CLAUDIO AUGUSTO GRÜNEWALD SOARES

IDENTIFICAÇÃO DE INDÍCIOS DE IRREGULARIDADE POR MEIO DE PADRÕES TOPOLÓGICOS EM GRAFOS:

Aplicação ao banco de vínculos da CGU

Brasília

2020

CLAUDIO AUGUSTO GRÜNEWALD SOARES

IDENTIFICAÇÃO DE INDÍCIOS DE IRREGULARIDADE POR MEIO DE PADRÕES TOPOLÓGICOS EM GRAFOS:

Aplicação ao banco de vínculos da CGU

Trabalho de conclusão do curso de pós-graduação *lato sensu* em Análise de Dados para o Controle realizado pela Escola Superior do Tribunal de Contas da União como requisito para a obtenção do título de especialista.
Orientador: Gustavo van Erven, M. Sc.

Brasília

2020

REFERÊNCIA BIBLIOGRÁFICA

SOARES, Claudio A. G. **Identificação de indícios de irregularidade por meio de padrões topológicos em grafos**: Aplicação ao banco de vínculos da CGU. 2020. Trabalho de Conclusão de Curso (Especialização em Análise de Dados para o Controle) – Escola Superior do Tribunal de Contas da União, Instituto Serzedello Corrêa, Brasília DF. 59 fl.

CESSÃO DE DIREITOS

NOME DO AUTOR: Claudio Augusto Grünewald Soares.
TÍTULO: Identificação de indícios de irregularidade por meio de padrões topológicos em grafos.
GRAU/ANO: Especialista/2020.

_____

Claudio Augusto Grünewald Soares
claudioags@gmail.com

Ficha catalográfica

Soares, Claudio Augusto Grünewald.

Identificação de indícios de irregularidade por meio de padrões topológicos em grafos: Aplicação ao Banco de Vínculos da CGU / Claudio Augusto Grünewald Soares; orientador, Gustavo van Erven, 2020. 118 p.

Trabalho de Conclusão de Curso (pós-graduação) - Instituto Serzedelo Corrêa, Especialização em Análise de Dados para o Controle, 2020.

Inclui referências.

1. Mineração de dados; vínculo; grafo; topologia em grafo; relacionamento de entidades; ciclo; circuito; descoberta de padrão; indício de irregularidade; regra de associação, licitação, compra. I. Erven, Gustavo. II. Instituto Serzedelo Corrêa, Especialização em Análise de Dados para o Controle. III. Identificação de indícios de irregularidade por meio de padrões topológicos em grafos.

CLAUDIO AUGUSTO GRÜNEWALD SOARES

IDENTIFICAÇÃO DE INDÍCIOS DE IRREGULARIDADE POR MEIO DE PADRÕES TOPOLÓGICOS EM GRAFOS:

Aplicação ao banco de vínculos da CGU

Trabalho de conclusão do curso de pós-graduação lato sensu em Análise de Dados para o Controle realizado pela Escola Superior do Tribunal de Contas da União como requisito para a obtenção do título de especialista.

Brasília, 27 de março de 2020.

Banca Examinadora:

_____

Prof.ª Gustavo van Erven, M. Sc.

Orientador

_____

Prof. Eduardo Chaves Ferreira, Dr.

Instituto Serzedelo Corrêa

AGRADECIMENTOS

À minha família, pela paciência e compreensão em vista das incontáveis horas em frente ao computador.

Ao meu orientador, pelo conhecimento compartilhado e pelo entusiasmo nos projetos desenvolvidos em conjunto.

Ao pessoal do Observatório da Despesa Pública da CGU, pelo insistente incentivo e pela companhia no esforço para melhoria constante do serviço público.

Aos organizadores e instrutores do curso de pós-graduação em análise de dados do ISC/TCU, pela iniciativa, pela dedicação, pelo altíssimo nível do curso e principalmente pela oportunidade oferecida aos colegas da CGU.

"I learned very early the difference between knowing the name of something and knowing something". Richard P. Feynman, 1988.

"The price of training is always a certain 'trained incapacity': the more we know how to do something, the harder it is to learn to do it differently". Abraham Kaplan, 1964

RESUMO

Relações entre entidades podem ser representadas de forma generalizada por meio de uma abstração chamada grafo. A topologia formada no grafo pode ajudar a revelar informações de interesse para a auditoria, em especial quando aparecem padrões cíclicos. Este trabalho se inicia pela busca de uma situação-alvo, que apareceria como um ciclo no grafo, e segue desenvolvendo melhorias para permitir a execução dessa busca em tempo viável. A situação-alvo escolhida poderia indicar simulação de concorrência em processos de compras governamentais. Ao longo do desenvolvimento, fica demonstrado que a busca da situação-alvo proporciona o levantamento de diversas outras situações de vínculos similares sem que haja necessidade de especificação prévia. Após a avaliação de todos os processos de compras do Poder Executivo Federal em duas Unidades da Federação no ano de 2018, há uma sugestão para o uso de regras de associação para priorização dos casos a auditar.

Palavras-chave: Mineração de dados; vínculo; grafo; topologia em grafo, relacionamento de entidades; ciclo, circuito, descoberta de padrão; indício de irregularidade; regra de associação, licitação, processo de compra.

ABSTRACT

Relationships between entities can be represented in a generalized way through an abstraction named graph. The topology formed in the graph can help to reveal information of interest to the audit, especially when cyclical patterns appear. This work begins with the search for a target situation, which would appear as a cycle in the graph, and continues to develop improvements to allow the execution of this search in a viable time. The target situation chosen could indicate simulation of competition in government procurement processes. Throughout the development, it is demonstrated that the search for the target situation provides the survey of several other similar linking situations without the need for prior specification. After evaluating all the purchasing processes of the Federal Executive Branch in two Federal States in 2018, there is a suggestion of using association rules to prioritize the cases to be audited.

Keywords: Data mining; link; graph; graph topology; entity relationship; cycle; circuit; pattern discovery; indication of irregularity; association rule, bidding, procurement process.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CGU	Controladoria Geral da União
ODP	Observatório da Despesa Pública
SIAPE	Sistema de Administração de Pessoal
SIASG	Sistema Integrado de Administração de Serviços Gerais
DW-SIASG	Data-Warehouse do Sistema Integrado de Administração de Serviços Gerais
TCU	Tribunal de Contas da União
ISC	Instituto Serzedello Corrêa
CPF	Cadastro de Pessoa Física
CNPJ	Cadastro Nacional de Pessoa Jurídica
NIS	Número de Identificação Social
SQL	Linguagem Estruturada de Consulta (*Structured Query Language*)
UASG	Unidade de Administração de Serviços Gerais
UF	Unidade da Federação
AC	Acre
DF	Distrito Federal
MG	Minas Gerais
RJ	Rio de Janeiro

SUMÁRIO

1 INTRODUÇÃO

Quaisquer interações entre pessoas, empresas, órgãos públicos e outras entidades do mundo "real" podem ser representadas em sua essência por meio de uma abstração denominada grafo. O grafo é composto de dois conjuntos: o conjunto das entidades (vértices do grafo), e o conjunto que representa as relações entre elas (arestas do grafo).

O Observatório da Despesa Pública[1] (ODP) da Controladoria Geral da União (CGU) criou e mantém um banco de dados que armazena os diversos relacionamentos entre as entidades encontrados nos diversos sistemas governamentais, como por exemplo, o parentesco declarado entre as pessoas no Sistema de Administração de Pessoal (SIAPE). Este banco de dados armazena um grafo com mais de um bilhão e meio de relacionamentos e é chamado genericamente de "banco de vínculos da CGU" ou referido pelo nome do sistema de consulta a estes vínculos: Yggdrasil[2].

O uso de grafo é prático por armazenar informações de fontes heterogêneas com uma formatação padronizada e uma representação uniforme, mesmo para vínculos tão distintos como "parentesco" ou "participação em processo licitatório". Com isso se diminui muito a quantidade de códigos computacionais diferentes necessários para a análise em conjunto das informações. A aplicação em grafos cada vez maiores, entretanto, rapidamente atinge um volume em que os algoritmos existentes, por conta de sua complexidade[3] de aumento exponencial [9], ficam quase inviabilizados.

O presente trabalho aborda o problema do resgate de situações relevantes – que possam indicar necessidade de apuração de irregularidades – a partir desse grafo de grande porte do banco de vínculos, considerando como pressuposto que essa informação pode ser encontrada por meio do padrão topológico dos relacionamentos.

Para isso, foi escolhida uma situação-alvo em uma área de grande relevância para a administração pública federal: as compras governamentais.

---

[1] https://www.gov.br/cgu/pt-br/assuntos/informacoes-estrategicas/observatorio-da-despesa-publica

[2] Yggdrasil: da mitologia nórdica a "árvore que se estende por todos os mundos", uma referência ao fato da representação dos vínculos lembrar os ramos de uma árvore.

[3] Complexidade computacional: termo técnico que expressa o número de operações elementares que precisam ser realizadas para completar um algoritmo, ligado ao tempo de execução e ao consumo de recursos de processamento e memória.

De acordo com os dados disponíveis no "Novo Painel de Compras do Governo Federal" do Ministério da Economia[4], no ano de 2019 foram homologados 178.584 processos de compras, representando um valor de R$ 104 bilhões de reais.

Para proteger o interesse público na gestão de processos que comprometem recursos nesse montante não faltam dispositivos legais e normativos, tais como a Lei 8.666/93 e a Lei 10.520/2002. Não obstante, fraudes como a "simulação de concorrência" pelas licitantes ainda podem ocorrer [4], tendo motivado a escolha da situação-alvo.

Ao longo do trabalho a situação-alvo será examinada, pressupondo a aparente inexistência de vantagem em duas ou mais empresas (representadas por seus respectivos CNPJs) pertencentes a uma mesma pessoa física (representada por um CPF) "concorrerem" no mesmo item de compra em uma licitação.

As ferramentas utilizadas para o exame serão as técnicas aplicáveis aos grafos e o uso de regras de associação para análise das relações entre os conjuntos encontrados nos resultados.

[4] https://comprasgovernamentais.gov.br/index.php/painel-de-compras-de-governo

2 PROBLEMA E JUSTIFICATIVA

Para melhor compreensão do contexto, partiremos da descrição do ambiente em que o trabalho se desenvolveu para demonstrar que a necessidade do estudo do tema emergiu naturalmente a partir das informações e infraestrutura disponível.

2.1 O ambiente corporativo

Como órgão de controle interno, a CGU possui como missão:

> Promover o aperfeiçoamento e a transparência da gestão pública, a prevenção e o combate à corrupção, com participação social, por meio da avaliação e controle das políticas públicas e da qualidade do gasto.

Em auxílio à missão da CGU, o ODP se coloca como uma unidade voltada à aplicação de metodologia científica, apoiada em tecnologia da informação de ponta, para a produção de informações que visam a subsidiar e a acelerar a tomada de decisões estratégicas, por meio do monitoramento dos gastos públicos.

O objetivo do ODP é contribuir para o aprimoramento do controle interno e funcionar como ferramenta de apoio à gestão pública, e os resultados gerados pela unidade servem como insumo para realização de auditorias e fiscalizações conduzidas pela CGU, bem como para informar aos gestores sobre indicadores gerenciais relativos à realização de gastos públicos, de modo a permitir análises comparativas, subsidiando a tomada de decisões para melhoria da aplicação dos recursos públicos.

2.2 As informações disponíveis

O ODP atualmente hospeda cópia de mais de cento e cinquenta bases de dados oriundas de diversas entidades do poder executivo federal. Com esses dados, um dos principais trabalhos é o desenvolvimento de "trilhas de auditoria", nome dado internamente na CGU ao processo de descoberta de fatos por meio de cruzamento de informações entre bases de dados.

Ocorre que muitas vezes a mesma informação, ou informações muito similares, estão armazenadas nessas bases de dados, mas estruturadas em formatos ou padrões completamente diferentes, o que torna o desenvolvimento de trilhas abrangentes um desafio de enorme complexidade.

É necessário que o auditor imagine todos os conjuntos de relacionamentos que ocorreriam em caso de ocorrência de alguma irregularidade[5] para que estes conjuntos sejam procurados por meio do cruzamento de informações dos bancos de dados, sendo então extraídos os casos reais onde os conjuntos de relacionamentos ocorreram exatamente da forma como imaginado, cada conjunto exigindo um programa de computador específico, ou uma "trilha".

Desse desafio nasceu a ideia da abstração das diversas informações em termos de relações entre pares de entidades genéricas, materializando o que é conhecido por grafo na teoria matemática.

As informações de interesse são extraídas em um mapeamento de entidades e vínculos entre elas, armazenando uma representação em grafo dos diversos relacionamentos que se apresentavam em outras estruturas distintas e, por vezes, incompatíveis.

Ao longo deste trabalho, o conjunto dessas informações mapeadas em grafo será denominado "banco de vínculos" e representa tanto a ideia dos dados em conjunto quanto o seu armazenamento, de fato, em um sistema de banco de dados.

2.3 O banco de vínculos

O sistema que opera o banco de vínculos do ODP é chamado Yggdrasil e conta atualmente com 1,6 bilhão de relacionamentos mapeados entre 600 milhões de entidades, tais como pessoas, empresas, itens de compra do governo federal, entre outras.

No Yggdrasil, atualmente, estão disponibilizadas apenas pesquisas de dois tipos: expandir o grafo a partir dos vizinhos[6] de uma entidade e encontrar o menor caminho[7] de relacionamentos que conecta duas entidades.

Se conhecermos previamente um dos relacionamentos, utilizando a funcionalidade de encontrar o menor caminho é possível detectar algumas configurações de padrões topológicos cíclicos de especial interesse para o trabalho de controle, uma vez que podem indicar a existência de indícios de irregularidades.

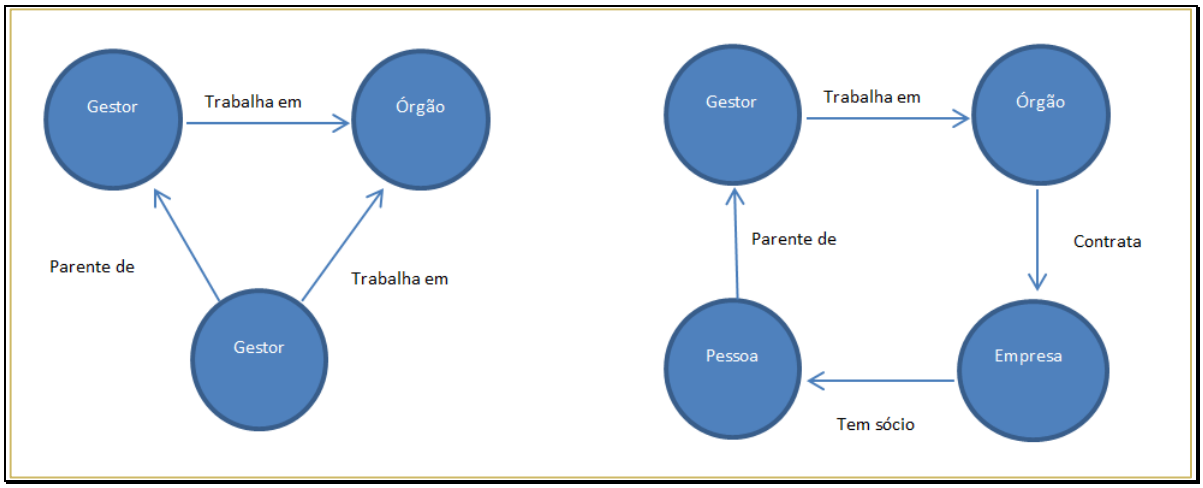
Por exemplo, se temos a lista prévia de quais são os gestores de cada Órgão, podemos tentar encontrar um caminho entre o Órgão e o gestor passando por uma contratação, mapeada no banco de vínculos.

---

[5] Entendida aqui genericamente como um desvio de conduta em relação a normativo.
[6] "Vizinho", conforme definição em 2.1.
[7] "Caminho", idem.

Dois exemplos de aplicação desta técnica no Yggdrasil estão ilustrados abaixo: detecção de nepotismo (vínculo de parentesco entre superior e subordinado, nas condições previstas em normativo próprio); detecção de indício de favorecimento em procedimento licitatório (vínculo entre gestor da área contratante e sócio da empresa contratada).

Figura 1 – Padrões topológicos cíclicos.



Fonte: Elaborada pelo autor (2020).

Como citado antes, a detecção dos casos em que ocorrem esses padrões não é difícil de ser realizada no próprio banco de dados, sem a necessidade de um banco de vínculos, desde que o exato padrão procurado seja conhecido de antemão, ou seja, a especificação de cada tipo de relacionamento, de cada tipo de entidade, de cada tabela (e o respectivo banco de dados) onde a informação é armazenada.

Com o aumento do "caminho" entre as entidades, a quantidade de informações a considerar simultaneamente aumenta em progressão geométrica, caso se queira gerar um código de "trilha" em SQL[8] que detecte todas as possibilidades [8].

Tomemos o padrão cíclico da contratação ilustrado acima: a pessoa pode ser representada por um CPF, NIS ou número de SIAPE, que estariam armazenados em formatos e tabelas diferentes; a relação de parentesco poderia ser "pai", "mãe", "irmão" e poderia estar em qualquer uma das bases disponíveis (Cadastro único, SIAPE, etc.); a relação entre a pessoa física e a empresa contratada poderia ser "sócia", "responsável", "contador", cada uma delas encontrada em uma tabela diferente. Um código que considerasse todas as opções fica-

[8] Linguagem estruturada de consulta a banco de dados (*Structured Query Language*)

ria rapidamente ilegível, e na prática o que acontece é que várias hipóteses deixam de ser testadas nas trilhas [8].

Com as informações mapeadas de forma padronizada, em um grafo, há algumas opções de algoritmos disponíveis em bibliotecas especializadas para recuperar certos elementos ou propriedades desse grafo. Uma das possibilidades, na qual o presente trabalho se baseou, é a detecção de circuitos em um grafo, uma vez que a partir deles os padrões cíclicos podem ser encontrados.

Entretanto, a aplicação de algoritmos de grafos em grandes volumes de dados, tais como do banco de vínculos, tem severas limitações por conta da complexidade computacional, que em geral é de ordem exponencial para esses algoritmos [9].

O *hardware* disponível para o projeto não é modesto: o servidor dedicado somente ao banco de dados tem quatro processadores Intel Xeon E5-4628L 1.8GHz, cada qual com 56 núcleos, e 1TB de memória de RAM. O servidor utilizado para a aplicação em *python* tem as mesmas especificações e embora seja compartilhado com diversas outras aplicações esteve praticamente disponível em sua totalidade para este projeto.

O que nos leva a seguinte questão:

- Seria computacionalmente viável, com o equipamento disponível, a identificação de indícios de "simulação de concorrência" por meio da investigação da topologia do grafo no banco de vínculos da CGU, em especial dos padrões cíclicos, encontrando resultados relevantes?

3 OBJETIVOS

3.1 Objetivo geral

- Identificação de situações de interesse para o trabalho de controle por meio da análise topológica do banco de vínculos.

3.2 Objetivos específicos

- Utilizar algoritmos de busca de circuitos em grafos[9] para identificar topologias que possam indicar indícios de "simulação de concorrência" em processos de compras;

- Aplicação de métricas de regras de associação[10] para priorizar a análise dos casos encontrados.

[9] Conforme 5.2
[10] Conforme 5.3

4 METODOLOGIA PROPOSTA

O presente trabalho pode ser dividido para melhor compreensão em algumas etapas lógicas, como proposto a seguir. Cabe comentar que as etapas não foram rigorosamente separadas e que várias das tarefas propostas em cada etapa acabaram acontecendo de forma transversal ao longo do projeto.

4.1 Preparação

Esta é a etapa inicial do trabalho, desenvolvida concomitantemente à etapa subsequente. Cada nova solução que se fez necessária, por exemplo, demandou novas pesquisas bibliográficas.

As atividades desenvolvidas entendidas como "preparatórias" foram:

- Especificação da situação "alvo" a ser procurada;
- Confirmar a existência de versão simplificada da situação "alvo" por meio das ferramentas tradicionais de bancos de dados relacionais;
- Pesquisa de trabalhos com temática similar para elencar limitações já encontradas e soluções propostas;
- Levantamento bibliográfico de soluções existentes de algoritmos de extração de informações topológicas.

4.2 Programação

Esta etapa foi a que mais consumiu o tempo do projeto. Contrariando a expectativa inicial de se apoiar exclusivamente em bibliotecas do *python*[11] e soluções disponíveis no Yggdrasil, as especificidades do projeto demandaram extensa programação em *python* e SQL, considerando:

- Fazer a busca da situação-alvo simplificada no banco de vínculos, por meio de algoritmos próprios para extração de informações da topologia do grafo, propondo soluções para as eventuais barreiras encontradas;
- Analisar a viabilidade das soluções, em especial quanto ao tempo computacional necessário;

[11] https://www.python.org/

- Analisar as variações da situação-alvo encontradas, evidenciando vantagem na abordagem proposta se for possível encontrar situações similares à procurada buscando minimizar a necessidade de escrita de código específico;
- Avaliar possibilidade de generalização da melhor solução para aplicação futura em casos diversos.

4.3 Experimento

Esta é a fase operacional do trabalho, onde foram aproveitadas as rotinas de automação e iniciadas as pesquisas dos dados que seriam objeto de análise, compreendendo:

- Aplicação do modelo desenvolvido;
- Seleção do escopo conforme resultados obtidos.

4.4 Avaliação

A fase final do trabalho criou uma visualização de cenário com base nos dados obtidos, traduzindo uma grande quantidade de resultados em uma síntese apropriada para análise, buscando:

- Avaliar a relevância e recorrência da situação-alvo detectada;
- Descartar “eventualidades”, com auxílio de regras de associação.

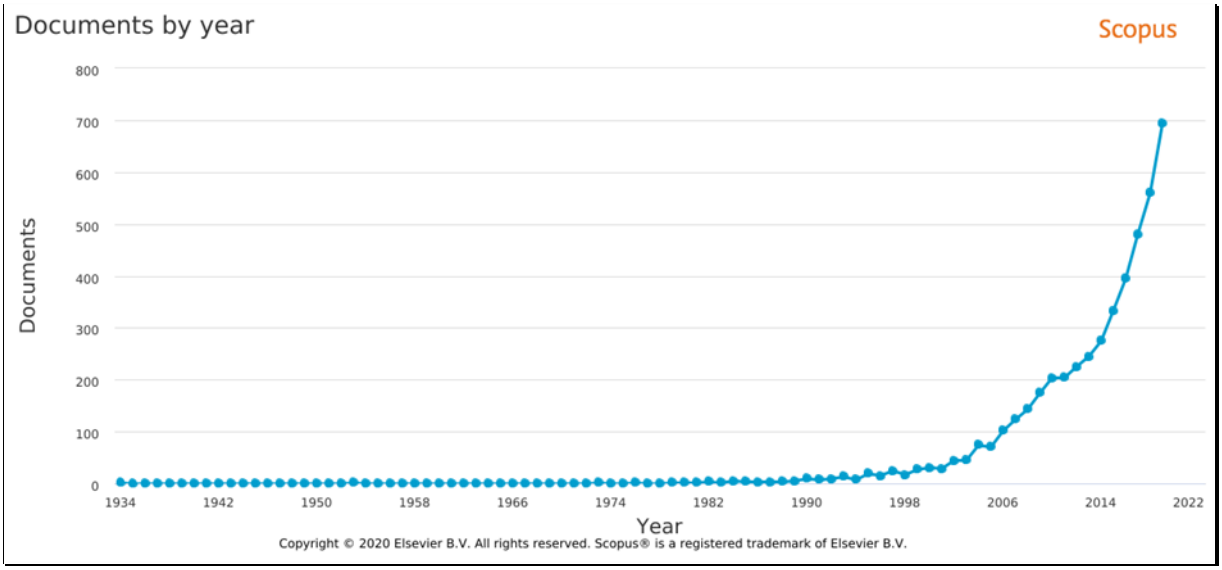
5 FUNDAMENTAÇÃO TEÓRICA

A seguir são apresentados os diversos conceitos abordados durante o desenvolvimento dos trabalhos.

5.1 Trabalhos relacionados

A busca de irregularidades e fraudes não é um tema recente. Vários trabalhos, de diversas áreas, já se dedicaram em elaborar ou avaliar novas abordagens e ferramentas nesse tema. Considerando apenas o indexador SCOPUS[12], a busca apenas por "*fraud detection*" já retornou 4.618 resultados, distribuídos em várias áreas de conhecimento, e com crescimento acentuado nos últimos anos, como apresentado nas Figuras 2 e 3.

Figura 2 – Quantidade de documentos com citação de "*fraud detection*".

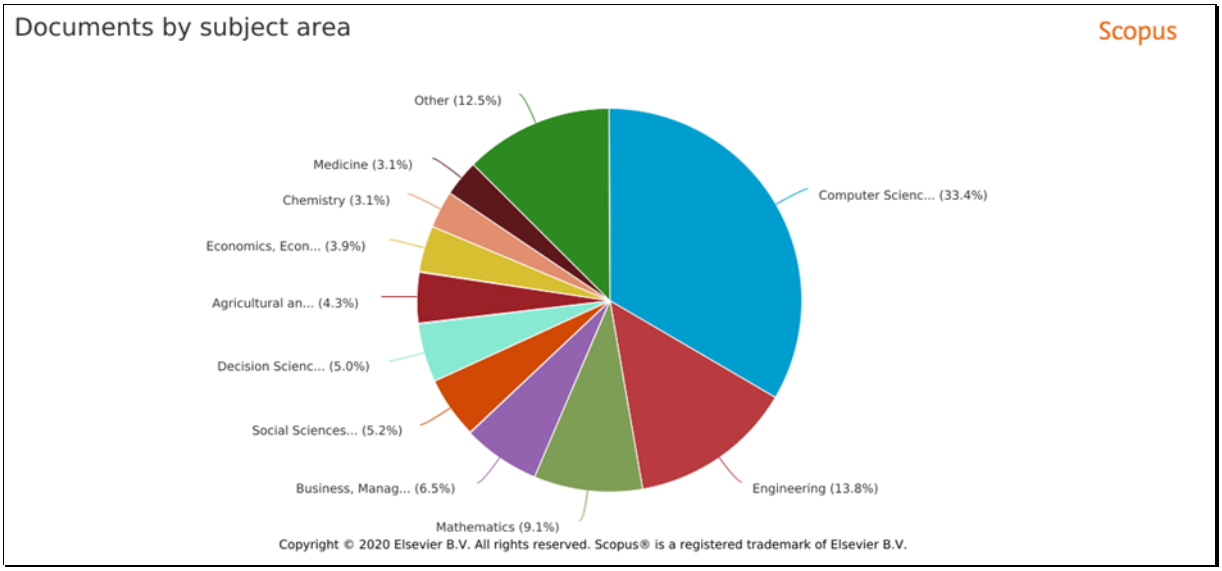


Fonte: Scopus (consultado em 15/03/2020).

Pela distribuição da Figura 3, percebe-se que a ciência da computação apresenta uma área de pesquisa intensa sobre o tema. Provavelmente o tema tem grande representação nesse grupo por depender muitas vezes de análises de dados, em geral financeiros e de perfil de usuários, e pelos avanços, em especial na área de Inteligência Artificial e Algoritmos de Busca, que permitem uma melhor identificação de padrões, previsões e anomalias [14, 15].

