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

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

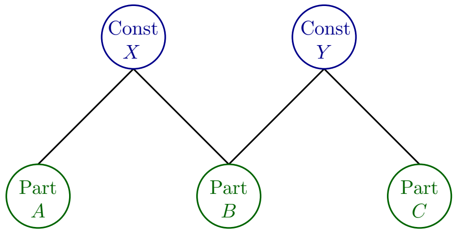


Figure 1: Merging of Calculation Families. Since parts A and B share constraint X, they must be in the same calculation family. Similarly, Parts B and C share constraint Y, so they must be in the same family. Therefore, Parts A, B, and C are grouped into the same calculation family.

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.

We propose **UCoD** (The Underutilized Constraint Detector) - a tool that will allow users to visualize, query, and search the calculation families in their product structures. This graphical representation of calculation families will facilitate the detection of underutilized constraints. 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.

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 [14] algorithm.

2 Related Work

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 [10]. Node-Link and its variants (Triangular vertical, spline radial, rectangular horizontal, bubble tree [13]) are often used to display graphs. Several different dimensional layouts have been explored.

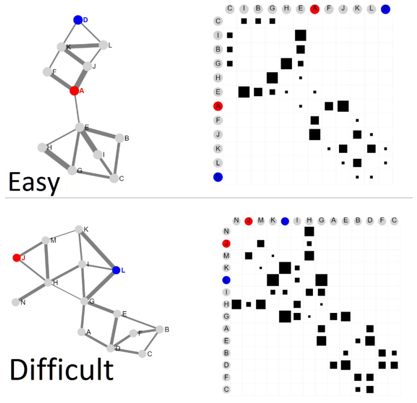


Figure 2: A visualization example of a Node-Link diagram mapped to a matrix [1].

Chord Diagrams lay the nodes on the circumference of the circle. Arc diagrams lay the nodes along one dimension [11].

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" [13] - 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 [5, 7]. 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 [5]. 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 [11]. 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 [13]. 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 [13]. 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 [13].

2.2 Supply Chain Management

Visualizing supply chains helps the user to:

1. Simplify the supply chain network,
2. Identify transportation bottlenecks or geographic concentrations, and
3. Automatically find alternate supply chain structures.

Traditionally, supply chains are represented as directed graphs or "netchains" [9]. 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 [2]. However, representing supply chains using visual interfaces is still in a nascent stage. We explore relevant literature related to understanding and communicating the underlying structure of supply chains.

Minegishi and Thiel [12] used causal loop diagrams to display supply chain interactions. Greer et al. [3] include geo-spatial data (Geographic Information System mapping - GIS) to visualize supply chains using their geographical coordinates. Hu et al. [6] developed a framework for visually representing geographical attributes of a supply chain using a case study from the transport container industry. Kassem et al. [8] developed a visualization scheme for mapping information relevant to the progress of building construction.

3 Dataset and Tasks

3.1 Dataset

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`. This table maps Parts to Constraints.
2. `Edges - Part to Family`. This table maps Parts to Calculation Families.
3. `Constraint Utilization`.

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.
- The constraint *bucket*: the date of the capacity and load observations.

Constraint	Load (l)	Capacity (p)	l/p	Bucket
c_1	10	20	0.5	01/01/2020
c_2	0.1	20	0.005	01/01/2020
c_1	11	20	0.55	02/01/2020
c_2	21	5	4.2	02/01/2020

Table 1: Constraint Metrics Observations Example. c_1 is underutilized in both buckets. Since the load on c_2 is increased and its capacity is decreased on bucket 02/01/2020, it is overutilized during that bucket. Note that for this set of observations $\max u_{c_1} = 0.55$, $\max u_{c_2} = 4.2$.

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 a sequence of bucket observations (See Table 1).

Figures 3, 4, and 5 show the distributions of Part, Constraint, and Underutilized Constraint counts across families. Approximately 99% of the families contain less than 5000 parts (Fig. 3) and less than 300 constraints (Fig. 4).

