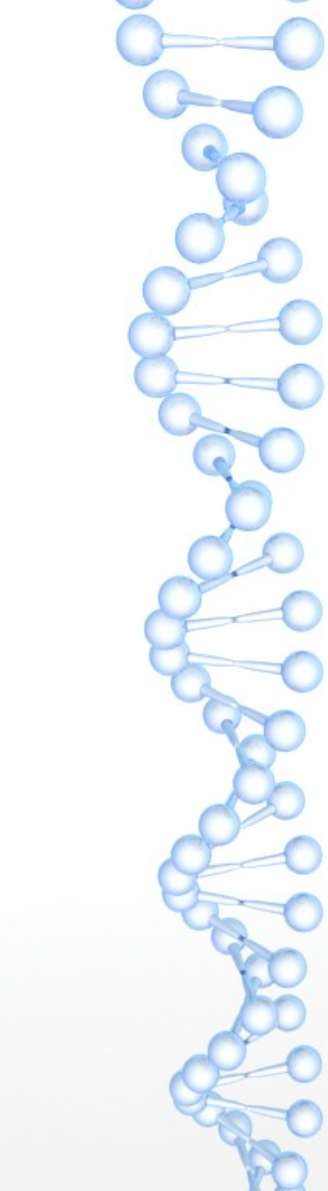


BCB330
Final Presentation

Bruno Pereira
1002407086

Max Expression

Alex Sullivan's work

- PHP API allows querying for max/min values
- Relies on JSON file containing correspondences:
  - Sample name to sample ID
  - Control sample to experimental sample

- JSON generated from XML files which contain extra information

We want to use this data at the BAR


```
{
  "wasSuccessful": true,
  "error": null,
  "statusCode": 200,
  "locus": "At4g11945",
  "species": "Arabidopsis",
  "sources": [
    {
      "maxAverageValue": {
        "ExpressionAverage": 0.390096,
        "StandardDeviation": 0.390096,
        "InductionValue": 0,
        "ReductionValue": 0,
        "ControlValue": 0,
        "FoldChange": 0,
        "SampleSize": 2,
        "Compendium": "Klepikova",
        "Sample": "S_H",
        "data_bot_id": "'SRR3581336', 'SRR3581740'",
        "ControlSample": "Med_CTRL",
        "control_data_bot_id": "'Med_CTRL'",
        "db": "klepikova"
      },
      "topMaxAverageValue": { … }, // 13 items
      "maxInductionValue": { … }, // 13 items
      "topMaxInductionValue": { … }, // 13 items
      "maxReductionValue": { … }, // 13 items
      "topMaxReductionValue": { … }, // 13 items
      "maxFoldChange": { … }, // 13 items
      "topMaxFoldChange": { … }, // 13 items
      "minAverageValue": { … }, // 13 items
      "topMinAverageValue": { … }, // 13 items
      "minFoldChange": { … }, // 13 items
      "topMinFoldChange": { … } // 13 items
    }
  ]
}
```

Max Expression

Problems with using this API:

- Each time we query it the API it calculates everything from scratch
- This involves many queries to the underlying SQL database

Solution:

