CPSC 547 Project Proposal

UCoD - Simplifying Supply Chain Structures in the Browser

Alex Trostanovsky
atrostan@cs.ubc.ca

Nikola Cucuk
nca3@sfu.ca

1 Introduction

NOTE:
Red Bold Text: Instructor's notes.
Green text: Author's Revisions.
Good! A bit more information on how and why you are visualizing the data and who the visualization is targeted for is necessary in the introduction. It's not so clear why there is a need for visualization in that writeup at this point. Kinaxis is a supply chain management company that models the product structures of its customers using graphs. To schedule a product structure, one must consider all of its constituent parts. In such a calculation, certain parts may depend on others. For example, the assembly of a bicycle depends on the assembly of a cassette, which itself depends on the assembly of individual cassette-cogs. These *one-way* dependencies can be calculated by maintaining that every dependant part relies on the scheduling of its parent parts.

However, applying the same logic without additional consideration to *multi-way* dependencies could cause deadlocks and conflicting schedules. In a *multi-way* dependency, multiple parts rely on a shared constraint. For example, a pair of different-sized cogs that get assembled on the same production line. If these parts were scheduled in isolation, the resulting production schedules could conflict with one another. To address this, parts that share common constraints are grouped into structures known as **Calculation Families**.

Inclusion in calculation families is a transitive operation that can result in the merging of large families. For example, if two calculation families contain parts that share common constraints, then the two families will be merged into a single calculation family (See Fig. 1). The scheduling algorithms used by Kinaxis to schedule

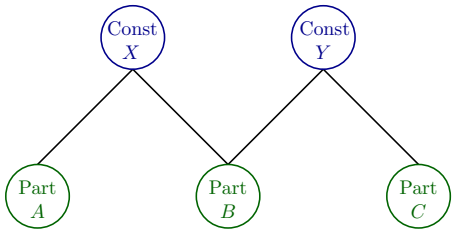


Figure 1: Merging of Calculation Families. Since parts A and B share constraint X, they must be in the same calculation family. Similarly, Parts B and C share constraint Y, so they must be in the same family. Therefore, Parts A, B, and C are grouped into the same calculation family.

their customers' product structures are parallelized across calculation families. Therefore, the fewer families that a customer has in their product structure(s), the slower the algorithms will be executed. Furthermore, having fewer families usually means that each family will be larger, further hindering parallel execution.

This project builds upon work completed in 2020 in collaboration with Carleton University's Computational Geometry Lab and Kinaxis with funding provided by the NSERC-Engage Alliance Grant. In said work, we defined a graphical model of calculation families, and automated the detection of underutilized constraints in calculation families using a recursive minimum-cut removal [16] algorithm. The mincut removal algorithm can efficiently detect underutilized constraints in calculation families and these results can be used by Kinaxis project managers to optimize supply chain scheduling. However, before committing to the removal of underutilized constraints from a calculation family, there would be an added advantage to the visualization of the structure of the families. Since calculation families may contain 1 000s of nodes and 10 000s of edges, developing a simplified visual representation that summarizes the calculation family graph and highlights the presence of underutilized constraints would help facilitate the interaction of Kinaxis Project Managers with these complex data structures.

We propose **UCoD** (The Underutilized Constraint Detector) - a tool that will allow Kinaxis' users to visualize, query, and search the calculation families in their product structures. This graphical representation of calculation families will take the form of an interactive, web-based, node link diagram that will display the high-level structure of calculation families and the properties of underutilized constraints. Given this information, the user may decide to remove the highlighted constraints from the graph. The removal of these constraints from the graph will split up calculation families substantially, while still maintaining a valid and feasible scheduling plan. As a result, the parallel scheduling of the (now smaller) families will be faster.

2 Related Work

there is a large and vast span of related work in visualization that targets similar problems and proposes similar solutions. After reading that subsection it's not all obvious which of the many approaches you mention might be relevant to your project - it reads like a general overview of how to draw graphs, rather than something targeted for your work. You might need to focus more on large graphs. Some suggestions below:

- **Link**
- **Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges. Von Landesberger et al.**
- **Abello et al. 2006: Ask-graphview: A large scale graph visualization system**
- **Archambault et al. 2007 Grouseflocks: Steerable exploration of graph hierarchy space**
- **Archambault et al. 2008 Grouse: Feature-based, steerable graph hierarchy exploration**
- **Archambault et al. 2009 Tuggraph: Path-preserving hierarchies for browsing proximity and paths in graphs**
- **Gansner et al. 2005: Topological fisheye views for visualizing large graphs**

NOTE:
The following Related Work Section will undergo a significant rewrite to address our review of the papers seen and comments made above.

