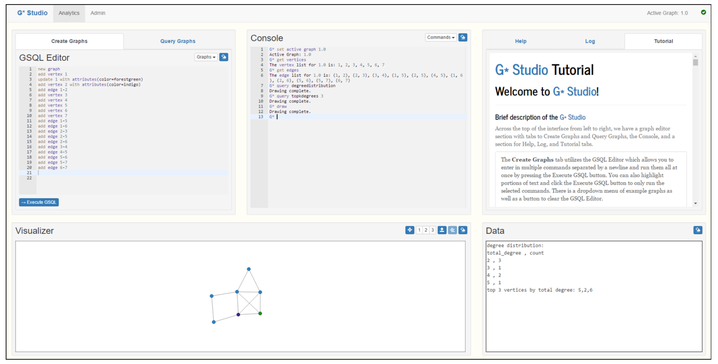
G* Graph Database Cross-Compiler

Author: Torin Reilly
Advisor: Alan Labouseur
Marist College School of Computer Science and Mathematics, Poughkeepsie, NY

G* The Dynamic Graph Database

A scalable graph database for big data



Problem

G* and G* Studio currently support two query methods:

PGQL: Procedural Graph Query Language

```
operator vertex@* = VertexOperator([], [0:1:4])
operator out_degree@* = ProjectionOperator([vertex@local], [id, graph.id, ca
rdinality(outgoing_edges)], [id, graph.id, out_degree])
operator partial_count@* = AggregateOperator([out_degree@local], [count],
[*], [count], [graph.id, out_degree])
operator union@@ = UnionOperator([partial_count@*])
operator total_count@@ = AggregateOperator([union@local], [sum], [count], [c
ount], [graph.id, out_degree])
operator sort@@ = SortOperator([total_count@local], [graph.id, out_degree:de
sc])
run sort@@
```

- Hard to use
- Unfamiliar
  but
- Very Flexible

Console Commands

Console

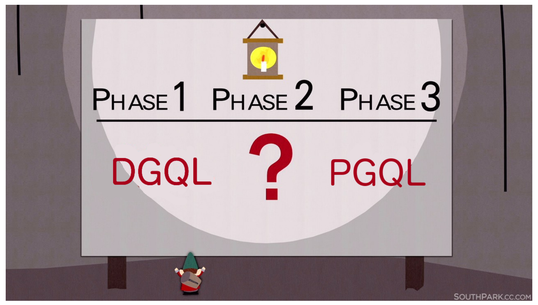
```
1  G* query degreedistribution
2  Drawing complete.
3  G* query topkdegrees 5
4  Drawing complete.
5  G*
```

- Easy to use
- Familiar
  but
- Very Limited

Solution

Develop a familiar and easy-to-use (SQL-like) query language that compiles to PGQL: DQGL (Declarative Graph Query Language)

Cross Compiler Requirements



- No additional strain on worker machines
- Integrate with existing G* Studio
- Be flexible and easy to use, while still leveraging all the power of our graph database

Tooling

PEG.js

- Generates a JavaScript- based parser
- Allows for tight control of token and AST generation
- Simple grammar syntax
- Allows for inline JavaScript in grammar

JavaScript

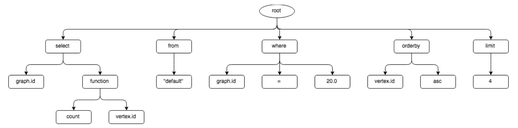
- Client-side
- Integrates directly into G* Studio
- Not as bad as people say

PEG.js Grammar for DGQL

```
Query = StatementList
StatementList = SelectStatement FromStatement? WhereStatement OrderByStatement? LimitStatement?
SelectStatement = _ s:(Select (FieldList)){
    return s;
}
Select =_ s:"select"{return s;}
LimitStatement = _ l: ("limit" integer){return l;}
FieldList =
    fields:(
      first:Field
      follow:(comma f:Field { return f; })*
    { return [first].concat(follow); }
    )
    { return fields }
Field = f:FieldType a:AliasAssign?{
    if(a == null)
        return f;
    else return {f,a}
}

FieldType = attr: (GraphAttribute / VertexAttribute /Operator) {return attr;}
AliasAssign = _ as:"as"i a:Alias{return a}
Alias       = _ alias:string{
return alias;
}
GraphAttribute = _ g:"graph.id"i{return g;}
VertexAttribute = _ v:("vertex.id"i/"vertex.c_coef"i/"vertex.total_degree"i/"vertex.page_rank"i){return v;}
GraphID = _ f:("id"){return f;} //should be float
Operator = I:(Count / AVG ) / All)
Count = _"count"i _ "(" v:VertexAttribute ")"{return ["count",v]}
AVG = _"avg"i _ "(" v:VertexAttribute ")"{return ["avg",v]}
All = "*"
FromStatement = _ from:("from"i FromDir){return from;}
FromDir "directory" = _ dir:string{return dir;}
WhereStatement = _ w: ("where" GraphAttribute equal GraphID){return w;}
OrderByStatement = _ o:(OrderBy (VertexAttribute/ExistingAlias) Order){return o;}
ExistingAlias =_ a:string {return a;}
OrderBy = _ o:"orderby"i _ "by"{return "orderby";}
Order   = _ o:(ASC/DESC)?{ if(o!=null && o != undefined){return o;}else{return "ASC";}}
ASC = _ a:"asc"i{return a;}
DESC = _ d:"desc"i{return d;}

//types
comma = _ ","
equal = _ e:"="{ return e;}
float "float"
    = left:[0-9]+ "." right:[0-9]+ { return parseFloat(left.join("") + "." +  right.join("")); }
integer
  = _ int:[0-9]+ {return int.join().replace(",","");}
  _"whitespace"= [ \t\n\r]*
string "string"
  = quotation_mark chars:char* quotation_mark { return chars.join(""); }
  char
  = unescaped
  / escape
    sequence:(
        '"'
      / "\\"
      / "/"
      / "b" { return "\b"; }
      / "f" { return "\f"; }
      / "n" { return "\n"; }
      / "r" { return "\r"; }
      / "t" { return "\t"; }
      / "u" digits:$(HEXDIG HEXDIG HEXDIG HEXDIG) {
          return String.fromCharCode(parseInt(digits, 16));
        }
    )
    { return sequence; }

  escape         = "\\"
  quotation_mark = '"'
  unescaped      = [\x20-\x21\x23-\x5B\x5D-\u10FFFF]

  DIGIT  = [0-9]
  HEXDIG = [0-9a-f]i
```

AST

- Generated AST as nested JavaScript array
- Tokenized every valid term
- Optimized

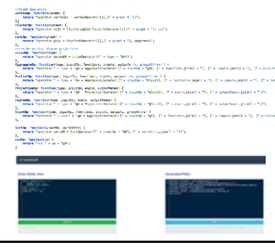


AST as tree

AST as formatted JS Array

Code Gen

- Accomplished via templating PGQL operators
- First prototyped as standalone application



Integration

- Compiler added to G* Studio
- Allows for editing of generated PGQL for quick alterations
- Provides simple and powerful interface to query graph data