[12] http://www.scopus.com

Figura 3 – Área temática de documentos com citação de "*fraud detection*".



Fonte: Scopus (consultado em 15/03/2020).

A análise dos dados das compras públicas no Brasil também tem sido tema encontrado em diversos trabalhos, por exemplo, com proposta do uso das bases de dados para criação de modelos para seleção de casos para auditoria [12].

Considerando a natureza de eventos e relacionamentos das entidades envolvidas nas fraudes, o uso de grafos para descoberta de irregularidades também vem sendo aplicado em diversos trabalhos, como na modelagem e análise de dados de licitação [8, 11], detecção de anomalias [13] ou identificação de evasão de impostos utilizando algoritmos de detecção de ciclos [10].

Como neste último, o foco do presente trabalho também é a detecção de ciclos, neste caso aplicada às compras governamentais para seleção de casos para auditoria. Serão apresentados os desafios, soluções e resultados específicos a esse domínio, mas sempre mantendo em vista o objetivo de obter soluções (algoritmos) que possam ser aproveitadas futuramente para pesquisas em outros temas no banco de vínculos.

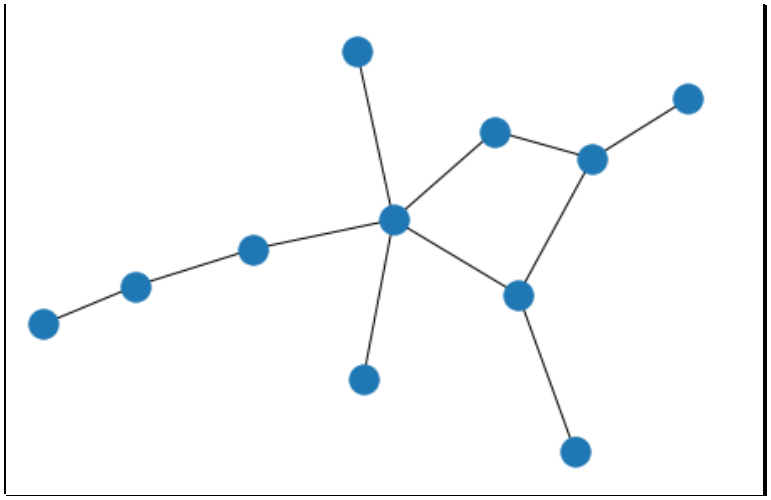
5.2 Grafos

Um grafo é uma abstração para representação genérica de um conjunto de elementos e as relações entre eles. Pode ser definido matematicamente e assim ter suas propriedades definidas de forma padronizada, o que facilita sua modelagem e o estudo de suas propriedades. A seguir, são apresentados alguns conceitos e definições conforme Goldbarg [2].

Graficamente, um grafo pode ser apresentado de como um conjunto de “pontos” ou “nós” (quase sempre representados como círculos) chamados **vértices** conectados por “linhas” ou “setas” chamadas **arestas** que representam quaisquer relações entre aqueles elementos. O conjunto dos vértices de um grafo em geral é denominado “V” e o conjunto das arestas “E” (do inglês, *Edge*). Matematicamente se representa o grafo como G = (V, E), isto é, o grafo G é definido como o conjunto dos vértices e arestas.

Vértices e arestas podem ter seus próprios atributos, os mais notáveis costumam ser um nome (uma designação) para o vértice e um tipo para a aresta. Dessa forma se representa relacionamentos como “A é sócio de B” ou “A trabalha na empresa C”.

Figura 4 – Grafo simples.

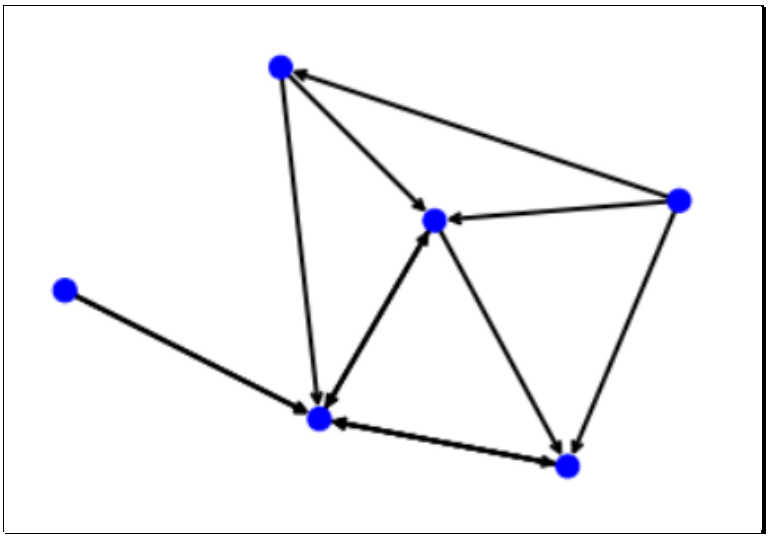


Fonte: Elaborada pelo autor (2020).

Quando a informação da direção da aresta é importante, o grafo é chamado de direcionado. A aresta do grafo direcionado é representada por uma seta e pode estar referenciada pelo nome de arco.

O grafo no qual a direção da aresta não é considerada é simplesmente chamado de **não direcionado**. Um grafo com arestas e arcos é um **grafo misto**.

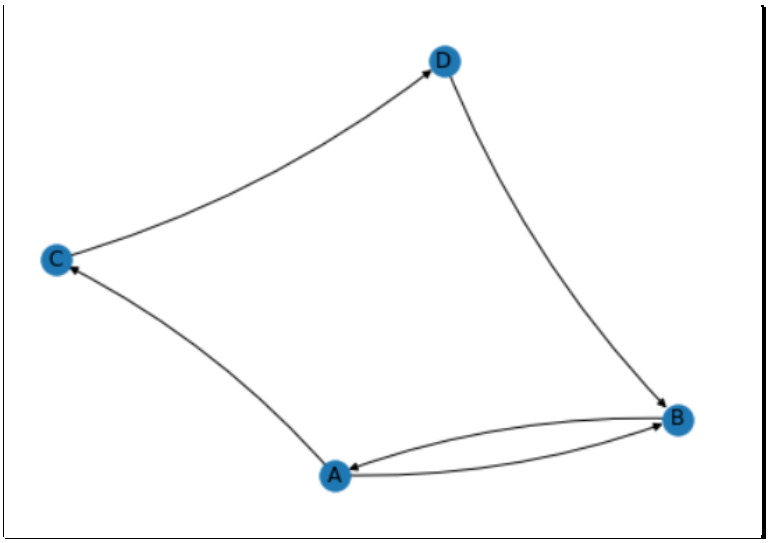
Figura 5 – Grafo direcionado.



Fonte: Elaborada pelo autor (2020).

Uma aresta sem direção definida pode ser considerada "equivalente" a dois arcos, um em cada direção, se necessário. Quando o grafo admite mais de uma aresta – direcionada ou não – entre um par de vértices ele é chamado **multigrafo**, se não admite ele é um **grafo simples**.

Figura 6 – Multigrafo direcionado.



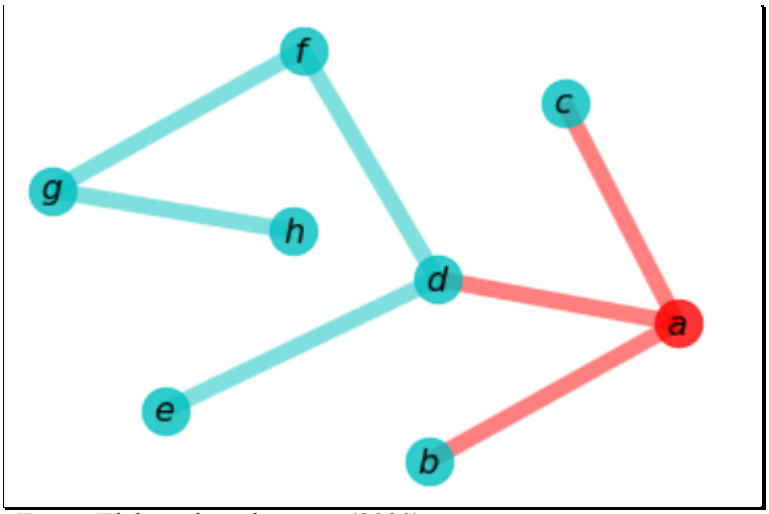
Fonte: Elaborada pelo autor (2020).

O arco e a aresta são representados pelo par de vértices que se conectam, lembrando que no arco é um **par ordenado**, a ordem dos vértices vai indicar a direção da relação representada. $E_i = (v_j, v_k)$, ou $E_i = (A,B)$, por exemplo.

Quando há uma aresta entre dois vértices eles são **adjacentes** ou **vizinhos**, quando há um arco entre eles são chamados **antecessor** e **sucessor**, conforme a direção do arco.

Arestas são **adjacentes** quando compartilham o mesmo vértice. O **grau de um vértice** se refere ao seu número de arestas adjacentes (em grafos orientados há um grau para arestas que chegam e um para as que saem do vértice – isso não será abordado neste trabalho).

Figura 7 – Arestas adjacentes destacadas em vértice de grau três.



Fonte: Elaborada pelo autor (2020).

Um vértice v_n no grafo é **alcançável** a partir de um vértice inicial v_1 se existe uma sequência de vértices partindo de v_1 de maneira a conectar cada qual com seu sucessor (ou vizinho, dependendo do tipo do grafo) até conectar a v_n. Essa sequência de conexões de um vértice a outro é chamada de **passeio** ou **percurso**.

Uma **trilha** ou **cadeia** é um passeio sem repetição de arestas. Um **caminho** é uma trilha sem repetição de vértices. O **comprimento do caminho** pode ser definido como o número de arestas no caminho (ou como a soma dos "pesos das arestas" – conceito não abordado no presente trabalho). A **distância entre dois vértices** é definida então como o tamanho do caminho de menor comprimento capaz de ligar os dois.

Tratando-se de grafo direcionado, pode-se dizer que dois vértices estão **fortemente conectados** se existe caminho de v_1 para v_2 assim como de v_2 para v_1. O conceito se aplica de forma um pouco diferente para grafos não direcionados, mas foge ao escopo do presente trabalho.

Um **ciclo** no grafo é um caminho fechado, ou seja, que inicia e termina no mesmo vértice. Se o grafo é orientado é comum se referir ao ciclo como **circuito**. O **tamanho do ciclo** é o equivalente ao comprimento do caminho, o número de arestas.

Um **subgrafo** G' de G é definido como G' = (V', E') se V' ⊆ V e E' ⊆ E, nesse caso G é um **supergrafo** de G'.

Uma **base de ciclos** em um grafo não direcionado é um conjunto mínimo de caminhos fechados de cujos elementos quaisquer ciclos do grafo podem ser descritos como um somatório. Os vértices e arestas da base de ciclos formam um subgrafo, com menos vértices e possivelmente com menos arestas entre estes vértices que no grafo original.

Por fim, é muito importante destacar que neste trabalho será tratado por **nível** o conjunto dos vizinhos de um vértice. Dessa forma, ao se descrever que determinado vértice foi aberto "em um nível", isso quer dizer que foram extraídos os dados dos seus vizinhos, aberto em "dois níveis" quer dizer que os dados dos vizinhos-dos-vizinhos também foram extraídos e assim sucessivamente.

5.3 Regras de associação

Dado um conjunto de transações, onde em cada transação há um conjunto de elementos, há técnicas de aprendizado de máquina (às vezes referidas como "mineração de dados") específicas para auxiliar a reconhecer padrões de associação dos elementos nas transações. O exemplo mais claro seria o conjunto das compras em um determinado comércio, cada compra contendo vários itens. A essas técnicas de aprendizado, nos referiremos genericamente a partir de agora como "detecção de regras de associação". A seguir, são apresentados alguns conceitos e definições conforme Tan [1] e Zaki [3].

É importante distinguir, de imediato, que a regras de associação não tratam de estabelecer causalidade, somente de medir a ocorrência concomitante dos elementos.

Aos "conjuntos de itens" que compõem os dados chamaremos **transações**. Ao conjunto das transações denominaremos **banco de dados**. **Item** e **elemento** poderão utilizados como sinônimos. Para ilustração dos conceitos, utilizaremos como exemplo o seguinte banco de dados fictício de uma padaria:

Banco de dados das compras

Compra 1: Pão, Leite.

Compra 2: Pão, Queijo, Presunto.

Compra 3: Pão, Leite, Açúcar.

Compra 4: Queijo, Presunto.

Compra 5: Leite, Queijo.

Uma representação equivalente do mesmo banco de dados seria:

Tabela 1: Banco de dados de compras

Item Transação	Pão	Leite	Queijo	Presunto	Açúcar
01	1	1	0	0	0
02	1	0	1	1	0
03	1	1	0	0	1
04	0	0	1	1	0
05	0	1	1	0	0

Fonte: Elaborado pelo autor (2020).

Dessa forma, nosso banco de dados contém cinco transações, cada uma podendo conter um determinado número de elementos até o limite do número total de elementos "existentes", cinco neste exemplo. Para efeito deste exemplo, e deste trabalho, não consideraremos a hipótese de quantificação dos elementos em cada transação nem a existência de transações vazias (sem nenhum elemento).

Com auxílio do exemplo, podemos definir alguns conceitos.

Suporte é a medida da frequência com que o item aparece no conjunto das transações. O suporte do pão seria "três", do açúcar "um". É mais conveniente trabalhar sempre com o valor do **suporte relativo**, então para o pão teríamos três de cinco transações 3/5 = 0,6.

Ao longo deste trabalho sempre que houver referência a suporte, e não houver indicação contrária, se tratará do suporte relativo e será representado como **sup()**.

O suporte também se aplica a um **conjunto de itens**, por exemplo, o suporte do conjunto {pão; leite} seria o número de transações onde esse conjunto aparece, no caso 2/5 = 0,4 (nas compras 01 e 03). Note-se que o suporte de um conjunto de itens será obrigatoriamente menor ou igual ao suporte de cada item em separado.

Uma regra de associação será representada por (X=>Y), onde X e Y são conjuntos de itens. Nessa representação, X será chamado de **antecedente** e Y de **consequente**. Assim são representadas ideias como ({Pão}=>{Leite}) "quem leva pão, leva leite" ou ({Pão, Leite}=>{Açúcar}) "quem leva pão e leite, leva açúcar".

O suporte de uma regra é tão somente o suporte do conjunto dos seus elementos: sup(X=>Y) = sup(X∪Y), ou seja, sup({Pão}=>{Leite}) = sup({Pão, Leite}).

A **confiança** de uma regra é definida de maneira a representar o percentual de vezes em que ela se mostra verdadeira ao se tentar segui-la e é representada por **conf()**. Dessa forma:

$$conf(X => Y) = \frac{\sup(X => Y)}{\sup(X)} \quad (5.1)$$

Essa é uma medida assimétrica, ou seja, conf(X=>Y) ≠ conf(Y=>X). O que no nosso exemplo faz sentido, vejamos: "quem leva presunto, leva queijo" tem confiança de 0,4/0,4 = 1,0, ou seja, todos que levaram presunto também levaram queijo, mas "quem leva queijo, leva presunto" tem confiança 0,40/0,60 = 0,66, pois só dois terços das pessoas que levaram queijo levaram também presunto.

Outra medida muito utilizada é o ***lift*** (do inglês: levantar, erguer ou estimular), cuja tradução não foi encontrada na literatura técnica, mas seria adequadamente entendida como "estímulo". O *lift* é definido como a relação entre o suporte da regra versus o produto do suporte do antecedente e consequente, isto é: quanto o suporte observado varia em relação ao que seria esperado se os conjuntos fossem independentes. Assim:

$$lift(X => Y) = \frac{\sup(X => Y)}{\sup(X).\sup(Y)} \quad (5.2)$$

O *lift* da regra ({Pão}=>{Leite}) seria então 0,40 / (0,60 . 0,60) = 1,11. Um *lift* maior que "1" indica, de certa forma, a utilidade da regra para fazer previsões. Um *lift* menor que "1" é interpretado de forma interessante: um item "substitui o outro", no sentido de que quando o primeiro é levado o outro "deixa de ser levado" com mais frequência do que o esperado. O *lift* é uma medida simétrica e, pela análise das respectivas formulações matemáticas, sempre maior que a confiança.

Como o *lift* é uma medida relativa, nem sempre sua análise é simples. Para auxiliar a interpretação do resultado, há uma medida que exprime a diferença simples entre o suporte da regra e os respectivos suportes do antecedente e consequente. Essa medida dá uma ideia do afastamento da regra do valor que teria se os eventos fossem independentes, em termos absolutos e tem o nome de ***leverage*** (do inglês, impulso, alavancagem, potencialização. Novamente não foi encontrada tradução na literatura técnica), definido como**:**

$$lev(X => Y) = \sup(X => Y) - \sup(X).\sup(Y) \quad (5.3)$$

6 DESENVOLVIMENTO

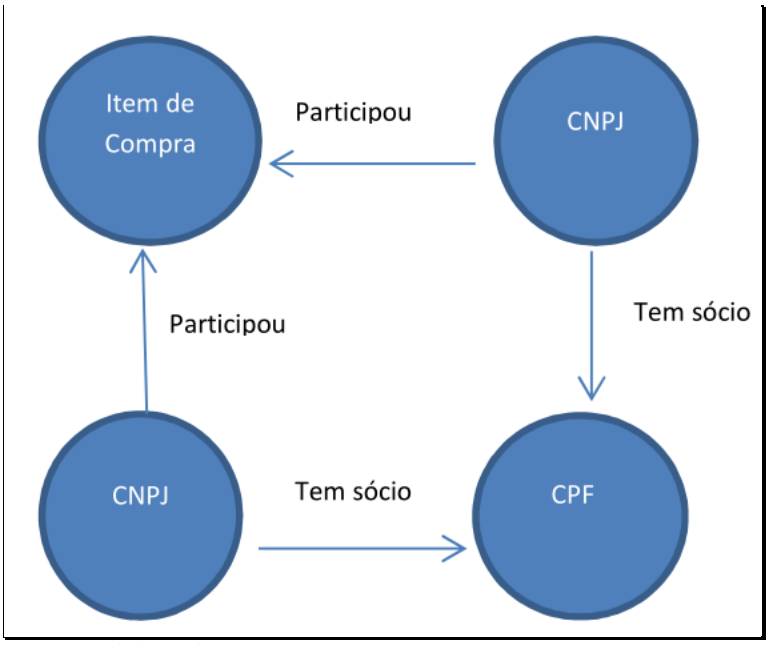
6.1 Preparação

6.1.1 Situação-alvo

A situação-alvo escolhida foi da **existência de vínculos entre empresas que concorrem em um mesmo item de compra em uma licitação**. Tais vínculos poderiam indicar uma "simulação de concorrência" e apareceriam como caminhos entre as empresas no Yggdrasil. Uma vez que foram inseridos os itens de compra e os respectivos vínculos de "participação em licitação" no Yggdrasil, a situação-alvo apareceria como um ciclo no grafo.

A versão mais simples da situação-alvo, escolhida para iniciar o levantamento, foi aquela onde duas empresas que disputam um item de compra teriam um sócio – pessoa física – em comum, conforme ilustrado abaixo.

Figura 8 – Situação-alvo simplificada.



Fonte: Elaborada pelo autor (2020).

6.1.2 Recursos disponíveis

Para o desenvolvimento do trabalho, primordialmente foi utilizado o banco de vínculos Yggdrasil. Foi necessária também a base de dados do sistema DW-SIASG[13] (*Data Warehouse* dos sistemas de compras governamentais) atualizada mensalmente pelo CGU. Completando a lista, temos a base societária das empresas, fornecida pela Secretaria da Receita Federal.

Os bancos de dados da CGU estão disponíveis no CGUDATA, um ambiente interno de acesso restrito disponibilizado por meio da plataforma *SQL Server* que reúne os dados disponíveis institucionalmente.

A análise dos dados foi realizada em *python*, utilizando a aplicação *Jupyter Notebook*[14] – uma aplicação *open source* que roda no navegador de internet e permite a visualização da execução de blocos de código.

Os algoritmos específicos para grafos da biblioteca *networkx*[15] foram utilizados, tendo sido necessária alguma adaptação do código-fonte, detalhada adiante. O ponto de partida da pesquisa no grafo é a função *simple_cycles* dessa biblioteca, uma adaptação do algoritmo de Johnson[16], que tem complexidade de tempo O((v+a)(c+1)) para "v" vértices, "a" arestas e "c" circuitos elementares[17].

6.1.3 Extração dos dados

Por meio de uma consulta direta no banco de dados do DW-SIASG, foram levantadas as ocorrências da versão mais simples da situação-alvo. A esses dados nos referiremos como **levantamento inicial**. O código SQL da consulta encontra-se no Apêndice A.1, para referência.

O levantamento inicial é uma tabela com 2.784.853 linhas, cada qual contendo um vínculo detectado, com as seguintes informações: os dois CNPJs das empresas, o CPF do sócio em comum e o item de compra na qual as duas empresas participaram. Cabe a ressalva que não foi verificada a concomitância entre os vínculos societários ou entre cada vínculo

[13] https://dw.comprasnet.gov.br
[14] https://jupyter.org
[15] https://networkx.github.io/
[16] https://doi.org/10.1137/0204007
[17] https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.cycles.simple_cycles.html#networkx.algorithms.cycles.simple_cycles

com a data da licitação, uma vez que essa informação também não foi considerada ao alimentar o Yggdrasil, devendo, portanto, ser reavaliada ao final do trabalho pelo auditor.

Note-se que no levantamento inicial constam explicitamente as informações dos CPFs e CNPJs encontrados. Por motivos de eventuais questionamentos quanto à privacidade das informações, tais informações não serão reproduzidas neste documento. Durante a execução de todo o trabalho, os identificadores foram substituídos por outros números gerados automaticamente pelo Yggdrasil, não havendo necessidade, doravante, de utilizar outra "máscara" para ocultar as informações.

Em testes preliminares, foi estabelecido que o limite prático do tamanho do grafo possível de analisar com *simple_cycles* não se distancia muito de 100.000 linhas (para o qual a execução pode passar de 24h a depender da configuração de cada grafo), portanto a detecção simultânea dos 2,7 milhões de casos detectados no levantamento inicial, de uma só vez, já estaria fora de consideração por exigir a análise de um grafo muito maior (somente o subgrafo em dois níveis com os dados das empresas já detectadas no levantamento inicial tem mais de dez milhões de linhas).

Um grande desafio deste trabalho, portanto, consiste em viabilizar a pesquisa dos vínculos de maneira a permitir a execução em tempo compatível com os recursos disponíveis. A estratégia vislumbrada, de saída, foi o recorte temporal e espacial da base de dados.

Inicialmente, um recorte dos dados foi extraído do Yggdrasil para testes, a esse conjunto nos referiremos doravante como **dados iniciais**: Subgrafo não direcionado dos itens de compra cujas compras têm como referência o ano de 2018, aberto em dois níveis, das unidades 194018, 200141, 160204, (cód. UASG - numeração do SIASG).

A limitação temporal, 2018, escolhida nesta etapa seguiu até o final do trabalho.

As unidades foram incialmente escolhidas por terem aparecido no levantamento inicial assim como pela contagem das linhas, representando o que foi considerado uma unidade pequena (3.422 linhas), uma média (47.142 linhas) e uma grande[18] (109.864 linhas) em termos de linhas retornadas. O código SQL para extração dos vínculos do Yggdrasil encontra-se no Apêndice, item A.2, para referência.

Dos dados iniciais, cada linha representa uma aresta e há 13 tipos de vínculos representados nos dois níveis. Temos as seguintes informações e características:

[18] "Grande" em relação ao limite de cem mil linhas citado anteriormente.

Tabela 2: Exemplo dos dados iniciais

Vértice 1	Tipo Vértice 1	Tipo Vínculo	Vértice 2	Tipo Vértice 2
2083818	ITEMCOMPRA	item de compra teve participante	332132	CNPJ
32530845	CPF	sócio(a) em	336387	CNPJ
2030207	CNPJ	tem sócio(a)	310618	CPF

Fonte: Elaborado pelo autor (2020).

Para formar a ideia do espaço representado, as quantidades de cada tipo de vínculo são:

Tabela 3: Vínculos que aparecem em dois níveis

Tipo Vínculo	Qtd
tem sócio(a)	78.540
item de compra teve participante	25.625
sócio(a) em	24.566
tem responsável como - sócio-administrador	11.680
tem responsável como - não informado	6.789
tem responsável como - titular p.fisica resid	5.787
item de licitação vencido por	4.220
tem representante como - administrador	1.539
tem responsável como - administrador	792
tem responsável como - presidente	386
tem representante como - não informado	333
tem responsável como - diretor	170
tem responsável como - administrador judicial	1

Fonte: Elaborado pelo autor (2020).

6.2 Programação

6.2.1 Itens em dois níveis e *simple_cycles*

A unidade 194018, com 3.422 relacionamentos, foi escolhida para o início dos testes, pelo seu menor tamanho. O passo a passo da execução é o seguinte:

a) Recorte do subgrafo ("*select*" no banco) relativo à unidade do conjunto de dados iniciais, utilizando dois níveis de relacionamentos;
b) Transformação do subgrafo em um grafo direcionado (necessário para o uso da função *simple_cycles*);
c) Pesquisa e listagem dos ciclos com *simple_cycles*;
d) Agrupamento e contagem, a partir dos ciclos listados, por tamanho do ciclo;
e) Impressão na tela dos ciclos de tamanho quatro (por tipo de vértice, sem repetição).

O roteiro rodou em **dois segundos** para a unidade, tendo detectado 1.239 ciclos de "tamanho dois" e 530 de "tamanho quatro", representados como:

{2: 1.239, 4: 530}.

Os ciclos impressos foram os seguintes (o segundo elemento da linha é o primeiro da seguinte):

```
(ITEMCOMPRA) -> item de compra teve participante -> (CNPJ)
(CNPJ) -> item de compra teve participante -> (ITEMCOMPRA)
(ITEMCOMPRA) -> item de compra teve participante -> (CNPJ)
(CNPJ) -> item de compra teve participante -> (ITEMCOMPRA)

(CNPJ) -> sócio(a) em -> (CNPJ)
(CNPJ) -> sócio(a) em -> (CNPJ)
(CNPJ) -> sócio(a) em -> (CNPJ)
(CNPJ) -> sócio(a) em -> (CNPJ)
```

Embora tenha sido executado rápido e a unidade não seja muito representativa em vista de tão diminuta, dois problemas de imediato chamam atenção: a detecção de ciclos sem itens de compra e, mais grave, a detecção de ciclos com mais de um item de compra, vínculo sem nenhuma relevância para estabelecer a "proximidade" entre as empresas.

A unidade 200141 foi a próxima a ser testada. O roteiro se completou em **cinco minutos**, um tempo muito bom em vista do tamanho da unidade, com 47.142 relacionamentos. A relação entre o tamanho relativo das unidades e a diferença de tempo, entretanto, não é de

bom auspício: a unidade é 13,77 vezes maior que a anterior, mas levou 150 vezes mais tempo para processar.

A diferença de número de ciclos encontrados é significativa:

{2: 2.208, 4: 79.796, 6: 8.177.332, 8: 71.172, 10: 385.400,
12: 1.063.928, 14: 462.308}.