2.1 Graph Visualization

Traditionally, graphs are visualized using Node-Link diagrams or Matrix Views. Node-link diagrams use nodes as point markers and connecting edges as line markers. Data attributes are encoded using colour, size, and shape channels for both nodes and edges [12]. Node-Link and its variants (Triangular vertical, spline radial, rectangular horizontal, bubble tree [15]) are often used to display graphs. Several different dimensional layouts have been explored.

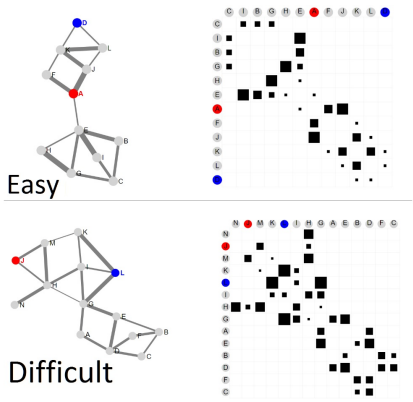


Figure 2: A visualization example of a Node-Link diagram mapped to a matrix [2].

Chord Diagrams lay the nodes on the circumference of the circle. Arc diagrams lay the nodes along one dimension [13].

In this work, we'll represent node-link diagrams in 2D space. These diagrams work well with small, sparse networks, but are poorly readable with large and highly connected graphs. Plotting large networks becomes difficult when nodes are placed in a constrained 2D space. A reoccurring problem that is observed when plotting large networks is the production of "hairballs" [15] - described as a visual clutter of occluded nodes and edges in large, dense graphs. To address this, node placement could be done using force-directed approaches [6, 8]. By assigning repelling forces to edges and nodes based on their relative position, these algorithms attempt to maintain that all edges are of more or less equal length and that there are as few crossing edges as possible.

Forced layout algorithms have difficulty converging to a placement solution in a reasonable run-time. Also, because the placement of nodes is not deterministic, identical layout reproduction can be challenging. Force-directed placement algorithms search in a way that can get stuck in local minima that may not be the optimal solution to the placement problem. To address these issues, multilevel force-directed placement algorithms have been developed [6]. These algorithms augment the original network with a derived cluster hierarchy to form a compound network. The cluster hierarchy is computed by coarsening the original network into successively simpler networks. This approach is better at avoiding local minima and can provide reproducible general placements. Finally, the run-time of multilevel force-directed placements is also improved because individual clusters are redefined independently as smaller networks.

The adjacency matrix view (AMV) is another way to visualize graphs. The rows and columns of this matrix are indexed by the nodes in the graph. If an edge e_{ij} exists between nodes $v_i, v_j \in G(V,E)$ (where $G(V,E)$ is the graph, and V,E are the sets of nodes and edges in G), then the i, j^{th} entry in the AMV will be filled. These entries can be filled with a boolean, an edge weight, or a colour to encode an edge attribute [13]. Such an implementation can achieve high information density, up to a limit of one thousand nodes and one million edges. An aggregated multilevel matrix view could handle up to ten billion edges [15]. While an NL diagram requires the entirety of the graph to be shown, only half of the AMV needs to be shown for an undirected graph, because a link from node v_i to node v_j implies a link from v_j to v_i [15]. Matrix views don't suffer from non-deterministic placement, because the area and the dimensions of the matrix are fixed. One major weakness of matrix views is unfamiliarity: most users can easily interpret NL views but not matrix views [15].

2.2 Supply Chain Management

Citation for beginning claim of 2.2? How and why does visualization help this? Good start in 2.2. Keep in mind that even if there are related problems in supply chain management, you should also consider whether there are related solutions in other fields that could be applied. Note it's "force-directed layout" not "forced layout"

Visualizing supply chains helps the user to:

1. Simplify the supply chain network (for example, by using node and edge contraction [9]),
2. Identify transportation bottlenecks or geographic concentrations [1],
3. Find alternate supply chain structures [1].

Traditionally, supply chains are represented as directed graphs or "netchains" [11]. In situations where heterogeneous data-sets are available, visual analysis reduces the user's cognitive load and expedites exploration by projecting emergent relationships between entities [3]. However, representing supply chains using visual interfaces is still in a nascent stage. We will explore relevant literature related to understanding and communicating the underlying structure of supply chains.

