrete position of a graph and an edge (spring and/or beam) of a graph that is connected between two spring nodes.

The properties of $SpringNode$ are the following

1. $List < Shape > \ shapes$: stores all shapes that are connected at this SpringNode.

2. $Point\ p$: stores the intersection point where all edges of the graph are connected

after beautification.

3. $String\ label$: stores the label of the node.

4. $Double[][]\ ProblemCoord$: stores the value of the user entered coordinates.

3.1.2 Edges of the Graph

$SNode$ is referred to as an edge of the undirected graph (spring system). An $SNode$ is connected between two $SpringNodes$ by a Shape. The shape of an edge i.e $vertex$ can be a helix, line, closed shape, or circle. The important properties of the $SNode$ are listed below, and the constructors are show in Algorithm 1:

1. $Shape\ vertex$: stores the the shape of the edges (Helix, line or closed shape).

2. $List < SNode > \ edges$: list of $SNodes$ which are connected to any of the endpoints.

3. $Point\ firstPoint,\ secondPoint$: stores the endpoints of the edge.

4. $boolean\ firstPointConnected$: boolean indicates if $firstPoint$ is connected with any other $SNode$ then $firstPointConnected = true$.

5. $boolean\ lastPointConnected$: boolean indiciates if $secondPoint$ is connected with any other $SNode$ then $lastPointConnected = true$.

$edges$ is used to store all other $Snodes$ which have one common endpoint, either $firstPoint$ or $secondPoint$. During creating a spring system, our recognizer updates the $edges$ by adding all connected edges of the graph which share the same endpoint. Mechanix considers an $Snode$ as a member of a spring-based truss system if the $vertex$ is either helix or line. In Figure 3.1, $AB,\ AC,\ AD,\ AE,\ BF,\ BG$ are $SNodes$. A and B are $firstPoint$ and $secondPoint$. $vertex$ of AB is a line $Shape$. A and B will be the a point

Algorithm 1 $SNode$

1: **procedure** $SNode(v)$
2: $vertex \leftarrow v$
3: $edges \leftarrow new\ ArrayList < SNode > ()$
4: $firstPoint \leftarrow \emptyset$
5: $endPoint \leftarrow \emptyset$
6: $firstPointConnected = false$
7: $lasstPointConnected = false$

Algorithm 2 $SNode$

1: **procedure** $SNode(v, first, end)$
2: $vertex \leftarrow v$
3: $edges \leftarrow new\ ArrayList < SNode > ()$
4: $firstPoint = first$
5: $endPoint = end$
6: $firstPointConnected = false$
7: $lasstPointConnected = false$

of $SpringNode$ in $SpringSystem$. Now $edge$ of AB contains $AC,\ AD,\ AE,\ BF,\ BG$. $firstPointConnected$ and $lastPointConnected$ are true since $firstPoint$ and $secondPoint$ are connected with other $SNode$.

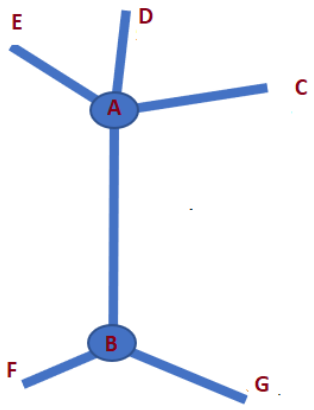


Figure 3.1: Example of $SNode$ data structure

3.1.3 Spring System

$SpringSystem$ is an undirected connected graph which is created by a set of shapes S. S contains clamped support, Circle, Closed Shape, Helix, and Line. The vertices of this graph are $SpringNode$ and edges are $SNode$. A helix shape can be visually represented as a spring, and a line can be a beam. A small closed shape with area 7000 or less is identified as a load of the spring-based truss system. Two components of this graph are connected if an endpoint of a spring or beam touches 1) another endpoint of a spring or beam, 2) any connected point of a clamp support, or 3) a point of the closed shape/circle. A $SpringSystem$ is considered as complete if it satisfies the following constraints:

1. Every spring/beam should be connected at both endpoints.
2. At least one clamped support should be present in the graph.
3. At least one load should be present in the graph.
4. Nodes are labeled with capital letters.

Our recognizer cannot create two $SpringNodes$ at the same geometric location. It satisfies one of our constraint of designing a valid structure, i.e, two nodes should have two different coordinates[1].

A spring system has the following properties, and the parameterized constructor supporting a Shape of a clamped support is given in Algorithm 3.

1. $String\ Label =$ "springNetwork",
2. $List < SNode > graph$: list of all edges,
3. $Double\ Constraint\_Confidence = .55$,

[1]Note: “spring system" henceforth directly refers to the $SpringSystem$ data structure.

4. $List < SpringNode > nodes$: stores all vertices.

5. $List < Point > clampedNodePoints$: list of all endpoints of the shapes which have other endpoint as $pointsOnClampedSupport$.

6. $List < Point > pointsOnClampedSupport$: list of all points which are considered as a connection of the clamp support and a $SNode$; this is also a point of a $SpringNode$.

7. $List < Point > loadedPoints$: list of all points where user added load shapes. This is also a point of a $SpringNode$.

Algorithm 3 $SpringSystem$

```
1: procedure SpringSystem(ClampedSupport)
2:     super()
3:     graph ← new ArrayList < SpringNode > ()
4:     nodes ← new ArrayList < SpringNode > ()
5:     clampedNodePoints ← new ArrayList < Point > ()
6:     pointsOnClampedSupport ← newArrayList < Point > ()
7:     loadedPoints ← new ArrayList < Point > ()
8:     SNode snode ← new SNode(clamp)
9:     snode.firstPointConnected ← true
10:    snode.lastPointConnected ← true
11:    add(snode)
```

To check whether a newly added Shape is a component of a graph, we define a new method named as $addAsSrpingComponent$. This method is called once a spring system has been created and the graph of this spring system has at least one clamped support. We use this method in the spring system Recognizer. We use some confidence values to verify if two points coincide or two shapes intersect that are already in Mechanix. Hammond and

Johnston developed the mechanisms to find out the confidence value based on LADDER [31, 22, 32]. To build a spring system we reuse the following thresholds in our research.

Coincident Constraint— To verify if two points are in same location we find out constraint value by using solve method of $CoincidentConstraint$. The default threshold of coincident constraint is 0.15. Here we refer $CC()$ and $CC(x)$ as constructors of $CoincidentConstraint$ where $x \in \mathcal{R}$ and $CC(p, q)$ represents the "$solve\ method$" to get the confidence value where p, q are two points.

Intersection Constraint— To verify if two shapes intersect we find out the confidence value by using "$solve\ methods$" of $IntersectsConstaint$ class. The default threshold is the distance for how close things have to be before they intersect. It is 0.1 for intersection constraint. Here we refer $IC(P, Q)$, $IC(p, Q)$ which represents the "$solve\ methods$" to get the confidence values where P, Q are two shapes and p is a point.

We also reuse Constrainable Point & Constrainable Line from Mechanix (Hammond and Davis [22]). At this juncture, $CS_{ShapeName}$ represents the constrainable Shape of a shape and $CS_{ShapeName}$ is the constructor. CL_{Line} is the constructor where $Line$ is a shape. We also use the following variables in our algorithms

1. $Piece$ denotes current shape which needs to verify.
2. $ShapeName_{px}$ denotes a $Point$ of a shape at location x. Example: $Piece_{pf}$ denotes the first point of $Piece$ and $Piece_{pl}$ denotes the last point of $Piece$.
3. Edg_j denotes j^{th} edge (shape) in graph of the current spring system.
4. nearness1, nearness2 and intersectionConfidence are used to store confidence values,

5. $SNode_{ShapeName}$ denotes the $SNode$ of a $Shape$, named by $ShapeName$.

6. $MakeNode(P, Q, x)$ is the same $getOrSetSpringNode$ method to create a $SpringNode$ where P,Q are shapes and x is a point.

The new spring network recognizer calls $addAsSrpingComponent$ method to check if the newly added shape is an edge of the $graph$ of current $SpringSystem$. It operates according to the following steps:

1. **Step 1:**

   It first verifies the shape label of $Piece$.

2. **Step 2: (Case 1)**

   If shape label of $Piece$ is either line or helix. recognizer then follows steps 2.1 to Step 2.3.

   (a) **Step 2:** Next it runs the following steps for each Edg_j of the $graph$ of the current $SpringSystem$. j = 1,..,m, m = number of $SNodes$ are in $graph$.

   (b) **Step 2.1:** It first checks if $Piece = Edg_j$ then goes for next Edg_{j+1}, otherwise follows next steps.

   (c) **Step 2.2:** If shape label of Edg_j is either line or helix it then follows steps 2.2.1 to step 2.2.6.

      i. **Step 2.2.1:** It first verifies if any endpoints of $piece$ is coincident with first point or last point of the Edg_j. It Identifies the endpoint of $piece$ which is connected with which endpoint of Edg_j. It can be either the first point or the last point of piece which is connected with the first point or last point of Edg_j. If they do it then follows Step 2.2.2 to Step 2.2.4.

      ii. **Step 2.2.2:** Creates a new $SNodes$ with $piece$ and itś $firstPoint$ & $secondPoint$. Say $SNode_{Piece}$.

iii. **Step 2.2.3:** Updates the $graph$ of the spring system and $edges$ of both $SNode_{Piece}$ and Edg_j.

iv. **Step 2.2.4:** Update that points of $piece$ and $Edgj$ by $firstPointConnected = true$ and/or $lastPointConnected = true$. Creates a new $SpringNode$.

v. **Step 2.2.5:** If step 2.2.1 fails it then checks if $piece$ and $vertex$ of Edg_j, say S, intersect. If they intersect it then creates a new $SNode_{Piece}$. Updates the $graph$ of the spring system and $edges$ of both $SNode_{Piece}$ and Edg_j. It then follows next steps

vi. **Step 2.2.6:** it checks if any endpoints of $piece$ intersects or lies at a point of S. If they do not, it then checks if any endpoint of S intersects or lies on $piece$. It updates the boolean values of $SNode_{Piece}$ and Edg_j (i.e., $firstPointConnected$ & $lastPointConnected$) according to the outcomes and creates a new $SpringNode$.

vii. **Step 2.2.7:** Continue for next Edg_{j+1}.

(d) **Step 2.3:** If shape of Edg_j is a clamped support it then follows steps 2.3.1 to Step 2.3.3.

i. **Step 2.3.1:** It finds out the base line of the clamped support.

ii. **Step 2.3.2:** Next it checks if either one endpoint of $piece$ intersects the base line. If one endpoint of $piece$ intersects the base line, it then creates $SNode_{Piece}$. It updates the $graph$ of the spring system and $edges$ of both $SNode_{Piece}$ and Edg_j. According to the outcomes, it updates the $clampedNodePoints$ and $pointsOnClampedSupport$ by adding the right points of the $Piece$ respectively. It also updates $firstPointConnected$ or $lastPointConnected$ of $SNode_{Piece}$ depend-

ing on outcomes.

iii. **Step 2.3.3:** Continue for next Edg_{j+1}.

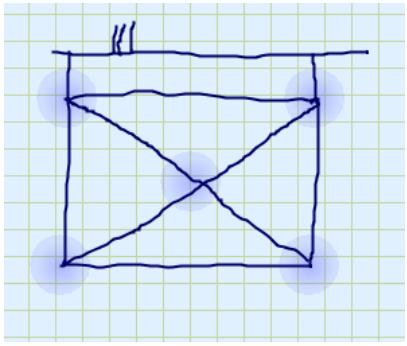
3. **Step 3: (Case 2)**

   If shape label of $Piece$ is either a closed shape or a circle it then follows the following step

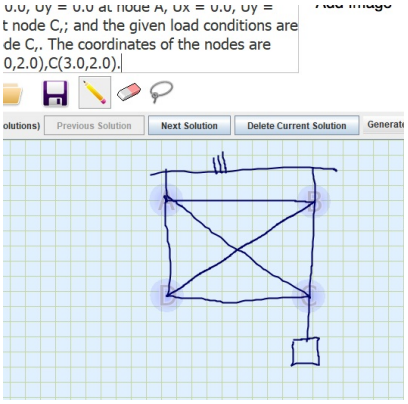
   (a) **Step 3.1:** It checks if any endpoint of Edg_j lies on the $Piece$. If any endpoint lies on $Piece$ it then create a new $SNode$ by using only $Piece$, update the properties of new $SNode$ by $firstPointConnected = true$ and $lastPointConnected = true$. Updates the Edg_j and $loadedPoints$.

Algorithm 4 is a detailed overview of the $addAsSpringComponent$ method. This is our new spring system recognizer. Once the above method returns a non-empty $SNode$ the spring recognizer (Section 3.2) is able to check if the graph is connected or not.

Sketching Constraint: This new recognizer helps a user to reduce the scope of drawing a faulty structure. A user can not draw two members (shapes) which intersect each other without creating a node. Spring system recognizer creates a spring node at the intersecting point when the user draws two shapes which intersect each other. It helps to satisfy one of our modeling constraint 5. Though the user can add a truss member (shape) which intersects another member without having a node by using "Add a new member" of our question generation model. This model checks if the system can add a member between two nodes which still guarantees the stability of the structure. In this case, two members (shapes) can intersect and our recognizer does not consider the point of intersection as a node.



(a) Recognizer created a spring node during drawing



(b) System added two members without node

Figure 3.2: In image 3.2a System assigned a $SpringNode$ at the intersection point during drawing.In image 3.2b system added a intersecting member without have a $SpringNode$.

3.2 Spring System Recognizer

Mechanix already has a high-level recognizer to recognize a free-body diagram and liner truss system. The inner workings of Mechanix and the artificial intelligence behind-the-scenes have been documented in detail in Field et al. [33] and Kebodeaux et al. [34] and most completely in Valentine et al. [19, 35]. The research on sketch recognition depends on gesture recognition [36, 37], vision-based recognition [38, 39, 40], and geometric recognition [41, 22, 42, 43, 44, 45, 46, 47, 48]. We develop another new recognizer to recognize any spring- and beam- based structure based on PaleoSketch [23] and a geometric recognition approach that is closely related to LADDER [22, 49, 50, 51]. We use the same bottom-up technique used in Mechanix [25] for linear truss body diagrams [52]. When the user selects "instructor spring mode", the system calls the Paleosketch recognizer. Paleo is a low-level recognition and beautification system developed by Hammond and Paulson [23, 53, 54, 55] that can recognize eight primitive shapes as well as combinations of these primitives with recognition rates of 98.56%. An upper-level has been created by merging

primitive shapes drawn with multiple strokes.

The spring network recognizer implemented here is a higher-level geometric shape created by a combination of lower-level of primitives meeting certain geometric constraints. We use four primitive shapes and one complex shape in our new recognizer. In this recognizer, we do not store any history of previous shapes. We use the same brute force algorithm to check whether all shapes added by the user satisfy our predefined conditions. We divide our recognizer into four categories (Paleo level, spring system level, Spring Network level, and truss/non-truss member level). We model the diagram drawn by the user visually as a spring system. Once we have recognized the diagram as a valid and complete spring system in a visual sense, we transform this visual model into a physical representation that we refer to as a $Spring\ Network$.

3.2.1 Algorithm

The first part of our recognizer forms a valid spring system. As discussed above, a spring system is a connected graph which is created by a set of shapes S. S contains clamp supports, helices, lines, circles, and closed shapes. A helix shape can visually represent a spring and a line shape can be a beam. Once the spring network recognizer runs, it first initiates the Paleo recognizer. When the user releases the mouse it sends the users drawn stroke or set of strokes which are currently in the working panel to the Paleo Recognizer. Once Paleo finishes recognizing primitive shape/shapes and labels the shape's name Mechanix then send them to our high-level recognizer to builds a spring system.

The spring system is considered as complete if it satisfies the following constraints:

1. At least 1 clamped support should be present.

2. At least 1 load should be present.

3. Every spring/beam should be connected at both endpoints.

4. Nodes are labeled with a letter. Currently, we give only 26 options (A to Z) for labeling, but it can be extended for any number of nodes.

The mechanism of spring system recognizer is as follows:

$Input$: List of shapes

$Output$: Spring system graph which is complete as per our constraints.

1. Send users sketch (stroke or strokes) to Paleo recognizer to recognize low-level primitive shapes including clamp support. Paleo uses geometric techniques to classify strokes into shapes and labels shapes with their geometric names.

2. Remove all $terminal$ shapes from the sketch. The terminal shapes are the node, tick mark, closed shape, circle, axis, and measurement.

3. Send the sketch to our high-level recognizer, named as $recognizeSpringNetwork$.

4. Next in $recognizeSpringNetwork$, copy all shapes in a list, named $pieces$.

5. Check all shapes one by one from $pieces$ if it contains any shape as spring system or any node. If it has a spring system then add all shapes from this spring system to $pieces$ and remove the current shape of the spring system from the list. If the current shape is a node then remove it from $pieces$.

6. Next check if $pieces$ has at least one clamp support or not. If it does not have a clamped support then return the control to the user to continue drawing. Otherwise, create a spring system with clamped support (only) and remove it from $pieces$.

7. Check each shape from the remaining elements of $pieces$ if it can be an edge or a load of the spring system by using $addAsSrpingComponent$ method. In each case where $addAsSrpingComponent$ method returns a $SNode$ which is already added to the current spring system then remove that shape from $pieces$.

8. Continue step 8 until $pieces$ is empty.

9. Once $pieces$ is empty then check the newly formed spring system is connected (and complete) or not. We use $isConnected$ method to check if we have a connected graph or not.

10. If we have a connected graph with at least two nodes, one edge and one clamp support then create a $DirectStiffnessQuestion$ by using newly found spring system.

11. Next create a $Shape\ S$ and label it as "SpringNetwork".

12. Next add each $vertex$ (which is a shape) of $SNode$ from the $graph$ of the newly found $SpringSystem$ to S and explode them with user drawn sketch.

13. Add S to users sketch.

14. Label each $SpringNode$ of the newly found $SpringSystem$ as "node" and update the $Point$ of the node.

15. Re-validate label and update the shape label.

16. Repaint GUI and update color of the sketch.

In each step, when the above algorithm creates a shape of the spring node, it immediately updates the force-displacement input panel for that respective node where an instructor can enter the coordinate, force, and displacement value for the node. Once the system recognizes a spring network system which is connected, it then creates a question to use the direct stiffness method to solve it. In our next chapter, we discuss how the system is able to extract information from a newly created spring system network and create a direct stiffness question for students.

We use another new method in $DirectStiffnessQuestion$ to create a $SpringSystem$. We name it as $toSketch()$. Our system builds another new version of $graph$ and a different set of $nodes$ during new question generation. The $toSketch()$ method adds $Shapes$ from these newly modified $graph$ and $nodes$ to a $Sketch$. We use this $Sketch$ to create a new version of $SpringSystem$ to form a new $DirectStiffnessQuestion$ by using a method, named as $getSpringSystemFromSketch\ (Sketch\ sketch)$. This method works as the same way as above algorithm for Spring network. The only difference is that we do not need to use brute force algorithm here.

4. RESOURCE ANALYSIS AND PROBLEM FORMULATION

We consider the scenario when an instructor wants to teach how to solve problems using the Finite Element Method (FEM). He/she usually follows a few steps orderly in a classroom. An instructor gives a detailed lecture about the topic concerned (in this case it is direct stiffness method) with one or few examples. To provide an example, first, he/she draws a picture of a spring based truss (since we consider this type of problem in this research for our recognizer) and labels the nodes. Once labeling is finished then he/she adds conditions/properties for this truss to formulate the problem. After formulating the problem visually for students, he/she shows how to distribute the conditions along nodes and members of the truss and then works on solving the problem using the direct stiffness method. Our system works in a similar way. After recognizing a spring system network, Mechanix works to build a question that is solvable using direct stiffness method from FEM. Once an instructor draws a spring system on panel Mechanix asks the user to label all the nodes. After all the nodes are labeled it then asks the user to enter initial valid boundary conditions on each node through the input panel. A user can add stiffness of the members. Once the system has a spring network which is a spring based truss and its proper initial boundary conditions then it generates new questions and their solutions.

The system informs the user about the validity of the current problem once the instructor finishes his/her part by drawing a structure and entering boundary conditions. Before verifying the validity of the given problem, Mechanix needs to identify the information about nodes and members based on boundary conditions and member properties if they exist. After recognizing the spring network it first creates a base level question and updates information for this question when the user finishes labeling the nodes and allocating boundary conditions. To construct a base level problem we create two data structures. One

is for the question which is named as $DirectStiffnessQuestion$ and second is for an individual member of the spring based truss which is named as $SpringNetworkMember$.

If the system finds valid boundary conditions for user given structure then it updates the $DirectStiffnessQuestion$, which is basically modeling the base problem. It updates all properties of $DirectStiffnessQuestion$ and $SpringNetworkMember$ so that our solver can solve the problem equation 4.4a. A resource allocation model is important here during this problem formulation time. Here we consider resource as structural properties which need to know when constraints need to be improved to make a stable structure [56, 57, 58, 59] . All properties may not be available from a user. The system needs the ability to generate resources for members of the truss.

We first discuss $SpringNetworkMember$ and $DirectStiffnessQuestion$ and then we explain how resource allocation works to model a base problem to create a stable structure.

4.1 Identify the Members of a Spring Based Truss

After the spring network recognizer recognizes a connected graph, it creates a base level question. Since we use the brute force algorithm in spring network recognizer, it updates this base question every time when a user adds another new stroke to the current spring network. During creating a base question an algorithm runs to identify members of this spring based truss network. Mechanix has another new data structure, $SpringNetworkMember$, for storing information about the member of a spring based truss. We will discuss about $SpringNetworkMember$ in Section 4.3. It is important to identify all truss members and their material properties in the network and add member list to the questions. Mechanix has a new algorithm that can recognize a truss member which is a shape (line/helix) and connected between two spring nodes. A user creates a truss member in different ways through our system. Our new algorithm recognizes this

new member and updates the member list of $DirectStiffnessQuestion$. The algorithm works according to how a user adds shapes to a spring network in working panel. An instructor can add three or four shapes in the following ways to create one or multiple truss members. This algorithm creates truss members according to the userś drawing pattern-

1. A user can draw three consecutive shapes, say Shape1, Shape2, and Shape3. They can be of any combination of lines and helices. Consider each endpoint of Shape2 connected to one endpoint B of Shape1 and one endpoint C of Shape3 so that Shape2 is connected between two spring nodes B and C. Our system considers Shape2 as a truss member [Figure 4.1].

2. A user first draws Shape1 with two endpoints X and Y. Consider X and Y are $SpringNodes$. Now user adds another Shape2 whose one endpoint intersects a point of Shape1 other than X and Y. This system creates a $SpringNode$ at this intersecting point, say A. Now again user draws another Shape3 in the same way, one endpoint of Shape3 intersects a point of Shape1 except X, A & Y. The system then the system creates another $SpringNode$, say B, on Shape1. Therefore the system divides Shape1 by three parts XA, AB, BY respectively. The system considers each of them as $SpringNetworkMember$ e.i. a truss member [Figure 4.3].

3. It can be possible that one endpoint of Shape1 is connected to another shape except for a clamp support or closed shape (i.e., load). Therefore we have a spring node at this point. If another endpoint of Shape1 intersects a closed shape, then Shape1 has only one $SpringNode$. If the user adds Shape2 and Shape3 in previous ways[1], then this algorithm creates only two members, in Figure 4.2, XA, XB are $SpringNetworkMember$.