Números astronômicos como os oito milhões de ciclos de "tamanho seis" encontrados, bem como o tamanho dos maiores ciclos (até 14) indicam o enorme desperdício computacional despendido.

Entre os mais de 30 ciclos impressos ao final, podem ser encontrados:

```
(ITEMCOMPRA) -> item de compra teve participante -> (CNPJ)
(CNPJ) -> tem sócio(a) -> (CPF)
(CPF) -> sócio(a) em -> (CNPJ)
(CNPJ)-> item de compra teve participante -> (ITEMCOMPRA)

(ITEMCOMPRA) -> item de compra teve participante -> (CNPJ)
(CNPJ) -> sócio(a) em -> (CNPJ)
(CNPJ) -> sócio(a) em -> (CNPJ)
(CNPJ) -> item de compra teve participante -> (ITEMCOMPRA)

(ITEMCOMPRA) -> item de compra teve participante -> (CNPJ)
(CNPJ) -> tem representante como - administrador -> (CPF)
(CPF) -> sócio(a) em -> (CNPJ)
(CNPJ) -> item de compra teve participante -> (ITEMCOMPRA)
```

O primeiro ciclo impresso é exatamente a situação-alvo simplificada que se esperava encontrar. A aparição do segundo e terceiro já demonstram o potencial do método em recuperar novas situações sem a necessidade de especificá-las previamente.

Por fim, o roteiro foi aplicado à unidade 160204, com mais de cem mil linhas. A execução foi interrompida após 50h, pois apenas um dos itens de compra levou mais de 24h, indicando que o processamento da unidade poderia não ser concluído em menos de 500h[19].

Sem dúvidas o recorte por unidade com periodicidade anual não seria uma estratégia viável. Além disso, a saída dos resultados nesse formato é difícil de interpretar ou armazenar e ainda há um enorme desperdício computacional pelo número desproporcional de ciclos detectados: seja pela detecção de ciclos sem itens de compra ou com mais de um item de compra,

[19] Talvez nem pudesse ser processado. O algoritmo chegou a ocupar toda a memória disponível – 1 TB de RAM.

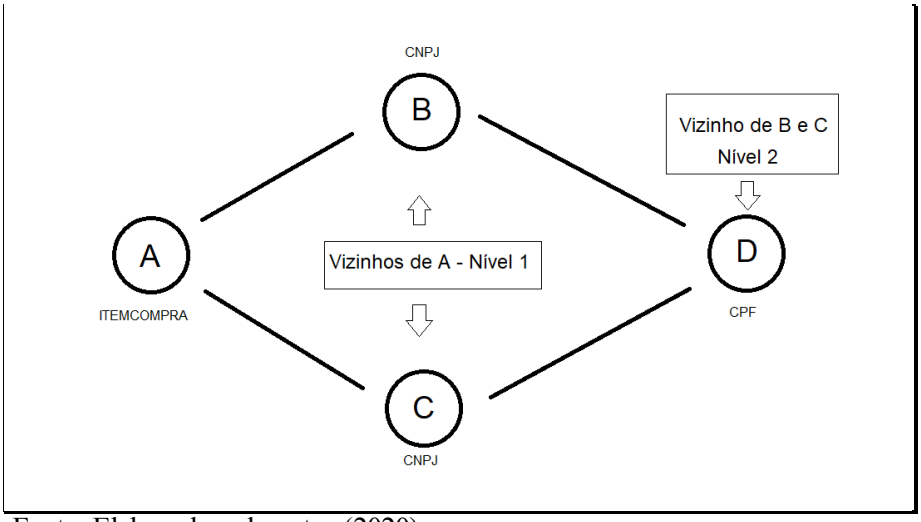
pela detecção de ciclos de tamanho muito maior que o esperado ou pelas astronômicas quantidades de ciclos. Também foi observado que a rotina detecta ciclos "inversos", que são meras repetições, retornando tanto "ABCD" quanto "DCBA".

6.2.2 Iteração item a item

O menor subgrafo que poderia ser extraído utilizando as rotinas disponíveis no Yggdrasil que contivesse os dados suficientes para encontrar os ciclos ligados a determinado item de compra seria aquele obtido por meio da pesquisa apenas do item de compra avaliado, aberto[20] em níveis suficientes para conter os demais vínculos procurados.

Para o ciclo alvo procurado, por exemplo, o número de níveis a abrir precisaria ser dois, como pode ser entendido pela ilustração:

Figura 9 – Níveis abertos a partir do item de compra.



Fonte: Elaborada pelo autor (2020).

A partir do momento que a pesquisa para um item estivesse funcionando, haveria necessidade de automatizar a iteração item a item e a estratégia natural para isso foi iterar por unidade.

Houve necessidade de rotinas específicas para "organização" dos ciclos encontrados anterior à impressão em tela: ordenar os ciclos para que começassem pelo item de compra, remover os ciclos que não continham item de compra e eliminar ciclos inversos e/ou repetidos.

[20] Conforme definido em 5.2

O *feedback* foi melhorado e feito em tela, enquanto em desenvolvimento no *Jupyter*, com a ideia de direcionar para um arquivo de "*log*" durante a execução em produção. As informações retornadas passaram a ser:

- Número do item;
- Número de relacionamentos retornado pelo banco de dados;
- Conjunto dos ciclos encontrados;
- Conjunto dos ciclos após a eliminação daqueles que não continham o item de compra;
- Conjunto dos ciclos após a eliminação dos ciclos repetidos ou de tamanho "2".

Desse último conjunto, são listados em sequência os vértices de cada ciclo e uma representação dos relacionamentos (arestas), conforme demonstrado a seguir com uma cópia da saída em tela:

```
Item 2001410500001201800015
279
{2: 266, 4: 6}
{2: 3, 4: 4}
{4: 2}
4 níveis de relacionamentos, 2 caso(s):

[401868983, 6801568, 17757327, 6823922]

 [ ITEMCOMPRA ] <-- item de compra teve participante --> [ CNPJ ]
 [ CNPJ ] <-- sócio(a) em --> [ CNPJ ]
 [ CNPJ ] <-- sócio(a) em --> [ CNPJ ]
 [ CNPJ ] <-- item de compra teve participante --> [ ITEMCOMPRA ]

[401868983, 6801568, 6514562, 6823922]

 [ ITEMCOMPRA ] <-- item de compra teve participante --> [ CNPJ ]
 [ CNPJ ] <-- tem representante como - administrador --> [ CPF ]
 [ CPF ] <-- tem sócio(a) --> [ CNPJ ]
 [ CNPJ ] <-- item de compra teve participante --> [ ITEMCOMPRA ]
```

A unidade escolhida para testes nesta etapa foi a 200141, cujo processamento simultâneo do grafo extraído a partir de todos os itens de compra levava cinco minutos. Com estratégia de iterar item a item, o tempo de execução obtido para processar todos os itens da unidade, em dois níveis, foi de quatro minutos, indicando que havia possibilidade real de ganho.

6.2.3 Aumento da profundidade da pesquisa

Uma vez que o número de arestas e vértice fica bastante reduzido ao se optar pela extração do menor grafo possível, é possível utilizar três níveis na pesquisa do Yggdrasil e ainda manter o número de relacionamentos abaixo do limite de 100 mil arestas experimentalmente estimado como crítico para a execução da função *simple_cycles*.

Com três níveis, a ocorrência de alguns problemas piora consideravelmente: muitos ciclos detectados não continham o item de compra e ciclos muito grandes continuavam a aparecer, todos descartados a posteriori. Persistia também o aparecimento de outros itens de compra conectados ao ciclo, conforme mostrado anteriormente.

Não obstante, o uso de três níveis enriquece muito a possibilidade de extração de informações a partir da topologia. Os tipos de vínculos representados ao se abrir os itens em três níveis são 50, sendo os que aparecem em maior e menor quantidade apresentados nas Tabelas 4 e 5, a seguir.

Tabela 4: Vínculos mais representados

Tipo Vínculo	Qtd
responsável por empresa - presidente	600.808
sócio(a) em	287.204
tem sócio(a)	189.857
responsável por empresa - diretor	167.486
responsável por empresa - administrador	156.696
possível filho(a)	62.311
responsável por empresa - sócio-admin	60.607
representa sócio como - procurador	56.223
possível pai/mãe	47.717

Fonte: Elaborada pelo autor (2020).

Tabela 5: Vínculos menos representados

Tipo Vínculo	Qtd
pensionista	116
instituidor de pensão	109
representa sócio como - curador	10
tem responsável como - procurador	6
responsável por empresa - sócio ostensivo	4
padrasto/madrasta	3
tem responsável como - administrador judicial	1
provedor - Siape	1
tem responsável como - sócio ostensivo	1

Fonte: Elaborada pelo autor (2020).

6.2.4 Modificação do Yggdrasil

Uma das configurações à disposição do usuário no Yggdrasil é a seleção de um subconjunto de tipo de arestas a considerar, ou não, na pesquisa dos vínculos. Isso é feito por meio de um "grupo" de pesquisa. Pode-se escolher, por exemplo, um grupo que contenha apenas vínculos familiares (parentesco), ou um grupo que não contenha vínculos empregatícios e assim por diante.

Com o objetivo de aprimorar o uso do Yggdrasil, foi proposta alteração em um dos grupos de pesquisa disponibilizados pela ferramenta, descrita sucintamente a seguir.

Uma vez realizada a pesquisa pelo item de compra, no primeiro nível vinham todas as empresas que participaram daquele item, naturalmente. Ocorria que no segundo nível, além dos demais vínculos daquelas empresas (os sócios, em geral), retornavam as informações de todos os outros itens de compra dos quais aquelas empresas haviam participado. Ao se abrir em três níveis, todos esses outros itens de compras eram por sua vez abertos.

Como há várias empresas que participam de muitas licitações, muitos itens em consequência, a diferença de magnitude do grafo aberto apenas em dois níveis ou três era imensa.

Para resolver esse problema, foi realizada uma alteração na definição do grupo "PADRAO" do Yggdrasil, de forma que esse grupo passou a abrir as arestas dos itens de compra

de forma direcionada, ou seja, a aresta é aberta se a direção do vínculo for do item de compra para outra entidade, mas nunca se for originada de outra entidade para o item de compra.

6.2.5 Alteração na função de ciclos

Alguns itens de compra em particular apresentaram um enorme problema em três níveis, a exemplo do item "200130050000920180018". Com 3276 relacionamentos apenas, a execução do *simple_cycles* apenas para esse item não retornava resultados em tempo viável, mesmo com todas as melhorias apresentadas até este ponto, tendo sido interrompida após 24h.

A partir do código fonte da biblioteca *networkx*, de extensa pesquisa online e de alguns testes de tentativa e erro, foi elaborada uma versão da função de pesquisa de ciclos com algumas simplificações e com limitação do tamanho máximo do ciclo a pesquisar.

Utilizando um tamanho máximo de ciclo "8 arestas", foi possível resolver o item "200130050000920180018" em pouco menos de 25 minutos, com os seguintes resultados:

```
Item 200130050000920180018
3276
{3: 570, 4: 6274, 5: 5690, 6: 35444, 7: 55950, 8: 253566}
{5: 2, 6: 82, 7: 76, 8: 834}
{5: 1, 6: 41, 7: 38, 8: 417}
```

Alguns exemplos dos vínculos encontrados:

```
[401917229, 12666399, 43216197, 42508042, 10498966, 96275985, 96779125]

 [ ITEMCOMPRA ] <-- item de compra teve participante --> [ CNPJ ]
 [ CNPJ ] <-- tem sócio(a) --> [ CPF ]
 [ CPF ] <-- cônjuge --> [ CPF ]
 [ CPF ] <-- sócio(a) em --> [ CNPJ ]
 [ CNPJ ] <-- tem sócio(a) --> [ CNPJ ]
 [ CNPJ ] <-- sócio(a) em --> [ CNPJ ]
 [ CNPJ ] <-- item de compra teve participante --> [ ITEMCOMPRA ]

[401917229, 97122368, 20684880, 54132140, 96227924, 6894303, 96090870,
6770471]

 [ ITEMCOMPRA ] <-- item de compra teve participante --> [ CNPJ ]
 [ CNPJ ] <-- tem sócio(a) --> [ CPF ]
 [ CPF ] <-- possível pai/mãe --> [ CPF ]
 [ CPF ] <-- tem responsável como - administrador --> [ CNPJ ]
 [ CNPJ ] <-- tem sócio(a) --> [ CNPJ ]
 [ CNPJ ] <-- tem sócio(a) --> [ CNPJ ]
 [ CNPJ ] <-- sócio(a) em --> [ CNPJ ]
 [ CNPJ ] <-- item de compra teve participante --> [ ITEMCOMPRA ]
```

Como se pode observar nesse último caso, ciclos muito grandes não têm muito interesse para auditoria. Embora na área investigativa haja valor em estabelecer possíveis conexões distantes, do ponto de vista da auditoria não é viável fazer uma análise aprofundada em todos os casos de sócios-de-sócios-de-sócios que forem detectados. Especialistas consultados por Erven [8] também se manifestaram pela necessidade de restringir o tamanho dos ciclos, adotando como referência o tamanho "6 arestas". Por esse motivo, o tamanho máximo de ciclo foi reavaliado de "8" para "6" e mantido até o final do estudo.

Com um limite máximo, **finalmente havia sido viabilizada a consulta de unidades inteiras**, varrendo-se item a item, de forma automatizada, com o bônus da **utilização de três níveis**, ou seja, detectando vínculos muito mais ricos que inicialmente era esperado.

O problema de mais de um item de compra aparecendo no ciclo encontrado havia sido sanado pela modificação no Yggdrasil. O desperdício computacional com ciclos muito grandes também havia sido sanado pela limitação do tamanho do ciclo, retornando resultados em tempo pequeno o suficiente para que o trabalho fosse considerado possível. A questão de muitos ciclos encontrados que não possuíam itens de compra, necessitando descarte a posteriori, persistia, com menor impacto (pois agora somente nos ciclos menores).

Não obstante, algumas unidades ainda demoravam alguns dias para finalizar a pesquisa e por isso várias outras melhorias foram tentadas, descritas a seguir.

6.2.6 Limitação de tempo por item

Uma vez viabilizada a pesquisa por unidade, o próximo passo foi automatizar a pesquisa de várias unidades em sequência.

A menor das Unidades da Federação, com 30 unidades (UASG) é o Acre, o que motivou sua escolha como UF de teste para agregação (optou-se pela agregação por critério geográfico ao invés de Órgão Superior).

Foi observado que em algumas unidades de grande tempo de execução o gargalo estava em um número muito reduzido de itens, muitos dos quais sequer retornavam vínculos interessantes ao final da pesquisa.

De forma a viabilizar a execução de uma UF inteira, foi necessário limitar de alguma forma o recurso computacional empregado em cada item de compra.

Foi estipulado um limite de dez minutos, ao fim dos quais uma nova pesquisa daquele item seria realizada no Yggdrasil, agora com apenas dois níveis de profundidade, seguido

de um novo processamento dos ciclos com o grupo reduzido de vínculos. A justificativa para tal redução encontra suporte no fato de que os ciclos mais curtos é que acabam sendo os mais interessantes, e a pesquisa em dois níveis é suficiente para detecção de ciclos de tamanho "4" (vide Figura 9).

Ao longo dos testes, com a entrada de outras melhorias, o tempo limite foi revisado para 5 minutos, valor que se manteve ao final do trabalho.

6.2.7 Processamento das similaridades

Outra causa comum de demora no processamento das unidades era o processamento sequencial de diversos itens da mesma licitação, todos com os mesmos concorrentes, ou seja, gerando todos o mesmo subgrafo.

A solução encontrada veio ao encontro de outra necessidade do projeto: o armazenamento dos resultados. Com grandes volumes de informação se acumulando, foi necessário um planejamento do que seria armazenado, em que formato, e de como isso seria programado sem adicionar impacto ao tempo de execução, até porque havia agora um limite de tempo para o processamento de cada item.

Para armazenar os resultados, além do log que passaria da tela para um arquivo de texto durante a execução fora do Jupyter, o melhor formato possível seria o próprio grafo. Com isso não haveria necessidade de processar transformações ao salvar ou abrir os arquivos, tudo já estaria no formato nativo usado pela biblioteca *networkx*. A biblioteca de serialização[21] *pickle* foi escolhida para a tarefa.

Para verificar a similaridade do grafo com os já processados, seria inviável abrir todos os demais arquivos com grafos a cada nova iteração. Por isso uma estrutura adicional de dados foi mantida na memória durante a execução, com o resumo dos resultados obtidos de cada item – um "histórico".

O histórico inicialmente foi programado por licitação, mas algumas unidades tinham "licitações repetidas" ao invés de "itens repetidos", motivo pelo qual o histórico foi reprogramado para verificar todas as licitações dentro da mesma unidade em busca de similaridade.

[21] Processo de traduzir estruturas de dados em um formato que pode ser armazenado.

O salvamento do histórico em disco ao final de cada iteração, combinado com a verificação e carga das informações a cada início de iteração, deixou operacional um esquema de recuperação em caso de falha (o item salvo não precisaria ser reprocessado), representando mais um bônus na melhoria do tempo de execução.

É interessante notar que várias restrições estavam sendo simultaneamente impostas durante a execução das iterações, tais como: limite de tempo, limite máximo do tamanho do ciclo, quantidade de níveis pesquisados no Yggdrasil. Essas restrições foram salvas no histórico em conjunto com os resultados, permitindo uma avaliação de compatibilidade durante o início da iteração.

6.2.8 Utilização da "base de ciclos" para poda

Em vista do tempo de execução depender fortemente do número de vértices e arestas do grafo[22], a estratégia de poda (remoção) dos vértices e arestas que pudessem ser descartados de antemão poderia melhorar o desempenho.

Uma alternativa simples seria eliminar os vértices com apenas um vizinho, pois todos os vértices em um ciclo precisam estar conectados ao menos a outros dois. Essa eliminação pode ser feita repetidamente, pois ao se eliminar os primeiros candidatos alguns dos demais podem ter passado a ter apenas outro vizinho. Essa estratégia foi tentada com pouco sucesso, a diferença de tempo não chegou a ser relevante.

Outra possibilidade encontrada para a poda durante a revisão bibliográfica foi da detecção da base de ciclos. Embora nenhum caso na literatura tenha especificamente citado o uso da base de ciclos com esse propósito, seu uso pareceu natural.

A base de ciclos é um conjunto de caminhos fechados a partir dos quais todos os demais ciclos poderiam ser descritos, com operações como soma ou subtração dos conjuntos. Pode-se construir o menor grafo que contém todos os caminhos ao se retirar todos os vértices do grafo original que não estejam na base de ciclos.

A extração da base de ciclos foi incorporada ao início do roteiro, obtendo redução percentual significativa do tempo de execução além do já observado com as demais melhorias. Para efeito de ilustração, o item de compra "20013005000092018000 18", já citado, cujo tempo de processamento havia sido reduzido de mais de 24h para 25 minutos, passou a retor-

[22] Complexidade de tempo O((v+a)(c+1)), já apresentado.

nar resultados idênticos em apenas 8 minutos (deixando de falhar no critério de limitação temporal).

6.2.9 Pesquisa pela aresta inicial

Em meio à avaliação dos resultados obtidos, os dados de três Unidades da Federação foram repetidamente analisados: Acre, Maranhão e Pará.

O Acre havia sido escolhido porque era a menor das UF, com 30 unidades, e seu tempo de execução era relativamente acessível para repetição de testes (9h após limitação de 10 minutos por item, 3h após uso também de base de ciclos e histórico). O Pará, com 118 unidades, estava no ponto mediano da listagem das UF. O Maranhão, por fim, foi escolhido como um intermediário entre as duas, com 64 unidades.

Em todos os casos, persistiam alguns problemas até então sem proposta de solução: a detecção de ciclos inversos, a detecção de ciclos que não continham o item de compra e diversos itens com a avaliação limitada pelo tempo máximo estipulado.

Devido ao grande número de itens de compra com ciclos detectados, a necessidade de uma nova melhoria apareceu durante a etapa de avaliação dos resultados: o descarte dos ciclos nos quais não se encontrava o vencedor da licitação. Esse descarte a posteriori acabou se mostrando um grande desperdício, pois implicava em ignorar um percentual razoável dos grafos encontrados após horas e horas de processamento.

Desse desperdício, surgiu a proposta de tentar evitar a detecção destes ciclos em primeiro lugar. Para isso, seria necessário retomar a rotina de pesquisa de ciclos, elaborada na primeira rodada de melhorias, e fazer adaptações mais direcionadas.

Como o grafo no qual a rotina que busca ciclos roda é uma versão direcionada, os ciclos desejados seriam sempre iniciados com o arco (item de compra, vencedor da proposta), retornando sempre pelos arcos adjacentes no vértice do item de compra.

Após algumas tentativas diferentes de solução, a ideia de restringir o primeiro arco da pesquisa se mostrou extremamente vantajosa: junto com o problema da detecção indevida de ciclos que não contivessem o vencedor, estariam resolvidos os problemas da detecção dos ciclos inversos e do desperdício computacional com ciclos que não continham o item de compra!

6.2.10 Encerrando o desenvolvimento do *software*

Assim se chegou à versão final do código, que se encontra reproduzida no Apêndice B.1. Ao longo do processo de construção dessa solução, sempre se manteve em mente a possibilidade da aplicação futura em outros casos de uso, de modo que o código foi construído de maneira a facilitar a adaptação.

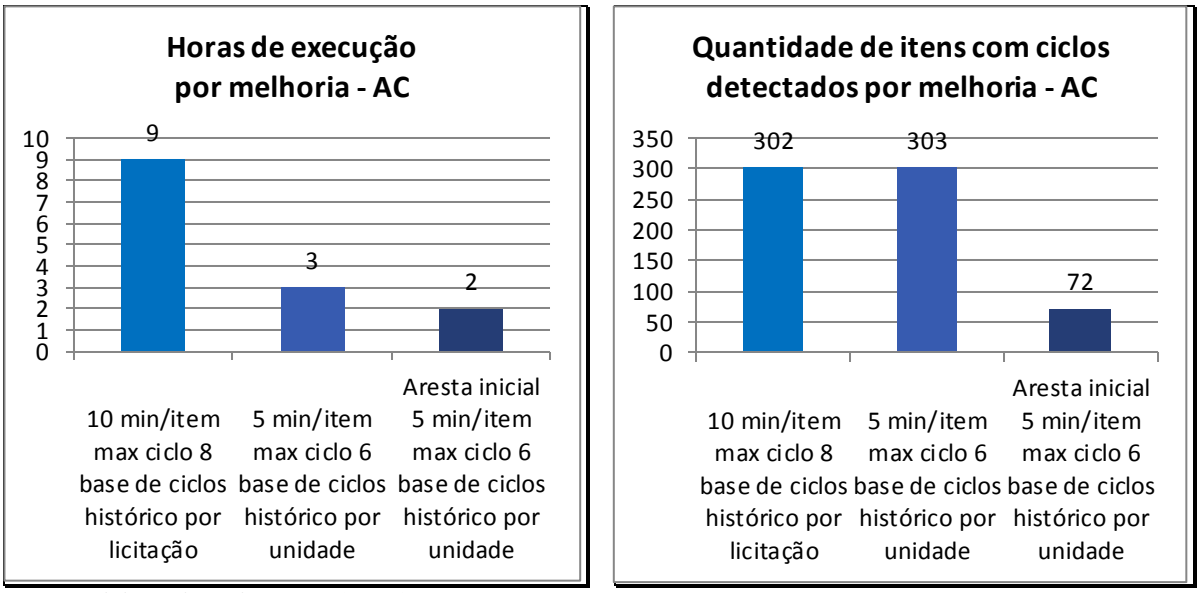
Durante o desenvolvimento, diversos ajustes foram experimentados, como a variação no tamanho máximo dos ciclos (de 6 a 8), a variação no tempo limite para o processamento dos itens de compra (5 ou 10 minutos), a variação na profundidade da pesquisa no Yggdrasil, a utilização ou não da base de ciclos, a realização de ajustes finos nas informações armazenadas no histórico, com fins tanto de retomada de execução em caso de parada como de evitar o reprocessamento de itens similares, além dos diversos testes de salvamento e recuperação das informações (com o respectivo desenho dos grafos encontrados, mostrado adiante na avaliação dos resultados).

Muitos testes tiveram que ser repetidos, a cada alteração de código, para garantir que a contagem de ciclos não se alterava em nenhum dos casos (sempre houve necessidade de correções).

Os resultados obtidos, antes mesmo da aplicação ao caso concreto, são bastante satisfatórios e representarão novas possibilidades de exploração do Yggdrasil.

A situação ao início do trabalho era que a avaliação dos vínculos de apenas um item de compra podia passar de 24h. A situação final demonstra que não só a avaliação de uma UF inteira foi viabilizada em um tempo realista, como também que as melhorias desenvolvidas resolveram os problemas inicialmente elencados, em especial da detecção de ciclos que seriam posteriormente descartados. Para comparação, a Figura 10 ilustra o progresso obtido na execução da rotina para o estado do Acre, que em 2018 tem 16.864 itens de compra distribuídos entre 30 unidades do poder executivo federal – executada uma vez para cada configuração:

Figura 10 – Resultados de acordo com as melhorias testadas – uma execução cada.



Fonte: Elaborada pelo autor (2020).

Cabe destacar que após a inclusão da aresta inicial na pesquisa, o limite de tempo foi ajustado em 300 segundos e não foi atingido por nenhum dos itens de compra, em nenhuma das Unidades da Federação, ou seja, nenhum item deixou de ser avaliado em três níveis. A limitação temporal foi mantida no código para evitar a possibilidade de um travamento por motivos imprevistos.

6.3 Experimento

6.3.1 Aplicação do modelo

Com a "melhor solução" encontrada, duas Unidades da Federação foram escolhidas para a pesquisa da situação-alvo, DF e MG. Em termos de tamanho (quantidade de unidades e itens), são a segunda e a terceira maiores do Brasil, ficando atrás apenas do RJ, que é praticamente do tamanho das duas somadas.

Escolhendo duas UF de grande porte, espera-se encontrar uma quantidade grande de casos relevantes, que precise do auxílio de técnicas suplementares para priorização dos casos a avaliar. Também se espera poder fazer alguma análise comparativa entra as duas, já que são de porte semelhante, mas com características bastante distintas.

O DF conta com 287 unidades do poder executivo federal (com licitações registradas - SIASG/2018), MG conta com 213 unidades com essas características. Os tempos necessários para a pesquisa foram 23h para o DF, que retornou 613 itens de compra com ciclos detectados, e 27h para MG, que retornou 882.

O numero de ciclos retornados, como esperado, é muito grande, havendo necessidade do desenvolvimento de uma estratégia de avaliação e priorização. O ponto de partida seria a visualização dos ciclos encontrados, para orientar a direção do trabalho, conforme tratado a seguir.

