Local separators in large networks

Masters Thesis

Supervisors: Johannes Carmesin and Samuel Johnson

Iyassou Shimels Kindie
Student ID Number: 1979865

May 2022

Abstract

In a recent graph theory paper Carmesin introduced local separators, a local analogue to separators in graphs. Local separators define the connectivity of smaller induced subgraphs within a graph and are therefore interesting to observe within large, connected graphs. In this paper we examine and interpret their presence in select large networks modelling real-world datasets using the Python programming language and the NetworkX package.

Contents

1 Introduction

Graph theory is a field rich in practical applications due to how real world structures and problems lend themselves easily to graph theoretical modelling. The underlying problem is then solved using an algorithm whose time complexity, in particular, is dependent on the number of vertices or edges in the graph. A notable example is Dijkstra's algorithm, which finds the shortest path between vertices in a graph $G = (V, E)$, with time complexity $O(|E| + |V|^2)$ [9].

As with the aforementioned example, the underlying problem that we're aiming to solve has an associated algorithm whose time complexity is dependent on the number of vertices. Therefore for large graphs, which arise in practise and are of interest due to the amount of information they represent, running these algorithms on the entire graph becomes infeasible. One approach would be to relax our search for an exact global optimum by opting for a divide-and-conquer strategy whereby we decompose the graph into smaller pieces, solve the problem on these more manageable pieces, and finally combine these locally optimal solutions into an approximate global optimum.

We then ask ourselves how should we decompose a graph? One approach would be to find critical points in the graph whose removal disconnects the graph into smaller pieces: these are known as cutvertices. By then replacing these smaller pieces with representative vertices, we can run our favourite algorithm on this more tractable graph. This is the idea behind the block decomposition of a graph [11].

In large, complex networks however it is unlikely that we'd find enough cutvertices to break the graph down into much finer pieces due to their connectivity. For a graph modelling one such network, how then can we analyse its resiliency to local perturbations or break it down into smaller pieces?

2 Background Knowledge

Unless specified otherwise, we shall refer to the undirected, unweighted graph $G := (V, E)$ in our definitions and lemmas. With the aim of making our notation explicit, let $u, v \in V$. $N_G(v)$ denotes the neighbourhood of v in G, $d_G(v)$ the degree of v in G, and $d_G(u, v)$ the length of a shortest path from u to v in G. If there is no path between u and v in G, we set $d_G(u, v) = \infty$.

For $X \subseteq V$, $G[X]$ denotes the induced subgraph of G on X, and $E(X)$ denotes the edges of $G[X]$. For $X, Y \subseteq V$, we define $E(X, Y)$ to be the edges of the form $\{x, y\}$ where $x \in X$ and $y \in Y$.

P^k and W^k denote the path and wheel graphs on $k+1$ vertices respectively, whereas C^k and K^k denote the cycle and complete graphs on k vertices respectively. For $k \in \mathbb{N}$ we define $[k] := \{1, \ldots, k\}$. For W^k we assume the central vertex is 0 and the rest $[k]$, otherwise more generally for a graph G on k vertices we assume its vertices to be $[k]$.

2.1 Network theory

When a real-world problem can be modelled into a graph where edges represent relations, the study of the resulting graph becomes the focus of network theory. The analysis of such networks has a large breadth of multi-disciplinary applications.

For the purpose of analysis, network theory concerns itself with certain graph metrics such as the characteristic path length and (local) clustering coefficient. In their paper,

Watts and Strogatz investigate how to quantity the small-world property of a graph by observing how these two graph properties varied when randomly rewiring an edge with probability p in a regular ring lattice graph [51]. Small-world networks are graphs where all pairwise paths between vertices are likely to be small, whereas most pairs of vertices are not adjacent in the graph. A notable interpretation of the small-world phenomena is when it's present in social networks, where individuals and acquaintances are represented by vertices and edges respectively: otherwise seemingly strangers are only a few acquaintances apart.

Watts and Strogatz defined characteristic path length and (local) clustering coefficients as follows.

Definition 2.1 (Characteristic path length)**.** The *characteristic path length* $L(G)$ of G denotes the average length of a shortest path between all pairs of distinct vertices in G. Note that if for a pair $u, v \in V$, there is no path connecting u and v, we set the length of a shortest path from u to v to be 0. Thus

$$L(G) = \frac{1}{|V|(|V|-1)} \cdot \sum_{\substack{u,v \in V \\ u \neq v}} d(u,v)$$

where

$$d(u,v) = \begin{cases} d_G(u,v) & \text{if } d_G(u,v) \in \mathbb{R} \\ 0 & \text{otherwise} \end{cases}$$

Definition 2.2 (Clustering coefficient)**.** Let $v \in V$. The *local clustering coefficient* C_v is the edge density of the subgraph induced on the neighbourhood of v, hence

$$\begin{aligned} C_v &= \frac{|\{\{a,b\} \mid a,b \in N_G(v) \text{ with } \{a,b\} \in E\}|}{\binom{d_G(v)}{2}} \\ &= \frac{2\,|\{\{a,b\} \mid a,b \in N_G(v) \text{ with } \{a,b\} \in E\}|}{d_G(v)(d_G(v)-1)} \end{aligned}$$

The *clustering coefficient* $C(G)$ *of* G is then the average of all local clustering coefficients:

$$C\,(G) = \frac{1}{|V|} \sum_{v \in V} C_v$$

We note that in their work, the graphs Watts and Strogatz observed did not test the boundaries of definition 2.2 and reveal the ambiguity around the local clustering coefficient of a leaf. For this reason, we only define the local clustering coefficient for vertices v with $d_G(v) \geq 2$ and amend the clustering coefficient of G to be the average local clustering coefficient over all such vertices.

Network theory also concerns itself with identifying community structures within graphs. A *community* is defined to be an induced subgraph of G which is more densely connected than the overall graph. This intuitively makes sense, and due to the loosely interpretable nature of this definition there are various methods for identifying communities.

One such method is the minimum-cut method. A *cut* F is a set of edges in G such that there exists a partition of $V =: \{V_1, V_2, \}$ with $F = E(V_1, V_2)$ [11]. In the minimum-cut method for finding communities, we generalise the idea of a cut F to be a set of edges in G such that there exists a partition of $V =: \{V_1, \ldots, V_k\}$ with $F = E(V_1, \ldots, V_k)$ and k is the hypothesised number of communities in G, and minimise $|F|$, the number of edges between these subsets. By minimising the edges between subsets of the partition, we ensure that the induced subgraphs $G[V_i]$ possess comparatively more edges, and thus are more densely connected. A downside of the minimum-cut method for finding communities is having to predetermine the number of communities k we seek, which isn't necessarily neither easy to hypothesise nor a bound we'd wish to impose.

Another method is the *Girvan-Newman* algorithm, which relies on the betweenness centrality of an edge, a centrality measure. A *centrality measure* of a graph is an ordering of its vertices or edges which reflects their importance according to some criteria.

Definition 2.3 (Vertex/Edge Betweenness Centrality)**.** The *betweenness centrality of a vertex* $v \in V$ is the fraction of all shortest paths in G between vertices $s, t \in V$ with $s \neq t \neq v$ of which v is an interior point.

Similarly, the *betweenness centrality of an edge* $e \in E$ is the fraction of all shortest paths in G between vertices $s, t \in V$ with $s \neq t$ that include e [16].

The Girvan-Newman algorithm computes all edge betweenness centralities, removes an edge with the largest centrality, and repeats this procedure until exhaustion, constructing a hierarchical partitioning of the vertices into communities [16][13]. The key insight around this procedure is that if the underlying graph does indeed present a community, the few edges that join these communities possess a high betweenness centrality since they'd be present in all inter-community shortest paths. Girvan and Newman later improved on their original algorithm and, when doing so, defined *modularity* Q, a metric for evaluating the quality of a partition $\mathcal{P}$ of V representing the communities in a graph [31].

The *Walktrap* algorithm by Pons and Latapy [34] makes use of the idea that random walks of small lengths starting at some vertex in a community are likely to end within the same community. The algorithm makes use of a custom inter-vertex distance measure, and makes use of an approximation using the Central Limit Theorem and other such improvements. The appeal of the Walktrap algorithm comes from its performance and availability in many standard graph theory software libraries. Both the Girvan-Newman and Walktrap algorithms however (and many more) are used in practise today given that they each perform better depending on the performance metric considered and the large breadth of such metrics.

2.2 Local separators

Before introducing local separators, we begin by defining a separator.

Definition 2.4 (Vertex Separator)**.** [11] Suppose G is a connected graph, and let $X = \{v_1, \ldots, v_k\} \subseteq V$ with $k \in \mathbb{N}$. X is a vertex *separator* of G if the subgraph of G induced on $V \setminus X$ is disconnected, i.e. if $G[V \setminus X]$ has 2 or more connected components.

As $|X| = k$, X is called a k-separator. If $k = 1$, then v_1 is a *cutvertex*.

From the definition of a separator, we deduce that it is a global structure in G. In [6], Carmesin introduces a local analogue to separators. We start with the definition of

a ball of radius r which introduces locality, and then present the definition of an r-local cutvertex.

Definition 2.5 (Ball of radius r)**.** Let $v \in V$. The *ball of radius* $r \in \mathbb{N} \cup \{\infty\}$ around v is the induced subgraph of G whose vertices are those which are at a distance of at most r from v, and without any edges joining vertices at a distance of precisely r. We denote the ball of radius r around v in G by $B_G(v, r)$.

If $r = s + \frac{1}{2}$, then $B_G(v, r)$ is the induced subgraph of G whose vertices are those of distance at most s from v.

To circumvent the special attention given to whether r is a half-integer or not, we present an alternative definition of the ball of radius r around a vertex and show its equivalence. We do this because this alternative definition of a ball of radius r is easier to implement in code.

Definition 2.6 (Ball of radius r)**.** Let $v \in V$. The *ball of radius* $r \in \mathbb{N} \cup \{\infty\}$ around v is formed by the vertices in G with distance at most r from v and all edges $\{x, y\} \in E$ satisfying

$$d_G(v, x) + d_G(v, y) + 1 \leq 2r$$

Lemma 2.1. For the same parameter r, the balls defined in definition 2.5 and definition 2.6 are equal.

Proof. Let $B(v, r)$ be defined according to definition 2.5 and let $B'(v, r)$ be defined according to definition 2.6.

From their definitions it follows that

$$V(B(v, r)) = V(B'(v, r))$$

We now prove that $E(B(v, r)) = E(B'(v, r))$. To this end, we will be considering the two cases

$$r = s \text{ and } r = s + \frac{1}{2}$$

where $s \in \mathbb{N}$.

For the edges between vertices which are at a distance $t < s$ from v, we observe that under definition 2.5 they are included in the parametric ball as they're part of the induced subgraph. Similarly, let $h < s$ and wlog assume $t \leq h$, and note that for $xy \in E$ with

$$d_G(v, x) = t \text{ and } d_G(v, y) = h$$

we have

$$\begin{aligned} d_G(v, x) + d_G(v, y) + 1 &= t + h + 1 \\ &\leq (s - 1) + (s - 1) + 1 \\ &= 2s - 1 \leq 2s \end{aligned}$$

and so $xy \in E(B'(v, r))$ as well.

Now we observe what happens to the edges joining vertices which are at a distance s from v.

Let $a, b \in V(B(v, r))$, $a \neq b$, with $ab \in E$ and

$$d_G(v, a) = d_G(v, b) = s$$

Case $r = s$: From its definition, $E(B(v,r))$ does not include the edges in G that join vertices of distance precisely r from v. Hence $ab \notin E(B(v,r))$ by definition. Note then that

$$\begin{aligned} d_G(v,a) + d_G(v,b) + 1 &= s + s + 1 \\ &= 2s + 1 \\ &= 2r + 1 > 2r \end{aligned}$$

and so $ab \notin E(B'(v,r))$ as well.

Case $r = s + \frac{1}{2}$: From its definition, we find that $ab \in E(B(v,r))$. Note then that

$$\begin{aligned} d_G(v,a) + d_G(v,b) + 1 &= s + s + 1 \\ &= 2\left(s + \frac{1}{2}\right) \\ &= 2r \leq 2r \end{aligned}$$

and so $ab \in E(B'(v,r))$ as well.

We conclude that $E(B(v,r)) = E(B'(v,r))$ and so that

$$B(v,r) = B'(v,r)$$

□

We subsequently define a punctured ball.

Definition 2.7 (Punctured ball of radius r)**.** Let $v \in V$. The *punctured ball of radius* $r \in \mathbb{N} \cup \{\infty\}$ around v is $B(v,r) - v$.

Finally we define an r-local cutvertex.

Definition 2.8 (r-local cutvertex)**.** Let $v \in V$, and let $r \in \mathbb{N} \cup \{\infty\}$. v is an *r-local cutvertex* if the punctured ball of radius $\frac{r}{2}$ around v is disconnected.

Having having defined an r-local cutvertex, our ultimate goal is to decompose a graph along its local cutvertices. An immediate step in this direction is to define what it means to split a graph at one of its r-local cutvertices.

Definition 2.9 (Splitting at a local cutvertex)**.** Let $r \in \mathbb{N} \cup \{\infty\}$ and let $v \in V$ be an r-local cutvertex. To *split at an r-local cutvertex v* equates to carrying out the following operation on the graph G.

Consider the induced subgraph $H := B\left(v, \frac{r}{2}\right) - v$ in G and let

$$C_1, \ldots, C_k$$

be the components of H. For $i \in [k]$, construct the *split vertex* v_i, the edge $\{v, v_i\}$, and the edges

$$\{\{v_i, c_i\} \mid c_i \in V(C_i) \text{ and } \{v, c_i\} \in E\}$$

and remove the edges

$$\{\{v, c_i\} \mid c_i \in V(C_i) \text{ and } \{v, c_i\} \in E\}$$

In words, splitting at an r-local cutvertex v consists of first associating copies of v, called *split vertices*, to each component of the punctured ball of radius $\frac{r}{2}$ around v, with edges between a split vertex and a vertex c_i in a component C_i if there exists an edge between v and c_i in G, along with the edge $\{v, v_i\}$, and second removing the edges between v and c_i.

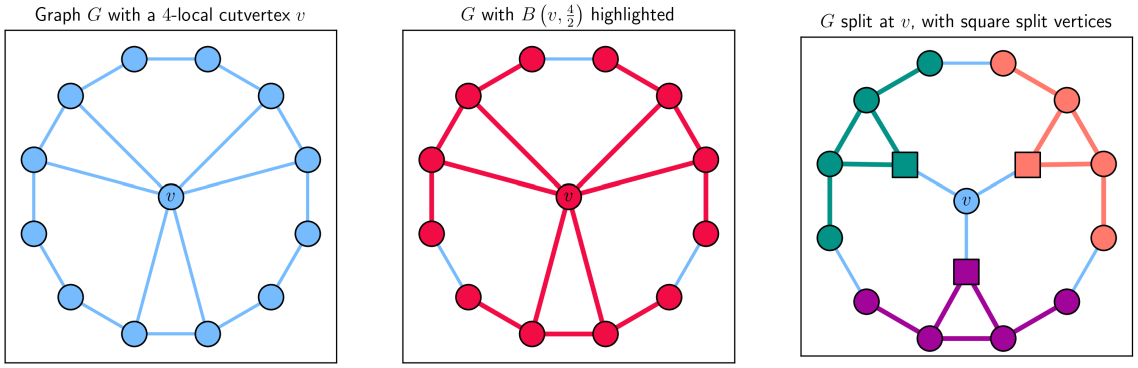

Figure 1: Splitting at a local cutvertex illustrated

The illustration in fig. 1 is exemplary of the possible usecases for local separators: indeed the graph G does not possess any separators, and so the critical nature of v in regards to its vicinity is otherwise gone unnoticed.

When implementing these definitions in algorithmic code, two natural questions arise around the definition of a local cutvertex. Is the radius of a local cutvertex unique? How do global cutvertices relate to local cutvertices? Below we present two lemmas answering these questions.

Lemma 2.2. Let $r \in \mathbb{N}$, and let $v \in V$ be an r-local cutvertex in G. Then v is also an $(r-1)$-local cutvertex in G.

Proof. In order to show that v is an $(r-1)$-local cutvertex in G we need to show that decreasing the locality doesn't introduce vertices or edges in $B\left(v, \frac{r-1}{2}\right)$ that aren't present in $B\left(v, \frac{r}{2}\right)$. Therefore we wish to show that

$$V\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq V\left(B\left(v, \frac{r}{2}\right)\right)$$
$$E\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq E\left(B\left(v, \frac{r}{2}\right)\right)$$

Note first that the distances between distinct vertices in an unweighted and undirected graph are in $\mathbb{N}$. Secondly, decreasing the locality of a local cutvertex by 1 modifies the radius of its parametric ball by $\frac{1}{2}$, a fraction.

Let $k \in \mathbb{N}$. We consider the two cases:

Case $r = 2k$: Note that $B\left(v, \frac{r}{2}\right) = B(v, k)$ contains the vertices that are at a distance k from v in G, if they exist, whereas $B\left(v, \frac{r-1}{2}\right) = B\left(v, k - \frac{1}{2}\right)$ cannot contain these same vertices. Therefore we have

$$V\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq V\left(B\left(v, \frac{r}{2}\right)\right)$$

This in turn implies that

$$E\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq E\left(B\left(v, \frac{r}{2}\right)\right)$$

as required.

Case $r = 2k + 1$: Note that $B\left(v, \frac{r}{2}\right) = B\left(v, k + \frac{1}{2}\right)$ contains the vertices that are at a distance k from v in G, if they exist, and $B\left(v, \frac{r-1}{2}\right) = B(v, k)$ does as well. Hence

$$V\left(B\left(v, \frac{r-1}{2}\right)\right) = V\left(B\left(v, \frac{r}{2}\right)\right) \implies V\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq V\left(B\left(v, \frac{r}{2}\right)\right)$$

We have $B\left(v, \frac{r}{2}\right) = B\left(v, k + \frac{1}{2}\right)$, which contains the edges between vertices at a distance k from v in G, if they exist. Now consider

$$B\left(v, \frac{r-1}{2}\right) = B(v, k)$$

which cannot possess these same edges, if they do in fact exist. Hence

$$E\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq E\left(B\left(v, \frac{r}{2}\right)\right)$$

We've shown that

$$V\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq V\left(B\left(v, \frac{r}{2}\right)\right)$$
$$E\left(B\left(v, \frac{r-1}{2}\right)\right) \subseteq E\left(B\left(v, \frac{r}{2}\right)\right)$$

therefore, if v is an r-local cutvertex then v is also a $(r-1)$-local cutvertex. □

Lemma 2.3. Let $v \in V$ be a cutvertex in G, and let C denote its component. Then v is a $|V(C)|$-local cutvertex.

Proof. Let P be a longest path in C. Then

$$\begin{aligned} &|V(P)| \leq |V(C)| \\ \implies &|P| \leq |V(C)| - 1 \end{aligned}$$

Let $x, y \in V(C)$ with $e := \{x, y\} \in E$. Notice that $x, y \in V\left(B\left(v, \frac{|V(C)|}{2}\right)\right)$ since

$$d_G(v, x), d_G(v, y) \leq |V(C)| - 1 \leq |V(C)|$$

Moreover

$$\begin{aligned} d_G(v,x) + d_G(v,y) + 1 &= d_C(v,x) + d_C(v,y) + 1 \\ &\leq |V(C)| - 1 + |V(C)| - 1 + 1 \\ &= 2|V(C)| - 1 \\ &\leq 2|V(C)| \end{aligned}$$

Therefore $e \in B\left(v, \frac{|V(C)|}{2}\right)$, and since e was arbitrary, we find that $E\left(B\left(v, \frac{|V(C)|}{2}\right)\right) = E(C)$.

We conclude that $B\left(v, \frac{|V(C)|}{2}\right) = C$. Further, since $C - v$ is disconnected by definition, we also have that $B\left(v, \frac{|V(C)|}{2}\right) - v$ is disconnected.

Thus v is a $|V(C)|$-local cutvertex. □

Now, let $u \in V$ be a cutvertex. Then by lemma 2.3, u is also a $|V(C)|$-local cutvertex. Further, we find that u is also a $(|V(C)| + 1)$-local cutvertex, since increasing the radius under consideration doesn't add any vertices or edges not already included in $B\left(v, \frac{|V(C)|}{2}\right)$. Formally, we present the following lemma.

Lemma 2.4. Let $u \in V$ be a cutvertex in G and let C be its component. Then

$$B\left(u, \frac{r}{2}\right) - u \text{ is disconnected } \forall r \in \mathbb{N} \cup \{\infty\} \setminus \{1\}$$

Proof. Let $c = |V(C)|$. By lemma 2.3, u is a c-local cutvertex.

By lemma 2.2 and induction on r, u is also an r-local cutvertex for $r \in \{2, \ldots, c-1\}$.

We claim that for $k \in \mathbb{N}$, $B\left(u, \frac{c}{2}\right) = B\left(u, \frac{c+k}{2}\right)$. Note that c is maximal as there are no vertices at more than a distance c apart in C. Therefore $V\left(B\left(u, \frac{c}{2}\right)\right) = V\left(B\left(u, \frac{c+k}{2}\right)\right)$.

Let $x, y \in C$, with $e := \{x, y\} \in E$. Notice then that

$$d_G(v,x) + d_G(v,y) + 1 \leq 2c - 1 \leq 2c \leq 2\left(c + \frac{k}{2}\right)$$

therefore $e \in E\left(B\left(u, \frac{c}{2}\right)\right)$ and $e \in E\left(B\left(u, \frac{c+k}{2}\right)\right)$. Since e was arbitrary, we conclude that $E\left(B\left(u, \frac{c}{2}\right)\right) = E\left(B\left(u, \frac{c+1}{2}\right)\right)$. Hence $B\left(u, \frac{c}{2}\right) = B\left(u, \frac{c+k}{2}\right)$, and so u is also a $(c+k)$-local cutvertex. Arguing this fact by induction, we find that u is an ∞-local cutvertex as well. □

We note that we exclude 1 from the range of possible radii in the statement of lemma 2.4 because of the connectivity of the null graph.

3 Code

3.1 Project Structure

This project uses the Python programming language and uses version 3.7. We chose the Python programming language due to its approachability and readibility, and the version used allows for type annotations throughout our project to further aid readibility and debugging.

We use the `networkx` [19] graph analysis library. The justification for this is the how the library lends itself well to fast prototyping, its simple graph visualisation suite, and our familiarity with this library.

This project has been structured as a Python package and made available on GitHub at `https://www.github.com/iyassou/local-separators`, although we will be presenting equivalent and sometimes identical code snippets in this paper. The justification for this is to facilitate sharing the project's source code, as well as the structure it offers us in the form of: source code that computes graph theoretical structures, source code that tests its correctness, and source code that visualises graphs.

3.2 Basic Building Blocks

As a goal of this project is to identify local cutvertices in real-world datasets and visualise them, we first need to establish the necessary code for finding the more basic structures in a graph.

`ball` of radius r around v

We begin with the code that identifies a ball of radius r around a vertex v in G.

```python
import networkx as nx

def ball(G: nx.Graph, v: Vertex, r: int|float) -> nx.Graph:
    fractional, _ = modf(r)
    if fractional and fractional != 0.5:
        raise ValueError()
    d: Dict[Vertex, int] = nx.single_source_shortest_path_length(
        G, v, cutoff=r
    )
    def filt_vertex(x: Vertex) -> bool:
        return x in d
    def filt_edge(x: Vertex, y: Vertex) -> bool:
        if not G.has_edge(x, y):
            return False
        try:
            return d[x] + d[y] + 1 <= 2 * r
        except KeyError:
            return False
    return nx.subgraph_view(
        G, filter_node=filt_vertex, filter_edge=filt_edge
    )
```

We begin by verifying that the radius value is indeed an integer or half-integer. We do this by using `modf`, which returns the fractional and integral parts of a real number respectively, and verifying that the fractional part is indeed either 0 or 0.5.

Having shown the equivalency of the alternative definition of a ball in lemma 2.1, we subsequently obtain all the shortest paths of length at most r from v. This is a mapping from the other endpoint of a calculated shortest path to its length. The underlying source code for finding these shortest paths from v employs a breadth-first search beginning at v and proceeding until the observed depth exceeds the specified cutoff value, which in our case is the radius r [1]. Using the aforementioned shortest paths, we obtain the following induced subgraph on G by filtering which vertices and edges we wish to keep.

The vertices we keep, according to definition 2.6, are those which are at a distance at most r from v, which we can verify by checking for their presence in the previously obtained shortest paths, by definition.

When it comes to the edges, note that the filtering function accepts the two endpoints $x, y \in V$ of a candidate edge. To ensure that the resulting subgraph is induced, we begin by rejecting all candidate edges that do not exist in the original graph G. Further, since the vertex and edge filtering are applied separately, we once more ensure that both x and y are at most r away from v. This is done by assuming their presence in the mapping of shortest paths lengths at most r from v, which if false results in a Python `KeyError`, in which case we do not keep the edge. Finally, we keep the candidate edge if it meets the criterion specified in definition 2.6.

Determining if v is an r-local cutvertex

Now that we've defined the `ball` function, we can determine whether a given vertex v is an r-local cutvertex in G by observing the connectivity of the punctured ball of radius $\frac{r}{2}$ around v. We name this function `is_local_cutvertex`.

```
def is_local_cutvertex(G: nx.Graph, v: Vertex, r: int) -> bool:
    punctured_ball: nx.Graph = nx.subgraph_view(
        ball(G, v, r/2), filter_node=lambda x: x != v
    )
    return not nx.is_connected(punctured_ball)
```

By filtering out the vertex v from $B\left(v, \frac{r}{2}\right)$ we obtain its punctured ball, whose connectivity we subsequently check.

`nx.is_connected` checks if a graph G is connected as follows [18]:

1) Pick an arbitrary vertex $w \in V$
2) Traverse G via a breadth-first search until exhaustion, counting the number c of vertices visited.
3) G is connected if $c = |V|$ and disconnected otherwise.

3.3 Representing a Local Cutvertex

In its simplest form, we can represent a local cutvertex as a tuple (v, r), with $v \in V$ the vertex and $r \in \mathbb{N} \cup \{\infty\}$ its associated radius. This simple representation is sufficient for visualising local cutvertices, however it is lacking when it comes to splitting at multiple

local cutvertices and visualising their decomposition of the original graph. To highlight the issue, consider the following graph G, and local cutvertices u, v with radii $r_u, r_v > 2$ respectively, which is representative of a scenario encountered in practise.

