Network Graph generation from *scripts*
Geração de grafos de rede a partir de scripts

Abstract

We can represent a computer network as a finite set of Nelements, $\{v_i\}_{i=1,\dots,N}$, where there are pairwise relations among distinct elements, (v_k, v_l), $k \neq l$. Through a table with the pairwise elements, we can generate a graph that represents the network. Besides that, it is also possible to calculate the betweenness centrality c_B of each one of the parts. This paper describes the tools to generate the graph and calculate c_B. All of this allows support to produce adequate documentation of a computer network.

Resumo

Uma rede de computadores pode ser representada como um conjunto finito de N elementos $\{v_i\}_{i=1,\dots,N}$, onde existem relações de pares (v_k, v_l), $k \neq l$. Por meio de uma tabela contendo os pares, é possível gerar um grafo representando a rede. Além disto, também é possível calcular o grau de centralidade de intermediação c_B de cada um dos elementos. Este artigo mostra as ferramentas utilizadas para a geração do grafo e o cálculo de c_B. Isto permite ajudar a produção de uma documentação adequada sobre uma rede de computadores.

Palavras-chaves: redes, grafos, Graphviz, Python
Keywords: networks, graphs, Graphviz, Python

1. Introdução

A tecnologia para redes de computadores foi desenvolvida nas últimas décadas do século passado e sua aplicabilidade permeia praticamente quase todas as atividades humanas nestas duas décadas do século XXI.

A expansão da Internet no mundo é fruto e também o propulsor desta tecnologia de rede: em 2016, o número de usuários de Internet no mundo ultrapassou a marca de 3,4 bilhões, o que representa aproximadamente 45% da população mundial [Roser *et al.*, 2015].

Dentro do mundo corporativo, a tecnologia de informação (TI) é parte da estratégia de negócios da grande maioria das empresas, mesmo que algumas não reconheçam este fato. Dentro das atividades da TI encontram-se o processamento de dados e a gerência das conexões de rede e de telecomunicações. Devido à abrangência que a rede de computadores assume dentro de uma corporação, sua descrição documentada é importante tanto para servir de embasamento para as atuais e futuras linhas de negócios e de trabalho, como para a segurança cibernética [Posey, 2009].

Existem inúmeros procedimentos e aplicativos para a produção da documentação de uma rede; contudo, vários problemas existem nesta produção, como relata D. Liu [Liu,2009]: constante mudanças e ampliações da infraestrutura de TI, alta rotatividade dos funcionários, falta de conhecimento e de treinamento, baixo investimento em tecnologia, dados sobre os ativos desatualizados ou inexatos.

Muitas ferramentas de mercado, com o intuito de solucionar ou contornar as situações acima descritas, buscam automatizar certas tarefas, fazendo um processo de descobertas (*discoveries*) sobre os próprios elementos das redes, utilizando o *Simple Network Management Protocol* (SNMP, vide a referência [Feit, 1995]) ou algum outro proprietário, como *Cisco Discovery Protocol*[1].

Pode-se afirmar que a maioria destas ferramentas se baseia na teoria dos grafos, como será explanada na próxima seção.

Teoria dos grafos

Historicamente, a teoria dos grafos se iniciou com o trabalho de Leonhard Euler, que discorreu sobre as Sete Pontes de Königsberg [Barr, 1989], publicado em 1736. Havia uma dúvida se era possível estabelecer um caminho em que uma pessoa percorreria todas as setes pontes que cruzavam o rio Pregel, onde havia duas ilhas, apenas uma vez. Euler provou que era impossível. Hoje, a teoria dos grafos é um ramo da matemática com uma grande aplicabilidade em diversas áreas, incluindo engenharia e ciências sociais.