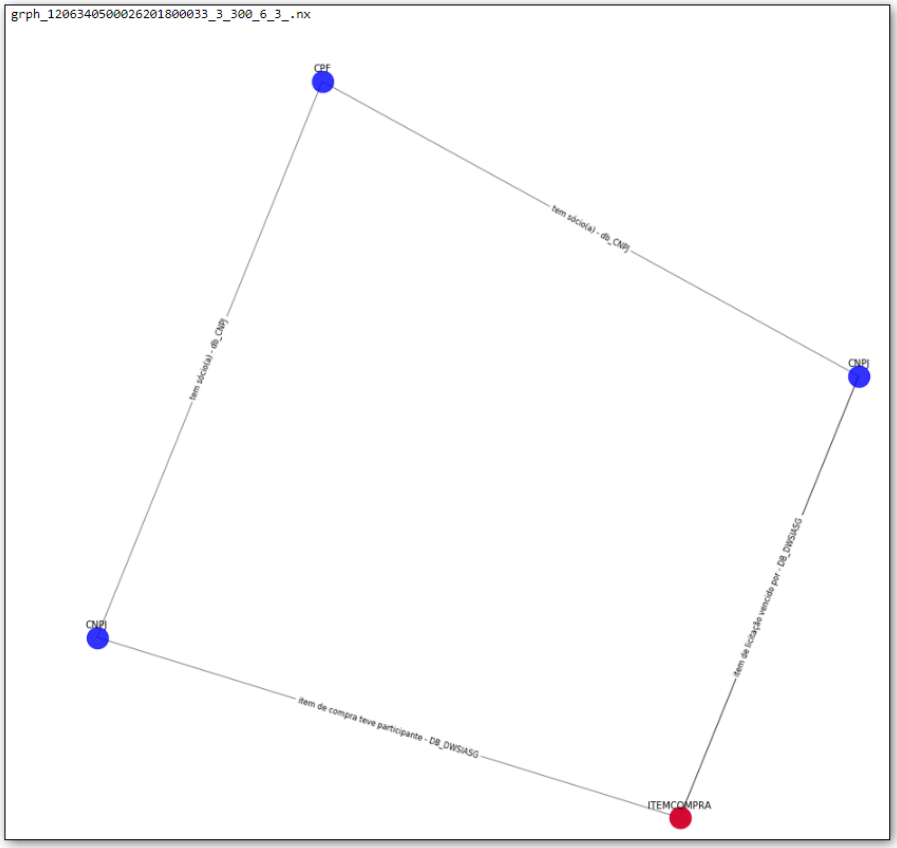
6.4 Avaliação

6.4.1 Visualização dos ciclos

Havendo 1495 itens de compra com ciclos detectados, a maneira preliminar de acompanhar os resultados foi pela inspeção visual. Uma rotina simples foi elaborada para imprimir em tela todos os grafos gerados de cada UF, um por item de compra, sequencialmente, de forma a permitir a inspeção com uma rolagem de tela.

Não é possível trazer toda a informação gerada. Entretanto, alguns casos de interesse particular serão reproduzidos abaixo.

Figura 11 – Situação-alvo simplificada - MG.

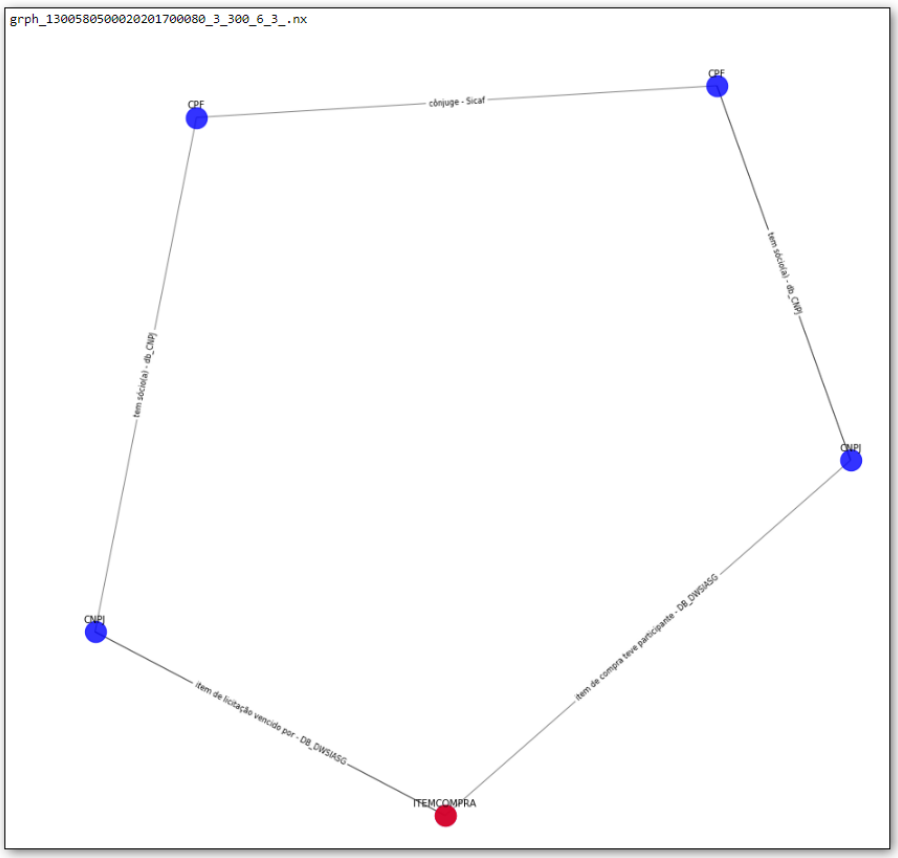


Fonte: Elaborada pelo autor (2020).

O primeiro caso de interesse, naturalmente, é a situação-alvo simplificada, mostrada na Figura 11. Todo o trabalho partiu da premissa de sua existência e encontrá-la seria o mínimo de validação necessária aos resultados. Foi encontrada em diversos casos em ambas a Unidades da Federação, mas poucas vezes de forma "isolada", tal como no caso ilustrado.

No caso das UF analisadas, está entre os menores ciclos detectados, com quatro vértices. Em uma das outras UF de testes apareceu um único caso de vínculo societário "direto" entre as participantes da licitação, formando um grafo de apenas três vértices.

Figura 12 – Concorrência entre empresas de cônjuges - MG.

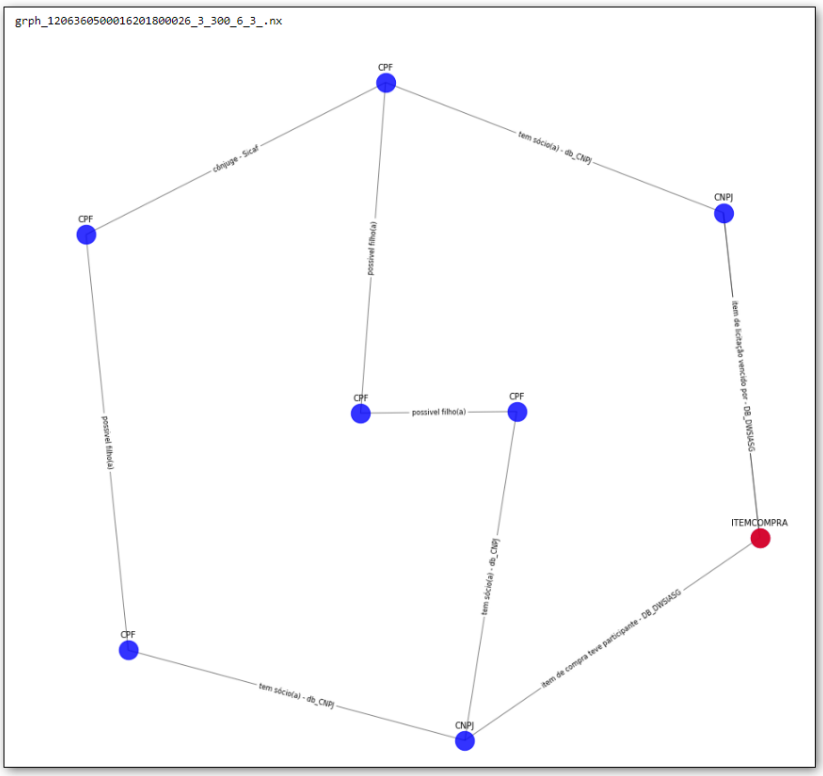


Fonte: Elaborada pelo autor (2020).

Diversos ciclos com cinco vértices foram encontrados, talvez o mais emblemático deles seja o da concorrência entre duas empresas pertencentes a cônjuges, tal como ilustrado na Figura 12.

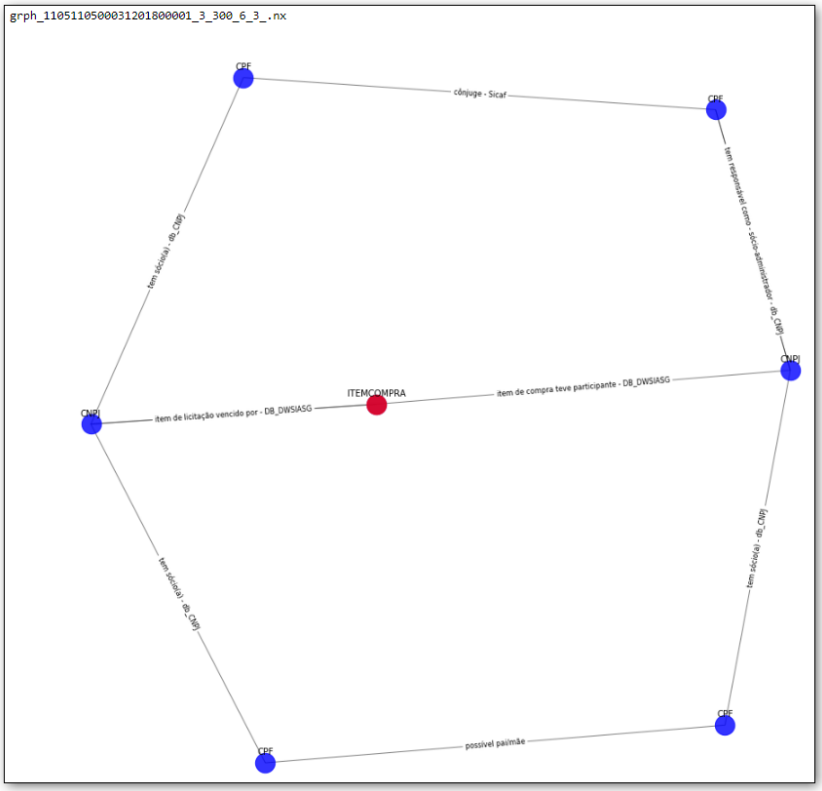
Vínculos familiares que conectam as empresas concorrentes são bastante comuns, e sua complexidade crescente com o número de vértices no ciclo torna a análise um tanto difícil, necessitando que cada caso seja aberto e cada vínculo validado (vínculos de parentesco são construídos com base nos nomes e, embora haja o descarte de homônimos, não são 100% confiáveis).

Figura 13 – Concorrência entre empresas da mesma família - MG.



Fonte: Elaborada pelo autor (2020).

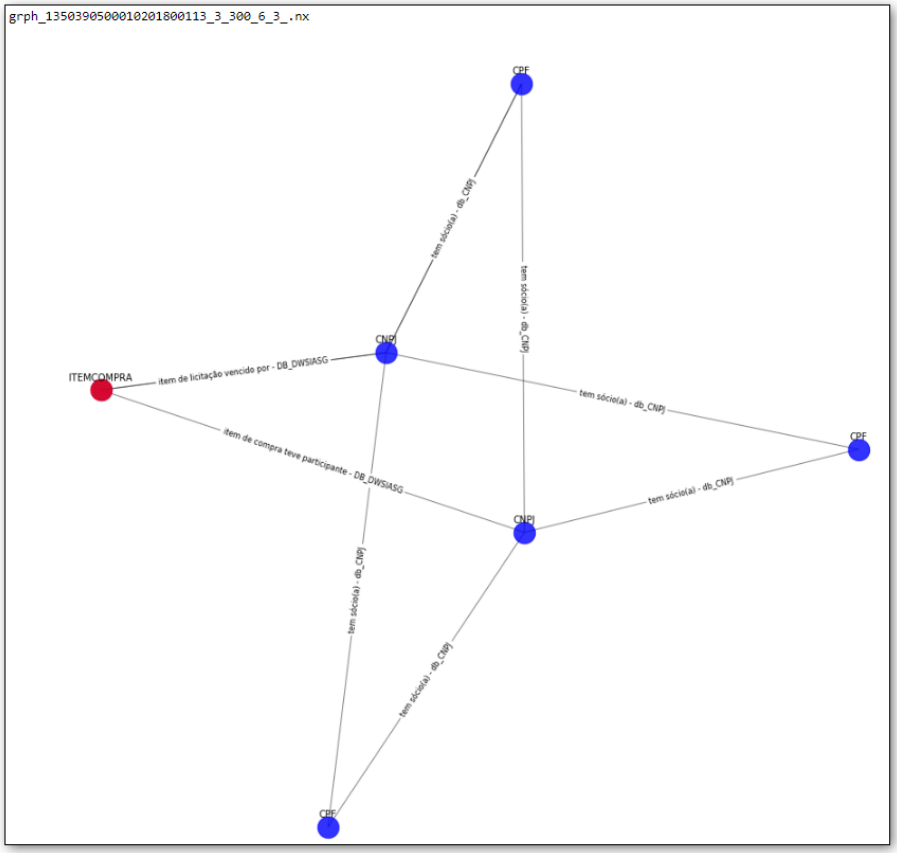
Figura 14 – Conexões familiares - DF.



Fonte: Elaborada pelo autor (2020).

A quantidade de vínculos também funciona como uma "validação" da conexão, de certa forma. Se for improvável haver uma conexão familiar "por engano" entre duas empresas, que dirá uma sequência delas, tal como ilustrado nas Figuras 13 e 14. Também em outros casos, como da Figura 15, que mostra duas empresas com três sócios em comum, a quantidade de vínculos reforça a proximidade da conexão.

Figura 15 – Três sócios em comum - DF.

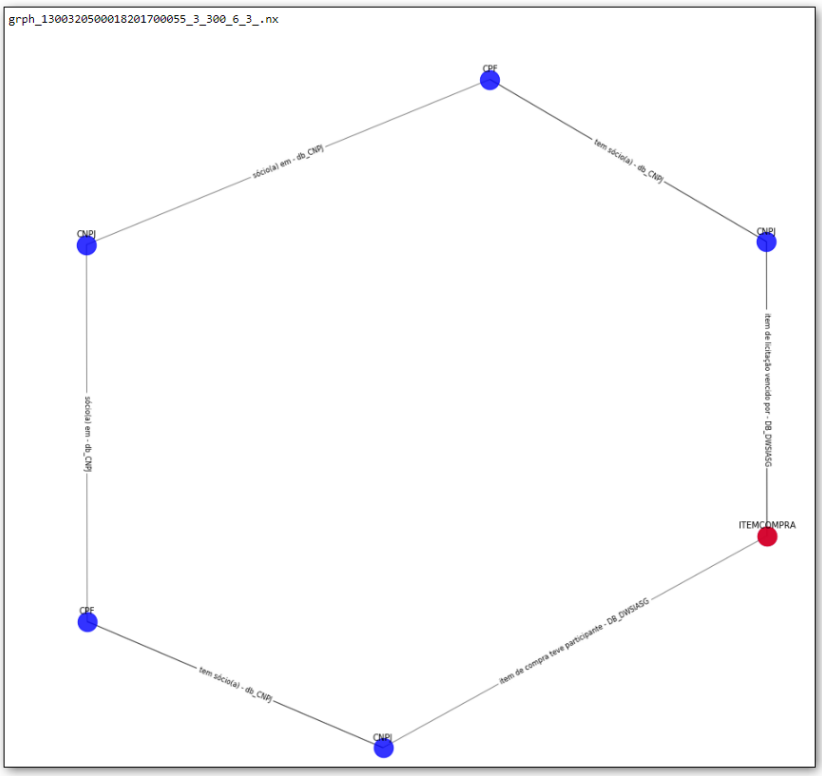


Fonte: Elaborada pelo autor (2020).

Em ciclos maiores, conexões mais "distantes" começam a aparecer, às vezes mais ou menos relevantes conforme o caso. Na Figura 16 encontra-se um caso que acabou se mostrando bastante comum: os respectivos sócios de duas empresas concorrentes, aparentemente sem conexão entre elas, têm uma terceira empresa na qual são sócios em comum.

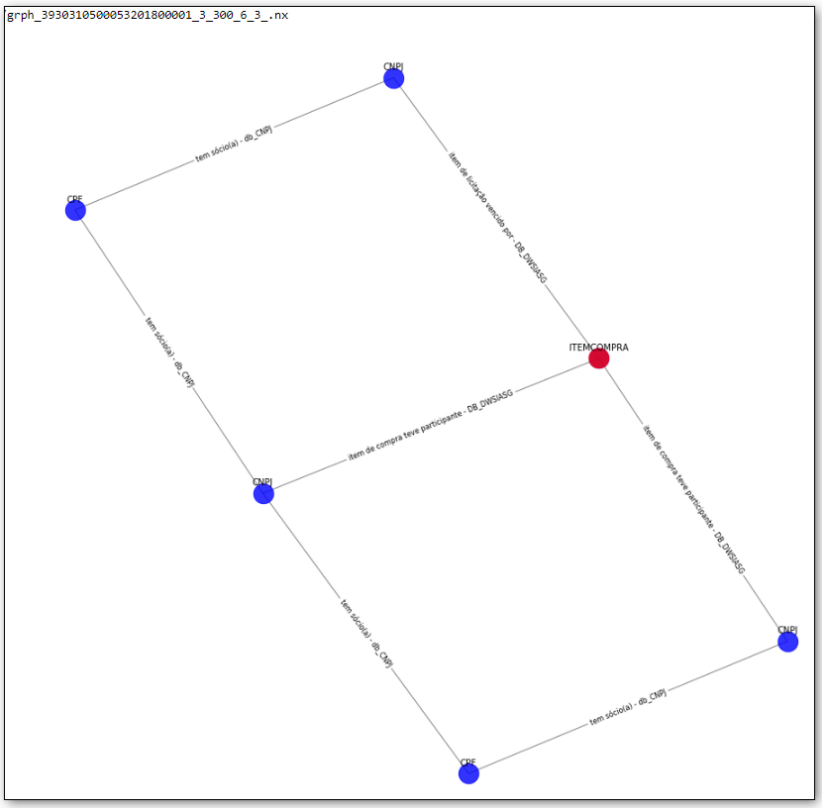
A importância da detecção da "quantidade de vínculos" não aparece somente no tamanho dos ciclos. Vários ciclos curtos "paralelos" também indicam situações indesejadas. A Figura 17 demonstra três empresas que parecem estar concorrendo, mas que aparentemente são controladas por duas pessoas, sócias em comum em uma das empresas. Essa situação apareceu nas duas Unidades da Federação por mais de uma vez, com variações.

Figura 16 – Sócios dos concorrentes são sócios em comum - MG.



Fonte: Elaborada pelo autor (2020).

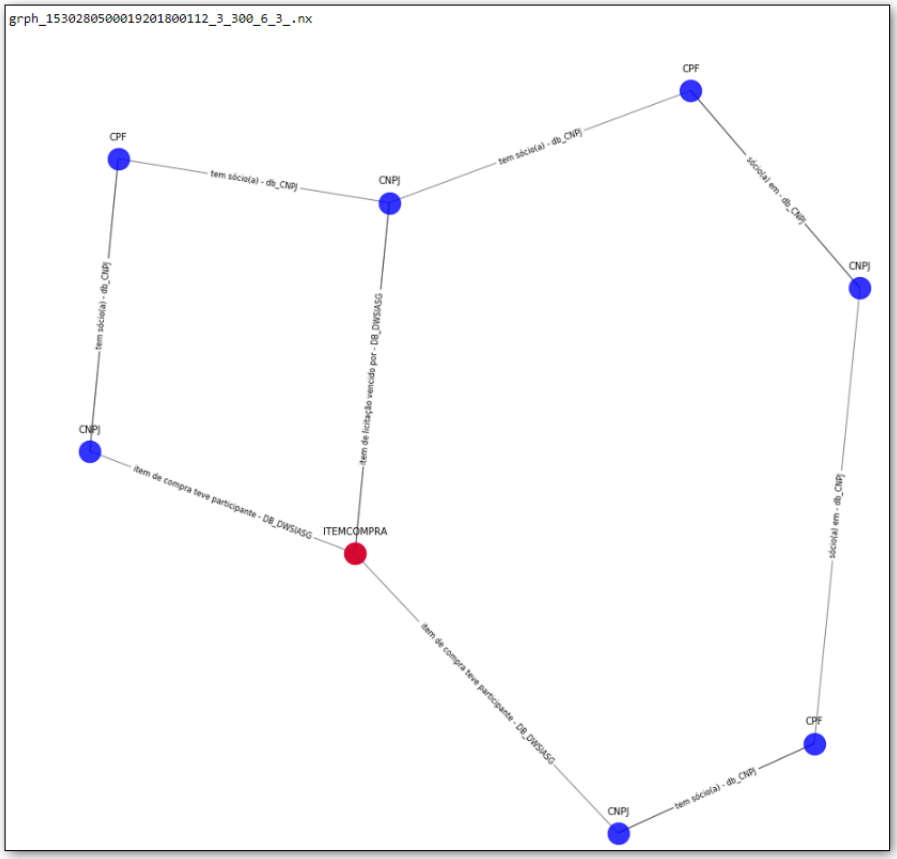
Figura 17 – Tripla “concorrência” simulada - MG.



Fonte: Elaborada pelo autor (2020).

Vários dos casos mostrados aparecem também em combinações distintas. A Figura 18 mostra a situação em que a tripla concorrência simulada foi detectada combinando a situação-alvo (Figura 11) e a situação em que os sócios tinham uma sociedade "oculta" em comum (Figura 16).

Figura 18 – Tripla "concorrência": situação-alvo + "sócio-do-sócio" - MG.

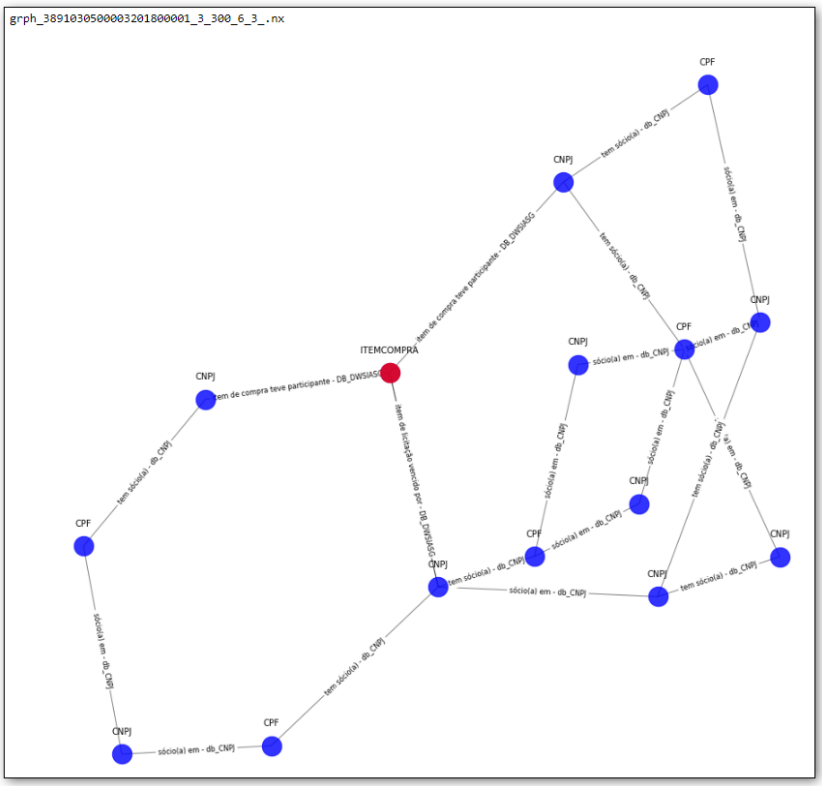


Fonte: Elaborada pelo autor (2020).

Naturalmente, não há limite para a complexidade das situações detectadas. Casos como os da Figura 19 demonstram bem a dificuldade da análise. Assim como no lado esquerdo, há dois vínculos de sociedade "oculta" (como da Figura 16) do lado direito da figura, mas são de difícil detecção à primeira vista. Ali também há vínculos que passam por sociedades "diretas" entre CNPJs.

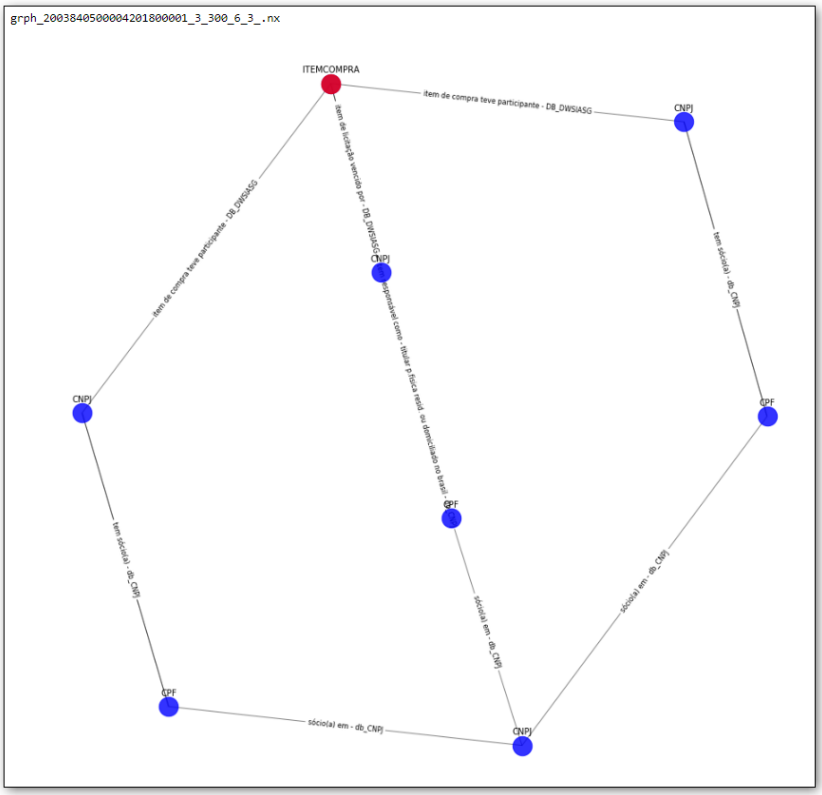
Embora haja "enfraquecimento" do vínculo com a distância, uma maior quantidade de vínculos pode vir a compensar por essa deficiência, a exemplo da Figura 20, onde fica demonstrada a "tripla concorrência" com três "sociedades ocultas", detectada no DF.

Figura 19 – Tripla “concorrência”: “sócio-do-sócio”, etc. - MG.



Fonte: Elaborada pelo autor (2020).