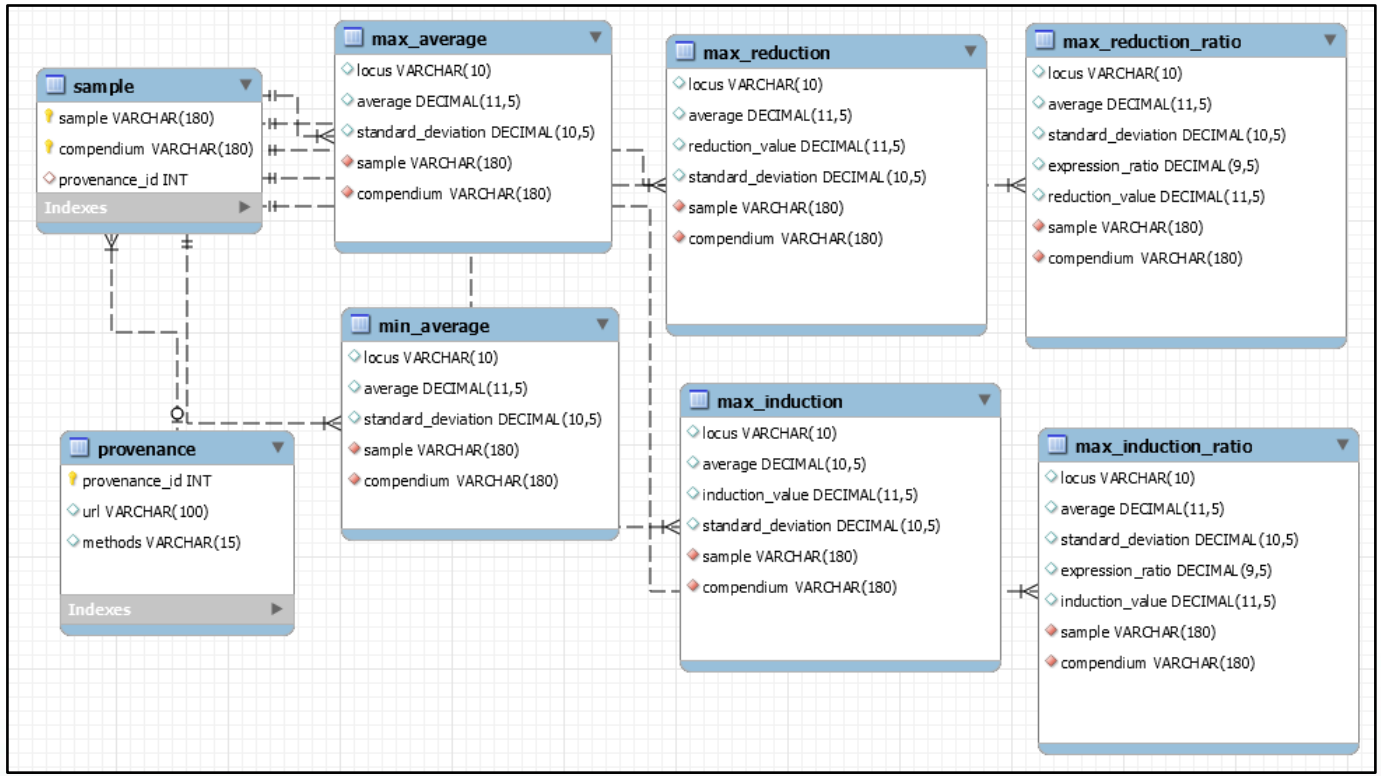
- Create new SQL database to store the output of this API for each gene
- This way we query the PHP API once to populate the database, and future users need only query the new SQL database


```
{
  "wasSuccessful": true,
  "error": null,
  "statusCode": 200,
  "locus": "At4g11945",
  "species": "Arabidopsis",
  "sources": [
    {
      "maxAverageValue": {
        "ExpressionAverage": 0.390096,
        "StandardDeviation": 0.390096,
        "InductionValue": 0,
        "ReductionValue": 0,
        "ControlValue": 0,
        "FoldChange": 0,
        "SampleSize": 2,
        "Compendium": "Klepikova",
        "Sample": "S_H",
        "data_bot_id": "'SRR3581336', 'SRR3581740'",
        "ControlSample": "Med_CTRL",
        "control_data_bot_id": "'Med_CTRL'",
        "db": "klepikova"
      },
      "topMaxAverageValue": { … }, // 13 items
      "maxInductionValue": { … }, // 13 items
      "topMaxInductionValue": { … }, // 13 items
      "maxReductionValue": { … }, // 13 items
      "topMaxReductionValue": { … }, // 13 items
      "maxFoldChange": { … }, // 13 items
      "topMaxFoldChange": { … }, // 13 items
      "minAverageValue": { … }, // 13 items
      "topMinAverageValue": { … }, // 13 items
      "minFoldChange": { … }, // 13 items
      "topMinFoldChange": { … } // 13 items
    }
  ]
}
```