Formalmente, de acordo com as referencias [Kolaczyk, 2009] e [Lucchesi *et al.*, 1979], um grafo $G = \{V, E\}$ é uma estrutura que consiste em um conjunto de vértices ou nós, $V = \{v_1, v_2, \dots, v_n\}$, e um conjunto E de arestas ou conexões, onde são listados pares não-ordenados de vértices distintos (v_i, v_j). O número de vértices dado por n é denominado ordem do grafo G e o número de arestas denomina-se tamanho de G. A Figura 1 mostra, de forma esquemática, um exemplo de grafo G.

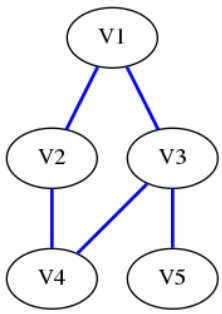

Figura 1: representação de um grafo G, cuja ordem e tamanho são 5.

A representação de um grafo, do ponto de vista matemático, não é tão necessária, a não ser para fins didáticos ou para a compreensão do conceito. Contudo, para o entendimento de uma topologia de rede, a representação gráfica é parte inerente tanto no seu planejamento como na sua manutenção.

[1] `https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cdp/configuration/15-mt/cdp-15-mt-book/nm-cdp-discover.html`, visto em 24 de junho de 2020.

Centralidade de intermediação (*betweenness centrality*)

A teoria de grafos apresenta um conceito que se tem mostrado de grande valia nos estudos de rede, particularmente possui grande repercussão no estudo de redes sociais: o conceito de centralidade de intermediação [Kolacyk, 2009].

Considere um vértice v_0 que esteja conectado a outros dois vértices distintos, respectivamente, v_i e v_j, formando os pares (v_0, v_i) e (v_0, v_j); o grau de centralidade de intermediação do vértice v_0, denotado por $c_B(v_0)$, é dado por:

$$c_B(v_0) = \sum_{v_0 \neq v_i \neq v_j} \frac{\sigma(v_i, v_j | v_0)}{\sigma(v_i, v_j)}.$$

No caso de uma rede de computadores, um nó com alto grau de centralidade de intermediação é um vértice que interliga um grande número de pares de nós e a sua eventual omissão prejudicaria em demasia as conexões da rede. No caso de uma rede social, um participante com alto grau é um membro da rede que é responsável pelo elo entre diversos outros participantes, inclusive de diferentes grupos com diferentes afinidades. Nem sempre é trivial encontrar quais membros possuem um alto grau de intermediação.

Por sua vez, apesar de uma rede de computadores ser um objeto de construção da engenharia e, portanto, seguiria um planejamento com um resultado supostamente determinístico, sua complexidade é tal que muitas vezes ela requer um tratamento de fenômeno natural, como enfatiza Abry, Baraniuk *et alli* [Abry *et al.*, 2002].

2. Objetivo

O objetivo deste estudo é avaliar a possibilidade de estabelecer uma visão gráfica da topologia de uma rede de computadores a partir de planilhas utilizando *scripts* que convertem dados descritivos em grafos. Por meio da análise de um caso, mostramos como funciona a técnica que propomos na seção *Metodologia* a seguir.

Os passos sugeridos pela técnica apresentada permite visualizar as conexões existentes e os respectivos grau de centralidade de intermediação.

3. Metodologia

As ferramentas utilizadas para a análise da topologia de rede e a centralidade são o *graphviz*[2] e o *OR-Tools*[3] e a programação dos *scripts* foi feita em linguagem *Python 3.6*.

O *graphviz* é um *software*[4] de código aberto que interpreta uma linguagem de marcação (*markup language*), denominada *DOT*, voltada para geração de diagramas.

O *OR-Tools* é uma suite de *software* de código aberto patrocinada pela *Google* que oferece rotinas de otimização para a resolução de diversos problemas.

---

[2] Vide portal `https://graphviz.gitlab.io/about/`

[3] Vide portal `https://developers.google.com/optimization/flow/maxflow`

[4] Está sob licença da Common Public License version 1.0.