[1] All shapes should be added in a way such that spring network recognizer can create a connected graph.

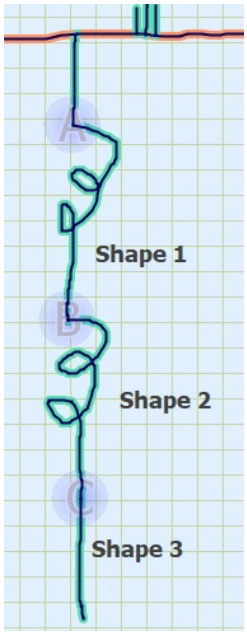


Figure 4.1: Shape 2 is a member which is connected between A and B

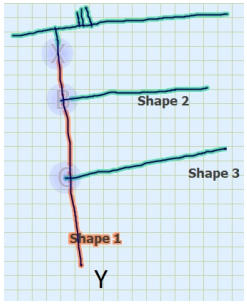


Figure 4.2: Shape 2 and Shape 3 touch Shape 1 at $SpringNodes\ B$ and C. XB, BC are two trusses members.

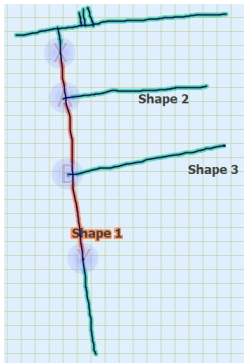


Figure 4.3: Shape 2 and Shape 3 touch Shape 1 at $SpringNodes\ B$ and C. XB, BC and BY are three trusses members.

4.2 Direct Stiffness Question

We use a new data structure to define a question and use it for direct stiffness method. We name it as $DirectStiffnessQuestion$. The initial reference is created when the system finds a spring network. If a user continues to change the sketch after recognizing first spring network, then the system updates this question. The system does not run any numerical method for this question until the user tries to generate a new question. To stop unnecessary computation there is a validation rule that prevents the user to run any question generation method until all coordinate values of the nodes are entered. The user cannot request to generate a question from his/her sketch until all the nodes are labeled and the coordinate values entered for each node. If the user does not provide the value for stiffness of a truss member then we consider that member is built by steel material. Few important properties of direct stiffness questions are $springMembers$, $springSystem$, $solution$, $elementNodesVectors$, $constantCrossSectionalwidth$, $constantElementThickness$, $constantMomentOfInertia$, $steelElasticity$, $constantYieldStrength$, $crossSectionalClassification$, $StrureWeight$, $density$, $alpha$. The $springMembers$ is a list of $SpringNetworkMember$ and it is used to store all truss members in the truss body. The $springSystem$ is an object of $SpringSystem$ data structure. It is important to attach the $SpringSystem$ with this question since we need to save a copy of the original $SpringSystem$. The same process can create multiple modified versions of the original $SpringSystem$ during generation of new questions. Therefore we need to keep the original version. A newly generated question has its own version of $SpringSystem$. It reduces the runtime to find out original $SpringSystem$. Therefore the user can create multiple questions by a single action. $solution$ is an object of a new data structure, named as $SpringNetworkSolution$. $solution$ is used to store all information about question's solution. $elementNodesVectors$ is used to store information about the connection between

nodes. We use this array list for computation purpose. Initially, these are empty until the user is ready to generate a question. The $setUpDispacementAndForce$ method is used to assign displacements, forces, and coordinates at each node. We will discuss about other important properties of $DirectStiffnessQuestion$ in resource allocation section.
The constructor of $DirectStiffnessQuestion$ is following

---

function DIRECTSTIFFNESSQUESTION( )
checkNodesAlongMembers(springSystem);
setUpDispacementAndForce(springSystem);
Initialize other fields..

4.3 Spring Network Member

To store truss member properties of a spring based truss system Mechanix uses another new data structure, named as $SpringNetworkMember$. Each object of $SpringNetworkMember$ has all important properties that we need to create a question for spring based truss and its solution. The important fields/properties are for nodal information (node labels, $SpringNodes$), the stiffness of the member, cross-sectional area, length of the member, thickness, the moment of inertia, elastic module, yield strength, cross-sectional classification, inner width.

4.4 Resource Allocation and Modeling

After recognizing a spring network and initiating an object of the $DirectStiffnessQuestion$, the resource allocation part comes to model the initial question. Our system analyzes resources which are related to the given structure like the nodal information, boundary conditions and the stiffness of the members of the truss. To get meta-data for the given structure Mechanix already has a recognition system and input

panel to get values of 2D forces, length of the beam etc. We focus on certain problems where the boundary conditions are related to coordinates, force, and displacements at the nodes and stiffness of the truss elements of the given structure. To get these meta-data for nodes and stiffness for a member there is a need where a user can add this information.

Our system in Mechanix has an input panel, named as the Force-Displacement panel where users (instructors/TAs) can enter information about nodes (coordinates, forces, and displacements). Once our recognizer recognizes a spring node it immediately updates this force displacement panel. The recognizer then creates a spring system when a user adds a clamped support in working panel. It creates a $SpringNode$ when a user draws two connected shapes (any combinations of helix and line) after a clamp support. During creating a $SpringNode$, it adds three JTextAreas for this node. These JTextAreas are for getting input on coordinate, force and displacement for this node respectively. Figure 4.4, shows that the input panel has three text areas since working panel has only one node. In Figure 4.5, once an user adds another connected truss element, recognizer creates second node. The force-displacement panel gets updated for this new node. If User labels the node, it immediately adds the labeling to the corresponding force-displacement panel of this node. It helps a user to identify the input areas for a specific node.

After recognizing spring networks, Mechanix immediately asks the user to label the nodes and enter the nodal information. In our research, we expect user should enter valid information for initially drawn truss structure which user wants to use to generate problems. Here we refer to a valid problem which has a unique solution. To get a unique solution from the given boundary conditions (coordinates, forces, displacements) on the initial user drawn structure, the stiffness of the members plays an important part. Mechanix can get stiffness information from the user or it can generate stiffness values for its members so that the initial problem should have a valid structure. It's a new feature that we have added to Mechanix.

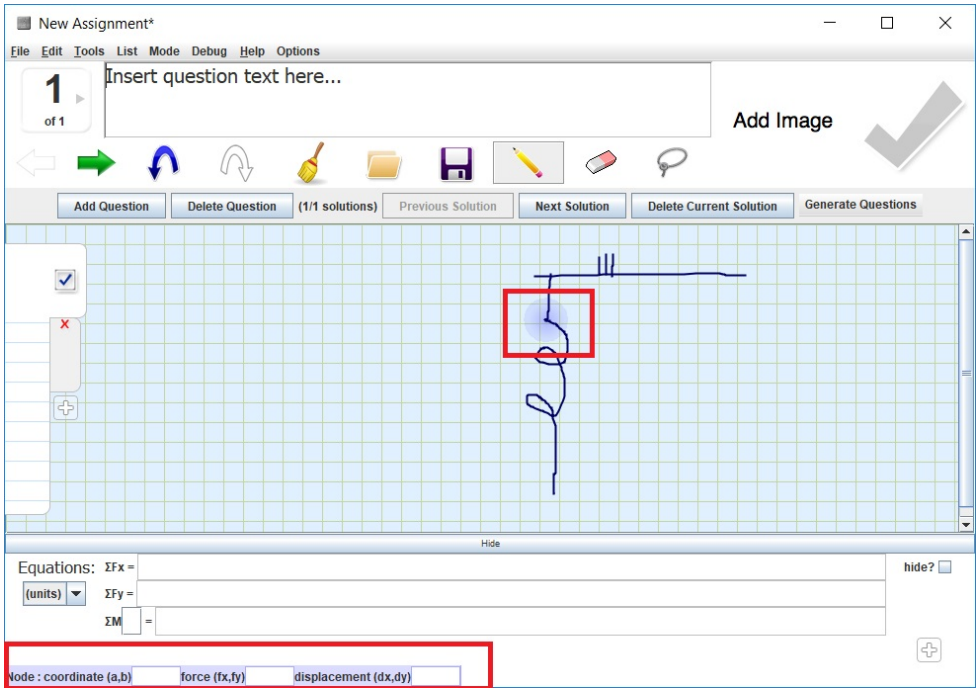


Figure 4.4: Force displacement panel has only three text area for coordinate, force and displacement of the node which is in working panel

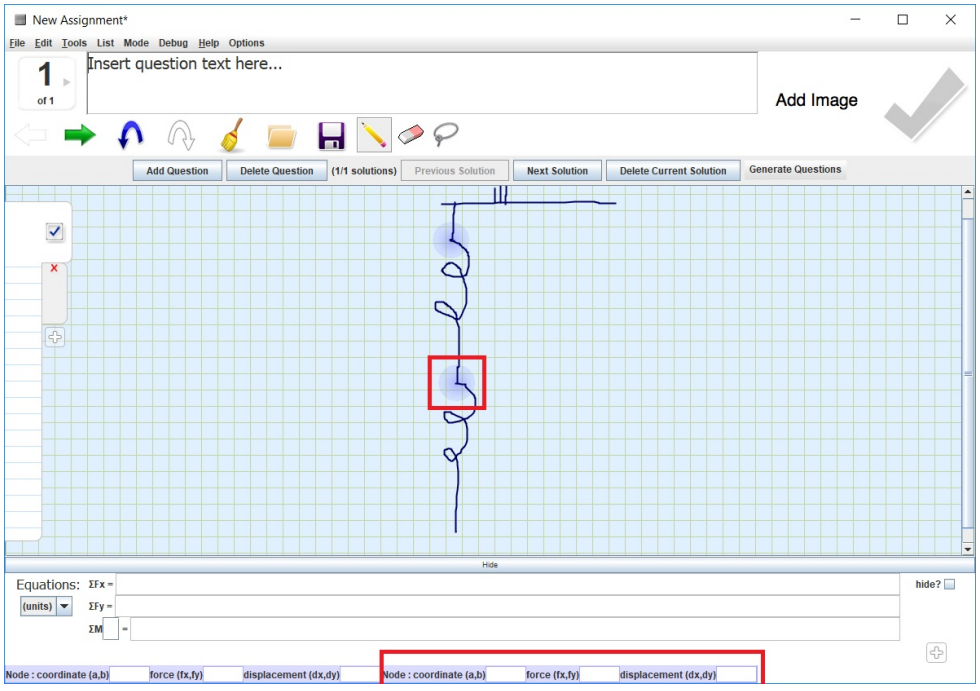


Figure 4.5: Force displacement panel has now two input panel for each node in working panel.

Mechanix can now generate stiffness for members of a truss to guarantee a unique solution for force displacement balancing equation. We will discuss in this chapter about how system modules and allocates the stiffness information. Stiffness information can come from user or system generates itself. We also discuss how to generate a solution for the given problem. If the system has a valid problem then it is ready for the user to generate new questions by using this initial template. First, we will discuss how to check if the user entered information is valid and generate a solution for the given problem. Next, we will discuss how to generate stiffness for each member which are not defined by the user so that it can generate a valid problem.

4.4.1 Apply Steel Properties

To define steel qualities, element thickness, cross-sectional widths, the moment of inertia Mechanix introduces a resource package in this research according to Eurocode 3 (or EC3) [29] and proposed a model by Max Hultman [1, 60, 28] (see appendix). During finding stiffness of a truss element this system allocates steel material properties for a problem. It assigns steel quality first to the truss element Ele_i since the yield strength of the steel depends on steel quality. Five different steel qualities are used, namely S 235, S 275, S 355, S 420 and S 460. The numbers represent the $yield\ strength^2$, f_y in N/mm^2. If the yield strength exceeds any of the numbers mentioned above, plastic deformation or even fractures will occur in the truss. An object of $DirectStiffnessQuestion$ is created during recognition process and it assigns values of $constantElementThickness$, $constantMomentOfInertia$, $constantCrossSectionalwidth$. This process chooses a steel quality randomly from the given set in the resource package. Then it assigns element thickness, the moment of inertia, inner width by EC3 from resource package.
Below, we show how we set thickness, cross-sectional width and moment of inertia for all elements followed by [1] in $DirectStiffnessQuestion$ -

1. $constantElementThickness_{Ele_i} =$
   $1e-3 * thickness[newRandom().nextInt(thickness.length)]]$

2. $constantCrossSectionalwidth_{Ele_i} = 1e-3 * crossSectionalWidth[$
   $newRandom().nextInt(crossSectionalWidth.length)]$

3. $constantMomentOfInertia_{Ele_i} = 1e-8 * momentOfInertia[newRandom()$
   $.nextInt(momentOfInertia.length)]$

Next it calculates the cross-sectional area of Ele_i by

$Ele_i\, CrossSectionArea = Ele_i\, crossSectionalWidth^2 - Ele_i\, Thickness^2$

Then it finally assigns the stiffness for this element Ele_i

$Ele_k = \frac{SteelElasticity * Ele_i\, CrossSectionArea}{Length\, of\, Ele_i}$

We consider the steel density is 7850 unit and steel elasticity is 210.

In the next sections, we will see how these properties are extracted from a known or unknown stiffness of a truss member. Later we can see how these properties have been used for different purposes in adding constraints to form a structural problem.

4.4.2 Designing Structure by Adding Constraints

Before going to discuss truss topology optimization we go back to our original research problem. One of our research problems is to generate questions for students and reduce teacher's/TA's working load to conduct a large class. This research will help to create a large question database for structural problems. The aspect to verify student's answer for auto-generated questions by Mechanix is not included in this research. Our aim is to provide a solution of the auto-generated question so that Mechanix can compare solution with student's solution in future. On basis of comparison, Mechanix will be able to correct student's solution. And it will also provide a few more problems to students with the same or different difficulty level. This interaction process with Mechanix and students can help students to improve their knowledge not only on the subject but also helps to increase

the ability to solve problems and the ability to understand the structure formations. Since we are working on a platform where student and teacher interaction is very important, therefore we assign some constraints and features that can help Mechanix later in this area. One of the most important section is solution checking and provide a real-time feedback for students by using validation rules. It will also focus on student's failures and strengths. We consider a situation where student's solution does not match with the solution of an auto-generated problem, Mechanix will be able to provide a similar type of problem with same difficulty level or it can provide a problem with added difficulty compared to the previous problem if a student gets the corrected solution. In both cases, we need to match the student's solution with the auto-generated problem. One guaranteed way to match a student's solution with that of an auto-generated question is to generate a problem which has a unique solution. If a problem has many solutions then it requires extra computations to verify a student's answer. In this research, we focus on problems with a unique solution. To reduce the extra computation, we ask teachers/TAs to enter valid boundary conditions for their drawn structure which have a unique solution. This Mechanix has some validation rules to assist a teacher/TA during defining initial problems. It helps them to know if their initial problem has many solutions or no solution. To reduce computation load, the system can not run any numerical operation until users request to generate a question from an initial problem. The validation rules come during the process of generating a problem. Therefore before formulating initial problem user needs to make sure that the given problem has a unique solution. Later we see that we use this uniqueness as a constraint in problem formulation.

Now we consider a scenario where the user forgets to add stiffness to some members or all truss members knowingly or unknowingly. In that case, the system needs to assign stiffness to these truss members from its end, otherwise, the system will not compute the stiffness matrix of the given truss by using direct stiffness method. If such a case arrives the

computational load can be really high and we do not allow this situation. Our system helps users to formulate their problem without knowing about stiffness. Stiffness assignment needs to analyze material properties and boundary conditions. A general numerical solver can fail to find a solution from a non-singular stiffness matrix. Since the system needs a unique solution from the initial problem, therefore it needs an appropriate approach to assign stiffness to a member of a truss. In the next section, we see how we assign stiffness.

4.4.2.1 Stiffness Allocation

In this section, we discuss how stiffness value of a truss member depends on other members and how current Mechanix solve the issue related to stiffness assignment. Consider the problem equation from Henri P. Gavin's study[2]:

$$[K]d = p \tag{4.1a}$$

where $[K]$ is the stiffness matrix, p and d are the set of forces and displacements applied at a set of coordinates on a structure. The structural stiffness matrix of a truss with a single member can be presented as

$$[K] = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}$$

let

$$d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

[2] `http://people.duke.edu/~hpgavin/cee421/matrix.pdf`

and

$$p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

The stiffness matrix represents a set of two equations with two unknowns, d_1 and d_2.

$$K_{11}d_1 + K_{12}d_2 = p_1 \tag{4.2a}$$

$$K_{21}d_1 + K_{22}d_2 = p_2 \tag{4.2b}$$

The equations (5.2a) & (5.2b) can be represented as

$$d_2 = -(K_{11}/K_{12})d_1 + (1/K_{12})p_1 \tag{4.3a}$$

$$d_2 = (K_{21}/K_{22})d_1 + (1/K_{22})p_2 \tag{4.3b}$$

These two lines are parallel if (K_{11}/K_{12}) = (K_{21}/K_{22}) and there is no unique solution for $\{d\}$ = $[K]^{-}1$ $\{p\}$. Now consider the problem where we have three springs

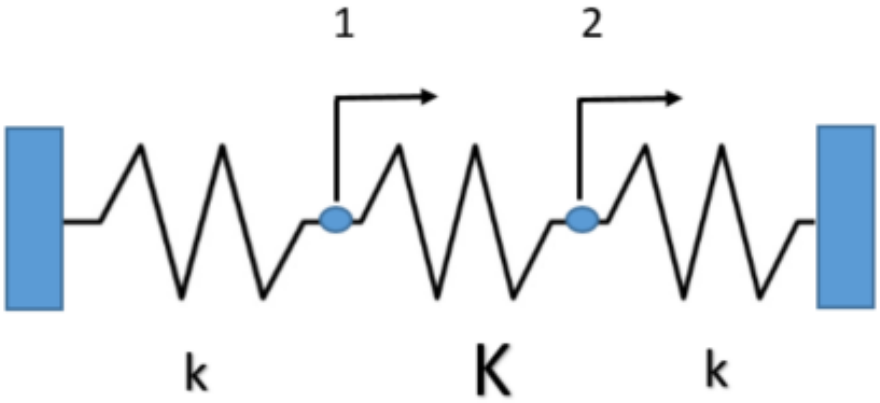


Figure 4.6: Sample illustration of spring stiffness.

The stiffness matrix of this problem is

$$[K] = \begin{bmatrix} K+k & -K \\ -K & K+k \end{bmatrix}$$

Now if $K >>> k$ then the stiffness matrix is close to

$$[K] = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}$$

This implies that $det([K])$ is close to zero and depends on the specified precision of the computation. Thus the problem can have many solutions. To assign a stiffness to a truss member where stiffness is missing, needs a comparison with other stiffness where they are present in the structure.

This study considers an objective function of sizing optimization for adding a constraint to design a truss problem. In size optimization, a design variable x, represents a structural thickness such as a distributed thickness or a cross-sectional area of a truss. The optimization problem is finding an optimal cross-sectional area that can minimize a physical quantity, while the equilibrium constraint has to be fulfilled. In our case, the objective function is to minimize the weight of the structure. [1, 61].

In this research we only consider few constraints to design the truss structure which has been used for three optimization processes as given by Max Hulmet.

In the following ways, we assign stiffness to a structure-

1. Consider that all member stiffness values are unknown initially. In this case, the system assigns $constantElementThickness$, $constantMomentOfInertia$ and $constantCrossSectionalwidth$ of $DirectStiffnessQuestion$ to $member\ Thickness$, $memberMomentOfInertia$ and $memberInnerWidth$ of each member of the truss respectively. It then calculates the cross-sectional area and stiffness

of each member. In this process, the system adds the same material properties to all members of the truss according to EC3. It chooses available properties from appendix A. Therefore all stiffness values belong within a range defined by EC3.
If $maxStiffness > p * minStiffness$ then p always less than 1000, where $maxStiffness$ and $minStiffness$ are maximum and minimum stiffness values of the truss respectively. Therefore system produces a non-singular stiffness matrix for this given structure if a user provides a valid boundary condition.

2. If a user provides stiffness values for multiple members and at least one member's stiffness is known. It then first compares between maximum stiffness and minimum stiffness from known stiffness values in following ways

   (a) It finds out maximum ($maxStiffness$) and minimum ($minStiffness$) stiffness from known $S = s_1, s_2, ..., s_m$ where s_i are know stiffness values, $i = 1, .., m$

   (b) If $maxStiffness > 999999 * minStiffness$ then it asks user to balance the stiffness properly.

   (c) If $maxStiffness < 999999 * minStiffness$ then it sets a constant for comparison $Stiff$ and $Stiff = minStiffness$.
   If $maxStiffness = minStiffness$ it then sets $Stiff = maxStiffness$.

   (d) It calculates stiffness $stiff_i$ for each unknown members as 1, $i = 1, .., n - m$ where n in the number of members in given structure.

   (e) If $Stiff > stiff_i$ and $Stiff > 999999 * stiff_i$ it then replace $stiff_i$ by $Stiff$ and updates member properties by using 3.

   (f) If $Stiff > 0$ and $stiff_i > 999999 * Stiff$ it then replace $stiff_i$ by $Stiff$ and updates member properties by using 3.

(g) Otherwise update member stiffness is $stiff_i$

This process guarantees that stiffness matrix of the structure is non-singular and therefore it has an unique solution.

3. If all stiffness values are known then system calculates $crossSectionArea$, $memberThickness$ of each member by using $constantElementThickness$, $constantMomentOfInertia$ and $constantCrossSectionalwidth$ of $DirectStiffnessQuestion$.

Since the system uses same $constantElementThickness$, $constantMomentOf Inertia$ and $constantCrossSectionalwidth$ of $DirectStiffnessQuestion$ for every members therefore it makes sure that structure is made by same material properties. Also all cases guarantee an unique solution of equation $\{d\}$ = $[K]^{-}1$ $\{p\}$. Therefore there is no failure case for our solver. This approach to allocating stiffness not only guarantees a unique solution for the base level problem but it also guarantees the uniqueness for a new problem which is auto-generated by Mechanix.

4.4.2.2 Add Other Constraints

We model our system to support another structure which is built by a combination of springs and beams. To create a spring & beam based structure which has a unique solution for equation $\{d\}$ = $[K]^{-}1$ $\{p\}$, it is important to add a few more constraints to our model. Mechanix can generate this type of structural problems by integrating these constraints and our new algorithms for question generation in this study. To support this model Mechanix just needs to add another solver which can solve the above equation for a spring & beam based structure. We work on this in future.

The main purpose of our study is to generate different structural problems for students. In the next chapter, we will see how we generate new truss questions from a base structural

problem. They are achieved by changing displacement and stiffness vectors, adding a new member in truss topology or removing existing members and nodes from current topology. In every case, there is a risk to generate a singular stiffness matrix. If this case arrives, the system then fails to complete the process of question generation and it needs to start a new process to form a problem which has a non-singular stiffness matrix. The system can fail multiple times to generate a single valid problem. To reduce this risk, we add some constraints to our module. We have already discussed the optimal cross-sectional area to create stiffness and light weighted truss in the previous section. In this section, we use a few more constraints which are important for the shape and topology optimization problem. Again, here we are not completely optimizing a structure. Therefore we do not need to run any optimization process. Formulating an optimization problem by adding constraints, we try to ensure that the newly created problem should have a unique solution for force displacement equilibrium equation through stiffness.