Max Expression

SQL schema contains:

- API output
- Sample to provenance correspondence
- Provenance information



Max Expression

Problems with using this API:

- Microarray and RNA-seq data are returned together

Solution:

- Edit PHP script to take an extra field in the query
- Possible future issue: databases are hard-coded into script

```
if ($db == 'klepikova' || $db == 'germination' || $db == 'root_Schaefer_lab' || $db == 'shoot_apex') {
    if (strcmp($method, "microarray") == 0){
        continue;
    }
    $sql ="SELECT * FROM ". $db . ".sample_data WHERE data_bot_id IN (". $sampleString .") AND data_probeset_id='". $locus ."'";
} else {
    if (strcmp($method, "rna-seq") == 0 || strcmp($method, "rnaseq") == 0){
        continue;
    }
    $sql = "SELECT * FROM ". $db . ".sample_data WHERE data_bot_id IN (". $sampleString .") AND data_probeset_id=(SELECT t2.probeset FROM an
}
$sql_output = mysqli_query($conn, $sql);
```

Building an API for accessing the database

Tools used
- Node.js
- Express.js
- Knex.js

POST request format

Example of output

```
knex.select('locus', 'average', 'standard_deviation', 'sample', 'compendium')
    .from(table).whereIn('locus', resultLocus)
    .orderBy([{column: 'average', order: 'asc'}, {column: 'locus', order: 'desc'}])
.then(
  function(result){
      out["wasSuccessful"] = true;
      while(result.length != 0){
        ret = result.pop();
        k = Object.keys(out["maxAverage"][ret.locus.toUpperCase()]).length + 1;
        out["maxAverage"][ret.locus.toUpperCase()][k] = ret.average;
        out["standardDeviation"][ret.locus.toUpperCase()][k] = ret.standard_deviation;
        out["sample"][ret.locus.toUpperCase()][k] = ret.sample;
        out["compendium"][ret.locus.toUpperCase()][k] = ret.compendium;
      }
      out["statusCode"] = 200;
      out["error"] = null;
      res.set('Content-type', 'application/json')
      res.json(out);
    }
  )
.catch(function(error){console.error(error);})
}
```

Input


```
{
  "loci": [
    "AT1G01010",
    "AT4G01020"
  ],
  "method": "Microarray"
}
```


Output


```
{
  "maxAverage": {
    "AT1G01010": {
      "1": 412.402,
      "2": 410.655,
      "3": 401.23,
      "4": 385.297,
      "5": 380.635,
      "6": 379.88733,
      "7": 374.899,
      "8": 372.80467,
      "9": 364.317,
      "10": 355.898
    },
    "AT4G01020": { … } // 10 items
  },
  "standardDeviation": {
    "AT1G01010": { … }, // 10 items
    "AT4G01020": { … } // 10 items
  },
  "sample": {
    "AT1G01010": { … }, // 10 items
    "AT4G01020": { … } // 10 items
  },
  "compendium": {
    "AT1G01010": { … }, // 10 items
    "AT4G01020": { … } // 10 items
  },
  "wasSuccessful": true,
  "statusCode": 200,
  "error": null
}
```

