SGPROV: MECANISMO DE SUMARIZAÇÃO PARA MÚLTIPLOS GRAFOS DE PROVENIÊNCIA

Daniele El-Jaick Bentes de Souza Chenú

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Alexandre de Assis Bento Lima

Rio de Janeiro
Setembro de 2015

SGPROV: MECANISMO DE SUMARIZAÇÃO PARA MÚLTIPLOS GRAFOS DE PROVENIÊNCIA

Daniele El-Jaick Bentes de Souza Chenú

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

____
Profª. Marta Lima de Queirós Mattoso, D.Sc.

____
Prof. Alexandre de Assis Bento Lima, D.Sc.

____
Prof. Mario Roberto Folhadela Benevides, D.Sc.

____
Profª. Maria Luiza Machado Campos, D.Sc.

____
Prof. Fabio Andre Machado Porto, D.Sc.

____
Prof. José Antonio Fernandes de Macêdo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2015

Chenú, Daniele El-Jaick Bentes de Souza

SGProv: Mecanismo de Sumarização para Múltiplos Grafos de Proveniência/ Daniele El-Jaick Bentes de Souza Chenú. – Rio de Janeiro: UFRJ/COPPE, 2015.

XII, 101 p.: il.; 29,7 cm.

Orientadores: Marta Lima de Queirós Mattoso

Alexandre de Assis Bento Lima.

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2015.

Referências Bibliográficas: p. 96 - 101

1. SGBD de Grafos. 2. Proveniência. 3. Sumarização. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título

DEDICATÓRIA

A minha mãe, que é minha maior inspiração na vida. Sem ela este trabalho não teria sido possível.

Ao meu avô, que sempre patrocinou meus estudos e torceu por mim. Onde quer que ele esteja, tenho certeza de que esta muito feliz com a nossa conquista.

Ao meu filho, pelo apoio, preocupação, por aguentar meu mau humor em várias situações e por entender que nem sempre pude estar presente durante esses anos de doutorado.

Ao meu noivo, pelo grande incentivo, pela enorme paciência que teve comigo e por entender a minha ausência em várias ocasiões.

Ao meu professor José Chenú, que foi o maior incentivador da minha carreira e por ter me ensinado muito durante os anos em que trabalhamos juntos.

A todos os familiares e amigos que me apoiaram e torceram por mim durante todo o doutorado.

AGRADECIMENTOS

Meus sinceros agradecimentos,

Aos meus orientadores, Marta Mattoso e Alexandre Assis, pela parceria, dedicação, cuidado, inspiração, amizade e incentivo durante esses longos anos de trabalho. Muito obrigada por me aceitarem como orientada e aluna.

Aos professores Mario Benevides, Maria Luiza Campos, Fabio Porto e José Antônio de Macêdo, por aceitarem o convite para a minha banca.

Ao professor José Antônio de Macêdo por todo incentivo que me deu durante todo o doutorado.

Aos funcionários da COPPE/UFRJ, Solange Santos, Cláudia Prata, Ana Paula Rabello, Patrícia Leal e Gutierrez da Costa, por toda a ajuda e amizade que me deram durante todos esses anos.

Agradeço muito pela sorte de ter trabalhado com todos vocês.

À CAPES, pela concessão da bolsa de doutorado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SGPROV: MECANISMO DE SUMARIZAÇÃO PARA MÚLTIPLOS GRAFOS DE PROVENIÊNCIA

Daniele El-Jaick Bentes de Souza Chenú

Setembro/2015

Orientadores: Marta Lima de Queirós Mattoso
Alexandre de Assis Bento Lima

Programa: Engenharia de Sistemas e Computação

Os Sistemas de Gerência de Workflows Científicos (SGWfC) têm o objetivo de automatizar a construção e execução de experimentos científicos computacionais. Várias execuções de *workflows* são necessárias em um experimento. SGWfC geram rastros das execuções dos *workflows* por meio de dados de proveniência. Os dados de proveniência contêm o histórico da derivação do resultado do *workflow*, assim, pode ser representado sob a forma de um grafo direcionado e acíclico. A proveniência é importante para que os cientistas possam compreender, reproduzir e analisar seus experimentos. Cada execução de um *workflow* gera um grafo de proveniência. Após várias execuções, por exemplo, explorando parâmetros, inúmeros grafos são gerados. A base de dados de proveniência, portanto, requer um espaço de armazenamento considerável e consultá-la envolve a manipulação de um grande volume de grafos. Consultas típicas de proveniência percorrem os diversos grafos para obter o caminho de derivação (linhagem) dos dados da consulta. Esta tese apresenta um mecanismo de sumarização para grafos de proveniência (SGProv), usando um banco de dados de grafos para armazenar e consultar esses grafos. O objetivo é gerar um único grafo sumário que represente todos os grafos de proveniência gerados durante um experimento, mas com tamanho reduzido e eliminando dados repetidos. Esta abordagem de sumarização tem como objetivo reduzir o tempo de processamento de consultas de proveniência utilizando apenas o grafo sumário para respondê-las sem precisar reconstruir os grafos originais. Resultados obtidos com consultas de proveniência feitas no grafo sumário mostraram o potencial da nossa solução.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SGPROV: SUMMARIZATION MECHANISM FOR MULTIPLE PROVENANCE GRAPHS