A partir das ferramentas descritas acima, seguimos os passos descritos abaixo:

1. Monta-se uma tabela com as conexões ou arestas, de acordo com a teoria dos grafos (vide seção *Teoria dos Grafos*), com os pares dos nós da rede, normalmente denominados *switches* ou *hubs*, e a sua respectiva velocidade de conexão (1000 ou 10000 Mbps[5], na maioria das situações). Um exemplo desta tabela está transcrita em Tabela 1. Apenas para convenção, denomina-se um dos nós como "origem" e o outro como "destino". Não se pode esquecer que as conexões de rede são bilaterais, ou seja, "origem" e "destino" devem ser alternados. Assim, é preciso considerar este fato no cálculo da centralidade.

2. Um *script* em Python foi desenvolvido (vide em *Apêndice - seção B*) para ler a tabela descrita acima onde se calculou o grau de centralidade e se montou o arquivo com o *markup language* para gerar o diagrama.

4. Resultados

O grafo gerado em nosso caso está ilustrado na Figura 2.

Em cada um dos vértices, além do rótulo que identifica os elementos da rede grau, também foi transcrito o grau de centralidade de intermediação. Os traços mais espessos mostram os *switches* com maior velocidade em Mbps. A topologia desta rede é do tipo estrela, com características hierárquicas, o que ficou bem evidente pelo grafo produzido. Observe que o elemento com o maior grau é o AGG ($D = 2525$); apesar de ser óbvio, o resultado mostra que é um ponto de falha. Outros pontos de falha são o SW013B ($D = 345$) e o SW021B ($D = 249$). Uma interrupção nestes elementos afetam muitas conexões.

Redes mais complexas podem se beneficiar com estas ferramentas. Particularmente no Hospital das Clínicas, a quantidade de pontos de rede está em torno de 8500, o que é inviável tratar sem alguma automatização.

[5] Mbps significa Megabits por segundo.