Supply chains have been an area of interest for a long time for the visualization community. An example of such work is representing supply chain interactions within a causal loop diagram [14]. Another example developed a framework for visually representing the geographical attributes of a supply chain using a case study from the transport container industry [7]. Furthermore, there has been a push to include geo-spatial data (GIS), in order to visualize supply chains across multiple dimensions [4] In this paper, the supply chain data-set does not have GIS data available, so this option will not be explored. On another note, Kassem et al. [10] developed a visualization scheme for mapping relevant information to the progress of building construction. In this project, we focus on developing a visualization scheme that would aid the user to identify underutilized constraints.

3 Dataset and Tasks

3.1 Dataset and Data Abstraction

Dataset, it's not clear - do you already have this dataset? Do you have to obtain it or gain access to it in some way? Is there no set of nodes? Or is that included in the edges tables? Abstraction of candidate constraints for removal is a useful one to help influence what you will draw user attention to in your visualization tool. Data and task abstraction is partway between domain-specific and domain-agnostic — Task abstraction is in a slightly better domain-agnostic place than data abstraction. I recommend having a section where you describe the data in domain-specific terms, then go deeper and abstract the data into domain-agnostic language. Sec 3.1 is a little hard to follow. Also, for making a point like "Approximately 99% of the families contain less than 5000 parts (Fig. 3) and less than 300 constraints (Fig. 4)" you would be better served by something more like a cumulative distribution plot where that could be read directly off the chart.

Kinaxis has provided the authors with an anonymized customer's dataset. Utilization of the Family Separation Dataset was conditioned upon compliance to a Non-Disclosure Agreement between

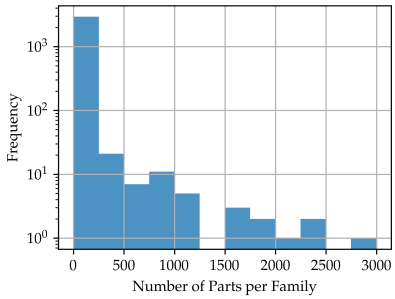


Figure 3: Figure will be revised to a Cumulative Distribution Plot

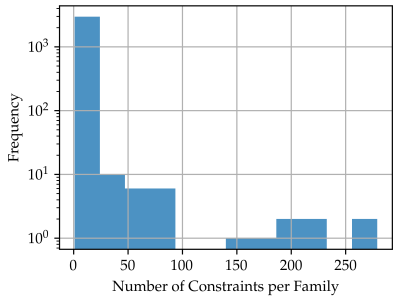


Figure 4: Figure will be revised to a Cumulative Distribution Plot

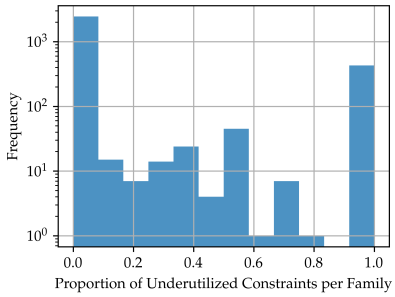


Figure 5: Figure will be revised to a Cumulative Distribution Plot

Kinaxis and the authors. Kinaxis' Family Separation Dataset contains 199 807 parts, 8 251 constraints, and 2 989 families. The dataset consists of three tables:

1. `Edges - Part to Constraint`: defines the edges between Parts and Constraints. Equivalently, which Parts depend on which Constraints. The Part and Constraint nodes are implied by this table.

2. `Edges - Part to Family`: defines which Parts belong to which Calculation Families. A key-value mapping between Families and their constituent parts.

3. `Constraint Utilization`: contains constraint utilization observations over time.

Every constraint utilization observation is attributed with the following features:

- The constraint *capacity*, $p \geq 0$: A scalar value that indicates the constraint's total capacity.

- The constraint *load*, $l \geq 0$: A scalar value that indicates how much of the constraint's capacity was utilized (in aggregate) by all of its dependant parts.

Using these attributes, the following metrics are derived:

- The constraint *utilization*, $u := l/p$.
  If $u \approx 0$ then that constraint is *underutilized*.
  If $u > 1$, then that constraint is *overutilized*.

- Due to the dynamic nature of constraint load and capacity, an important metric to observe is the *maximum constraint utilization*. This metric is defined to be the largest constraint utilization observed over the entirety of the dataset.

In a calculation family graph F, there exist two types of node sets:

- P - the set of *Part* nodes in the calculation family.

