BOMGenerator 使用手册

Version 1.0

DBIIR 教育部重点实验室

中国人民大学信息学院

引言

编写目的

详细清晰介绍 BOMGenerator 的功能与使用流程，以供用户参考。

背景

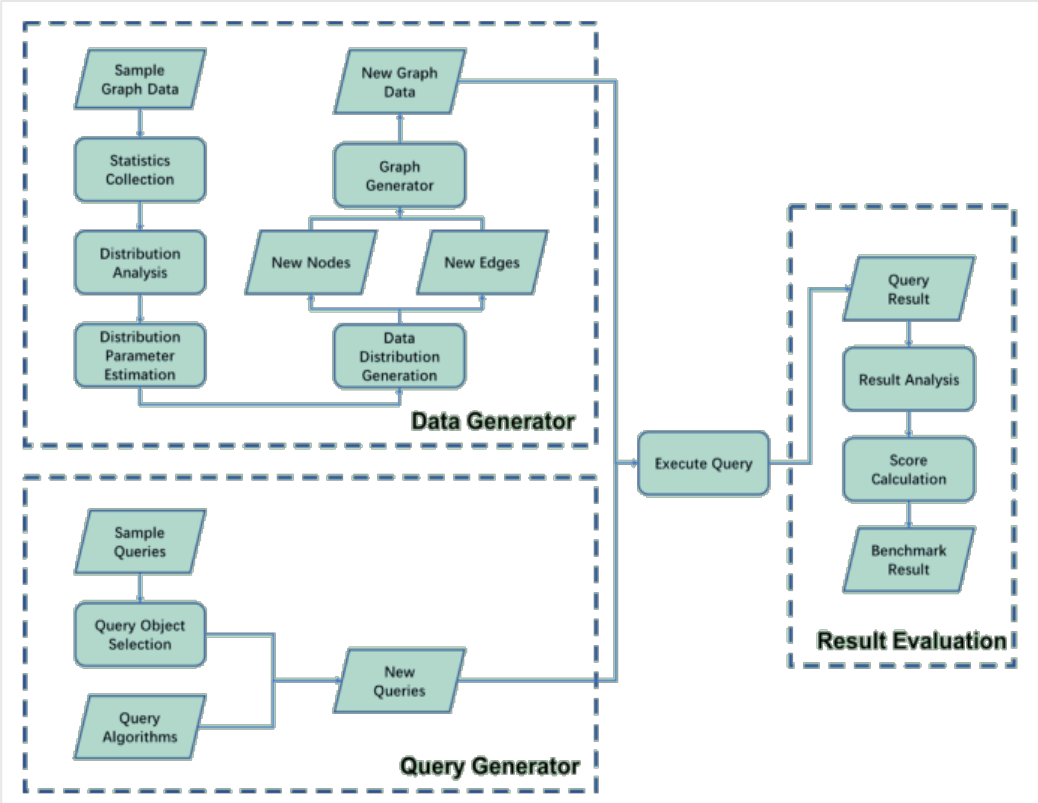
高端制造业中图的管理是最典型的一种应用。在大规模的图操作上，传统的关系模型常常会显示出明显的缺点。因此越来越多的专用图数据库开始涌现，这些图数据库在模型设计之初便考虑了图数据查询的特点并进行了相关的优化。随着图数据库快速发展，我们亟需一套评测基准来评判图数据库的性能优劣，但目前的图数据库基准大多面向通用的图数据，尤其是社交网络的数据，仍然缺乏针对高端制造业的基准，而高端制造业图数据的查询管理与社交网络图数据查询管理又展现出完全不同的特点。因此我们的研究目标为设计一个高端制造行业的图数据库评测基准。以此为目标，我们开发了 BOMGenerator 工具用于 BOM 图数据的生成和工业生产查询负载的生成来服务我们的项目。

1.软件简介

1.1 功能

本软件主要功能为 Bom 图数据的生成以及响应查询负载的生成，同时对 neo4j 数据库进行性能上的测试和综合评估。目前支持以下功能，分析已有的 Bom 图数据结构，大规模生成指定目标结构的 Bom 图数据，生成工业生产中常见的查询负载，生成 neo4j 下可自动执行测试数据库性能的脚本文件 。