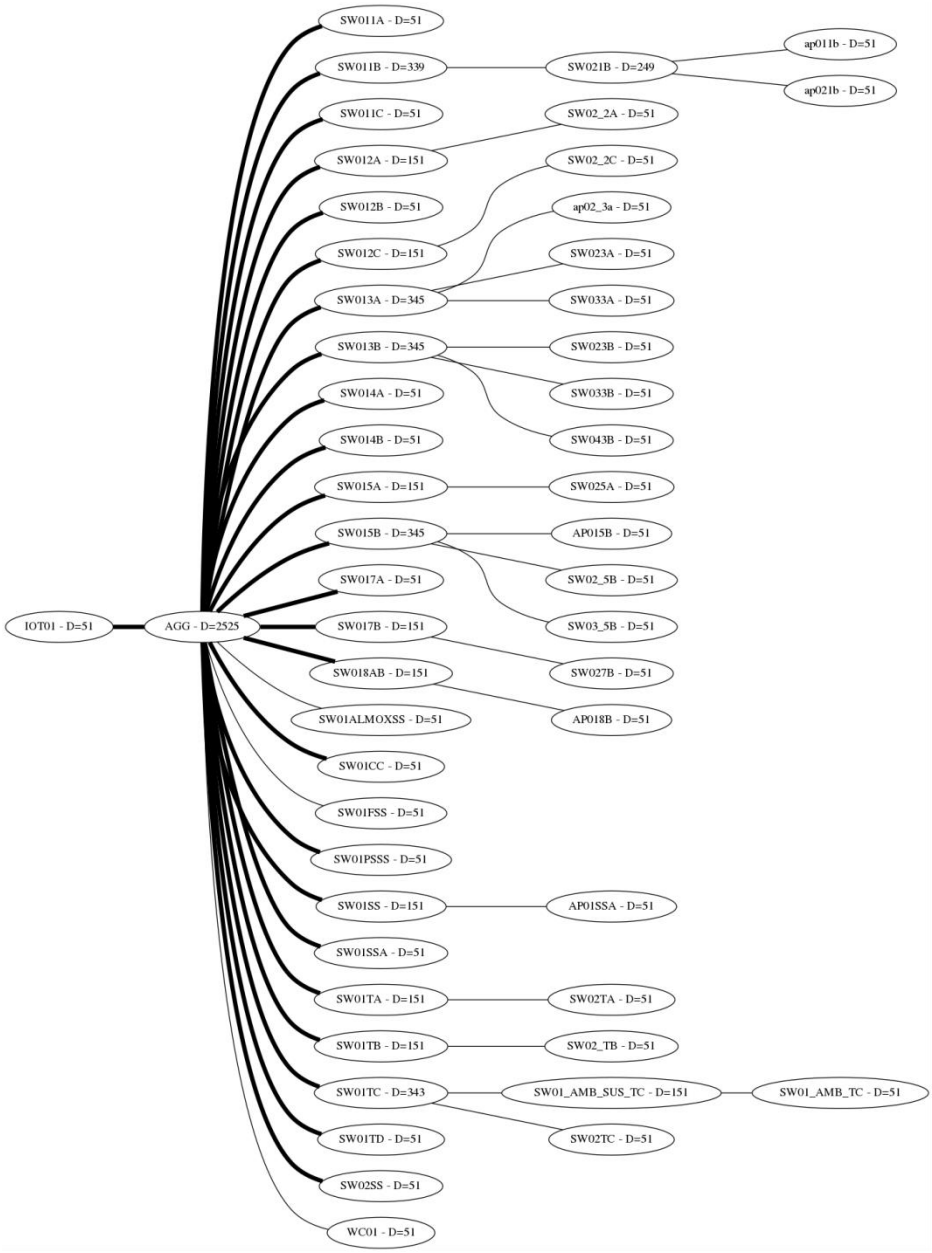

Figura 2: Diagrama da rede onde se encontra em cada balão o nome do nó e o seu respectivo grau de centralidade D.

5. Conclusão

A utilização destas ferramentas mostram-se bastante úteis para a produção de uma documentação de rede com menos enganos e mais atualizados, com a possibilidade de montar diagnósticos que apresenta possíveis gargalos, pontos de falha e de atenção.

6. Próximos passos

1. A intenção é estender esta análise para toda a rede do hospital;
2. Em seguida, pretende-se analisar as chamadas VLANs, que são as redes virtuais que correm sobre a chamada rede física;

3. O tráfego da rede também será incorporado;

4. Muitos elementos de rede hoje possuem um nível de automatização que pode ser aproveitado para fazer a identificação e a análise dentro dos moldes apresentados neste trabalho.

7. Referências

Abry, Patrice; Baraniuk, Richard; Flandrin, Patrick; Riedi, Rudolf; Veitch, Darryl. 2002. *Multiscale nature of network traffic*. IEEE Signal Processing Magazine, 19(3), 28–46.

Barr, Stephen. 1989. *Experiments in Topology*. Dover Books on Mathematical and Logical Puzzles, Cryptography and Word Recreations. Dover Publications.

Feit, Sidnie. 1995. *SNMP: A Guide to Network Management*. McGraw-Hill Computer Science Series. McGraw-Hill.

Kolaczyk, Eric D. 2009. *Statistical Analysis of Network Data - Methods and Models*. Springer Series in Statistics. Springer Science+Business Media.

Liu, Dale. 2009. *Chapter 5 - Diagramming the Network Infrastructure*. Pages 111 – 147 of: Liu, Dale (editor), Cisco Router and Switch Forensics. Boston: Syngress.

Lucchesi, Cláudio L.; Simon, Imre; Simon, Istvan; Simon, Janos; Kowaltowski,Tomasz. 1979. *Aspectos Teóricos da Computação*. Projeto Euclides. Livros Técnicos e Científicos Editora S.A.

Posey, Brien. 2009. *Chapter 6 - Inventories and Auditing*. Pages 107 – 157 of: Posey, Brien (ed), GFI Network Security and PCI Compliance Power Tools. Boston: Syngress.