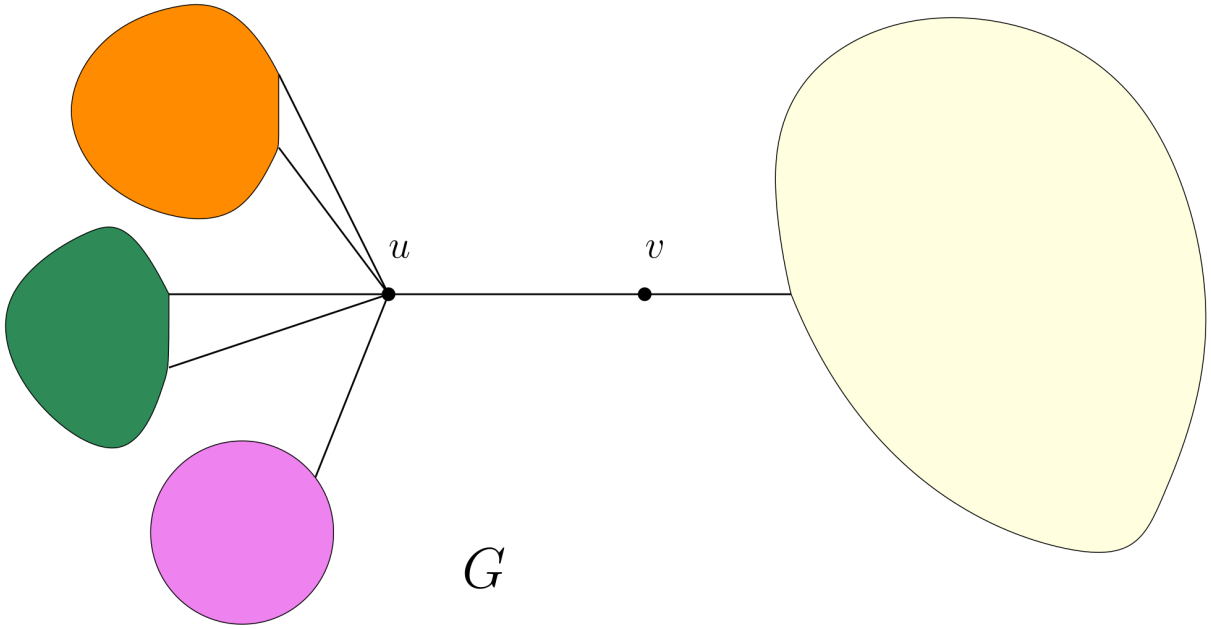

Figure 2: Graph G with adjacent local cutvertices u and v

When splitting at G's local cutvertices, we do so sequentially, either first splitting at u then v or otherwise. Without loss of generality let's first split at u then v. In code, this is done by obtaining the components of $B_G\left(u, \frac{r_u}{2}\right) - u$ and assigning a numbered copy of u to each resulting component.

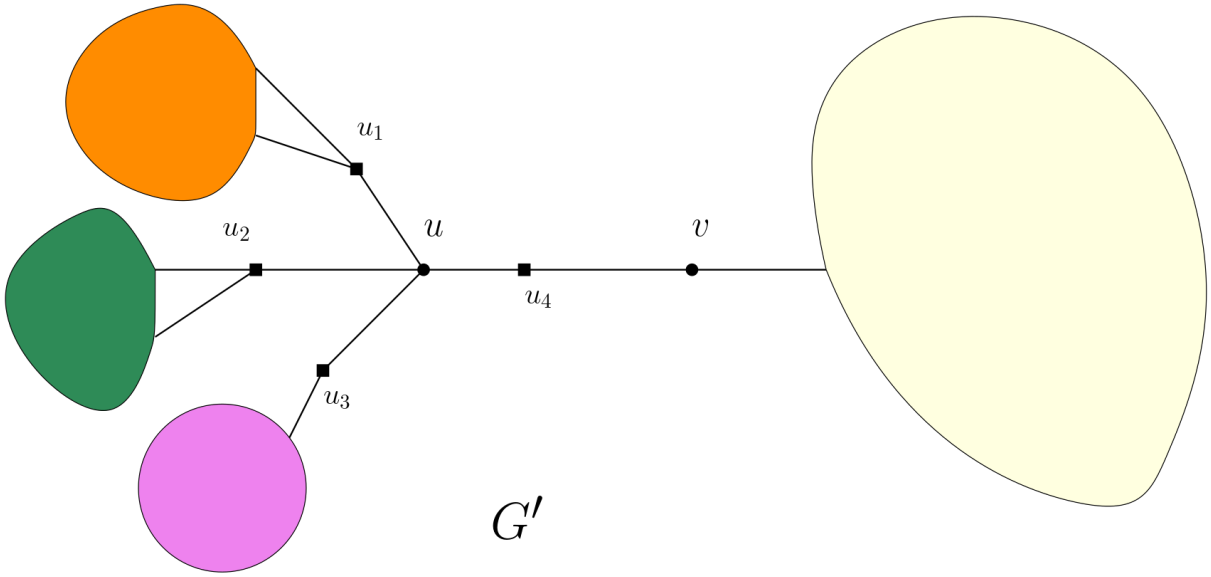

Figure 3: G', the graph obtained after splitting at u in G

Let G' be the graph obtained after splitting at u. G' possesses vertices u_1, u_2, u_3, u_4, which were not present in G. Splitting at v now however requires the components of $B_{G'}\left(v, \frac{r_v}{2}\right) - v$, which aren't identical to those of $B_G\left(v, \frac{r_v}{2}\right) - v$ due to the introduction

of the split vertices and in particular the subdivision of the edge $\{u, v\}$ into edges $\{u, u_4\}$ and $\{u_4, v\}$.

Given that this issue arises since we're trying to split v at the original graph G and not the resulting graph, the information we'd like to have access to is how the punctured ball around v partitions the edges incident to v in the original graph G. Our solution would then consist of first checking for the absence of an edge otherwise previously present before splitting at some other local cutvertex in the current graph. In the example above, the absent edge is the edge $\{u, v\}$ between local cutvertices. Since we know one endpoint in this edge partition will always be the local cutvertex itself, we decide to only keep the other endpoint.

Let $\mathcal{C}_u$ be the components of $B_G\left(u, \frac{r_u}{2}\right) - u$. Consider then the following representation of a local cutvertex u:

$$u \sim (u, r_u, \{N_G(u) \cap C \mid C \in \mathcal{C}_u\})$$

We'll use the following constructor for local cutvertices.

```python
from typing import NamedTuple, Tuple, Set

class LocalCutvertex(NamedTuple):
    vertex: Vertex
    locality: int
    edge_partition: Set[Tuple[Vertex, ...]]

    @classmethod
    def new(cls, G: nx.Graph, v: Vertex, r: int) -> LocalCutvertex:
        punctured_ball: nx.Graph = nx.subgraph_view(
            ball(G, v, r/2), filter_node=lambda x: x != v
        )
        punctured_ball_components = nx.connected_components(
            punctured_ball
        )
        neighbourhood: Set[Vertex] = set(G.neighbors(v))
        edge_partition: Set[Tuple[Vertex, ...]] = set(
            tuple(neighbourhood.intersection(comp))
            for comp in punctured_ball_components
        )
        return LocalCutvertex(
            vertex=v, locality=r, edge_partition=edge_partition
        )
```

3.4 Splitting at Multiple Local Cutvertices

Given the above representation of a local cutvertex, we now define how we'll be splitting at multiple local cutvertices.

Definition 3.1 (Splitting at multiple local cutvertices)**.** Let $u_1, \ldots, u_k$ be local cutvertices in G with $r_1, \ldots, r_k$ their respective radii, and define

$$\mathcal{U}_j = \{u_1, \ldots, u_j\}$$

Let $G^{(1)}$ be the graph obtained after splitting at u_1 in the usual way, i.e. as described in definition 2.9. $G^{(i)}$ is the graph that results from the following operation on $G^{(i-1)}$.

Let $\mathcal{C}_i$ denote the components of $B_G\left(u_i, \frac{r_i}{2}\right) - u_i$. Given the following representation of u_i

$$u_i \sim (u_i, r_i, \mathcal{N}_i)$$

where $\mathcal{N}_i$ is the aforementioned edge partition given by

$$\mathcal{N}_i = \{N_G(u_i) \cap C \mid C \in \mathcal{C}_i\}$$

we modify the edge partition to exclude all previously split at local cutvertices $\mathcal{U}_{i-1}$ from its subsets, i.e. we modify $\mathcal{N}_i$ to obtain $\mathcal{N}_i'$

$$\mathcal{N}_i' := \{N \setminus \mathcal{U}_{i-1} \mid N \in \mathcal{N}_i\}$$

Then, for each subset $N_{i,j} \in \mathcal{N}_i'$, we construct a split vertex $u_{i,j}$, the edge $\{u_i, u_{i,j}\}$, the edges

$$\{\{u_{i,j}, n_j\} \mid n_j \in N_{i,j}\}$$

and remove the edges

$$\{\{u_i, n_j\} \mid n_j \in N_{i,j}\}$$

Repeating this procedure for each local cutvertex in order yields $G^{(k)}$ i.e. the graph G split at its local cutvertices $\mathcal{U}$.

Proceeding as instructed by definition 3.1, we'd would split G' at v to obtain G'' pictured in fig. 4.

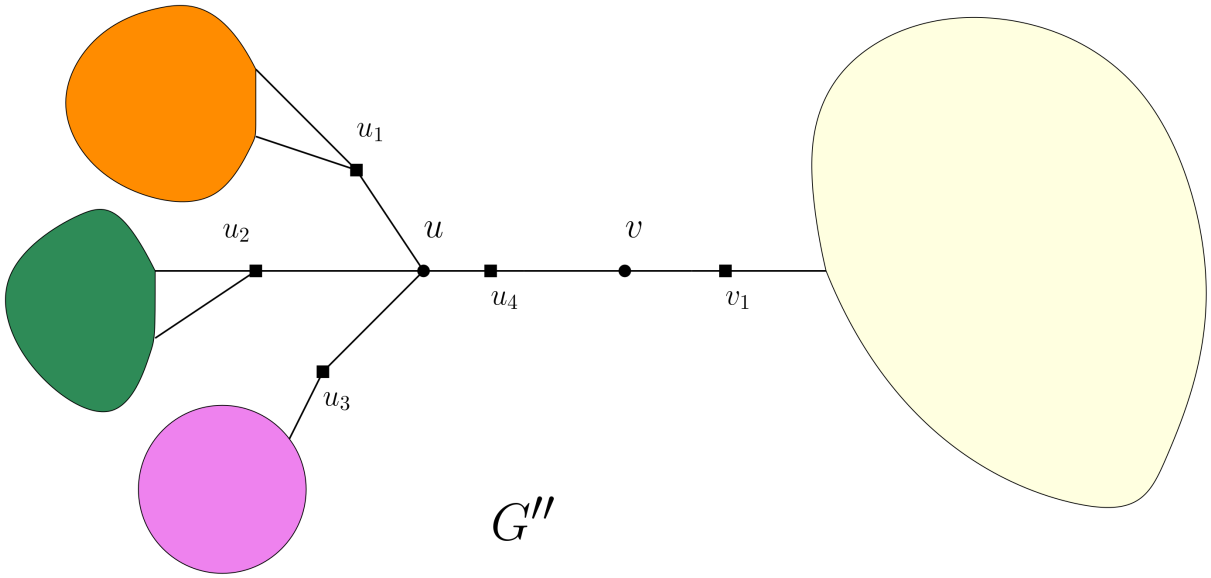

Figure 4: G'', the graph obtained after splitting at u and then v in G

We've detailed below our implementation of definition 3.1.

```
def split(G: nx.Graph, lcvs: List[LocalCutvertex]) -> nx.Graph:
    G_prime: nx.Graph = G.copy()
    lcv_vertices: Set[Vertex] = set()
    for lcv in lcvs:
        i: int = 1
        for subset in lcv.edge_partition:
            subset: Tuple[Vertex, ...] = tuple(
                vertex for vertex in subset
                if vertex not in lcv_vertices
            )
            if not subset:
                continue
            split_vertex: Vertex = create_split_vertex(lcv.vertex, i)
            G_prime.add_edge(split_vertex, lcv.vertex)
            G_prime.add_edges_from(
                (split_vertex, neighbour) for neighbour in subset
            )
            G_prime.remove_edges_from(
                (lcv.vertex, neighbour) for neighbour in subset
            )
            i = i + 1
            lcv_vertices.add(lcv.vertex)
    return G_prime
```

where `create_split_vertex` generates the name for a split vertex v_i given its local cutvertex v and some unique identifier i. In this project the naming convention we follow simply subscripts the local cutvertex with the index $i \in \mathbb{N}$, e.g. $a_1, a_2, \ldots$ are split vertices of the local cutvertex a.

3.5 Finding Local Cutvertices

Armed with the basic routines explicited above, we now seek an algorithm for finding local cutvertices.

From lemma 2.2, we deduce that there is a tight value for the radius of a local cutvertex v, namely, the largest $r \in \mathbb{N}$ such that v a r-local but not $(r+1)$-local. Hence when trying to determine if a vertex is a local cutvertex, we can iterate through all possible radii values from largest to smallest until we either exhaust all possible values or find that the vertex is indeed a local cutvertex.

It then remains to choose the range of possible radii values. When it comes to choosing the lower bound, observe first that for $v \in V$ with $d_G(v) > 0$

$$B\left(v, \frac{2}{2}\right) = B(v, 1) = K_{1, d_G(v)}$$

Hence $B\left(v, \frac{2}{2}\right) - v$ is trivially always disconnected. Therefore, we set our lower bound to be 3.

As for the upper bound, since our algorithm would iterate through the range of possible values in decreasing order, trying to minimise the radius for which the vertex under

consideration is a local cutvertex, we can pick a sufficiently large enough arbitrary value. We note that this value should be large enough to cover all vertices in the parametric ball's reach. From lemma 2.3, we note that $|V(C)|$ is the appropriate value for the upper bound.

As this project is focused on finding local cutvertices, which have a tight value for their radii, we make the decision to exclude cutvertices, whose corresponding local cutvertices can have arbitrary radii by lemma 2.4, from our search for local cutvertices, despite their correct qualification as local cutvertices.

Taking these decisions into account, we now present an algorithm for finding local cutvertices in G.

Algorithm 1 An algorithm for finding local cutvertices in a graph

```
local_cutvertices = []
for v ∈ V do
    C ← component of v
    min_locality ← 3
    max_locality ← |V(C)|
    if min_locality ≥ max_locality then
        continue
    end if
    if v separates C then
        continue
    end if
    locality = max_locality
    while locality ≥ min_locality do
        if is_local_cutvertex(C, v, locality) then
            local_cutvertices.append(LocalCutvertex.new(C, v, locality))
            break
        end if
        locality = locality − 1
    end while
end for
return local_cutvertices
```

Let C_{max} denote the largest component of G. Note then that checking if v separates its component C, as performed by `networkx`, requires a breadth-first search which has worst case time complexity $\mathcal{O}\left(|V(C_{max})| + |E(C_{max})|\right)$. Secondly, iterating over the range of possible radii in the worst case is $|V(C_{max})|$. Since obtaining the punctured ball about a vertex and checking its connectivity each require distinct breadth-first searches, they each have worst case time complexity $\mathcal{O}(|V(C_{max})| + |E(C_{max})|)$ as well.

Therefore, assuming that all other operations in algorithm 1 have elementary time cost, the worst case time complexity of algorithm 1 is

$$
\begin{aligned}
&|V| \cdot \left[\mathcal{O}\left(|V(C_{max})| + |E(C_{max})|\right) + |V(C_{max})| \cdot \mathcal{O}\left(\left(|V(C_{max})| + |E(C_{max})|\right)^2\right) \right] \\
=&|V| \cdot \left[\mathcal{O}\left(|V| + |E|\right) + |V| \cdot \mathcal{O}\left(\left(|V| + |E|\right)^2\right) \right] \\
=&\mathcal{O}\left(|V|^2 + |V| \cdot |E| + |V|^4 + |V|^2 \cdot |E|^2\right) \\
=&\mathcal{O}\left(|V|^4 + |V|^2 \cdot |E|^2\right)
\end{aligned}
$$

We can use lemma 2.2 to make an improvement to algorithm 1. We note that a non-global, local cutvertex u possesses a tight value for its radius, i.e. $r \in \mathbb{N} \cup \{\infty\}$ such that u is an r-local cutvertex but not an $(r+1)$-local cutvertex. Since the range of possible radius values form an ascending sequence of numbers, we can employ a binary search to find the tight value of r instead of sequentially iterating through all values in the range.

Algorithm 2 An improved algorithm for finding local cutvertices in a graph

```
local_cutvertices = []
for v ∈ V do
    C ← component of v
    min ← 3
    max ← |V(C)|
    if min ≥ max then
        continue
    end if
    if v separates C then
        continue
    end if
    v_is_a_local_cutvertex: bool = None
    while True do
        # Update the locality
        if max − min = 1 and v_is_a_local_cutvertex is not None then
            if not v_is_a_local_cutvertex then
                v_is_a_local_cutvertex ← is_local_cutvertex(C, v, min)
                locality ← min
                break
            else
                v_is_a_local_cutvertex ← is_local_cutvertex(C, v, max)
                if v_is_a_local_cutvertex then
                    locality ← max
                else
                    v_is_a_local_cutvertex ← True
                end if
                break
            end if
        else
            locality ← min + ⌊(max−min)/2⌋
        end if
        # Check if v is a local cutvertex
        if v_is_local_cutvertex then
            local_cutvertices.append(LocalCutvertex.new(C, v, locality))
        end if
    end while
end for
return local_cutvertices
```

The difference between algorithm 2 and algorithm 1 lies in how the value of the current radius or locality is computed. Let us first walk through the typical update rule which occurs when the search space is large i.e. the difference between the upper and lower bounds is strictly greater than 1. When in this scenario, we update the locality under consideration according to the typical binary search update rule

$$\text{locality} = \min + \left\lfloor \frac{\max - \min}{2} \right\rfloor$$

Consider now the special case where the difference between the upper and lower bounds is 1. We need to give this case special attention because under the previous update rule we'd enter an infinite loop, as

$$\begin{aligned}\text{locality} &= \min + \left\lfloor \frac{\max - \min}{2} \right\rfloor \\ &= \min + \left\lfloor \frac{1}{2} \right\rfloor \\ &= \min + 0 = \min\end{aligned}$$

resulting in the local cutvertex test executing indefinitely for the same locality. In fact, the only scenario in which we'd like to proceed with executing the local cutvertex test for at least run despite this is if the difference between the upper and lower bounds is 1 *and* we are at the first iteration of the algorithm.

Now note that we can determine if we are at the first iteration of the algorithm by observing the value of `v_is_a_local_cutvertex`. This is because `v_is_a_local_cutvertex` is initialised to `None`, therefore `v_is_a_local_cutvertex is None` indicates that we're at the first iteration of the binary search. Since a check is performed for the negation of this condition to decide if we're branching into the special update rule, we correctly execute the typical update rule instead.

Consider now how our algorithm the special case where the difference between the upper and lower bounds is 1 and we are *not* at the first iteration of the binary search. Let $\min_{\text{curr}}, \max_{\text{curr}}$ denote the lower and upper bounds at the current iteration of algorithm 2, and similarly $\min_{\text{prev}}, \max_{\text{prev}}$ denote the lower and upper bounds and $\text{locality}_{\text{prev}}$ denote the locality tested at the previous iteration of algorithm 2. Now note that:

$$\begin{aligned}&\max_{\text{curr}} - \min_{\text{curr}} = 1 \\ \implies &\frac{\max_{\text{prev}} - \min_{\text{prev}}}{2} = 1 \\ \implies &\begin{cases} \max_{\text{curr}} & = \text{locality}_{\text{prev}} \\ & \text{OR} \\ \min_{\text{curr}} & = \text{locality}_{\text{prev}} \end{cases}\end{aligned}$$

We therefore know that one of either $\max_{\text{curr}}$ or $\min_{\text{curr}}$ was tried at the previous iteration. In order to determine which one, we make use of the fact that at this stage in algorithm 2, `v_is_a_local_cutvertex` states whether v was a local cutvertex at the previous iteration of the algorithm.

If v was indeed a local cutvertex at the previous iteration, then the previously tried value of `locality` was $\min_{\text{curr}}$, in which case we check if v is a $\max_{\text{curr}}$-local cutvertex. If it is, we set the locality to $\max_{\text{curr}}$ since $\max_{\text{curr}} > \min_{\text{curr}}$ and we're seeking the tight value. If it isn't, we leave the locality unchanged i.e. set to $\min_{\text{curr}}$ and also set `v_is_a_local_cutvertex = True`, as we've already determined this to be the case at the previous iteration.

If v wasn't a local cutvertex at the previous iteration, then the previously tried value of `locality` was $\max_{\text{curr}}$, in which case we check if v is a $\min_{\text{curr}}$-local cutvertex and update the state accordingly.

Iterating over the range of possible radii in the worst case now has time complexity $\log|V(C_{max})|$, where C_{max} is the largest component of G. Once more assuming that all other operations in algorithm 2 have elementary cost, we find that the worst case time complexity of algorithm 2 is

$$\begin{aligned}&|V| \cdot \left[\mathcal{O}\left(|V(C_{max})| + |E(C_{max})|\right) + \log|V(C_{max})| \cdot \mathcal{O}\left(\left(|V(C_{max})| + |E(C_{max})|\right)^2\right)\right] \\ =&\mathcal{O}\left(|V|^3 \log|V| + |V|^2 \cdot |E|^2\right)\end{aligned}$$

a better result.

3.6 Unit Testing

Unit testing refers to the testing and assertion of the correct functionality of the fundamental components of a software project. This has the benefit of both verifying program correctness and creating small software routines which can later be combined into larger dependable routines.

We tested the correct functionality of the functions described in the previous section by running them on small examples whose correct output, a graph, can be determined by hand, and then verifying that the actually outputted graph is isomorphic to the expected result.

`ball` of radius r around v

The correct behaviour of the `ball` function was asserted by testing it on tractable graphs for which the correct, expected answer can be entered and subsequently tested for isomorphism to the actual output. The NetworkX library function for determining if there exists an isomorphism between graphs is `nx.is_isomorphic`, which is an implementation of the vf2 algorithm by Cordella et al [8].

During unit testing we refer to the graph H which consists of W^{12} with every other pair of spokes removed, as well as H', which is $B_H\left(1, \frac{3}{2}\right)$.

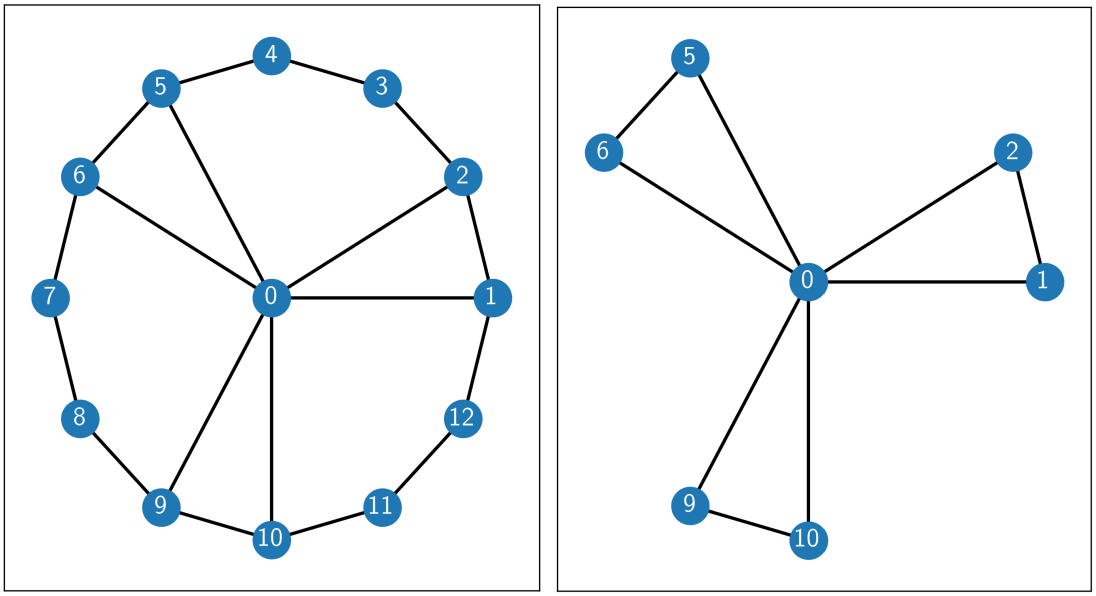

Figure 5: Graphs H and H'

We detail the testcases used for `ball` in the tables below.

Graph	Vertex	Radius	Expected
P^4	3	1	P^2
P^4	3	4	P^4
P^4	3	5	P^4
P^4	3	1.5	P^2
P^4	3	2	P^4
P^4	1	2	P^2
P^4	3	2.5	P^4
P^4	1	2.5	P^2
P^4	3	1.5	P^2

Graph	Vertex	Radius	Expected
C^5	3	1	P^2
C^5	1	2	P^4
C^5	3	5	C^5
C^5	3	6	C^5
C^5	3	1.5	P^2
C^5	3	2.5	C^5
C^6	1	3	C^6
C^5	1	3.5	C^5

Graph	Vertex	Radius	Expected
$([5], \emptyset)$	5	3	$(\{5\}, \emptyset)$
$([5], E(K^4))$	5	2.5	$(\{5\}, \emptyset)$
$([5], E(K^4))$	1	1	$K_{1,3}$

Graph	Vertex	Radius	Expected
W^4	0	1	$K_{1,4}$
W^4	0	1.5	W^4
W^9	0	2	W^9

Graph	Vertex	Radius	Expected
H	0	1.5	H'
H	0	1	$K_{1,6}$

`is_local_cutvertex`

The correct behaviour of `is_local_cutvertex` was verified in a similar manner to that of `ball`, using the test cases detailed below.

Graph	Vertex	Radius	Expected
P^{100}	51	20	**`True`**
P^8	6	2	**`True`**
P^{100}	51	∞	**`True`**

Graph	Vertex	Radius	Expected
C^5	1	2	**`True`**
C^5	1	4	**`True`**
C^5	1	10	**`False`**
C^5	1	12	**`False`**
C^5	1	∞	**`False`**
C^5	1	3	**`True`**
C^5	1	5	**`False`**
C^5	1	6	**`False`**
C^5	1	7	**`False`**

When testing `is_local_cutvertex` on K^k, note that $v \in K^k$ is an r-local cutvertex if and only if $r = 2$. To verify this in practise, 50 random values of $k \in \{30, 31, \ldots, 100\}$ were selected, and `is_local_cutvertex` was ran with $r = 2$, for which it returned **`True`**. It was similarly checked that for $r > 2$ `is_local_cutvertex` returned **`False`**.