3.2 Data Abstraction - Modelling Calculation Families as Graphs

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" [4].

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

1. Significantly reduce the family size, and

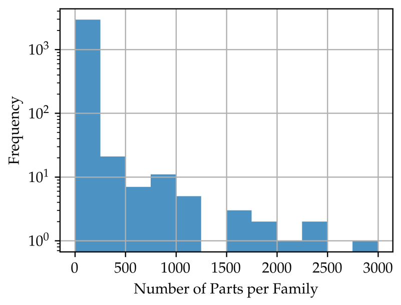


Figure 3: Distribution of Part Counts per Family

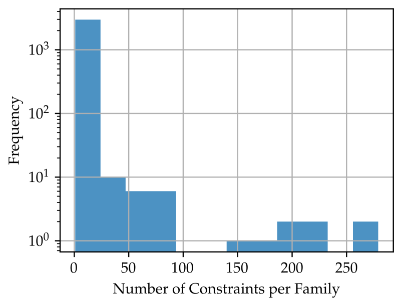


Figure 4: Distribution of Constraint Counts per Family

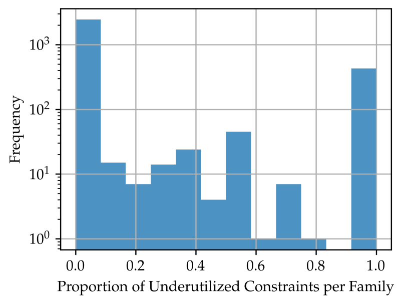


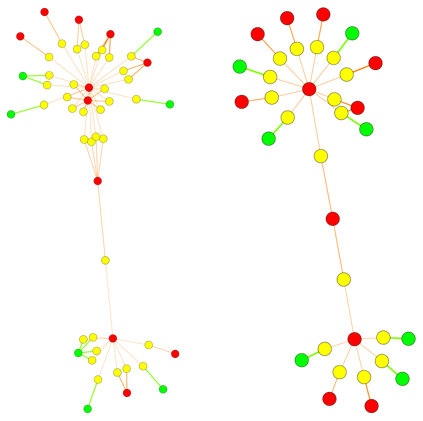
Figure 5: Proportion of Underutilized Constraints per Family

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. 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.

3.3 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.

3.4 Task Abstraction

We intend UCoD to provide the following functionality:

1. At the highest level, the user will **derive**:
   - Constraint utilization metrics,
   - Aggregate Constraint utilization metrics (max, mean, with a focus on the possibility of outlier observations in constraint utilization), 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.

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.

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.

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. 7).

6. Working with this simplified graph, Andrea can click on Constraint c_1 (See Fig. 10) 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.

5 Milestones

- **Red** milestones are mandatory for project completion.

- **Blue** milestones would enhance UCoD's aesthetic, functionality, and/or interactivity.

1. **Interactive Constraint Contraction.**
   Description:

   (a) The user is able to select a constraint and click it to contract all of its dependent parts.

   (b) The user can contract all underutilized constraints by clicking `Contract All`

   Owner: Alex.
   Estimate: 8 hours
   Due Date: Nov. 1

2. **Varying Epsilon Represented in Graph.**
   Description:

   (a) A slider can be manipulated to vary the amount of $0 \leq \varepsilon \leq 1$. Varying this amount is dynamically rendered in the browser.

   Owner: Alex.
   Estimate: 16 hours
   Due Date: Nov. 15

3. **Adjacency Matrix View.**
   Description: The node-link view of a calculation family is accompanied by the adjacency matrix view.
   Owner: Nikola.
   Estimate: 16 hours
   Due Date: Nov. 15

4. **Data/Graph Input Handler.**
   Description:

   (a) The user is able to upload a constraint utilization dataset.

   (b) The recursive minimum cut removal algorithm computes candidate constraints for removal.

   (c) The families containing those constraints are displayed to the user.

   Owner: Alex.
   Estimate: 4 hours
   Due Date: Nov. 15
   Contingent on: 1, 2, 3

5. **Animate Yifan Hu's `sfdp` layout.**
   Description:

   (a) Apply the `sfdp` layout to a given calculation family.