Daniele El-Jaick Bentes de Souza Chenú

September/2015

Advisors: Marta Lima de Queirós Mattoso

Alexandre de Assis Bento Lima

Department: Systems Engineering and Computing

Scientific workflow management systems (SWfMS) are powerful tools in the automation of scientific experiments. Several workflow executions are necessary to accomplish one scientific experiment. SWfMS generate workflows executions traces through data provenance. Data provenance is about workflow results derivation, thus it is typically represented in the form of a directed acyclic graph. Data provenance is important for scientists to understand, reproduce and analyze their experiments. For each workflow execution, a provenance graph is generated. Numerous graphs are generated after several workflow runs, exploring different parameters. The resulting provenance database requires considerable storage space and querying it involves handling a large volume of graphs. Typical provenance queries process many graphs to get data derivation paths (lineage). This thesis presents SGProv, a summarization mechanism for provenance graphs, using a graph database to store and query them. The goal is to generate a single small summary graph that represents all provenance graphs generated during an experiment, eliminating redundant data. This summarization approach aims to reduce the processing time of provenance queries by using only the summary graph to answer them without the need for rebuilding the original graphs. Experimental results of provenance queries on the summary graph show performance improvements without data loss on query results.

Índice

Capítulo 1 Introdução ... 1
1.1 Problema ... 10
1.2 Hipótese ... 11
1.3 Objetivos ... 12
1.4 Contribuições ... 12
1.5 Organização ... 13
Capítulo 2 Proveniência, Modelo PROV e Consultas ... 15
2.1 Proveniência ... 15
2.2 PROV-DM ... 18
2.3 Consultas de Proveniência ... 20
Capítulo 3 Armazenamento de Dados de Proveniência ... 22
3.1 SGBD Relacional ... 22
3.2 SGBD de Grafos ... 23
3.2.1 Neo4j ... 23
3.2.2 Cypher ... 24
3.3 Abordagem Híbrida ... 25
3.4 SGBD XML ... 26
3.5 Comparação entre SGBD Relacional e SGBD de Grafo ... 27
3.5.1 Consultas Estruturais ... 28
3.5.2 Consultas aos Dados ... 29
3.5.3 Considerações Sobre a Comparação entre SGBDs ... 30
Capítulo 4 Técnicas de Compactação de Grafos ... 31
4.1 Sumarização de Grafos ... 31
4.1.1 Sumarização de Grafo com Erro Limitado ... 31
4.1.2 Sumarização Iterativa de Grafos ... 34
4.1.3 Sumarização de Grafo Orientada a Descoberta ... 36
4.1.4 Sumarização de Grafo Homogênea ... 38
4.2 Considerações Sobre a Sumarização ... 39
4.3 Compressão de Grafos ... 41
4.4 Redução de Grafos ... 47
4.4.1 Fatoração ... 48
4.4.2 Herança de Proveniência ... 52

4.5 Conclusões do Capítulo 55
Capítulo 5 O Mecanismo de Sumarização SGProv 57
5.1 Definições 57
5.2 Grafo Sumário 58
5.3 Exemplo de Aplicação do SGProv 64
Capítulo 6 Desenvolvimento do SGProv 71
6.1 Algoritmo 71
6.2 Compressão do Atributo Execução 77
6.3 Obtenção dos Grafos Originais a Partir do Sumário 79
Capítulo 7 Avaliação Experimental 82
7.1 Consultas de Proveniência 84
7.2 Avaliação dos Resultados Obtidos nas Consultas 87
7.3 Aplicação da Operação Inversa do Sumário 91
Capítulo 8 Conclusão 92
Referências Bibliográficas 96

