Thesis

# Introduction

Puzzle – other man – “Brothers and sisters I have none, but this man’s father is my father’s son”.

This puzzle’s solution is “my son”. However in this particular situation, with greek mythology, it could be something far more obscure – divine father, disputed father.

# Graph data formats considered

## Straight JSON

Easy to import the current relationship data in

## DOT Notation

## GraphLib

Not directly a data representation format, however is used to store the information about nodes and edges.

Contextualises the information into nodes and edges, saves having to generate this ourselves. Has a lot of implicit functions already provided for finding edges and nodes, and formatting them etc.

However does not create an actual database – we generally use an existing database of information to use GraphLib. Difficult to understand the complexities of this kind of data.

However upon investigation this seems to be the best method of storing our information. A combination of JSON converted information from the original CSV file, to graphlib objects.

## GEDCOM

The standard format for genealogical data.

(GEnealogical Data COMmunications)

Developed by the Church of Jesus Christ and Latter-Day Saints. Used for recording baptisms(?)

Text file.

**Pros:**

Recognised standard for genealogical data

**Cons:**

# Types of graph layout software / graph visualisation tools considered

## GraphViz + Dot

<https://stackoverflow.com/questions/2271704/family-tree-layout-with-dot-graphviz> – making it so it looks like t-junctions.

## DagreJS asnd D3.js + SVG?

Really good

Allows for interactable elements – good for linking back to the source content and highlighting the individual unusual relationships for focus.

Unfortunately is impossible / difficult to format the existing layout algorithm. And also genealogical graphing is not accessible.

Automatic formatting. Also doesn’t do the t-relations required in genealogical charts.

## Canvas.js

No direct visualisation algorithm for directed acyclic graphs

But can use as a blank canvas for my own graph layout algorithms

Allows for interactable elements of the graph, see canvas.js layouts

## OrgChart

<https://github.com/dabeng/OrgChart>

Simple family tree generator

## Legacy 9 – not PC compatible

Accessible for different types of data input, for example can import GEDCOM text files.

**Pros:**

* Relatively intuitive. Used to import information about general genealogies.
* Can put as much information as necessary into the graph (e.g. can just input first name, date of birth is optional, etc.)
* Can input the number of generations wanting to show
* Automatically displays the information in genealogical format
* Can determine whether to show ancestry or descendants
* Can choose which entity is the focus entity (usually the left-most one)
* Imports existing file data such as GEDCOM
* Allows for different genealogical (family tree) formats, e.g. traditional nodes and edges, or ancestry fan
* Allows for different layouts depending on page size (e.g. horizontal and vertical family tree representation)
* Immediately obvious is a genealogy chart
* Allows for mistresses, not just spouses

**Cons:**

* Can’t show descendants and ancestry in the same graph? – can (X-chart)
* Costs money
* Can’t modify the layout
* Layout algorithms not directly accessible and not able to be converted (no API) for putting that data on the screen – generate dynamically, is not interactable
* Assume every entity is human (but can name them “void”)
* Can handle brother and sister relationships, but not inter-generational relationships (integral to greek mythology)
* Can’t handle disputed parents (e.g. multiple mothers, disputed)
* Can’t list alternative parents (‘other parents’) and cannot express uncertainty/dispute in the graph

## Roots (MacOSX)

Not built for Mac Catalina – currently in development

# Different layout algorithms

## Binary tree structure

**Pros:** commonly associated with genealogy, shows hierarchies and generations

**Cons:** Don’t deal with strange intergenerational connections. Certain that each entity will have more than one parent, and will possibly have more than 2 children. G tree is not valid either as does not deal with intergenerational.

## Cyclic graphs & Bidirectional Acyclic Graphs, Directed Acyclic Graphs,

Not possible. Person can’t be their own grandparent.

Needs to have both direct and undirected components. Don’t need to do operations on them yet so direction does not matter.

Maybe allow specification of directed or undirected connection? Hybrid?

## Force-directed layout for DAG

**Pros:**

Standard visualisation format for DAGs. Readily available for use and experimentation in different visualisation software, esp. D3.