   Owner: Alex.
   Estimate: 4 hours
   Due Date: Dec. 1

6. **Develop Extensible API utilizing `graphviz`' drawing library.**
   Description:

   (a) Allow the user to select from a variety of layout calculations (implemented in the `graphviz` library)

   Owner: Alex.
   Estimate: 8 hours
   Due Date: Dec. 10
   Contingent on: 5

References

[1] Chunlei Chang, Benjamin Bach, Tim Dwyer, and Kim Marriott. Evaluating perceptually complementary views for network exploration tasks. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 1397–1407, 2017.

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

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

[4] John Howat. Personal Communication.

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

[6] 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.

[7] 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.

[8] 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 International Conference on Construction Applications of Virtual Reality*, pages 77–86. CONVR2010 Organizing Committee, 2010.

[9] 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.

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

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

[12] 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.

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

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

Appendix A Scenario Illustrations

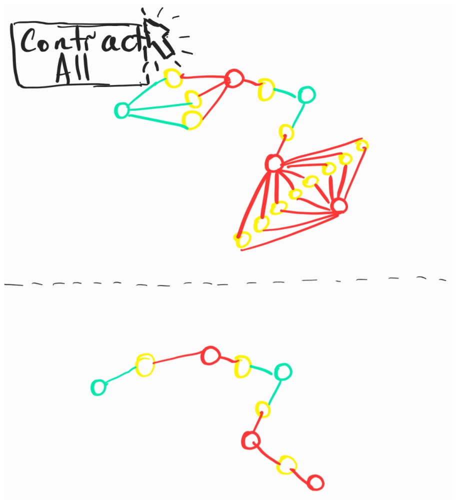


Figure 7: Graph Contraction

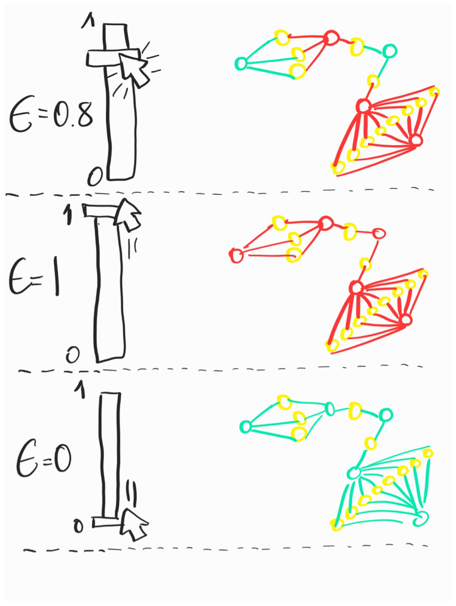


Figure 8: Variation of epsilon and its effect on the graph

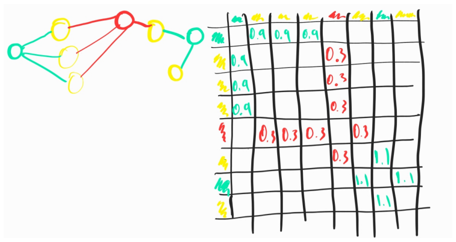


Figure 9: A simple calculation family accompanied by the adjacency matrix view. (Visual encoding of matrix cells and column and row headers are not finalized).

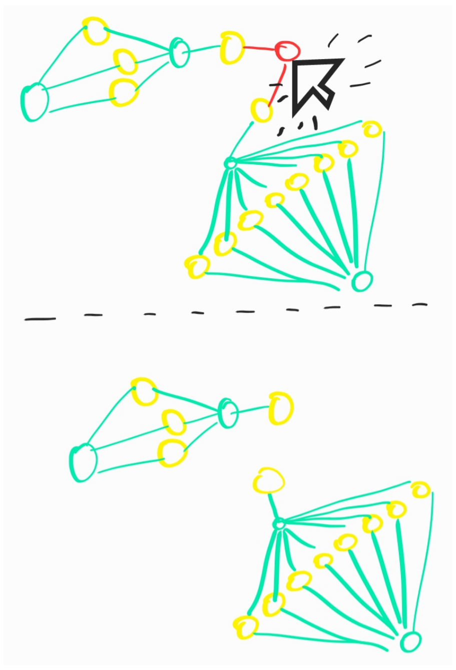


Figure 10: Removal of an underutilized constraint from the graph.