Índice de Figuras

Figura 1. Experimento científico computacional. 3
Figura 2. Grafo sumário gerado pelo SGProv. a) Supervértice e superaresta. b) Subgrafo de atributos do grafo sumário. 8
Figura 3. Proveniência prospectiva e retrospectiva (Freire *et al.,* 2008)........................ 16
Figura 4. Base de proveniência. 17
Figura 5. Estruturas centrais do PROV-DM.. 18
Figura 6. Grafo de pessoas e seus amigos. 24
Figura 7. Grafo original (G), sumário (S) e lista de arestas de correção (C) (Navlakha *et al.,* 2008)... 32
Figura 8. Expansão das arestas de y. 33
Figura 9. Aplicando lista de arestas de correção. 33
Figura 10. Grafo original (a) e Grafo Sumário (b) (Yu *et al.,* 2010)............................. 35
Figura 11. Grafo original de artigos e três exemplos de sumários com categorização de atributos (Zhang *et al.*, 2010). 37
Figura 12. Dados de proveniência com grafo e lista de adjacência correspondentes (Xie *et al.,* 2011).. 42
Figura 13. Primeiro passo da compressão por referência (Xie *et al.,* 2011). 43
Figura 14. Segundo passo da compressão por referência (Xie *et al.,* 2011). 44
Figura 15. Terceiro passo da compressão por referência (Xie *et al.,* 2011).................. 44
Figura 16. Buscar lista de sucessores similares usando como referência para um vértice a lista de outro vértice com o mesmo nome (Xie *et al.,* 2011)..................................... 45
Figura 17. Codificar lacunas entre os sucessores de um único vértice e entre sucessores de vértices diferentes (Xie *et al.,* 2011)... 46
Figura 18. Exemplo de fatoração básica (Chapman *et al.*, 2008)................................. 48
Figura 19. Registro de proveniência com tamanhos diferentes (Chapman *et al.,* 2008).50
Figura 20. Exemplo da figura 19 com e sem fatoração de vértices (Chapman *et al.,* 2008)... 51
Figura 21. Exemplo de registros de proveniências iguais, mas com o argumento PubMed diferente (Chapman *et al.,* 2008). 51
Figura 22. Fatoração de argumentos (Chapman *et al.* 2008).. 52
Figura 23. Herança de Estrutura (Chapman *et al.,* 2008)... 53
Figura 24. Herança de predicado de dados com mesmo nome (Chapman *et al.,* 2008). 54
Figura 25. Exemplo de sumarização do SGProv.. 60

Figura 26. Exemplo de grafo de atributos. 62
Figura 27. Primeira iteração do SGProv... 65
Figura 28. Parte do grafo de atributos de G_{s1}. 67
Figura 29. Segunda iteração do mecanismo de sumarização. 69
Figura 30. Grafo sumário antes da compressão do atributo "*execução*"........................ 78
Figura 31. Grafo sumário após a compressão do atributo "*execução*"........................... 78
Figura 32. Grafo sumário e seu grafo de atributos. 80
Figura 33. Vértices do grafo de proveniência 0 (G_{p0}). 80
Figura 34. Grafo de proveniência 0 (G_{p0})... 81
Figura 35. Grafo de proveniência 0 e 1 (G_{p0} e G_{p1}) e grafo sumário (G_s). 81
Figura 36. Exemplos de grafos de proveniência gerados para a avaliação experimental. 82
Figura 37. Resultado da consulta 8 na base do sumário sem compressão e na base do sumário. 90
Figura 38. Consultas usadas na operação inversa do sumário....................................... 91
Figura 39. Redução das Bases de Testes obtida com o SGProv. 92

Índice de Tabelas

Tabela 1. Tamanhos das bases de dados (Vicknair *et al.* 2010). 27

Tabela 2. Resultados das consultas estruturais, em milissegundos (Vicknair *et al.,* 2010). 28

Tabela 3. Resultados das consultas D1 e D2, em milissegundos (Vicknair *et al.,* 2010). 29

Tabela 4. Resultado da consulta D3, em milissegundos, onde d é o tamanho da cadeia de caracteres procurada (Vicknair *et al.,* 2010). 29

Tabela 5. Resumo da Sumarização 40

Tabela 6. Resumo da Compressão.. 47

Tabela 7. Resumo da Redução 55

Tabela 8. Características da base de testes e da base do sumário produzida pelo SGProv. 83

Tabela 9. Tamanhos em disco das bases de dados. 84

Tabela 10. Média do tempo de processamento das consultas. 88

Tabela 11. Redução aproximada do tempo de processamento das consultas................ 88

Tabela 12. Tempo de processamento da consulta 8 (C8). 90

Capítulo 1 Introdução

A validação de uma hipótese científica pode ser realizada mediante a execução de uma grande quantidade de experimentos. Com o desenvolvimento das pesquisas científicas, o aumento do volume de dados manipulados por elas e o surgimento de áreas da ciência multidisciplinares, o uso de ferramentas computacionais se tornou fundamental para apoiar a execução e análise de tais experimentos (Anand *et al.,* 2009).