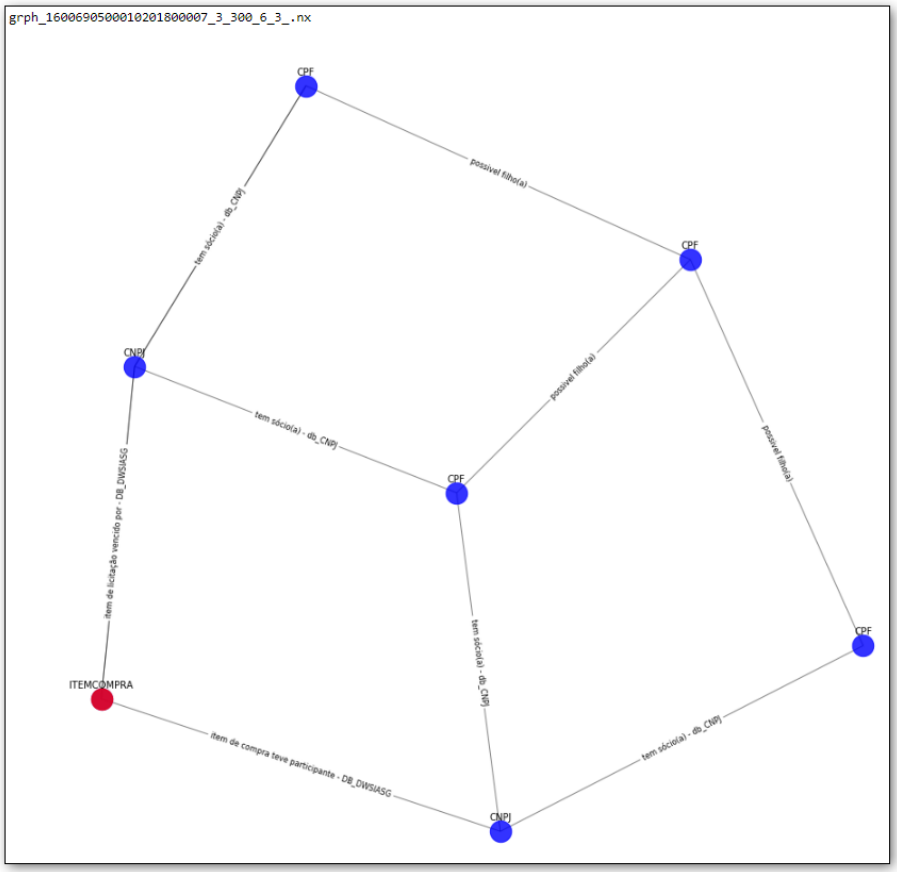
Figura 20 – Tripla concorrência, conexão distante - DF.



Fonte: Elaborada pelo autor (2020).

Outra demonstração da força da técnica de pesquisa no grafo é a informação complementar obtida sem qualquer necessidade de especificação prévia, limitada apenas pela especificação do tamanho máximo de ciclo na aplicação do algoritmo de busca.

Figura 21 – Situação-alvo: informações “complementares” - DF.



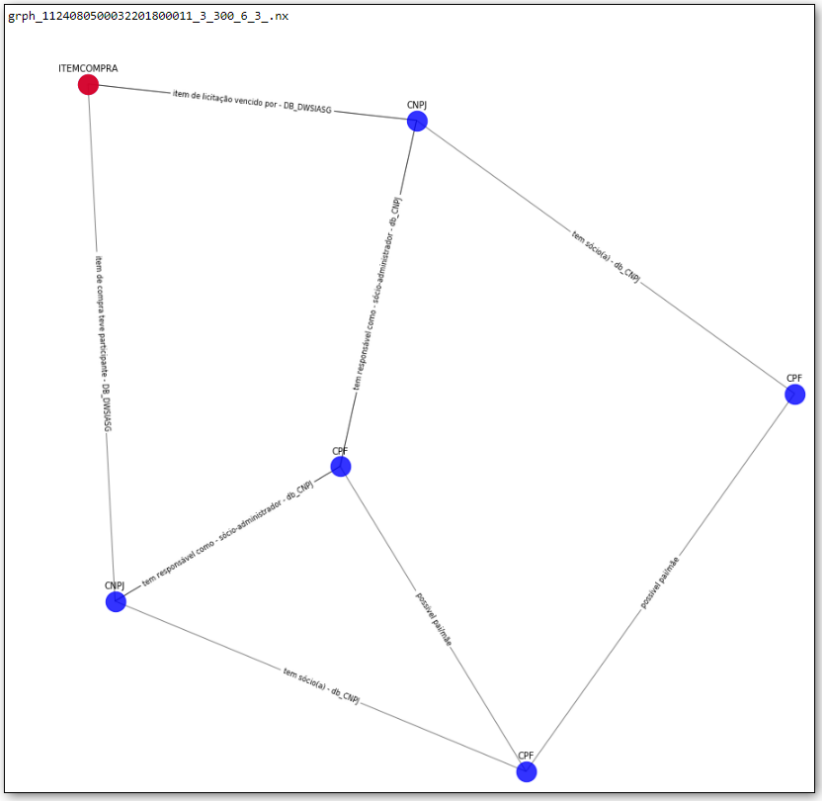
Fonte: Elaborada pelo autor (2020).

Como se observa na Figura 21, assim como na Figura 22, a “proximidade” da conexão entre as empresas fica muito mais evidente ao se perceber que, além de ter um sócio em comum, as empresas pertencem a irmãos.

Por fim, outras situações que aparecem em grande quantidade sugerem a existência de “grupos de influência” ou “aglomerados” agindo por trás dos processos licitatórios. A Figura 23 mostra seis pessoas, sócias em uma empresa que não aparece no processo licitatório, simulando a concorrência entre a vencedora e outra participante.

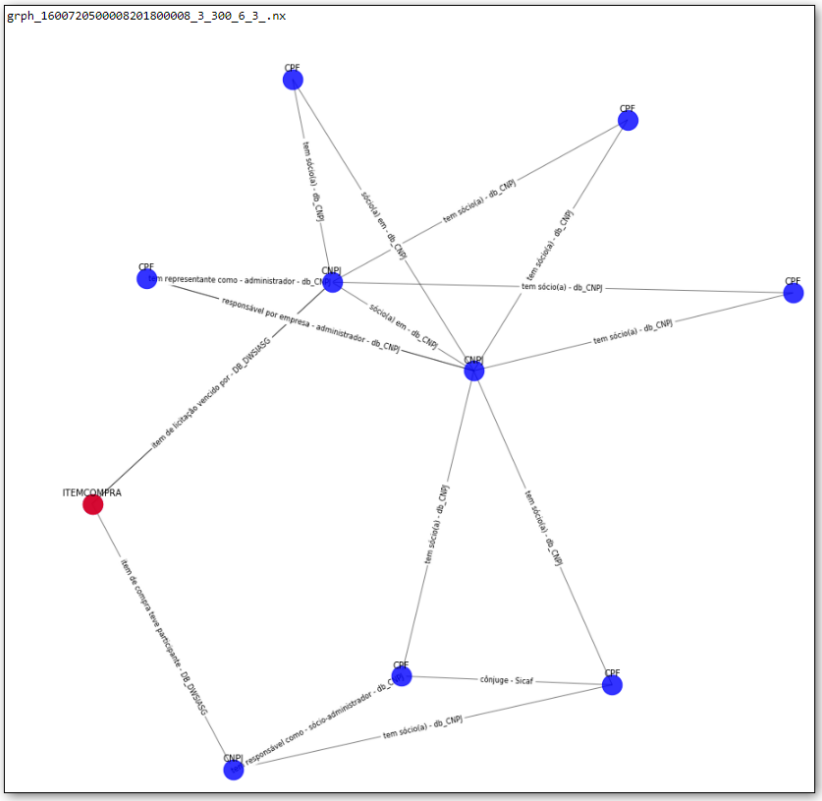
A situação chega a outras proporções na Figura 24, onde quinze pessoas, todas sócias em duas empresas em comum, entre elas a vencedora da licitação, simulam competição por meio de uma empresa controlada por outra empresa do quadro societário, em uma aparente tentativa de acobertamento.

Figura 22 – Situação-alvo: informações “complementares” (2) - DF.



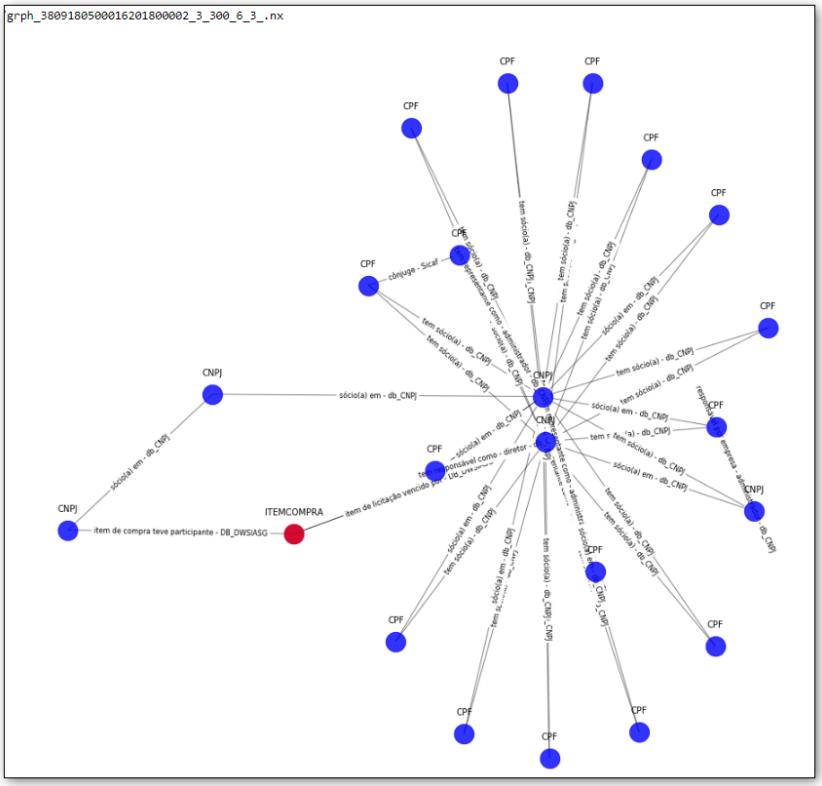
Fonte: Elaborada pelo autor (2020).

Figura 23 – Grupos de influência - DF.



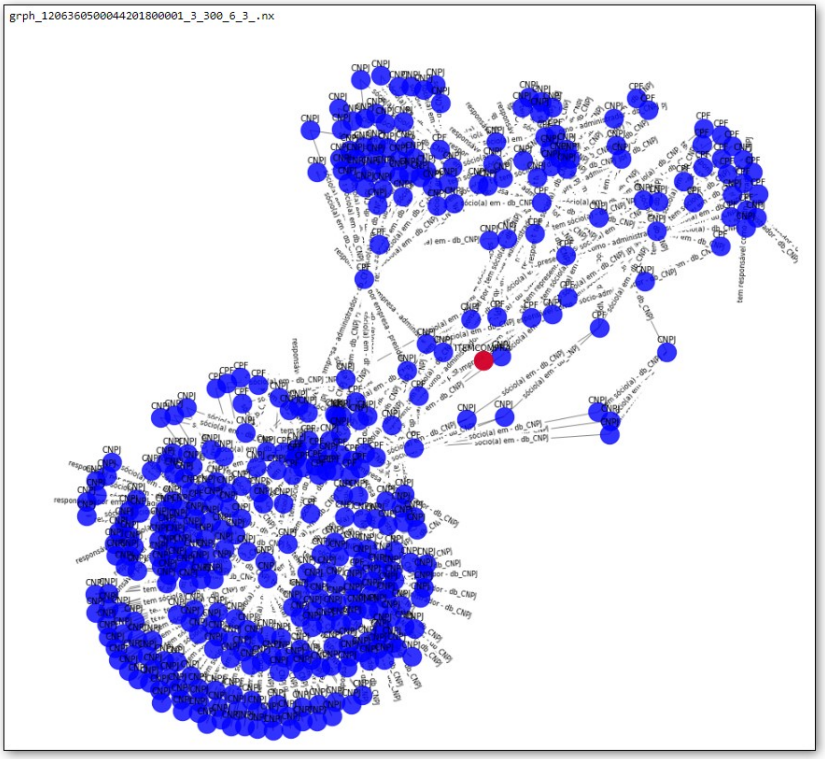
Fonte: Elaborada pelo autor (2020).

Figura 24 – Grupos de influência (2) - DF.



Fonte: Elaborada pelo autor (2020).

Figura 25 – Detecção de "aglomerados" - MG.



Fonte: Elaborada pelo autor (2020).

A Figura 25 apareceu diversas vezes no DF e em MG. Esse verdadeiro aglomerado de pessoas e empresas chama bastante à atenção e sua configuração é praticamente a mesma em todos os casos, fato que levou a apuração mais detalhada ainda durante este trabalho. Trata-se de licitações de contratos de serviços de telefonia, e os aglomerados que podem ser vistos nas figuras são as grandes empresas que atuam neste setor no Brasil.

6.4.2 Regras de associação

Com 1495 itens de compra com 'simulação de competição' em apenas duas UF, facilmente percebe-se que a inspeção visual não seria viável ao avaliar recortes diferentes, seja o país inteiro ou com uma maior abrangência temporal.

Para viabilizar o uso da informação pelas equipes de auditoria, no mínimo tem que ser possível encaminhar as informações acompanhadas de algum critério que permita a priorização da análise dos casos, ficando a quantidade a ser auditada por conta do planejamento da auditoria.

As questões que precisariam ser esclarecidas ao priorizar seriam: com que frequência ocorre essa "associação" entre as empresas relacionadas em relação à participação dessas empresas "sozinhas"; e, com que frequência essa "associação" ocorreu em relação à quantidade de licitações "similares".

Essas questões guardam bastante similaridade com os conceitos de confiança e suporte das regras de associação, já apresentados, bastando fazer a definição de como seria encontrado o grupo de licitações "similares" para tornar operacional a automação da rotina.

Ao se utilizar o cálculo das regras de associação, aproveitamos ainda outra vantagem: têm-se os valores das métricas para as demais relações nos grupos "similares" para a comparação.

Para extração de um grupo "similar" no SIASG foi elaborado o seguinte artifício: a partir do item de compra "do ciclo detectado" localizam-se todos os concorrentes; a partir dos concorrentes localizam-se todos os itens de compra dos quais eles participaram (parte-se da premissa que os demais itens são similares, da mesma área de atuação da empresa); a partir dessa lista de itens de compras, localiza-se por fim todos os participantes item a item.

O banco de dados[23] do qual sai o cálculo de regras de associação para cada item de compra "do ciclo detectado" vai ter como transação os itens de compra encontrados (que são considerados "similares") e como elementos os concorrentes nesses últimos itens.

Para o cálculo das regras de associação foi utilizada a biblioteca *mlxtend*, o código fonte se encontra no Apêndice B.3.

O cálculo do suporte dos conjuntos de itens foi feito com a função *apriori*, utilizando como parâmetro de suporte mínimo o menor suporte entre as empresas alvo (as empresas detectadas no ciclo). O maior número de elementos no conjunto foi estabelecido como "2", que foi considerado suficiente em vista dos casos observados em MG e DF (máximo de três empresas alvo, desejável que estivessem representadas em ambos os lados da regra).

O cálculo das regras de associação foi feito com a função *association_rules*, utilizando como métrica a confiança e parâmetro mínimo calculado a partir do suporte das empresas alvo (para garantir que as relações apareçam nos resultados) da forma a seguir.

Considerando a equação (5.1) e:

$$\min\big(sup(X => Y)\big) = \sup(X)\,.\,\sup(Y) \tag{6.1}$$

Tem-se, por consequência:

$$\min\big(conf(X => Y); conf(Y => X)\big) = \frac{\sup(X)\,.\,\sup(Y)}{\max(\sup(X)\,;\sup(Y))} \tag{6.2}$$

A utilização dos valores de mínimo suporte e confiança trouxe o menor conjunto de transações e de regras que continham as informações de interesse. Algumas exceções surgiram e precisaram de tratamento no código, a exemplo da licitação que não tinha outros participantes além dos envolvidos no ciclo e estes não participavam de nenhum outro item de licitação.

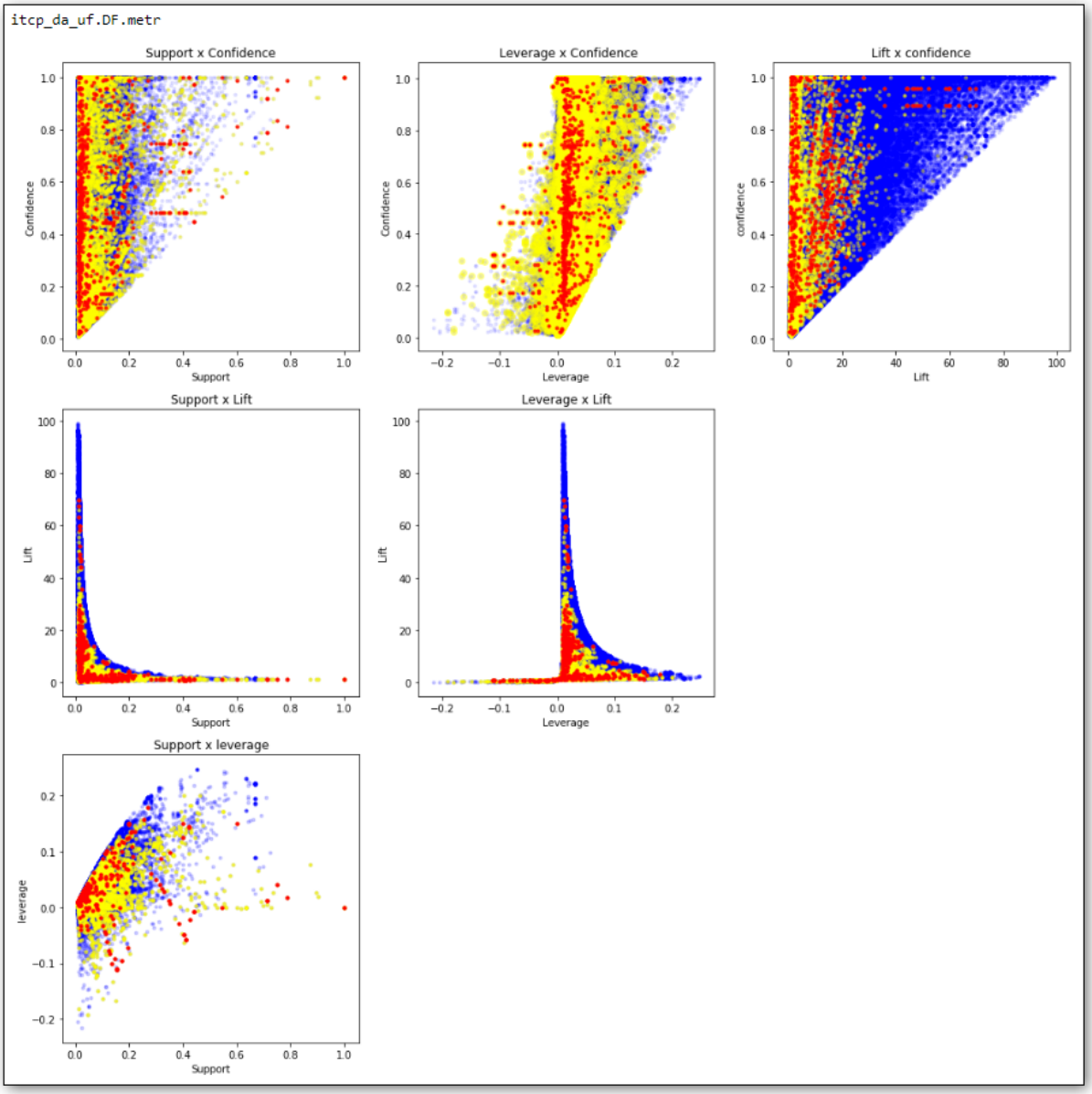
6.4.3 Inspeção visual das métricas

Nas Figuras 26 e 27, as métricas de cada regra (X=>Y) encontrada, para os itens da UF, foram impressas em cor **azul**. A seguir, em **amarelo**, foram impressas as regras nas quais as empresas-alvo apareceram em um dos lados, isto é, as empresas que "simularam competi-

[23] Como definido em 5.3

ção" aparecem com outras empresas. Por último, em **vermelho**, foram impressas as regras em cujos relacionamentos as empresas-alvo aparecem tanto no antecedente como no consequente, ou seja, essas representam de fato a repetição da situação de "simulação de competição", onde essas empresas participam juntas em um mesmo item de compra.

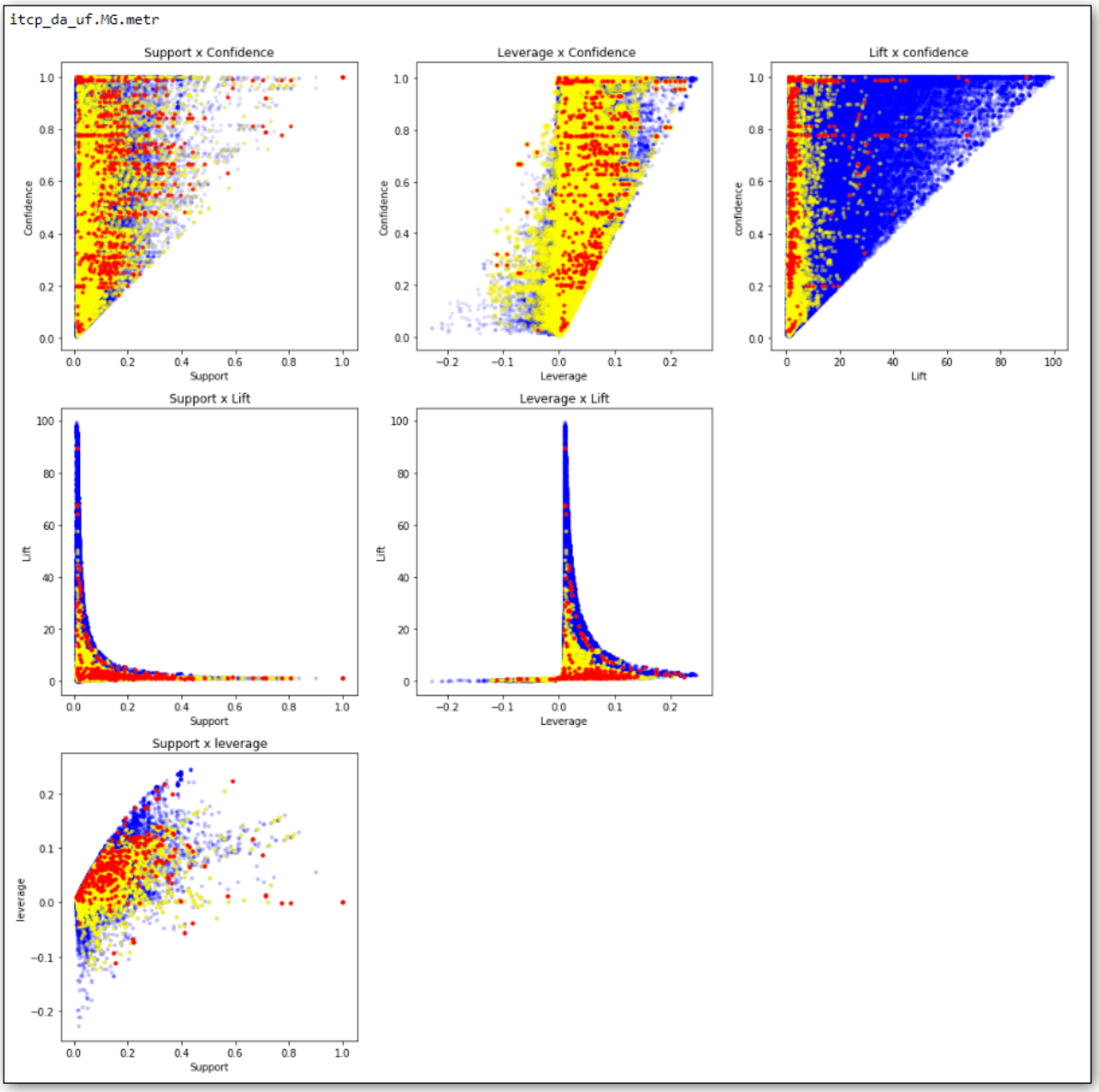
Figura 26 – Métricas de todas as regras - DF.



Fonte: Elaborada pelo autor (2020).

O único aspecto que chama a atenção é que os pontos vermelhos não se distribuem uniformemente pela mesma área ocupada pelos pontos azuis e amarelos, ou seja, parece que as empresas se "comportam" de maneira diferente ao simular competição.

Figura 27 – Métricas de todas as regras - MG.



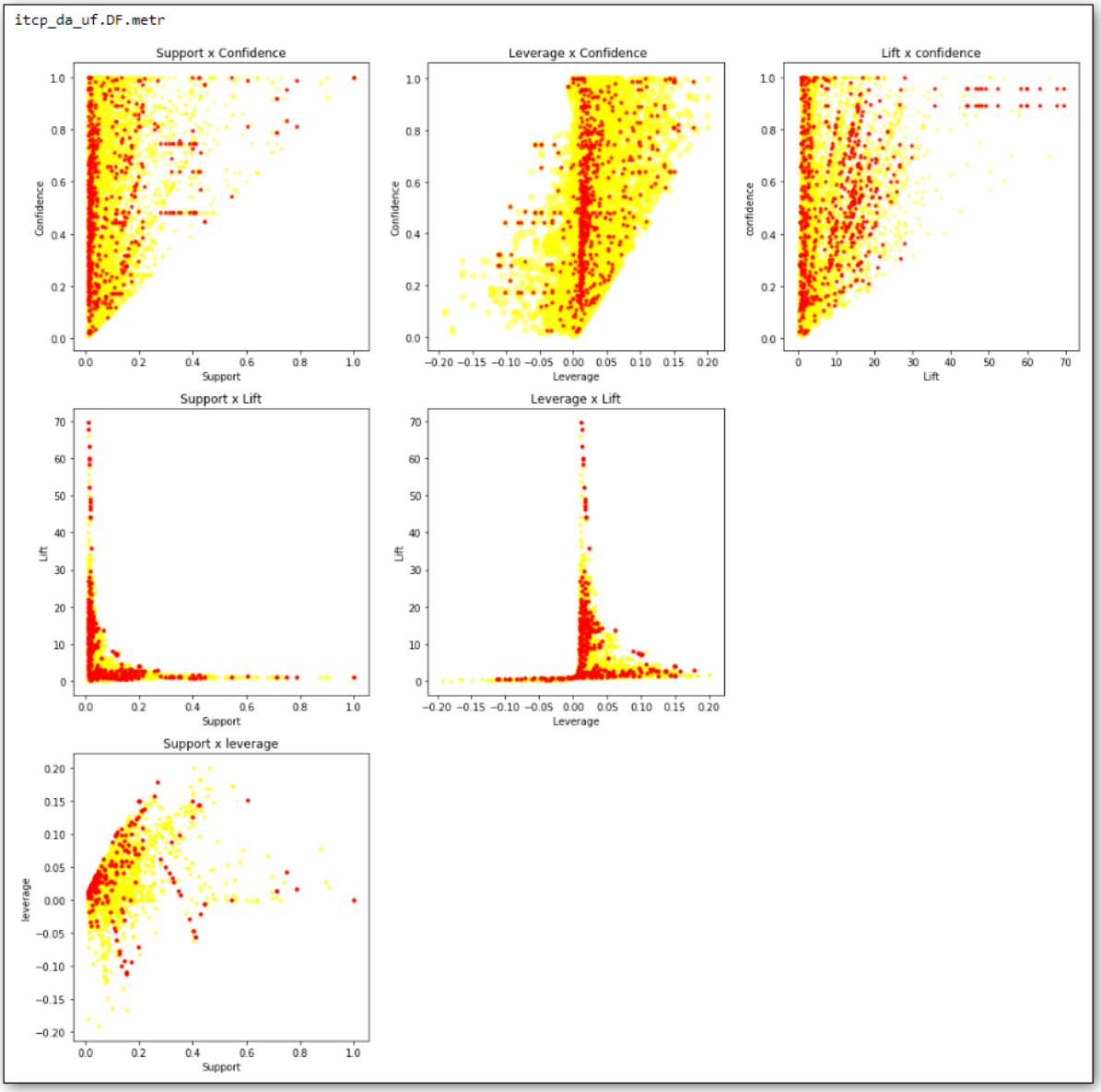
Fonte: Elaborada pelo autor (2020).