First, we see how to formulate our optimization problem with an objective function f, design variables x and the state variables y as described in [62] where f could be minimized or maximized. A typical objective function can be the stiffness or a volume of the structure [61]. The design variables x describes the design of the structure, it may represent the geometry. The state variables y represents the structural response which for example can be recognized as stress, strain or displacement.The state variable x depends on the design variable $y(x)$. The objective function is subjected to the design and state variable constraints to steer the optimization to a sought solution.

$$\begin{aligned} &\underset{x}{\text{minimize}} && f(x, y(x)) \\ &\text{subject to} \begin{cases} design\ constraint\ on\ x \\ state\ constraint\ on\ y(x) \\ equilibrium\ constraint \end{cases} \end{aligned}$$

A state function $g(y)$ that represents the state variables can be introduced, for example, a displacement in a certain direction. This state function can be incorporated as a constraint to the optimization task, where it is usually formulated such that $g(y) \leq 0$. Consider the case where g(y) is represented by a displacement vector $g(u(x))$ in a discrete finite element problem. To establish the state function, this requires that nodal displacement are solved for

$$u(x) = K(x)^{-1} f(x) \tag{4.4a}$$

The optimization formulation in equation (4.4.2.2) is called simultaneous formulation in comparison. Equation (4.4a) is usually solved by evaluating derivatives of f and g with respect to x. In this context, x will represent a geometrical feature. Based on what geometrical feature that is parameterized, the structural optimization problem can be classified into size, shape and topology [61]. A multi-objective optimization can be done with respect to multiple different objective functions [1, 63, 64, 65, 66, 67]. We use topology optimization to find the best inner connectivity of the members. In question generation, we will describe how we use to search two nodes to add a member using the same methodology of topology optimization.

We add the same important constraints in our model as defined by Max Hultman [1]:

1. **Fabricational:** To have a practical application of the structure, a feasible truss must

consist of elements of available dimensions. We use same square hollow profile which is taken from Budapest University of Technology and Economics [68]. A hot finished profile with a square hollow section has been created by letting hot steel material pass through rolls that gives the bar its intended shape and dimensions. Afterwards it is left to cool down, and depending on the element thickness, the different parts might cool down at a different rate, creating built in stresses in the element. We applied bulking resistance on the every truss elements by satisfying the following conditions-

$\frac{N_{Ed}}{N_{b,Rd}} < 1.0$

where N_{Ed} is the design value of the compression force, $N_{b,Rd}$ is the design buckling resistance of the compression member.

$N_{b,Rd}$ should be taken as :

$N_{b,Rd} = \frac{\chi A f_y}{\gamma_{M1}} \, for \, Class \, 1, 2, 3 \, cross - sections$

$N_{b,Rd} = \frac{\chi A_{eff} f_y}{\gamma_{M1}} \, for \, Class \, 4 \, cross - sections$

where χ is reduction factor for the relevant buckling mode, A is the cross-sectional area, A_{eff} is the effective cross-sectional area,

γ_{M1} is a partial factor for instability resistance (the recommendation is γ_{M1} = 1.0 for buildings and f_y is the yield strength in $\frac{N}{mm^2}$).

2. **Basic Nodes:** In this system, a user provides the coordinates of all basic nodes (i.e. nodes where there is either a support, a load and a joint of two truss members) in 2D. The given structure and auto-generated structures must have the basic nodes to be feasible. In recognition model, a basic node is $SpringNode$.

3. Nodal Displacements: The displacement is bounded within a limit [1], and the limit depends on the span width of the structure, e.g. $\delta_{max} = \frac{L}{300}$. In general the maximum deflection is chosen from $[\frac{L}{500}, \frac{L}{300}]$. During generating a new problem we choose

the maximum deflection from the interval [0, $\frac{L_x}{250}$] for horizon moves and [0, $\frac{L_y}{250}$] for vertical moves, where L_x and L_y are the x-directional and y-directional maximum span. In section 5.1.1, we describe how we use a modified version of this limit to generate different problems.

4. **Constructability:** Satisfying deflection and the element stresses which are within limits given in EC3 are not a necessary condition of a given or a generated truss to be feasible. The resource has to be given some additional constructability constraints. This is taken into consideration by not allowing two or more elements to have both their nodes in common and two nodes cannot exist in the very same place. Furthermore, to avoid having infinite elements stuck in any of the nodes, elements are not allowed to start and end up in the very same node. A violation of any of these constraints will result in a penalty.

5. **Number of Truss Elements:** The number of truss elements in a truss system are dependent on the number of nodes in the structure. The number of elements, m is given by the relation

   $m = 2n - 3$, where n is the number of nodes.

   The number of free elements, $nevt$ in the structure are chosen from the interval $0 \leq nevt \leq \binom{n}{2} - m$, where n is the number of nodes. To increase the probability of generating a feasible solution and reduce the computational complexity for students (undergraduate) we minimize the number of free elements. The resource allocation process follows the proposed guideline of Max Hultman [1] and the proposed number of free elements correspond to about one-tenth of $\binom{n}{2} - m$.

6. **Number of Allowed Loads** We follow following ruler-

   $n = sn + ln$

   n = number of nodes in structure, sn = number of nodes for support, ln = number

of nodes where user applies loads.

System has a validation rule to assist user to remove unnecessary loads from force-displacement input panel. if it finds $n < (sn + ln)$ and $ln > sn$ it removes some extra loads (f_x, f_y) from some spring nodes except those nodes which are connected to the shapes for clamp supports and loads.

Number of nodes	Number of free elements
6	1
7	1
8	2
9	2
10	3
11	4
12	5
13	6
14	7
15	8
16	9
17	10
18	12
19	14
20	15

Table 4.1: The proposed number of free elements

5. QUESTION GENERATION

In this chapter, we discuss all important methods which are used to generate questions and different structures from a base level structural problem. We consider the base level structure as our problem template. We can generate new problems by changing the boundary conditions, changing stiffness, adding a new truss member, and removing truss members and nodes from the given structural problem. We already discussed assigning stiffness in Section 4.4.2.1. We discuss the remaining methods here and provide some examples of the implementation in Mechanix.

5.1 Generate New Question by Adding New Boundary Conditions and Stiffness

The boundary conditions (force and displacement) play a major role on question generation and depend on the given stiffness of the truss members. By changing boundary conditions and stiffness, we can create several different questions on the same structure. In Chapter 4 we discussed why Mechanix has a stable structure during question generation by adding some constraints on the base problem. Instructors can create a set of questions related to local and global displacements, reaction forces, and stiffness matrices by using our new question generation module. Each question should have a unique solution that can be compared with a student's answer. In this section, we discuss the important steps to generate questions by changing displacement and stiffness.

Mechanix can create an infinite number of valid questions from a given problem template by using our methods. The system does not need to store all the problems. The instructor only needs to store those questions when he/she wants to change the structure, but not boundary conditions and stiffness. Therefore, Mechanix can create distinct real-time problems for students in a manner that saves space on the database and server. It lets a single question be extrapolated into a large number of additional questions without the

instructor having to make a question for every one, saving the instructor a lot of time.

5.1.1 Create New Question by Changing Displacement

We discuss important constraints on displacement in Section 1; here we modify that constraint by lowering the limit of the displacement. Our new method first identifies the structural composition. Let us assume all nodes of this structure are parallel to x-axis or y-axis. The new question generation method can easily find if they are parallel to any of the axes and then change the user's provided nodal displacement by choosing a value $\delta_x \in [0, \frac{L_x}{250}]$ to change x-displacement or $\delta_y \in [0, \frac{L_y}{250}]$ to change y-displacement. It can change both x and y displacements if other coordinates of the nodes do not lie either on the x-axis or y-axis in a structure which is parallel to any axis. In this case, we add another new limit on displacement: the new maximum displacement is $\delta_{max} = \frac{L}{500}$ where L is the maximum span of x or y depends on which coordinates are constant in the structure. If the structure is not parallel to any axis, then it updates both displacements by choosing $\delta_x \in [0, \frac{L_y}{250}]$ and $\delta_y \in [0, \frac{L_x}{250}]$.

The specific cases the system considers when changing nodal displacement are listed below.

1. The structure is parallel to x-axis, and y-coordinates are zero in every node. In this case, the system considers it as one-dimensional problem. Therefore the displacements are towards x direction. A maximum displacement δ_x towards x-direction is chosen from $\delta_x \in [0, \frac{L_x}{250}]$. System can substitute given nodal displacement value by δ_x.

2. The structure is parallel to x-axis and y-coordinates are not zero in every node. Since this is a two-dimensional problem, the problem can have displacements towards x-direction or y-direction or both directions. In this case, The maximum displacements are chosen in the following ways:

(a) $\delta_x \in [0, \frac{L_x}{250}]$ only for x-directional displacement change by δ_x

(b) $\delta_y \in [0, \frac{L_x}{500}]$ only for y-directional change by δ_y

(c) $\delta_x \in [0, \frac{L_x}{250}]$ and $\delta_y \in [0, \frac{L_x}{500}]$ for both directional displacement change by (δ_x, δ_y)

3. The structure is parallel to y-axis and x-coordinates are zero in every node. In this case, the system considers it as a one-dimensional problem. Therefore the displacements should be towards y direction. A maximum displacement y towards y-direction is chosen from $\delta_y \in [0, \frac{L_y}{250}]$. System substitutes a given displacement value by δ_y.

4. The structure is parallel to y-axis and x-coordinates are not zero in every node. Since this is a two-dimensional problem, the problem can have displacements towards x-direction or y-direction or both directions. In this case, The maximum displacements are chosen in the following ways:

   (a) $\delta_x \in [0, \frac{L_y}{500}]$ only for x-directional displacement change by δ_x

   (b) $\delta_y \in [0, \frac{L_y}{250}]$ only for y-directional change by δ_y

   (c) $\delta_x \in [0, \frac{L_y}{500}]$ and $\delta_y \in [0, \frac{L_y}{250}]$ for both directional displacement change by (δ_x, δ_y)

5. The structure is not either parallel to x-axis nor y-axis. Therefore the problem can have displacements towards x-direction or y-direction or both directions. In this case, The maximum displacements are chosen in the following ways:

   (a) $\delta_x \in [0, \frac{L_y}{250}]$ only for x-directional displacement change by δ_x

   (b) $\delta_y \in [0, \frac{L_x}{250}]$ only for y-directional change by δ_y

(c) $\delta_x \in [0, \frac{L_y}{250}]$ and $y \in [0, \frac{L_x}{250}]$ for both directional displacement change by (δ_x, δ_y)

The main steps to generate a new question by updating a given nodal displacement condition are as follows:

1. Clone the current question to a new a $DirectStiffnessQuestion$, named as $newQuestion$.

2. Determine the $SpringNode$ where the instructor entered a displacement condition, $changeIndex =$ position of this $SpringNode$ in spring system.

3. Determine if the structure is parallel to x-axis, y-axis, or not parallel to any axes from nodal coordinates.

4. Depends on returning value of step 2, evaluate the new nodal displacement condition by using the previously-mentioned cases in 5.1.1.

   (a) If the structure is parallel to x-axis

      i. If case 1 is $true$ and returns only δ_x then update $displacementBoundaryCondInfo$ of $newQuestion$ by new $double[](double)(NodeLabel), 1, \delta_x)$

      ii. If case 2a is $true$ and returns only δ_x then update $displacementBoundaryCondInfo$ of $newQuestion$ by new $double[](double)(NodeLabel), 1, \delta_x)$

      iii. If case 2b is $true$ and returns only δ_y then update $displacementBoundaryCondInfo$ of $newQuestion$ by new $double[](double)(NodeLabel), 2, \delta_y)$

iv. If case 2c is $true$ and returns only (δ_x, δ_y) then update $displacementBoundaryCondInfo$ of $newQuestion$ by new $double[](double)(NodeLabel), 1, \delta_x)$ and $double[](double)(NodeLabel), 2, \delta_y)$

(b) If the structure is parallel to y-axis

i. If case 3 is $true$ and returns only δ_y then update $displacementBoundaryCondInfo$ of $newQuestion$ by new $double[](double)(NodeLabel), 1, \delta_x)$

ii. If case 4a is $true$ and returns only δ_x then repeat 4(a)ii

iii. If case 4b is $true$ and returns only δ_y then repeat 4(a)iii

iv. If case 4c is $true$ and returns only (δ_x, δ_y) then repeat 4(a)iv

(c) If case 5 is true

i. If case 5a is $true$ and returns only δ_x then repeat 4(a)ii

ii. If case 5b is $true$ and returns only δ_y then repeat 4(a)iii

iii. If case 5c is $true$ and returns only (δ_x, δ_y) then repeat 4(a)iv

5. Update the attribute of the $Point\ p$ of $SpringNode$ for "$displacements$".

(a) If previous step 4 updates only $displacementBoundaryCondInfo$ of $newQuestion$ for x-direction, it then updates the attribute for x-directional displacement and leaves old y-directional displacement as is, e.g., displacement = (δ_x, old_value).

(b) If step 4 updates only $displacementBoundaryCondInfo$ of $newQuestion$ for y-direction, it then updates the attribute for y-directional displacement and leaves old x-directional displacement as is, e.g., displacement = (old_value, δ_y).

(c) If step 4 updates $displacementBoundaryCondInfo$ of $newQuestion$ for both x and y directions, it then updates the attribute for x-directional and y-directional displacements, e.g., displacement = (δ_x, δ_y).

6. Find out a node where nodal displacement information is empty or unknown. Set question of $newQuestion$ by asking displacement info at this node.

7. If 6 fails to find a node where displacement information is empty or unknown then set a question for any arbitrary node.

8. Run solver to set the answer for this new question.

We consider the difficulty level of this newly-generated question to be the same as the base problem, although the difficulty setting is not currently active or tested.

5.1.2 Create New Question by Changing Stiffness

We apply the same methodology to creating a new question by changing the stiffness vector as we discussed in Chapter 4.4.2.1. Here the system does not change the given boundary conditions on displacement and force; it just reallocates stiffness to all those members where stiffness was not assigned by the instructor. Assigning stiffness to a member depends on the stiffness of other members. Since the system has already balanced the stiffness vector s during the initial problem formation, $any\ Stiff_i \notin s$ satisfies $Stiff_i > 999999 * Stiff_j$ where $i \neq j$

1. Select a member k where the user did not define stiffness in initial problem template. Otherwise, select a member k randomly.

2. Change the cross-sectional area of member k and calculate the new stiffness $Stiff_k$ for member k.

3. If newly generated $Stiff_k$ is less than 999999 times of $minStiffness$ of s then choose a random value from $[\frac{minStiffness}{1000}, maxStiffness]$ if $maxStiffness \neq minStiffness$ in s. Otherwise, use $Stiff_k$.

4. If $maxStiffness = minStiffness$ and $Stiff_k > 999999 * maxStiffness$ then pick a random stiffness from $[\frac{maxStiffness}{1000}]$, $\frac{maxStiffness}{500}]$.

The difficulty level of this newly generated question is the same as the initial question.

5.2 Generate New Question by Changing the Structure

In this section, we see how Mechanix is able to create a different structure from the initial template using a growth strategy. In this strategy, the system grows the ground structure method to add a member in the current truss topology [69], and we develop a new algorithm to reduce the truss topology by using the same concept as the ground structure method. Since in both cases, it creates two different structures, these new problems need to be saved in a database or file server to reuse them as a base template. The system uses the same constraints in both methods that it develops during forming the base structure; therefore, newly-generated problems through these methods always have a unique solution.

5.2.1 Add a New Truss Member and Generate New Structural Problem

Researchers use different kind of optimization methods to reduce or add a member in truss topology [70, 71, 72, 73, 74, 1]. In this section, we modify the given truss topology by removing nodes, members, or adding a new truss member. Through this process, the system is able to generate a different base template, which can be used again by the instructor to create a wholly-different set of questions.

5.2.2 Add New Truss Element Between Two Nodes

We already optimized the volume of the problem by reducing cross-sectional areas of each member if a user did not provide cross-sectional info for each of its members through stiffness. We apply the growth strategies of T. Hagisita & M. Ohsaki for adding a member [69]. The ground structure method [75, 76] can find the optimal solutions, however it is a tedious process to prepare the densest truss with a bar for every pair of nodes [72, 77, 78]. Because the initial truss system must have valid boundary conditions, the given problem always has a solution, and we know the displacements of each node. To implement Growth Strategy Method 1 of T. Hagisita & M. Ohsaki, the system needs to calculate the absolute value of the potential strain of each candidate member and scale by unit volume. This method adds the most-strained member to the new graph to create a question.

$$Find\ k \in candidatebars, which\ maximize\ \frac{1}{l_k}\,|\overline{\eta_{\mathrm{k}}}| \tag{5.1a}$$

$$\overline{\eta_{\mathrm{k}}} = \frac{{u_{ij}}^T d_{ij}}{||x_i - x_j||}, \tag{5.1b}$$

$where\ u_{ij} = u_j - u_i\ and\ d_{ij} = \frac{x_j - x_i}{||xi - xj||}$

Now

$$(K + K_k)\delta u_k = K_k u, \tag{5.2a}$$

$$K_k = \frac{E_k}{L_k^2} b_k b_k^T, \tag{5.2b}$$

where $K \in R^{n\times n}$ is the global stiffness matrix, u is the nodal displacement vector before adding new candidate member k. E_k, l_k are Young's modulus [79] and length of the member k. $-K_k\delta u_k$ is the internal force after adding new candidate member k, and it

generates displacements δu_k. If we assign $E_k = 1$, the right hand side is equivalent to the potential strain multiplied by cross-sectional area $a_k = \frac{1}{l_k}$. Therefore, Growth Strategy Method 1 does not need an additional structural analysis since it does not directly evaluate the nodal displacements of the new truss structure after adding this new member. It only needs to evaluate the internal force from u without calculating δu_k.

As we mention in Section 5, the number of free elements can be chosen from interval $0 \leq nevt \leq \binom{n}{2} - m$ and the maximum number of elements of the structure is $\binom{n}{2}$. In this method, the system first checks if the number of nodes is more than 12 and if the existing elements have already reached to the maximum number. If so, then it stops to add a member and exits the method. If the maximum number is not reached, then it checks how many free elements can be added in this structure according to Table 4.4.2.2.

Before adding a free truss member, the system also checks if there exists an element between two nodes to avoid overlapping. Given the possibility to add x number of free elements, it adds only one element between two nodes using the previously-described Growth Strategy Method 1, where no truss element exists. If a system has fewer than 12 nodes, then it does not count the free elements. Else, in the case of 12 or more nodes, the system classifies this problem as “difficult," one which would require extensive time to solve without the aid of software tools like MATLAB [1] for computation. Therefore, adding another member increases the difficulty level for students. The system assigns a stiffness to its new members by picking a random value from $[minStiffness, maxStiffness]$, or only $maxStiffness$ if $maxStiffness = minStiffness$. It also updates other material properties of this new member which is basically a $SpringNetworkMember$. Once it updates the spring system of the new question, it then attaches the solution of the new problem along with this new question.

The important steps to add a new member in a given truss system are following:

[1] `https://www.mathworks.com/products/matlab.html`

1. Find out the possible candidate members.

2. Find out the most strained bar among candidate bars.

3. Get a new stiffness for this new member.

4. Create a new spring member. To create a new shape use $generateHelix$ method.

5. Update member list and generate new $SpringSystem$ by using $addJointBetweenSpringNodes$ method.

6. Increase difficulty level by 1.

7. Set solution.

Our approach adds a few methods in Mechanix that help our system to do the following important steps while adding a member:

1. **generateHelix:** We add this method to create a new, beautified helix shape between our candidate nodes where the system chooses to add an additional member by using Growth Strategy Method 1.

2. **AddJointBetweenTwoNodes:** This method helps to add a new $SNode$ to the graph of the spring system by a newly-created beautified helix. It also updates the relative nodes.

Adding a member increases the complexity thereby increases the difficulty level of the problem. For now, we add only one free element to increase the difficulty level, although in future development, it is possible to scale difficulty increasingly by adding more members.

5.2.3 Reduce Structure Method

The final method for question generation is based on removing free elements (zero-force members) from the base structure [1, 74, 80]. We add another new algorithm that can identify free elements and remove the connected nodes between zero-force members from the structure. Thus, if a system is able to remove a zero-member and a node, then we will yield a different truss topology. This helps not only to change the given structure but also create a new problem template which can be used for generation of other questions.

Technically, we are not removing all zero-member elements from the structure. This approach can help to form a base template which is not trivial, like one parallel to the x or y axes (depending on the base template). Even by removing a single zero-member, instructors can employ reduction of truss members with the growth strategy discussed in the previous section to easily create many templates for problem types.

We use the following two rules to identify zero-force members from base-level problems:

1. If the current node connects with only two members and the angle between them is not 180 degrees, then both members are zero-force members. Remove both member, and remove the current node.

2. If the current node connects with only three members and two of them are collinear, then the third one is a zero-force member. Remove this third member from member list, and keep the node.

The important steps in our algorithm are listed below.

1. Select each node which is not connected with support and no force applied.

2. If node is a joint of two members and it satisfies Rule 1, then add the node to $deleteNodes$ and both members to $deleteShapes$.

3. If node is a joint of three members and it satisfies Rule 2, then add the non-collinear member to $deleteShapes$.

4. Create a new $SpringSystem$ by using $reducedSketchByZeroElements$ method.

5. Create new $DirectStiffnessQuestion$ by using the newly created $SpringSystem$.

6. Set new difficulty level (new difficulty = old difficulty - number of deleted Nodes - number of deleted Shapes).

7. Set answer.

We create a new method, named as $reducedSketchByZeroElements$, which helps to rebuild a $SpringSystem$ for this new problem. This method removes all zero elements and nodes from the base template's $SpringSystem$. It also updates the current graph and nodes.

In the next chapter, we will show a few results about our system-generated problems from a base template.

6. EVALUATION

In this chapter, we discuss both the direct results of testing and running our updated version of Mechanix as well as feedback from a small user study. We have conducted a survey with faculty, teaching assistants, and postdoctoral researchers from multiple departments and different universities. Much of our evaluation is based on these results, including success and failure rates.

6.1 Evaluating Problem Generation

First, the instructor needs to create a base problem by using the spring network recognizer. Mechanix provides four options to create new problems and different base template. In the following subsections, we show how our users created different structural problems using our new features.

6.1.1 Steps to Create Base Template

1. **Draw a Spring-Based Structure in Mechanix:** A user can draw a truss structure in the creative working panel of Mechanix to make a spring-based problem. The structure must have a clamp support, at least one truss member (line or helix), and a load (closed shape or circle). To draw a load, the user needs to add a closed shape to the graph of the structure so that total area of the bounding box of the closed shape is less than 7000.0 (empirically selected). All shapes need to create an undirected connected graph to design a structure. Below, we add few examples where our system recognized all these shapes as a spring-based truss structure. We have three different spring-based structures in Figures 6.1, 6.2a, and 6.2b. We show a few more structure during generating new templates.