Experimentos científicos apoiados por computadores são baseados em simulações realizadas tipicamente por cadeias de programas, que, geralmente, são representadas por *workflows* científicos. A natureza exploratória dos experimentos científicos faz com que um mesmo *workflow* seja executado sob diversas concepções (Taylor *et al,* 2007). Por exemplo, um cientista pode querer analisar os resultados da execução de uma mesma especificação do *workflow* com diferentes programas, dados de entrada ou combinações de parâmetros, como na varredura de parâmetros (Abramson *et al*., 2011). Em outra concepção o cientista pode alargar ou estreitar a faixa de dados sendo analisada pelo *workflow*, ou ainda mudar critérios de similaridade (Ocaña *et al.,* 2011b) ou convergência (Guerra *et al.,* 2012) do experimento. Com base nas análises destas diversas execuções do *workflow*, o cientista pode também optar por trocar, na especificação do *workflow*, alguns dos programas utilizados. Esta troca pode ser motivada por uma substituição de programas que estejam em maior sintonia com o comportamento do fluxo de dados que vem sendo gerado (Santos *et al.,* 2013). Um exemplo é o *workflow SciHMM* descrito em Ocanã *et al.* (2011a), onde uma das atividades (*Construção MSA*) pode ser realizada usando-se um entre cinco programas. Cada programa testado exige a reexecução do *workflow*. Assim, para um único experimento, inúmeras execuções são realizadas (Gil *et al.,* 2007; Mattoso *et al.,* 2010).

Os dados intermediários e finais produzidos durante a execução de um experimento são fundamentais para que este seja considerado consistente e válido pelo cientista. Os Sistemas de Gerência de *Workflows* Científicos (SGWfC), como Kepler[1] , Taverna[2] e VisTrails[3] coletam automaticamente os dados gerados durante as execuções

1 https://kepler-project.org
2 http://www.taverna.org.uk
3 http://www.vistrails.org/index.php/Main Page

dos *workflows.* A este tipo de informação é dado o nome de proveniência (Buneman *et al.,* 2001). Para que os cientistas possam entender, reproduzir e analisar seus experimentos científicos, os dados de proveniência devem ser coletados, armazenados e consultados pelos mesmos (Davidson *et al.,* 2008). Assim, os dados de proveniência são tradicionalmente explorados nas pesquisas científicas com o objetivo de explicar o processo usado para gerar determinado resultado (dados produzidos) e permitir que os cientistas avaliem seu valor (Moreau *et al.,* 2009).

O uso de dados de proveniência como ferramenta de análise e tomada de decisão se tornou ainda mais abrangente com a iniciativa de padronização da representação de dados de proveniência por parte do *World Wide Web Consortium* (W3C) através do PROV[4]. O PROV consiste em um conjunto ou família de especificações para vários aspectos envolvidos na representação de dados de proveniência. A especificação que descreve o modelo de dados adotado é denominada PROV-DM (Belhajjame *et al.,* 2013) e (Moreau e Missier, 2012), que visa a representação de inúmeras áreas de aplicação englobando a web semântica, os sistemas de bancos de dados e as aplicações científicas. De acordo com o PROV-DM, os dados de proveniência são baseados em objetos (dados e programas) e seus relacionamentos (dependências), que são representados na forma de grafos direcionados e acíclicos (DAG) (Freire *et al.,* 2008). Considere, por exemplo, um objeto *O* derivado de outro objeto *P*. Diz-se então que, *P* é antecessor de *O*, ou ainda, que *O* é descendente de *P*. Este é um relacionamento do tipo **linhagem**. Como um objeto pode ser derivado de várias fontes e também ser fonte de vários outros objetos, a forma de representação destes dados deve ser um grafo. Como um objeto não pode ser descendente dele mesmo, este tipo de grafo deve ser acíclico. Como são possíveis vários tipos de relações de linhagem, deve ser possível nomear as arestas do grafo. Independente da complexidade do *workflow*, o grafo de proveniência resultante é um DAG. Por exemplo, execução condicional e paralelismo afetam a complexidade do *workflow*, mas não afetam o DAG de proveniência (Anand *et al.,* 2010).

Cada execução de um *workflow* gera um grafo de proveniência (Figura 1). Após várias execuções são gerados inúmeros grafos, individualmente não tão volumosos quanto, por exemplo, os das redes sociais. Uma base de dados de proveniência,

---

4 http://www.w3.org/TR/prov-primer/

portanto, é composta por vários DAG e, conforme o número de experimentos aumenta, necessita de um espaço de armazenamento cada vez maior.