1.2 架构



1.3 开发环境

本软件生成器部分基于 python 进行开发，数据接口和前端构建基于 SpringMVC 和 html 等。目前生成的数据保存在本地，数据来源于真实的起重机数据，本软件主要进行生成数据和脚本的任务，故可扩展性强，可对其他数据库进行扩展，代码灵活度高，轻量级无侵入。

2.安装

2.1 运行环境

（1）操作系统：Linux System

（2）其他软件：Neo4j,Tomcat

（3）Python 版本：python3.X

2.2 安装

2.2.1 neo4j 数据库

（1） 下载 neo4j 安装包

（2）解压至相应目录，如 /software/mark/neo4j-community-3.4.6

（3）为 neo4j 配置全局环境变量

2.2.2 tomcat

（1）下载 tomcat 安装包

（2）解压至相应目录，如/software/mark/apache-tomcat-8.5.24

2.2.3 python 库

（1）安装包括 numpy, py2neo, scipy, numpy 等

2.2.4 工具

（1）上述内容安装完成后，解压 GenGraph.zip (由数据生成代码生成)至相应目录
如/software/mark/GenGraph

（2）解压 graph-benchmark.war（由数据接口及前端界面代码生成）至 tomcat 的 webapp 目录下

（3）将附件中的 procedure-template-1.0.0-SNAPSHOT.jar（由存储过程创建代码生成）放入 neo4j 的 plugin 目录下

3.准备

3.1 Neo4j 准备

3.1.1 启动 neo4j

通过 neo4j 的启动命令启动 neo4j（命令清单：命令 1），查看 neo4j 状态（命令清单：命令 2），如果提示 Neo4j is running 则开启成功；

3.1.2 向 neo4j 导入样本数据

例如将 node.csv 和 relation.csv 分别放入 neo4j 目录下，通过调用附件 load.cypher 脚本来进行数据的导入（命令清单：命令 3）；

3.2 工具准备

3.2.1 开启数据生成器服务

进入 GenGraph 目录中，运行命令（命令清单：命令 4）开启数据生成服务，如果出现提示“Started httpserver on port 5000”说明服务已经开始正常运行。

3.2.2 开启可视化服务

进入 tomcat 的 bin 目录，开启 tomcat 服务（命令清单：命令 5）即可启动可视化服务

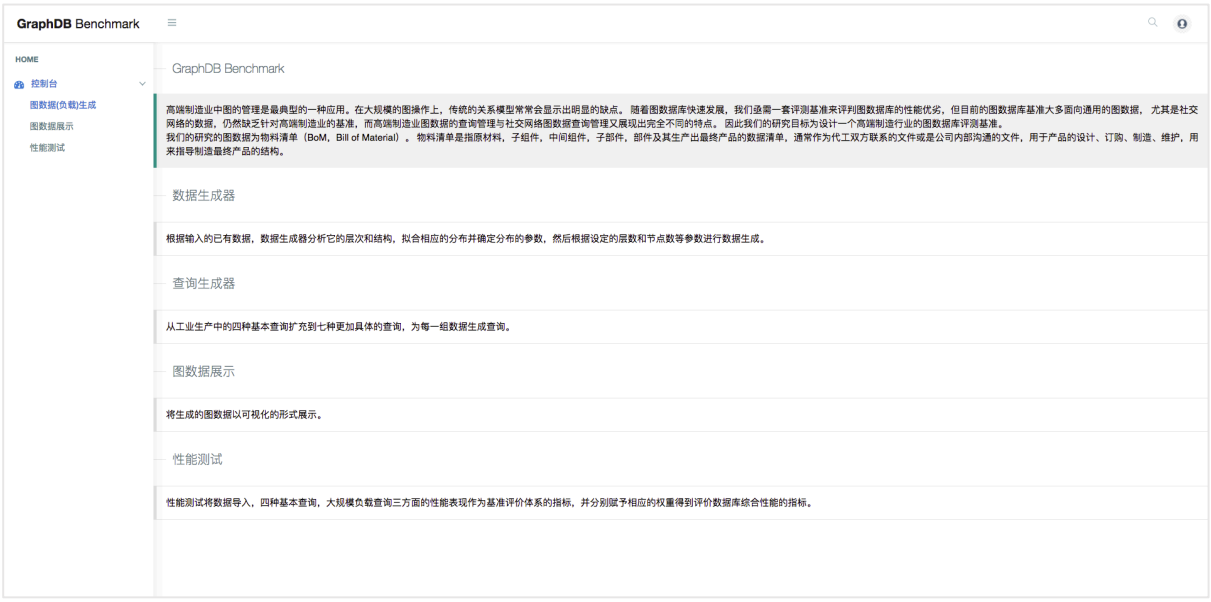
命令清单
neo4j start 命令 1
neo4j status 命令 2
neo4j-shell –file load.cypher 命令 3
python3 genServer.py 命令 4
sh startup.sh 命令 5