Finding local cutvertices

The correct behaviour of `find_local_cutvertices`, which is an implementation of algorithm 2, was verified by running it on a set of parametrised tests on path, cycle, and complete graphs.

For P^k, 40 random values of $k \in \{3, \ldots, 100\}$ were sampled, for which we expect algorithm 2 to find no local cutvertices, given that algorithm 2 disregards separators.

Similarly for C^k, 40 random values of $k \in \{4, 5, \ldots, 100\}$ were sampled, for which we expect and found all vertices to be $(k-1)$-local cutvertices.

Finally for K^k, 40 random values of $k \in \{5, 6, \ldots, 50\}$ were sampled, for which we expect algorithm 2 to find no local cutvertices, given that the minimum considered locality is 3, which was indeed the case.

4 Dataset Analysis

We used the previously exposited code to analyse datasets of various sources modelling real-world phenomena. Due to speculated privacy and data regulation concerns, a considerable amount of the encountered datasets have been anonymised and so the results obtained for them are structural in nature, with speculatory room for interpretation. In contrast, datasets where metadata is available lend themselves well to interpretation and have been so analysed.

4.1 Stack Overflow Developer Story Tags

Stack Overflow (SO) is a question and answer website for professional and amateur programmers. In 2016, SO introduced Developer Stories, a feature which allows its employment-seeking users to construct an online CV that includes their SO contributions to the numerous available topic tags. These tags include programming languages, fields of computer science, operating systems, productivity methodologies for programming, software stacks, and more.

In [43], SO analysed the correlation between tags using the data mined from their users' Developer Stories in order to gain an understanding of the relationships between technology stacks and ecosystems. The dataset consists of an undirected, weighted graph. The vertices are a subset of the tags mined from the Developer Stories, with each having an associated size attribute that's proportional to its popularity. The subset of tags that were kept are those that are used by at least 0.5% of the userbase and possess a correlation coefficient with any other such tag of 0.1 or higher [43]. The weight of an edge between vertices is the correlation coefficient between tags, as identified by detecting communities in the dataset using the Walktrap algorithm.

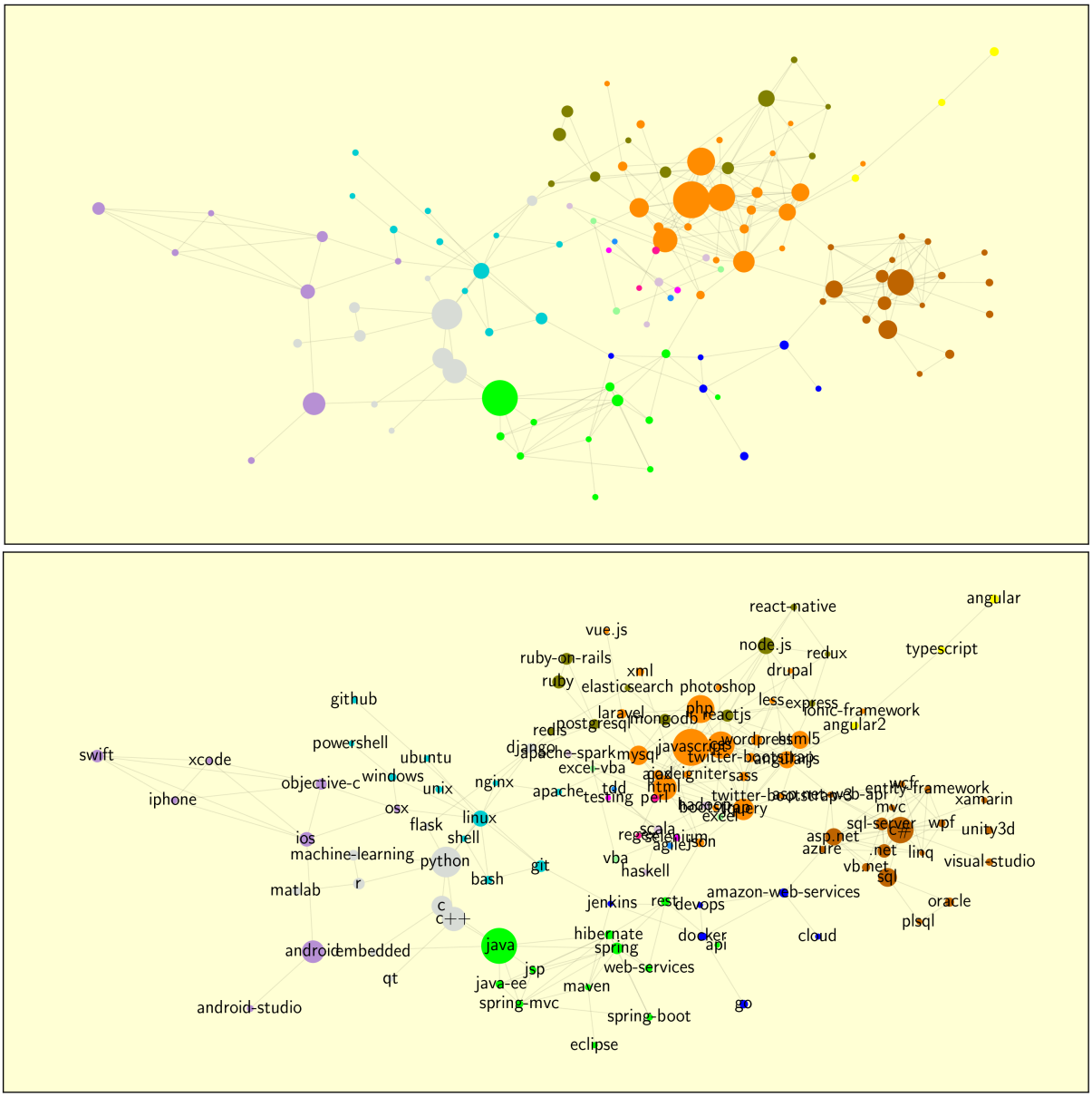

Figure 6: The stackoverflow dataset with communities coloured distinctly

We've also included plots of subgraphs induced on the communities in appendix B to showcase the tags and the intra-community correlations.

We'd like to note that the community structures identified by the Walktrap algorithm are semantically valid and relevant. Walktrap identified the 14 different communities we've grouped in table 1.

Indeed, community 1 consists of general-purpose programming languages, save for *qt* and *embedded*. Qt is a cross-platform software and framework for making graphical user interfaces, and and *embedded* refers to embedded systems, which are the self-contained computer subsystems that underpin the facilities of modern life. Community 2 groups Microsoft-owned technologies, including cloud computing service Azure and cross-platform mobile development framework Xamarin. Community 3 groups technologies found in variants of the MERN (MEAN) solution stack. MERN/MEAN is an acronym for MongoDB, Express.js, React.js / Angular.js, and Node.js: these four technologies are used in conjunction for dynamic website development. Although Ruby is a general-

Community Number	Community Vertices
1	*c*, *c++*, *django*, *python*, *flask*, *machine-learning*, *qt*, *r*, *embedded*, *matlab*
2	*asp.net*, *c#*, *.net*, *sql-server*, *entity-framework*, *linq*, *wcf*, *wpf*, *asp.net-web-api*, *plsql*, *oracle*, *visual-studio*, *sql*, *vb.net*, *unity3d*, *xamarin*, *azure*, *mvc*
3	*ruby*, *ruby-on-rails*, *redux*, *reactjs*, *react-native*, *express*, *node.js*, *mongodb*, *redis*, *elasticsearch*, *postgresql*
4	*ios*, *swift*, *objective-c*, *iphone*, *android*, *xcode*, *android-studio*, *osx*
5	*github*, *git*, *apache*, *nginx*, *bash*, *linux*, *windows*, *ubuntu*, *unix*, *shell*, *powershell*
6	*html*, *css*, *html5*, *javascript*, *jquery*, *php*, *mysql*, *less*, *sass*, *ajax*, *angularjs*, *laravel*, *json*, *xml*, *wordpress*, *codeigniter*, *twitter-bootstrap*, *ionic-framework*, *vue.js*, *drupal*, *bootstrap*, *twitter-bootstrap-3*, *photoshop*
7	*angular2*, *typescript*, *angular*
8	*hibernate*, *spring*, *spring-mvc*, *spring-boot*, *java*, *java-ee*, *maven*, *jsp*, *web-services*, *rest*, *eclipse*, *api*
9	*jenkins*, *docker*, *amazon-web-services*, *go*, *devops*, *cloud*
10	*hadoop*, *apache-spark*, *scala*, *haskell*
11	*testing*, *selenium*
12	*tdd*, *agile*
13	*regex*, *perl*
14	*excel*, *excel-vba*, *vba*

Table 1: Communities identified by the Walktrap algorithm in the stackoverflow dataset

purpose programming language, Ruby on Rails is a server-side web application framework written in Ruby, hence its inclusion. Community 4 groups together Apple's technology stack and Android, the latter likely due to their joint monopoly of mobile devices and therefore presence in bilateral discussions around mobile application development.

Community 5 appears to group together operating systems, webservers, and shell scripting languages, seemingly reflective of system administrators (sysadmins). Community 6 groups together front-end web development tools and platforms, web application frameworks, and Adobe Photoshop interestingly enough. Community 7 groups together TypeScript, Angular, and Angular2. TypeScript was created by MicroSoft and is a syntactical superset of the JavaScript programming language, used to develop both front- and back-end web applications. Angular, otherwise known as AngularJS, was a JavaScript front-end web framework, which was superseded by Angular2, now known as Angular, a rewrite of Angular by Google. Community 8 mainly comprises technologies in the Java ecosystem. It also contains webserver-related terminology.

Community 9 groups together software development and IT operations or *devops*-related products and services, and the general purpose programming language Go. Community 10 represents the realm of parallel and distributed computing, as evidenced by the presence of Hadoop, Apache Spark, and Scala. It also contains Haskell, a notable functional programming language. Community 11 grouped together Selenium, a web browser automation tool typically used either for web scraping or testing, with the testing tag:

an apt association. Community 12 groups test-driven development (TDD) and Agile, which are software development methodologies. Community 13 combines Perl, a general purpose programming language which gained popularity in the 1990s partly because of its powerful string and regular expression parsing capabilities, and regular expressions (regex) which are strings that define a search pattern. Finally, community 14 correctly groups Microsoft Excel and Visual Basic for Applications (VBA) given that the latter is the former's macro programming language.

In fig. 7, we present the results of running algorithm 2 on the unweighted graph representting the Stack Overflow dataset, which we obtained by discarding edge weights.

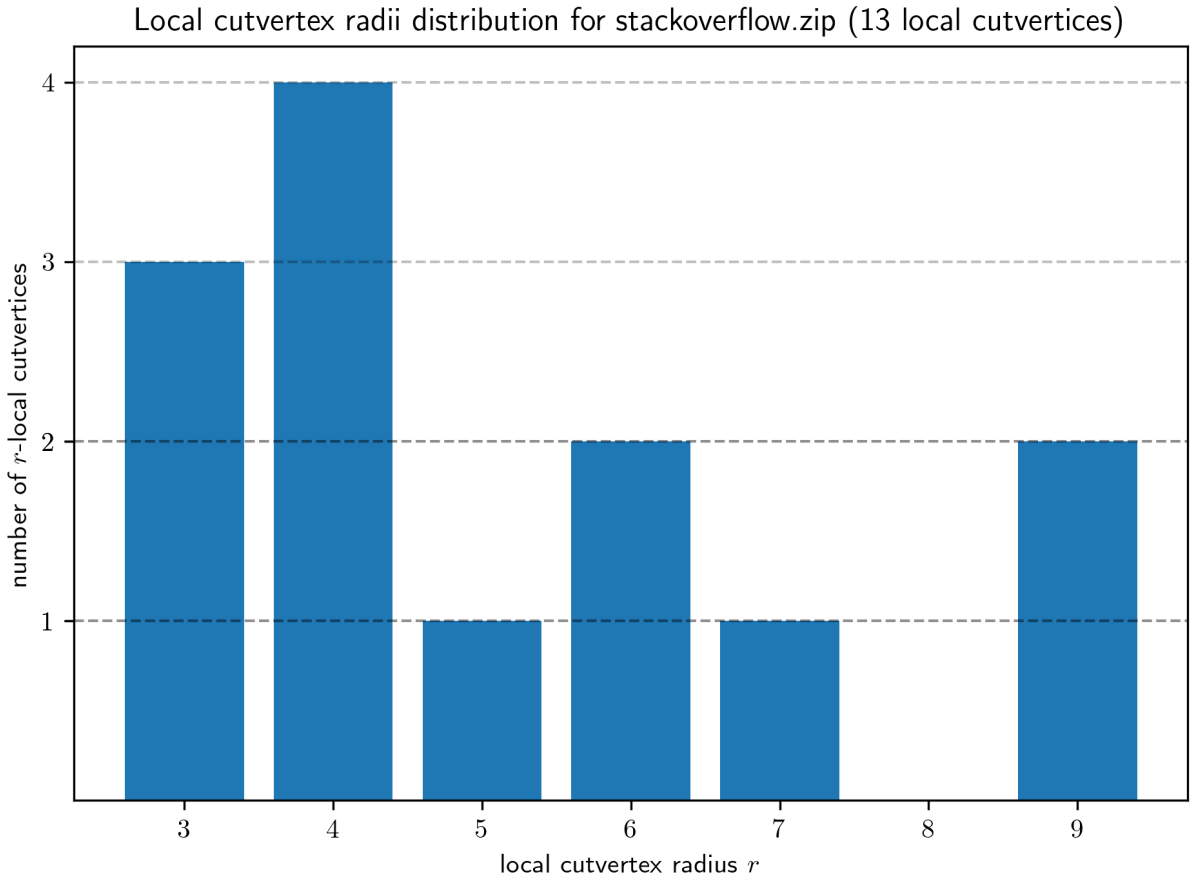

Figure 7: Distribution of local cutvertex radii in the stackoverflow dataset

For each identified local cutvertex v, we then obtained the parametric ball around v and split at v, distinctly highlighting the components of $B\left(v, \frac{r}{2}\right) - v$. In this section we've chosen to turn our attention to the interpretations of only a subset of all identified local cutvertices, but we've included the drawings for the other local cutvertices in appendix C.

Generally, we found that for each local cutvertex, the number of components in its punctured ball was 2. We can interpret this as each local cutvertex straddling two categories of relevance.

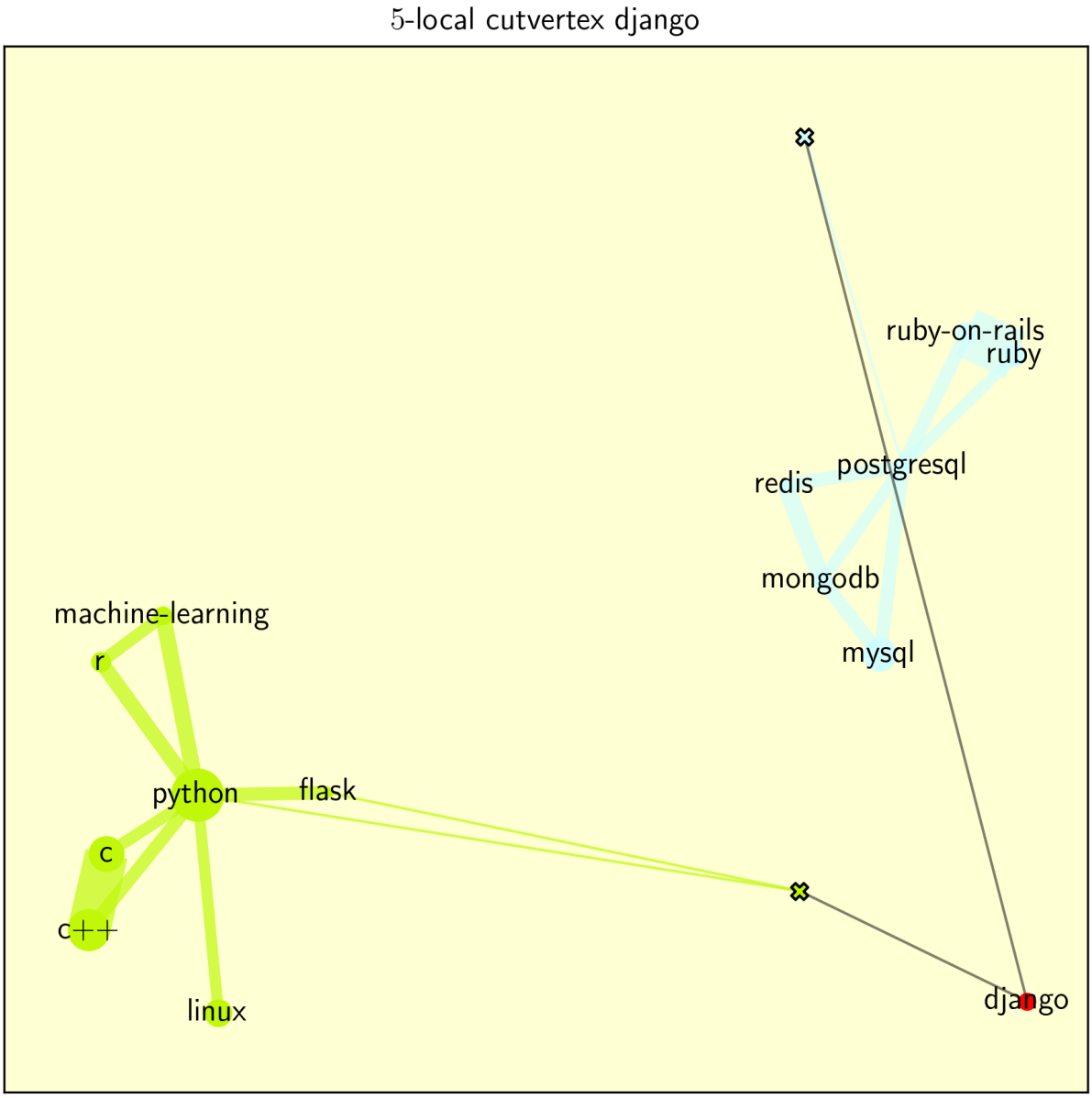

Figure 8: Components of the punctured ball around the local cutvertex django

For example, Django fig. 8, a 5-local cutvertex, separates out Python, the programming language Django is written in, Flask, another Python web framework, from tags such as MySQL, PostGreSQL, MongoDB, Redis, which are data structure stores used in conjunction with Django in web development, and Ruby on Rails, a Django equivalent. Django then appears to be a critical link between communities 1 and 3.

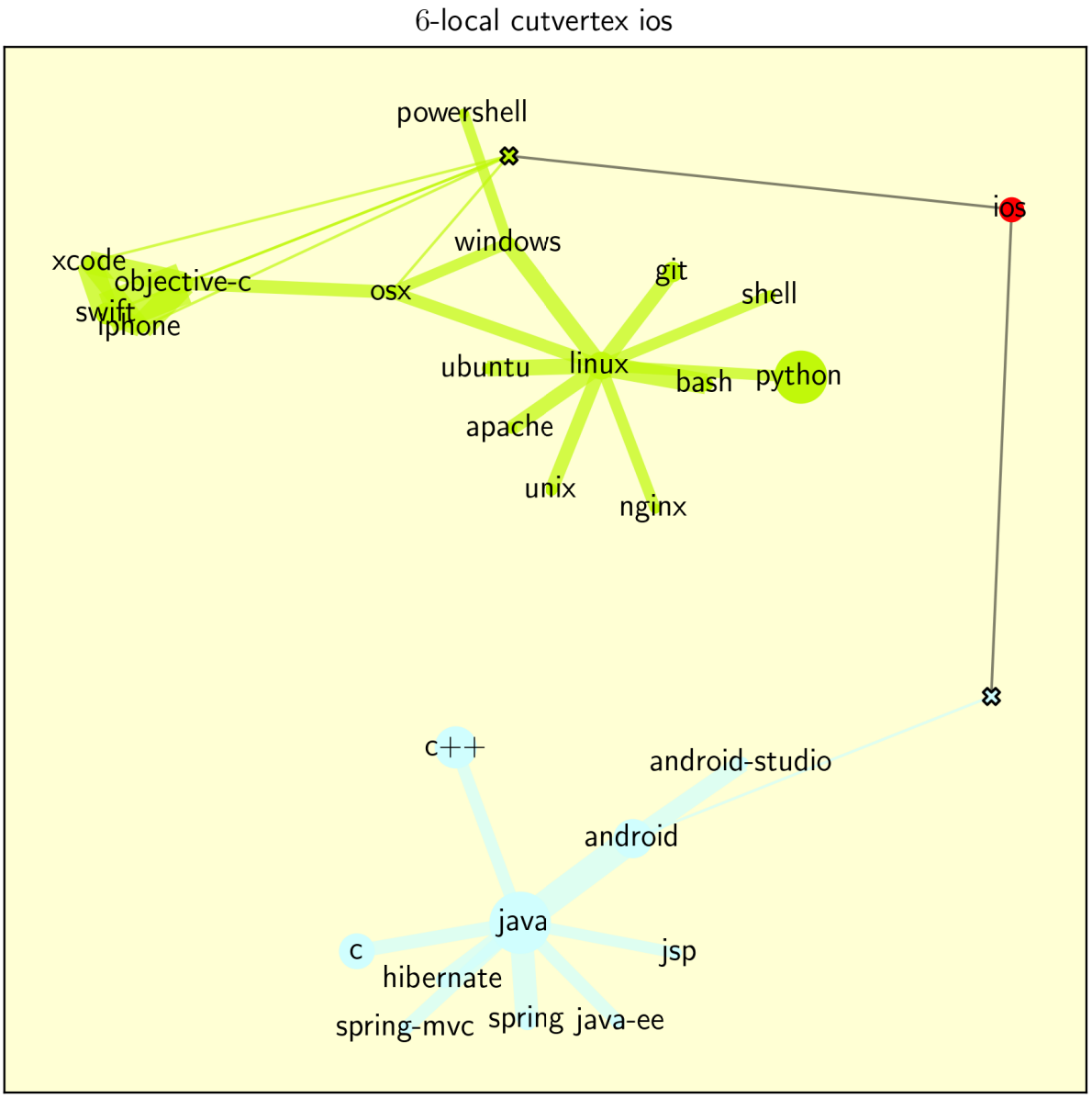

Figure 9: Components of the punctured ball around the local cutvertex ios

iOS represents a 6-local cutvertex in this dataset. The immediate neighbours of iOS fig. 9 in its parametric ball are, in the one component of the punctured ball, the rest of the Apple ecosystem, and in the other, Android. However the components of its punctured parametric ball are mergers of different communities. The green component comprises the rest of the Apple ecosystem (community 4), joined together with the sysadmins group (community 5) via OSX, Apple's computer operating system. The blue component is composed of Android and Android Studio which are members of iOS' own community (4), the Java ecosystem (community 8), and programming languages C and C++ from community 1. Given this local cutvertex's small radius, this reflects how the communities identified by the Walktrap algorithm are, at a local scale, tight-knit.

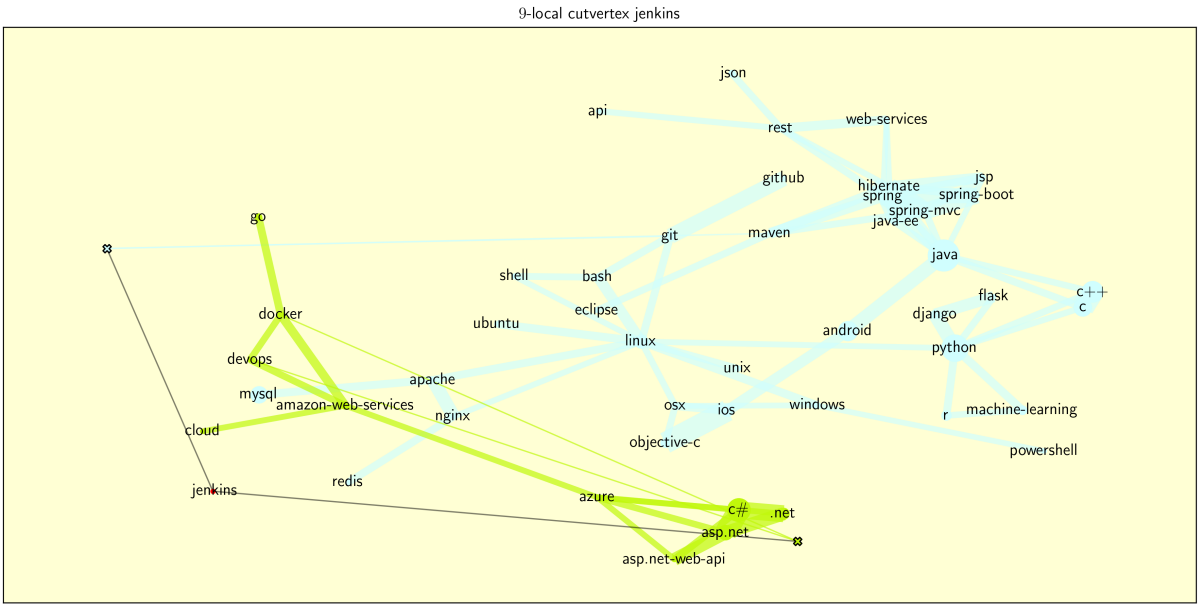

Figure 10: Components of the punctured ball around the local cutvertex jenkins

In contrast to this, Jenkins fig. 10 represents a 9-local cutvertex in this dataset. Jenkins is an automation server used to automate parts of the software development cycle including but not limited to testing and continuous integration. As such, it would be a tool used by a wide range of developers and not be limited to a single community. This is shown both by the fact that it is a local cutvertex whose punctured ball contains two large, heterogeneous components, and by the fact that its radius is the largest in the dataset.

4.2 Network_Data_MJS20 dataset

In ecology, the trophic level of an organism refers to the level it occupies in a food web. By representing a food web as a directed graph, this notion now corresponds to the graph property known as trophic coherence. For their paper linking looplessness and trophic coherence, Johnson compiled a trove of directed graphs [25] under the Network_Data_MJS20 dataset. Although most are representative of food webs, these graphs are organised into seven different categories. We've listed the names of each graph used as well as detailed their sources in appendix A.