Infelizmente, na análise realizada durante o andamento deste trabalho, nenhum padrão foi delineado que pudesse ajudar a prever a ocorrência da "simulação de competição" a partir das métricas calculadas, ficando como sugestão para trabalhos futuros.

Como nota complementar, é interessante notar que a visualização permite a comparação do comportamento dos grupos entre as diferentes Unidades da Federação. Em Minas Gerais há um "espalhamento" dos pontos de mesma confiança no eixo do suporte (pontos vermelhos das Figuras 26 e 27), indicando que lá as empresas "arriscam" mais a "simulação de concorrência" nas licitações do que no Distrito Federal.

Com o intuito de priorizar a análise dos casos encontrados, foi proposta uma segunda visualização, incluindo somente as regras em que as empresas alvo participam de alguma forma, ou seja, apenas os pontos amarelos e vermelhos das figuras apresentadas.

Figura 28 – Métricas das regras contendo as empresas alvo - DF.

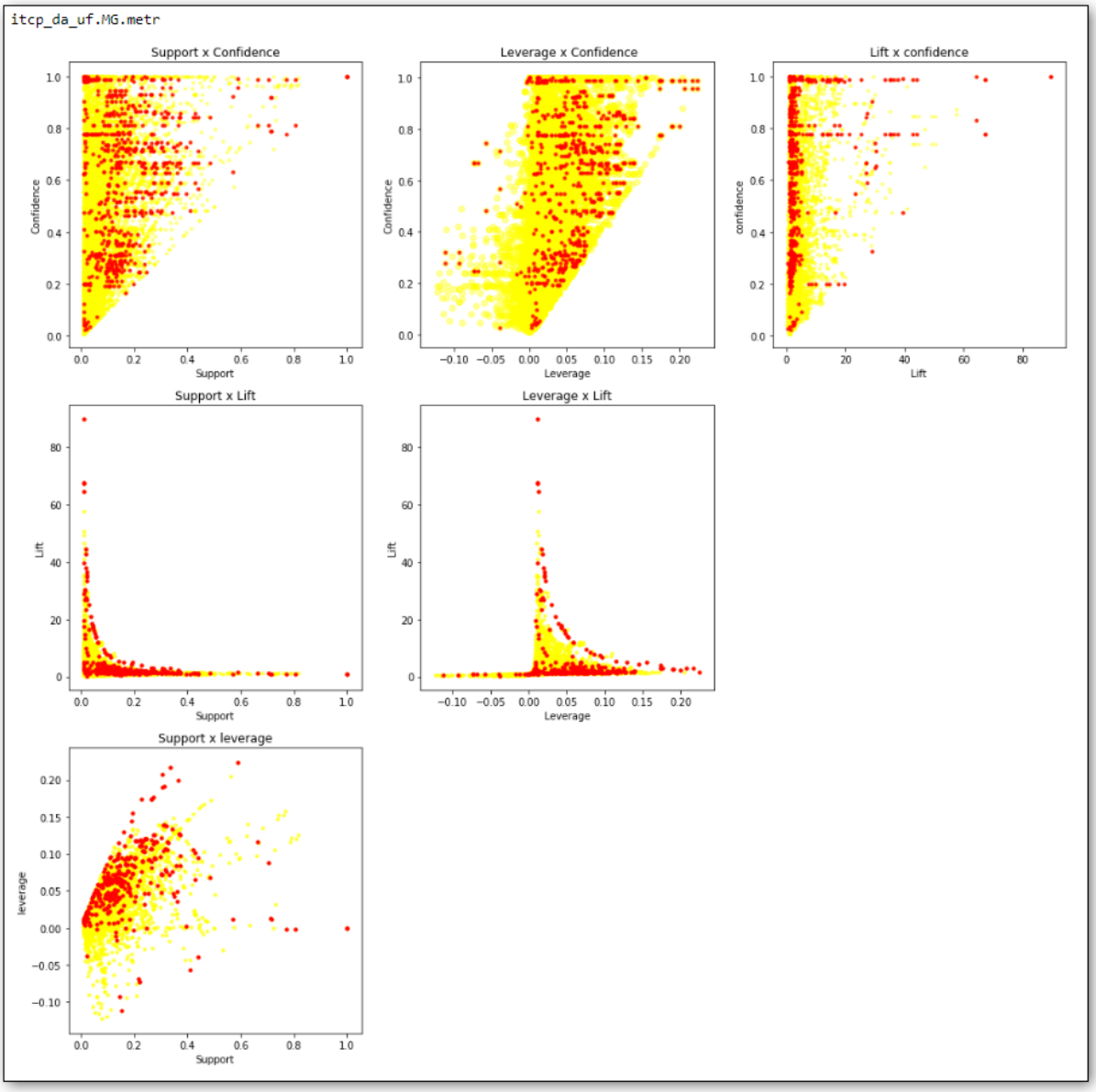


Fonte: Elaborada pelo autor (2020).

6.4.4 Sugestão de priorização para análise

Para seleção para auditoria, em vista da distribuição dos pontos e do significado das métricas avaliadas, foram consideradas como mais relevantes o suporte e a confiança, representadas no primeiro dos "sub-gráficos", em cima à esquerda, nas Figuras 28 e 29.

Figura 29 – Métricas das regras contendo as empresas alvo - MG.



Fonte: Elaborada pelo autor (2020).

O suporte como primeira regra a analisar, indica a frequência com que aquela regra apareceu em meio às demais, representando, dessa forma, a frequência em que aquelas empresas se apresentaram para "simular concorrência" no mesmo item de compra.

A confiança representa a relação entre a frequência em que essas empresas "concorrem juntas" e a frequência em que elas "concorrem separadas" com outras empresas.

Pode-se então sugerir que seria adequado priorizar a análise dos ciclos cujas regras acabaram representadas no quadrante superior direito do gráfico Suporte vs. Confiança – valores altos em ambas as métricas.

A seguir, viriam aqueles cujo suporte já não é tão grande, ou seja, não ocorreram com tanta frequência, mas cuja confiança ainda é alta, isso é, as empresas estão mais frequentemente "juntas" que "separadas" em licitações.

Ao se chegar à região de menor suporte e confiança ainda alta, onde se observa um adensamento dos pontos, sugere-se o uso das métricas de lift e leverage como critérios de priorização, caso necessário. Por último seriam avaliados os demais casos, com confiança cada vez menor.

7 CONCLUSÕES

O banco de vínculos é um recurso bastante interessante, cuja utilização não é trivial e cuja manutenção somente seria possível em um Órgão estatal com acesso a muitas fontes de informação distintas e, principalmente, com legitimidade para o uso de tais informações. É um recurso do qual poucos Órgãos além dos Órgãos de Controle poderiam dispor.

O presente trabalho surgiu em meio a uma situação de inviabilidade na exploração desse recurso, embasado em outros projetos cujo sucesso obtido não foi mais que parcial.

A situação inicial enfrentada era de mais de 24h de processamento para obtenção das informações de apenas um item de compra (conforme destacado em 6.2.1). A partir daí se conseguiu evoluir para uma unidade (UASG) pequena, depois para uma unidade maior, para uma Unidade da Federação com poucas unidades e, por fim, para uma realidade de processamento de um ano inteiro de compras de duas das três maiores UF, "de um dia para o outro".

A questão proposta foi adequadamente respondida, demonstrando-se que é computacionalmente viável a identificação de indícios de "simulação de concorrência" por meio da investigação da topologia do grafo no banco de vínculos da CGU e que os resultados encontrados são relevantes.

Mais que isso, a priorização dos resultados com auxílio de regras de associação trouxe um viés de real aplicabilidade aos resultados gerados, entregando os resultados de forma adequada para a eventual inclusão no ciclo de auditoria.

Com a velocidade de processamento obtida, não é difícil imaginar que as verificações possam ser incorporadas à carga trimestral do SIASG realizada pela CGU no banco de dados. Estima-se que a verificação dos dados do último trimestre, se realizada de forma paralela com relação às UF, poderia ser completada em dois ou três dias, gerando uma informação relevante e quase concomitante sobre os itens de compra recém-licitados.

Como extensão do trabalho, um grande número de informações complementares pode ainda ser obtido de outros bancos de dados disponíveis no CGUDATA, por exemplo, a verificação da concomitância entre os vínculos societários e as licitações, que fica como sugestão para desenvolvimento futuro.

Uma possibilidade importante a explorar seria a análise da disputa de preço, principalmente nos casos de pregão, onde há a diversas informações dos lances a considerar, tais como o tempo entre os lances e a diferença percentual entre eles. Essa informação poderia ser

incorporada ao processo para auxiliar na priorização dos casos, ou também explorada de uma forma mais arrojada: utilizando os lances dos casos encontrados para um aprendizado de máquina visando encontrar empresas que seguem o mesmo padrão, ou seja, que simulam competição mesmo não tendo os vínculos em comum (ou tendo vínculos que ainda não são mapeados no Yggdrasil).

Por último, alcançando um dos maiores objetivos no que diz respeito à utilidade e aplicação para a CGU, os algoritmos gerados poderão ser adaptados a um sem número de casos de uso cuja necessidade já foi detectada, tais como detecção de nepotismo e apuração de denúncias de contratação/favorecimento de empresas ligadas a gestores.

REFERÊNCIAS

[1] TAN, Pang-Ning. STEINBACH, Michael Scott. KUMAR, Vipin. **Introduction to Data Mining.** 1ª Edição. Boston: Addison-Wesley Longman, 2005.

[2] GOLDBARG, Marco Cesar. GOLDBARG, Elizabeth. **Grafos: conceitos, algoritmos e aplicações.** 1ª Edição. Rio de Janeiro: Elsevier, 2012.

[3] ZAKI, Mohammed. MEIRA Jr., Wagner. **Data mining and analysis.** 1ª Edição. New York: Cambridge University Press, 2014.

[4] CORDEIRO Galvão van Erven, Gustavo. CARVALHO, Rommel. HOLANDA, Maristela. RALHA, Célia. (2017). **Graph database: A case study for detecting fraud in acquisition of Brazilian Government.** 1-6. 10.23919/CISTI.2017.7975974.

[5] KAVITHA, Telikepalli. LIEBCHEN, Christian. MEHLHORN, Kurt. RIZZI, Dimitrios Michail Romeo. UECKERDT, Torsten. ZWEIG, Katharina A. (2009). **Cycle Bases in Graphs Characterization, Algorithms, Complexity, and Applications**. https://people.mpi-inf.mpg.de/~mehlhorn/ftp/SurveyCycleBases.pdf

[6] XIONG, Hui. TAN, Pang-Ning. KUMAR , Vipin. (2005) **Hyperclique Pattern Discovery**. https://www-users.cs.umn.edu/~kumar001/papers/clique_dmkd.pdf

[7] JOHNSON, D. B. (1975) **Finding all the elementary circuits of a directed graph**. SIAM Journal on Computing 4, no. 1, 77-84

[8] ERVEN, Gustavo C. Galvão Van. **MDG-NoSQL: Modelo de dados para bancos NoSQL baseados em grafos**. Dissertação (Mestrado Profissional em Computação Aplicada). Universidade de Brasília, Brasília, 2015.

[9] GISCARD, Pierre-Louis. KRIEGE, Nils. WILSON, Richard C. **General purpose algorithm for counting simple cycles and simple paths of any length.** 1612.05531v1 [cs.DS], 2016

[10] MEHTA, Priya et al. **A graph theoretical approach for identifying fraudulent transactions in circular trading.** The Sixth International Conference on Data Analytics, 2017.

[11] ERVEN, Gustavo C. G. V.; HOLANDA, Maristela; CARVALHO, Rommel N. **Detecting evidence of fraud in the brazilian government using graph databases**. In: World conference on information systems and technologies. Springer, Cham, 2017. p. 464-473.

[12] SALES, Leonardo Jorge. P**roposta de modelo de classificação do risco de contratos públicos**. Dissertação (Mestrado em Economia do Setor Público). Universidade de Brasília, Brasília, 2016.

[13] MAGOMEDOV, Shamil et al. **Anomaly detection with machine learning and graph databases in fraud management.** International Journal of Advanced Computer Science and Applications, v. 9, p. 33-38, 2018.

[14] KOU, Yufeng et al. **Survey of fraud detection techniques**. In: IEEE International Conference on Networking, Sensing and Control, 2004. IEEE, 2004. p. 749-754.

[15] PHUA, Clifton et al. **A comprehensive survey of data mining-based fraud detection research**. arXiv preprint arXiv:1009.6119, 2010.

APÊNDICE A – Código SQL

A.1 - Levantamento inicial

```
drop table if exists dbo.levantamento_inicial
drop table if exists #mesmo_socio
drop table if exists #socio
drop table if exists #forn


SELECT distinct
f.ID_FRND_FORNECEDOR_COMPRA as cnpj1

into #forn

FROM db_dwsiasg.dbo.F_ITEM_FORNECEDOR as f;



select aa.*, cc.CpfCnpjSocio

into #socio

from #forn aa

inner join [db_CNPJ].[dbo].[Socios] cc
   on cc.Cnpj = aa.cnpj1;



select prim.*, seg.cnpj1 as cnpj2

into #mesmo_socio

from #socio  prim

inner join #socio  seg
   on prim.CpfCnpjSocio = seg.CpfCnpjSocio

inner join db_CPF.dbo.CPF dd
   on dd.CPF = prim.CpfCnpjSocio

inner join db_CPF.dbo.CPF ee
   on ee.CPF = seg.CpfCnpjSocio

where prim.cnpj1 < seg.cnpj1;



SELECT ff.*, ic.ID_ITCP_ITEM_COMPRA,
   CAST(ic.CH_ITCP_ITEM_COMPRA as VARCHAR(22)) item_compra
into dbo.levantamento_inicial

from #mesmo_socio ff

inner join db_dwsiasg.dbo.F_ITEM_FORNECEDOR as f
   on ff.cnpj1 = f.ID_FRND_FORNECEDOR_COMPRA

INNER JOIN db_dwsiasg.dbo.D_ITCP_ITEM_COMPRA as ic
   ON f.ID_ITCP_ITEM_COMPRA = ic.ID_ITCP_ITEM_COMPRA

inner join db_dwsiasg.dbo.F_ITEM_FORNECEDOR as fr
   on ff.cnpj2 = fr.ID_FRND_FORNECEDOR_COMPRA

INNER JOIN db_dwsiasg.dbo.D_ITCP_ITEM_COMPRA as icr
   ON fr.ID_ITCP_ITEM_COMPRA = icr.ID_ITCP_ITEM_COMPRA

where ic.ID_ITCP_ITEM_COMPRA = icr.ID_ITCP_ITEM_COMPRA;

drop table if exists #mesmo_socio
drop table if exists #socio
drop table if exists #forn
```

A.2 - Extração do Yggdrasil

```
USE yggdrasil;

SET QUOTED_IDENTIFIER on
GO

DECLARE @tipo varchar(15) = 'ITEMCOMPRA';
DECLARE@grupo varchar(20) = 'PADRAO';
DECLARE@niveis_a_pesquisar int = 3;

DECLARE @nivel int;
DECLARE @linhas_antes int;
DECLARE@linhas_depois int;

SET NOCOUNT ON

IF OBJECT_ID('tempdb.dbo.#consulta', 'U') IS NOT NULL drop table #consulta;
IF OBJECT_ID('tempdb.dbo.#temp_consulta', 'U') IS NOT NULL drop table #temp_consulta;

CREATE TABLE #consulta
    (
vertice_anterior bigint,
nivel int,
vertice_nivel_1 bigint,
vertice_consultado bigint,
qtd_aresta_vertice_consultado int,
tipo_vinculo int,
vertice_complementar bigint,
qtd_aresta_vertice_complementar int
    )

CREATE TABLE #temp_consulta
    (
id_temp bigint NOT NULL IDENTITY ( 1,1 ),
vertice_anterior bigint,
nivel int,
vertice_nivel_1 bigint,
vertice_consultado bigint,
qtd_aresta_vertice_consultado int,
tipo_vinculo int,
vertice_complementar bigint,
qtd_aresta_vertice_complementar int
    )
-----------------------------------------------------------

IF OBJECT_ID('tempdb.dbo.#ITEM_COMPRA_2018', 'U') IS NOT NULL drop table #ITEM_COMPRA_2018;

SELECT DISTINCT ic.[CH_ITCP_ITEM_COMPRA_EDIT]

INTO #ITEM_COMPRA_2018

FROM [db_dwsiasg].[dbo].[D_ITCP_ITEM_COMPRA] as ic

INNER JOIN db_dwsiasg.dbo.F_ITEM_UNIDADE as iu
ON ic.ID_ITCP_ITEM_COMPRA = iu.ID_ITCP_ITEM_COMPRA

INNER JOIN db_dwsiasg.dbo.D_UNDD_UNIDADE as un
ON iu.ID_UNDD_UNID_PARTICIPANTE = un.ID_UNDD_UNIDADE

INNER JOIN db_dwsiasg.dbo.F_ITEM_COMPRA as ficomp
ON ic.ID_ITCP_ITEM_COMPRA = ficomp.ID_ITCP_ITEM_COMPRA

INNER JOIN db_dwsiasg.dbo.D_CMPR_COMPRA as comp
ON ficomp.ID_CMPR_COMPRA = comp.ID_CMPR_COMPRA

where year(comp.DT_REFERENCIA_COMPRA) = '2018'
     AND un.ID_UNDD_UNIDADE in ( 200141, 160204, 194018)
-----------------------------------------------------------------

insert into #consulta

select
'' as vertice_anterior,
1 as nivel,
a_v2.id_vertice2 as vertice_nivel_1,
a_v2.id_vertice2 as vertice_consultado,
null  as qtd_aresta_vertice_consultado,
tp_vinc.id_tipo_reciproco as tipo_vinculo,
a_v2.id_vertice1 as vertice_complementar,
null  as qtd_aresta_vertice_complementar

from [dbo].[yggdrasil_tipo_identificador] tp_id
```

```
inner join [dbo].[yggdrasil_vertice] v_ini
on v_ini.[id_tp_identificador] = tp_id.id_tp_identificador

INNER JOIN #ITEM_COMPRA_2018 ICC
ON ICC.CH_ITCP_ITEM_COMPRA_EDIT = v_ini.identificador

inner join [yggdrasil_aresta_detalhada] a_v2
on a_v2.id_vertice2 = v_ini.id_vertice

inner join [dbo].[yggdrasil_tipo_vinculo] tp_vinc
on tp_vinc.id_tipo_vinculo = a_v2.id_tipo_vinculo

inner join [dbo].[vw_Grupos_tipos_pesos] vwGtp with (noexpand)
on vwGtp.grupo = @grupo
and tp_vinc.id_tipo_reciproco = vwGtp.id_tipo_vinculo

-----------------------------------------------------------

set @nivel = 2

WHILE @nivel <= @niveis_a_pesquisar
begin

select @linhas_antes = (select count(*) from #consulta)

raiserror('Iniciando nivel %i com %i linhas antes', 10, 1, @nivel, @linhas_antes) with nowait

insert into #temp_consulta

        select
vertice_a_consultar.vertice_anterior as vertice_anterior,
@nivel as nivel,
vertice_a_consultar.vertice_nivel_1,
a_v1.id_vertice1 as vertice_consultado,
null as qtd_aresta_vertice_consultado,
a_v1.id_tipo_vinculo as tipo_vinculo,
a_v1.id_vertice2 as vertice_complementar,
null as qtd_aresta_vertice_complementar

from   (SELECT distinct
#consulta.vertice_nivel_1,
#consulta.vertice_consultado as vertice_anterior,
#consulta.vertice_complementar as vertice_a_consultar

FROM #consulta

where 1=1
and #consulta.nivel = (@nivel-1)

) as vertice_a_consultar

inner join  [yggdrasil_aresta_detalhada] a_v1
on a_v1.id_vertice1 = vertice_a_consultar.vertice_a_consultar
and a_v1.id_vertice2 <> vertice_a_consultar.vertice_anterior

inner join [dbo].[vw_Grupos_tipos_pesos] vwGtp with (noexpand)
on vwGtp.grupo = @grupo
and a_v1.id_tipo_vinculo = vwGtp.id_tipo_vinculo

LEFT JOIN #consulta as vertice_repetido
on a_v1.id_vertice1 = vertice_repetido.vertice_consultado
and a_v1.id_vertice2 = vertice_repetido.vertice_complementar
and vertice_a_consultar.vertice_nivel_1 =
                                vertice_repetido.vertice_nivel_1
                    and vertice_repetido.nivel = (@nivel-1)

LEFT JOIN #consulta as vertice_repetido_inv
on a_v1.id_vertice2 = vertice_repetido_inv.vertice_consultado
and a_v1.id_vertice1 = vertice_repetido_inv.vertice_complementar
and vertice_a_consultar.vertice_nivel_1 =
                            vertice_repetido_inv.vertice_nivel_1
                    and vertice_repetido_inv.nivel = (@nivel-1)

where vertice_repetido.vertice_consultado is null
and vertice_repetido_inv.vertice_consultado is null

insert into #temp_consulta

select
vertice_a_consultar.vertice_anterior as vertice_anterior,
```

```
@nivel as nivel,
vertice_a_consultar.vertice_nivel_1,
a_v2.id_vertice2 as vertice_consultado,
null as qtd_aresta_vertice_consultado,
tp_vinc.id_tipo_reciproco as tipo_vinculo,
a_v2.id_vertice1 as vertice_complementar,
null as qtd_aresta_vertice_complementar

from (SELECT distinct
      #consulta.vertice_nivel_1,
      #consulta.vertice_consultado as vertice_anterior,
      #consulta.vertice_complementar as vertice_a_consultar

FROM #consulta

where 1=1
and #consulta.nivel = (@nivel-1)

) as vertice_a_consultar

inner join  [yggdrasil_aresta_detalhada] a_v2
on a_v2.id_vertice2 = Vertice_a_consultar.vertice_a_consultar
and a_v2.id_vertice1 <> vertice_a_consultar.vertice_anterior

inner join [dbo].[yggdrasil_tipo_vinculo] tp_vinc
on tp_vinc.id_tipo_vinculo = a_v2.id_tipo_vinculo

inner join [dbo].[vw_Grupos_tipos_pesos] vwGtp with (noexpand)
on vwGtp.grupo = @grupo
and tp_vinc.id_tipo_reciproco = vwGtp.id_tipo_vinculo

LEFT JOIN #consulta as vertice_repetido_inv
on a_v2.id_vertice2 = vertice_repetido_inv.vertice_consultado
and a_v2.id_vertice1 = vertice_repetido_inv.vertice_complementar
and vertice_a_consultar.vertice_nivel_1 =
                          vertice_repetido_inv.vertice_nivel_1
                    and vertice_repetido_inv.nivel = (@nivel-1)

LEFT JOIN #consulta as vertice_repetido
on a_v2.id_vertice1 = vertice_repetido.vertice_consultado
and a_v2.id_vertice2 = vertice_repetido.vertice_complementar
and vertice_a_consultar.vertice_nivel_1 =
                           vertice_repetido.vertice_nivel_1
                     and vertice_repetido.nivel = (@nivel-1)

where vertice_repetido.vertice_consultado is null
and vertice_repetido_inv.vertice_consultado is null

insert into #consulta

select vertice_anterior,
nivel,
vertice_nivel_1,
vertice_consultado,
qtd_aresta_vertice_consultado,
tipo_vinculo,
vertice_complementar,
qtd_aresta_vertice_complementar

from #temp_consulta

inner join (
select min(id_temp) as min_id_temp

from #temp_consulta t

inner join [dbo].[yggdrasil_tipo_vinculo] tp
on  t.tipo_vinculo = tp.id_tipo_vinculo

group by
case when vertice_consultado<vertice_complementar
 then cast(vertice_nivel_1 as varchar) + cast(vertice_consultado as varchar) + cast(vertice_complementar as varchar)
+ cast(t.tipo_vinculo as varchar)
 else cast(vertice_nivel_1 as varchar) + cast(vertice_complementar as varchar) + cast(vertice_consultado as varchar)
+ cast(tp.id_tipo_reciproco as varchar)
end
) temp
on temp.min_id_temp = #temp_consulta.id_temp

truncate table #temp_consulta

select @linhas_depois = (select count(*) from #consulta)

if @linhas_antes = @linhas_depois
```

```
begin set @nivel = @niveis_a_pesquisar+1 end
else
begin set @nivel = @nivel + 1 end
END;

IF OBJECT_ID('tempdb.dbo.#temp_consulta', 'U') IS NOT NULL drop table #temp_consulta;
IF OBJECT_ID('tempdb.dbo.#ITEM_COMPRA_2018', 'U') IS NOT NULL drop table #ITEM_COMPRA_2018;
-----------------------------------------------------------
raiserror('Iniciando final com %i linhas DEPOIS', 10, 1, @linhas_depois) with nowait;

DROP TABLE IF EXISTS [work_claudioags].dbo.TCC_05_PADRAO_3_NIVEIS

select
#consulta.nivel,
#consulta.vertice_nivel_1,
        #consulta.vertice_anterior,
        #consulta.vertice_consultado,
#consulta.qtd_aresta_vertice_consultado,
tp_id_v_cons.ds_tp_identificador tp_vert_cons,
#consulta.tipo_vinculo as id_tipo_vinculo,
vwGtp.tipo_vinculo,
#consulta.vertice_complementar,
#consulta.qtd_aresta_vertice_complementar,
tp_id_v_comp.ds_tp_identificador tp_vert_comp

into [work_claudioags].dbo.TCC_05_PADRAO_3_NIVEIS

from #consulta

inner join [dbo].[yggdrasil_vertice] vert_cons
on vert_cons.id_vertice = #consulta.vertice_consultado

inner join [dbo].[yggdrasil_tipo_identificador] tp_id_v_cons
on tp_id_v_cons.id_tp_identificador = vert_cons.id_tp_identificador

inner join [dbo].[yggdrasil_vertice] vert_comp
on vert_comp.id_vertice = #consulta.vertice_complementar

inner join [dbo].[yggdrasil_tipo_identificador] tp_id_v_comp
on tp_id_v_comp.id_tp_identificador =  vert_comp.id_tp_identificador

inner join [dbo].[vw_Grupos_tipos_pesos] vwGtp with (noexpand)
on vwGtp.id_tipo_vinculo = #consulta.tipo_vinculo
and vwGtp.grupo = @grupo

ORDER BY #consulta.nivel,#consulta.vertice_nivel_1,#consulta.vertice_anterior,#consulta.vertice_consultado

-----------------------------------------------------------
IF OBJECT_ID('tempdb.dbo.#consulta', 'U') IS NOT NULL drop table #consulta;

SET NOCOUNT OFF
GO
```