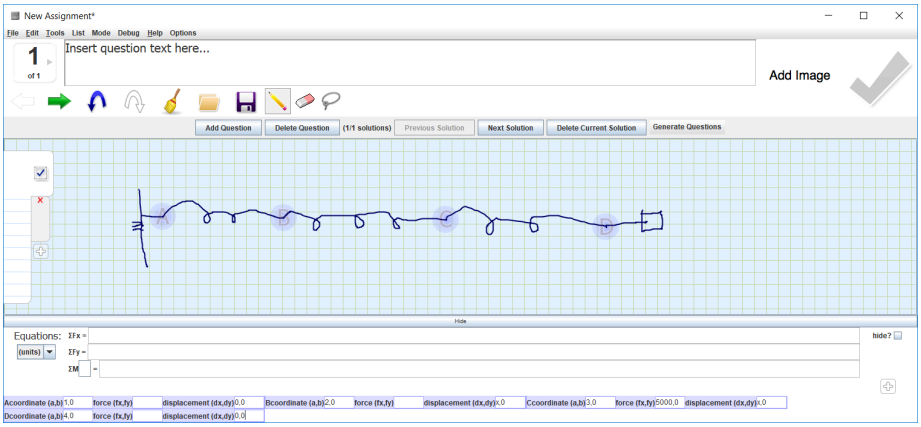
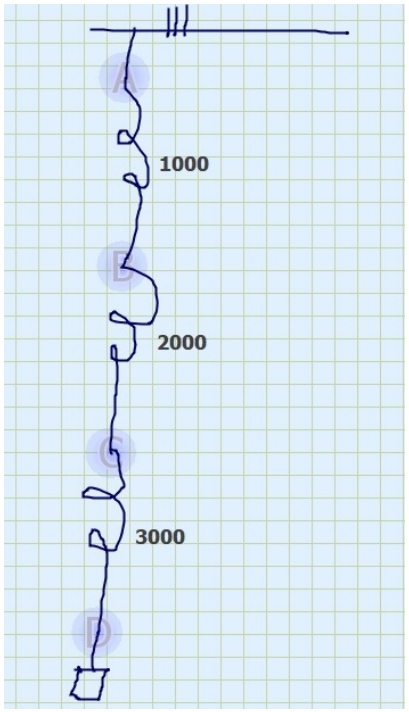
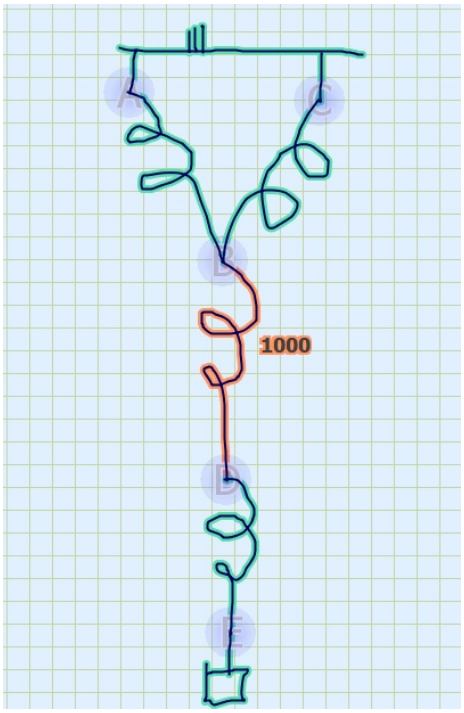


Figure 6.1: User 1 created a base structure without stiffness. Mechanix successfully recognized the structure



(a) User 1 drew a problem with three springs and assigned stiffness for all members



(b) User 2 drew a problem with four springs and assigned stiffness for one member

Figure 6.2: User 1 and User 2 drew two different structures and Mechanix successfully recognized them.

2. **Add Boundary Conditions for Drawn Structure in Mechanix:** An instructor can add boundary conditions in Mechanix through the force displacement panel. The coordinate is a mandatory field; without providing coordinate values for each node, Mechanix cannot create a base template. On the other hand, stiffness is optional because Mechanix can allocate stiffness as per Chapter 4.4.2.1. In Figures 6.3 and 6.4, we show how User 1 and User 2 added their boundary conditions through the force displacement panel.

3. **Add Stiffness for Members:** Users can assign a stiffness value of a member by just labeling the shape of the member. They can also assign stiffness to all members of the structure or few of them.

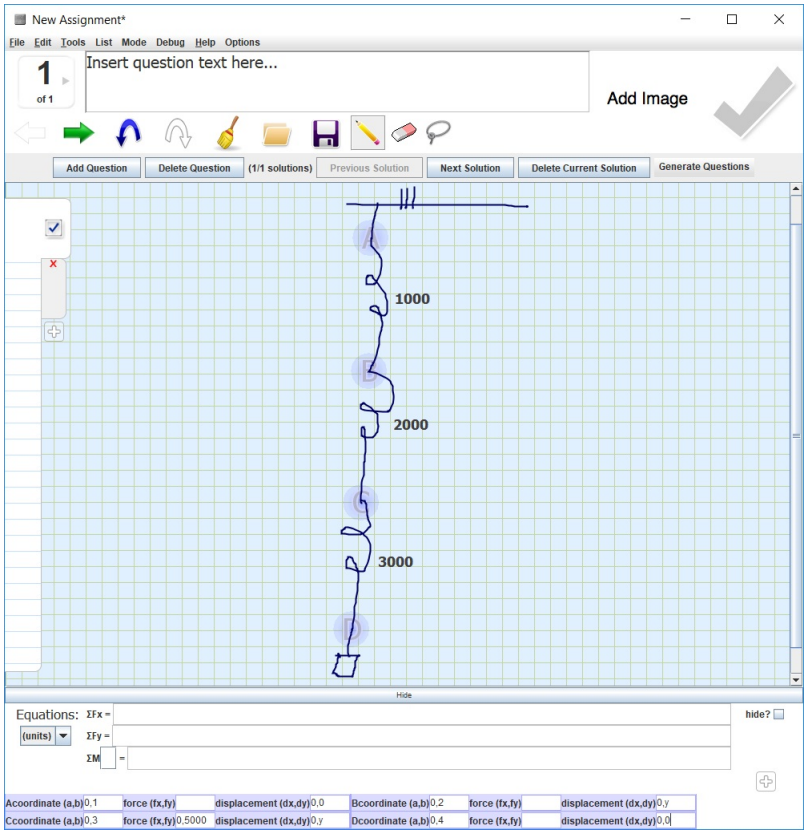


Figure 6.3: User 1 added stiffness for all members and provided boundary conditions for problem 6.2a

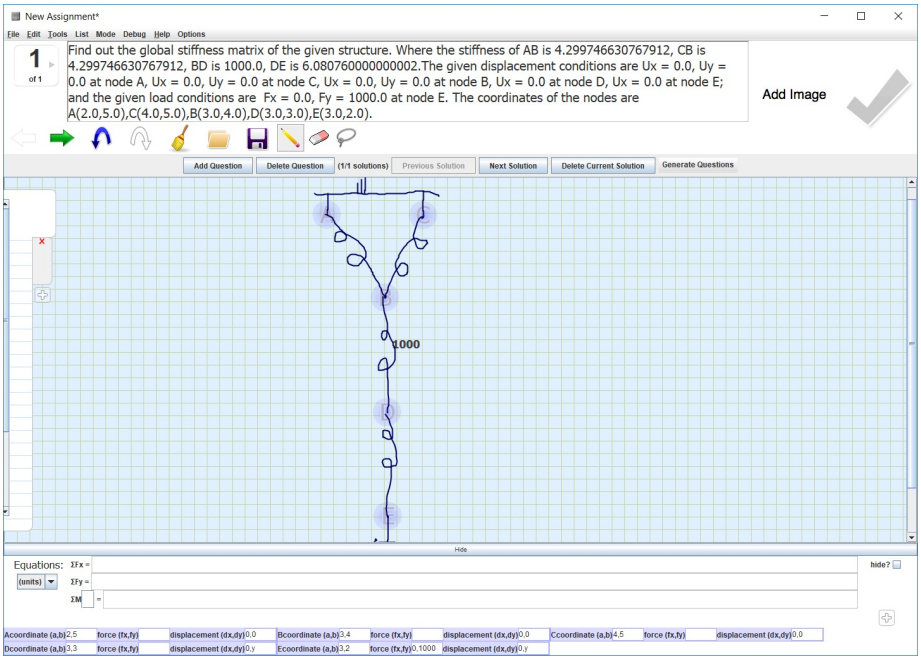
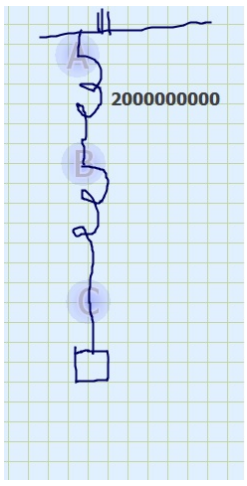
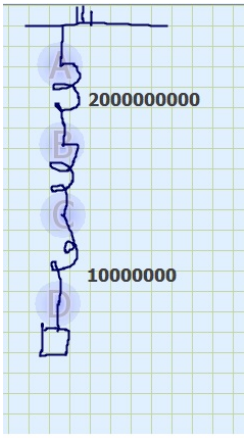


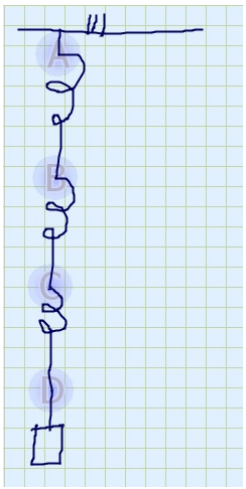
Figure 6.4: User 2 added stiffness for member BD and provided boundary conditions for problem 6.2b



(a) Maximum Stiffness



(b) Minimum Stiffness



(c) Calculated stiffness

Figure 6.5: In 6.5a, the maximum stiffness is 2.0E9 for BC; in 6.5b, it is 1.0E7 for BC; in 6.5c, the values are AB $\approx$ 2.499, BC $\approx$ 2.499, CD $\approx$ 2.499 to create a minimum weighted structure.

4. **Check Base Template for Creating an Initial Question:** We do not run our computational model until the user asks to generate a problem. Therefore, to check the initial information (coordinate, force, displacement, and stiffness) if users can create a base template, the user needs to select the “Create Stiffness" operation from Question Generation layout. Mechanix immediately creates an initial question from the base structure and boundary information if valid. Otherwise, it informs the user what prevented creation of a base template.

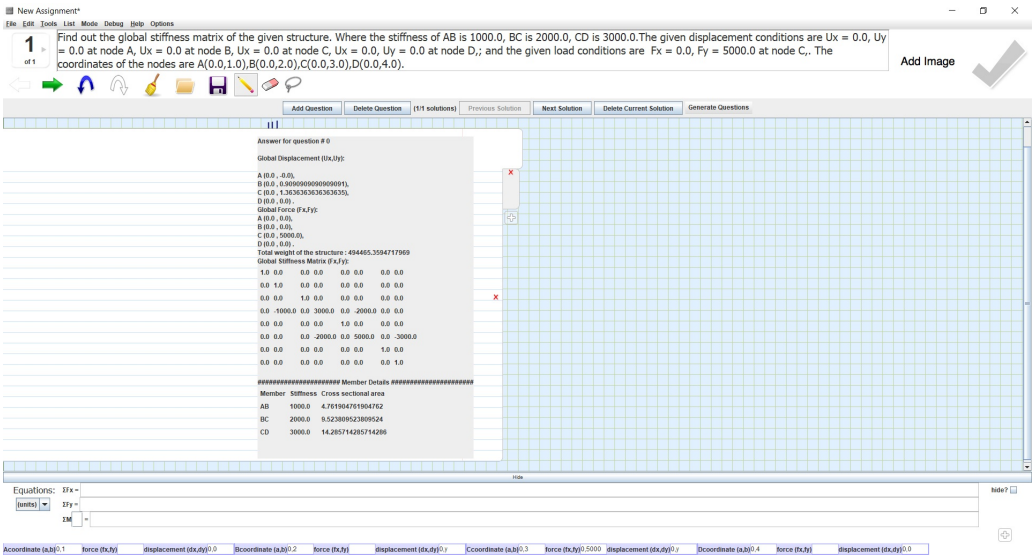


Figure 6.6: User 1 created a base template by asking stiffness related question from given problem in Figure 6.3. Mechanix provided the correct answer.

Figure 6.6 shows the generated problem from Figure 6.3. Figure 6.7 is the problem generated from User 2’s sketch in Figure 6.4. For this problem, Mechanix created the following base question and successfully generated a base template.

Computer Generated Question: `Find out the global stiffness matrix of the given structure. Where the stiffness of AB is 1.0665260467938322, CB is 1.0665260467938322, BD is 1000.0, DE is 1.5082955999999998. The given displacement conditions are Ux = 0.0, Uy = 0.0 at node A, Ux = 0.0, Uy = 0.0 at node C, Ux = 0.0, Uy = 0.0 at node B, Ux = 0.0 at node D, Ux = 0.0 at node E; and the given load conditions are Fx = 0.0,`

Fy = 1000.0 at node E. The coordinates of the nodes are A(2.0,5.0), C(4.0,5.0), B(3.0,4.0), D(3.0,3.0), E(3.0,2.0).

It assigned stiffness for AB, CB and DE. Mechanix also provided the correct answer for this problem.

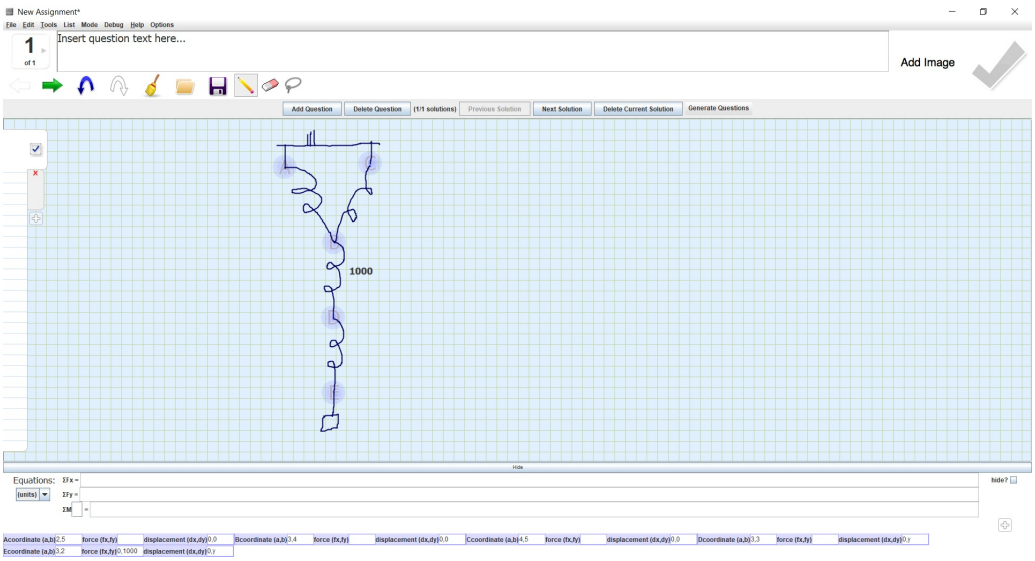


Figure 6.7: User 2 created a base template by asking stiffness related question from given problem in Figure 6.4

6.2 Steps and Examples to Generate New Problems from Base Template

In this section, we show how users generated problems successfully from previous base problems. A user can create problems from a base template by changing boundary conditions, stiffness, or creating a different base template.

1. **Create Problem by Changing Displacements:** User 2 generated a second problem from Figure 6.7 by using this method.

   Computer Generated Problem: Find out the global displacment at node D. Where the stiffness of AB is 1.0665260467938322, CB is 1.0665260467938322, BD is 1000.0, DE is 1.5082955999999998. The given displacement conditions are Ux = 0.0, Uy = 0.0 at node A, Ux = 0.0, Uy = 0.0 at node C, Ux = 0.0, Uy = 0.0 at node B, Ux = 0.0 at node D, Ux = 0.0, Uy = 4.058375213788885E-4 at node E;

and the given load conditions are Fx = 0.0, Fy = 1000.0 at node E. The coordinates of the nodes are A(2.0,5.0), C(4.0,5.0), B(3.0,4.0), D(3.0,3.0), E(3.0,2.0).

In the previous problem, U_x was 0 and U_y was undefined. Through this method, the new problem has changed the boundary condition at node E. The new displacement condition at node E is $U_x = 0$ and $U_y = 4.058375213788885E - 4$.

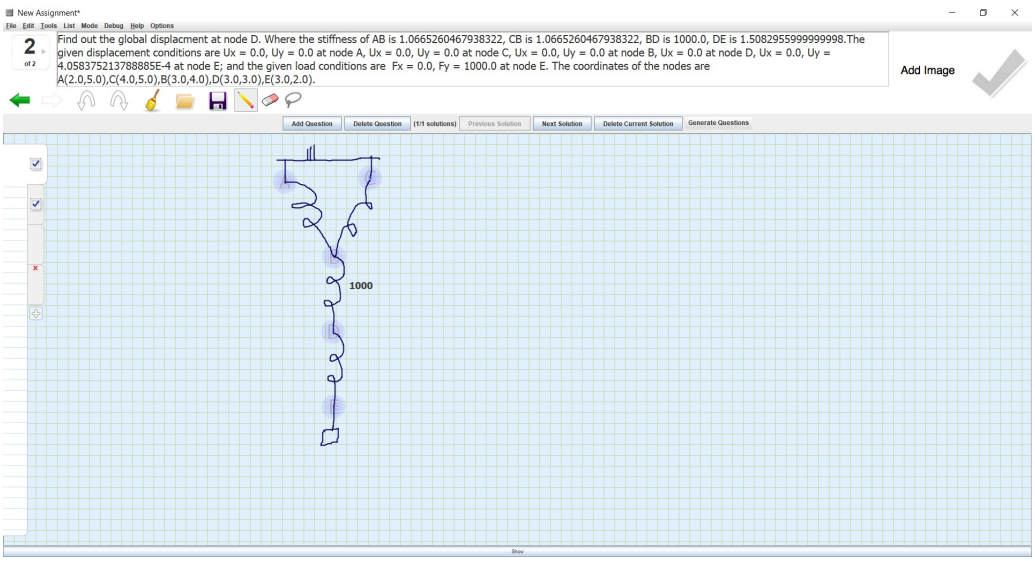


Figure 6.8: User 2 created a new problem by changing displacements from the given problem Figure 6.4

2. **Remove Members and Nodes from Base Template:** In Figure 6.9, an instructor drew an arbitrary structure with springs (both helix and line representations) and added boundary conditions through the force displacement panel. Our system successfully created the base problem and the following initial problem:

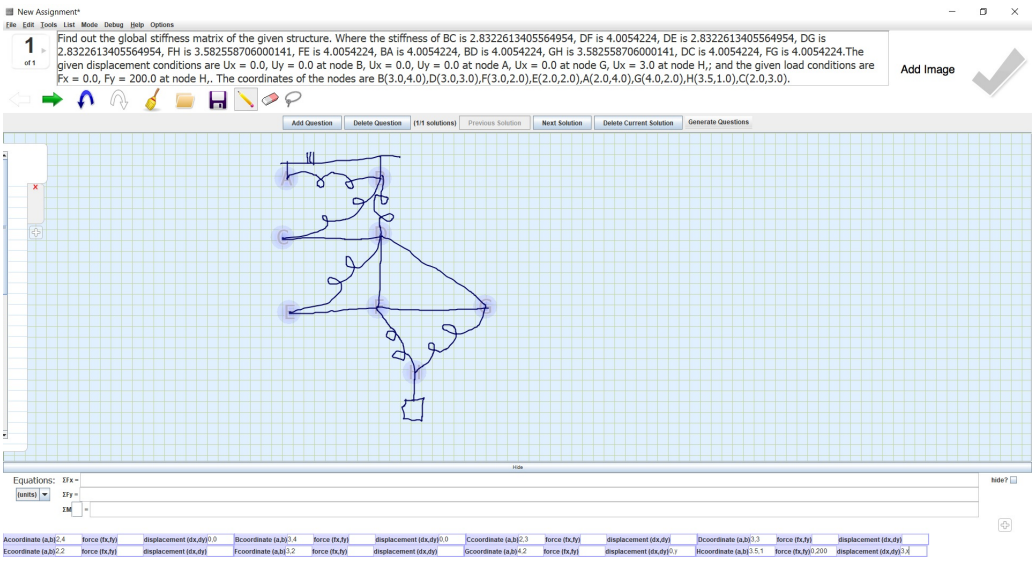


Figure 6.9: User 7 created an arbitrary base structure without stiffness.

Computer Generated Problem: `Find out the global stiffness matrix of the given structure. Where the stiffness of BC is 2.8322613405564954, DF is 4.0054224, DE is 2.8322613405564954, DG is 2.8322613405564954, FH is 3.582558706000141, FE is 4.0054224, BA is 4.0054224, BD is 4.0054224, GH is 3.582558706000141, DC is 4.0054224, FG is 4.0054224. The given displacement conditions are Ux = 0.0, Uy = 0.0 at node B, Ux = 0.0, Uy = 0.0 at node A, Ux = 0.0 at node G, Ux = 3.0 at node H; and the given load conditions are Fx = 0.0, Fy = 200.0 at node H. The coordinates of the nodes are B(3.0,4.0), D(3.0,3.0), F(3.0,2.0), E(2.0,2.0), A(2.0,4.0), G(4.0,2.0), H(3.5,1.0), C(2.0,3.0).`

In the next example, depicted in Figure 6.10, Mechanix successfully removed a few zero force members and their corresponding nodes and it created a new base problem template for the instructor with the following question:

Computer Generated Problem: `Calculate the force and displacement at each of the internal nodes Where the stiffness of BC is 2.2162224000000017, BE is 1.5671058876575263, CF is 1.9822495758630934, AD is 2.2162224000000017, AB is 2.2162224000000017, EF is 1.9822495758630934, CE is 2.2162224000000017. The given displacement conditions are Ux = 0.0, Uy = 0.0 at node A, Ux = 0.0, Uy = 0.0 at node D, Ux = 0.0 at node E, Ux = 3.0 at node F; and the given load conditions are Fx = 0.0, Fy = 200.0 at node F. The coordinates of the nodes are A(3.0,4.0), B(3.0,3.0), C(3.0,2.0), D(2.0,4.0), E(4.0,2.0), F(3.5,1.0).`

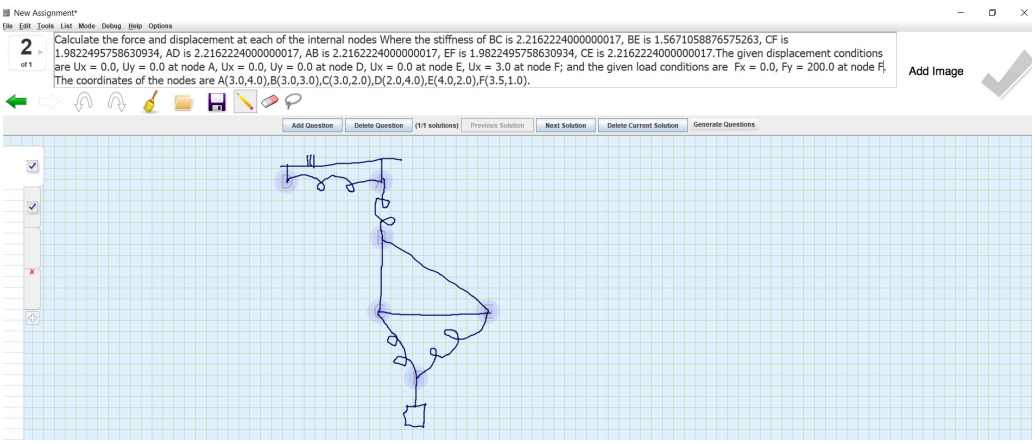


Figure 6.10: User 7 successfully created a new base template and problem by using our reduce method.

For both problems 6.9 and 6.10 system successfully created their correct answers.

3. **Add a New Member to a Base Template:** User 2 had created several questions from the last question (refer to Figure 6.4), and subsequently, User 2 added a member to the base template (refer to Figure 6.11). Mechanix successfully added new member EC and assigned a proper stiffness of 738.8560544987274 units as well as finding the solution.