Category	Food Webs	Genetic	Language	Metabolic	Neural	Social	Trade
# of graphs	42	8	1	7	8	3	5

Table 2: Categories of graphs in the NDMJS20 dataset

For our analysis, we chose to interpret each directed graph in the dataset as an undirected graph by viewing directed edges as undirected edges, as well as stripping weighted edges of their weights. We also chose to preserve loops.

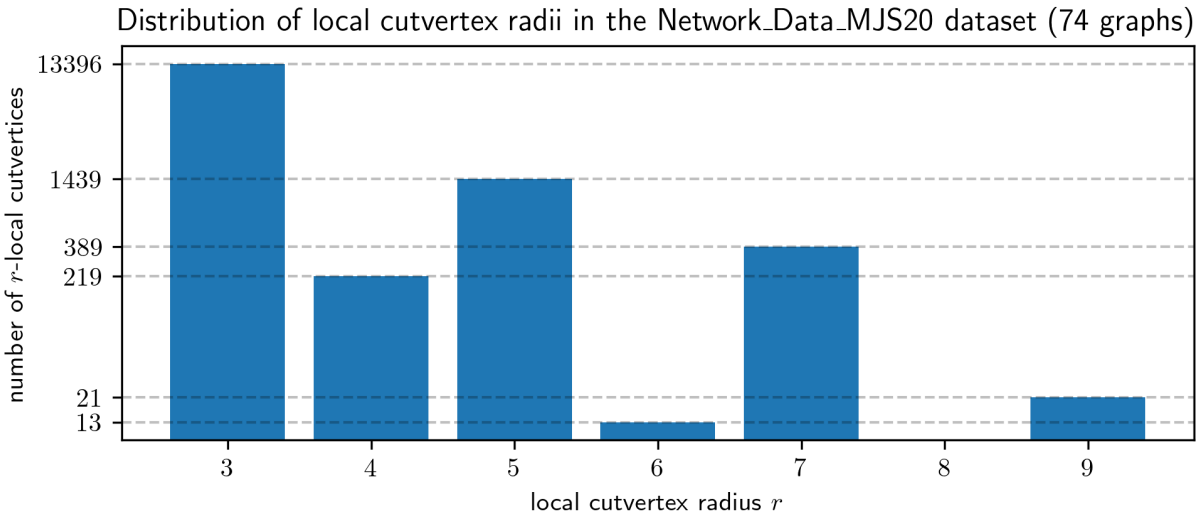

Figure 11: Distribution of local cutvertex radii in the NDMJS20 dataset (74 graphs)

In fig. 11 we've presented the distribution of the radii of the local cutvertices identified across all graphs in this dataset. Our first observation is that none of the graphs in this dataset possess local cutvertices with radii greater than 9, which given that these graphs possess a comparatively large number of vertices and edges, speaks to their high local connectivity. We also see that the majority of identified local cutvertices are 3-local. Note that if v is a 3-local cutvertex then it is does not form a wheel graph with its neighbours, for which it would have been the central vertex. This in turn implies that v does not form any triangles with its neighbours, therefore the presence of a 3-local cutvertex indicates the absence of a triangle. Let $n_3(G)$ denote the number of 3-local cutvertices in G and let $t(G)$ denote the percentage of triangles present in G among the maximum possible. We've calculated these values for each graph in the dataset and presented them in table 3.

Indeed, when comparing the number of triangles present in each graph in the dataset to the maximum possible number of triangles we do find that the graphs in the dataset possess relatively few triangles.

G	$n_3(G)$	$t(G)$
net_cancer	2001	0
net_non_cancer	1749	0
net_s_cerevisiae	1117	0
net_SC	1021	0
net_AG	770	0
net_CE	749	0
net_TH	710	0
net_MJ	679	0
net_m_tuberculosis	588	0
net_e_coli	454	0
Lough_Hyne	276	0.13
net_CT	260	0
net_CQ	212	0
cayman_islands	198	0.26
net_yeast	192	0
Weddel_sea	179	0.71
net_p_aeruginosa	125	0
net_celegans_neural	121	0.07
el_verde	105	0.58
DempstersSutxt	89	0.29
canton	84	0.07
Kyeburntxt	83	0.14
Healytxt	81	0.19
stony	80	0.06
Stonytxt	80	0.05
DempstersAutxt	76	0.03
DempstersSptxt	70	0.02
Blackrocktxt	68	0.07
Broadtxt	68	0.01
Martinstxt	68	0.06
LilKyeburntxt	63	0.05
Ythan96	62	0.42
net_coli	61	0
SuttonSutxt	60	0.01
Berwicktxt	58	0.02
SuttonAutxt	57	0.02
SuttonSptxt	57	0.01

G	$n_3(G)$	$t(G)$
Powdertxt	55	0.05
NorthColtxt	54	0.07
Germantxt	50	0.09
Venlawtxt	47	0.04
Troy	42	0.03
Narrowdaletxt	40	0.05
st_marks	35	1.29
rhesus_brain_1	33	0.55
Catlins	32	0.05
Coweeta17txt	32	0.05
Coweeta1txt	25	0.12
broom	23	0.27
st_martin	23	2.13
net_moreno_highschool	20	0.56
little_rock	18	3.02
chesapeake	15	0.47
net_green_eggs	15	0.1
net_social_leader	14	0.65
shelf	13	13.36
bridge	12	4.43
grass	10	0.08
net_social_prison	8	0.12
skipwith	6	25.43
reef	3	11.21
benguela	0	14.07
coachella	0	22.14
cat_brain	0	8.27
mouse_brain_1	0	39.2
rat_brain_1	0	2.73
rat_brain_2	0	3.17
rat_brain_3	0	3.81
rhesus_brain_2	0	1.57
net_trade_basic	0	42.74
net_trade_crude	0	43.33
net_trade_diplomats	0	43.13
net_trade_food	0	45.06
net_trade_minerals	0	8.1

Table 3: Number of 3-local cutvertices and percentage of triangles present ([2dp]) in the NDMJS20 dataset

Another phenomenon we've noticed is that 13 of the graphs in this dataset did not present any local cutvertices as identified by algorithm 2 i.e. no r-local cutvertices that aren't global cutvertices for $r \geq 3$.

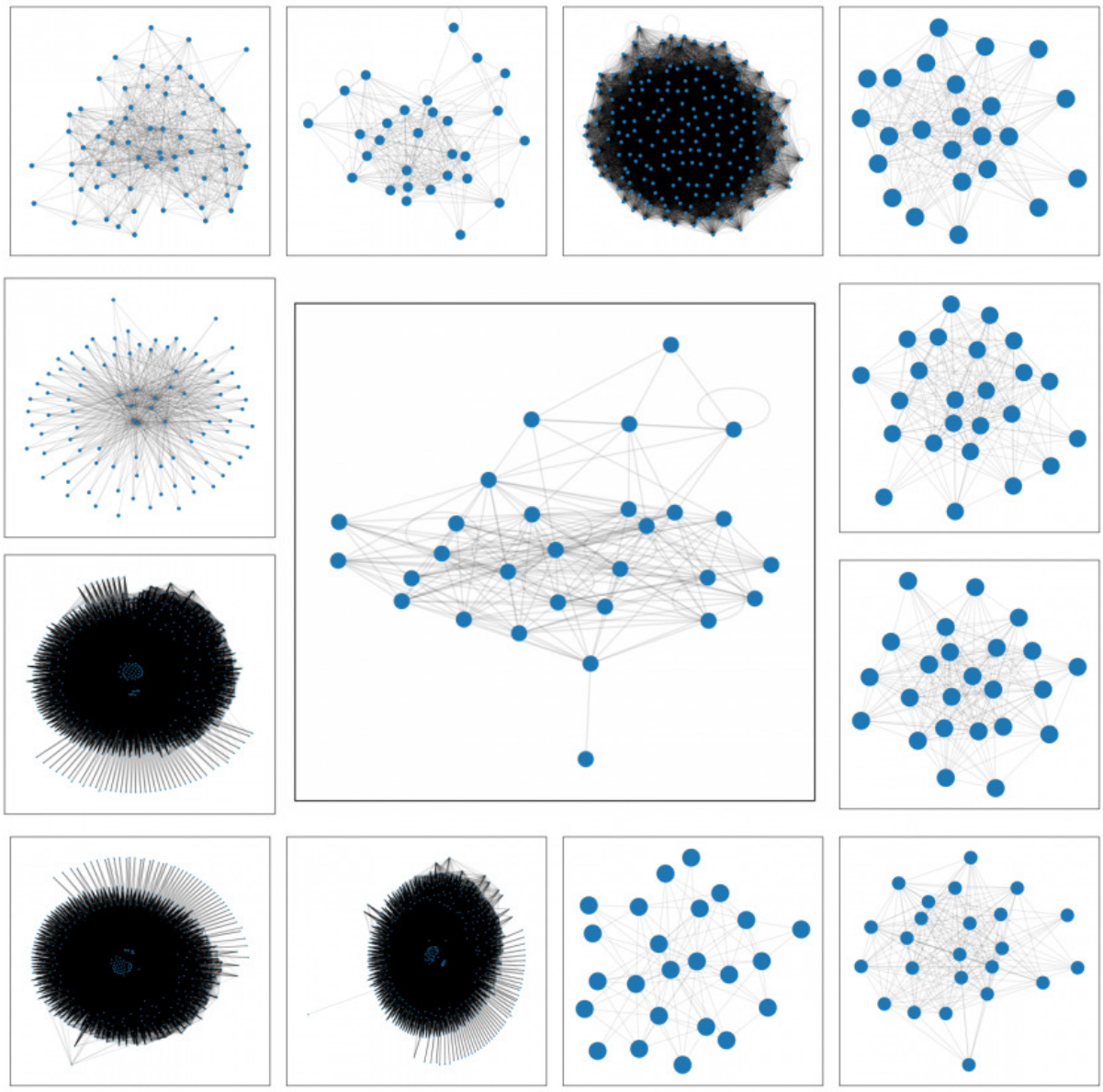

Figure 12: Graphs in the NDMJS20 dataset with no local cutvertices. In the middle, benguela, and clockwise from the top left: cat_brain, coachella, mouse_brain_1, net_trade_basic, net_trade_crude, net_trade_diplomats, net_trade_food, net_trade_minerals, rat_brain_1, rat_brain_2, rat_brain_3, and, rhesus_brain_2,

By inspection, we suspect this may be because these graphs appear 'highly-connected', a term which we'd like to able quantify. To this end, we've computed several metrics for these graphs in table 4. Recall that $L(G)$ and $C(G)$ are the characteristic path length and clustering coefficient of G respectively. We denote the connected components of G by $\mathcal{C}_G$, and the ratio

$$\rho\left(G\right) := \frac{|E|}{\binom{|V|}{2}} \in [0,1]$$

is the edge density of G.

From table 4 we see that each of the 13 graphs are connected and possess a large number of edges. They also present significant clustering coefficients, small characteristic

$G=(V,E)$	$\lvert V\rvert$	$\lvert E\rvert$	$\delta(G)$	$\Delta(G)$	$\lvert\mathcal{C}_G\rvert$	$L(G)$	$C(G)$	$\rho(G)$
benguela	29	198	1	26	1	1.6207	0.5785	0.4877
coachella	29	240	5	26	1	1.4754	0.654	0.5911
cat_brain	65	730	3	45	1	1.6995	0.6614	0.351
mouse_brain_1	213	16242	86	207	1	1.2874	0.7535	0.7194
rat_brain_1	503	23030	1	497	1	1.8222	0.8911	0.1824
rat_brain_2	502	24656	7	499	1	1.8047	0.8908	0.1961
rat_brain_3	493	25988	7	479	1	1.7897	0.8831	0.2143
rhesus_brain_2	91	582	1	87	1	1.8681	0.8696	0.1421
net_trade_basic	24	195	6	23	1	1.2935	0.8543	0.7065
net_trade_crude	24	195	5	23	1	1.2935	0.8535	0.7065
net_trade_diplomats	24	199	8	23	1	1.279	0.8223	0.721
net_trade_food	24	201	7	23	1	1.2717	0.842	0.7283
net_trade_minerals	24	97	2	22	1	1.6594	0.7322	0.3514

Table 4: Metrics for the graphs in the NDMJS20 dataset presenting no local cutvertices, according to algorithm 2

path lengths, and typically large minimum degrees. Their edge densities however are quite heterogeneous.

However to explain the absence of local cutvertices of any radius r, let us first observe what it means for there to be no local cutvertices at the smallest non-trivial radius $r=3$. For any $v\in V$, we claim that $B\left(v,\frac{3}{2}\right)-v$ is $G\left[N_G(v)\right]$. If then there are no cutvertices in $G\left[N_G(v)\right]$ then we deduce that $G\left[N_G(v)\right]$ is 2-connected, since removing any of its vertices does not disconnect it. Although not a sufficient condition, note that $\delta(G)\geq 2$ is necessary for a graph G to be 2-connected.

According to table 4, only 3 graphs do not meet the condition $\delta(G)\geq 2$: *benguela*, *rat_brain_1*, and *rhesus_brain_2*. However, they only possess one leaf each, as shown in fig. 13.

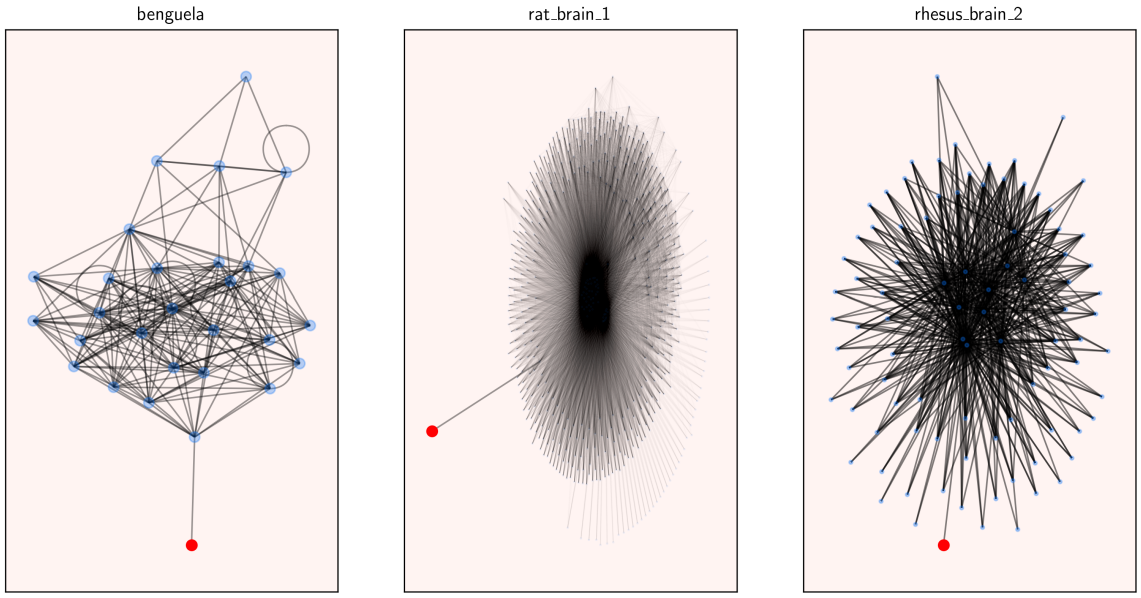

Figure 13: Outlier graphs with leaves highlighted in red

For this reason we will still proceed with a preliminary test of the following hypothesis on the non-outlier graphs, and then if successful we'll include the outlier graphs but with their leaves omitted, with further investigation for the case of their leaves to follow. First, let us formalise our argument.

Lemma 4.1. Let G be a connected graph. If

$$\forall v \in V : G\left[N_G(v)\right] \text{ is connected}$$

then G has no r-local cutvertices for all $r \geq 3$.

Proof. Fix $v \in V$. We'd like to show that $G\left[N_G(v)\right] = B\left(v, \frac{3}{2}\right) - v$. Since we're dealing with unweighted graphs, all distances between vertices are integer values, therefore it follows that

$$V\left(G\left[N_G(v)\right]\right) = V\left(B\left(v, \frac{3}{2}\right) - v\right)$$

By definition 2.5, $E\left(B\left(v, \frac{3}{2}\right)\right)$ comprises all edges where v is one endpoint, as well as the edges between vertices x, y with $d_G(v, x) = d_G(v, y) = 1$ that are also present in G. Removing v from $B\left(v, \frac{3}{2}\right)$ then discards the former while preserving the latter types of edges. This indeed corresponds to $E\left(G\left[N_G(v)\right]\right)$.

Thus $G\left[N_G(v)\right] = B\left(v, \frac{3}{2}\right) - v$. Now if $B\left(v, \frac{3}{2}\right) - v$ is connected, then v is not a 3-local cutvertex, and so v isn't a 4-local cutvertex by the contrapositive of lemma 2.2. Arguing in this way ad infinitum, and given that v was arbitrary, we conclude that G has no r-local cutvertices for all $r \geq 3$. □

We now proceed to check if for each of the 10 graphs the condition in lemma 4.1 is met. We do so by verifying that the induced subgraph on the neighbourhood of each vertex for each graph is indeed connected. We present our implementation below, with `get_outlier_graphs` a function that returns a list of the aforementioned 10 graphs, which did indeed find that all 10 graphs did in fact meet this necessary condition, a reassuring result.

```
def check_hypothesis() -> List[bool]:
    graphs: List[nx.Graph] = get_outlier_graphs()
    results: List[bool] = []
    for G in graphs:
        for v in G.nodes():
            Nv: Set[Vertex] = set(G.neighbors(v))
            GNv: nx.Graph = nx.subgraph_view(
                G, filter_node=lambda x: x in Nv
            )
            if not nx.is_connected(GNv):
                results.append(False)
                break
        else:
            results.append(True)
    return results
```

Following this, we removed the leaves from each of the 3 outlier graphs *benguela*, *rat_ brain_ 1*, and *rhesus_ brain_ 2* and found that the resulting graphs did too meet the necessary condition in lemma 4.1.

In regards to their leaves, recall that algorithm 2 only identifies local cutvertices that do not separate their component. table 4 shows us that each of these graphs have a single component, which is why the leaves' only neighbours were not identified as local cutvertices by algorithm 2, given that those neighbours separate their respective graphs.

4.3 Major Open Road Network

The Major Road Network (MRN) is a classification of roads in the United Kingdom conceived in a Rees Jeffreys Road Fund study by Daniel Quarmby and Phil Carey in October 2016 [35]. The MRN comprises the Strategic Road Network, a collection of major A roads which are nationally controlled, and more locally controlled A roads. Altogether, and at the time, the MRN represented only 4% of the UK's road length while containing 43% of the UK's traffic flow [35].

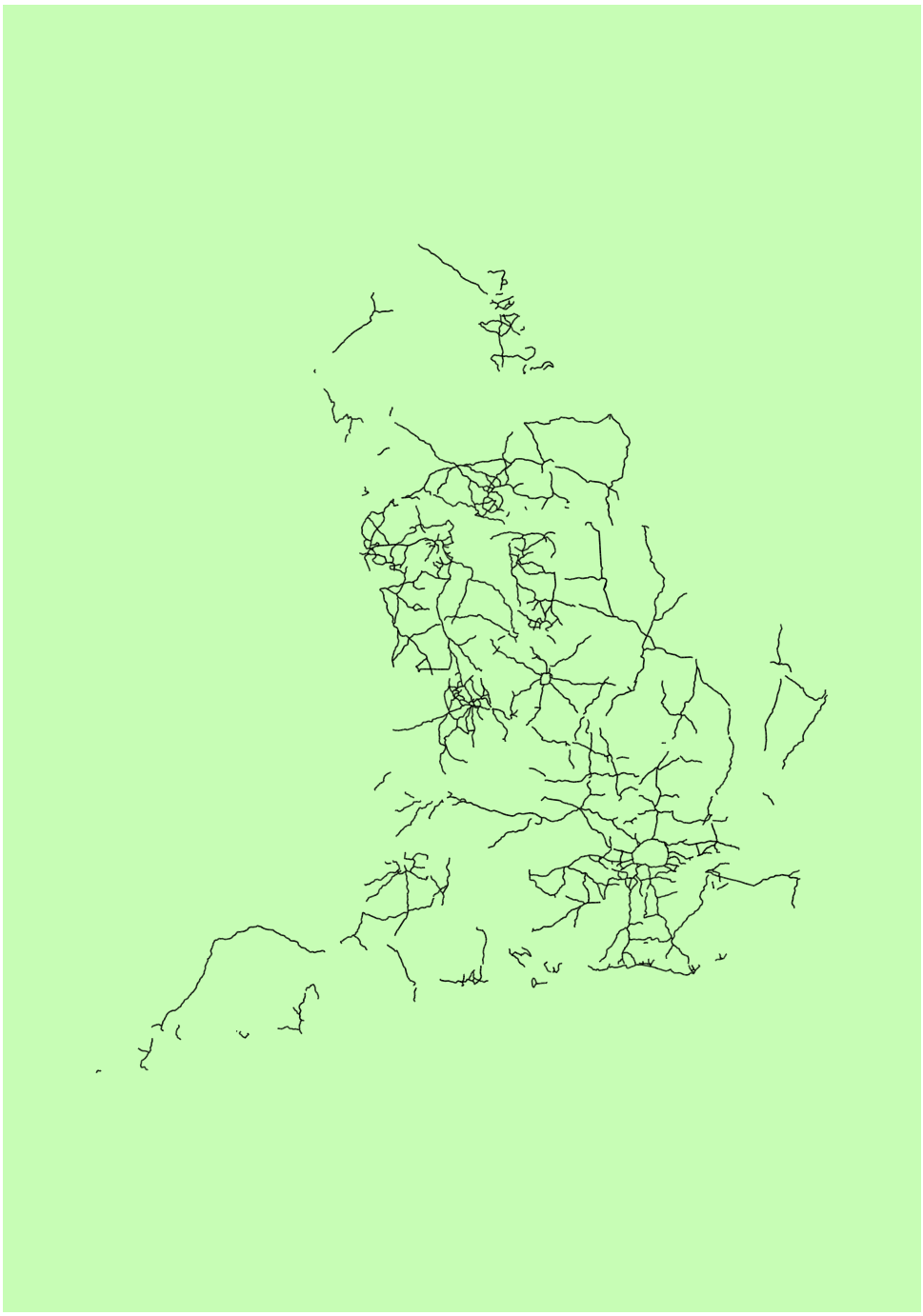

Figure 14: High-level view of the MRN dataset

The dataset for the MRN, which consists of a shapefile, was released by the Depart-

ment for Transport in December 2020 [48]. This dataset is rather highly detailed and fragmented, being composed of exactly 62292 disconnected path graphs whose lengths are in $[2, 127]$, totalling 297788 vertices and $297788 - 62292 = 235496$ edges in its representative graph G. We note that in the dataset, each point in every path graph has an associated Cartesian coordinate pair, and it is drawing G using these coordinates that recovers the familiar geography of the UK.

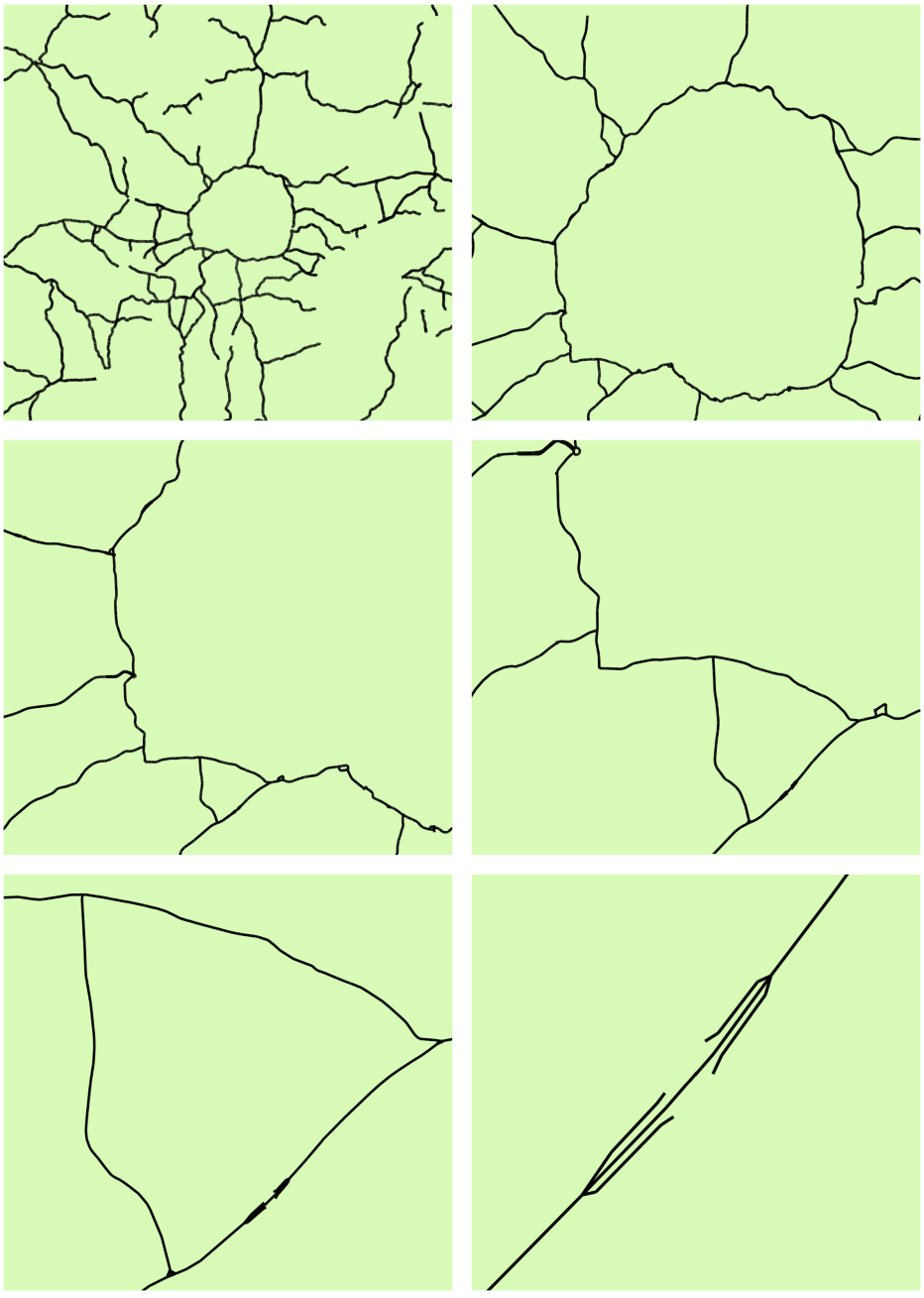

Figure 15: London, identified in the MRN dataset. Features such as roundabouts and exit roads are visible in this dataset.