Roser, Max; Ritchie, Hannah; Ortiz-Ospina, Esteban. 2015. *Internet. Our World in Data*. https://ourworldindata.org/internet (acesso em 25 de junho de 2020).

8. Apêndice

A - Exemplo de listagem dos equipamentos em formato tabular

Abaixo, na Tabela 1, transcrevemos uma parte das conexões e os pares dos switches com as respectivas velocidades de conexão.

Tabela 1: exemplo de dados contendo os nomes dos vértices e a velocidade de conexão entre os componentes dos pares.

Origem	Destino	Mbps
IOT01	AGG	20000
AGG	SW011A	10000
AGG	SW01FSS	1000
SW01SS	AP01SSA	1000
SW01TA	SW02TA	2000
SW01TB	SW02_TB	1000
SW01TC	SW01_AMB_SUS_TC	1000
SW01TC	SW02TC	1000
SW01_AMB_SUS_TC	SW01_AMB_TC	1000
SW011B	SW021B	1000
SW021B	ap011b	1000
SW017B	SW027B	1000
SW018AB	AP018B	1000

B - Listagem do *script* em Python utilizado no trabalho

O *script* abaixo faz a leitura da listagem de equipamentos, como mostra o exemplo listado na Tabela 1, analisa as conexões, faz o cálculo dos graus de centralidade e gera um texto em *markup language* a ser lido pela ferramenta *Graphviz*, gerando a Figura 2.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  9 12:33:44 2020
@author: futoshi
"""
import pandas as pd
import sys
from ortools.graph import pywrapgraph
from graphviz import Graph
"""
Functions
"""
# Create list without repetition
def norepeat(listaA, listaB):
    conjS = set()
    for e in listaA + listaB:
        if e not in conjS:
            conjS.add(e)
    return list(conjS)
# A trouble: the ortools functions don't accept string values, just numbers;
#We must associate a number to each string
def conversion(lista0, lista1):
    # First, we create a set (without repetition)
    # of string names from all lists
    listaS = norepeat(lista0,lista1)
    # Second, create a sorted list with conjS set
    listaS.sort()
    # Third, create a dictionary to associate a number to each string in listS
    rotuloDic = {}
    for i in range(0, len(listaS)):
        rotuloDic[listaS[i]] = i
    # Forth, create new lists correspondent to original lists (lista0, lista1)
    lista0N = []
    lista1N = []
    for i in range(0, len(lista0)):
        lista0N.append(rotuloDic[lista0[i]])
    for i in range(0, len(lista1)):
        lista1N.append(rotuloDic[lista1[i]])
    return rotuloDic, lista0N, lista1N
# Find key in dictionary by value
#                    from                    https://thispointer.com/python-how-to-find-keys-by-value-in-
dictionary/#:~:text=Find%20keys%20by%20value%20in%20dictionary,in%20a%20separate%20list%20i.e.
def getKbV(dicionario,valor):
    listofKeys = list()
    listOfItems = dicionario.items()
    for i in listOfItems:
```