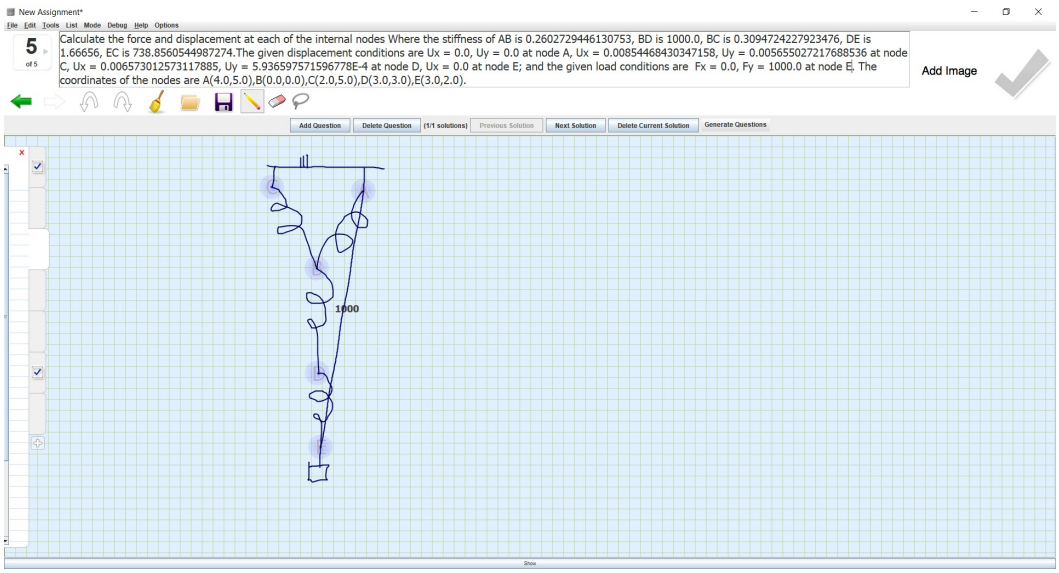


Figure 6.11: User 2 successfully created a new base template and problem by using our add method.

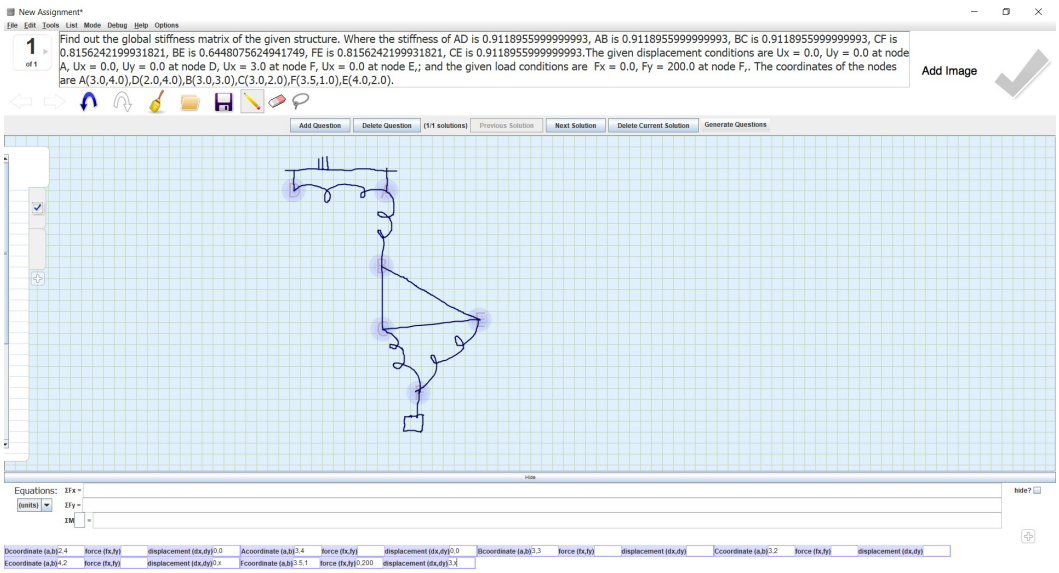


Figure 6.12: User 7 successfully created a new base template and problem by using reloading problem 6.10. Since the user asked to change the stiffness, it created a new problem 6.12

In Figure 6.12, User 7 reloaded the previous template and changed the stiffness, which created a new problem.

Computer Generated Problem: `Find out the global stiffness matrix of the given structure. Where the stiffness of AD is 0.9118955999999993, AB is 0.9118955999999993, BC is 0.9118955999999993, CF is 0.8156242199931821, BE is 0.6448075624941749, FE is 0.8156242199931821, CE is 0.9118955999999993. The given displacement conditions are Ux = 0.0, Uy = 0.0 at node A, Ux = 0.0, Uy = 0.0 at node D, Ux = 3.0 at node F, Ux = 0.0 at node E; and the given load conditions are Fx = 0.0, Fy = 200.0 at node F. The coordinates of the nodes are A(3.0,4.0), D(2.0,4.0), B(3.0,3.0), C(3.0,2.0), F(3.5,1.0), E(4.0,2.0).`

Next, the user modified Figure 6.12 by using the “Add a member” method, which successfully created another new problem shown in Figure 6.13.

Computer Generated Problem: `Calculate the force and displacement at each of the internal nodes Where the stiffness of AD is 0.9118955999999993, AB is 0.9118955999999993, BC is 0.9118955999999993, CF is 0.8156242199931821, BE is 0.6448075624941749,`

FE is 0.8156242199931821, CE is 0.9118955999999993, BD is 0.6843093817196116.
The given displacement conditions are Ux = 0.0, Uy = 0.0 at node A,
Ux = 0.0, Uy = 0.0 at node D, Ux = 3.0 at node F, Ux = 0.0 at node E; and
the given load conditions are Fx = 0.0, Fy = 200.0 at node F. The coordinates
of the nodes are A(3.0,4.0), D(2.0,4.0), B(3.0,3.0), C(3.0,2.0),
F(3.5,1.0), E(4.0,2.0).

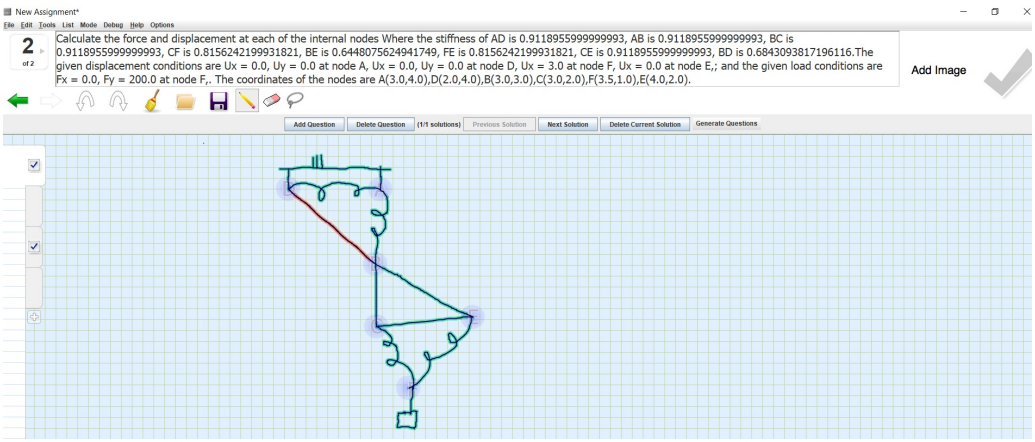


Figure 6.13: User 7 successfully created a new base template and problem using our reduce method.

Mechanix can create an infinite set of different problems from each of the problems shown in Figures 6.9, 6.10, and 6.13 by using change displacement and stiffness methods.

6.3 User study

We conducted a user study consisting of eleven participants made up of faculty and teaching assistants of courses that might make use of this software. The participants are from five different departments (Aerospace Engineering, Mechanical Engineering, Civil Engineering, Physics, and Mathematics). We chose these five departments since structural analysis and FEM are in their course curricula. We provided the participants with an introduction of Mechanix and its new feature for spring-based problem generation. Since all participants have a teaching background with mechanics or FEM, we did not have to provide any sample problems to the participants.

During this survey process, we asked the users to create a base level problem and generate new structural problems through Mechanix. Some users found spring-based questions to use online, and other users created questions from their own experience. Once Mechanix successfully generated questions, we asked them to solve those problems and check the correctness of the auto-generated answers. In most cases, the users checked only one or two answers for each set of auto-generated questions created from the base question. This is because answering each new question can be a time-consuming process (sometimes taking up to an hour depending on the problem complexity). After finishing these three steps (creating the base level problem template, generating questions, and checking solutions) we asked them to fill out a survey form. We divided our survey into two categories, quantitative and qualitative focused around evaluating the validity and value of the system. Participants spent an average of four hours each evaluating the software.

6.3.1 Quantitative Analysis

In our quantitative analysis, we asked the following questions:

1. Please rate the ease of creating a question.
2. Please rate the ease of solving the question.
3. How likely would you be to use this solution for giving students practice?
4. Would students benefit?
5. Did the auto-generated problem and solution appear correct?

For each question, users would rate their response on a Likert scale of one to five. Participants gave a very good rating to the validity of the auto-generated questions and their solutions, with a mean of 1.09 +/- 0.301 where 1 meant perfect solution. Note that because we did not ask the participants to solve every problem, the question focused on

the apparent correctness of the problem. Given rounding accuracy of the early system, the appearance confused a couple users, but in every case where the participants solved the problem, they verified correct answers.

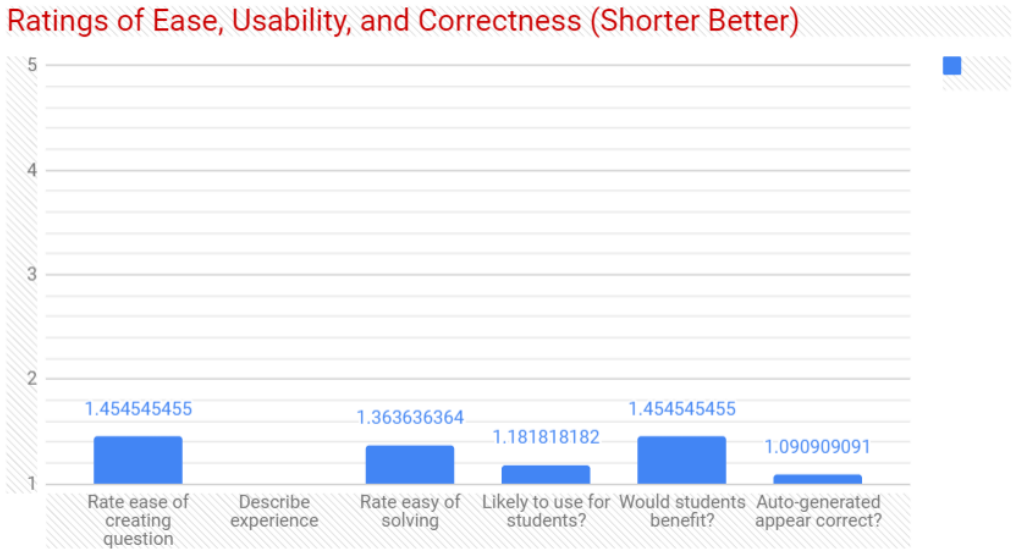


Figure 6.14: Mean participant ratings of individual aspects of question generation through Mechanix. Each of these questions required participants to respond using a 5-point Likert scale, with 1 being most positive and 5 being most negative.

Despite the small sample size imposed by the difficulty of obtaining faculty participants, the system received high scores in terms of ease and usability. The outcome of the ease of creating a question was matched with our expected result since we used Mechanix's original user interface which is not as user-friendly as development versions. We are expecting the system to receive better evaluations when the development version integrates the problem generation features. Overall, every participant agreed they would use this new feature for generating truss-based questions and students would definitely benefit from this application, supporting the value of such a tool. Figure 6.14 shows the average ratings from the participants.

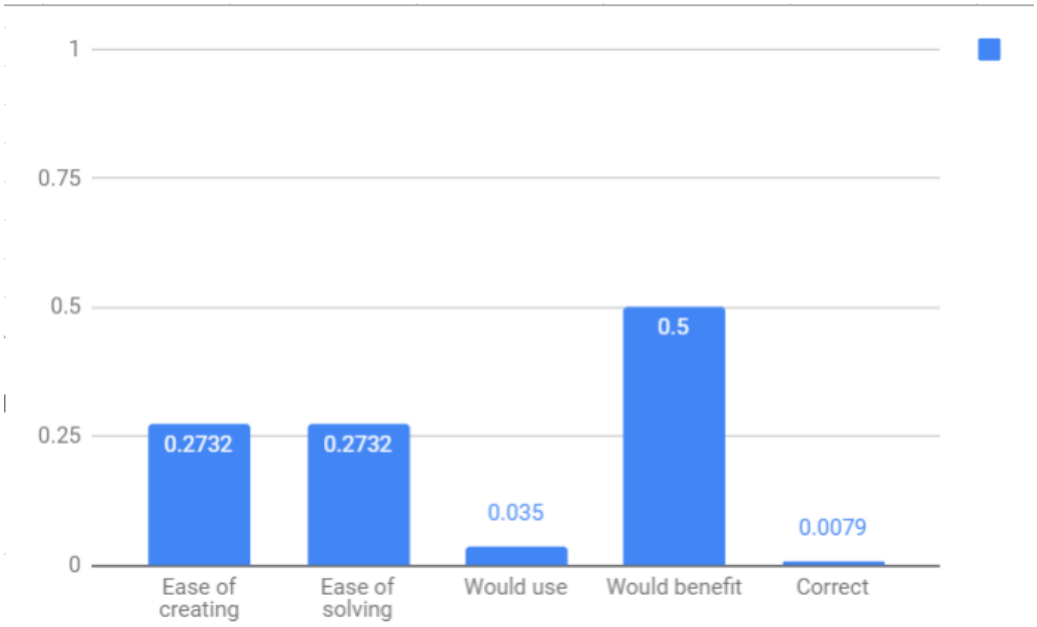


Figure 6.15: Proportion test (P-value): if proportion of 1 > 0.5 or not H: $P(1) = 0.5$ vs H_{alt} : $P(1) > 0.5$

We also performed a proportional-test, p-test, in order to determine the level of agreement among participants. The results show no statistically significant differences in the users' opinions, as none of the values were above the 0.5 threshold, although the question of student benefit did have more variance than the other ratings. Figure 6.15 shows the p-test scores. A key point is that everyone highly agreed that they would use the tool and that it was correct.

6.3.2 Qualitative Discussion

Additionally, we asked several qualitative, free-form questions. These included whether or not the participant had used educational teaching tools in the classroom previously, what their experience with the problem generator was like, and would they use this tool in their classroom?

From their answers to whether or not they had used educational software in the class-

room before, we divided users into two populations: those who had and those who had not. We performed a standard unpaired, two-tail t-test between these populations. These results, shown in Figure 6.16, demonstrate that there was no statistically significant difference between the two populations. This supports the argument that the software is both easy to use and likely to be adopted as it was equally approachable for users of different backgrounds.

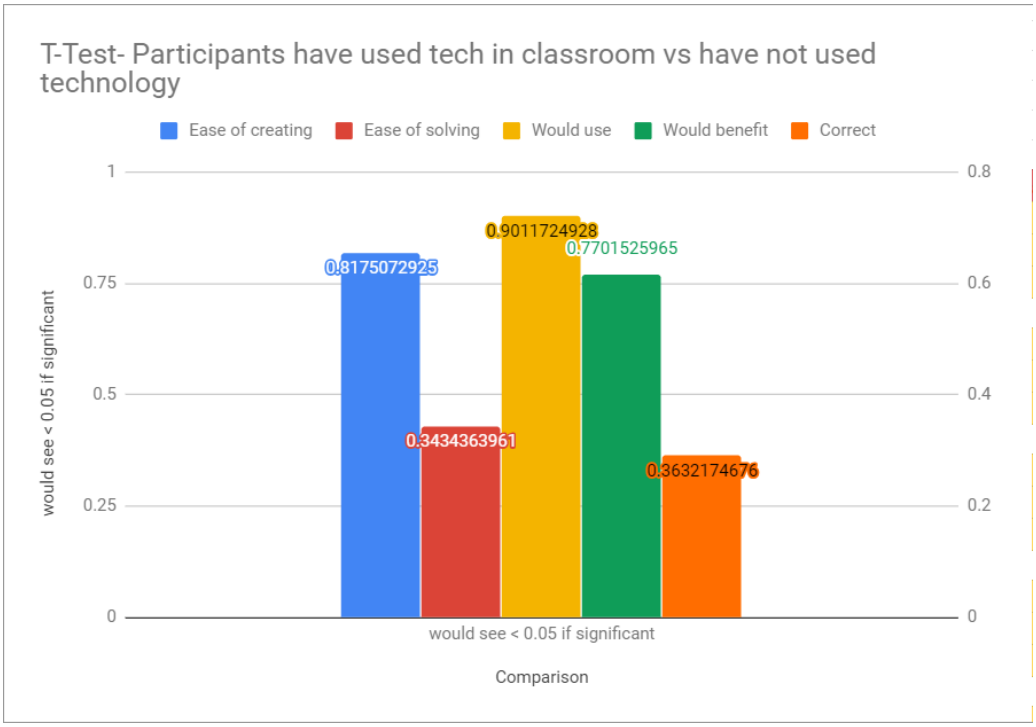


Figure 6.16: T-test: participants ratings between who have used technology in classroom Vs Who have not used technology in classroom during question generation through Mechanix. Each of these questions required participants to respond using a 5-point Likert scale, where 1 was the most positive answer and 5 was the most negative answer.

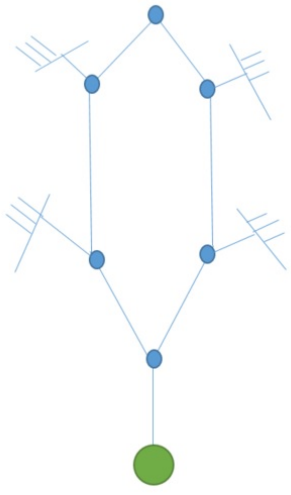
From discussion with our participants, they mostly stated that the experience was easier than other tools they had used. Several expressed surprise at how well the system generated new problems, which was not something they had seen in any software tool for their field before. We did receive excellent feedback from multiple users requesting support for new

problem types and extended domains.

7. FUTURE WORK

7.1 Create Realistic Practical Problems from Truss Structures

We are developing this system to create a realistic figure by using user given structural template. For example, if user given template has following structure as in Figure 7.1, then system can create a lizard from it. All clamped supports represent the legs and the section where the node is connected to the load represents the head of the lizard. Now if system has itś medial axis then by increasing the medial axis towards opposite direction of head and adding another node there, we can create a tail of the lizard. If we consider the load as an insect which has weight, we can create a base level realistic problem where student can solve the problem by using direct stiffness method. By bending the longest medial axis we can change the lizardś body within the same structural configuration by moving all nodes which are connected to clamp supports. We can then model another different shape of a lizard and create a different problem set.



]

Figure 7.1: User given structure with 6 nodes and one load.

7.2 Add a New Nodes and Extend the Structure

There are several ways we can fit a new node in current truss topology. As we mentioned in previous example we use medial axis to create a realistic problem, therefore we introduce the following ways to extend the given structure by adding new nodes-

1. Create a set with the points of medial axis of the given structure.

2. Find out the set of points, say L which belong to the straight line or curve which has maximum length (longest path) or maximum diagonal length of the bounding box of the curve, say l_{max}.

3. Create a set L_1 of approximated points of the straight line or curve using least square method. Add points until it has a minimum length δ_m between one of the end point of medial points of L and the last added approximated point (δ_m is equal to minimum length of the truss members).

4. Join this point with nearest two nodes by springs or beams.

5. Assign a stiffness for these two new members so that the new system has a non-singular stiffness matrix.

7.3 Improve the Spring Network Recognizer

During survey, we understood that there are several scopes to improve our recognizer.

1. Adding circle or a closed shape can be optional to create a spring & beam based structure.

2. There is a need to provide a recognizer which can identify one dimensional problem. Current system is able to identify one dimensional problem. During the survey, we saw that the instructor needs to put extra effort to provide boundary conditions (force

and displacement) for one dimensional problem through our force-displacement panel. He/she needs to provide additional information on x or y directional displacement and force respectively.

7.4 Support Template Selection from Auto-Generated Questions

We already mentioned that the system can create infinite number of questions from a base template by changing boundary conditions (stiffness, forces and displacements). Once instructor uses "add a member" method or "Reduce the structure" method system changes the topology of the current structure. Therefore these two methods create a new template. Two instructors can create two same topological structures during question generation. Storing these two structures as considered as base templates is wasting of space and time. Therefore we need to introduce a learning method in Mechanix that can identify the best fit of the newly created topology from already stored structural topology. If it finds a best fit of the newly formed truss topology it then ignores this new template to consider it as an another new template. Otherwise it needs to save the new one. Similarly if it finds a best fit topology in database/file server for user drawn structure, it then immediately update the boundary conditions. If it generates a boundary conditions automatically for an instructor's drawn structure it then reduces instructor's workload to create an initial problem.

8. CONCLUSION

In this research, we were able to develop a system with the potential to aid instructors and teaching assistants to reduce their workload when managing large STEM courses. We have implemented new features in the Mechanix educational software that enable it to recognize spring-based truss systems and automatically generate new questions that can be individualized for each student to further learning outcomes and support concept mastery through practice.

We present two main contributions. First, we develop a recognition system that recognizes a spring- and beam- based truss structure and create a base level problem template by analyzing user-given boundary conditions. We develop a new algorithm that helps to balance the stiffness of the members of truss structure (known or unknown) so that the base template has a unique solution for force-displacement balancing equations through stiffness.

Secondly, to generate structural problems automatically from the newly created base template, we develop different algorithms to change boundary conditions, stiffness, add new members in current structures, and remove members and nodes from current truss topology in Mechanix. Through changing boundary conditions and stiffness, instructors can create an infinite number of the same type of problem with the same difficulty level. By adding members or removing members and nodes, instructors can recreate different base problem templates. Every generated problem has a unique solution that can be compared with student's answers in the future.

We also plan to improve our system by modeling realistic problems for students from a base template, as described in Section 7.1. This approach will improve students' knowledge and understanding to apply learning concepts and outcomes to real-world problems.

Additionally, we will work to compare the newly created truss topology with a best-fit of existing truss topologies through Mechanix to select a base template for storing procedures. This will help reduce storing redundant base templates in the database of the file server.

In closing, we believe that educational software is an important part of classrooms of the future. In the midst of ongoing cost concerns, paired with the increasing value of education, we see a continual shift towards larger classes with fewer instructors. By incorporating educational software into the classroom capable of recognizing natural user interactions, such as hand-drawn spring-based truss systems, students will be able to gain benefits from teaching tools despite limited availability of teachers. Furthermore, software that can automatically generate new problems for limitless practice and individual assessment will become indispensable as tools for instructors because they will not only reduce the workload of managing large classes, but they will also support enhanced student achievements.

REFERENCES

[1] M. Hultman, "Weight optimization of steel trusses by a genetic algorithm - size, shape and topology optimization according to eurocode," 2010. Student Paper.

[2] B. C. Nizamettin Koc, "The impact of number of students per teacher on student achievement," vol. 117, pp. 65–70, 2017.