Gene Regulatory Networks

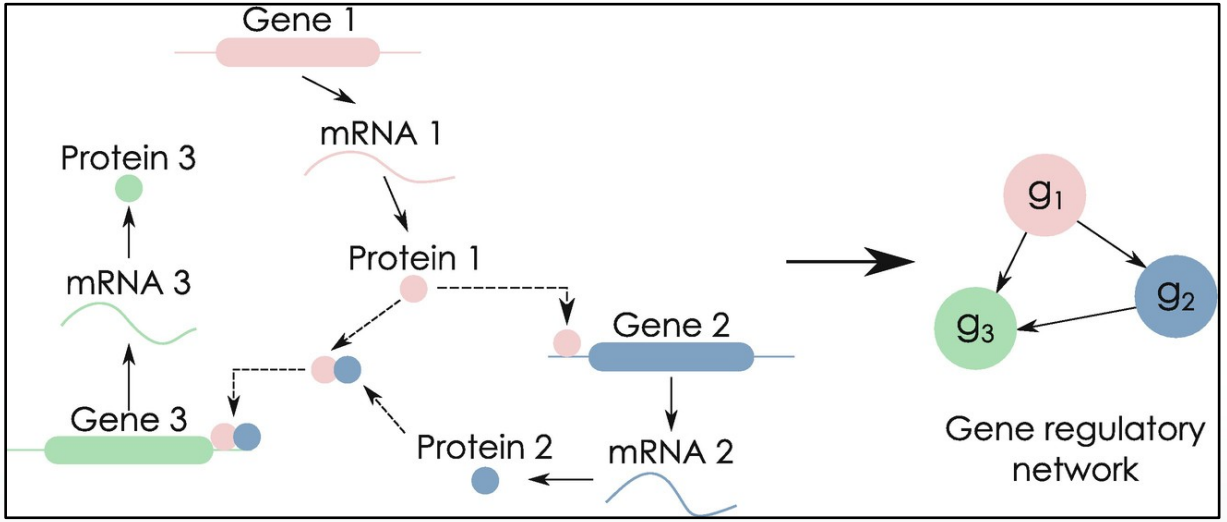
Directional network

Nodes represent genes and their regulators

Edges represent physical and regulatory interactions between them

Data obtained through high throughput methods such as DAP-seq, Y1H, or from previous literature

Can be used to predict response to treatments, unknown gene functions, as biomarkers, to discover important genes...



Huynh-Thu VA, Sanguinetti G. Gene Regulatory Network Inference: An Introductory Survey. In: Sanguinetti G, Huynh-Thu VA, editors. Gene Regulatory Networks: Methods and Protocols. New York, NY: Springer; 2019. p. 1–23. (Methods in Molecular Biology). https://doi.org/10.1007/978-1-4939-8882-2_1. doi:10.1007/978-1-4939-8882-2_1

AGENT

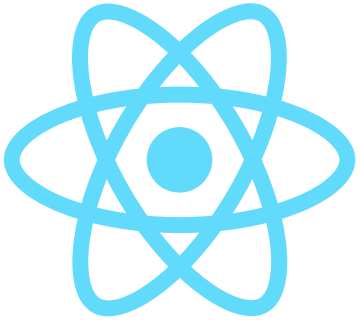
- Many GRNs currently published

- Current tools for visualization outdated and not user-friendly

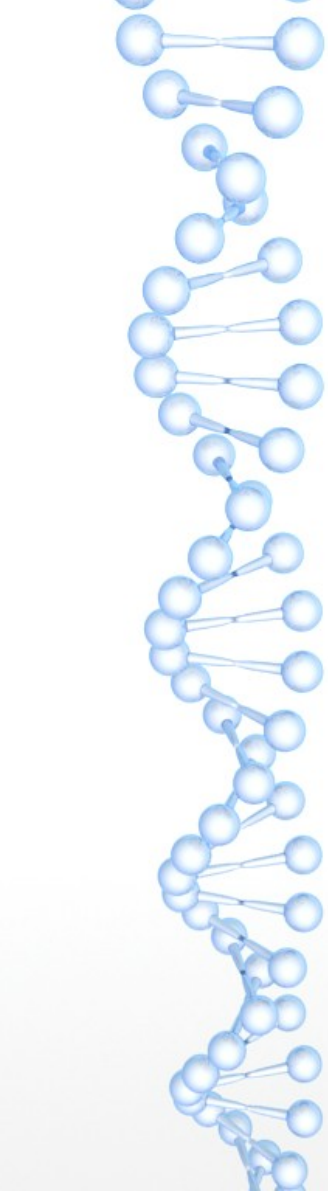
- AGENT (Arabidopsis Gene Regulatory Network Viewer) is a web-based GRN visualization tool created by Vincent Lau, built with modern technology and with user-friendliness in mind