APÊNDICE B – Código Python

B.1 - Pesquisa dos ciclos

```
# coding: utf-8
import sqlalchemy as sa
import pandas as pd
import networkx as nx
import itertools
import numpy as np

from collections import defaultdict
from itertools import tee

from networkx.utils import not_implemented_for, pairwise

from unicodedata import normalize
import urllib.request
import matplotlib.pyplot as plt
import operator
import getpass
import pickle
import os
import sys

import time
from datetime import datetime
import signal
from contextlib import contextmanager

params = urllib.parse.quote_plus("DRIVER={ODBC Driver 17 for SQL Server};"
                                 "SERVER=sdh-die-bd;"
                                 "DATABASE=db_dwsiasg;"
                                 "Trusted_Connection=yes")

conn = sa.create_engine("mssql+pyodbc:///?odbc_connect={}".format(params))

# Padrão sugerido:  ANO_PESQUISADO = '2018', UNIDADE_PESQUISADA = '200141',
#                   NIVEIS_A_PESQUISAR = '3', MAX_N = 9, TAMANHO_MINIMO = 3
#                   TEMPO_LIMITE_SEGUNDOS = 300
# MAX_VERT = 8  # número máximo de vértices por ciclo
# MIN_VERT = 3  # tamanho mínimo do relacionamento, número de vértices no ciclo

ANO_PESQUISADO = '2018'
UNIDADE_PESQUISADA = '200141'
NIVEIS_A_PESQUISAR = 3
```

```
TEMPO_LIMITE_SEGUNDOS = 300
MAX_VERT = 6
MIN_VERT = 3
NIVEIS_A_PESQUISAR_TIMEOUT = NIVEIS_A_PESQUISAR - 1
ARQ_HIST = 'historico_das_pesquisas'

################################################

def ciclos_simples_limite(G, startnode, startedge, limit=7):
    """
    Versão inicial obtida de https://stackoverflow.com/questions/46590502/
                      how-to-modify-johnsons-elementary-cycles-algorithm-to-cap-maximum-cycle-length/46591473
                      "This is a highly modified version of this code, but at least it is working."

    Com alterações minhas, usando como referência a versão da biblioteca networkx em:
                      https://networkx.github.io/documentation/stable/_modules/networkx/algorithms/cycles.html
    """
    subG = type(G)(G.edges())
    sccs = [scc for scc in nx.strongly_connected_components(subG)
            if len(scc) > 1]
    ## sccs é uma lista com os conjuntos de vertices conectados.
    ## Nos casos em estudo um conjunto com todos os vértices.

    while sccs:

        scc = sccs.pop() # só deve ter uma lista aqui em sccs inicialmente

        if(startnode not in scc):
            break; # não deveria acontecer, mas como o loop retorna aqui é a condição de saída

        if(startedge[1] not in scc):
            break; # Teste de sanidade

        scc.remove(startnode)  # comeca com o IC, elimina necessidade de ordenação ao final
                               # e elimina computar ciclos que não interessam
        path = [startnode]
        blocked = set()
        blocked.add(startnode) # blocked guarda os vértices já excluídos da solução

        if(len(list(subG[startnode])) == 0):
            break; # se não há caminhos com startnode

        # começa já somente com a aresta de vencedor do item, isso evita computar os ciclos inversos
        stack = [(startnode, [startedge[1]])]
        #stack = [(startnode, list(subG[startnode]))]
        # [  (v_ini, [v2, v3, v4, ...] ) ]  o vértice e uma lista dos diretamente conectados

        while stack:
```

```
            thisnode, nbrs = stack[-1]

            if nbrs and len(path) <= limit:
                nextnode = nbrs.pop() # seleciona da lista dos conectados
                if nextnode == startnode:
                    yield path[:]
                elif nextnode not in blocked:
                    path.append(nextnode)
                    stack.append((nextnode, list(subG[nextnode]))) # Avança na "profundidade"
                    blocked.add(nextnode)
                    continue
            if not nbrs or len(path) > limit:
                blocked.remove(thisnode) # precisa nessa versão?
                stack.pop()
                path.pop()

##################################################

class TimeoutException(Exception): pass

@contextmanager
def time_limit(seconds):
    def signal_handler(signum, frame):
        raise TimeoutException("Foi atingido o tempo máximo sugerido pelo usuário!")
    signal.signal(signal.SIGALRM, signal_handler)
    signal.alarm(seconds)
    try:
        yield
    finally:
        signal.alarm(0)

##################################################

##################################################

def consulta_compras_ano_top(ano, uf=False, top=False):
    """
    """

    if(top):
        top_str = ' TOP ' + str(top)
    else:
        top_str = ''

    if(uf):
        uf_str = '''and un.ID_LCAL_UF_UNIDADE = \'''' + uf + '''\' '''
```

```
    else:
        uf_str = ''

    sql = '''SELECT DISTINCT ''' + top_str + '''
             un.ID_UNDD_UNIDADE

             FROM [db_dwsiasg].[dbo].[D_ITCP_ITEM_COMPRA] as ic

             INNER JOIN db_dwsiasg.dbo.F_ITEM_UNIDADE as iu
             ON ic.ID_ITCP_ITEM_COMPRA = iu.ID_ITCP_ITEM_COMPRA

             INNER JOIN db_dwsiasg.dbo.D_UNDD_UNIDADE as un
             ON iu.ID_UNDD_UNID_PARTICIPANTE = un.ID_UNDD_UNIDADE

             INNER JOIN db_dwsiasg.dbo.F_ITEM_COMPRA as ficomp
             ON ic.ID_ITCP_ITEM_COMPRA = ficomp.ID_ITCP_ITEM_COMPRA

             INNER JOIN db_dwsiasg.dbo.D_CMPR_COMPRA as comp
             ON ficomp.ID_CMPR_COMPRA = comp.ID_CMPR_COMPRA

             where year(comp.DT_REFERENCIA_COMPRA) = ''' + ano + '''
             and un.ID_UNDD_ESFERA = 3 and un.ID_UNDD_PODER = 1
             ''' + uf_str + '''

             order by 1
    '''

    dados = pd.read_sql(sql, conn)

    return(dados)

#######################################################

def consulta_unidade_ano_top(unidade, ano, top=False):
    """
    """

    if(top):
        top_str = ' TOP ' + str(top)
    else:
        top_str = ''

    sql = '''SELECT DISTINCT ''' + top_str + '''
             ic.[CH_ITCP_ITEM_COMPRA_EDIT]

             FROM [db_dwsiasg].[dbo].[D_ITCP_ITEM_COMPRA] as ic

             INNER JOIN db_dwsiasg.dbo.F_ITEM_UNIDADE as iu
```

```
            ON ic.ID_ITCP_ITEM_COMPRA = iu.ID_ITCP_ITEM_COMPRA

            INNER JOIN db_dwsiasg.dbo.D_UNDD_UNIDADE as un
            ON iu.ID_UNDD_UNID_PARTICIPANTE = un.ID_UNDD_UNIDADE

            INNER JOIN db_dwsiasg.dbo.F_ITEM_COMPRA as ficomp
            ON ic.ID_ITCP_ITEM_COMPRA = ficomp.ID_ITCP_ITEM_COMPRA

            INNER JOIN db_dwsiasg.dbo.D_CMPR_COMPRA as comp
            ON ficomp.ID_CMPR_COMPRA = comp.ID_CMPR_COMPRA

            where year(comp.DT_REFERENCIA_COMPRA) = ''' + ano + '''
            AND un.ID_UNDD_UNIDADE in ( ''' + unidade + ''' )

            order by 1
    '''

    dados = pd.read_sql(sql, conn)

    return(dados)

#####################################################

def consulta_item_niveis(item, niveis):
    """
    """

    sql = '''EXEC [work_claudioags].[dbo].[p_TCC_Consulta_tp_id_gr_niv_lim]
      @tipo = N'ITEMCOMPRA',
      @num_identificador = N\'''' + str(item) + '''\',
      @grupo = N'PADRAO',
      @niveis_a_pesquisar = ''' + str(niveis) + ''',
      @limite_arestas_para_abrir_vertices = 9999999;
    '''

    dados_item = pd.read_sql(sql, conn)

    return(dados_item)

#####################################################

#####################################################

def gera_subgrafo(vinculos_item_de_compra):
    """
    """
```

```
    dict_tipo_vert = {**vinculos_item_de_compra.set_index('vertice_complementar')['tp_vert_comp'].to_dict(),
                      **vinculos_item_de_compra.set_index('vertice_consultado')['tp_vert_cons'].to_dict()}

    subgrafo_do_item = nx.from_pandas_edgelist(vinculos_item_de_compra,
                                               'vertice_consultado', 'vertice_complementar', ["tipo_vinculo"], crea-
te_using=nx.MultiGraph)

    nx.set_node_attributes(subgrafo_do_item, dict_tipo_vert, 'tipo_vert')

    return(subgrafo_do_item)

#####################################################

def gera_subgrafo_direcionado(vinculos_item_de_compra):
    """
    """
    subgrafo_do_item = gera_subgrafo(vinculos_item_de_compra)

    return(subgrafo_do_item.to_directed())

#####################################################

def gera_lista_dos_ciclos_base(grf):
    """
    """
    return( list(nx.cycle_basis(grf)) )

#####################################################

def gera_lista_dos_ciclos(grf_direc, start_node, start_edge, n_max):
    """
    """
    return( [ x for x in ciclos_simples_limite(grf_direc, start_node, start_edge, n_max) if len(x) > 2] )

#####################################################

def gera_lista_dos_ciclos_networkx(grf_direc):
    """
    """
    return( [ x for x in nx.simple_cycles(grf_direc) if len(x) > 2] )

#####################################################

def gera_dict_qtd_ciclos_por_tamanho(lista_de_ciclos):
    """
    """
    tamanho_cada_ciclo = {}
    for i in range(0, len(lista_de_ciclos)):
```

```
        tamanho_cada_ciclo[i] = len(lista_de_ciclos[i])

    qtd_ciclos_por_tamanho = {}
    for i in set(list(tamanho_cada_ciclo.values())):
        qtd_ciclos_por_tamanho[i] = list(tamanho_cada_ciclo.values()).count(i)

    return(qtd_ciclos_por_tamanho)

################################################

def extrai_ciclos_contendo_item_de_compra(lista_de_ciclos, grf):
    """
    """
    lista_de_ciclos_it_cp = []

    for i in range(0, len(lista_de_ciclos)):
        ok = False
        for e in lista_de_ciclos[i]:
            if grf.nodes[ e ]['tipo_vert'] == 'ITEMCOMPRA':
                ok = True
        if (ok):
            lista_de_ciclos_it_cp.append(lista_de_ciclos[i])

    return(lista_de_ciclos_it_cp)

################################################

def elimina_duplicidade(lista_de_ciclos, minim_vert=MIN_VERT):
    """
    """
    def isCircular(arr1, arr2):
        if len(arr1) != len(arr2):
            return False
        str1 = ' '.join(map(str, arr1))
        str2 = ' '.join(map(str, arr2))
        str3 = ' '.join(map(str, arr1[::-1]))
        if len(str1) != len(str2):
            return False
        return ((str1 in str2 + ' ' + str2) or (str3 in str2 + ' ' + str2))

    lista_de_ciclos_sem_dup = []

    for i in range( 0, len(lista_de_ciclos) ):
        tam = len(lista_de_ciclos[i])
        ok = True
        if tam >= minim_vert:
```

```
            for l in lista_de_ciclos_sem_dup:
                if isCircular(l, lista_de_ciclos[i]):
                    ok = False
                    break
            if(ok):
                lista_de_ciclos_sem_dup.append(lista_de_ciclos[i])

    return(lista_de_ciclos_sem_dup)

##################################################

def ordena_ciclo_pelo_item_de_compra(lista_de_ciclos, grf):
    """
    """
    lista_de_ciclos_ord = []

    for i in range(0, len(lista_de_ciclos)):
        for pos, elem in enumerate(lista_de_ciclos[i]):
            if grf.nodes[ elem ]['tipo_vert'] == 'ITEMCOMPRA':
                lista_de_ciclos_ord.append(lista_de_ciclos[i][pos:]+lista_de_ciclos[i][:pos])
                break

    return(lista_de_ciclos_ord)

##################################################

def ordena_ciclo_pelo_tamanho(lista_de_ciclos):
    """
    """
    lista_de_ciclos_ord = []

    if(lista_de_ciclos):
        maior = len(max(lista_de_ciclos,key=len))
    else:
        maior = 0

    for j in range(0, maior+1):

        for i in range(0, len(lista_de_ciclos)):

            if len(lista_de_ciclos[i]) == j:
                    lista_de_ciclos_ord.append(lista_de_ciclos[i])

    return(lista_de_ciclos_ord)

##################################################

def expoe_resultados(lista_de_ciclos, grf, minim_vert=MIN_VERT):
```

```
    """
    Para imprimir os ciclos em formato simples de texto durante a execução
    """
    dic_qtd_cicls_por_tam = gera_dict_qtd_ciclos_por_tamanho(lista_de_ciclos)

    if dic_qtd_cicls_por_tam:
        maior_dic = max(dic_qtd_cicls_por_tam, key=int)
    else:
        maior_dic = 0

    ciclos_tam = [[] for i in range(  maior_dic+1   )]

    for i in range( 0, len(lista_de_ciclos) ):

        taman = len(lista_de_ciclos[i])

        if taman >= minim_vert:
            ciclos_tam[taman].append(lista_de_ciclos[i])

    for tam in sorted(dic_qtd_cicls_por_tam):

        print('############', tam, 'níveis de relacionamentos,', dic_qtd_cicls_por_tam[tam], 'caso(s):',
'############','\n', )

        for count, c in enumerate(ciclos_tam[tam]):
            if (count>=10):
                print('Imprimindo somente os', count, 'primeiros.')
                break
            print(c, '\n')
            for i in range(tam-1):
                print( '',
                       '[',grf.nodes[ c[i] ]['tipo_vert'], '] <--', grf[ c[i] ][ c[i+1] ]['tipo_vinculo'], '-->',
                       '[',grf.nodes[ c[i+1] ]['tipo_vert'], ']',)  # multigrafo tem mais de um valor em grf[ c[i] ][
c[i+1] ]['tipo_vinculo']
            print( '',
                   '[',grf.nodes[ c[tam-1] ]['tipo_vert'], '] <--', grf[ c[tam-1] ][ c[0] ]['tipo_vinculo'], '-->',
                   '[',grf.nodes[ c[0] ]['tipo_vert'], ']\n'
                 )

    return(True)

#####################################################

#####################################################

def pesquisa_em_um_item(numero_item_compra, niv=NIVEIS_A_PESQUISAR,
```

```
                    t_lim=TEMPO_LIMITE_SEGUNDOS,
                    ciclo_max_n=MAX_VERT, ciclo_min_n=MIN_VERT,
                    arquivo=ARQ_HIST,
                    historico=[[]], netx=False, todos_vert=False):
    """
    Layout do arquivo histórico (somente pesquisas completadas):
    [ITCP, niveis, qtd_vinc, qtd_elem_sbgrf, qtd_elem_base, t_lim_segundos, ciclo_max_n, ciclo_min_n, ho-
ra_da_pesquisa]
    """
####------------------------------------------------------------------------
    ''' Consulta ao banco de dados
    '''
    vinculos_item_de_compra = consulta_item_niveis(numero_item_compra, niv)
    qtd_vinc = len(vinculos_item_de_compra)
    print("Quantidade de vínculos: ", qtd_vinc)

####------------------------------------------------------------------------
    ''' Gera um multigrafo, pois entre os vértices há mais de uma aresta.
        Encontra o vértice do item de compra e a respectiva aresta de
        vencedor da licitação.
        Caso o relacionamento venha invertido, como às vezes ocorre no Ygg embora
        não no caso dos itens de compra, há previsão de "correção" da direção da aresta
        (já fica pronto para outros casos de uso)
    '''
    multigrafo = gera_subgrafo(vinculos_item_de_compra)

    start_node = ''
    for nd in list(multigrafo.nodes):
        if (multigrafo.nodes[ nd ]['tipo_vert'] == 'ITEMCOMPRA'):
            start_node = nd
            break

    edge_labels = nx.get_edge_attributes(multigrafo, 'tipo_vinculo')

    aresta_ini = ''
    for elemento in edge_labels:
        if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
            aresta_ini = elemento
            break

    if( aresta_ini == ''):
        print('Fim da pesquisa do item', numero_item_compra, '- NENHUM VENCEDOR DO ITEM NA LISTA ----')
        return(True)

    aresta_inicial = ''
    A, B, C = aresta_ini    ### A e B utilizados também no teste do ciclo base
    if( A==start_node):
        aresta_inicial = (A, B)
```

```
    else:
        A, B = B, A
        aresta_inicial = (A, B)

    subgrf = nx.Graph(multigrafo)
##  subgrf.remove_edges_from(subgrf.selfloop_edges()) ## não existem no Ygg

####--------------------------------------------------------------------------
    ''' Verificação de sanidade: a aresta inicial tem que estar no grafo gerado
        a partir do multigrafo, caso tenha sido descartada é "atualizada".
        Há uma segunda verificação, para debug do código.
    '''
##  garantir que a aresta vencedora não saiu na conversão do multigrafo:

    subgrf_edge_labels = nx.get_edge_attributes(subgrf, 'tipo_vinculo')
    elemento = ''
    for elemento in subgrf_edge_labels:
        if elemento == aresta_inicial:
            if('item de licitação vencido por - DB_DWSIASG'==subgrf_edge_labels[elemento]):
                print('OK aresta_inicial no subgrafo')
                break
            else:
                print('atualizando vínculo da aresta inicial')
                subgrf.add_edge(A, B, tipo_vinculo='item de licitação vencido por - DB_DWSIASG')
                break

    elemento = ''
    for elemento in subgrf_edge_labels:
        if elemento == aresta_inicial:
            if('item de licitação vencido por - DB_DWSIASG'!=subgrf_edge_labels[elemento]):
                print('ERRO: segunda aresta_inicial no subgrafo')
                break

####--------------------------------------------------------------------------
    ''' Contagem dos vértices para comparação de desempenho.
        Verificação de sanidade: o grafo nunca deveria estar vazio.
    '''

    lista_vert = list(subgrf.nodes)
    conjunto_elementos_subgrf = set(lista_vert) ############################
    qtd_elem_sbgrf = len(conjunto_elementos_subgrf)
    print("Quantidade de vértices: ", qtd_elem_sbgrf)

    if(qtd_elem_sbgrf==0):
        print('Fim da pesquisa do item', numero_item_compra, '- ERRO: GRAFO SEM VÉRTICES   ----------')
        return(True)
```

```
####-------------------------------------------------------------------------
    ''' Gera a lista de vértices que compõem a base dos ciclos possíveis no grafo
        Essa lista é gerada rapidamente e reduz o escopo de solução a computar
    '''
    lista_ciclos_base = gera_lista_dos_ciclos_base(subgrf)
    conjunto_elementos_base = { x  for elem in lista_ciclos_base  for x in elem }
    qtd_elem_base = len(conjunto_elementos_base)
    print("Quantidade de vértices parte da solução: ", qtd_elem_base)

    if(qtd_elem_base==0):
        print('Fim da pesquisa do item', numero_item_compra, '- NENHUM CICLO BASE DETECTADO ---------')
        return(True)


    ## A e B definidos acima como os vértices da aresta de interesse (inicial)
    if( A not in conjunto_elementos_base):
        print('Fim da pesquisa do item', numero_item_compra, '- NENHUM CICLO INCLUI O ITEM DE COMPRA!')
        return(True)

    if( B not in conjunto_elementos_base):
        print('Fim da pesquisa do item', numero_item_compra, '- NENHUM CICLO INCLUI O VENCEDOR! -----')
        return(True)

####-------------------------------------------------------------------------
    ''' Verifica se na mesma unidade, nesta execução ou no arquivo histórico salvo utilizado,
        já houve pesquisa de item com mesmas características.
        Reduzido de numero_item_compra[:17] para numero_item_compra[:6] pois há unidades que
        ao invés de fazer vários itens na mesma compra fazem várias compras sequenciais
        onde entram os mesmos concorrentes
    '''
    ja_pesquisado=False
    for elem in historico:

        if((numero_item_compra[:6]==elem[0][:6]) and
           (niv==int(elem[1])) and
           (qtd_vinc==int(elem[2])) and
           (qtd_elem_sbgrf==int(elem[3])) and
           (qtd_elem_base==int(elem[4])) and
           (t_lim<=int(elem[5])) and
           (ciclo_max_n<=int(elem[6])) and
           (ciclo_min_n>=int(elem[7])) ):

            print("Item ", numero_item_compra , 'similar ao ', elem[0], ': \n'
                  'já pesquisado em ', elem[8], ' usando', elem[1], 'níveis,', elem[5],
                  'segundos de limite e retornando ciclos de',elem[7], 'a', elem[6], 'vértices.')
            print('----------  Fim do item', numero_item_compra, ' - HÁ SIMILAR  ----------------------')
            ja_pesquisado=True
```

```
            break

    if(ja_pesquisado):
        return(False)

####--------------------------------------------------------------------------
    ''' Salva o histórico em arquivo, também para permitir eventual
        recomeço do ponto onde interrompeu
    '''
    now = datetime.now()
    historico.append([numero_item_compra, niv, qtd_vinc, qtd_elem_sbgrf, qtd_elem_base,
                      t_lim, ciclo_max_n, ciclo_min_n, now.strftime("%Y-%m-%d %H:%M")])
    with open(arquivo,'wb') as wfile:
        pickle.dump(historico, wfile)


####--------------------------------------------------------------------------
    ''' Todas as verificações satisfeitas, inicia a procura dos ciclos.
        Três otimizações são muito significativas: o ciclo tem limitação de tamanho máximo;
        é procurado somente entre os vértices retornados na rotina de ciclo base; e,
        sempre iniciando pela aresta de interesse (Item de compra -> vencido por -> Vencedor)
        Observa-se que a implementação pressupõe um grafo direcionado e como temos um
        multigrafo não-direcionado há que se fazer as conversões antes da aplicação e
        resgatar as informações das arestas descartadas caso haja interesse posteriormente
    '''
    subgrf_direc = subgrf.to_directed()

    if(todos_vert):
        pass
    else:
        vertices_a_remover = conjunto_elementos_subgrf - conjunto_elementos_base
        subgrf_direc.remove_nodes_from(vertices_a_remover)  # [n for n in G if n not in set(nodes)]


    ## Aqui são procurados os ciclos ##
    lista_dos_ciclos = gera_lista_dos_ciclos(subgrf_direc, start_node, aresta_inicial, ciclo_max_n)
    print("Ciclos detectados com as regras definidas: ", gera_dict_qtd_ciclos_por_tamanho(lista_dos_ciclos))

    if(len(lista_dos_ciclos)==0):
        print('Fim da pesquisa do item', numero_item_compra, '- NENHUM CICLO A PESQUISAR ------------')
        return(True)

####--------------------------------------------------------------------------
    ''' Ordena o dicionário dos ciclos pelo tamanho, somente para facilitar a apresentação.
        Remove os vértices do grafo direcionado, somente para facilitar a apresentação.
        Remove os vértices do multigrafo não-direcionado do item, que será armazenado em arquivo.
```

```
    '''
    lista_dos_ciclos_it_cp_sem_dup_ordenada = ordena_ciclo_pelo_tamanho(lista_dos_ciclos)

    subgrf.remove_nodes_from([n for n in subgrf if n not in
                              { x for elem in lista_dos_ciclos_it_cp_sem_dup_ordenada for x in elem }])

    multigrafo.remove_nodes_from([n for n in multigrafo if n not in
                                  { x for elem in lista_dos_ciclos_it_cp_sem_dup_ordenada for x in elem }])

    arq_grph = '_'.join( ('grph', numero_item_compra, str(niv), str(t_lim), str(ciclo_max_n), str(ciclo_min_n),
'.nx') )
    with open(arq_grph,'wb') as wfile:
        pickle.dump(multigrafo, wfile)

####------------------------------------------------------------------------
    ''' Apresenta resultados em formato de texto
    '''
    if( expoe_resultados(lista_dos_ciclos_it_cp_sem_dup_ordenada, subgrf_direc, ciclo_min_n) ):
        print('Fim da pesquisa do item', numero_item_compra, '-------------------------------------')

####------------------------------------------------------------------------
    ''' Aqui termina a função de pesquisa em um item
    '''
    return(True)

#################################################

def pesquisa_unidade(numero_unidade, ano, niv_p=NIVEIS_A_PESQUISAR,
                     t_lim=TEMPO_LIMITE_SEGUNDOS,
                     ciclo_max_n=MAX_VERT, ciclo_min_n=MIN_VERT,
                     arquivo=ARQ_HIST, top=False, nx=False, todos_vert=False):
    """
    Layout do arquivo histórico (somente pesquisas completadas):
    [ITCP, niveis, qtd_vinc, qtd_elem_sbgrf, qtd_elem_base, t_lim_segundos, ciclo_max_n, ciclo_min_n, ho-
ra_da_pesquisa]
    """
####------------------------------------------------------------------------
    ''' Consulta ao banco de dados, indicando a unidade a consultar e o ano.

    '''
    inicio_unidade = datetime.now()

    if(top):
        itens_de_compra = consulta_unidade_ano_top(numero_unidade, ano, top)
    else:
        itens_de_compra = consulta_unidade_ano_top(numero_unidade, ano)

    qtd_ic = len(itens_de_compra)
```

```
    print('Unidade:', numero_unidade, 'Ano:', ano, 'Qtd:', qtd_ic, 'itens de compra.')
    print('Pesquisa em', niv_p, 'níveis, com limite de', t_lim, 'segundos cada item.')
    print('Buscando ciclos de', ciclo_min_n, 'a', ciclo_max_n, 'vértices.')
    print(inicio_unidade.strftime("%Y-%m-%d %H:%M"))

####------------------------------------------------------------------------
    ''' Inicializa o arquivo de histórico, se não existir.
    '''
    if os.path.exists(arquivo):
        with open(arquivo,'rb') as rfile:
            historico = pickle.load(rfile)
    else:
        historico = [['00000000000000000000', '0', '0', '0', '0', '0', '0', '0', inicio_unidade.strftime("%Y-%m-%d
%H:%M")]]

        with open(arquivo,'wb') as wfile:
            pickle.dump(historico, wfile)

####------------------------------------------------------------------------
    ''' Inicializa a pesquisa de cada item de compra da unidade consultada
    '''
    niv_timeout = niv_p-1 # em caso de timeout, refazer a pesquisa no banco de dados com menos um nível

    for i, item in enumerate(itens_de_compra.values):

        print('\n\n\n')
        print('Item', i+1, 'de', qtd_ic)

        ja_pesquisado=False
        inicio = datetime.now()

        try:
            ''' Há uma definição de limite máximo de tempo, deixada para a chamada da função pelo usuário.
                Mesmo com todas as otimizações implementadas, é prudente manter uma limitação para evitar
                o travamento ao longo de pesquisas muito extensas.
            '''
            with time_limit(t_lim):

                for elem in historico:

                    ''' Verifica se este item já foi pesquisado, com restrições equivalentes.
                        Permite a retomada ao interromper o processo de pesquisa.
                    '''

                    if((item[0]==elem[0]) and
                       (niv_p==int(elem[1])) and
```

```
                (t_lim<=int(elem[5])) and
                (ciclo_max_n<=int(elem[6])) and
                (ciclo_min_n>=int(elem[7])) ):

                 print('Item', item[0], 'já pesquisado em', niv_p,
                       'níveis, com tempo limite', str(elem[5]), ', no dia', str(elem[8]) )
                 print('##########  Fim do item', item[0], ' - JÁ PESQUISADO - ####################')
                 print()
                 ja_pesquisado=True
                 break

        if( not(ja_pesquisado) ):

            ''' Aqui é iniciada a pesquisa do item, inclusive consulta ao banco e ciclos:
            '''

            print('Item', item[0], 'em', niv_p, 'níveis, com tempo limite de', t_lim, 'segundos.')

            if( pesquisa_em_um_item(item[0], niv_p, t_lim,
                                    ciclo_max_n, ciclo_min_n,
                                    arquivo,
                                    historico, nx, todos_vert ) ):

                print('Tempo do item:', datetime.now()-inicio )
                print('##########  Fim do item', item[0], ' sem restrição. ########################')

            else:
                print('Tempo do item:', datetime.now()-inicio )
                print('##########  Fim do item', item[0], ' sem restrição - SIMILAR ENCONTRADO ###')

except TimeoutException as e:

    print('Item', item[0], 'EXCEDEU o tempo limite de', t_lim, 'segundos.')
    historico.append([item[0], niv_p, '0', '0', '0', t_lim,
                      ciclo_max_n, ciclo_min_n, inicio.strftime("%Y-%m-%d %H:%M")])

    with open(arquivo,'wb') as wfile:
        pickle.dump(historico, wfile)

    ja_pesquisado=False
    for elem in historico:

        ''' Verifica se este item já foi pesquisado, com restrições equivalentes.
        '''
        if((item[0]==elem[0]) and
           (niv_timeout==int(elem[1])) and
           (t_lim<=int(elem[5])) and
           (ciclo_max_n<=int(elem[6])) and
```

```
                (ciclo_min_n>=int(elem[7])) ):

                    print('Item', item[0], 'já pesquisado em', niv_timeout,
                              'níveis, com tempo limite', str(elem[5]), ', no dia', str(elem[8]))
                    print('##########  Fim do item', item[0], ' COM RESTRIÇÂO - JÁ PESQUISADO. #######')
                    print()
                    ja_pesquisado=True
                    break

            if( not(ja_pesquisado) ):

                print()
                print('Reiniciando com restrição:')
                print('Item', item[0], 'em', niv_timeout, 'níveis, com tempo limite de', t_lim, 'segundos.')

                ''' Aqui é iniciada a pesquisa do item, inclusive consulta ao banco e ciclos, com um nível a
                    menos no banco de dados uma vez que a consulta "inicial" tenha se provado muito complexa
                    e excedido o limite estipulado pelo usuário:
                '''
                if( pesquisa_em_um_item(item[0], niv_timeout, t_lim,
                                        ciclo_max_n, ciclo_min_n,
                                        arquivo,
                                        historico, nx, todos_vert ) ):

                    print('Tempo do item:', datetime.now()-inicio )
                    print('##########  Fim do item', item[0], ' COM RESTRIÇÂO. ######################')

                else:
                    print('Tempo do item:', datetime.now()-inicio )
                    print('##########  Fim do item', item[0], ' COM RESTRIÇÂO - SIMILAR ENCONTRADO ###')


    print('Tempo para processar os itens da unidade', numero_unidade, ':', datetime.now()-inicio_unidade )
    print()

#####################################################

def main(args):

    if (len(args)>=1):
        arg=args[0]
    else:
        arg='AC'

    inicio = datetime.now()
```

```
    unid_exec_fed = consulta_compras_ano_top('2018', arg)

    print(unid_exec_fed.shape)
    print(unid_exec_fed.head())

    niv=3
    t_lim=300
    ciclo_max_n=6
    for i, unid in enumerate(unid_exec_fed.values):
        print('Unidade', i+1, ' - ', unid[0] )
        arq = '_'.join( (str(unid[0]), str(ANO_PESQUISADO), str(niv), str(t_lim), str(ciclo_max_n), '.hist') )
        pesquisa_unidade(str(unid[0]), ANO_PESQUISADO,
                    niv_p=niv, t_lim=t_lim, ciclo_max_n=ciclo_max_n,
                    arquivo=arq)

    print('Tempo para processar', arg, ':', datetime.now()-inicio )

if __name__ == "__main__":
    main(sys.argv[1:])
```