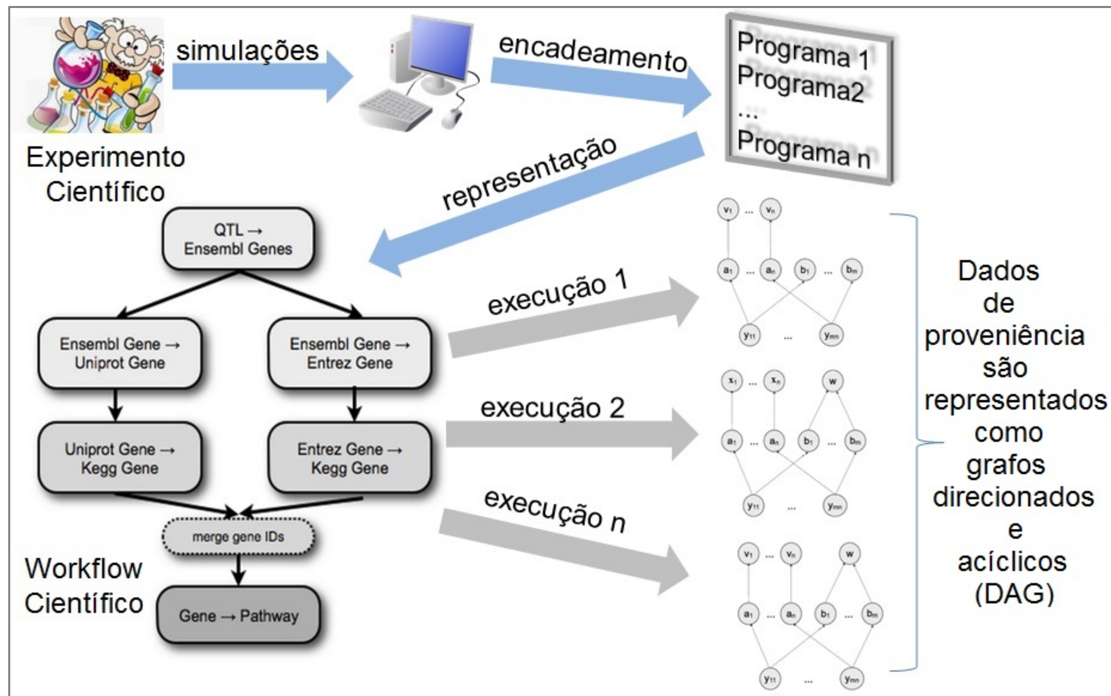


Figura 1. Experimento científico computacional.

Em cada DAG, os vértices representam atividades (programas), ou entidades (dados), e possuem esquemas variados que, na maioria das vezes, não são previamente conhecidos. Os vértices podem ainda ter muitas arestas entre eles. Neste cenário, as consultas são complexas, pois envolvem buscas em todo o conjunto de grafos gerados em um experimento científico. Além disso, elas precisam retornar subgrafos que preservem os relacionamentos de linhagem dos objetos, para que o cientista possa avaliar a derivação dos mesmos e identificar, por exemplo, possíveis erros no seu experimento. Estas características fazem com que a habilidade de armazenar e consultar os dados de proveniência ainda seja um desafio significativo (Davidson *et al.,* 2008).

Um exemplo de consulta de proveniência é "*Q1. Encontre todas as dependências diretas e indiretas do dado X*". Esta é consulta mais genérica, considerada a linha de base das consultas de proveniência, retorna todas as dependências diretas e indiretas de um dado. O retorno de *Q1* é um subgrafo do grafo de proveniência que inclui todos os caminhos que partem de *X,* ou que chegam até ele. A consulta *Q1* pode ter duas variações: a busca por dependências anteriores, como, por exemplo, "*Q2. Encontre todos os dados usados para gerar o dado X*", ou por dependências posteriores, como, por exemplo, "*Q3. Encontre todos os dados que são derivados a partir do dado X*". *Q2* é usada para explicar a presença do dado *X* no resultado obtido. *Q3* é usada para

analisar o impacto da influência direta ou indireta do dado *X* no resultado obtido Woodman *et al.* (2011).

Há outros exemplos de consultas como: "*Q4. Encontre todas as execuções de workflows em que o programa Y foi executado em algum momento posterior ao programa X*". Tal consulta poderia envolver todos os grafos de proveniência, exigindo a execução de travessias em cada um deles, o que pode resultar em grande tempo de processamento. Outra consulta útil para um cientista seria: "*Q5. Encontre todas as execuções de workflows que utilizam o programa Z e que geraram resultados com margem de erro superior a W*". Esta consulta ajudaria o cientista a entender melhor os resultados e possivelmente executar os *workflows* envolvidos com um programa similar a *Z*, verificando se a margem de erro persiste. Ela também pode envolver vários grafos e exige a execução de travessias.