AGENT

- React
    - User Interface library maintained by Facebook



- Cytoscape.js
    - Full-featured graph library written in Javascript



AGENT

- AGENT shows interaction data from GRNs

- Already allowed for overlaying expression on the nodes

- Two modes: absolute & relative

- Idea for new mode: potential (expression as percentage of a maximum)

AGENT

- New mode requires calculating the max average

- We have top 10 maximum expression values in the DB built before

- Can edit get_sample_data.php (written by Asher Pasha)

- Need to query script, calculate max average, return expression value / max average

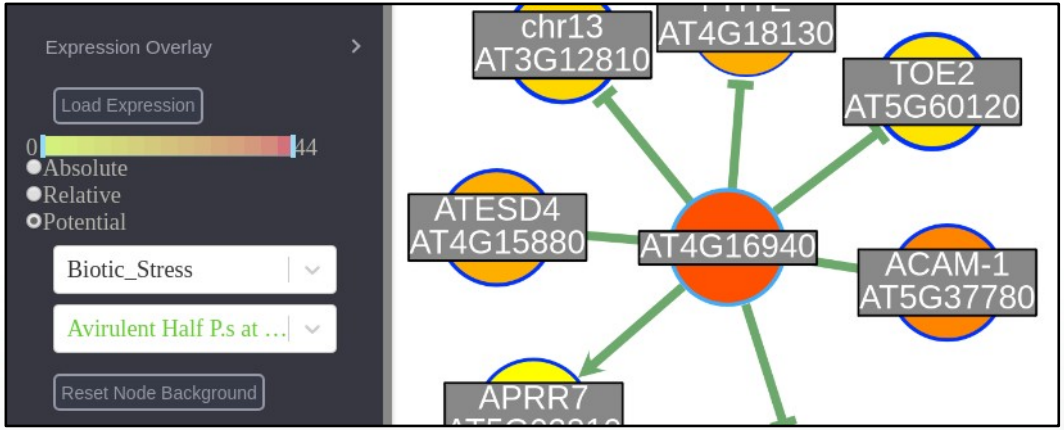
```php
function get_expression_potential($conn, $data_type, $gene) {
    $gene = mysqli_real_escape_string($conn, $gene);
    if ($data_type == "Microarray"){
        $query = "SELECT AVG(average) FROM max_average_microarray WHERE locus=?";
    } else if ($data_type == "RNASeq"){
        $query = "SELECT AVG(average) FROM max_average_rnaseq WHERE locus=?";
    } else {
        output_error('Invalid data type');
    }
    $stmt = mysqli_prepare($conn, $query);
    if ($stmt) {
        mysqli_stmt_bind_param($stmt, "s", $gene);
    } else {
        output_error('Failed to prepare MySQL statement in get_expression_potential()');
    }

    // Execute query
    mysqli_stmt_execute($stmt);

    // Bind results
    mysqli_stmt_bind_result($stmt, $average);

    // Get data
    mysqli_stmt_fetch($stmt);

    // Close
    mysqli_stmt_close($stmt);

    return (float)$average;
}
```

AGENT

- Now we can add the option in AGENT

- React component
    - Colour scale generated according to expression scale
    - Slider overlaid on it