```
            if i[1] == valor:
                listofKeys.append(i[0])
        return listofKeys
    # Analyze all network and find all betweenness centrality degrees
    #
    def flowanalysis(verticesN, arestas, laD, grau, PF):
        for v1 in list(verticesN):
            for v2 in list(verticesN):
                if v1 > v2:
                    PF.write('From %s to %s\n' %(getKbV(laD,v1)[0], getKbV(laD,v2)[0]))
                    if arestas.Solve(v1, v2) == arestas.OPTIMAL:
                        PF.write('-> Maximum flow: %d\n' %arestas.OptimalFlow())
                        PF.write('%-36s%-7s |%7s\n' %('Arcs','Flow','Capacity'))
                        for j in range(arestas.NumArcs()):
                            if arestas.Flow(j) != 0:
                                grau[arestas.Tail(j)] += 1
                                grau[arestas.Head(j)] += 1
                                PF.write('%-16s  ->  %-16s  %-7s|%7s\n'  %(getKbV(laD,  arestas.Tail(j))[0],
getKbV(laD,arestas.Head(j))[0], arestas.Flow(j), arestas.Capacity(j)))
                        PF.write('Source side min-cut:\n')
                        for k in range(0,len(arestas.GetSourceSideMinCut())):
                            PF.write('%25s\n' %getKbV(laD,arestas.GetSourceSideMinCut()[k])[0])
        return
    #
    # main
    #
    # read table values
    nfileCSV = '../Data/links-consolidado.csv'
    tabular = pd.read_csv(nfileCSV, header=0, sep=';')
    # create lists
    origemL = list(tabular.Origem)
    destinoL = list(tabular.Destino)
    mbpsL = list(tabular.Mbps)
    # make equivalent lists with numbers associated to each string in origemL and destinoL,
    # except mbpsL, which just contains numbers.
    labelDic = {}
    origemLN = []
    destinoLN = []
    labelDic, origemLN, destinoLN = conversion(origemL, destinoL)
    # Check if origemLN and destinoLN have the same lenght
    if len(origemLN) != len(destinoLN):
        sys.exit('Different sizes of lenght in both sets!\n')
    else:
        print('Lenght of list = %d' %len(origemLN))
    # Create all edges (i to j, j to i) with capacities (Mbps)
    edges = pywrapgraph.SimpleMaxFlow()
    for i in range(0, len(origemLN)):
        edges.AddArcWithCapacity(origemLN[i], destinoLN[i], mbpsL[i])
    #
    # Analysing MaxFlows - an extensive routine by Google Developers
    #
    # First, make a list without repetition with labels from origemLN and destinoLN
    vortexN = norepeat(origemLN, destinoLN)
    # Second, create a file with results
```

```
resultsFN = '../Results/results.txt'
resultsFp = open(resultsFN, 'w')
# Third, make a dictionary with betweenness centrality degree
# zero-values now
degreeD = {}
for i in range(0, len(origemL)):
    degreeD[i] = 0
# Forth, make the analysis
flowanalysis(vortexN, edges, labelDic, degreeD, resultsFp)
#
# Register the betweenness centrality degree in file results.txt
resultsFp.write('\n\n%-16s:%6s\n\n' %('Vortex', 'Degree'))
for i in range(len(origemLN)):
    if degreeD[i] != 0:
        resultsFp.write('%-16s:%6d\n' %(getKbV(labelDic,i)[0],degreeD[i]))
resultsFp.close()
# create graphviz schemes
gvNF = '../Results/results.gv'
gvPF = open(gvNF, 'w')
network = Graph()
for j in origemLN:
    labelR = getKbV(labelDic,j)[0] + ' - D=' + str(degreeD[j])
    network.node(getKbV(labelDic,j)[0], labelR)
for k in range(len(origemLN)):
    V1 = getKbV(labelDic,origemLN[k])[0]
    V2 = getKbV(labelDic,destinoLN[k])[0]
    if mbpsL[k]>=10000:
        network.edge(V1,V2,penwidth='5')
    else:
        network.edge(V1,V2)
network.graph_attr['rankdir'] = 'LR'
gvPF.write(network.source)
gvPF.close()
```

C - Transcrição do arquivo com o Markup Language para geração do grafo

A Figura 2 foi gerada a partir do *script* abaixo em *markup language*, utilizando o comando (sistema operacional Linux):

```
dot -Tpng <arquivo markup language> -O
```

A opção `-Tpng` representa o formato do resultado que é, no caso, uma figura em PNG.

Por sua vez, o *script* abaixo foi gerado a partir do *script* em Python transcrito no Apêndice - B.