A abordagem mais usada na literatura para tratar o problema de armazenamento e consulta de dados de proveniência é utilizar um SGBD relacional como em Huahai e Singh (2008) e Anand *et al.* (2012). Para isso, o grafo de proveniência deve ser mapeado para um esquema relacional e as consultas devem ser formuladas em SQL. O principal problema desta abordagem é a complexidade e o tempo de processamento das consultas que, na maioria das vezes, envolvem muitas junções entre relações. A exigência de esquemas de dados rígidos, característica do modelo relacional, também representa um problema para a sua utilização, uma vez que os dados de proveniência, geralmente, possuem esquemas flexíveis ou variáveis (Davidson *et al.,* 2008).

Armazenar os grafos de proveniência em um SGBD de grafos (Angles e Gutierrez 2008) e utilizar suas operações nativas para percorrer, recuperar e consultar dados é uma alternativa para os problemas relacionados ao uso de SGBD relacionais, como advogado em (Woodman *et al.*, 2011). Os SGBD de grafos, como o nome sugere, representam os dados diretamente como grafos: seu modelo de dados possui, como construtos básicos, vértices e arestas, podendo, cada um, conter atributos para representar e armazenar informações. Na maioria dos SGBD desta classe, os vértices não possuem esquema rígido, podendo, cada um, possuir uma estrutura diferente dentro de um mesmo grafo. Por exemplo, vértices que representam atividades podem possuir quantidades e tipos de atributos diferentes. Além disso, um SGBD de grafos implementa operações clássicas sobre grafos tais como travessias, buscas em largura, buscas em profundidade e determinação do menor caminho entre dois vértices. Desta

forma, as consultas são feitas percorrendo os caminhos dos grafos, que já se encontram armazenados no banco de dados em formato conveniente, não havendo a necessidade de execução de junções entre as relações, como no modelo relacional.

Pesquisas relacionadas aos SGBD de grafos foram populares nos anos 90 (Guting, 1994), mas perderam a força por vários motivos, como o surgimento dos hipertextos e de pesquisas de SGBD XML. Com o aumento significativo do volume de dados e dos relacionamentos entre eles, como, por exemplo, em aplicações científicas e nas redes sociais, as pesquisas relacionadas a grafos voltaram a ganhar importância (Vicknair *et al.,* 2010). Desta forma, foram desenvolvidos SGBD voltados especificamente para a manipulação de grafos, como, por exemplo, o Neo4j[5] e o Sparksee[6].

Na literatura, são encontrados trabalhos que avaliam o desempenho dos SGBD de grafos para armazenar e consultar os dados de proveniência. O trabalho de Cuervas-Vicenttín *et al.* (2014) propõe um repositório (PBase) de dados de proveniência originados pelas execuções de *workflows* científicos utilizando o SGBD de grafos e sua linguagem de consultas Cypher[7] para executar as consultas de proveniência. O trabalho concluiu que consultas feitas em grafos de proveniência muito volumosos tiveram desempenho insatisfatório no Neo4j. Segundo os autores, para contornar este problema é necessário incorporar técnicas de indexação e codificação de dados no Neo4j. Desta forma, eles aplicaram a técnica de compressão de relacionamentos transitivos (Agrawal *et al.,* 1989), baseada na atribuição de intervalos numéricos aos vértices, que são computados somente uma vez e são armazenados para serem usados nas consultas. Estes intervalos indicam quais são os sucessores diretos e indiretos de um vértice, sem que seja necessário listar individualmente todos os vértices que pertencem a um determinado intervalo. Entretanto, a aplicação desta técnica obteve resultados significativos somente para as consultas de **acessibilidade**, tipo particular de **linhagem**, que determina se existe um caminho entre dois vértices. Aplicando esta técnica em um grafo com 10 mil vértices, consultas aleatórias de **acessibilidade** foram executadas em menos de um minuto, enquanto sem a utilização da técnica o tempo de execução foi superior a duas horas.

[5] http://www.neo4j.org/
[6] http://www.sparsity-technologies.com
[7] http://www.neo4j.org/learn/cypher

Na literatura, também são encontrados trabalhos que comparam o desempenho de SGBD relacionais e de grafos para armazenar e consultar dados de proveniência. O trabalho descrito em Vicknair *et al.* (2010) compara o desempenho do SGBD relacional MySQL e do SGBD de grafos Neo4j considerando o tempo de processamento de consultas de proveniência. O trabalho concluiu que o Neo4j obteve melhor desempenho no processamento da maioria das consultas, principalmente nas consultas estruturais, que envolvem travessias em grafos de proveniência.