**Cons:**

Nothing like genealogy graphs. Very few of them have formatting for genealogies. Need the hierarchical structure since otherwise it is unintuitive and doesn’t look like a genealogy chart (connections not immediately obvious)

Tried this, worked well as an early prototype but was clear that this was unintuitive. So started from scratch with a bespoke algorithm.

## Hierarchical layout for DAG

**Pros:**

Significantly closer to expectations for genealogy charts.

**Cons:**

Generally does not have support for more unusual relationships (e.g. intergenerational).

## Ancestor Fan

**Pros:**

Another well-known version of genealogy charts.

Could be interactable.

Immediately obvious who is the center entity / point of focus.

**Cons:**

Would not be able to show intergenerational incest, and is difficult to read.

Would be hard to modify

# Different relationship types considered

Insert the screenshots of the digitized plans for showing relationships.

A screenshot of a cell phone

Description automatically generated

# Early prototypes:

Prototype developed using GraphLib, DagreJS + D3.js and rendered as an interactable SVG format. Used to determine the pros and cons of the various data formats considered, used as inspiration for the bespoke graphing tools and graph formats.

Existing relationship data reformatted into GraphLib obejcts, and then converted to graph visualisation using the DagreJS + D3 graphing tool that creates a hierarchical graph layout. Shown on the page as an interactable SVG format

A picture containing text

Description automatically generated

Graph allows for automatic hierarchical formatting and placement. As you can see, all of the parents of Apollo’s children are in the same horizontal placement as Apollo. Also each node (thanks to GraphLib) can have “parent” attached to it to allow automatic detection of hierarchies, and the edges can “appear” as bi-directional if you want by omitting the arrow in the style tag. However the siblings of Apollo were put on the same level as his children, and there was no way to change this.

Is highly interactable, scrollable, zoomable and scalable so is very usable. By clicking on the nodes you can go to the entity’s page. However, the layout of the graph itself cannot be modified easily, and it also lacks the t-connections that genealogy charts are known for. As this is the case, it’s difficult to distinguish between siblings and mistresses and spouses, and even more confusing to add things like intergenerational incest. Both parents of an entity have to be inferred.

I added a superficial solution which is to add labels to each of the edges – makes the edge type more explicit (child vs sibling vs other parent), but this is not very user-friendly, especially if there are lots of different kinds of labels in the same location. For example, Zeus has a huge number of mistresses and offspring. This was impossible to read, even if the diagram was scrollable and interactable. The graph layout algorithm for D3 is also not customizable, so we can’t find alternative layout formats for these graphs.

A screenshot of a social media post

Description automatically generated

GraphLib also has removes duplicates by default – counter-productive for duplicate relationships such as incest. For example, Zeus and Hera are siblings but also husband and wife. So Hera node is created, Zeus node is created, and then the algorithm encounters “Hera is wife of Zeus” so the edge Zeus->Hera with label “wife” is added to the graphlib object. However, then the algorithm encounters “Hera is sibling of Zeus” and instead of creating a new edge between Zeus and Hera with the label “sibling”, it just overwrites the existing “wife” label. As such the resulting graph will only have one edge between Hera and Zeus – “sibling”.

Of course you could modify it so that “sibling” is appended to the existing “wife” label to create “sibling/wife”, but with this it’s not immediately obvious that there is an unusual connection there. We want to make it immediately obvious to the user that it’s unusual. As such we couldn’t get these unusual relationships to work in the prototype.

A close up of a mans face

Description automatically generated

We did create an easier prototype for unusual aspects in greek myths – disputed parents. These edges were different colours to signify that it’s not a usual connection, and the links were clickable, but only upon request.