```
graph {
 graph [rankdir=LR]
 IOT01 [label="IOT01 - D=51"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 AGG [label="AGG - D=2525"]
 SW01SS [label="SW01SS - D=151"]
 SW01TA [label="SW01TA - D=151"]
 SW01TB [label="SW01TB - D=151"]
 SW01TC [label="SW01TC - D=343"]
 SW01TC [label="SW01TC - D=343"]
 SW01_AMB_SUS_TC [label="SW01_AMB_SUS_TC - D=151"]
 SW011B [label="SW011B - D=339"]
 SW021B [label="SW021B - D=249"]
 SW021B [label="SW021B - D=249"]
 SW012A [label="SW012A - D=151"]
 SW012C [label="SW012C - D=151"]
 SW013A [label="SW013A - D=345"]
 SW013A [label="SW013A - D=345"]
 SW013A [label="SW013A - D=345"]
 SW013B [label="SW013B - D=345"]
 SW013B [label="SW013B - D=345"]
 SW013B [label="SW013B - D=345"]
 SW015A [label="SW015A - D=151"]
 SW015B [label="SW015B - D=345"]
 SW015B [label="SW015B - D=345"]
 SW015B [label="SW015B - D=345"]
 SW017B [label="SW017B - D=151"]
 SW018AB [label="SW018AB - D=151"]
 AGG [label="AGG - D=2525"]
 SW01SS [label="SW01SS - D=151"]
 SW011A [label="SW011A - D=51"]
 SW012A [label="SW012A - D=151"]
 SW013A [label="SW013A - D=345"]
 SW014A [label="SW014A - D=51"]
 SW015A [label="SW015A - D=151"]
 SW01CC [label="SW01CC - D=51"]
 SW017A [label="SW017A - D=51"]
 SW01TA [label="SW01TA - D=151"]
 SW01SSA [label="SW01SSA - D=51"]
 SW01FSS [label="SW01FSS - D=51"]
 WC01 [label="WC01 - D=51"]
```

```
SW011B [label="SW011B - D=339"]
SW012B [label="SW012B - D=51"]
SW013B [label="SW013B - D=345"]
SW014B [label="SW014B - D=51"]
SW015B [label="SW015B - D=345"]
SW017B [label="SW017B - D=151"]
SW018AB [label="SW018AB - D=151"]
SW01TB [label="SW01TB - D=151"]
SW02SS [label="SW02SS - D=51"]
SW01TC [label="SW01TC - D=343"]
SW011C [label="SW011C - D=51"]
SW012C [label="SW012C - D=151"]
SW01ALMOXSS [label="SW01ALMOXSS - D=51"]
SW01PSSS [label="SW01PSSS - D=51"]
SW01TD [label="SW01TD - D=51"]
AP01SSA [label="AP01SSA - D=51"]
SW02TA [label="SW02TA - D=51"]
SW02_TB [label="SW02_TB - D=51"]
SW01_AMB_SUS_TC [label="SW01_AMB_SUS_TC - D=151"]
SW02TC [label="SW02TC - D=51"]
SW01_AMB_TC [label="SW01_AMB_TC - D=51"]
SW021B [label="SW021B - D=249"]
ap011b [label="ap011b - D=51"]
ap021b [label="ap021b - D=51"]
SW02_2A [label="SW02_2A - D=51"]
SW02_2C [label="SW02_2C - D=51"]
ap02_3a [label="ap02_3a - D=51"]
SW023A [label="SW023A - D=51"]
SW033A [label="SW033A - D=51"]
SW033B [label="SW033B - D=51"]
SW043B [label="SW043B - D=51"]
SW023B [label="SW023B - D=51"]
SW025A [label="SW025A - D=51"]
SW02_5B [label="SW02_5B - D=51"]
SW03_5B [label="SW03_5B - D=51"]
AP015B [label="AP015B - D=51"]
SW027B [label="SW027B - D=51"]
AP018B [label="AP018B - D=51"]
IOT01 -- AGG [penwidth=5]
AGG -- SW011A [penwidth=5]
AGG -- SW012A [penwidth=5]
AGG -- SW013A [penwidth=5]
AGG -- SW014A [penwidth=5]
AGG -- SW015A [penwidth=5]
AGG -- SW01CC [penwidth=5]
AGG -- SW017A [penwidth=5]
AGG -- SW01TA [penwidth=5]
AGG -- SW01SS [penwidth=5]
AGG -- SW01SSA [penwidth=5]
AGG -- SW01FSS
AGG -- WC01
AGG -- SW011B [penwidth=5]
AGG -- SW012B [penwidth=5]
AGG -- SW013B [penwidth=5]
AGG -- SW014B [penwidth=5]
AGG -- SW015B [penwidth=5]
AGG -- SW017B [penwidth=5]
AGG -- SW018AB [penwidth=5]
AGG -- SW01TB [penwidth=5]
AGG -- SW02SS [penwidth=5]
AGG -- SW01TC [penwidth=5]
AGG -- SW011C [penwidth=5]
AGG -- SW012C [penwidth=5]
AGG -- SW01ALMOXSS
AGG -- SW01PSSS [penwidth=5]
AGG -- SW01TD [penwidth=5]
SW01SS -- AP01SSA
SW01TA -- SW02TA
SW01TB -- SW02_TB
SW01TC -- SW01_AMB_SUS_TC
SW01TC -- SW02TC
SW01_AMB_SUS_TC -- SW01_AMB_TC
SW011B -- SW021B
SW021B -- ap011b
SW021B -- ap021b
SW012A -- SW02_2A
SW012C -- SW02_2C
SW013A -- ap02_3a
SW013A -- SW023A
SW013A -- SW033A
SW013B -- SW033B
SW013B -- SW043B
SW013B -- SW023B
SW015A -- SW025A
SW015B -- SW02_5B
SW015B -- SW03_5B
SW015B -- AP015B
```