[3] K. H. Miles, "Transformation or decline?: Using tough times to create higher-performing schools," *Phi Delta Kappan*, vol. 93, no. 2, pp. 42–46, 2011.

[4] E. Graue and E. Rauscher, "Researcher perspectives on class size reduction.," *education policy analysis archives*, vol. 17, p. 9, 2009.

[5] E. Graue, E. Rauscher, and M. Sherfinski, "The synergy of class size reduction and classroom quality," *The Elementary School Journal*, vol. 110, no. 2, pp. 178–201, 2009.

[6] R. Phillips, "Pedagogical, institutional and human factors influencing the widespread adoption of educational technology in higher education," 2007.

[7] C. Liao, P. Palvia, and J.-L. Chen, "Information technology adoption behavior life cycle: Toward a technology continuance theory (tct)," *International Journal of Information Management*, vol. 29, no. 4, pp. 309–320, 2009.

[8] D. J. McFarland and D. Hamilton, "Adding contextual specificity to the technology acceptance model," *Computers in human behavior*, vol. 22, no. 3, pp. 427–447, 2006.

[9] Y.-H. Lee, Y.-C. Hsieh, and C.-N. Hsu, "Adding innovation diffusion theory to the technology acceptance model: Supporting employees' intentions to use e-learning systems.," *Journal of Educational Technology & Society*, vol. 14, no. 4, 2011.

[10] D. R. Garrison, *E-learning in the 21st century: A framework for research and practice*. Routledge, 2011.

[11] S. Y. Park *et al.*, "An analysis of the technology acceptance model in understanding university students' behavioral intention to use e-learning.," *Educational technology & society*, vol. 12, no. 3, pp. 150–162, 2009.

[12] C. W. ouis Deslauriers, Ellen Schelew, *improved Learning in a Large-Enrollment Physics Class*, vol. 332. 2011.

[13] M. Agarwal, R. Shah, and P. Mannem, "Automatic question generation using discourse cues," in *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 1–9, Association for Computational Linguistics, 2011.

[14] J. Yoo, S. Yoo, and S. Rusek, "A question generator for an online tutoring system," in *EdMedia: World Conference on Educational Media and Technology*, pp. 1820–1826, Association for the Advancement of Computing in Education (AACE), 2005.

[15] S. Cheema and J. LaViola, "Physicsbook: a sketch-based interface for animating physics diagrams," in *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pp. 51–60, ACM, 2012.

[16] M. G. Sutton and C. Jong, "A truss analyzer for enriching the learning experience of students," *age*, vol. 5, p. 1, 2000.

[17] T. F. Stahovich, "Sketchit: A sketch interpretation tool for conceptual mechanical design," 1996.

[18] S. Valentine, R. Lara-Garduno, J. Linsey, and T. Hammond, "Mechanix: A sketch-based tutoring system that automatically corrects hand-sketched statics homework," in *The impact of pen and touch technology on education*, pp. 91–103, Springer, 2015.

[19] S. Valentine, F. Vides, G. Lucchese, D. Turner, H.-h. Kim, W. Li, J. Linsey, and T. Hammond, "Mechanix: A sketch-based tutoring system for statics courses.," in *IAAI*, 2012.

[20] O. Atilola, M. Field, E. McTigue, T. Hammond, and J. Linsey, "Evaluation of a natural sketch interface for truss fbds and analysis," in *Frontiers in Education Conference (FIE), 2011*, pp. S2E–1, IEEE, 2011.

[21] O. Atilola, C. Osterman, T. Hammond, and J. Linsey, "Mechanix: the development of a sketch recognition truss tutoring system," *Proceedings of the American*, 2012.

[22] T. Hammond and R. Davis, "Ladder, a sketching language for user interface developers," in *ACM SIGGRAPH 2007 courses*, p. 35, ACM, 2007.

[23] B. Paulson and T. Hammond, "Paleosketch: accurate primitive sketch recognition and beautification," in *Proceedings of the 13th international conference on Intelligent user interfaces*, pp. 1–10, ACM, 2008.

[24] C. C. Goh, "A cognitive perspective on language learners' listening comprehension problems," *System*, vol. 28, no. 1, pp. 55–75, 2000.

[25] O. Atilola, S. Valentine, H.-H. Kim, D. Turner, E. McTigue, T. Hammond, and J. Linsey, "Mechanix: A natural sketch interface tool for teaching truss analysis and free-body diagrams," *AI EDAM*, vol. 28, no. 2, pp. 169–192, 2014.

[26] B. Paulson, P. Rajan, P. Davalos, R. Gutierrez-Osuna, and T. Hammond, "What!?! no rubine features?: using geometric-based features to produce normalized confidence values for sketch recognition," in *HCC Workshop: Sketch Tools for Diagramming*, pp. 57–63, 2008.

[27] T. M. Sezgin and R. Davis, "Hmm-based efficient sketch recognition," in *Proceedings of the 10th international conference on Intelligent user interfaces*, pp. 281–283,

ACM, 2005.

[28] K. Weynand, J.-P. Jaspart, and M. Steenhuis, "-the stiffness model of revised annex j of eurocode 3," in *Connections in steel structures III*, pp. 441–452, Elsevier, 1996.

[29] *Eurocode 3: design of steel structures : part 1-5 : plated structural elements*. London: BSI, 2010. Incorporating corrigendum April 2009.

[30] D. Menon, *Advanced structural analysis*. Alpha Science International, 2009.

[31] J. Johnston and T. Hammond, "Computing confidence values for geometric constraints for use in sketch recognition," in *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, pp. 71–78, Eurographics Association, 2010.

[32] T. A. Hammond, "New investigators: Artificial intelligence,"

[33] M. Field, S. Valentine, J. Linsey, and T. Hammond, "Sketch recognition algorithms for comparing complex and unpredictable shapes," in *IJCAI*, vol. 11, pp. 2436–2441, 2011.

[34] K. Kebodeaux, M. Field, and T. Hammond, "Defining precise measurements with sketched annotations," in *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pp. 79–86, ACM, 2011.

[35] S. Valentine, F. Vides, G. Lucchese, D. Turner, H.-h. Kim, W. Li, J. Linsey, and T. Hammond, "Mechanix: a sketch-based tutoring and grading system for free-body diagrams," *AI Magazine*, vol. 34, no. 1, p. 55, 2012.

[36] D. Rubine, *Specifying gestures by example*, vol. 25. ACM, 1991.

[37] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes," in *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pp. 159–168, ACM, 2007.

[38] L. B. Kara and T. F. Stahovich, "An image-based, trainable symbol recognizer for hand-drawn sketches," *Computers & Graphics*, vol. 29, no. 4, pp. 501–517, 2005.

[39] E. G. Miller, N. E. Matsakis, and P. A. Viola, "Learning from one example through shared densities on transforms," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 1, pp. 464–471, IEEE, 2000.

[40] Y. Amit, U. Grenander, and M. Piccioni, "Structural image restoration through deformable templates," *Journal of the American Statistical Association*, vol. 86, no. 414, pp. 376–387, 1991.

[41] T. Hammond and R. Davis, "Ladder: A language to describe drawing, display, and editing in sketch recognition," in *ACM SIGGRAPH 2006 Courses*, p. 27, ACM, 2006.

[42] T. Hammond and R. Davis, "Automatically transforming symbolic shape descriptions for use in sketch recognition," in *AAAI*, vol. 4, pp. 450–456, 2004.

[43] P. Taele and T. Hammond, "Using a geometric-based sketch recognition approach to sketch chinese radicals.," in *AAAI*, vol. 8, pp. 1832–1833, 2008.

[44] P. Taele and T. A. Hammond, "A geometric-based sketch recognition approach for handwritten mandarin phonetic symbols i.," in *DMS*, pp. 270–275, 2008.

[45] K. Forbus, K. Lockwood, M. Klenk, E. Tomai, and J. Usher, "Open-domain sketch understanding: The nusketch approach," in *AAAI Fall Symposium on Making Pen-based Interaction Intelligent and Natural*, pp. 58–63, 2004.

[46] T. Hammond and R. Davis, "Tahuti: A geometrical sketch recognition system for uml class diagrams," in *ACM SIGGRAPH 2006 Courses*, p. 25, ACM, 2006.

[47] T. Hammond and R. Davis, "A domain description language for sketch recognition," *Proceedings of 2002 SOW*, 2002.

[48] T. Hammond, "Natural sketch recognition in uml class diagrams," in *Proceedings of the MIT Student Oxygen Workshop*, 2001.

[49] C. Alvarado and R. Davis, "Sketchread: a multi-domain sketch recognition engine," in *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pp. 23–32, ACM, 2004.

[50] L. Gennari, L. B. Kara, T. F. Stahovich, and K. Shimada, "Combining geometry and domain knowledge to interpret hand-drawn diagrams," *Computers & Graphics*, vol. 29, no. 4, pp. 547–562, 2005.

[51] T. Hammond, M. Sezgin, O. Veselova, A. Adler, M. Oltmans, C. Alvarado, and R. Hitchcock, "Multi-domain sketch recognition," in *Proceedings of the 2nd Annual MIT Student Oxygen Workshop*, 2002.

[52] J. M. Peschel and T. A. Hammond, "Strat: a sketched-truss recognition and analysis tool.," in *DMS*, pp. 282–287, 2008.

[53] T. A. Hammond, D. Logsdon, B. Paulson, J. Johnston, J. M. Peschel, A. Wolin, and P. Taele, "A sketch recognition system for recognizing free-hand course of action diagrams.," in *IAAI*, 2010.

[54] T. Hammond and R. Davis, "Automatically transforming symbolic shape descriptions for use in sketch recognition," in *AAAI*, vol. 4, pp. 450–456, 2004.

[55] T. Hammond and R. Davis, "Shady: A shape description debugger for use in sketch recognition," in *AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural*, 2004.

[56] L.-W. Tsai, *Mechanism design: enumeration of kinematic structures according to function*. CRC press, 2000.

[57] D. Ashlock, *Evolutionary computation for modeling and optimization*. Springer Science & Business Media, 2006.

[58] P. E. Austrell, O. Dahlblom, J. Lindemann, A. Olsson, K.-G. Olsson, K. Persson, H. Petersson, M. Ristinmaa, G. Sandberg, and P.-A. Wernberg, "Calfem-a finite element toolbox, version 3.4," 2004.

[59] S. A. Salam, A. EL-shihy, A. Eraky, and M. Salah, "Optimum design of trussed dome structures,"

[60] G. Guerlement, R. Targowski, W. Gutkowski, J. Zawidzka, and J. Zawidzki, "Discrete minimum weight design of steel structures using ec3 code," *Structural and Multidisciplinary Optimization*, vol. 22, no. 4, pp. 322–327, 2001.

[61] R. Larsson, "Methodology for topology and shape optimization: Application to a rear lower control arm," 2016. Masters thesis 2016:30.

[62] P. W. Christensen, *An introduction to structural optimization. English*, vol. 153. 2008.

[63] H. Kawamura, H. Ohmori, and N. Kito, "Truss topology optimization by a modified genetic algorithm," *Structural and Multidisciplinary Optimization*, vol. 23, no. 6, pp. 467–473, 2002.

[64] P. Kripakaran, A. Gupta, and J. W. Baugh Jr, "A novel optimization approach for minimum cost design of trusses," *Computers & Structures*, vol. 85, no. 23-24, pp. 1782–1794, 2007.

[65] U. T. Ringertz, "On topology optimization of trusses," *Engineering optimization*, vol. 9, no. 3, pp. 209–218, 1985.

[66] M. W. Dobbs and L. P. Felton, "Optimization of truss geometry," *Journal of the Structural Division*, vol. 95, no. 10, pp. 2105–2118, 1969.

[67] C. Coello and A. D. Christiansen, "Multiobjective optimization of trusses using genetic algorithms," *Computers & Structures*, vol. 75, no. 6, pp. 647–660, 2000.

[68] B. U. o. T. Department of Structural Engineering and Economics, "Square hollow profile."

[69] T. Hagishita and M. Ohsaki, "Topology optimization of trusses by growing ground structure method," *Structural and Multidisciplinary Optimization*, vol. 37, no. 4, pp. 377–393, 2009.

[70] M. Ohsaki and N. Katoh, "Topology optimization of trusses with stress and local constraints on nodal stability and member intersection," *Structural and Multidisciplinary Optimization*, vol. 29, no. 3, pp. 190–197, 2005.

[71] M. Ohsaki, "Simultaneous optimization of topology and geometry of a regular plane truss," *Computers & structures*, vol. 66, no. 1, pp. 69–77, 1998.

[72] P. Martinez, P. Marti, and O. Querin, "Growth method for size, topology, and geometry optimization of truss structures," *Structural and Multidisciplinary Optimization*, vol. 33, no. 1, pp. 13–26, 2007.

[73] K. Deb and S. Gulati, "Design of truss-structures for minimum weight using genetic algorithms," *Finite elements in analysis and design*, vol. 37, no. 5, pp. 447–465, 2001.

[74] G. Guerlement, R. Targowski, W. Gutkowski, J. Zawidzka, and J. Zawidzki, "Discrete minimum weight design of steel structures using ec3 code," *Structural and Multidisciplinary Optimization*, vol. 22, no. 4, pp. 322–327, 2001.

[75] W. Dorn, "Automatic design of optimal structures," *J. de Mecanique*, vol. 3, pp. 25–52, 1964.

[76] A. Gersborg-Hansen, M. P. Bendsøe, and O. Sigmund, "Topology optimization of heat conduction problems using the finite volume method," *Structural and multidisciplinary optimization*, vol. 31, no. 4, pp. 251–259, 2006.

[77] J. McKeown, "Growing optimal pin-jointed frames," *Structural optimization*, vol. 15, no. 2, pp. 92–100, 1998.

[78] W. K. Rule, "Automatic truss design by optimized growth," *Journal of Structural Engineering*, vol. 120, no. 10, pp. 3063–3070, 1994.

[79] R. P. Feynman, R. B. Leighton, and M. Sands, *The Feynman lectures on physics, Vol. I: The new millennium edition: mainly mechanics, radiation, and heat*, vol. 1. Basic books, 2011.

[80] C. C. Coello, M. Rudnick, and A. D. Christiansen, "Using genetic algorithms for optimal design of trusses," in *Proceedings Sixth International Conference on Tools with Artificial Intelligence. TAI 94*, pp. 88–94, IEEE, 1994.

APPENDIX A

IMPORTANT METHODS OF RECOGNITION SYSTEM

A.1 Spring Recognition Algorithms

A.1.1 SpringSystem's addAsSpringComponent Method

Algorithm 4 Check if $Piece$ can be a $SNode$, if yes, then return $SNode$ and update current spring system, else return null

Require: A shape $Piece$

if $Piece = Line$ OR $Piece = Helix$ **then**
  $SNode_{Piece} \leftarrow \emptyset$
  $pfirst \leftarrow CP(Piece_f, Piece)$ AND $plast \leftarrow CP(Piece_l, Piece)$
  for all Edg_j in Spring System **do**
    if $Edg_j = Line$ OR $Edg_j = Helix$ **then**
      if $Piece = Edg_j$ **then**
        continue
      $CPoint_{Edg_{j1}} \leftarrow CP(Edg_{jf}, Edg_j)$
      $nearness1 \leftarrow CC(CPoint_{Edg_{j1}}, pfirst)$ AND $nearness2 \leftarrow CC(CPoint_{Edgj1}, plast)$
      if $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**
        Create $SNode_{Piece}$ and add it to the graph of current spring system
        $SNode_{Piece.firstPointConnected} = true$
        Add Edg_j to the edge of $SNode_{Piece}$
        Add $SNode_{Piece}$ to the edge of Edg_j
        $Edg_{j,firstPointConnected} = true$
        $MakeNode(Edg_j, Piece, point\ of\ CPoint_{Edg_{j1}})$
        Continue for next Edg_{j+1}
      if $nearness2 > nearness1$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**
        Create $SNode_{Piece}$ and add it to the graph of current spring system
        $SNode_{Piece.lastPointConnected} = true$

21: Add Edg_j to the edge of $SNode_{Piece}$
22: Add $SNode_{Piece}$ to the edge of Edg_j
23: $Edg_{j,firstPointConnected} = true$
24: $MakeNode(Edg_j, Piece, point\ of\ CPoint_{Edg_{j1}})$
25: Continue for next Edg_{j+1}
26: $CPoint_{Edg_{j2}} \leftarrow CP(Edg_{jl}, Edg_j)$
27: $nearness1 \leftarrow CC(CPoint_{Edg_{j2}}, pfirst)$ AND $nearness2 \leftarrow CC(CPoint_{Edgj2}, plast)$
28: **if** $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**
29: Repeat steps 11 to 14
30: $Edg_{j,lastPointConnected} = true$
31: $MakeNode(Edg_j, Piece, point\ of\ CPoint_{Edg_{j2}})$
32: Continue for next Edg_{j+1}
33: **if** $nearness2 > nearness1$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**
34: Repeat steps 19 to 22
35: $Edg_{j,lastPointConnected} = true$
36: $MakeNode(Edg_j, Piece, point\ of\ CPoint_{Edg_{j2}})$
37: Continue for next Edg_{j+1}
38: $intersectionConfidence \leftarrow IC(CS_{Piece}, CS_{Edge_j})$
39: **if** $intersectionConfidence > CONSTRAINT\_CONFIDENCE$ **then**
40: Create $SNode_{Piece}$ and add it to the graph of current spring system
41: Add Edg_j to the edge of $SNode_{Piece}$

42: $CC \leftarrow CC(0.30), nearness1 \leftarrow CC(pfirst, CPoint_{Edg_{j1}})$
43: $CC \leftarrow CC(0.30), nearness2 \leftarrow CC(pfirst, CPoint_{Edg_{j2}})$
44: **if** $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**
45: $SNode_{Piece,firstPointConnected} = true$
46: $Edg_{j,firstPointConnected} = true$
47: $MakeNode(Edg_j, Piece, point\ of\ CPoint_{Edg_{j1}})$
48: Continue for next Edg_{j+1}
49: **if** $nearness2 > nearness1$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**
50: $SNode_{Piece,firstPointConnected} = true$
51: $Edg_{j,lastPointConnected} = true$
52: $MakeNode(Edg_j, Piece, point\ of\ CPoint_{Edg_{j2}})$
53: Continue for next Edg_{j+1}
54: $IC(); nearness1 \leftarrow IC(pfirst, CS_{Edge_j})$
55: $IC(); nearness2 \leftarrow IC(plast, CS_{Edge_j})$
56: **if** $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**
57: $SNode_{Piece,firstPointConnected} = true$
58: $MakeNode(Edg_j, Piece, point\ of\ pfirst)$
59: Continue for next Edg_{j+1}
60: **if** $nearness2 > nearness2$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**
61: $SNode_{Piece,firstPointConnected} = true$
62: $MakeNode(Edg_j, Piece, point\ of\ plast)$
63: Continue for next Edg_{j+1}

64: $Point \quad pieceEdgeIntersection = LI(pfirst, plast, CPoint_{Edg_{j1}}, CPoint_{Edg_{j2}})$

65: **if** $pieceEdgeIntersection \neq \emptyset$ **then** $MakeNode(Edg_j, Piece, pieceEdgeIntersection)$

66: Continue for next Edg_{j+1}

67: **if** $Edg_j = Clamped\ Support$ **then**

68: $Shape_{base} \leftarrow \emptyset$

69: $Shape_{base} = longest\ shape\ of\ Edg_j$

70: $CL_{Shape_{base}} = CL(Shape_{base})$

71: $IC(), nearness1 = IC(pfirst, CL_{Shape_{base}})$

72: $IC(), nearness2 = IC(plast, CL_{Shape_{base}})$

73: **if** $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**

74: Create $SNode_{Piece}$ and add it to the graph of current spring system

75: $SNode_{Piece.firstPointConnected} = true$

76: Add Edg_j to the edge of $SNode_{Piece}$

77: Add $SNode_{Piece}$ to the edge of Edg_j

78: Update $clampedNodePoints$ by adding point of $pfirst$

79: Update $pointsOnClampedSupport$ by adding point of $plast$

80: Continue for next Edg_{j+1}

81: **if** $nearness2 > nearness1$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**

82: Repeat 74 to 77

83: Update $clampedNodePoints$ by adding point of $plast$

84: Update $pointsOnClampedSupport$ by adding point of $pfirst$

85: Continue for next Edg_{j+1}

86: **if** $Piece = Circle$ OR $Piece = ClosedShape$ **then**

87: **if** $Piece = ClosedShape$ **then**

88: $BoundingBox\ box \leftarrow Piece.getBoundingBox()$

89: $currentArea \leftarrow box.width * box.height$

90: **if** $currentArea \leq AREA\_THRESHOLD$ **then**

91: $CS_{Piece} \leftarrow CS(Piece)$

92: **for all** Edg_j in Spring System **do**

93: **if** $Piece = Edg_j$ **then**
94: continue
95: $CPoint_{Edg_{j1}} \leftarrow CP(first\ point\ of\ Edg_j, Edg_j)$
96: $IC(); nearness1 = IC(CPoint_{Edg_{j1}}, CS_{Piece})$
97: $CPoint_{Edg_{j2}} CP(last\ point\ of\ Edg_j, Edg_j)$
98: $IC(); nearness2 = IC(CPoint_{Edg_{j2}}, CS_{Piece})$
99: **if** $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**
100: Create $SNode_{Piece}$ and add it to the graph of current spring system
101: $SNode_{Piece.firstPointConnected} = true$
102: $SNode_{Piece.lastPointConnected} = true$
103: Add $SNode_{Piece}$ to the edge of Edg_j
104: $Edg_{j,firstPointConnected} = true$
105: Add last point of Edg_j to $loadedPoints$
106: Continue for next Edg_{j+1}
107: **if** $nearness2 > nearness1$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**
108: Create $SNode_{Piece}$ and add it to the graph of current spring system
109: $SNode_{Piece.firstPointConnected} = true$
110: $SNode_{Piece.lastPointConnected} = true$
111: Add $SNode_{Piece}$ to the edge of Edg_j
112: $Edg_{j,lastPointConnected} = true$
113: Add first point of Edg_j to $loadedPoints$
114: Continue for next Edg_{j+1}

115: **if** $Piece = Circle$ **then**
116: $CS_{Piece} \leftarrow CS(Piece)$
117: **for all** Edg_j in Spring System **do**
118: **if** $Piece = Edg_j$ **then**
119: continue
120: $CPoint_{Edg_{j1}} \leftarrow CP(first\ point\ of\ Edg_j, Edg_j)$
121: $IC()$; $nearness1 = IC(CPoint_{Edg_{j1}}, CS_{Piece})$
122: $CPoint_{Edg_{j2}} CP(last\ point\ of\ Edg_j, Edg_j)$
123: $IC()$; $nearness2 = IC(CPoint_{Edg_{j2}}, CS_{Piece})$
124: **if** $nearness1 > nearness2$ AND $nearness1 > CONSTRAINT\_CONFIDENCE$ **then**
125: Create $SNode_{Piece}$ and add it to the graph of current spring system
126: $SNode_{Piece.firstPointConnected} = true$
127: $SNode_{Piece.lastPointConnected} = true$
128: Add $SNode_{Piece}$ to the edge of Edg_j
129: $Edg_{j,firstPointConnected} = true$
130: Add last point of Edg_j to $loadedPoints$
131: Continue for next Edg_{j+1}
132: **if** $nearness2 > nearness1$ AND $nearness2 > CONSTRAINT\_CONFIDENCE$ **then**
133: Create $SNode_{Piece}$ and add it to the graph of current spring system
134: $SNode_{Piece.firstPointConnected} = true$
135: $SNode_{Piece.lastPointConnected} = true$
136: Add $SNode_{Piece}$ to the edge of Edg_j
137: $Edg_{j,lastPointConnected} = true$
138: Add first point of Edg_j to $loadedPoints$
139: Continue for next Edg_{j+1}
140: Return $SNode_{Piece}$