4.使用及效果

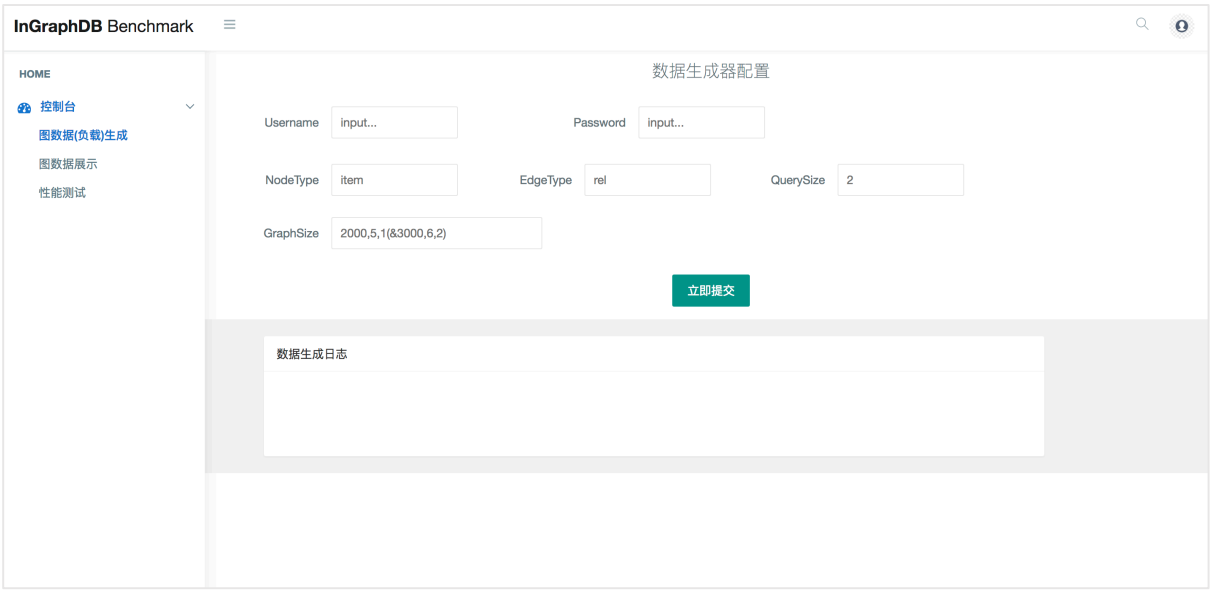
4.1 数据生成器配置

通过浏览器访问 http://server_ip:8080/graph-benchmark

即可进入到如下界面



点击“图数据库（负载）生成”模块

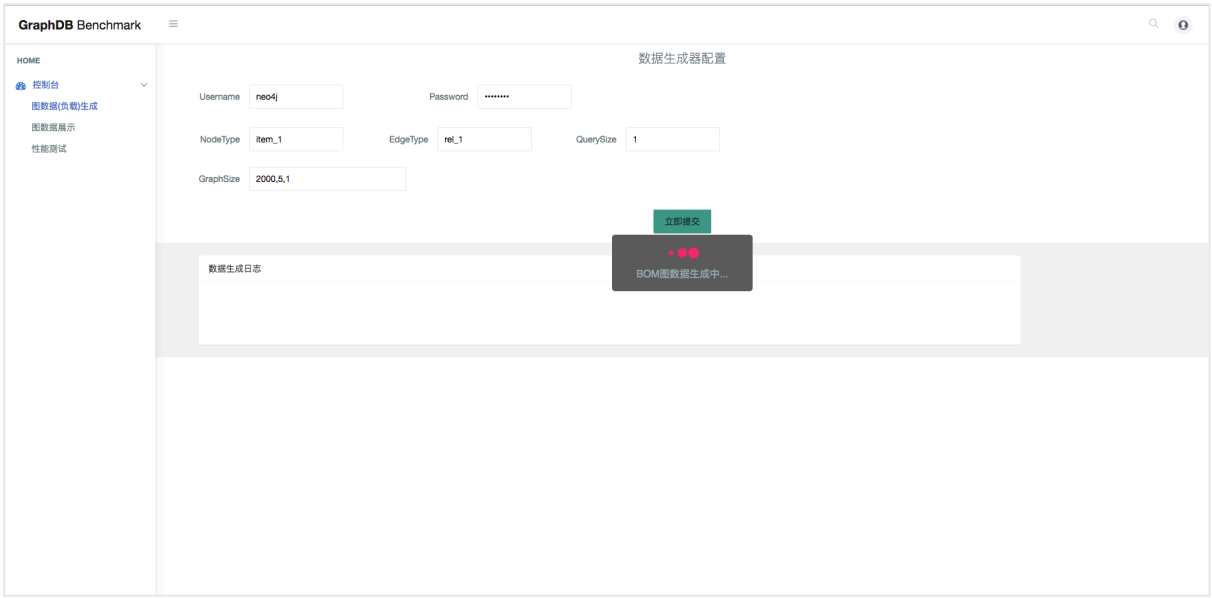


参数配置共包含以下信息：

参数名称	参数意义	备注信息
Username	neo4j 数据库的用户名	这两项需正确输入保证与 neo4j 数据库的正确连接
Password	neo4j 数据库的对应用户的密码	
NodeType	所要分析的 Bom 图数据在数据库中存的结点类型	指定了结点的类型和边的类型就确定了 Bom 图，软件将会自动分析该 Bom 图的连接结构，进行公式化，用于数据生成
EdgeType	所要分析的 Bom 图数据在数据库中存的边的类型	
QuerySize	指的是查询的大小	这里查询的大小为对每一个图生成的查询组数，一组查询包括由 4 个基本查询组成的 7 类复杂查询，每一组查询的内容均为按梯度的随机生成
GraphSize	指的是图的信息,输入内容为一个个三元组，三元组的第一个元素表示图的规模，第二个元素表示图的最大层次，第三个元素表示生成这样的图的个数	graphsize 输入的三元组个数没有限制,多组用&连接。如”2000,5,1&3000,6,2”软件将会生成结点数为 2000，层次为 5 层的图 1 个，结点数为 3000，层数为 6 层的图 2 个。且图的结构近似于指定的起重机数据图的结构