```
SW017B -- SW027B
SW018AB -- AP018B
AGG -- IOT01 [penwidth=5]
SW01SS -- AGG [penwidth=5]
SW011A -- AGG [penwidth=5]
SW012A -- AGG [penwidth=5]
SW013A -- AGG [penwidth=5]
SW014A -- AGG [penwidth=5]
SW015A -- AGG [penwidth=5]
SW01CC -- AGG [penwidth=5]
SW017A -- AGG [penwidth=5]
SW01TA -- AGG [penwidth=5]
SW01SSA -- AGG [penwidth=5]
SW01FSS -- AGG
WC01 -- AGG
SW011B -- AGG [penwidth=5]
SW012B -- AGG [penwidth=5]
SW013B -- AGG [penwidth=5]
SW014B -- AGG [penwidth=5]
SW015B -- AGG [penwidth=5]
SW017B -- AGG [penwidth=5]
SW018AB -- AGG [penwidth=5]
SW01TB -- AGG [penwidth=5]
SW02SS -- AGG [penwidth=5]
SW01TC -- AGG [penwidth=5]
SW011C -- AGG [penwidth=5]
SW012C -- AGG [penwidth=5]
SW01ALMOXSS -- AGG
SW01PSSS -- AGG [penwidth=5]
SW01TD -- AGG [penwidth=5]
AP01SSA -- SW01SS
SW02TA -- SW01TA
SW02_TB -- SW01TB
SW01_AMB_SUS_TC -- SW01TC
SW02TC -- SW01TC
SW01_AMB_TC -- SW01_AMB_SUS_TC
SW021B -- SW011B
ap011b -- SW021B
ap021b -- SW021B
SW02_2A -- SW012A
SW02_2C -- SW012C
ap02_3a -- SW013A
SW023A -- SW013A
SW033A -- SW013A
SW033B -- SW013B
SW043B -- SW013B
SW023B -- SW013B
SW025A -- SW015A
SW02_5B -- SW015B
SW03_5B -- SW015B
AP015B -- SW015B
SW027B -- SW017B
AP018B -- SW018AB
}
```