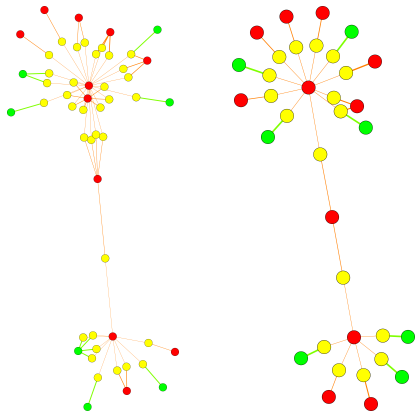
- C - the set of *Constraint* nodes in the calculation family.

Every part is connected to at least one constraint via an edge. This edge is used to model the dependency that exists between part p_i and constraint c_j. The weight of the edge between p_i and c_j is defined to be the **Maximum Constraint Utilization** for constraint c_j. If the maximum constraint utilization for a constraint c_i is below some predefined constant $0 < \varepsilon < 1$, constraint c_i is said to be *consistently* underutilized.

Modelling the edge weights in a family graph F as the maximum constraint utilization (over *all* entries in the dataset) was proposed by Kinaxis' algorithm developers. The justification for this choice was the fact that if a constraint is consistently well below its capacity, "then it may make sense to simply remove that constraint. Since it does not bottleneck production, it is not adding very much value" [5].
NOTE:
what is the proper way to cite a personal correspondence?

3.2 Part Contraction

Some edges and parts in the graphical representation of a family may be redundant and could be contracted to reduce family size. There may exist parts that are similar in the sense that they are connected to the same set of constraints. These "identical" part nodes can be contracted to form a single meta-node (See Fig. 6). The weight of the edge between the new meta-node and the constraint c_i remains u_{c_i} (since it was equal across all parts in P).



(a) A calculation family

(b) A contracted calculation family

Figure 6: Part Contraction. Red nodes are underutilized constraints. Green nodes are fully utilized constraints. Yellow nodes are parts.

3.3 Task Abstraction

The goal of this work is to propose the removal of constraint nodes that would both:

1. Significantly reduce the family size, and

2. Continue to represent a feasible and reasonable plan.

The set of candidate constraints for removal will be identified using the recursive minimum-cut removal algorithm developed in our previous work. The candidate constraints are constraint nodes in a family graph whose removal from the graph will result in two disconnected subgraphs of roughly equal size. These candidates will be displayed to the user, who may then decide to remove the proposed constraints from the family graph. If the family size is substantially reduced, then the scheduling of the partitioned family will be optimized. The new scheduling calculation induced by the split-up family will be monitored by Kinaxis' users to verify that it continues to represent a valid plan.
We intend UCoD to provide the following functionality:

1. At the highest level, the user will **derive**:
   - Constraint utilization metrics,
   - Aggregate Constraint utilization metrics, and
   - the Graph topology of Calculation Families.
2. Since the target (underutilized constraints) is *known*, but its existence and location in a family graph is *unknown*, the user will **locate** underutilized constraints in the graph.
3. The user will **select** constraints that are connected to multiple ("identical") parts, and contract all parts associated with said constraints. This **aggregation** will simplify the visual representation.
4. The user will **manipulate** the value of $\varepsilon \in (0,1)$ (the constant defining constraint underutilization) and this variation will be dynamically represented in the graph. Different values of ε produce different family separations. Allowing the user to tweak this parameter could result in the proposal of candidate constraints for removal that would not have been possible with a hard-coded ε value.

4 Solution: your proposed InfoVis solution.

Sketches or potential prototypes will help a lot here. I suspect interaction will play a huge role in the project, so the faster you can prototype that, the better shape your project will be in. While not necessary for the updates, for the final report, you will have to note more details on the implementation: what was difficult, potential roadblocks, and a description of why you chose the tools that you did. Scenario of use - eventually will want to split apart into a different section, but very good start as a driving scenario.

4.1 Visual Encoding and Idiom

We display Calculation Families as Node Link Diagrams. We accompany this display with an adjacency matrix view. Point Marks will signify nodes and line marks will signify edges between parts and constraints. The channels we will use to distinguish between the type of nodes in the graphs and the constraint utilization magnitude are not finalized.
NOTE:
Still a work-in-progress. We continue to experiment with the different marks and channels we will use to represent:

- Part nodes
- Constraint Utilization magnitude
- Candidate Constraints for Removal

4.2 Implementation

1. Data ingestion and preprocessing: `pandas`
2. Graph construction and layout: `igraph`, `graphviz`
3. Graph rendering, visualization: `sigmajs`