输入完所有的参数信息后，点击生成，软件便开始生成相应的数据，查询以及可执行脚本文件。

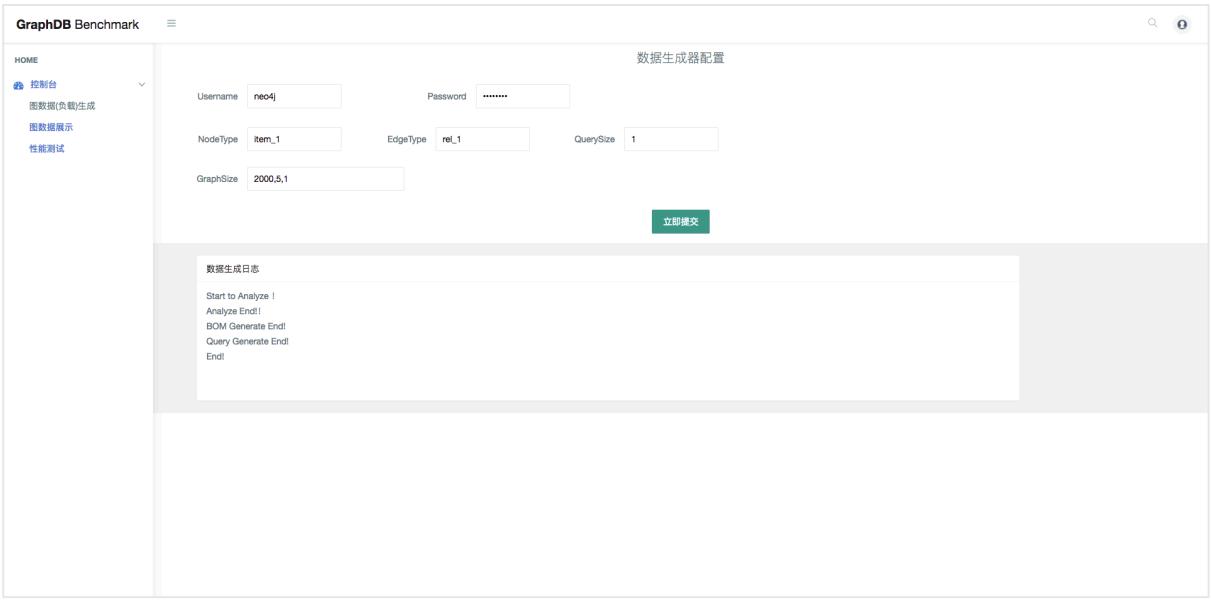
生成等待页面



生成完毕之后，可以看到数据生成器服务端日志信息如下：

```
[mark@PowerEdge-T640:/software/mark/GenGraph$ python3 pythonServer.py
Started httpserver on port  5000
10.77.50.193 - - [12/Sep/2018 11:42:35] "GET /startGen?neo4j&xiaojian&item_1&rel_1&1&2000,5,1 HTTP/1.1" 200 -
protocol:
hostname: None
port: None
path: /startGen
query: neo4j&xiaojian&item_1&rel_1&1&2000,5,1
[INFO] Start. (2018-09-12 11:42:35)
[INFO] Finish getting raw graph from database. (2018-09-12 11:43:55)
Analyze End!
[INFO] Finish extraction the node of the raw graph (size-1: 2426, size-2: 1538). (2018-09-12 11:43:55)
[INFO] Finish extraction the edge of the raw graph (size-1: 3018, size-2: 1747). (2018-09-12 11:43:55)
[INFO] Finish creating 2097 node of the graph 0-1. (2018-09-12 11:44:02)
[INFO] Finish creating 1367 edge of the graph 0-1. (2018-09-12 11:44:24)
BOM Generate End!
Query Generate End!
[INFO] End. (2018-09-12 11:44:24)
```

前端的状态如下



4.2 生成过程分析

（1）对于指定的 Bom 图数据，软件通过 neo4j 对其进行分析完毕，在 generator/thesis/data 目录下得到如下图所示的内容指导进行数据生成。

相应的 Bom 图数据生成完毕，在 generator/thesis/result 目录下可以看到生成的内容。

文件名	文件介绍
Deploy_Neo4j.cypher	将数据从 csv 加载到图数据库的脚本
Query_Neo4j.cypher	包含插入、简单查询、复合查询三个部分来测试数据库的性能，其中记录了分别的运行时间
Execute_Command.sh	自动化性能测试脚本
g1_new_nodes_as_files_item.csv	生成的图数据-节点数据
g1_new_nodes_as_files_relation.csv	生成的图数据-边数据

其中 Deploy_Neo4j.cypher 为向数据库中导入数据的脚本：

```
MATCH ()-[r:PointTo_test]->()