O trabalho de Soares (2013) avalia a utilização do SGBD relacional PostgreSQL e do SGBD de grafos Neo4j para o armazenamento e acesso aos dados de proveniência, a fim de descobrir a mais adequada. Para avaliar o desempenho das consultas, foram utilizadas diferentes abordagens, como o uso de visões recursivas e procedimentos armazenados para o PostgreSQL, além da linguagem Cypher e da implementação de consultas em Java para o Neo4j. O trabalho concluiu que o tempo de processamento das consultas estruturais (que levam em conta as topologias dos grafos) implementadas em Java e em Cypher, no Neo4j, tiveram um desempenho muito superior às implementadas com visões recursivas no PostgreSQL. Para algumas destas consultas, as versões implementadas como procedimentos armazenados no PostgreSQL tiveram desempenho próximo aos das versões implementadas no Neo4j. O trabalho também concluiu que o uso de visões recursivas deve ser evitado, principalmente se o SGBD relacional armazenar diversas execuções de um *workflow* em uma base com grande volume de dados. Neste caso, o uso de procedimentos armazenados pode ser utilizado como alternativa, pois melhora consideravelmente o desempenho.

Avaliando os resultados obtidos pelos trabalhos citados, é possível partir da premissa de que utilizar um SGBD de grafos para armazenar e consultar os dados de proveniência é a melhor alternativa. Entretanto, na maioria dos casos, estes SGBD são projetados para armazenar um único grafo, mesmo que muito volumoso, em cada base de dados, e não um conjunto com um grande número de grafos independentes em uma única base, como é o caso dos dados de proveniência. Desta forma, permanece o problema da manipulação do grande volume de grafos de proveniência produzidos por diversas execuções de *workflows* em um experimento científico, o que pode tornar as consultas complexas e comprometer seu tempo de resposta. Neste caso, é preciso projetar métodos eficientes para armazenar, recuperar e consultar estes dados, a fim de melhorar o tempo de processamento das consultas (Zhang *et al.,* 2010).

Novas técnicas estão sendo estudadas para atender estes requisitos das aplicações atuais. Como exemplo, existem as técnicas para projetar a construção compactada de um grafo, que procuram reduzir significativamente o seu tamanho com base na contração de vértices e arestas, mas mantendo informações suficientes para responder consultas sobre o mesmo. O objetivo é definir um grafo compactado, que contenha a estrutura relevante do grafo original, e que possa ser alocado na memória principal do computador, para que a ele sejam aplicados os algoritmos clássicos de grafos, como buscas e determinação de caminho mínimo (Aggarwal e Wang, 2010). Na literatura são encontradas três abordagens de compactação de grafos: **compressão** (Xie *et al.*, 2011; Fan *et al.,* 2012; Teixeira *et al.*, 2012); **redução** (Chapman *et al.,* 2008) **e sumarização** (Navlakha *et al.,* 2008; Liu e Yu, 2011). O desenvolvimento de uma técnica de sumarização é a proposta desta tese.

Esta tese propõe o SGProv (El-Jaick *et al.,* 2013; 2014), um mecanismo de sumarização para grafos de proveniência gerados a partir de dados coletados pelos SGWfC durante várias execuções de *workflows* em um experimento científico computacional. Como o PROV-DM é o modelo recomendado pela W3C para a representação de dados de proveniência, o SGProv trata e representa grafos de proveniência de acordo com este modelo. O objetivo do SGProv é gerar um único grafo sumário reduzido que represente todos os grafos de proveniência originados em um experimento, preservando suas estruturas e eliminando dados redundantes. Esta abordagem de sumarização tem como foco reduzir o tempo de processamento das consultas de proveniência, uma vez que estas podem ser respondidas utilizando somente o grafo sumário e sem a necessidade de refazer os grafos de proveniência originais. O SGProv utiliza um SGBD de grafos para armazenar o grafo sumário.

Para agrupar vértices e arestas dos grafos de proveniência o SGProv se baseia fundamentalmente em uma característica dos experimentos científicos computacionais: neles é comum que cada atividade seja executada repetidamente, mas com diferentes combinações de parâmetros, em cada variação de um *workflow*. Assim, geralmente, vértices que representam uma mesma atividade repetem-se nos vários grafos de proveniência, enquanto vértices que representam entidades apresentam maior variação, pois correspondem aos dados de entrada das atividades, e também aos resultados intermediários e finais obtidos nas execuções dos experimentos. Com base nesta característica, o SGProv agrupa conjuntos de vértices que representam atividades (ou

entidades) correspondentes, ou seja, conjuntos de atividades (ou entidades) que possuem o mesmo atributo identificador.

Atividades (entidades) correspondentes são representadas uma única vez como um vértice do grafo sumário (chamado de supervértice). Atividades (entidades) que foram utilizadas numa única execução também são representadas no sumário como um supervértice com um único elemento. As arestas correspondentes dos grafos de proveniência são agrupadas em uma aresta do grafo sumário (chamada de superaresta), quando possuem mesmo tipo, vértice de origem e de destino (Figura 2.a). Atividades e entidades podem ter atributos como nomes iguais e, algumas vezes, valores também iguais, em diferentes grafos de proveniência. Conjuntos de nomes e valores de atributos repetidos nos grafos de proveniência são armazenados em supervértices de um subgrafo do sumário, chamado de grafo de atributos, para evitar redundâncias no sumário (Figura 2.b).