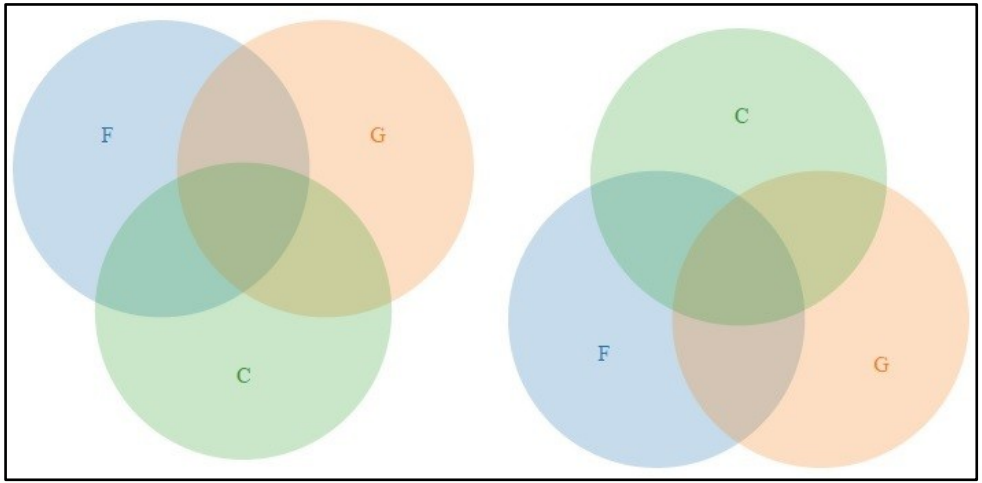
```
render(){
    return(
        <>
        <ColorScale
            scale={this.props.scaleClrs}
            minNum={Math.ceil(this.props.scaleMin)}
            maxNum={Math.ceil(this.props.dataAbsMax)}
        />
        <Range
            value={this.state.expressionSliderVal}
            defaultValue={[Math.ceil(this.props.scaleMin),
                           Math.ceil(this.props.dataAbsMax)]}
            min={Math.ceil(this.props.scaleMin)}
            max={Math.ceil(this.props.dataAbsMax)}
            step={1}
            onChange={this.onExpressionSliderChange}
            tipFormatter={value => `${value}`}
            tipProps={{
            placement: "top",
            } }
            allowCross={false}
            className="ExpressionSlider"
        />
        </>
    )
}
```



AGENT

- Users may want to compare GRNs
    - Which nodes are common?