        DELETE r;
        MATCH (n:item_test)
        DELETE n;
        return 'Start time: ' + timestamp();
        LOAD CSV WITH HEADERS FROM "file:///g1_new_nodes_as_files_item.csv" AS csvLine
        CREATE (p:item_test { plm_m_oid: csvLine.plm_m_oid, plm_oid: csvLine.plm_oid, plm_m_id: csvLine.plm_m_id, plm_i_name: csvLine.plm_i_name, plm_i_createtime: csvLine.plm_i_createtime, plm_i_checkintime: csvLine.plm_i_checkintime, plm_cailiao: csvLine.plm_cailiao, plm_checkintime: csvLine.plm_checkintime, plm_weight: csvLine.plm_weight, plm_wlly: csvLine.plm_wlly, plm_guige: csvLine.plm_guige, plm_gylx: csvLine.v });
        CREATE CONSTRAINT ON (p:item_test) ASSERT p.plm_m_oid IS UNIQUE;
        CREATE INDEX ON :item_test(plm_oid);
        CREATE INDEX ON :item_test(plm_m_id);
        LOAD CSV WITH HEADERS FROM "file:///g1_new_nodes_as_files_relation.csv" AS csvLine
        MATCH (p1:item_test { plm_m_oid: csvLine.plm_leftobj}), (p2:item_test { plm_m_oid: csvLine.plm_rightobj})
        CREATE (p1)-[:PointTo_test { plm_rightobj:csvLine.plm_rightobj, plm_leftobj:csvLine.plm_leftobj, plm_oid: csvLine.plm_oid, plm_createtime: csvLine.plm_createtime, plm_order: csvLine.plm_order, plm_jianhao: csvLine.plm_jianhao, plm_number: csvLine.plm_number }]->(p2);
        return 'End time: ' + timestamp();
        CREATE INDEX ON :PointTo_test(plm_rightobj);
        CREATE INDEX ON :PointTo_test(plm_leftobj);
        CREATE INDEX ON :PointTo_test(plm_oid);
```