As highlighted in the Code section of this report, we're interested in strictly local cutvertices, which we recall to have defined as local cutvertices which do not separate their components. Given that this dataset consists of a large number of disconnected

components, our algorithm wouldn't be able to find local cutvertices of interest as G currently stands. It is this reason that motivated the following post-processing operations on G.

The discrepancy between the large number of disconnected components in G and the visibly fewer number of components when drawing G prompted an investigation into the distribution of Cartesian coordinates of the points in the polylines. We initially hypothesised that subsets of vertices in G, which were not part of the same component, had similar Cartesian coordinates. To test this hypothesis, we began by finding, for a vertex a in a component C of G, a vertex b in a component $D \neq C$ of G with the smallest Euclidean distance to a.

Distance	0	$(0, 1]$	$(1, 10]$	$(10, 100]$	$(100, 1000]$	$(1000, 2000]$	> 2000
#	123634	676	34786	99805	37403	1187	297

Table 5: Distribution of distances in meters from a to b

As hypothesised, table 5 does indeed show that a considerable subset of vertices were drawn atop one another. Spurred by this, we then identified the groups of overlapping vertices in order to determine whether or not we could carry out a simplification of the graph. We clarify that the overlapping groups of vertices we identified are maximal.

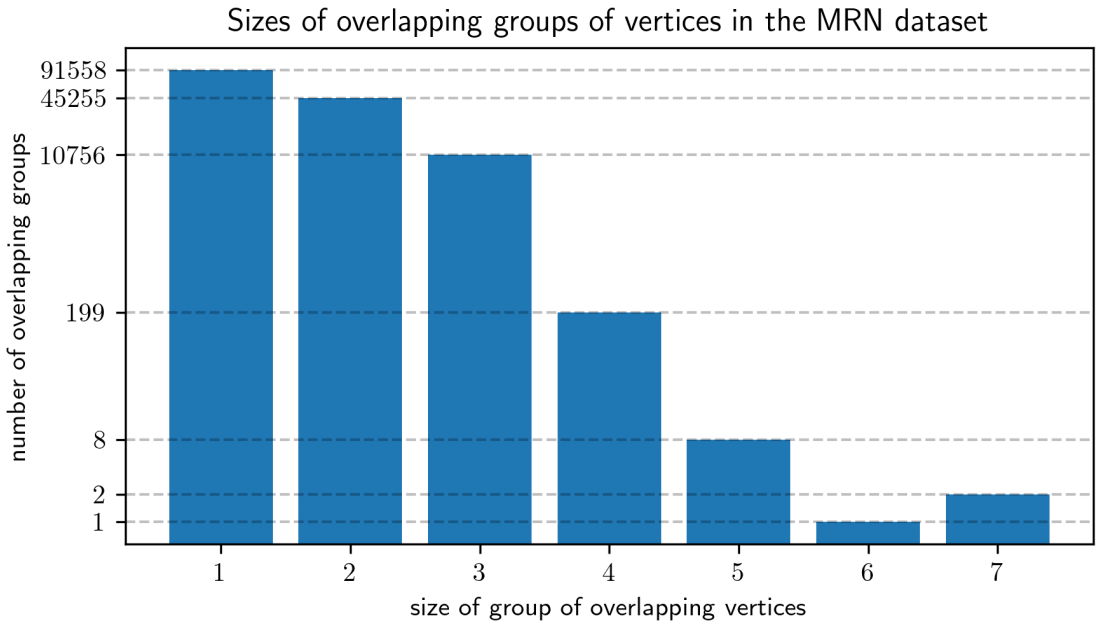

Figure 16: Distribution of sizes of overlapping groups

fig. 16 shows the distribution of the sizes of the overlapping groups of vertices. We note that the groups of size 1 reflect vertices which were not drawn atop others, and that 56222 distinct maximal groups of overlapping vertices were identified.

According to fig. 16, roughly 30% of the vertices in the MRN dataset's representative graph are not drawn atop other vertices. With the aim of obtaining a representative graph that's closer to a navigable road network, we've opted to iteratively collapse an overlapping group of vertices to a single vertex, merging all neighbourhoods into one, which we'll refer to as *collapsing*.

Although we've been able to identify 5 components of the original 62292 to be entirely redundant given that each of their vertices are part of an overlapping group of vertices, we do note that collapsing vertices results in an idealised reduction of the initial dataset. Nonetheless, we posit that the resulting graph is still of interest for analysis given its strong grounding in reality.

The exact procedure is as follows. Let G be the representative graph of the dataset. Given a maximal overlapping group of vertices

$$\{v_1, \dots, v_k\} \subseteq V(G)$$

without loss of generality we choose v_1 as our anchor and construct the edges

$$\{\{v_1, u\} \mid u \in \mathcal{N}\}$$

where

$$\mathcal{N} = \bigcup_{i=2}^{k} N_G(v_i)$$

and remove the vertices $v_2, \dots, v_k$ from G. Note that $v_1 \notin \mathcal{N}$ since $v_1, \dots, v_k$ are in k distinct components by definition.

Lemma 4.2. Let

$$\{a_1, \dots, a_k\} =: A \neq B := \{b_1, \dots, b_l\}$$

be two distinct groups of overlapping vertices in G found according to the above conditions, i.e. no $a_i \neq a_j$ nor $b_i \neq b_j$ share a component in G and both A and B are maximal.

Let G' denote the graph obtained by first collapsing A then B, and G'' denote the graph obtained by first collapsing B then A. We have that $G' = G''$.

Proof. Collapsing a group of vertices deletes all but one vertex from the group, hence the only way for the order in which we collapse A, B to be relevant is if there exists $w \in V$ with $w \notin A$ and $w \in B$, such that w overlaps with a vertex $a_i \in A$ while there are no $a_j \in A$ that are in the same component as w (without loss of generality).

Suppose for a contradiction that such a vertex w exists. Then w shares the same Cartesian coordinates with A and $B \setminus \{w\}$. Therefore all vertices in $A \cup B$ overlap, and namely so does

$$X := \{a_1, \dots, a_k, w\} \subseteq A \cup B$$

Note then that $|X| = |A| + 1 > |A|$, contradicting A's maximality. Therefore no such w exists, and so $G' = G''$. □

Following lemma 4.2, we sequentially collapse each maximal overlapping group of vertices and note the number of components in the resulting graph. We've plotted this in fig. 17, where we've managed to reduce the number of components from 62292 to 139.

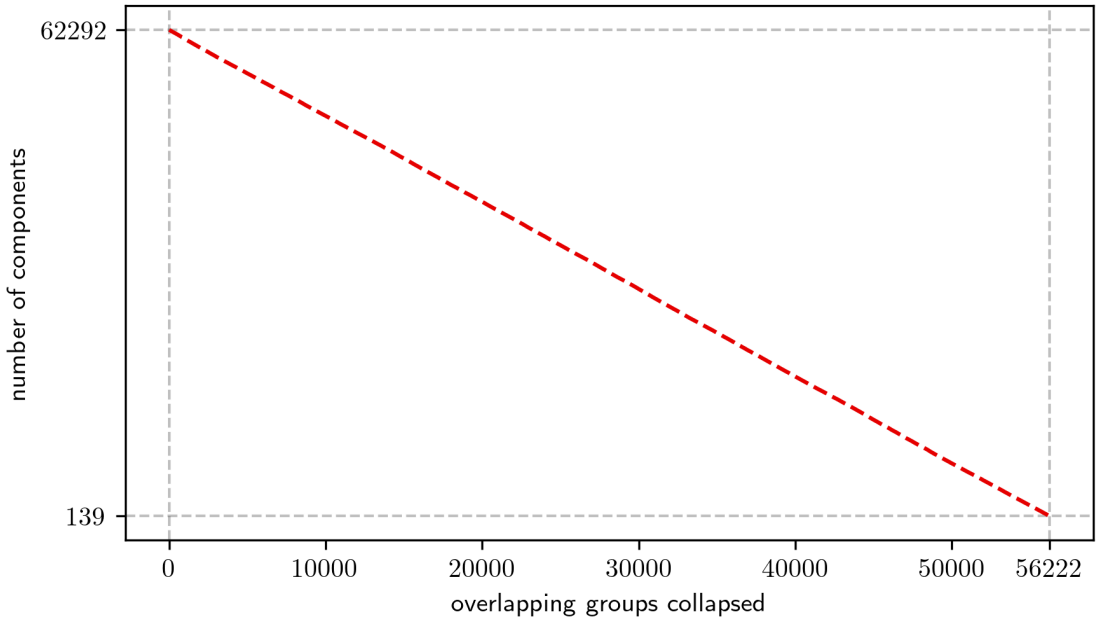

Figure 17: Number of components while collapsing overlapping groups of vertices

We now denote the graph obtained by collapsing all the aforementioned groups of overlapping vertices in G by H. Following this procedure, we are now in a position to apply algorithm 2 to H. However, upon visual inspection of H, we noticed the following.

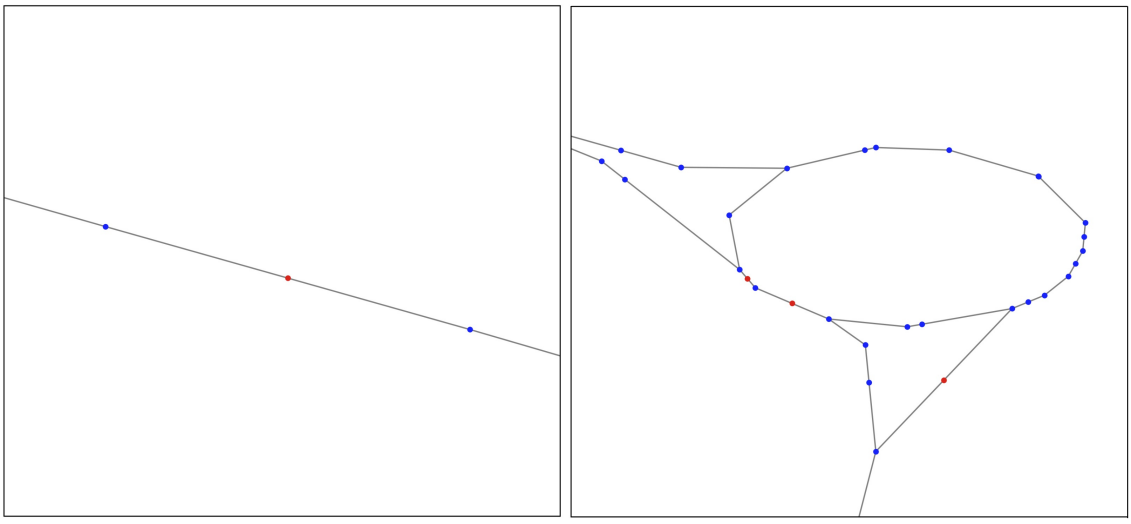

Figure 18: Examples of redundant vertices (red) within induced subgraphs of H

Along certain paths in H, there are vertices of degree 2 which appear collinear with their neighbours. If these vertices are indeed collinear with their neighbours, then they're not adding any pertinent information to the road network. In this case, we choose to label these vertices as redundant and wish to contract the edges on which they're present, both so as to improve the pertinence of each vertex and reduce the number of vertices in the graph given the time complexity of algorithm 2.

Formally, we envision redundant vertices to be subdividing vertices in H, wishing to

then contract the paths between branch vertices i.e. non-redundant vertices of H, in order to obtain the graph X of which H is a subdivision.

Considering all vertices in H of degree 2, we found that 15183 vertices are collinear with their neighbours. After having contracted all paths formed entirely by these redundant vertices, we obtained X, which has 6.59% fewer vertices than H.

Finally, we now apply algorithm 2 to X. algorithm 2 identified 80832 local cutvertices, which is over 37% of all vertices in the graph. This is a surprising result given that algorithm 2 does not recognise local cutvertices that separate their components and that the graph X still has a large number of connected components. We've plotted the distribution of local cutvertices radii in fig. 19.

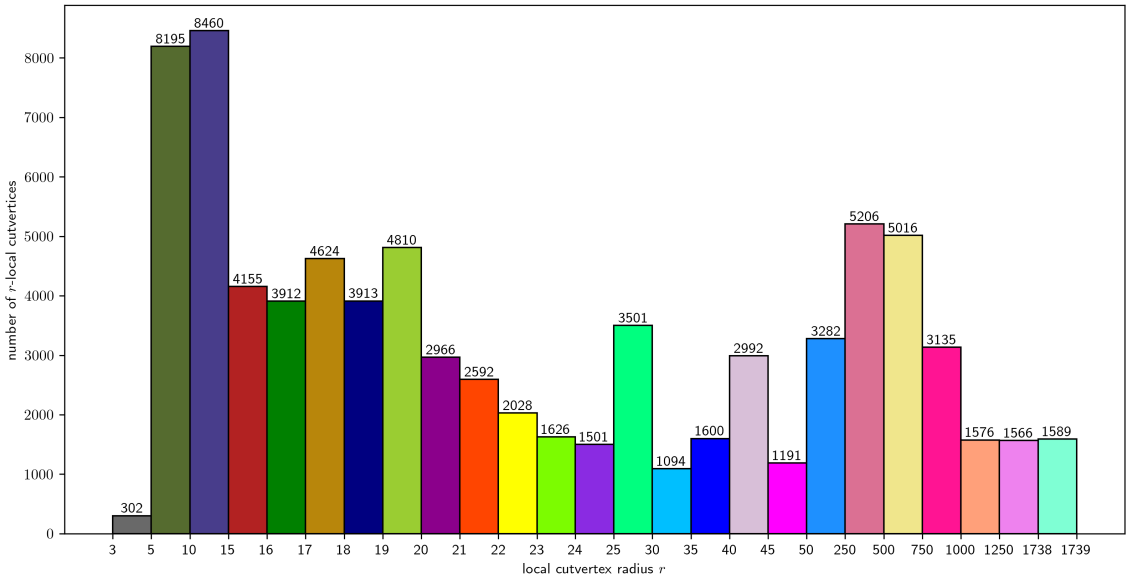

Figure 19: Distribution of local cutvertices' radii in the MRN dataset. Due to the large number and variety of magnitudes of radii we've opted for a histogram rather than a bar chart.

Overall, we note that this dataset contains a considerable and varied quantity of local cutvertices. There is a large concentration of r-local cutvertices in different ranges of r, most notably in the intervals $[5, 10]$, $[10, 15]$, $[15, 20]$, and $[20, 25]$, which we've highlighted out in fig. 19.

We turn our attention in particular to the large number of 1739-local cutvertices in this dataset. Given that we have access to metric information in this dataset, we've been able to compute the actual distance of roads that are separated by an r-local cutvertex v. We've plotted the correlation between the radius r and the aforementioned distance in fig. 20.

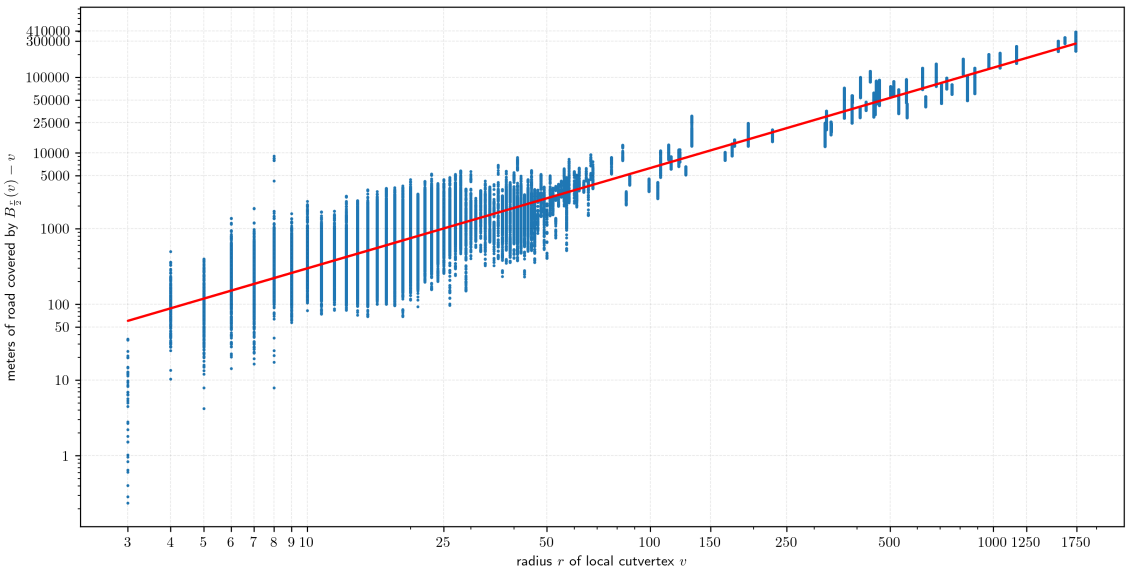

Figure 20: Radius of a local cutvertex in relation to the distance of road in meters separated by its punctured ball in a log-log plot. The red line is a fitted linear function $f(r) = 2.64427185 \cdot \log(r) + 1.32565675$.

The red line in fig. 20 is a linear function that relates the logarithms of the plotted variables, fitted to the data by minimising the squared error. The linear relationship between these logarithmic variables then indicates a relationship of the form

$$d = ar^k \text{ with } a, k \in \mathbb{R}$$

where d is the distance covered by the punctured ball of v and r is the locality of v, which is an interesting result. We hypothesise the relationship to be this way due to how long, straight roads are likely to occur, and how after removing redundant vertices they are guaranteed to be modelled by a single edge.

In particular, we find that the group of 1739-local cutvertices each separate between 250 and 390 kilometres of A roads. Moreover, we've identified all 1739-local cutvertices to both be concentrated, upon visual inspection, around Glasgow, and part of the same connected component in X.

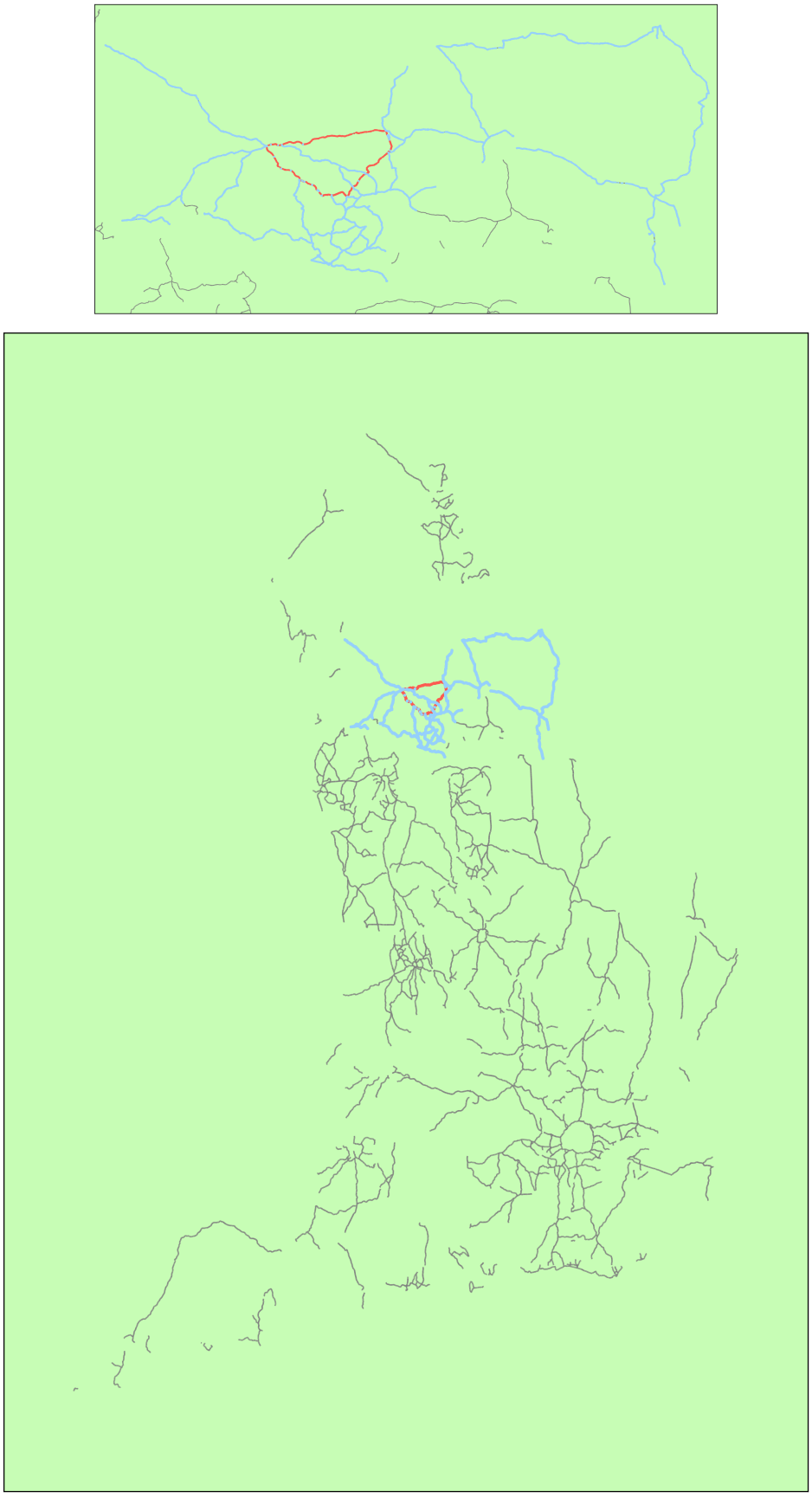

Figure 21: In red, all of the 1739-local cutvertices in the MRN dataset. In light blue, the rest of the vertices in the component to which all 1739-local cutvertices belong.

We've further managed to identify all but one of the roads in which each of the 1739-local cutvertices belong, which we've shown in fig. 22.

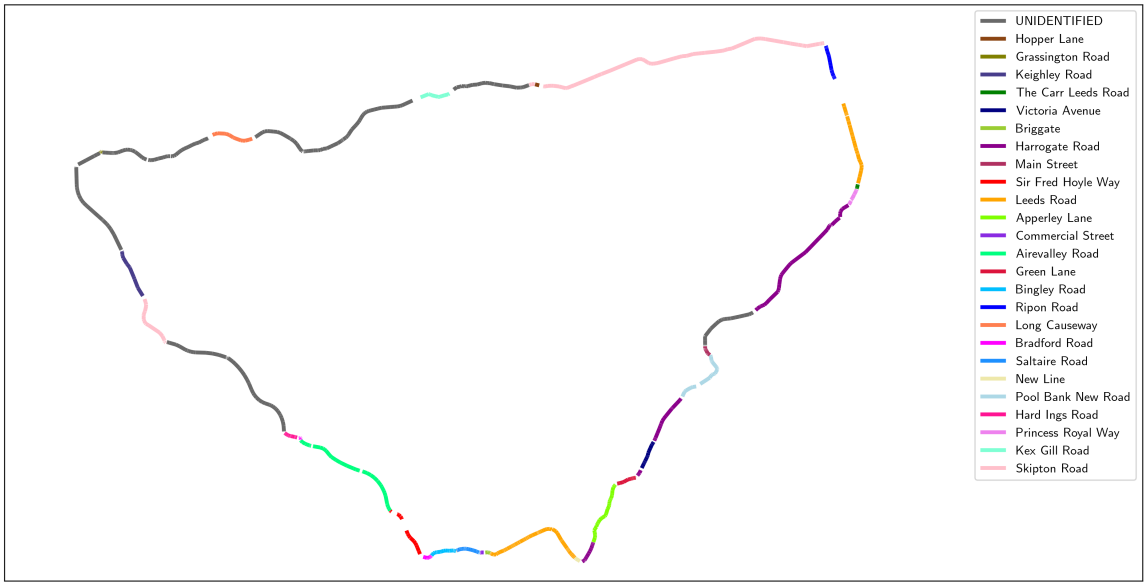

Figure 22: Names of the A roads in which each 1739-local cutvertices are located. Note that *UNIDENTIFIED* refers to edges for which the dataset did not have road metadata. The discontinuities in this plot come from our disregard of the transition zones between road names.

Recall that this dataset is but the major A roads in the UK, not all roads. Therefore, the conclusion we can draw from this observation is that the above A roads are critical for servicing traffic flow in the Eastern Glaswegian area.

We then split this graph at all its local cutvertices according to definition 3.1 and removed all split vertices to obtain X', the purpose of this being to obtain some idea of the decomposition of X along its local cutvertices. To observe the effect of this operation, we've plotted the size of the connected components identified in X'.

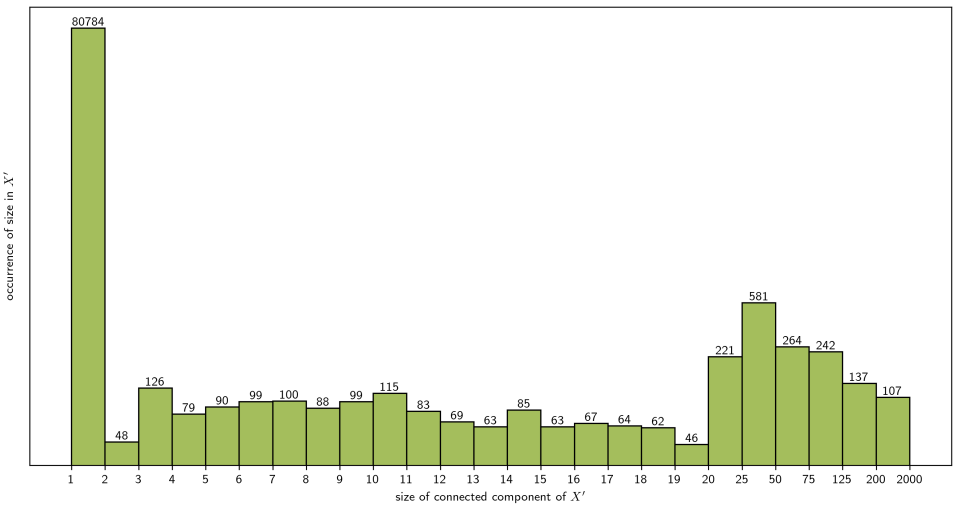

Figure 23: Distribution of the sizes of the connected components of X'

Since removing the split vertices isolates the local cutvertices from the graph, the

large number of isolated vertices is to be expected and in fact comparable to the number of local cutvertices we split at.