4.3 Scenario of Use

Andrea is a Solution Architect at Kinaxis. She has been tasked with the integration of a new client's supply chain into Kinaxis' concurrent planning platform. Given the constraint utilization history and projections in the customer's product structure, she constructs a Calculation Family dataset. However, on the first few iterations of the scheduling calculations, she notes that the calculations take too long. Andrea suspects that some of the calculation families in the dataset might be too large. She wonders whether the client has erroneously included *underutilized* constraints in certain families, and whether removing said constraints would speed up the parallel scheduling calculations. To find out whether this is true:

1. Andrea uploads the new client's Calculation Family dataset to UCoD.
2. The recursive minimum-cut removal algorithm returns that Family f_1 contains an underutilized constraint whose removal from the graph will split up the family substantially.
3. Andrea selects this family in UCoD to visualize it (See Fig. 7).
4. Andrea is unable to view the occluded and unordered clusters of constraints and parts in the family, so she decides to apply the `sfdp` layout to the graph.
5. Andrea further simplifies the graphical representation by clicking `Contract All` to contract all underutilized constraints in Family f_1. (See Fig. 8).
6. Working with this simplified graph, Andrea can click on Constraint c_1 to remove it from the graph. She sees that this removal produces two disconnected components. Also, before removing Constraint c_1, Andrea queries the attributes of Constraint c_1 and sees that the last constraint utilization observation associated with this constraint was 6 months ago.
7. Andrea's findings allow her to conclude that Constraint c_1 can be safely removed from Family f_1, and updates the scheduling calculation that includes this family.

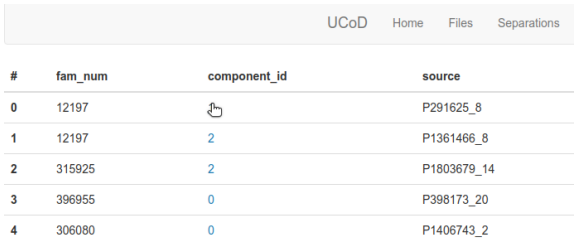


Figure 7: Current User Interface of UCoD. This endpoint allows the user to select which family they would like to inspect.

5 Milestones

See Table 1. **Seems like a lot of the work is on Alex as an owner, do you need divvy up the work in a more reasonable way? You don't have any time estimates for things like writeup and presentation preparation, do include those as well so that you have a complete picture of time required. You've only accounted for <60 hours between two people, so there's definitely some missing terrain.**

- **Red** milestones are mandatory for project completion.
- **Blue** milestones would enhance UCoD's aesthetic, functionality, and/or interactivity.

Style - a few cites sloppy [2, 4, 10]

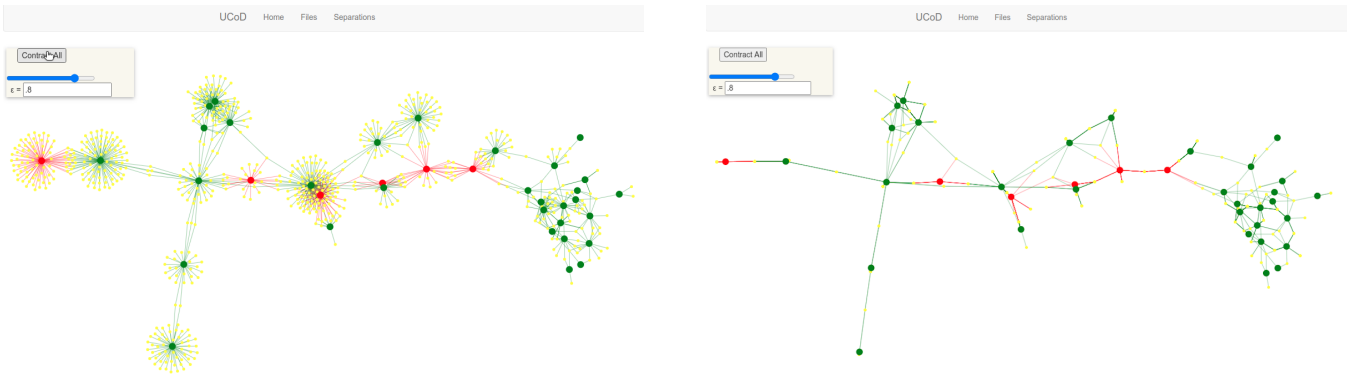


Figure 8: Current implementation of Graph Contraction in UCoD

REFERENCES

[1] William Z Bernstein, Devarajan Ramanujan, Niklas Elmqvist, Fu Zhao, and Karthik Ramani. Viser: Visualizing supply chains for eco-conscious redesign. In *Intl. Design Eng. Tech. Conf. and Computers and Information in Eng. Conf.*, volume 46353, page V004T06A049. American Society of Mechanical Engineers, 2014.