其中 Query_Neo4j.cypher 为执行脚本的相应命令：

```
return 'Start time: ' + timestamp();
CALL example.where_used2('139', 6, 'item_test', 'PointTo_test') yield pid, cid, poid, coid, rel_oid, cnt, lev, path return 'where_used2: '+count(*);
        call example.generate_structure2('484', 6, 'item_test', 'PointTo_test') yield ROOTID, PITEM_OID, CITEM_OID, PITEM_ID, CITEM_ID, ITEM_NAME, ITEM_OID, ITEM_NUMBER, ITEM_LEVEL, IS_LEAF, ITEM_PATH return 'generate_structure2: '+count(*);
        call example.structure_diff2('484', '484', 6, 'item_test', 'PointTo_test') yield diff_type, src_rel_oid, src_pid, src_cid, src_number, dst_rel_oid, dst_pid, dst_cid, dst_number return 'structure_diff2: '+count(*);
        CALL example.struct_aggr2('484', 6, 'item_test', 'PointTo_test') yield root_id, root_name, cid, cname, usage_count, min_lev, max_lev, qty_sum, weight_sum return 'struct_aggr2: '+count(*);
        call example.struct_aggr2('484', 6, 'item_test', 'PointTo_test') yield root_id, root_name, cid, cname, usage_count, min_lev, max_lev, qty_sum, weight_sum, plm_wlly with root_id, root_name, cid, cname, usage_count, min_lev, max_lev, qty_sum, weight_sum, plm_wlly where plm_wlly="外购" return 'plan: '+count(*);
        call example.struct_aggr2('484', 6, 'item_test', 'PointTo_test') yield root_id, root_name, cid, cname, usage_count, min_lev, max_lev, qty_sum, weight_sum, plm_wlly with root_id, root_name, cid, cname, usage_count, min_lev, max_lev, qty_sum, weight_sum, plm_wlly where usage_count > 1 return 'usage: '+count(*);
        call example.struct_aggr2('484', 6, 'item_test', 'PointTo_test') yield root_id, root_name, cid, cname, usage_count,
```

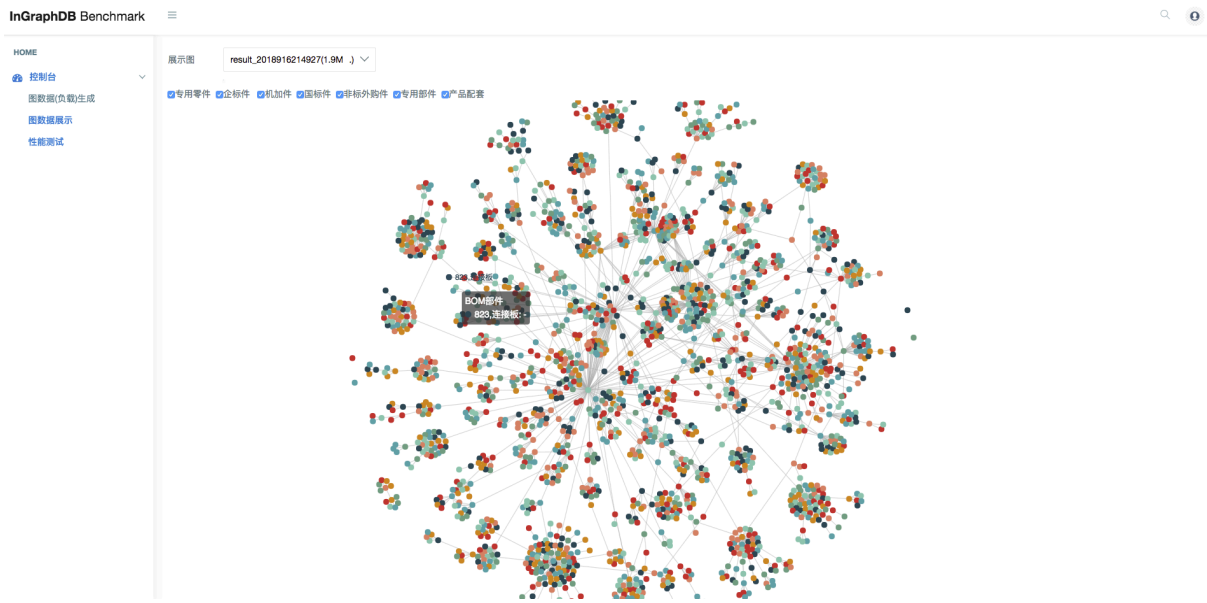
```
min_lev, max_lev, qty_sum, weight_sum, plm_wlly with root_id, root_name, count(cid) as count_cid, max(max_lev) as max_max_lev, sum(qty_sum) as sum_qty_sum, sum(weight_sum) as sum_weight_sum return 'statistic: '+count(*);
        return 'All execution time on Graph 0: ' + timestamp();
return 'End time: ' + timestamp();
```