Decomposing X along its local cutvertices in this case has created both small components having fewer than 10 vertices and larger ones possessing 25 or more vertices. We thought it would be interesting to note the graphs to which some of these components are isomorphic. Since the isomorphisms for graphs on 2 or fewer vertices is uninteresting, and a large number of vertices becomes quickly untractable, we've restricted our attention to isomorphisms to graphs on 3, 4, and 5 vertices. We obtained all the connected graphs on $3, 4, 5$ vertices from the *House of Graphs* website [5] and checked for isomorphism by once more making use of the `nx.is_isomorphic` function.

We found that each of the 48, 79, and 90 components on 3, 4, and 5 vertices were isomorphic to P^3, P^4, P^5 respectively. The absence of isomorphisms to other graphs is interesting. Our interpretation of why this is the case is the fact that the observed dataset in question is a road network, where isomorphisms to path graphs are the most likely scenario.

4.4 Western US Power Grid

In [51], Watts and Strogatz compiled an undirected graph representation of the Western United States power grid, where vertices correspond to generators, transformers, and substations, and edges represent high-voltage transmission lines between them. The representative graph is connected, comprising 4941 vertices and 6594 edges.

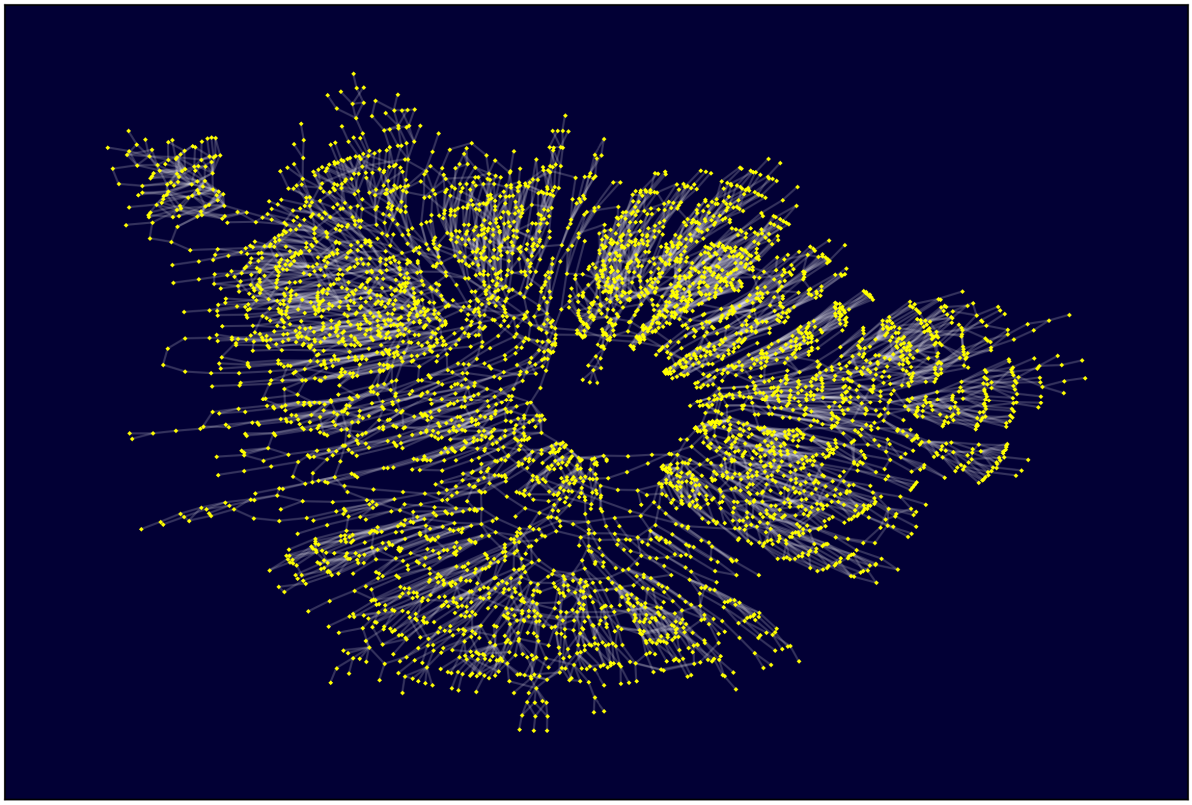

Figure 24: Western United States power grid as an undirected graph. The graph is drawn using the Kamada-Kawai graph drawing algorithm, a force-directed drawing algorithm, and is *not* a reflection of the geography of the Western United States.

We applied algorithm 2 to this dataset and present in fig. 25 the distribution of local cutvertex radii.

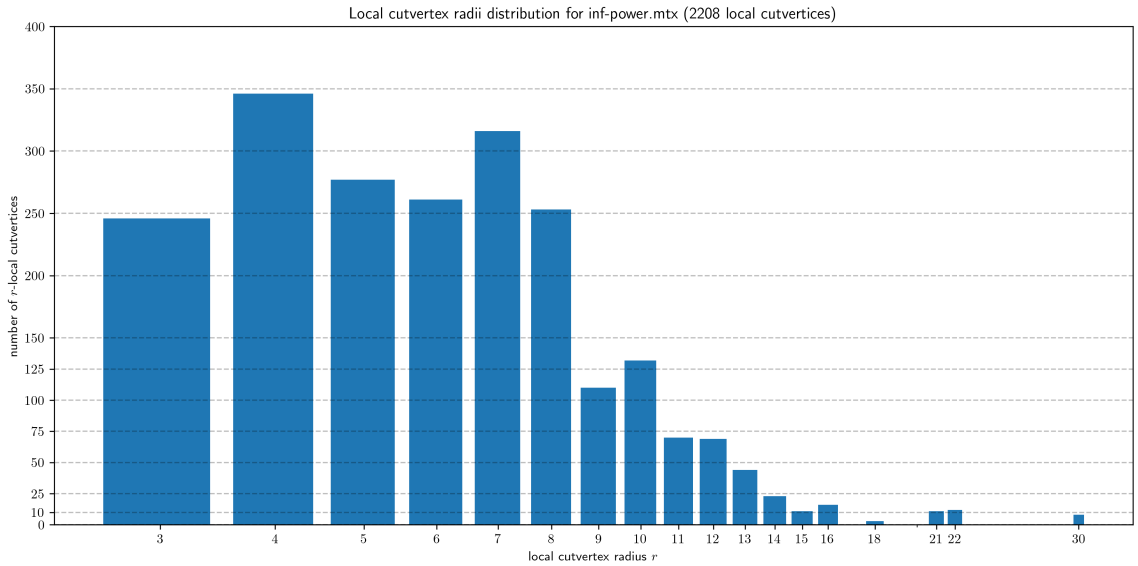

Figure 25: Distribution of local cutvertex radii in the infpower dataset

algorithm 2 has identified 2208 vertices to be local cutvertices in this dataset, which is over 44% of all vertices in the graph. However, we note that the largest local cutvertex radius is 30, which we hypothesise to be small relative to the number of vertices in the single connected component in the graph. We present fig. 26 for comparison, which is, for each connected component that has a local cutvertex in each graph in the combined Network_Data_MJS20, stackoverflow, and MRN datasets, a log plot of the size of the component as a function the largest radius of a local cutvertex in said component.

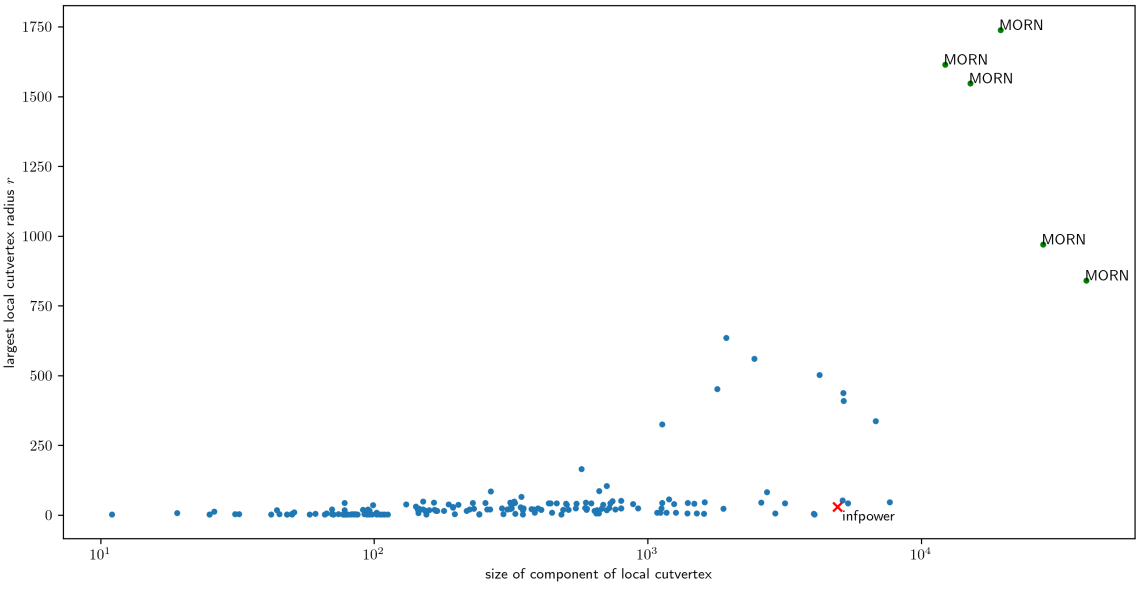

Figure 26: Sizes of the components of a local cutvertex and the largest local cutvertex radius across the Network_Data_MJS20, stackoverflow, and MRN datasets.

Save for a few local cutvertices in the MRN dataset annotated and highlighted in green, the infpower dataset possesses one of the smallest "largest local cutvertex radius" to "component size" ratios, being situated in the bottom right of fig. 26. This dataset

does not supply any information in regards to scale, so we cannot accurately determine or estimate how vital a vertex may be or what physical distances may depend on a vertex or are covered by an edge. However, we can interpret the aforementioned ratio as power failures in the network causing immediate fallout strain only locally rather than globally, which is a sought-after design criterion for a large continental power grid.

We now split the graph at its local cutvertices as per definition 3.1 and remove all split vertices from the resulting graph to obtain X. We've plotted the size of the connected components of X in fig. 27.

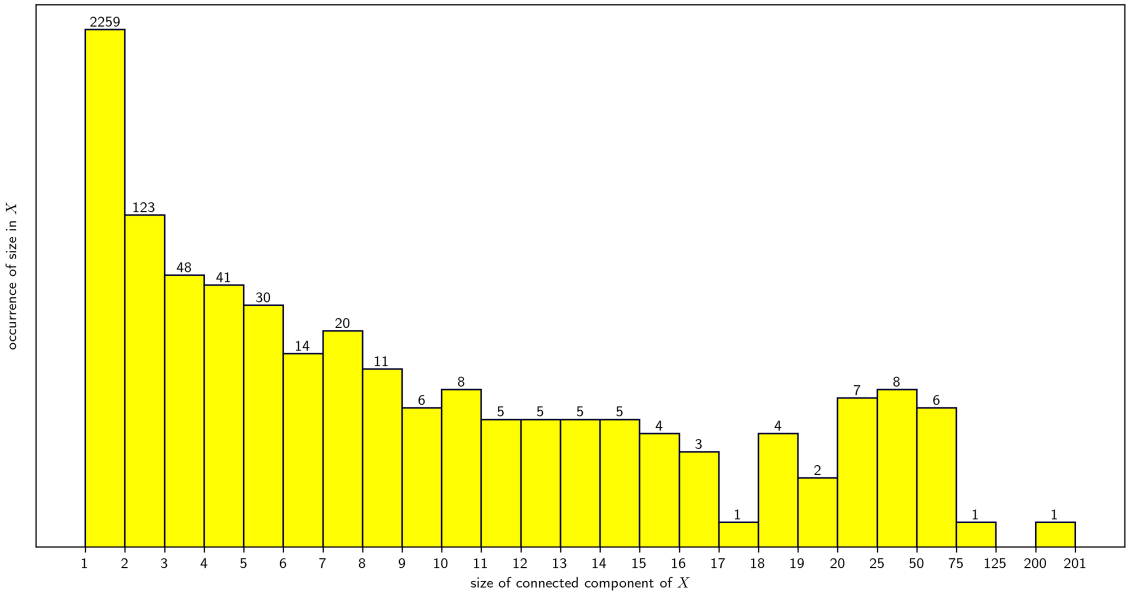

Figure 27: Distribution of the sizes of the connected components of X

Save for 16 components however this dataset has been decomposed down to components with 25 or fewer vertices. We then opted for the same isomorphism analysis, and found that all 48 components on 3 vertices are isomorphic to P^3. We've presented the isomorphisms of components on 4 and 5 vertices in fig. 28.

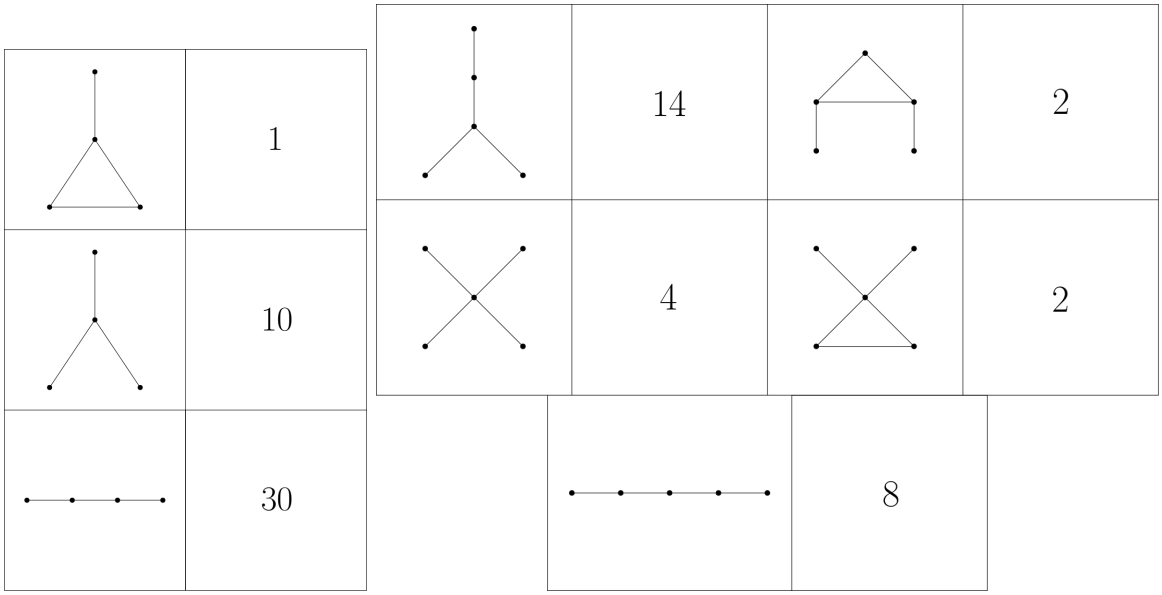

Figure 28: Distribution of isomorphisms of connected components of sizes 4 and 5 in X. X possess 41 components on 4 vertices and 30 components on 5 vertices.

Although this dataset possesses more varied components in regards to their isomorphims, it is interesting to note that in both this and the the MRN dataset, the most common isomorphic class of connected component on $3, 4, 5$ vertices that result from this operation is that of the path graphs P^3, P^4, P^5 respectively. The presence of components that are not isomorphic to path graphs in this dataset indicates relatively better redundancy in their real-world representatives, another sign of a well-designed infrastructure network.

5 Conclusion

Local cutvertices represent an exciting new area of graph theory that lend themselves well to analysing graphs corresponding to real-world networks since they capture the nuance around their fragility in spite their high global connectivity. In particular, they can be used together with elements of network theory to granularise network resiliency analysis, an important quality in strong network design.

This project made me aware of the challenging difficulties that arise when implementing mathematical structures and ideas in computer code. In particular the testing and debugging required to verify correct functionality, and the efficiency considerations that come into play. I've also learned again that small, incremental progress is the best approach to take when it comes to tackling large tasks.

In regards to the future of this project, two main areas of improvement should be considered. The first is porting this project's critical codebase to a faster, albeit less flexible, graph theory library, as well as its visualisation suite to a more powerful one, in order to handle larger graphs. Currently this project uses NetworkX and its visualisation suite, which relies on matplotlib. NetworkX provides excellent facilities and allows for rapid debugging, however it has at times felt lacking in terms of speed. I've found its visualisation suite, which I once more state to be straightforward to use, to be better suited for graphs with fewer than 5000 vertices as the graph drawing algorithms are rather slow, slowing down the aforementioned prototyping cycle. This project regularly serialised results for later retrieval to aid with this lack of speed. Candidates for this improvement are retworkx (a port of NetworkX to Rust) or igraph, two libraries with Python interfaces to minimise the amount of work involved when porting, and GraphViz, a graph drawing tool well-equipped to handle large graphs.

The second is implementing a tree-like decomposition of a graph along its local cutvertices. Splitting at multiple local cutvertices and subsequently removing the split vertices isolates the local cutvertices and creates new components. By contracting the new components to a single vertex we can obtain a simplified view of the original graph, and we could potentially repeat this process as many times as we please. This is challenging to implement, let alone efficiently, but would be a tremendous addition to this project.

6 Acknowledgements

I would like to thank my supervisors, Johannes Carmesin and Samuel Johnson, for their help and patience in implementing this project, Jan Kurkofka, for their invaluable time and help debugging and discussing improvements to this project, Emily Nevinson, Will Turner, and Tsvetomir Mihaylov for their attention, insights, and input to the development of this project during our lunch meetings.

I would also like to thank the GitHub contributors to the open source software libraries that made this project possible, as well as the Network Repository website [39].

Bibliography

[1] networkx.algorithms.shortest_paths.unweighted — NetworkX 2.8 documentation, 2010.

[2] Luis G. Abarca-Arenas and Robert E. Ulanowicz. The effects of taxonomic aggregation on network analysis. *Ecological Modelling*, 149(3):285–296, 2002. Publisher: Elsevier.

[3] Luca Albergante, J. Julian Blow, and Timothy J. Newman. Buffered Qualitative Stability explains the robustness and evolvability of transcriptional networks. *Elife*, 3:e02863, 2014. Publisher: eLife Sciences Publications Limited.

[4] Jordi Bascompte, Carlos J. Melián, and Enric Sala. Interaction strength combinations and the overfishing of a marine food web. *Proceedings of the National Academy of Sciences*, 102(15):5443–5447, 2005. Publisher: National Acad Sciences.

[5] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. House of Graphs: A database of interesting graphs. *Discrete Applied Mathematics*, 161(1):311–314, January 2013.

[6] Johannes Carmesin. Local 2-separators. *arXiv:2008.03032 [math]*, August 2020. arXiv: 2008.03032.

[7] Robert R. Christian and Joseph J. Luczkovich. Organizing and understanding a winter's seagrass foodweb network through effective trophic levels. *Ecological modelling*, 117(1):99–124, 1999. Publisher: Elsevier.

[8] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*, pages 149–159, 2001.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Section 24.3. In *Introduction to Algorithms*. MIT press, 2009.

[10] Maria C. Costanzo, Matthew E. Crawford, Jodi E. Hirschman, Janice E. Kranz, Philip Olsen, Laura S. Robertson, Marek S. Skrzypek, Burkhard R. Braun, Kelley Lennon Hopkins, and Pinar Kondu. YPD™, PombePD™ and WormPD™: model organism volumes of the BioKnowledge™ Library, an integrated resource for protein information. *Nucleic Acids Research*, 29(1):75–79, 2001. Publisher: Oxford University Press.

[11] Reinhard Diestel. *Graph Theory*. Number 136 in Graduate Texts in Mathematics. Springer, 5th edition, 2017.

[12] Anna Eklöf, Ute Jacob, Jason Kopp, Jordi Bosch, Rocío Castro-Urgal, Natacha P. Chacoff, Bo Dalsgaard, Claudio de Sassi, Mauro Galetti, and Paulo R. Guimaraes. The dimensionality of ecological networks. *Ecology letters*, 16(5):577–583, 2013. Publisher: Wiley Online Library.

[13] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, February 2010. arXiv:0906.0612 [cond-mat, physics:physics, q-bio].

[14] Edgardo Galán-Vásquez, Beatriz Luna, and Agustino Martínez-Antonio. The regulatory network of Pseudomonas aeruginosa. *Microbial informatics and experimentation*, 1(1):1–11, 2011. Publisher: BioMed Central.

[15] Mark B. Gerstein, Anshul Kundaje, Manoj Hariharan, Stephen G. Landt, Koon-Kiu Yan, Chao Cheng, Xinmeng Jasmine Mu, Ekta Khurana, Joel Rozowsky, and Roger Alexander. Architecture of the human regulatory network derived from ENCODE data. *Nature*, 489(7414):91–100, 2012. Publisher: Nature Publishing Group.

[16] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, June 2002.

[17] Lloyd Goldwasser and Jonathan Roughgarden. Construction and Analysis of a Large Caribbean Food Web: Ecological Archives E074-001. *Ecology*, 74(4):1216–1233, 1993. Publisher: Wiley Online Library.

[18] Aric Hagberg. networkx Graph Connectivity, October 2013.

[19] Aric Hagberg, Daniel Schult, and Pieter Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11–15. 2008.

[20] Christopher T. Harbison, D. Benjamin Gordon, Tong Ihn Lee, Nicola J. Rinaldi, Kenzie D. Macisaac, Timothy W. Danford, Nancy M. Hannett, Jean-Bosco Tagne, David B. Reynolds, and Jane Yoo. Transcriptional regulatory code of a eukaryotic genome. *Nature*, 431(7004):99–104, 2004. Publisher: Nature Publishing Group.

[21] Karl Havens. Scale and structure in natural food webs. *Science*, 257(5073):1107–1109, 1992. Publisher: American Association for the Advancement of Science.

[22] Mark Huxham, S. Beaney, and Dave Raffaelli. Do parasites reduce the chances of triangulation in a real food web? *Oikos*, pages 284–300, 1996. Publisher: JSTOR.

[23] Ute Jacob, Aaron Thierry, Ulrich Brose, Wolf E. Arntz, Sofia Berg, Thomas Brey, Ingo Fetzer, Tomas Jonsson, Katja Mintenbeck, and Christian Möllmann. The role of body size in complex food webs: A cold case. In *Advances in ecological research*, volume 45, pages 181–223. Elsevier, 2011.

[24] Hawoong Jeong, Bálint Tombor, Réka Albert, Zoltan N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000. Publisher: Nature Publishing Group.

[25] Samuel Johnson and Nick S. Jones. Looplessness in networks is linked to trophic coherence. *Proceedings of the National Academy of Sciences*, 114(22):5618–5623, May 2017. arXiv: 1505.07332.

[26] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1):2–es, March 2007.

[27] Neo D. Martinez. Artifacts or attributes? Effects of resolution on the Little Rock Lake food web. *Ecological monographs*, 61(4):367–392, 1991. Publisher: Wiley Online Library.

[28] Neo D. Martinez, Bradford A. Hawkins, Hassan Ali Dawah, and Brian P. Feifarek. Effects of sampling effort on characterization of food-web structure. *Ecology*, 80(3):1044–1055, 1999. Publisher: Wiley Online Library.

[29] J. Memmott, N. D. Martinez, and J. E. Cohen. Predators, parasitoids and pathogens: species richness, trophic generality and body sizes in a natural food web. *Journal of Animal Ecology*, 69(1):1–15, 2000. Publisher: Wiley Online Library.

[30] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, 2004. Publisher: American Association for the Advancement of Science.

[31] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004. Publisher: APS.

[32] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. Exploratory Social Network Analysis with Pajek: Revised and Expanded Edition for Updated Software, July 2018. ISBN: 9781108565691 9781108474146 9781108462273 Publisher: Cambridge University Press.

[33] Gary A. Polis. Complex trophic interactions in deserts: an empirical critique of food-web theory. *The American Naturalist*, 138(1):123–155, 1991. Publisher: University of Chicago Press.

[34] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks (long version). Technical Report arXiv:physics/0512106, arXiv, December 2005. arXiv:physics/0512106 type: article.

[35] David Quarmby and Phil Carey. A Major Road Network for England: A Rees Jeffreys Road Fund Study. Study, October 2016.

[36] Douglas P. Reagan and Robert B. Waide. *The food web of a tropical rain forest*. University of Chicago Press, 1996.

[37] Jens O. Riede, Ulrich Brose, Bo Ebenman, Ute Jacob, Ross Thompson, Colin R. Townsend, and Tomas Jonsson. Stepping in Elton's footprints: a general scaling model for body masses and trophic levels across ecosystems. *Ecology letters*, 14(2):169–178, 2011. Publisher: Wiley Online Library.

[38] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *arXiv:cs/0209028*, September 2002. arXiv: cs/0209028.

[39] Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[40] Heladia Salgado, Martin Peralta-Gil, Socorro Gama-Castro, Alberto Santos-Zavaleta, Luis Muniz-Rascado, Jair S. García-Sotelo, Verena Weiss, Hilda Solano-Lira, Irma Martínez-Flores, and Alejandra Medina-Rivera. RegulonDB v8. 0: omics data sets, evolutionary conservation, regulatory phrases, cross-validated gold standards and more. *Nucleic acids research*, 41(D1):D203–D213, 2013. Publisher: Oxford University Press.

[41] Joaquín Sanz, Jorge Navarro, Ainhoa Arbués, Carlos Martín, Pedro C. Marijuán, and Yamir Moreno. The transcriptional regulatory network of Mycobacterium tuberculosis. *PloS one*, 6(7):e22178, 2011. Publisher: Public Library of Science San Francisco, USA.

[42] Theodore Seuss. *Green Eggs and Ham*. Random House, 1960.

[43] Julia Silge. Stack Overflow Tag Network, September 2017.

[44] Denis Thieffry, Araceli M. Huerta, Ernesto Pérez-Rueda, and Julio Collado-Vides. From specific gene regulation to genomic networks: a global analysis of transcriptional regulation in Escherichia coli. *Bioessays*, 20(5):433–440, 1998. Publisher: Wiley Online Library.

[45] Ross M. Thompson and Angus R. Mcintosh. Disturbance, resource supply, and food-web architecture in streams. *Ecology Letters*, 1:200ą209, 1998.

[46] Ross M. Thompson and C. R. Townsend. Impacts on stream food webs of native and exotic forest: an intercontinental comparison. *Ecology*, 84(1):145–161, 2003. Publisher: Wiley Online Library.

[47] Ross M. Thompson and C. R. Townsend. Energy availability, spatial heterogeneity and ecosystem size predict food-web structure in streams. *Oikos*, 108(1):137–148, 2005. Publisher: Wiley Online Library.