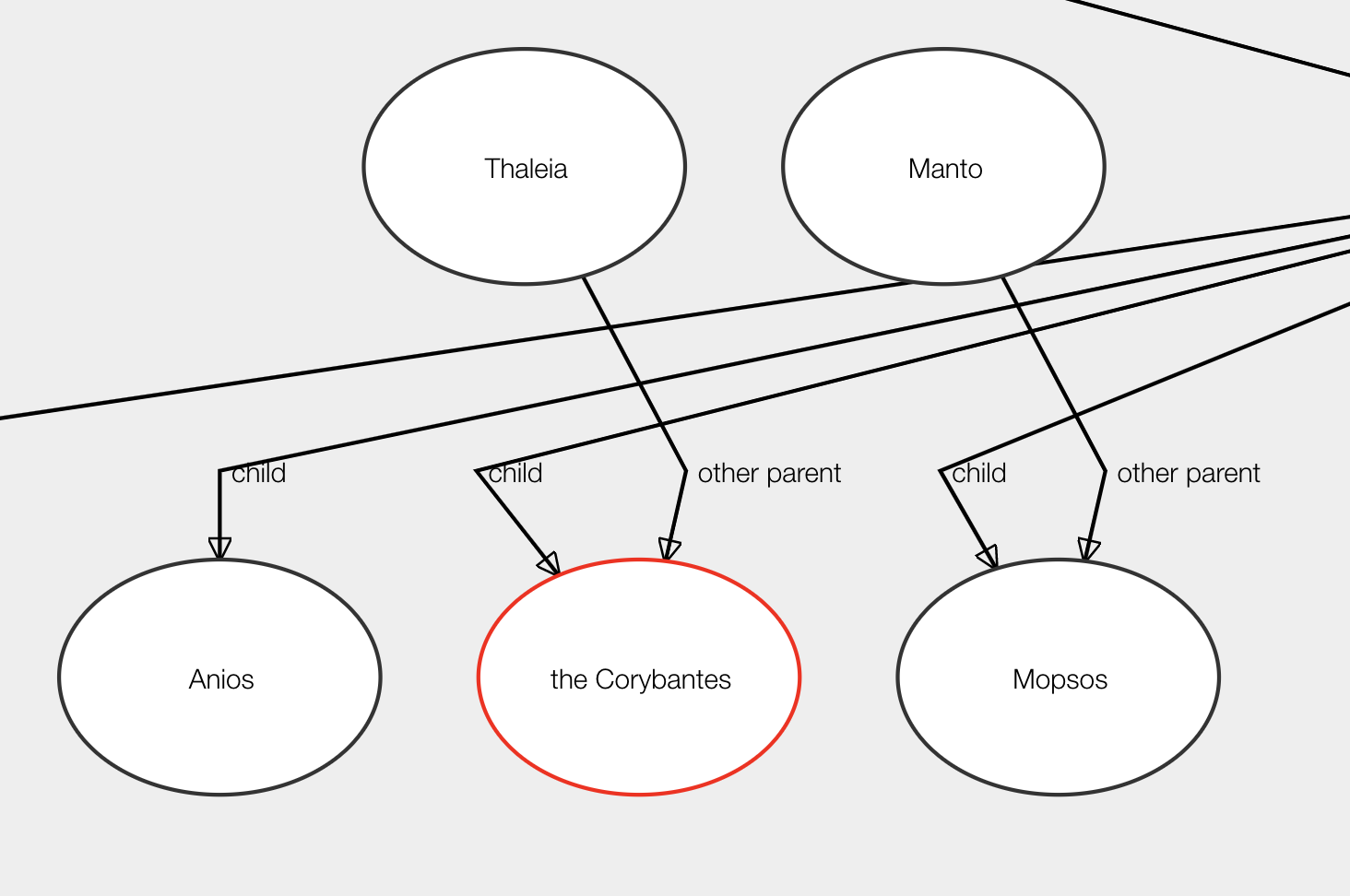
By clicking on the disputed edge, a new page would pop up detailing the unusual parts of this relationship, with links to the location in Scaife of the disputes, and as well as links to their entity pages.

A picture containing screenshot

Description automatically generated

There was also a brief addition of member nodes – signifying the differences between the collectives and the parent nodes.

Originally each of the collective pages contained a graph with the members of the collectives as \*children\* in dotted lines, however this was removed as it was deemed irrelevant to the genealogy by Greta.



The following collective graphs were removed. As you can see they were barely useful for showing collectives that were genealogical, such as Danaids: the daughters of Danaos. However they were not at all useful for other kinds of collectives, such as the Greek Contingents at Troy.

A close up of a device

Description automatically generated

^ Usefulish

A close up of a device

Description automatically generated^ Not at all useful

So decided to get rid of them.