其中 Execute_Command.sh 为自动化测试的脚本，调用了 Deploy_Neo4j.cypher 和 Query_Neo4j.cypher 两个脚本；

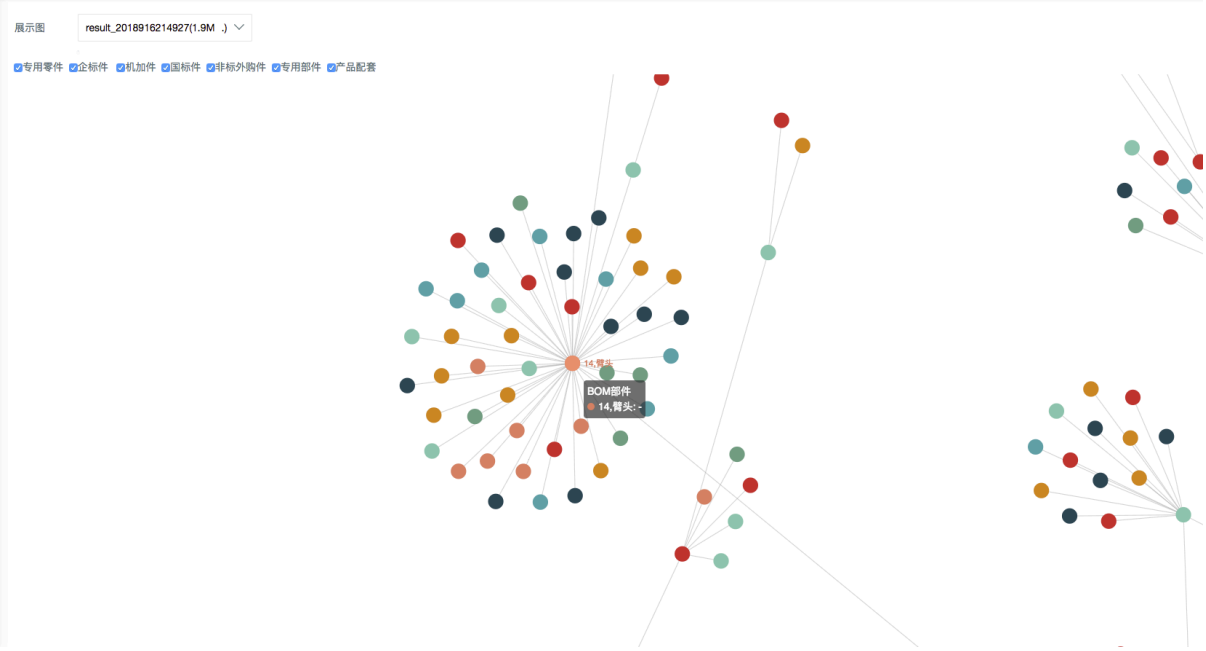
```
#Neo4j:
        #copy csv to neo4j import dir
        cp /software/mark/GenGraph/result/*.csv /software/mark/neo4j-community-3.4.6/import
        #execute Deploy.cypher to load csv to database instance
        cd /software/mark/neo4j-community-3.4.6/bin
        ./neo4j-shell -file /software/mark/GenGraph/result/Deploy_Neo4j.cypher
        #performance test
        ./neo4j-shell -file /software/mark/GenGraph/result/Query_Neo4j.cypher
```

4.3 数据可视化

点击“图数据展示”模块即可将生成的数据可视化的展示出来



局部效果图



4.4 性能测试

点击“性能测试”模块即可自动将生成的数据导入图数据库并自动化的调用生成的查询负载进行性能测试，最后通过统计直方图可视化的展示出来。