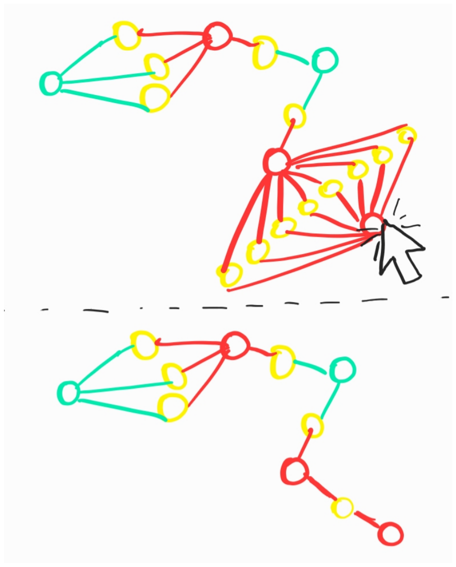


Figure 11: Contraction of a single underutilized constraint in the graph.

Appendix B Planning Roadmap

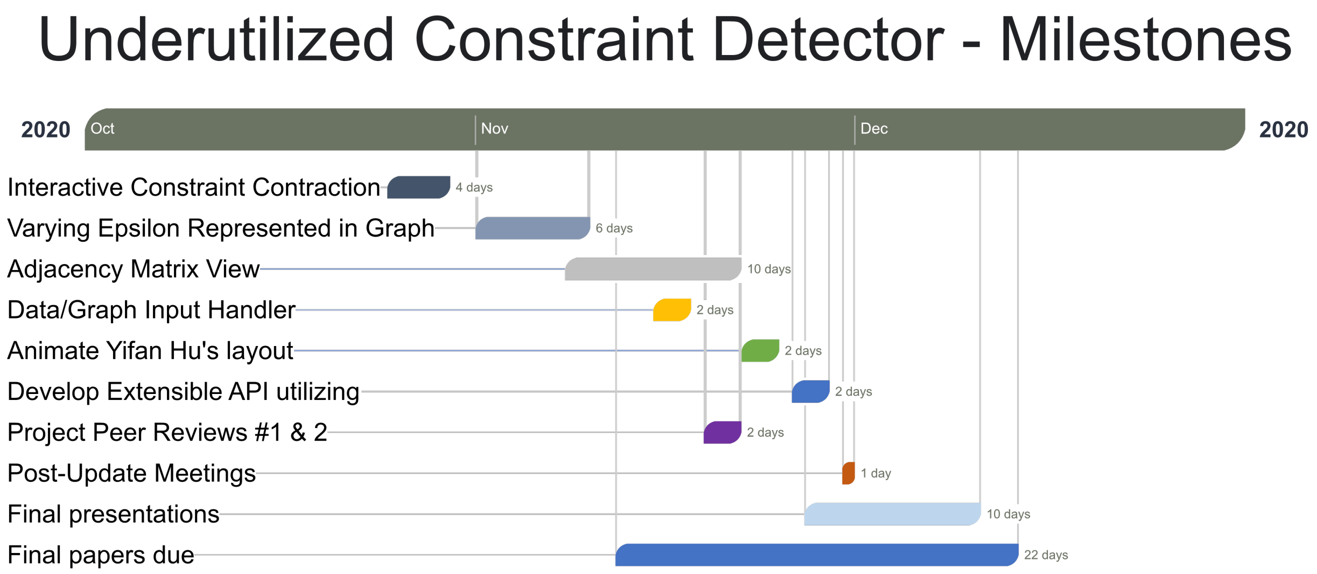


Figure 12: A visualization of the project's timeline.