- Venn Diagram
    - Venn.js, D3.js

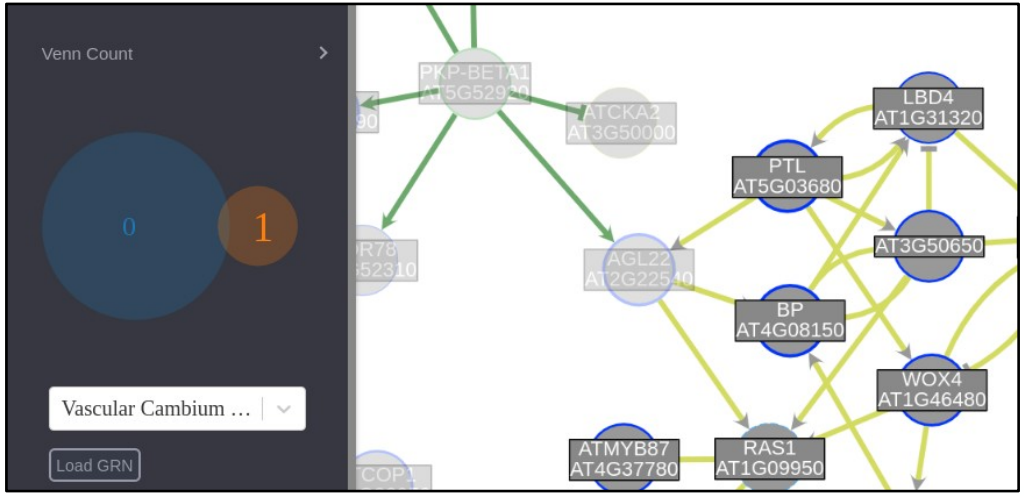


AGENT

- React component using Venn.js

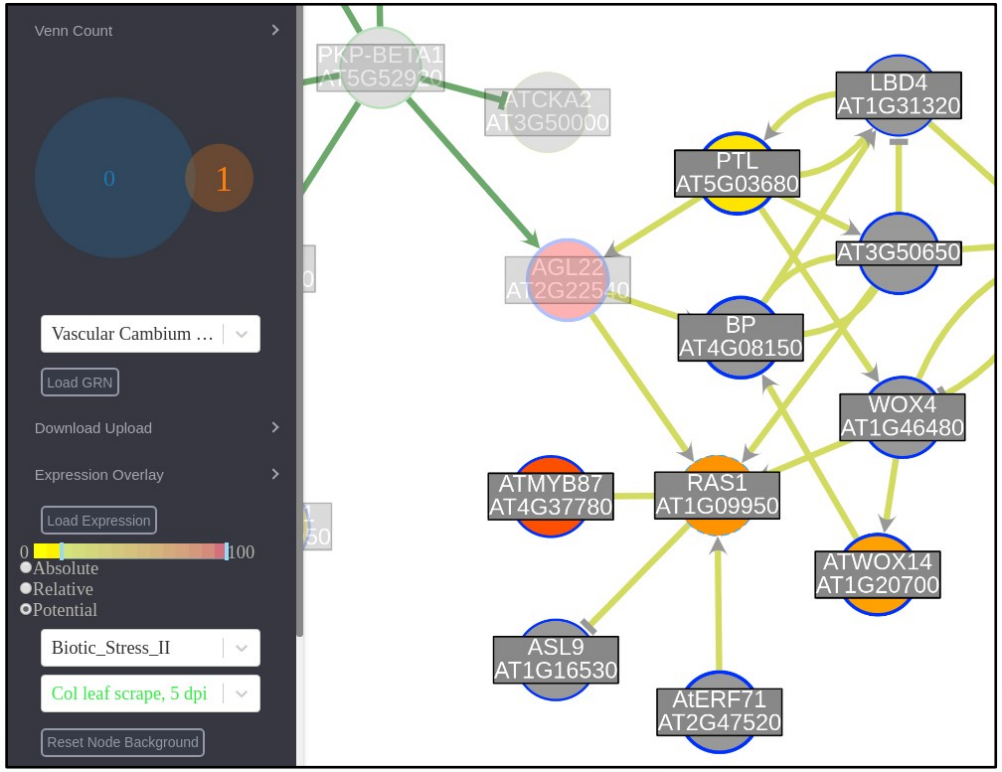
- Store network indices in nodes

- Use Cytoscape.js filters to retrieve sizes



AGENT

- Visual mapping of information



Additional work

- Python scripts
    - Parsing XML to JSON
    - Query get_max_expression.php to populate database
    - Add new database information

```
def buildJSON(fileInput, fileOutput):
    fileList = getFiles(fileInput)
    print(fileList)
    sinf = []
    scon = []
    jsonstr = "{"
    for i in fileList:
        fn = i.strip("xml\\").split(".")[0].replace("_", "")
        # Sample info
        sinf = getSampleInfo(i)
        # Sample controls
        scon = getSampleControls(i)
        jsonstr = jsonstr + "\"" + fn + "\""
        jsonstr = jsonstr + ": {\n \"sample\": {"
        jsonstr = jsonstr + json.dumps(scon[0])[1:-1]
        jsonstr = jsonstr + "}, \n \"controlComparison\": {"
        jsonstr = jsonstr + json.dumps(scon[1])[1:-1]
        jsonstr = jsonstr + "}, \n \"db\": \"" + getDbInfo(i) + "\" },"
        with open(fileOutput, "w+", encoding="utf-8") as f:
            f.write(jsonstr)
    # TODO Remove final comma and close curly brackets

    with open("srcs.txt", "w+", encoding='utf-8') as f:
        for i in range(len(allSamples)):
            f.write(str(allSamples[i]))
            f.write("\n")
            f.write(str(allDbs[i]))
            f.write("\n")
```