A.1.2 recognizeSpringNetwork

Algorithm 5 $recognizeSpringNetwork$ method

Require: $Sketch\ sketch$

```
1: pieces ← sketch.getShapes()
2: for all i ← 0, pieces.size() − 1 do
3:     if SpringSystem.Label.equals((pieces.get(i).getInterpretation().label)) then
4:         ss ← pieces.get(i)
5:         for all each s in ss.getShapes() do
6:             pieces.add(s)
7:         pieces.remove(i)
8:     if NODE_LABEL.equals(pieces.get(i).getInterpretation().label) then
9:         pieces.remove(i)
10: start ← −1
11: for all i ← 0, pieces.size() − 1 do
12:     if pieces.get(i).getInterpretation().label.equals(newClampedSupport()
    .getLabel()) then
13:         start ← i
14: if start = −1 then
15:     return
16: support ← pieces.get(start)
17: pieces.remove(start)
18: springSystem ← newSpringSystem(support)
19: while pieces.isEmpty() = false do
20:     done ← true
21:     for all i ← 0, pieces.size() − 1 do
22:         snode ← springSystem.checkAddComponent1(pieces.get(i))
23:         if snode ≠ ∅ then
24:             pieces.remove(i)
25:             done ← false
```

```
26:     if done then
27:         break
28: if springSystem.isConnected() && springSystem.hasError = false then
29:     if springSystem.nodes.size() ≥ 2 then
30:         AssingNodeLableIfNot(springSystem, sketch)
31:         newSpringSystem ← springSystem
32:         directStiffnessQuestion ← newDirectStiffnessQuestion(
    springSystem)
33:     else
34:         if springSystem.hasError = false then
35:             CivilSketchGUI.m_workspacePanel.getFeedbackPanel()
    .displayResponse("A spring based truss must have two nodes.
    Please create at least one member with two nodes before adding load. You can
    continue adding sketch on current structure.")
36: else if springSystem.hasError then
37:     CivilSketchGUI.m_workspacePanel.getFeedbackPanel()
    .displayResponse("Please remove the current close shape and draw a smaller
    rectangle/closedshape/circle for load.")
38: ss ← newShape()
39: ss.addInterpretation(newInterpretation(SpringSystem.Label, 1.0));
40: snodes ← springSystem.graph
41: for all snode : snodes do
42:     ss.add(snode.vertex)
43:     explode(snode.vertex, sketch)
44: sketch.add(ss)
45: for all i ← 0, springSystem.nodes.size() − 1 do
46:     springNode ← springSystem.nodes.get(i)
47:     dot ← newShape()
48:     s ← newStroke()
49:     s.addPoint(springNode.p)
50:     dot.add(s)
51:     dot.setLabel(NODE_LABEL)
52:     dot.setAttribute("radius","" + edu.tamu.civilSketch.recognition
    .stupid.Node.maxRadius)
53:     ShapeLabelManager.setShapeLabelText(dot,"")
54:     p ← newjava.awt.Point()
55:     p.x ← (int)springNode.p.getBoundingBox().getCenterX()
56:     p.y ← (int)springNode.p.getBoundingBox().getCenterY()
57:     ShapeLabelManager.setShapeLabelLocation(dot, p)
58:     sketch.add(dot)
```

Algorithm 6 $isConnected$ method

```
1: isComplete ← true
2: supportConn ← newboolean[graph.size()]
3: Arrays.fill(supportConn, false)
4: que is an array list of integers
5: explored is a HashSet of integers
6: que.add(0)
7: while que.isEmpty() = false do
8:     index ← que.poll()
9:     snode ← graph.get(index)
10:    for all i ← 0, snode.edges.size() − 1 do
11:        ind ← snode.edges.get(i).index
12:        if graph.get(ind).firstPointConnected && graph.get(ind)
   .lastPointConnected then
13:            supportConn[ind] ← true
14:        if explored.contains(ind) = false then
15:            que.add(ind)
16:            explored.add(ind)
17: st is Stack of integers
18: loadConn ← newboolean[graph.size()]
19: Arrays.fill(loadConn, false)
20: for all i ← 0, supportConn.length − 1 do
21:    if supportConn[i] = false then
22:        isComplete ← false
23:        break
24:    if loadConn[i] = false then
25:        st.clear()
26:        explored.clear()
27:        snode ← graph.get(i)
```

```
28:         st.add(snode.index)
29:         explored.add(snode.index)
30:         while st.isEmpty() = false do
31:             index ← st.peek()
32:             if graph.get(index).vertex.getInterpretation().label.equals("circle")
    || graph.get(index).vertex.getInterpretation().label.equalsIgnoreCase(
    ClosedShape.CLOSED_SHAPE_LABEL) then
33:                 loadConn[graph.get(index).index] ← true
34:                 while st.isEmpty() = false do
35:                     loadConn[st.pop()] ← true
36:                 break
37:             done ← true
38:             for all j ← 0, graph.get(index).edges.size() − 1 do
39:                 sn1 ← graph.get(index).edges.get(j)
40:                 if loadConn[sn1.index] = true then
41:                     while st.isEmpty() = false do
42:                         loadConn[st.pop()] ← true
43:                     break
44:                 else
45:                     if explored.contains(sn1.index) = false then
46:                         done ← false
47:                         st.add(sn1.index)
48:                         explored.add(sn1.index)
49:                         break
50:             if done && st.isEmpty() = false then
51:                 st.pop()
52:     if loadConn[i] = false then
53:         isComplete ← false
54:         break
55: return isComplete
```

APPENDIX B

IMPORTANT METHODS OF QUESTION GENERATION

B.1 Material Resources

We used same material resources according to Eurocode 3 and [1]

// Fabricational dimensions (in SI units):

//Element thickness:

thickness[] = {2.5, 3.0, 3.2, 4.0, 4.9, 5.0, 2.5, 3.0, 3.2, 4.0, 4.9, 5.0, 6.0, 6.3, 3.0, 3.2, 4.0, 4.9, 5.0, 6.0, 6.3, 8.0, 3.0, 3.2, 3.6, 4.0, 4.9, 5.0, 6.0, 6.3, 7.1, 8.0, 3.2, 3.6, 4.0, 4.9, 5.0, 6.0, 6.3, 7.1, 8.0, 3.2, 3.6, 4.0, 4.9, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 3.6, 4.0, 4.9, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 3.6, 4.0, 4.9, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 10.0, 4.0, 4.9, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 4.9, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 4.9, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 16.0, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 5.0, 5.6, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 6.0, 6.3, 7.1, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 8.0, 8.8, 10.0, 12.0, 12.5, 14.2, 16.0, 20.0, 19.0, 22.0, 25.0, 22.0, 25.0, 12.0, 16.0, 19.0, 22.0, 25.0, 28.0, 32.0, 12.0, 16.0, 19.0, 22.0, 25.0, 28.0, 32.0, 36.0, 16.0, 19.0, 22.0, 25.0, 28.0, 32.0, 36.0, 40.0, 25.0, 28.0, 32.0, 36.0, 40.0, 25.0, 28.0, 32.0, 36.0, 40.0};

Cross-sectional width:

crossSectionalWidth[] = {40, 40, 40, 40, 40, 40, 50, 50, 50, 50, 50, 50, 50, 50, 60, 60, 60, 60, 60, 60, 60, 60, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 76.2, 76.2, 76.2, 76.2, 76.2, 76.2, 76.2, 76.2, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 90, 90, 90, 90, 90, 90, 90, 90, 90, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 260, 260, 260, 260, 260, 260, 260, 260, 260, 260, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 350, 350, 350, 350, 350, 350, 350, 400, 400, 400, 400, 400, 400, 400, 400, 350, 350, 350, 400, 400, 450, 450, 450, 450, 450, 450, 450, 500, 500, 500, 500, 500, 500, 500, 500, 550, 550, 550, 550, 550, 550, 550, 550, 600, 600, 600, 600, 600, 700, 700, 700, 700, 700};

Moment of inertia:

momentOfInertia[] = {8.54, 9.78, 10.20, 11.80, 13.20, 13.40, 17.50, 20.20, 21.20, 25.00, 28.50, 28.90, 32.00, 32.80, 36.20, 38.20, 45.40, 52.50, 53.30, 59.90, 61.60, 69.70, 59.00, 62.30, 68.60, 74.70, 87.20, 88.50, 101.00, 104.00, 112.00, 120.00, 81.50, 89.90, 98.00, 115.00, 117.00, 133.00, 138.00, 149.00, 160.00, 95.00, 105.00, 114.00, 135.00, 137.00, 149.00, 156.00, 162.00, 176.00, 189.00, 152.00, 166.00, 196.00, 200.00, 218.00, 230.00, 238.00, 260.00, 281.00, 212.00, 232.00, 275.00, 279.00, 306.00, 323.00, 336.00, 367.00, 400.00, 462.00, 410.00, 489.00, 498.00, 547.00, 579.00, 603.00, 663.00, 726.00, 779.00, 852.00, 958.00, 982.00, 793.00, 807.00, 891.00, 944.00, 984.00, 1086.00, 1195.00, 1287.00, 1416.00, 1609.00, 1653.00, 984.00, 1002.00, 1106.00, 1174.00, 1223.00, 1352.00, 1491.00, 1608.00, 1773.00, 2023.00, 2080.00, 2430.00, 1225.00, 1353.00, 1437.00, 1499.00, 1659.00, 1831.00, 1978.00, 2186.00, 2502.00, 2576.00, 2809.00, 3028.00, 1765.00, 1954.00, 2077.00, 2168.00, 2404.00, 2661.00, 2880.00, 3193.00, 3677.00, 3790.00,

4154.00, 4504.00, 2445.00, 2710.00, 2883.00, 3011.00, 3345.00, 3709.00, 4021.00, 4471.00, 5171.00, 5336.00, 5872.00, 6394.00, 4861.00, 5399.00, 5752.00, 6014.00, 6701.00, 7455.00, 8107.00, 9055.00, 10556.00, 10915.00, 12094.00, 13267.00, 6491.00, 6788.00, 7567.00, 8423.00, 9164.00, 10242.00, 11954.00, 12365.00, 13714.00, 15061.00, 10080.00, 10547.00, 11775.00, 13128.00, 14305.00, 16026.00, 18777.00, 19442.00, 21637.00, 23850.00, 21129.00, 23055.00, 25884.00, 30435.00, 31541.00, 35211.00, 38942.00, 31857.00, 34798.00, 39128.00, 46130.00, 47839.00, 53526.00, 59344.00, 71535.00, 43360.00, 48360.00, 52890.00, 74710.00, 82150.00, 65430.00, 84070.00, 97060.00, 109200.00, 120600.00, 131200.00, 144100.00, 90750.00, 117100.00, 135500.00, 153000.00, 169400.00, 184900.00, 204000.00, 221500.00, 157700.00, 183000.00, 207100.00, 230000.00, 251600.00, 278600.00, 303500.00, 326500.00, 303400.00, 332700.00, 369400.00, 403700.00, 435500.00, 494100.00, 543500.00, 606200.00, 665400.00, 721200.00};

Steel qualities, f_v= The yield strength of steel, numbers represent the yield $strength^2$, f_y in $\frac{N}{mm^2}$. If the yield strength is exceeded in any of the members, plastic deformation or even fractures will occur in the truss.

```
steelqualities[] = {235, 275, 355, 420, 460}
getThickness(){
return 1e-3*thickness[new Random().nextInt(thickness.length)]
}
getCrossSectionalWidth(){
return 1e-3*crossSectionalWidth[new Random().nextInt(crossSectionalWidth.length)]
}
getMomentOfInertia(){
return 1e-8*momentOfInertia[new Random().nextInt(momentOfInertia.length)]
}
getSteelqualities(){
return steelqualities[new Random().nextInt(steelqualities.length)]
}
```

B.2 Question Generation

B.2.1 Change displacement

Algorithm 7 Create new $DirectStiffnessQuestion$ by changing displacements conditions

Require: Current $DirectStiffnessQuestion\ q$

$newQuestion \leftarrow q.Clone()$
$givenDisplacements,\ displacementsForQuestion$ are array lists of integers
$SpringNodenode1 \leftarrow \emptyset$
for all $i \leftarrow 0, n-1$ **do**
    if $q.displacement\_given[i] \neq true$ **then**
        $givenDisplacements.add(i)$
    else
        $displacementsForQuestion.add(i)$
$boolean\ xPrallel \leftarrow CheckIfParallelToXAxix(q.coordinatesInfo)$
$boolean\ yPrallel \leftarrow CheckIfParallelToYAxix(q.coordinatesInfo)$
$Randomr \leftarrow new\ Random();$
$Integer\ changeIndex \leftarrow givenDisplacements$
$.get(r.nextInt(givenDisplacements.size() - 1))$
$node1 \leftarrow q.springSystem1.nodes.get(changeIndex)$
$intlabel \leftarrow SpringNetworkMember.getLabelAsInt(node1.label)$
$int[]\ intArray \leftarrow 1, 2, 3$
$intselect \leftarrow intArray[new\ Random().nextInt(intArray.length)]$

```
17: if xPrallel && q.coordinatesInfo[0][1] = 0.0 then
18:     for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1 do
19:         if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
20:             newQuestion.displacementBoundaryCondInfo.remove(i)
21:             break
22: else if xPrallel then
23:     if elect = 1 then
24:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1 do
25:             if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
26:                 newQuestion.displacementBoundaryCondInfo.remove(i)
27:                 break
28:     if select = 2 then
29:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1 do
30:             if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
31:                 newQuestion.displacementBoundaryCondInfo.remove(i)
32:                 break
33:     if select = 3 then
34:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1 do
35:             if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
36:                 newQuestion.displacementBoundaryCondInfo.remove(i)
37:             if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
38:                 newQuestion.displacementBoundaryCondInfo.remove(i)
39: if yPrallel && q.coordinatesInfo[0][0] = 0.0 then
40:     for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1 do
41:         if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
42:             newQuestion.displacementBoundaryCondInfo.remove(i)
43:             break
```

```
44: else if yPrallel then
45:     if select = 1 then
46:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1
    do
47:             if  newQuestion.displacementBoundaryCondInfo.get(i)[0][1]  =
    1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label) then
48:                 newQuestion.displacementBoundaryCondInfo.remove(i)
49:                 break
50:     if elect = 2 then
51:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1
    do
52:             if  newQuestion.displacementBoundaryCondInfo.get(i)[0][1]  =
    2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
53:                 newQuestion.displacementBoundaryCondInfo.remove(i)
54:                 break
55:     if elect = 3 then
56:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1
    do
57:             if  newQuestion.displacementBoundaryCondInfo.get(i)[0][1]  =
    1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
58:                 newQuestion.displacementBoundaryCondInfo.remove(i)
59:             if  newQuestion.displacementBoundaryCondInfo.get(i)[0][1]  ==
    2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
60:                 newQuestion.displacementBoundaryCondInfo.remove(i)
61: else
62:     if select = 1 then
63:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1
    do
64:             if  newQuestion.displacementBoundaryCondInfo.get(i)[0][1]  =
    1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
65:                 newQuestion.displacementBoundaryCondInfo.remove(i)
66:                 break
67:     if elect = 2 then
68:         for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1
    do
69:             if  newQuestion.displacementBoundaryCondInfo.get(i)[0][1]  =
    2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
```

```
                newQuestion.displacementBoundaryCondInfo.remove(i)
                break
    if elect = 3 then
        for all i ← 0, newQuestion.displacementBoundaryCondInfo.size() − 1 do
            if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 1 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
                newQuestion.displacementBoundaryCondInfo.remove(i)
            if newQuestion.displacementBoundaryCondInfo.get(i)[0][1] = 2 && newQuestion.displacementBoundaryCondInfo.get(i)[0][0] = label then
                newQuestion.displacementBoundaryCondInfo.remove(i)
int i ← 0
if xPrallel then
    if q.coordinatesInfo[0][1] = 0 then
        newXDisplacement ← generateRandomDouble(0, q.xProblemSpan/250
        adddisp1 ← {label, 1, newXDisplacement}
        newQuestion.displacementBoundaryCondInfo.add(adddisp1)
        position ← 1
    else
        if select = 1 then
            newXDisplacement ← generateRandomDouble(0, q.xProblemSpan/250)
            adddisp1 ← {label, 1, newXDisplacement}
            newQuestion.displacementBoundaryCondInfo.add(adddisp1)
            position ← 1
        else if select = 2 then
            newYDisplacement ← generateRandomDouble(0, q.xProblemSpan/500)
            adddisp1 ← {label, 2, newYDisplacement}
            newQuestion.displacementBoundaryCondInfo.add(adddisp1)
            position ← 2
        else
            newXDisplacement ← generateRandomDouble(0, q.xProblemSpan/250)
            adddisp1 ← {label, 1, newXDisplacement}
            newQuestion.displacementBoundaryCondInfo.add(adddisp1)
```

100: $newYDisplacement \leftarrow generateRandomDouble(0, q.xProblemSpan/500)$
101: $adddisp2 \leftarrow \{label,\ 2,\ newYDisplacement\}$
102: $newQuestion.displacementBoundaryCondInfo.add(adddisp2)$
103: $position \leftarrow 3$
104: **else if** $yPrallel$ **then**
105: **if** $q.coordinatesInfo[0][0] = 0$ **then**
106: $newYDisplacement \leftarrow generateRandomDouble(0, q.yProblemSpan/250)$
107: $adddisp1 \leftarrow \{(double)(label),\ 2,\ newYDisplacement\}$
108: $newQuestion.displacementBoundaryCondInfo.add(adddisp1)$
109: $position \leftarrow 2$
110: **else**
111: **if** $select = 1$ **then**
112: $newXDisplacement \leftarrow generateRandomDouble(0, q.yProblemSpan/500)$
113: $adddisp1 \leftarrow \{label,\ 1,\ newXDisplacement\}$
114: $newQuestion.displacementBoundaryCondInfo.add(adddisp1)$
115: $position \leftarrow 1$
116: **else if** $select = 2$ **then**
117: $newYDisplacement = generateRandomDouble(0, q.yProblemSpan/250)$
118: $adddisp1 \leftarrow \{label,\ 2,\ newYDisplacement\}$
119: $newQuestion.displacementBoundaryCondInfo.add(adddisp1)$
120: $position \leftarrow 2$
121: **else**
122: $newXDisplacement = generateRandomDouble(0, q.yProblemSpan/500)$
123: $adddisp1 \leftarrow \{label, 1, newXDisplacement\}$
124: $newQuestion.displacementBoundaryCondInfo.add(adddisp1)$
125: $newYDisplacement \leftarrow generateRandomDouble(0, q.yProblemSpan/250)$
126: $adddisp2 \leftarrow \{label,\ 2,\ newYDisplacement\}$
127: $newQuestion.displacementBoundaryCondInfo.add(adddisp2)$
128: $position \leftarrow 3$

```
else
    if select = 1 then
        newXDisplacement ← generateRandomDouble(0, q.yProblemSpan/250)
        adddisp1 ← {label, 1, newXDisplacement}
        newQuestion.displacementBoundaryCondInfo.add(adddisp1)
        position ← 1
    else if select = 2 then
        newYDisplacement ← generateRandomDouble(0, q.xProblemSpan/250)
        adddisp1 ← {label, 2, newYDisplacement}
        newQuestion.displacementBoundaryCondInfo.add(adddisp1)
        position ← 2
    else
        newXDisplacement ← generateRandomDouble(0, q.yProblemSpan/250)
        adddisp1 ← {label, 1, newXDisplacement}
        ewQuestion.displacementBoundaryCondInfo.add(adddisp1)
        newYDisplacement ← generateRandomDouble(0, q.xProblemSpan/250)
        adddisp2 ← {label, 2, newYDisplacement}
        newQuestion.displacementBoundaryCondInfo.add(adddisp2)
        position ← 3
lines is an array of strings.
lines ← GetDisplacementStringAtNode(changeIndex)
if position = 1 then
    SetDisplacementAttributeAtNode(changeIndex, newXDisplacement, lines[1])
else if position = 2 then
    SetDisplacementAttributeAtNode(changeIndex, lines[0], newYDisplacement)
else
    SetDisplacementAttributeAtNode(changeIndex, newXDisplacement, newYDisplacement)
node2 ← ∅
changeIndex2 ← −1
if displacementsForQuestion = ∅ && givenDisplacements ≠ ∅ then
    unknown ← UnknownDisplacement(q.unknowndisplacementInfo)
```

```
160:     if unknown ≠ ∅ then
161:         k ← unknown[0][0]
162:         node2 ← setNodeLabelByString(k)
163:         DefineStiffnessInfo(newQuestion)
164:         givenCondtions ← DefineQuestionInfo(newQuestion, true)
165:         SetQuestionText(newQuestion, givenCondtions, node2)
166:     else
167:         changeIndex2 ← givenDisplacements.get(r
    .nextInt(givenDisplacements.size()))
168: else if displacementsForQuestion ≠ ∅ then
169:     unknown ← UnknownDisplacement(q.unknowndisplacementInfo)
170:     if unknown ≠ ∅ then
171:         k ← unknown[0][0]
172:         node2 ← setNodeLabelByString(k)
173:         DefineStiffnessInfo(newQuestion)
174:         givenCondtions ← DefineQuestionInfo(newQuestion, true)
175:         SetQuestionText(newQuestion, givenCondtions, node2)
176:     else
177:         changeIndex2 ← displacementsForQuestion.get(r
    .nextInt(displacementsForQuestion.size()))
178: if changeIndex2 ≠ −1 then
179:     for all j ← 0, newQuestion.displacementBoundaryCondInfo.size() do
180:         if newQuestion.springSystem1.nodes.get(changeIndex2).p =
    newQuestion.springMembers.get(j).firstNode) then
181:             node2 ← newQuestion.springMembers.get(j).firstNodeLabel
182:             break
183:         if newQuestion.springSystem1.nodes.get(changeIndex2).p =
    newQuestion.springMembers.get(j).secondNode then
184:             node2 ← newQuestion.springMembers.get(j).secondNodeLabel
185:             break
186:         DefineStiffnessInfo(newQuestion)
187:         givenCondtions ← DefineQuestionInfo(newQuestion, true)
188:         SetQuestionText(newQuestion, givenCondtions, node2)
189: SetSolution(newQuestion)
190: Return newQuestion
```