B.2 – Impressão dos ciclos

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
import pickle
import os

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))
##################################################

def plot_cycle(grph, pt_central, nm_atrib_node='tipo_vert', nm_atrib_edge='tipo_vinculo', x=30, y=30, fnt1_sz=16,
fnt2_sz=8, afast_ini=1, offset=0.05, save_name=None):

    node_list = grph.nodes()

    listsize = len(node_list)
    k= 1/np.sqrt(listsize+1)

    #pos = nx.spring_layout(grph, k=k*afast_ini)
    pos=nx.fruchterman_reingold_layout(grph, k=k*afast_ini)
    #pos=nx.spectral_layout(grph, scale=afast_ini)

    plt.figure(1,figsize=(x,y))
    plt.axis('off')

    nx.draw_networkx_nodes(node_list, pos,
                           node_color='b',
                           node_size=500,
                           alpha=0.8) #, dpi=1000

    nx.draw_networkx_nodes(pt_central, pos,
                           node_color='r',
                           node_size=500,
                           alpha=0.8)

    nx.draw_networkx_edges(grph, pos, width=1.0, alpha=0.5)

    edge_labels = nx.get_edge_attributes(grph, nm_atrib_edge)

    # as arestas de multigrafos retornam três valores na chave, a função draw precisa de dois
    new_edge_labels = {}
    for key, value in edge_labels.items():
        a, b, c = key
        tipo = value
        new_edge_labels.update( {(a,b):tipo} )
```

```
    nx.draw_networkx_edge_labels(grph, pos, new_edge_labels, font_size=fnt2_sz)

    labels = nx.get_node_attributes(grph, nm_atrib_node)
    for p in pos:  # raise text positions
        pos[p][1] += offset
    nx.draw_networkx_labels(grph, pos, labels, font_size=fnt1_sz)

    if save_name is not None:
        plt.savefig(save_name)
    else:
        plt.show()

#####################################################
def desenha_ciclos_vencedores(path='./', afast_ini=1, save_name=None):

    for entry in os.scandir(path):

        if entry.is_file() and entry.name[-2:]=='nx':

            ok=False

            print(entry.name)

            with open(os.path.join(path,entry.name),'rb') as rfile:
                grafo = pickle.load(rfile)

            # para garantir que desenha apenas ciclos "com vencedor"
            edge_labels = nx.get_edge_attributes(grafo, 'tipo_vinculo')
            for elemento in edge_labels:
                if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
                    ok=True
                    break

            if(not(ok)):
                # não vai desenhar
                continue

            ic = [x for x,y in grafo.nodes(data=True) if y['tipo_vert']=='ITEMCOMPRA']

            plot_cycle(grafo, ic, x=16, y=16, fnt1_sz=10, fnt2_sz=8, afast_ini=afast_ini, offset=0.03,
save_name=save_name)

#####################################################
desenha_ciclos_vencedores()
```

B.3 - Análise dos ciclos

```
# coding: utf-8
import sqlalchemy as sa
import urllib
import pandas as pd
import networkx as nx
import itertools
import numpy as np
import os
import pickle
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

import sys
import time
from datetime import datetime

params = urllib.parse.quote_plus("DRIVER={ODBC Driver 17 for SQL Server};"
                                 "SERVER=sdh-die-bd;"
                                 "DATABASE=db_dwsiasg;"
                                 "Trusted_Connection=yes")

conn = sa.create_engine("mssql+pyodbc:///?odbc_connect={}".format(params))

################################################

def conecta():

    conn = pymssql.connect(
        host=r'sdh-die-bd',
        user=r'CGU\claudioags',
        password=password,
        database='work_claudioags'
    )

    return conn

################################################

def consulta_unid_itcp_forn_top(ids, top=False):
    """
    """

    if(top):
```

```
    top_str = ' TOP ' + str(top)
else:
    top_str = ''


sql = '''SELECT DISTINCT ''' + top_str + '''
            un.ID_UNDD_UNIDADE  id_unid,
            ygv_itcp.id_vertice id_itcp,
            ygv_frn.id_vertice  id_frn

        from Yggdrasil.dbo.yggdrasil_vertice ygv

        INNER JOIN [db_dwsiasg].[dbo].[D_ITCP_ITEM_COMPRA] as ic
        on ic.CH_ITCP_ITEM_COMPRA_EDIT = ygv.identificador

        INNER JOIN db_dwsiasg.dbo.F_ITEM_FORNECEDOR ifo1
        ON ifo1.ID_ITCP_ITEM_COMPRA = ic.ID_ITCP_ITEM_COMPRA

        INNER JOIN db_dwsiasg.dbo.F_ITEM_FORNECEDOR ifo2
        ON ifo2.ID_FRND_FORNECEDOR_COMPRA = ifo1.ID_FRND_FORNECEDOR_COMPRA

        INNER JOIN db_dwsiasg.dbo.F_ITEM_FORNECEDOR ifo3
        ON ifo2.ID_ITCP_ITEM_COMPRA = ifo3.ID_ITCP_ITEM_COMPRA

        INNER JOIN db_dwsiasg.dbo.D_CMPR_COMPRA as comp2
        ON ifo2.ID_CMPR_COMPRA = comp2.ID_CMPR_COMPRA

        INNER JOIN db_dwsiasg.dbo.D_UNDD_UNIDADE as un
        ON comp2.ID_UNDD_RESP_COMPRA = un.ID_UNDD_UNIDADE

        INNER JOIN [db_dwsiasg].[dbo].[D_ITCP_ITEM_COMPRA] as ic2
        on ifo2.ID_ITCP_ITEM_COMPRA = ic2.ID_ITCP_ITEM_COMPRA

        INNER JOIN Yggdrasil.dbo.yggdrasil_vertice ygv_frn
        on ygv_frn.identificador = ifo3.ID_FRND_FORNECEDOR_COMPRA

        INNER JOIN Yggdrasil.dbo.yggdrasil_vertice ygv_itcp
        on ygv_itcp.identificador = ic2.CH_ITCP_ITEM_COMPRA_EDIT

            where ygv.id_vertice IN ({})
            and  year(comp2.DT_REFERENCIA_COMPRA) >= '2017'
            and un.ID_UNDD_ESFERA = 3 and un.ID_UNDD_PODER = 1

'''

dados = pd.read_sql(sql.format(str(ids).strip('[]')), conn)

return(dados)
```

```
################################################

def consulta_unid_org_itcp_forn_ano_uf_top(ano, uf=False, forn=False, top=False):
    """
    """

    if(forn):
        forn_str = '''and ifo.ID_FRND_FORNECEDOR_COMPRA IN (''' + forn + ''') '''
    else:
        forn_str = ''



    if(top):
        top_str = ' TOP ' + str(top)
    else:
        top_str = ''

    if(uf):
        uf_str = '''and un.ID_LCAL_UF_UNIDADE = \'''' + uf + '''\' '''
    else:
        uf_str = ''

    sql = '''SELECT DISTINCT ''' + top_str + '''
              un.ID_UNDD_UNIDADE id_unid,
              un.ID_UNDD_ORGAO id_org,
              ic.CH_ITCP_ITEM_COMPRA_EDIT id_it_cp,
              ifo.ID_FRND_FORNECEDOR_COMPRA id_frn


             FROM [db_dwsiasg].[dbo].[D_ITCP_ITEM_COMPRA] as ic

             INNER JOIN db_dwsiasg.dbo.F_ITEM_FORNECEDOR ifo
             ON ifo.ID_ITCP_ITEM_COMPRA = ic.ID_ITCP_ITEM_COMPRA

             INNER JOIN db_dwsiasg.dbo.D_CMPR_COMPRA as comp
             ON ifo.ID_CMPR_COMPRA = comp.ID_CMPR_COMPRA

             INNER JOIN db_dwsiasg.dbo.D_UNDD_UNIDADE as un
             ON comp.ID_UNDD_RESP_COMPRA = un.ID_UNDD_UNIDADE

             where year(comp.DT_REFERENCIA_COMPRA) = ''' + ano + '''
             and un.ID_UNDD_ESFERA = 3 and un.ID_UNDD_PODER = 1
             ''' + uf_str + '''
              ''' + forn_str + '''
```

```
            order by 1
    '''

    dados = pd.read_sql(sql, conn)

    return(dados)

#####################################################

def busca_cnpjs_no_ygg(ids):

    sql = '''
    SELECT identificador
      FROM [Yggdrasil].[dbo].[yggdrasil_vertice] as v
      WHERE id_vertice IN ({})
    '''

    ids_ = pd.read_sql(sql.format(str(ids).strip('[]')), conn)

    return(ids_)

#####################################################

def busca_ids_no_ygg(ids):

    sql = '''
    SELECT id_vertice
      FROM [Yggdrasil].[dbo].[yggdrasil_vertice] as v
      WHERE identificador IN ('{}')
    '''

    ids_ = pd.read_sql(sql.format(str(ids).strip('[]')), conn)

    return(ids_)

#####################################################

def consulta_caminho(id1, tp1, id2, tp2, niv, lim='999'):
    """
    """

    sql = '''EXEC [Yggdrasil].[dbo].[p_Caminho_tp1_id1_tp2_id2_gr_niv_lim]
        @tipo1 = N\'''' + tp1 + '''\',
        @num_identificador1 = N\'''' + id1 + '''\',
        @tipo2 = N\'''' + tp2 + '''\',
        @num_identificador2 = N\'''' + id2 + '''\',
        @grupo = N'PADRAO',
        @niveis_a_pesquisar = ''' + niv + ''',
```

```
        @limite_arestas_para_abrir_vertices = ''' + lim + '''
    '''

    dados_item = pd.read_sql(sql, conn)

    return(dados_item)

##################################################


def mesmo_elemento(a, b):

    cj_a = set((map(str, a)))
    cj_b = set((map(str, b)))

    if (cj_a & cj_b):
        return True
    else:
        return False


##################################################
##################################################

def busca_itcp_frn_nos_ciclos(path='./'):

    cj_ids = {}
    itcp = ''

    for entry in os.scandir(path):

        if entry.is_file() and entry.name[-2:]=='nx':

            with open(os.path.join(path,entry.name),'rb') as rfile:
                grafo = pickle.load(rfile)

            # para garantir que abre apenas ciclos "com vencedor"
            ok=False
            edge_labels = nx.get_edge_attributes(grafo, 'tipo_vinculo')
            for elemento in edge_labels:
                if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
                    ok=True
            if(not(ok)):
                continue

            for elemento in edge_labels:
                if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
```

```
                    a, b, c = elemento
                    if(grafo.node[a]['tipo_vert']=='ITEMCOMPRA'):
                        itcp = a
                        cj_ids[itcp] = ([], [])
                    break

            for elemento in edge_labels:
                if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
                    a, b, c = elemento
                    if(grafo.node[b]['tipo_vert']=='CNPJ'):
                        cj_ids[itcp][0].append(b)
                if('item de compra teve participante - DB_DWSIASG'==edge_labels[elemento]):
                    a, b, c = elemento
                    if(grafo.node[b]['tipo_vert']=='CNPJ'):
                        cj_ids[itcp][1].append(b)


    return(cj_ids)

################################################

def analisa_grafo(grafo):

    dict_itcp = {}

    # para garantir que abre apenas ciclos "com vencedor"
    ok=False
    edge_labels = nx.get_edge_attributes(grafo, 'tipo_vinculo')
    for elemento in edge_labels:
        if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
            ok=True
    if(not(ok)):
        return(dict_itcp)

    for elemento in edge_labels:
        if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
            a, b, c = elemento
            if(grafo.node[a]['tipo_vert']=='ITEMCOMPRA'):
                itcp = a
                dict_itcp[itcp] = {}
                dict_itcp[itcp]['vencedor'] = []
                dict_itcp[itcp]['participante'] = []
                dict_itcp[itcp]['periodo_analisado'] = '2017-2019'
                dict_itcp[itcp]['qtd_itens_partic_juntos'] = 0
                dict_itcp[itcp]['qtd_itens_levant_inicial'] = 0
                dict_itcp[itcp]['qtd_itens_maiores_partic'] = ''
                dict_itcp[itcp]['qtd_itens_apos_limpeza'] = 0
                dict_itcp[itcp]['metricas_partic_e'] = ''
```

```
                dict_itcp[itcp]['metricas_partic_ou'] = ''
                dict_itcp[itcp]['metricas_grupo'] = ''
            break

    for elemento in edge_labels:
        if('item de licitação vencido por - DB_DWSIASG'==edge_labels[elemento]):
            a, b, c = elemento
            if(grafo.node[b]['tipo_vert']=='CNPJ'):
                dict_itcp[itcp]['vencedor'].append(b)
        if('item de compra teve participante - DB_DWSIASG'==edge_labels[elemento]):
            a, b, c = elemento
            if(grafo.node[b]['tipo_vert']=='CNPJ'):
                dict_itcp[itcp]['participante'].append(b)

    partic = dict_itcp[itcp]['participante']


####------------------------------------------------------------------------
    ''' Consulta bancos de dados
    '''
    grupo_frns = consulta_unid_itcp_forn_top( itcp )

    print (grupo_frns)

    qtd_frn = grupo_frns['id_frn'].unique().shape[0]
    qtd_it_cp = grupo_frns['id_itcp'].unique().shape[0]

    if( qtd_frn*qtd_it_cp > 2147000000 ):
        print('Dataframe muito grande para esta versão do pandas. Aguarde correção ou retorne à versão 0.21')
        return({})

####------------------------------------------------------------------------
    ''' Contagem inicial das licitações em que os alvos participam
    '''

    dados_grp = (grupo_frns[ grupo_frns['id_frn'].isin(partic) ].groupby(['id_itcp', 'id_frn'])
                 .agg({'id_unid': [np.size]})
                 .rename(columns={'id_unid': 'Qtd'})).dropna()


    dados_grp.columns = [tup[1] for tup in dados_grp.columns.values]

    df1 = dados_grp.copy()
    df1 = df1.reset_index()

    itcp_cjto = pd.merge(df1, df1, on='id_itcp', how='inner')
```

```
    dict_itcp[itcp]['qtd_itens_partic_juntos'] = (itcp_cjto.loc[itcp_cjto.duplicated(subset='id_itcp',
                                                                                    keep=False), :]['id_itcp']
                                                  .unique().shape[0])

    print('qtd_itens_partic_juntos:',dict_itcp[itcp]['qtd_itens_partic_juntos'])

    if ( dict_itcp[itcp]['qtd_itens_partic_juntos'] == 0 ):
        print ('ALERTA: este é um dos casos em que o histórico do SIASG não mantém consistência!')
        return({})

####-------------------------------------------------------------------------
    ''' Remove Itens com só um fornecedor e fornecedores que só participam em um item:
        Elimina itens em que só existe um frn (id_itcp não aparece em mais de uma linha),
        após essa eliminação, o frn pode ter participado em apenas um outro itcp além
        daquele e portanto agora só aparece em um itcp (id_frn não aparece em mais de uma linha).
        A se eliminar os frn que só participam em um itcp somente aquele itcp é afetado, mas
        pode acontecer de naquele itcp somente haver mais um frn, o que torna esse último frn o
        único do itcp (situação inicial), por isso a necessidade de fazer o loop até que todos
        os casos estejam removidos.
        Todas a regras removidas com esse procediemento são de baixo suporte e não são de interesse
        para o trabalho, só mascaram (muito pouco) os demais resultados.
    '''
    dados_mais_de_um_frn = grupo_frns.copy()

    while True:

        tamanho = dados_mais_de_um_frn.shape[0]

        # remove itcp com apenas um frn -> filtra apenas os "duplicados"
        dados_mais_de_um_frn = dados_mais_de_um_frn.loc[dados_mais_de_um_frn.duplicated(subset='id_itcp',
keep=False), :].copy()

        #depois do filtro, sobram alguns fornecedores, que somente participam em um item, para remoção
        dados_mais_de_um_frn = dados_mais_de_um_frn.loc[dados_mais_de_um_frn.duplicated(subset='id_frn', keep=False),
:].copy()

        if dados_mais_de_um_frn.shape[0]==tamanho:
            break

    dict_itcp[itcp]['qtd_itens_levant_inicial'] = qt_inic = dados_mais_de_um_frn['id_itcp'].unique().shape[0]
    print('qtd_itens_levant_inicial:',qt_inic)

    if(qt_inic==0):
        print('Sem concorrência, só um participante por item!')
        return({})

    qt_inic2 = dados_mais_de_um_frn[dados_mais_de_um_frn['id_frn'].isin(partic)]['id_itcp'].unique().shape[0]
```

```
    if(qt_inic2==0):
        print('Participantes alvo eliminados na limpeza!')
        return({})
####------------------------------------------------------------------------
    ''' Calcula maior suporte e o menor dos participantes "alvo" do item de compra analisado
        O suporte mínimo é multiplicado por 0.95 para evitar que valor exato
        seja eliminado
        Nenhum dos valores realmente utilizados deverá ser menor que 0.01
    '''

    conta_reps_frn_all  = pd.DataFrame(dados_mais_de_um_frn['id_frn'].value_counts().reset_index(name='qtd_itcp'))

    # num_particp por itcp
    dict_itcp[itcp]['qtd_itens_maiores_partic'] = conta_reps_frn_all.head()



    dict_itcp[itcp]['qtd_itens_apos_limpeza'] = qt_itcp = qt_inic
    dados_frn_limpeza = dados_mais_de_um_frn.copy()


    conta_reps_frn = pd.DataFrame(dados_frn_limpeza[ dados_frn_limpeza['id_frn'].isin(partic) ]['id_frn']
                                  .value_counts().reset_index(name='qtd_itcp'))
    conta_reps_frn['sup'] = conta_reps_frn.apply(lambda row: row['qtd_itcp']/qt_itcp, axis=1)

    conta_reps_frn.rename(columns={'index': 'id_frn', }, inplace=True)

    menor_suporte_partic = 0.95*conta_reps_frn.sup.min()

    maior_suporte_partic = conta_reps_frn.sup.max()

    produto_suportes = conta_reps_frn.sup.prod()

    confianca_partic = produto_suportes / maior_suporte_partic

    sup_ut =  max(0.01, menor_suporte_partic)
    conf_ut = max(0.01, confianca_partic)

    print('suporte min utilizado:',sup_ut)
    print('confidence min utilizado:',conf_ut)

####------------------------------------------------------------------------
    ''' Formata os dados para aplicação do apriori
    '''

    dados_grp = (dados_frn_limpeza.groupby(['id_itcp', 'id_frn'])
                 .agg({'id_unid': [np.size]})
```

```
                .rename(columns={'id_unid': 'Qtd'})).dropna()

    dados_grp.columns = [tup[1] for tup in dados_grp.columns.values]

    desempilhada = (dados_grp.unstack().reset_index().fillna(0).set_index('id_itcp'))

    desempilhada_array = desempilhada.values
    desempilhada_array = np.where( desempilhada_array==0, 0, 1 )
    desempilhada = pd.DataFrame(data=desempilhada_array, columns=desempilhada.columns, index=desempilhada.index)

    desempilhada.columns = [str(tup[1]) for tup in desempilhada.columns.values]


####-------------------------------------------------------------------------
    ''' O 'min_support' será o menor suporte para os itemsets retornados
        para que retorne as conexoes entre os alvos, precisa ser um pouco menor que o min
        suporte entre os participantes.
        O 'max_len' é o tamanho máximo no número de elementos no itemset, inicialmente pareceu que
        o tamanho igual ao número de participantes alvo seria ideal, uma vez que são em geral 2 ou 3,
        mas não há praticidade no uso desse valor pois o numero de partipantes aumenta muito em alguns
        casos particulares, o que não só dificulta a análise como gera um sem número de itemsets sem
        qualquer interesse para o trabalho. Além disso, mesmo que no item de compra detectado tenham
        entrado três empresas relacionadas, a detecção que interessa é quando até duas delas
        tenham entrado em conjunto em algum outro item. Por isso foi fixado no valor 2.
    '''
    freq_itemsets = apriori( desempilhada, min_support=sup_ut, use_colnames=True, max_len=2 )

    rules = association_rules(freq_itemsets, metric="confidence", min_threshold=conf_ut )


####-------------------------------------------------------------------------
    ''' Resultados que indicam participação conjunta dos alvos
    '''

    result = rules[ rules['antecedents'].apply(lambda x: mesmo_elemento(set(x),partic) ) &
                    rules['consequents'].apply(lambda x: mesmo_elemento(set(x),partic) )]

    dict_itcp[itcp]['metricas_partic_e'] = result


    result_ou = rules[ rules['antecedents'].apply(lambda x: mesmo_elemento(set(x),partic) ) |
                       rules['consequents'].apply(lambda x: mesmo_elemento(set(x),partic) )]

    dict_itcp[itcp]['metricas_partic_ou'] = result_ou


####-------------------------------------------------------------------------
    ''' Resultados para comparação da participação conjunta dos alvos com os demais
```

```
        colunas de interesse: .iloc[:, np.r_[0,1,4,5,6,7,8]]
    '''

    dict_itcp[itcp]['metricas_grupo'] = rules
    return(dict_itcp)

#####################################################

def busca_associacoes_nos_ciclos(path='./'):

    dict_uf = {}
    itcp = ''
    i=0
    pos=path.find('Compras_')
    uf = path[pos+8:pos+10]

    nome_arquivo = '.'.join( ['itcp_da_uf', uf, 'metr'] )
    arquivo = os.path.join(path, nome_arquivo)

    for entry in os.scandir(path):

        if entry.is_file() and entry.name[-2:]=='nx':

            i+=1

            inic = datetime.now()
            print('Inicio item', i, ' - ', inic.strftime("%Y-%m-%d %H:%M"))
            print(entry.name)

            with open(os.path.join(path, entry.name),'rb') as rfile:
                grafo = pickle.load(rfile)

            dict_uf.update(analisa_grafo(grafo))

            with open(arquivo,'wb') as wfile:
                pickle.dump(dict_uf, wfile)

            print('Fim item', i,' - ', inic.strftime("%Y-%m-%d %H:%M"))
            print('Tempo de processamento do item:', datetime.now()-inic)
            print()

    return(i)

#####################################################
#####################################################

def main(arg):
```

```
    inicio_uf = datetime.now()
    print('Pasta:', arg, 'Início:', inicio_uf.strftime("%Y-%m-%d %H:%M"))
    print()

    n = busca_associacoes_nos_ciclos(arg)

    print('Fim:', arg, ' - ', n, 'itens', ' - ', 'Tempo decorrido:', datetime.now()-inicio_uf)
    print('Tempo médio por item:', (datetime.now()-inicio_uf)/n)


if __name__ == "__main__":
    pasta =''
    base_path = os.getcwd()
    if len(sys.argv) > 1:
        pasta = sys.argv[1:][0]
        main(os.path.join(base_path, pasta))
    else:
        main(base_path)
```