[48] Department for Transport. Major Road Network, December 2020.

[49] Robert E. Ulanowicz and Daniel Baird. Nutrient controls on ecosystem dynamics: the Chesapeake mesohaline community. *Journal of Marine Systems*, 19(1-3):159–172, 1999. Publisher: Elsevier.

[50] Philip H. Warren. Spatial and temporal variation in the structure of a freshwater food web. *Oikos*, pages 299–311, 1989. Publisher: JSTOR.

[51] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998. Number: 6684 Publisher: Nature Publishing Group.

[52] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 314(1165):1–340, November 1986.

[53] Peter Yodzis. Local trophodynamics and the interaction of marine mammals and fisheries in the Benguela ecosystem. *Journal of Animal Ecology*, 67(4):635–658, 1998. Publisher: Wiley Online Library.

Appendix A

Sources for the Network_Data_MJS20 dataset

The Network_Data_MJS20 dataset comprises 74 graphs, compiled by Samuel Johnson from the following various sources.

Food web	Reference	Food web	Reference
Benguela Current	[53]	Berwick Stream	[46] [47] [45]
Blackrock Stream	[46] [47] [45]	Bridge Brook Lake	[21]
Broad Stream	[46] [47] [45]	Canton Creek	[45]
Cayman Islands	[4]	Catlins Stream	[46] [47] [45]
Chesapeake Bay	[49] [2]	Coachella Valley	[33]
Coweeta 1	[46] [47] [45]	Coweeta 17	[46] [47] [45]
Dempsters (Au)	[46] [47] [45]	Dempsters (Sp)	[46] [47] [45]
Dempsters (Su)	[46] [47] [45]	El Verde Rainforest	[36]
German Stream	[46] [47] [45]	Healy Stream	[46] [47] [45]
Kyeburn Stream	[46] [47] [45]	LilKyeburn Stream	[46] [47] [45]
Little Rock Lake	[27]	Lough Hyne	[37] [12]
Martins Stream	[46] [47] [45]	Narrowdale Stream	[46] [47] [45]
NE Shelf	[29]	North Col Stream	[46] [47] [45]
Powder Stream	[46] [47] [45]	Scotch Broom	[29]
Skipwith Pond	[50]	St Marks Estuary	[7]
St Martin Island	[17]	Stony Stream	[45]
Stony Stream 2	[46] [47] [45]	Sutton (Au)	[46] [47] [45]
Sutton (Sp)	[46] [47] [45]	Sutton (Su)	[46] [47] [45]
Troy Stream	[46] [47] [45]	UK Grassland	[28]
Venlaw Stream	[46] [47] [45]	Weddel Sea	[23]
Ythan Estuary	[22]		

Gene regulatory network	Reference
Human (healthy)	[15] [3]
Human (cancer)	[15] [3]
E. coli (Salgado)	[40] [3]
E. coli (Thieffry)	[44] [30]
S. cerevisiae (Harbison)	[20] [3]
S. cerevisiae (Costanzo)	[10] [30]
P. aeruginosa	[14] [3]
M. tuberculosis	[41] [3]

Metabolic network	Reference
A. fulgidus	[24]
M. thermoatuotrophicum	[24]
M. jannaschii	[24]
C. pneumoniae	[24]
C. trachomatis	[24]
S. cerevisae (yeast)	[24]
C. elegans	[24]

Network (miscellaneous)	Reference
Neural (C. elegans)	[52] [51]
P2P (Gnutella 2008)	[26] [38]
Trade (manufactured goods)	[32]
Trade (minerals)	[32]
Words from *Green Eggs and Ham*	[42]

Appendix B

Stack Overflow Communities

From the stackoverflow dataset, we've obtained the subgraphs induced on each community and plotted them here for convenience. Note that the size of a vertex is proportional to its popularity in the dataset, and the thickness of an edge proportional to the correlation coefficient between its vertices as calculated by the Walktrap algorithm.

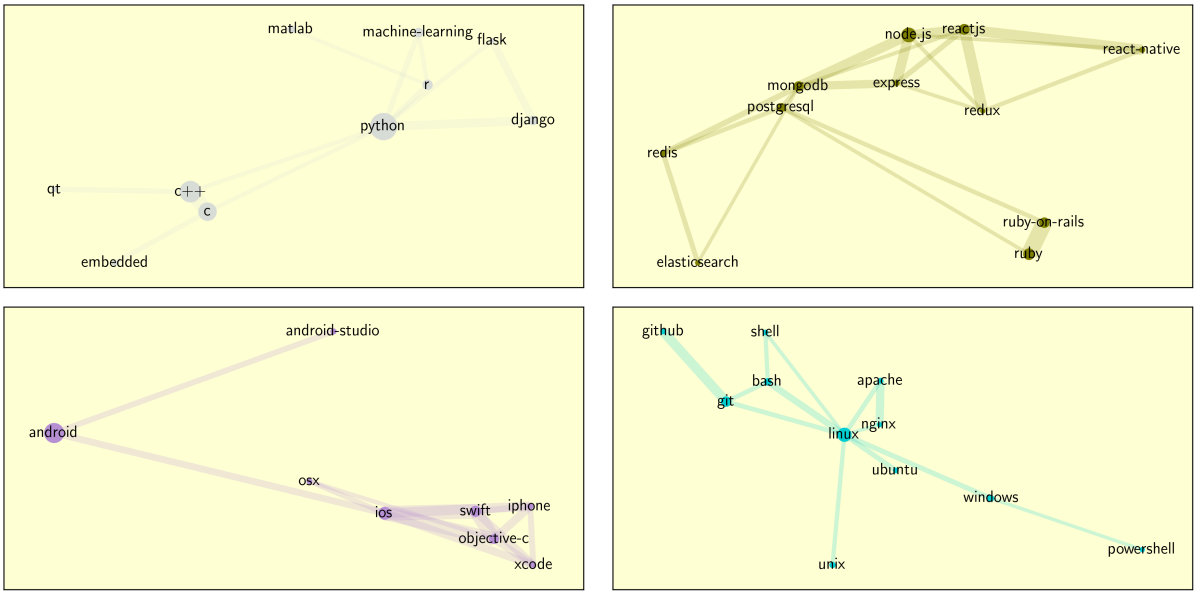

Figure B.1: Communities 1, 3, 4, and 5

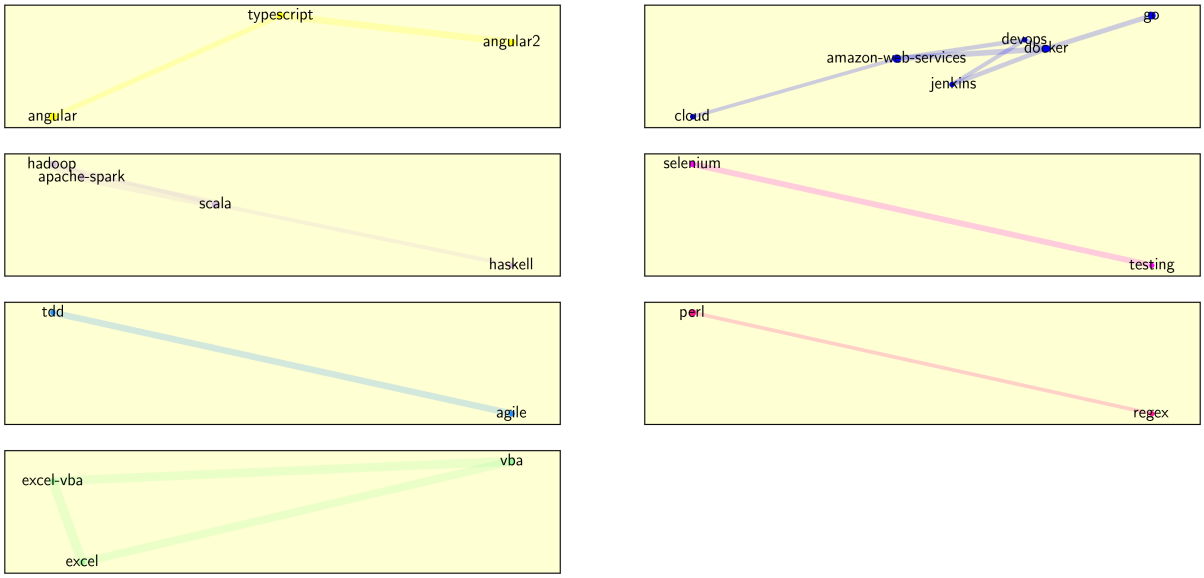

Figure B.2: Communities 7, 9, 10, 11, 12, 13, and 14

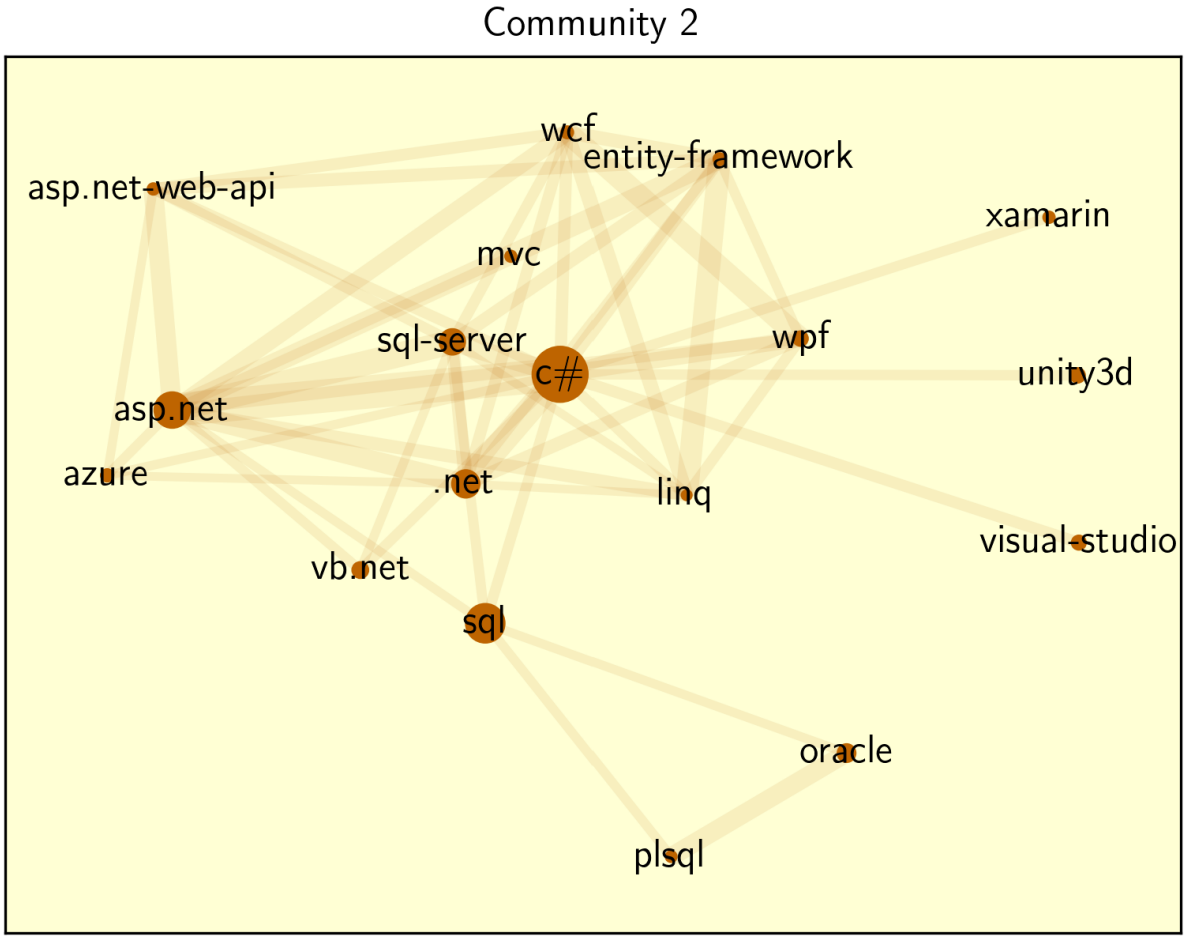

Figure B.3: Community 2

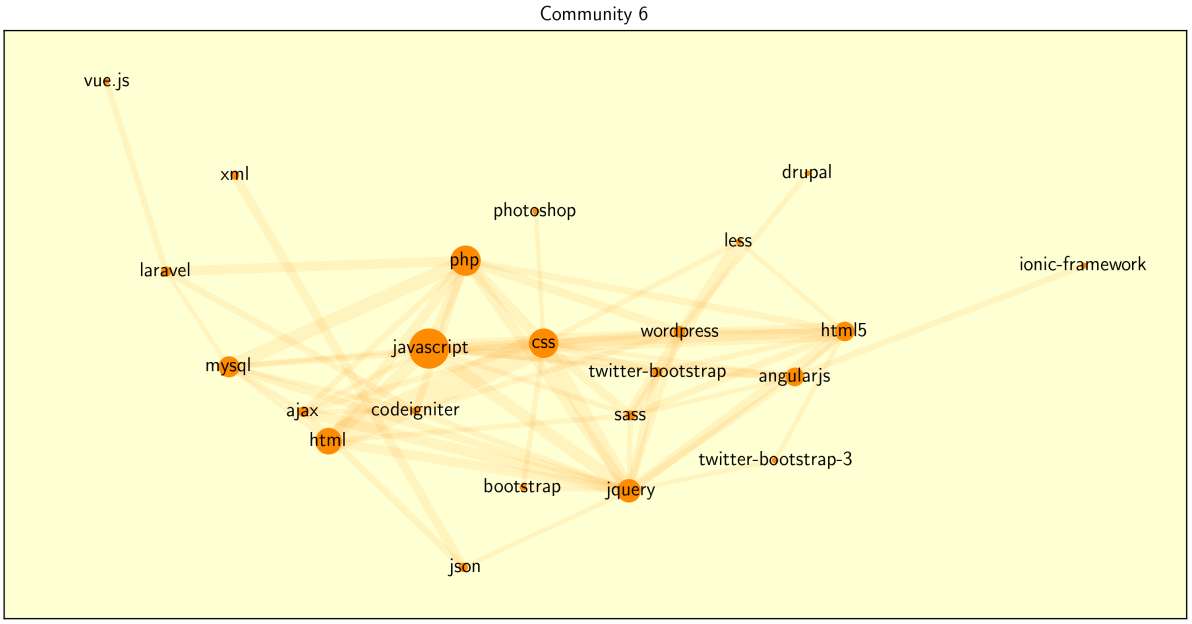

Figure B.4: Community 6

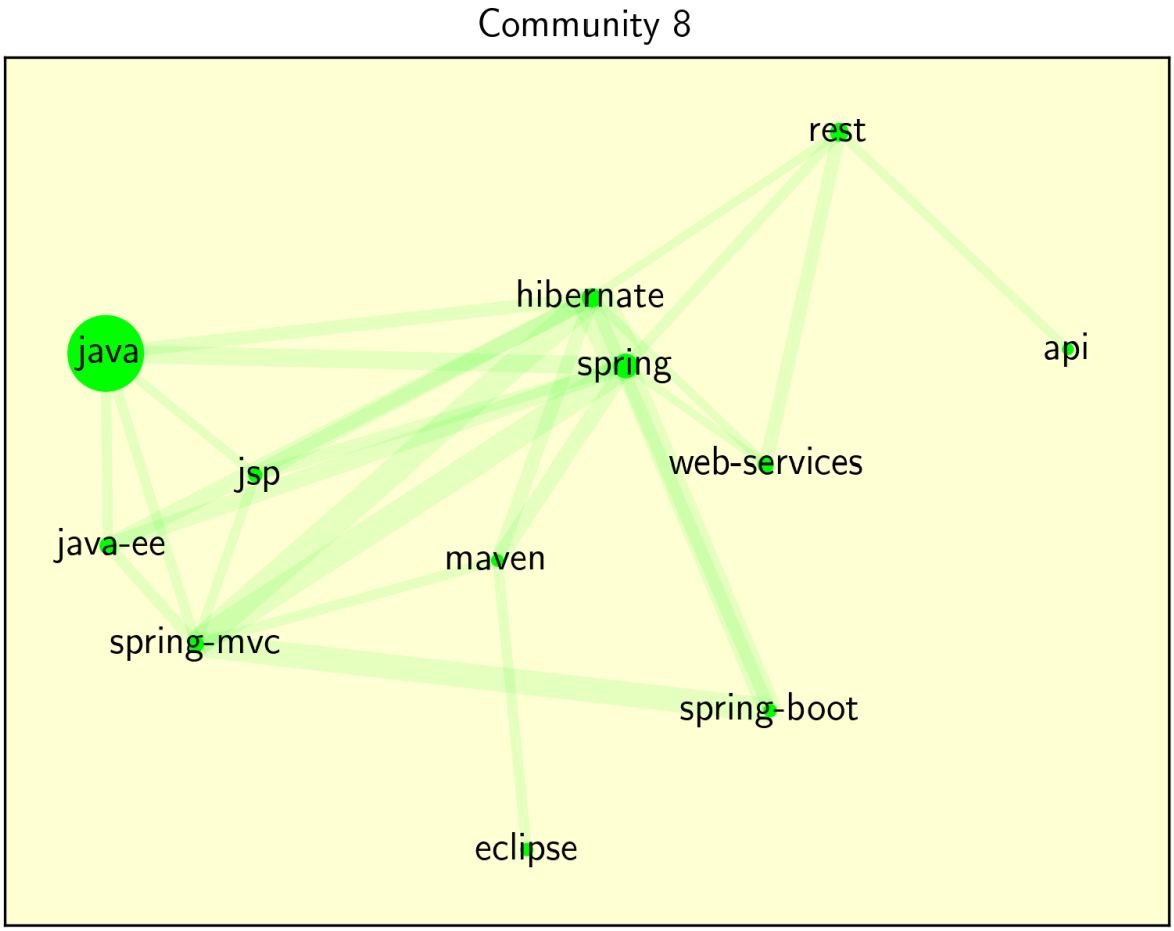

Figure B.5: Community 8

Appendix C

Stack Overflow Local Cutvertices Punctured Balls Components

For each r-local cutvertex v in the stackoverflow dataset, we've plotted below the ball of radius $\frac{r}{2}$ around v, split at v. The local cutvertex in each plot is drawn in red. The two other colours used reflect the components of the punctured ball of radius $\frac{r}{2}$ around the local cutvertex v to which a vertex or edge belongs to. The thickness of an edge is proportional to the correlation coefficient between tags, and the size of a vertex representing a tag is proportional to its popularity in the mined data. Finally, the cross-shaped vertices are the split vertices that arise from splitting at the local cutvertex in question.

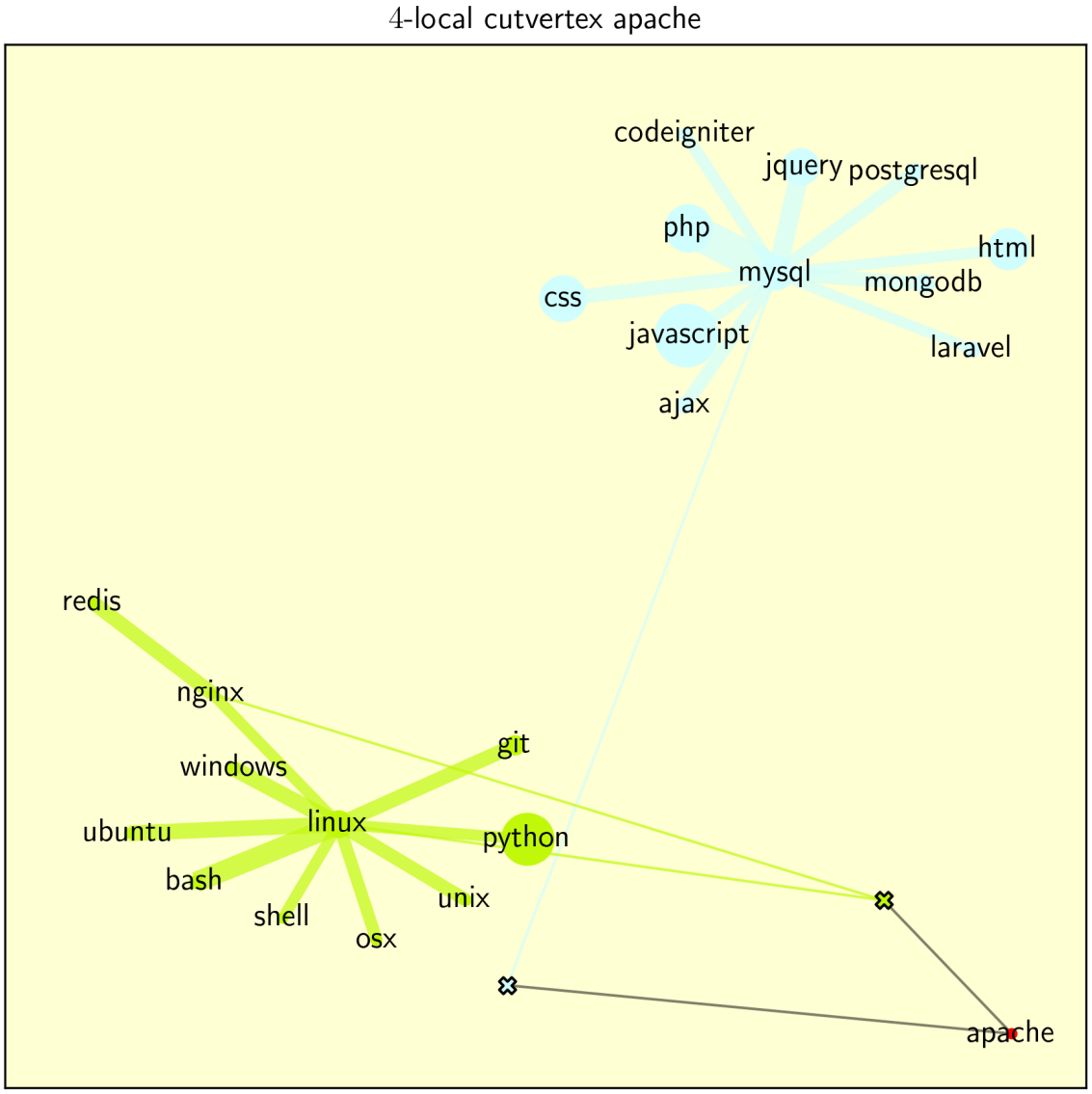

Figure C.1: Components of the punctured ball around the local cutvertex apache

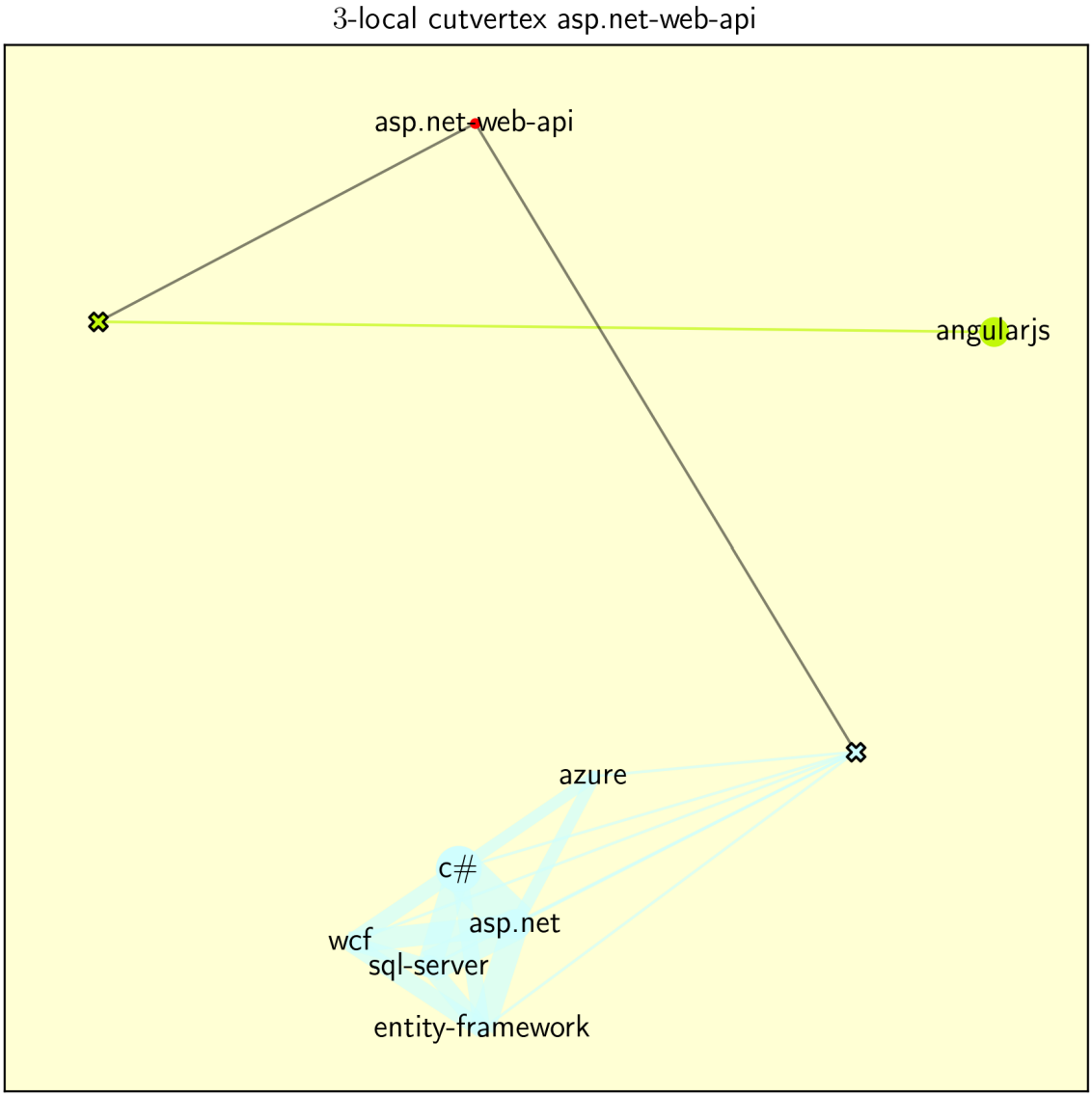

Figure C.2: Components of the punctured ball around the local cutvertex asp.net-web-api