[2] Chunlei Chang, Benjamin Bach, Tim Dwyer, and Kim Marriott. Evaluating perceptually complementary views for network exploration tasks. In *Proc. 2017 CHI Conf. Human Factors Computing Systems*, pages 1397–1407, 2017.

[3] Tera Marie Green, William Ribarsky, and Brian Fisher. Visual analytics for complex concepts using a human cognition model. In *2008 IEEE Symp. Visual Analytics Science and Technology*, pages 91–98. IEEE, 2008.

[4] Jeff Greer et al. Gis: The missing tool for supply-chain design. *Foresight: The International Journal of Applied Forecasting*, (28):44–49, 2013.

[5] John Howat. Personal Communication.

[6] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[7] Zhi-Hua Hu, Bin Yang, You-Fang Huang, and Yan-Ping Meng. Visualization framework for container supply chain by information acquisition and presentation technologies. *J. Software*, 5(11):1236–1242, 2010.

[8] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one*, 9(6):e98679, 2014.

[9] Ying-Jer Kao, Yun-Da Hsieh, and Pochung Chen. Uni10: An open-source library for tensor network algorithms. In *Journal of Physics: Conference Series*, volume 640, page 012040, 2015.

[10] Mohamad Kassem, Nashwan N Dawood, Claudio Benghi, Mohsin Siddiqui, and Donald Mitchell. Coordinaton and visualization of distributed schedules in the construction supply chain: A potential solution. In *10th Intl. Conf. Construction Applications of Virtual Reality*, pages 77–86. CONVR2010 Organizing Committee, 2010.

[11] Sergio Lazzarini, Fabio Chaddad, and Michael Cook. Integrating supply chain and network analyses: the study of netchains. *Journal on chain and network science*, 1(1):7–22, 2001.

[12] Vaden Masrani. Sapvis: An interactive system explorer.

[13] Michael J McGuffin. Simple algorithms for network visualization: A tutorial. *Tsinghua Science and Technology*, 17(4): 383–398, 2012.

[14] Shotaro Minegishi and Daniel Thiel. System dynamics modeling and simulation of a particular food supply chain. *Simulation practice and theory*, 8(5):321–339, 2000.

[15] Tamara Munzner. *Visualization analysis and design*. CRC press, 2014.

[16] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *JACM*, 44(4):585–591, 1997.

Name	Description	Owner	Estimate	Due Date	Update
Interactive Constraint Contraction	1. The user is able to select a constraint and click it to contract all of its dependent parts. 2. The user can contract all underutilized constraints by clicking `Contract All`	Alex	16 hours	Nov. 15	Complete
Varying Epsilon Represented in Graph	1. A slider can be manipulated to vary the amount of $0 \leq \varepsilon \leq 1$. Varying this amount is dynamically rendered in the browser.	Alex	16 hours	Nov. 22	In progress
Adjacency Matrix View	1. The node-link view of a calculation family is accompanied by the adjacency matrix view.	Nikola	16 hours	Nov. 22	In progress
User interface	1. The families containing candidate constraints are can be chosen (clicked) by the user.	Alex	16 hours	Nov. 15	Complete
Proposal Update	1. Revise project report based on notes	Alex, Nikola	4 hours	Nov. 18	In progress
Implement the visual encoding of Candidate Constraints for Removal	1. Upon loading a family graph in UCoD, the user's attention should be immediately directed to the candidate constraint for removal. 2. Potentially, redundant encodings could be used to signify the importance of this data point. For example, a combination of shape, colour, size, and/or movement (jitter).	Alex, Nikola	10 hours	Dec. 1	In progress
Force directed placement of contracted families	1. Maintain two versions for each family to be displayed 2. This will allow: • users to revert between contracted and uncontracted versions of families. • visualize a force directed placement of contracted families	Alex, Nikola	10 hours	Dec. 1	In progress
Presentation Preparation	1. Prepare Presentation slides 2. Rehearse Live Presentation	Alex, Nikola	8 hours	Dec. 10	Not started
Final Report	1. Complete literature review (Recommended papers) 2. Rehearse Live Presentation	Alex, Nikola	16 hours	Dec. 14	Not started

Table 1: Project Milestones