B.2.2 Reduce Method

Algorithm 8 Create new $DirectStiffnessQuestion$ by removing zero force elements

Require: Current $DirectStiffnessQuestion\ q$

1: $newQuestion \leftarrow \emptyset$
2: $DirectStiffnessQuestion\ newQuestion \leftarrow q.Clone()$
3: $deleteShapes$ is an array list of $Shape$
4: $deleteNodes$ is an array list of $SpringNode$
5: **for all** $j \leftarrow 0,\ newQuestion.springSystem1.nodes.size() - 1$ **do**
6: $nodeadded \leftarrow false$
7: $breaking \leftarrow false$
8: **for all** $i \leftarrow 0,$ $newQuestion.springSystem1.pointsOnClampedSupport.size() - 1$ **do**
9: **if** $newQuestion.springSystem1.nodes.get(j).p = newQuestion.springSystem1.pointsOnClampedSupport.get(i)$ **then**
10: $breaking \leftarrow true$
11: $break$
12: **if** $breaking = true$ **then**
13: $continue$
14: **else if** $newQuestion.springSystem1.nodes.get(j).shapes.size() = 1$ **then**
15: $deleteShapes.add(newQuestion.springSystem1.nodes.get(j).shapes.get(0))$
16: $deleteNodes.add(newQuestion.springSystem1.nodes.get(j))$
17: $nodeadded \leftarrow true$

```
18:     else if newQuestion.springSystem1.nodes.get(j).p.getAttribute("force") =
    ∅ OR newQuestion.springSystem1.nodes.get(j).p
    .getAttribute("force").equals("") then
19:         if newQuestion.springSystem1.nodes.get(j).shapes.size() = 2 then
20:             coordinateList is an array list to store coordinates
21:             coordinateList.add(newQuestion.springSystem1.nodes.get(j)
    .ProblemCoord)
22:             for all k ← 0, newQuestion.springMembers.size() − 1 do
23:                 if newQuestion.springMembers.get(k).FirstNode.label =
    newQuestion.springSystem1.nodes.get(j).label then
24:                     coordinateList.add(newQuestion.springMembers.get(k)
    .SecondNode.ProblemCoord)
25:                 else if newQuestion.springMembers.get(k).SecondNode.label =
    newQuestion.springSystem1.nodes.get(j).label then
26:                     coordinateList.add(newQuestion.springMembers.get(k)
    .FirstNode.ProblemCoord)
27:             if coordinateList.size() ≥ 3&& coordinateList ≠ ∅ then
28:                 if CheckIfThreePointsCollinear(coordinateList) = false then
29:                     for all k ← 0, 1 do
30:                         exist ← false
31:                         if deleteShapes.size() > 0 then
32:                             for all m ← deleteShapes.size() − 1 do
33:                                 if deleteShapes.get(m).getId().equals(newQuestion
    .springSystem1.nodes.get(j).shapes.get(k).getID()) then
34:                                     exist ← true
35:                                     break
36:                         else
37:                             deleteShapes.add(newQuestion.springSystem1
    .nodes.get(j).shapes.get(k))
38:                             deleteNodes.add(newQuestion.springSystem1.
    nodes.get(j))
39:                             exist ← true
40:                             nodeadded ← true
41:                         if exist = false then
42:                             deleteShapes.add(newQuestion.springSystem1
    .nodes.get(j).shapes.get(k))
43:                             if nodeadded = false then
44:                                 deleteNodes.add(newQuestion.springSystem1
    .nodes.get(j))
```

45: $nodeadded \leftarrow true$
46: **else if** $newQuestion.springSystem1.nodes.get(j).shapes.size() = 3$ **then**
47: $coordinateList$ is an array list to store coordinate
48: $memberList$ is an array list of $SpringNetworkMember$
49: $coordinateList.add(newQuestion.springSystem1.nodes.get(j).ProblemCoord)$
50: $m \leftarrow 1$
51: **for all** $k \leftarrow 0,\ newQuestion.springMembers.size() - 1$ **do**
52: **if** $newQuestion.springMembers.get(k).FirstNode.label = newQuestion.springSystem1.nodes.get(j).label$ **then**
53: $coordinateList.add(newQuestion.springMembers.get(k).SecondNode.ProblemCoord)$
54: $memberList.add(newQuestion.springMembers.get(k))$
55: $m \leftarrow m + 1$
56: **else if** $newQuestion.springMembers.get(k).SecondNode.label = newQuestion.springSystem1.nodes.get(j).label$ **then**
57: $coordinateList.add(newQuestion.springMembers.get(k).FirstNode.ProblemCoord)$
58: $memberList.add(newQuestion.springMembers.get(k))$
59: $m \leftarrow m + 1$
60: **if** $coordinateList.size() = 4$ **then**
61: $coordinateListis$ an array list to store coordinates
62: $deletdThird \leftarrow false$
63: $testCoordinateList.add(coordinateList.get(0))$
64: **if** $CheckIfThreePointsCollinear(testCoordinateList)$ **then**
65: $secondTestCoordinateList2$ is an array list
66: $secondTestCoordinateList2.add(testCoordinateList.get(0))$
67: $secondTestCoordinateList2.add(testCoordinateList.get(1))$
68: $secondTestCoordinateList2.add(coordinateList.get(3))$
69: **if** $CheckIfThreePointsCollinear(secondTestCoordinateList2) = false$ **then**
70: $exist \leftarrow false$
71: **if** $deleteShapes.size() > 0$ **then**
72: **for all** $t \leftarrow 0,\ deleteShapes.size() - 1$ **do**
73: **if** $deleteShapes.get(t).getId() = memberList.get(2).joint.getID()$ **then**
74: $exist \leftarrow true$
75: $break$

```
76:                         else
77:                             deleteShapes.add(memberList.get(2).joint)
78:                             deletdThird ← true
79:                             exist ← true
80:                         if exist = false then
81:                             deleteShapes.add(memberList.get(2).joint)
82:                             deletdThird ← true
83:                 else
84:                     secondTestCoordinateList2 is an array list
85:                     secondTestCoordinateList2.add(testCoordinateList.get(0))
86:                     secondTestCoordinateList2.add(testCoordinateList.get(1))
87:                     secondTestCoordinateList2.add(coordinateList.get(3))
88:                     if CheckIfThreePointsCollinear(
    secondTestCoordinateList2) then
89:                         exist ← false
90:                         if deleteShapes.size() > 0 then
91:                             for all t ← 0, deleteShapes.size() − 1 do
92:                                 if deleteShapes.get(t).getId() =
    memberList.get(1).joint.getID() then
93:                                     exist ← true
94:                                     break
95:                         else
96:                             eleteShapes.add(memberList.get(1).joint)
97:                             deletdThird ← true
98:                             exist ← true
99:                         if exist = false then
100:                            deleteShapes.add(memberList.get(1).joint)
101:                            deletdThird ← true
102:                if deletdThird = false then
103:                    testCoordinateList.clear()
104:                    testCoordinateList.add(coordinateList.get(0))
105:                    estCoordinateList.add(coordinateList.get(2))
106:                    testCoordinateList.add(coordinateList.get(3))
107:                    if CheckIfThreePointsCollinear(testCoordinateList) then
108:                        thirdTestCoordinateList is an array list
109:                        thirdTestCoordinateList.add(coordinateList.get(0))
110:                        thirdTestCoordinateList.add(coordinateList.get(1))
111:                        thirdTestCoordinateList.add(coordinateList.get(3))
```

```
112:                         if CheckIfThreePointsCollinear(
    thirdTestCoordinateList) = false then
113:                             exist ← false
114:                             if deleteShapes.size() > 0 then
115:                                 for all t ← 0, deleteShapes.size() − 1 do
116:                                     if deleteShapes.get(t).getId().equals(
    memberList.get(0).joint.getID()) then
117:                                         exist ← true
118:                                         break
119:                             else
120:                                 deleteShapes.add(memberList.get(0).joint)
121:                                 exist ← true
122:                                 nodeadded ← true
123:                             if exist = false then
124:                                 deleteShapes.add(memberList.get(0).joint)
125:                                 if nodeadded = false then
126:                                     nodeadded = true
127:                         else
128:                             continue
129:     else
130:         continue
131: if deleteShapes.size() = 0 then
132:     return∅
133: springSystem ← reducedSketchByZeroElements1(newQuestion,
    deleteShapes, deleteNodes)
134: finalQuestion ← new DirectStiffnessQuestion(springSystem)
135: finalQuestion.AssignConstraints()
136: if finalQuestion.elementsNodesVectors = ∅ then
137:     finalQuestion.setUpDispacementAndForce(finalQuestion
    .springSystem1)
138: SetSolution(finalQuestion)
139: givenCondtions ← DefineQuestionInfo(finalQuestion, true)
140: finalQuestion.text = SetQuestionText(finalQuestion, givenCondtions)
141: return finalQuestion
```

Algorithm 9 Create new $SpringSystem$ by removing nodes and truss elements

Require: Current $DirectStiffnessQuestion\ question$, $ArrayList < Shape > deleteShapes$, $ArrayList < SpringNode > deleteNodes$

$sketch \leftarrow new\ Sketch()$
$newSpringSystem \leftarrow \emptyset$
$SNodesList$ is an array list to store $SNode$
$SpringNodeList$ is an array list of $SpringNode$
for all $i \leftarrow 0,\ question.springSystem1.graph.size() - 1$ **do**
    $delete \leftarrow false$
    for all $j \leftarrow 0,\ deleteShapes.size() - 1$ **do**
        if $question.springSystem1.graph.get(i).getVertex().getId() = deleteShapes.get(j).getId()$ **then**
            $delete \leftarrow true$
            $break$
    if $delete = false$ **then**
        $SNodesList.add(question.springSystem1.graph.get(i))$
$nodeCount \leftarrow 1$
$copyDeleteNodes \leftarrow new\ ArrayList <> (deleteNodes)$
for all $i = 0,\ question.springSystem1.nodes.size() - 1$ **do**
    $springNode \leftarrow \emptyset$
    $deleted \leftarrow false$
    $delete \leftarrow 0$
    for all $j \leftarrow 0,\ copyDeleteNodes.size() - 1$ **do**
        if $question.springSystem1.nodes.get(i).p.equals(copyDeleteNodes.get(j).p)$ **then**
            $deleted \leftarrow true$
            $delete \leftarrow j$
    if $deleted$ **then**
        $copyDeleteNodes.remove(delete)$
    else
        $springNode \leftarrow question.springSystem1.nodes.get(i)$
        $SpringNodeList.add(question.springSystem1.nodes.get(i))$
for all $i \leftarrow 0,\ SNodesList.size() - 1$ **do**
    if $SNodesList.get(i).vertex.getInterpretation().label.equals(new\ ClampedSupport().getLabel())$ **then**
        $newSpringSystem \leftarrow new\ SpringSystem(SNodesList.get(i).vertex)$
        $SNodesList.remove(i)$
        $break$

33: **if** $newSpringSystem \neq \emptyset$ **then**
34: **for all** $i \leftarrow 0,\ SNodesList.size() - 1$ **do**
35: $newSpringSystem.graph.add(SNodesList.get(i))$
36: **for all** $i \leftarrow 0,\ SpringNodeList.size() - 1$ **do**
37: $newSpringSystem.nodes.add(SpringNodeList.get(i))$
38: $newSpringSystem.clampedNodePoints \leftarrow question.springSystem1.clampedNodePoints$
39: $newSpringSystem.pointsOnClampedSupport \leftarrow question.springSystem1.pointsOnClampedSupport$
40: $newSpringSystem.loadedPoints \leftarrow question.springSystem1.loadedPoints$
41: **if** $newSpringSystem.isConnected()$ **then**
42: $System.out.println(\text{"}@SRL: WE\ FOUND\ A\ SPRING\ NETWORK\ HERE,\ KEEP\ CONTINUE\text{"})$
43: $AssignNewNodeLabel(newSpringSystem)$
44: $returnnewSpringSystem$
45: **else**
46: $return\ \emptyset$

B.2.3 Add new member

Algorithm 10 Create $DirectStiffnessQuestion$ by adding new member to the current structure

Require: Current $DirectStiffnessQuestion\ q$

1: $newQuestion \leftarrow \emptyset$
2: $n \leftarrow q.getNbrOfNodes()$
3: $x \leftarrow check(n, q.springMembers.size() + 1)$
4: **if** $x = false$ **then**
5: $CivilSketchGUI.m_workspacePanel.getFeedbackPanel()$ $.displayResponse("Structurehasmaximumnumberofmembers.", FeedbackPanel.INCORRECT)$
6: $newQuestion \leftarrow q.clone1()$
7: $newQuestion.elementsNodesVectors \leftarrow new\ int[q.getNbrOfMembers()+1][2]$
8: **for all** $i \leftarrow 0,\ q.elementsNodesVectors.length - 1$ **do**
9: $newQuestion.elementsNodesVectors[i][0] \leftarrow q.elementsNodesVectors[i][0]$
10: $newQuestion.elementsNodesVectors[i][1] \leftarrow q.elementsNodesVectors[i][1]$
11: **for all** $i \leftarrow 0,\ n - 1$ **do**
12: **for all** $j \leftarrow 0,\ n$ **do**
13: **if** $j \neq i$ **then**
14: $exists \leftarrow false$
15: **for all** $k \leftarrow 0,\ q.elementsNodesVectors.length - 1$ **do**
16: **if** $(q.elementsNodesVectors[k][0] = i + 1\ \&\&\ q.elementsNodesVectors[k][1] = j+1)\ ||(q.elementsNodesVectors[k][1] = i + 1\ \&\&\ q.elementsNodesVectors[k][0] = j + 1)$ **then**

```
                    exists ← true
                    break
            if exists = false then
                newQuestion.elementsNodesVectors[q.getNbrOfMembers()][0] =
    i + 1
                newQuestion.elementsNodesVectors[q.getNbrOfMembers()][1] =
    j + 1
stiffness ← getNewStiffness(newQuestion.springMembers)
candidatePoints ← GetCandidateNodes(q)
springNetworkMember ← ∅
if candidatePoints ≠ ∅ && candidatePoints.length = 2 then
    first ← candidatePoints[0]
    last ← candidatePoints[1]
    springNetworkMember ← SpringNetworkMember
    createNewMember(first, last, stiffness)
else
    first ← newQuestion.findNodePoint(newQuestion.
elementsNodesVectors[q.getNbrOfMembers()][0])
    last ← newQuestion.findNodePoint(newQuestion
.elementsNodesVectors[q.getNbrOfMembers()][1])
    springNetworkMember = SpringNetworkMember
.createNewMember(first, last, stiffness)
springNetworkMember.firstNodeLabel ← springNetworkMember
.setNodeLabelByStringStartFromOne(newQuestion.elementsNodesVectors
[q.getNbrOfMembers()][0])
springNetworkMember.secondNodeLabel ← springNetworkMember
.setNodeLabelByStringStartFromOne(newQuestion.elementsNodesVectors
[q.getNbrOfMembers()][1])
springNetworkMember.areNodesLabeled ← true
newQuestion.difficulty ← SetDifficulty(q.difficulty)
newQuestion.springMembers.add(springNetworkMember)
newQuestion.springSystem1.addJointBetweenSpringNodes(
springNetworkMember.joint, springNetworkMember.firstNodeLabel,
springNetworkMember.secondNodeLabel)
newQuestion.AssignConstraints(newQuestion)
SetSolution(newQuestion)
givenCondtions ← DefineQuestionInfo(newQuestion, true)
newQuestion.text = SetQuestionText(newQuestion, givenCondtions)
return newQuestion
```

Algorithm 11 Get candidate members from $DirectStiffnessQuestion$

Require: $DirectStiffnessQuestion\ q$

1: $n \leftarrow q.springSystem1.nodes.size()$
2: $x \leftarrow (n * (n-1))/2$
3: $notConnected1 \leftarrow new\ int[x-n][2]$
4: $notConnected$ is an array
5: **for all** $i \leftarrow 1,\ q.springSystem1.nodes.size()$ **do**
6: **for all** $j \leftarrow i+1,\ q.springSystem1.nodes.size()$ **do**
7: **if** $i \neq j$ **then**
8: $exists \leftarrow false$
9: **for all** $k \leftarrow 0,\ q.elementsNodesVectors.length - 1$ **do**
10: **if** $(q.elementsNodesVectors[k][0] = i\ \&\&\ q.elementsNodesVectors[k][1] = j)\ ||\ (q.elementsNodesVectors[k][1] = i \&\&\ q.elementsNodesVectors[k][0] = j)$ **then**
11: $exists \leftarrow true$
12: $break$
13: **if** $exists = false$ **then**
14: $not = new\ int[][new\ int[]i, j$
15: $notConnected.add(not)$
16: $p \leftarrow 0$
17: $number \leftarrow 0$
18: $potentialStrain \leftarrow 0$
19: **if** $notConnected.isEmpty() = false$ **then**
20: $U1x \leftarrow 0$
21: $U1y \leftarrow 0$
22: $U2x \leftarrow 0$
23: $U2y \leftarrow 0$
24: $X1 \leftarrow 0$
25: $Y1 \leftarrow 0$
26: $X2 \leftarrow 0$
27: $Y2 \leftarrow 0$
28: **while** $p < notConnected.size()$ **do**
29: $joint \leftarrow notConnected.get(p)$
30: **for all** $i \leftarrow 0,\ q.solution.newGlobalDisplacementInfoAnswerPanel.size() - 1$ **do**
31: $disp \leftarrow Arrays.copyOf(q.solution.newGlobalDisplacementInfoAnswerPanel.get(i), 1)$
32: **if** $joint[0][0] = (int)disp[0][0]$ **then**
33: $U1x \leftarrow disp[0][1]$
34: $U1y \leftarrow disp[0][2]$

```
            else if joint[0][1] = (int)disp[0][0] then
                U2x ← disp[0][1]
                U2y ← disp[0][2]
        for all i ← 0, q.springSystem1.nodes.size() − 1 do
            a ← setNodeLabelByString((int)joint[0][0])
            a1 ← q.springSystem1.nodes.get(i).label
            b ← setNodeLabelByString((int)joint[0][1])
            if a.toString().equals(a1.toString()) then
                X1 ← q.springSystem1.nodes.get(i).ProblemCoord[0][0]
                Y1 ← q.springSystem1.nodes.get(i).ProblemCoord[0][1]
            else if b.toString().equals(a1.toString()) then
                X2 ← q.springSystem1.nodes.get(i).ProblemCoord[0][0]
                Y2 ← q.springSystem1.nodes.get(i).ProblemCoord[0][1]
        currentPotentialStrain = calculatePotentialStrain(U1x, U1y, U2x,
    U2y, X1, Y1, X2, Y2)
        if currentPotentialStrain > potentialStrain then
            potentialStrain ← currentPotentialStrain
            number ← p
        p ← p + 1
else
    return ∅
connectiom ← notConnected.get(number)
newmemberpoints ← new SpringNode[2]
newmemberpoints[0] ← new SpringNode()
newmemberpoints[1] ← new SpringNode()
for all i ← 0, q.springSystem1.nodes.size() − 1 do
    if q.springSystem1.nodes.get(i).label.equals(setNodeLabelByString(
    (int)connectiom[0][0])) then
        newmemberpoints[0] ← q.springSystem1.nodes.get(i)
    else if q.springSystem1.nodes.get(i).label.equals(setNodeLabelByString(
    (int)connectiom[0][1])) then
        newmemberpoints[1] ← q.springSystem1.nodes.get(i)
if newmemberpoints[0].p.x = newmemberpoints[1].p.x &&
    newmemberpoints[0].p.y = newmemberpoints[1].p.y &&
    newmemberpoints[0].p.x = 0 && newmemberpoints[0].p.y = 0 then
    return ∅
else
    return newmemberpoints
```

Algorithm 12 $createNewMember$ method to create a $SpringNetworkMember$

Require: $SpringNode\ start,\ SpringNode\ end, Double\ stiffness$

1: $springNetworkMember \leftarrow new\ SpringNetworkMember()$
2: $spring \leftarrow generateHelix(start.p, end.p)$
3: $springNetworkMember.joint \leftarrow spring$
4: $springNetworkMember.firstNode \leftarrow start.p$
5: $springNetworkMember.FirstNode \leftarrow start$
6: $springNetworkMember.firstNodeLabel \leftarrow start.label$
7: $springNetworkMember.secondNode \leftarrow end.p$
8: $springNetworkMember.SecondNode \leftarrow end$
9: $springNetworkMember.secondNodeLabel \leftarrow end.label$
10: $springNetworkMember.memberStiffness \leftarrow stiffness$
11: $return\ springNetworkMember$

Algorithm 13 $generateHelix$ method to create a helix shape

Require: $Point\ first,\ Point\ last$

1: $m\_beautified \leftarrow new\ Shape()$
2: $m\_beautified.setLabel(HelixFit.HELIX)$
3: $m\_beautified.setAttribute(IsAConstants.PRIMITIVE, IsAConstants.PRIMITIVE)$
4: $m\_shape \leftarrow new\ GeneralPath()$
5: $m\_startX \leftarrow first.getX()$
6: $m\_startY \leftarrow first.getY()$
7: $m\_endX \leftarrow last.getX()$
8: $m\_endY \leftarrow last.getY()$
9: $m\_revolutions \leftarrow 3$
10: $((GeneralPath)m\_shape).moveTo((float)m\_startX,\ (float)m\_startY)$
11: $theta \leftarrow 0$
12: $x, y, t \leftarrow 0$
13: $c\_startX \leftarrow m\_startX,\ c\_startY \leftarrow m\_startY,\ c\_endX \leftarrow m\_endX,\ c\_endY \leftarrow m\_endY$
14: $m\_avgRadius \leftarrow 1$
15: **while** $Point2D.distance(c\_startX,\ c\_startY,\ m\_startX,\ m\_startY) < m\_avgRadius\ \&\&\ m\_endX \neq m\_startX\ \&\&\ m\_endY \neq m\_startY$ **do**
16: $t = t + 0.001$
17: $c\_startX \leftarrow m\_startX + (m\_endX - m\_startX) * t$
18: $c\_startY \leftarrow m\_startY + (m\_endY - m\_startY) * t$
19: $t \leftarrow 1$
20: **while** $Point2D.distance(c\_endX,\ c\_endY,\ m\_endX,\ m\_endY) < m\_avgRadius\ \&\&\ m\_endX \neq\ m\_startX \&\&\ m\_endY \neq\ m\_startY$ **do**
21: $t \leftarrow t - 0.001$
22: $c\_endX \leftarrow m\_startX + (m\_endX - m\_startX) * t$
23: $c\_endY \leftarrow m\_startY + (m\_endY - m\_startY) * t$
24: $t \leftarrow 0$
25: $startAngle = Math.atan2(m\_startY - c\_startY,\ m\_startX - c\_startX)$
26: **while** $Math.abs(theta) < ((m\_revolutions + 0.5) * Math.PI * 2.0)$ **do**
27: $t \leftarrow Math.abs(theta)/((m\_revolutions + 0.5) * Math.PI * 2.0)$
28: $x \leftarrow c\_startX + (c\_endX - c\_startX) * t$
29: $y \leftarrow c\_startY + (c\_endY - c\_startY) * t$
30: $((GeneralPath)\ m\_shape).lineTo((float)\ (m\_avgRadius\ * (Math.cos(theta + startAngle)) + x), (float)\ (m\_avgRadius * (Math.sin(theta + startAngle)) + y))$
31: $theta = theta + 0.1$

```
32: (GeneralPath) m_shape).lineTo((float)m_endX, (float)m_endY)
33: strokeList is an array list of Stroke
34: stroke ← new Stroke()
35: pathIterator = new FlatteningPathIterator(
    m_shape.getPathIterator(new AffineTransform()), 1)
36: while pathIterator.isDone() = false do
37:     coords ← new float[6]
38:     pathIterator.currentSegment(coords)
39:     stroke.addPoint(new Point(coords[0], coords[1]))
40:     pathIterator.next()
41: strokeList.add(stroke)
42: mbeautified.setStrokes(strokeList)
43: if m_beautified instanceof IBeautifiable then
44:     ((IBeautifiable)m_beautified).setBeautifiedShape(m_shape)
45: return m_beautified
```