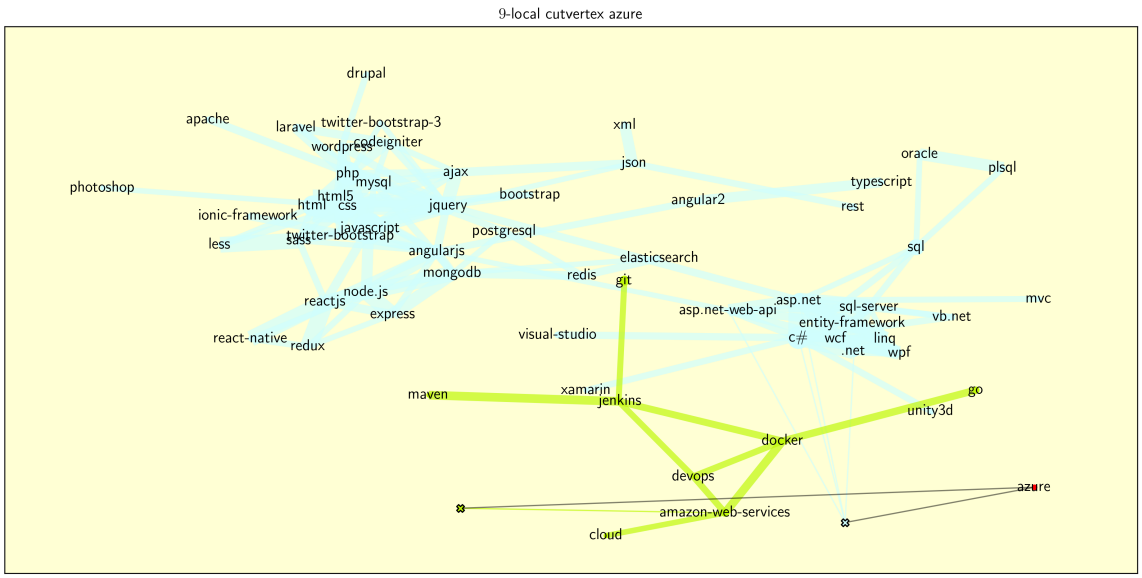

Figure C.3: Components of the punctured ball around the local cutvertex azure

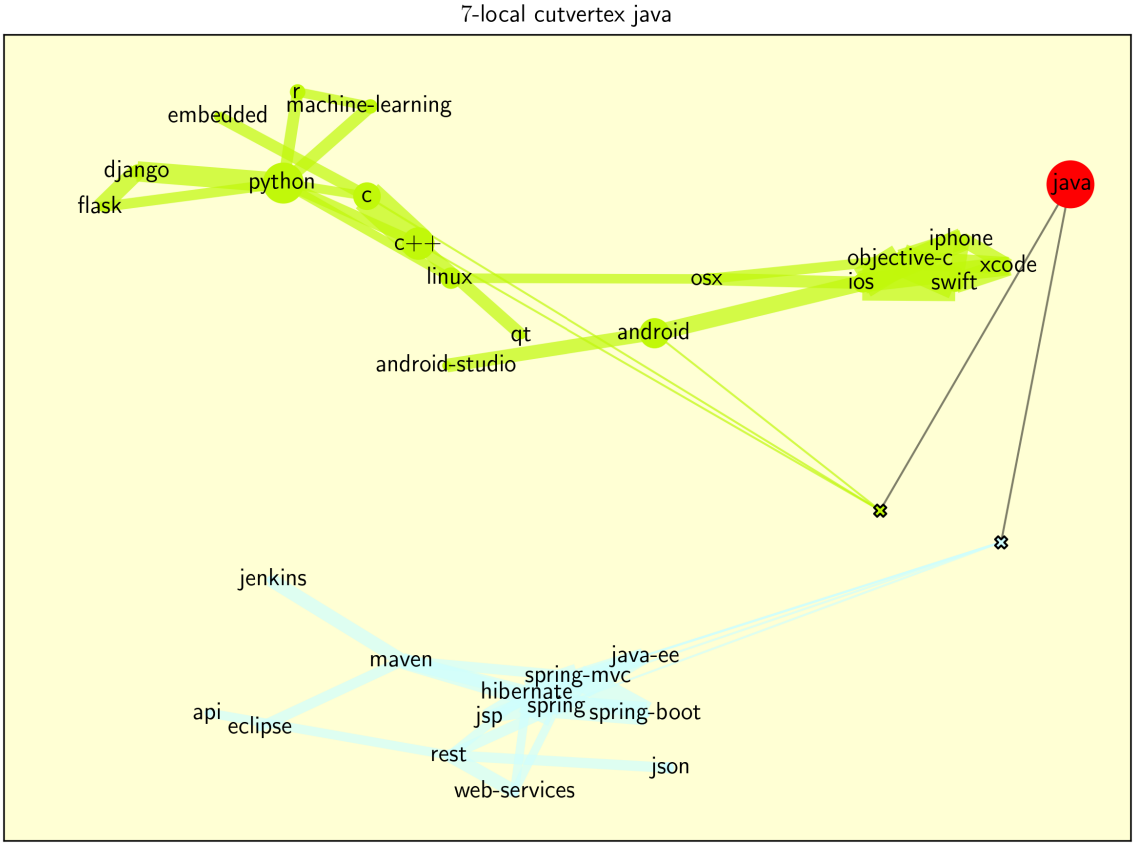

Figure C.4: Components of the punctured ball around the local cutvertex java

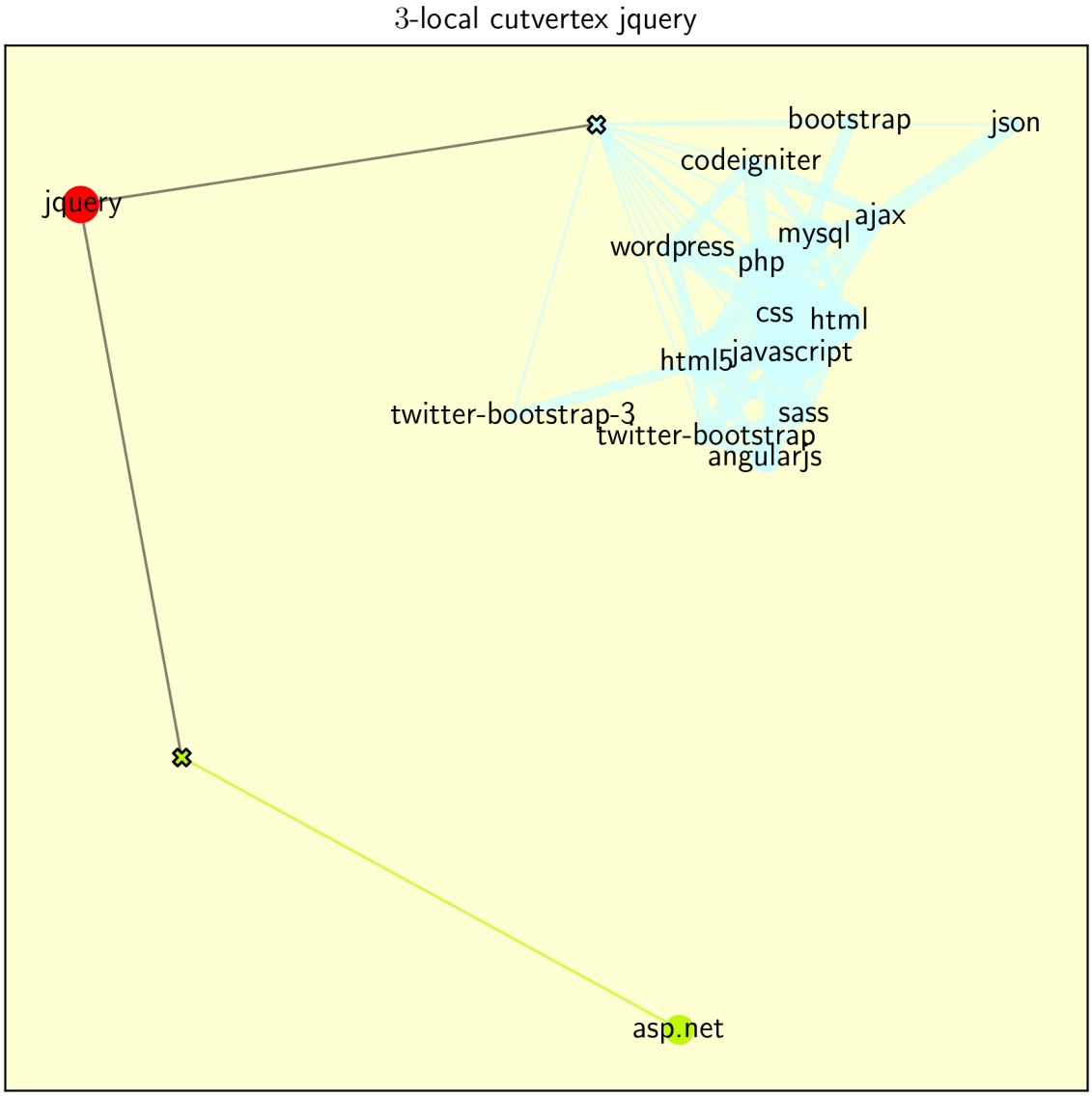

Figure C.5: Components of the punctured ball around the local cutvertex jquery

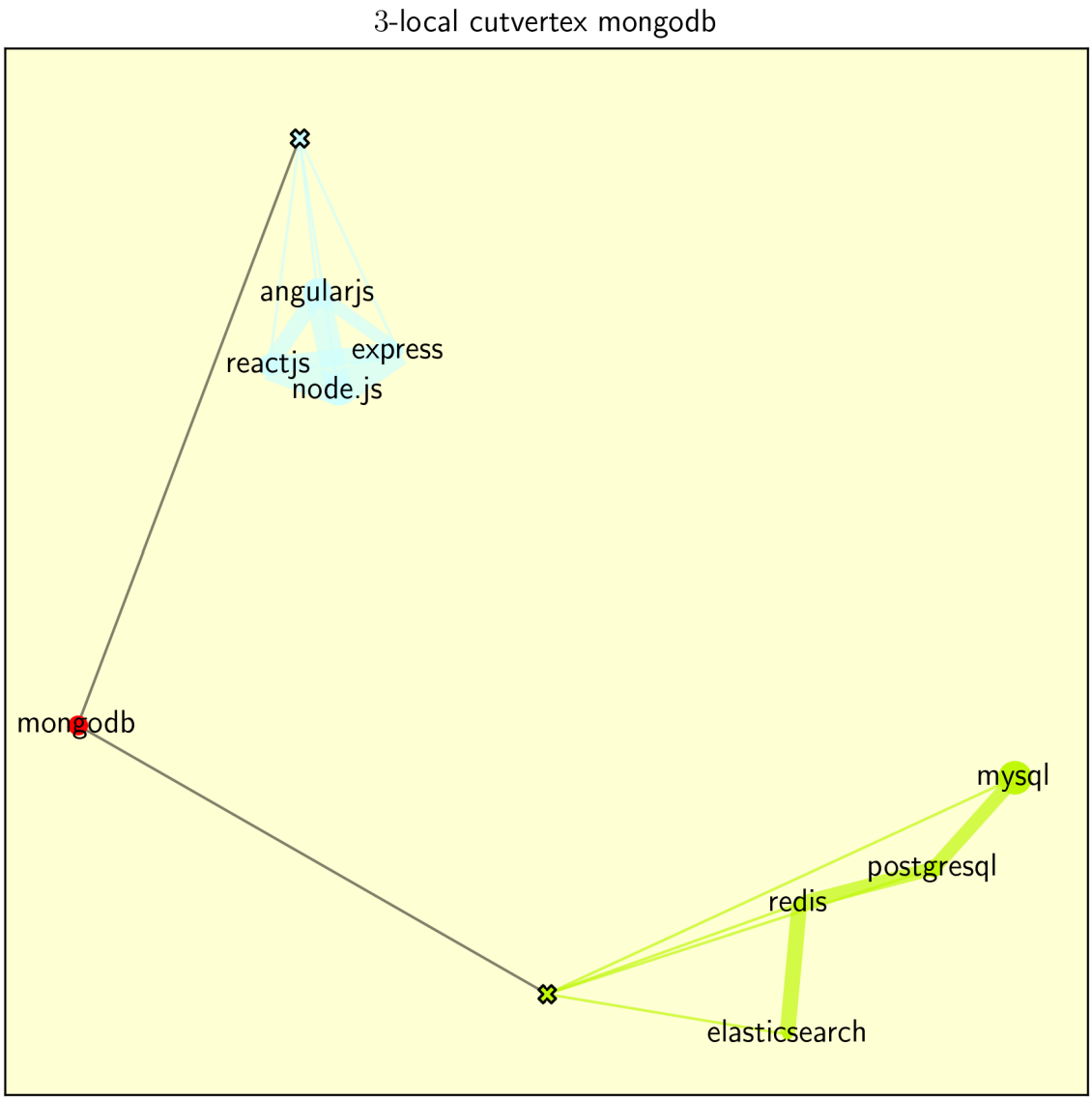

Figure C.6: Components of the punctured ball around the local cutvertex mongodb

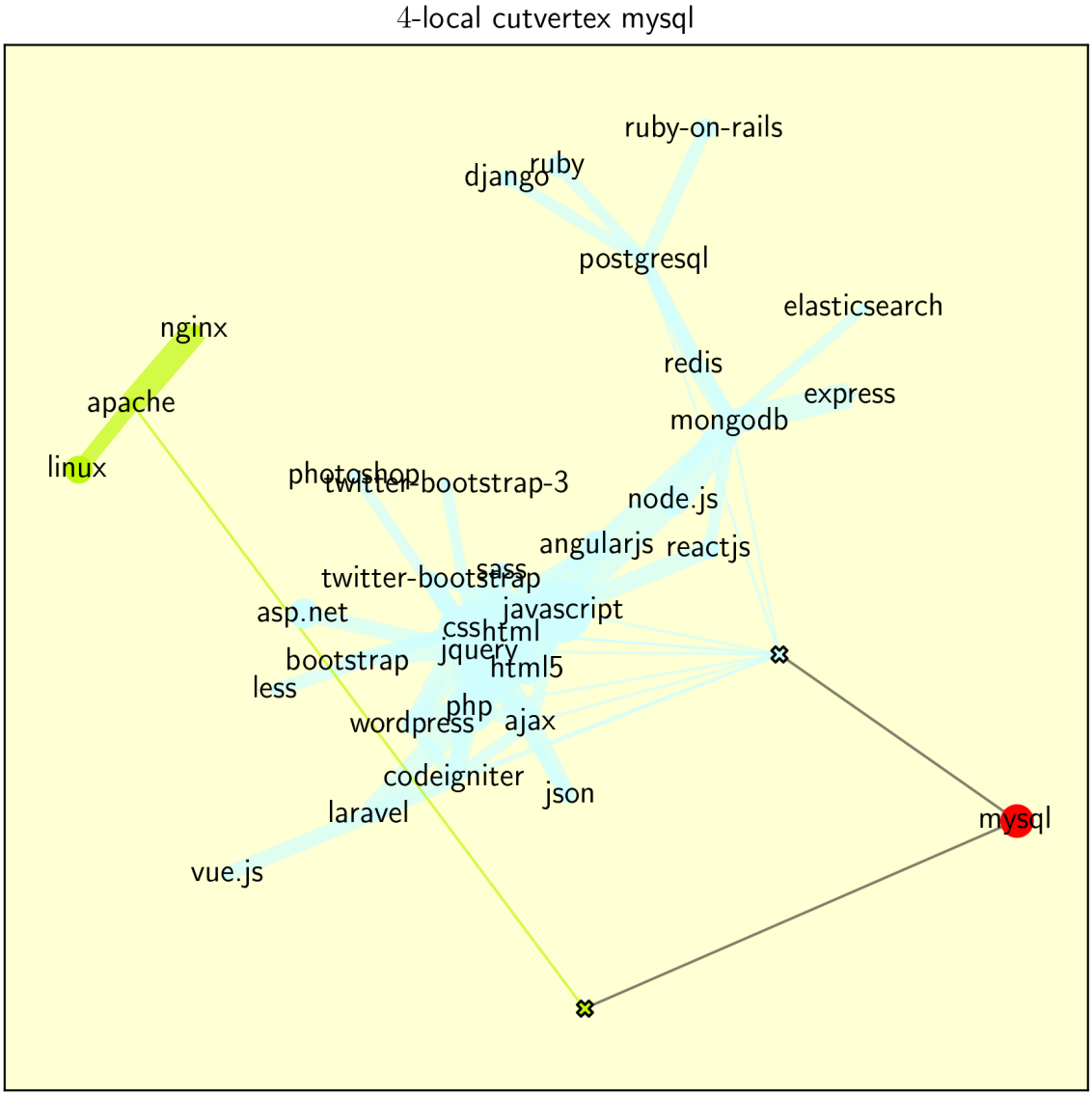

Figure C.7: Components of the punctured ball around the local cutvertex mysql

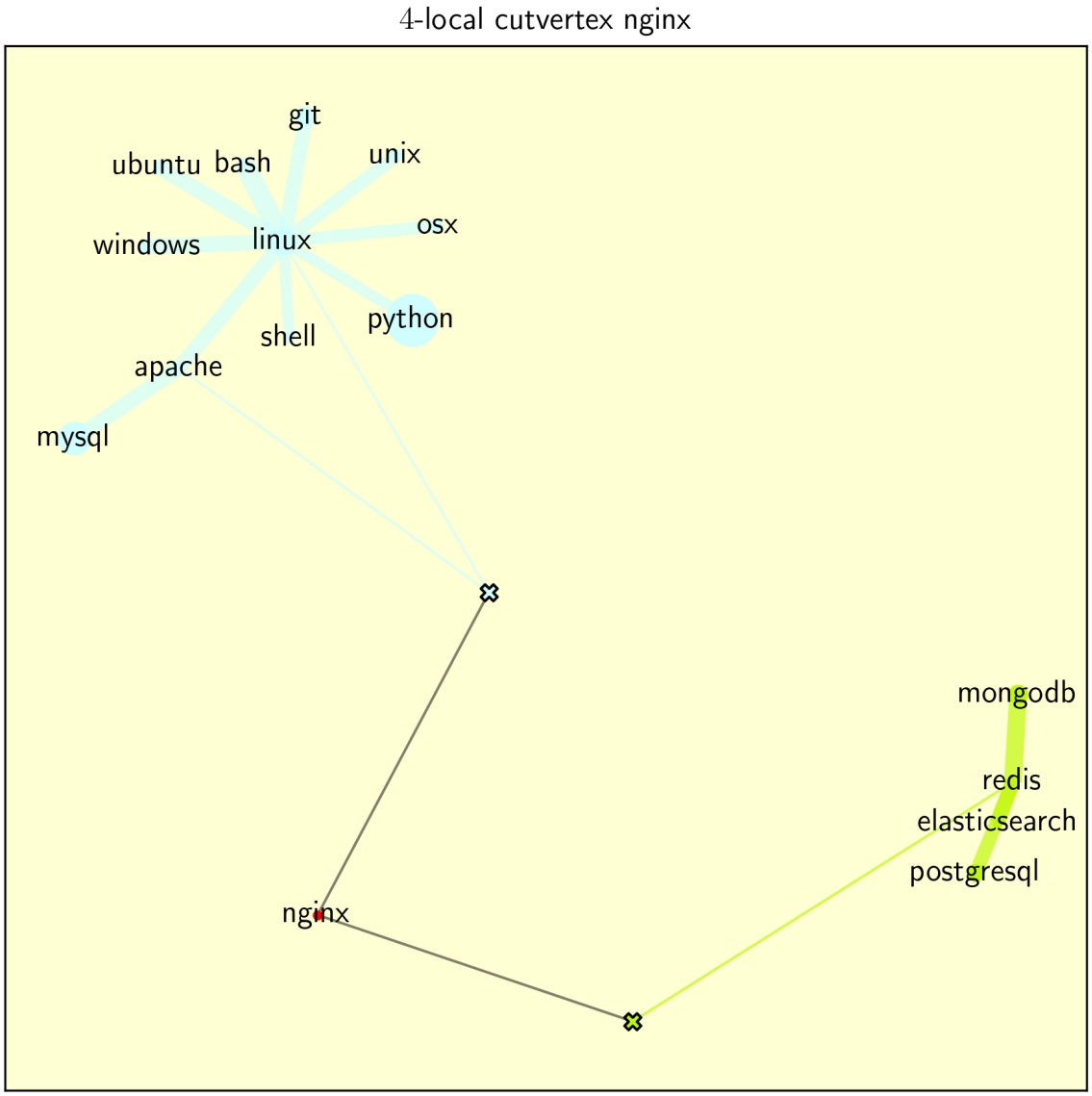

Figure C.8: Components of the punctured ball around the local cutvertex nginx

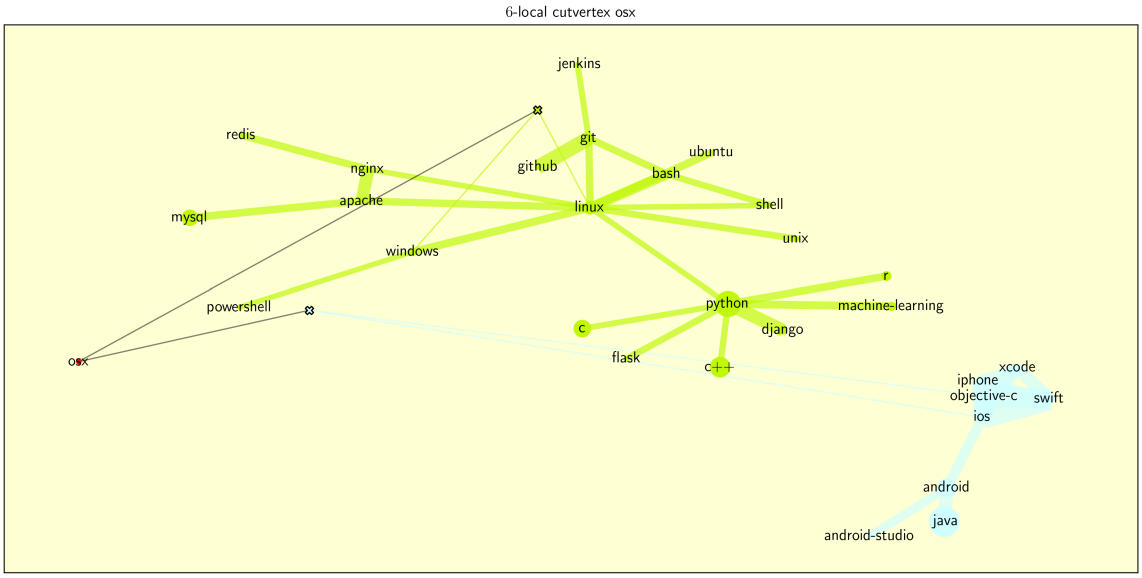

Figure C.9: Components of the punctured ball around the local cutvertex osx

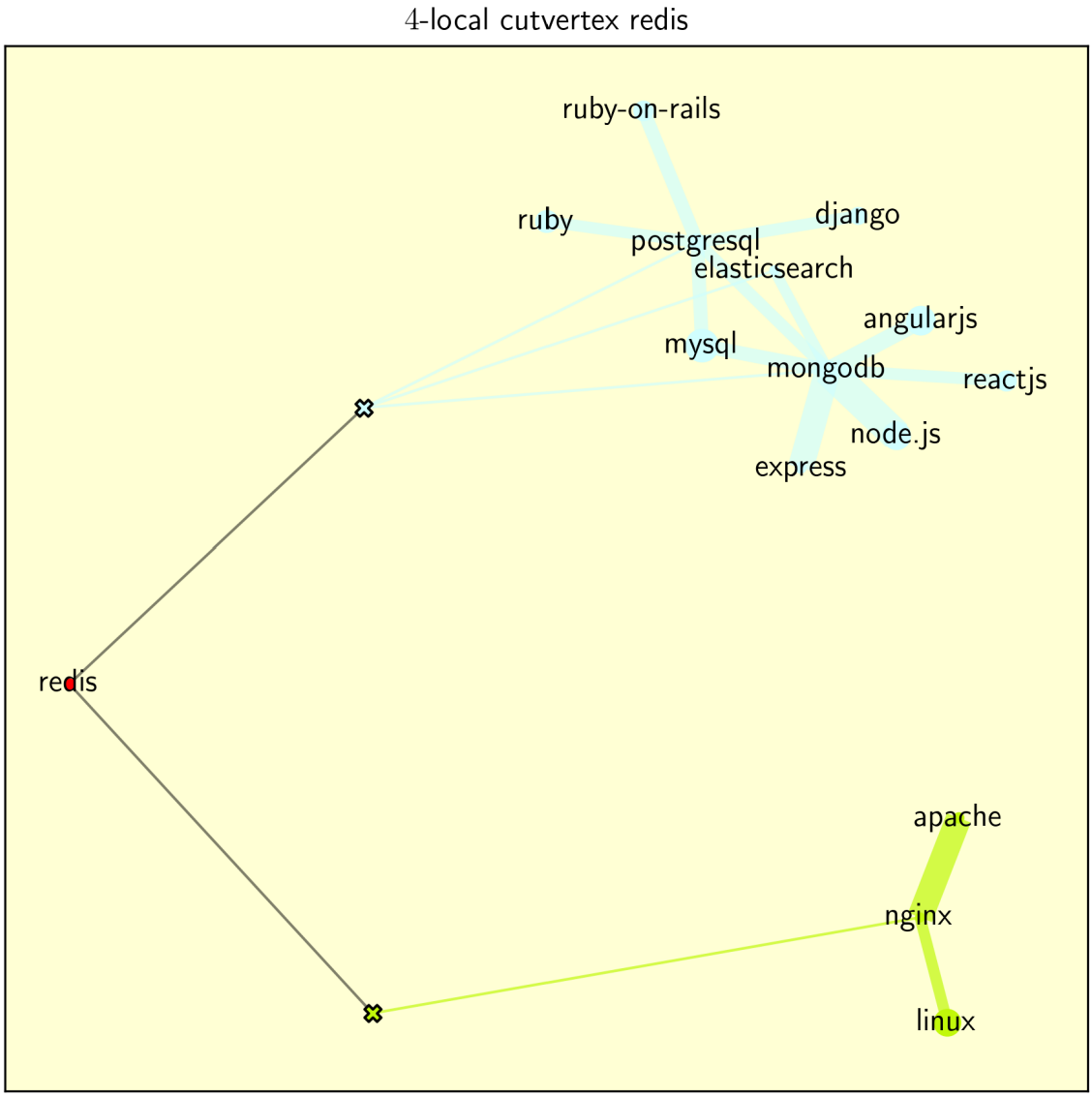

Figure C.10: Components of the punctured ball around the local cutvertex redis