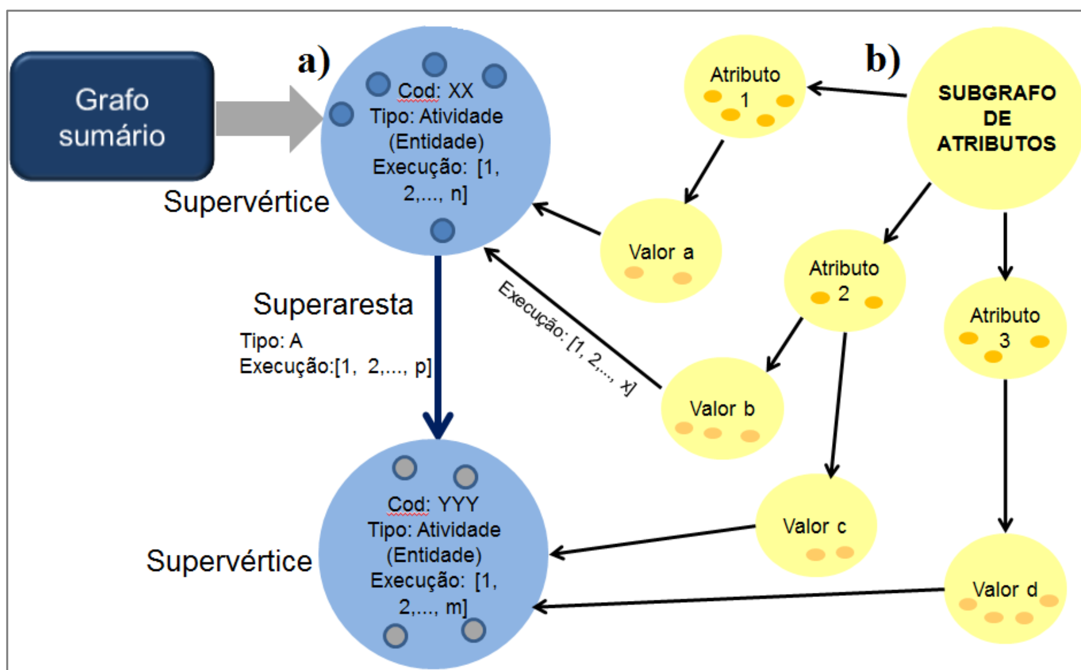


Figura 2. Grafo sumário gerado pelo SGProv. a) Supervértice e superaresta. b) Subgrafo de atributos do grafo sumário.

Para que seja possível identificar a qual execução de um *workflow* um supervértice ou superaresta pertence é adicionado a eles um atributo identificador de execução ("*execução*"), com uma lista de valores das execuções de que fizeram parte. Como, por exemplo, uma mesma atividade pode ser usada em *n* execuções, o atributo "*execução*" pode ser uma lista com muitos valores. Para não comprometer o tempo de execução das consultas de proveniência, que envolvam buscas no atributo "*execução*",

foi desenvolvida uma melhoria deste atributo. Esta melhoria consiste na compressão dos valores do atributo, com base na codificação dos intervalos de seus valores. Assim, o SGProv gera um único grafo sumário que representa todo o conjunto de grafos de proveniência gerados durante um experimento.

Como a sumarização deve ser feita sem perda de dados, para que os cientistas possam validar seus experimentos sem serem induzidos a erros, foi desenvolvida a operação inversa do sumário, para mostrar que é possível reconstruir os grafos de proveniência originais a partir do sumário. A operação inversa consiste na expansão de supervértices e superarestas (Navlakha *et al.,* 2008) e na verificação dos valores dos seus atributos "*execução*", para refazer os grafos de proveniência originais.

Os trabalhos encontrados na literatura, como (Navlakha *et al.,* 2008), (Yu *et al.,* 2010), (Zhang *et al.,* 2010) e (Liu e Yu, 2011), se concentram na sumarização de um único grafo muito volumoso e sem igualdades entre seus vértices. Assim, a sumarização é feita com base na teoria dos grafos (Szwarcfiter e Markenzon, 2012). Por exemplo, um clique (Szwarcfiter, 2003) é sumarizado em um único supervértice, com base nas suas arestas (vértices que compartilham arestas com um mesmo conjunto de vizinhos são sumarizados em um único supervértice), ou com base no desenvolvimento de funções de similaridade complexas. Além disso, estes trabalhos fazem a sumarização de grafos onde a semântica do modelo não é previamente conhecida.

A sumarização proposta nesta tese, por outro lado, tem ciência da semântica do modelo, uma vez que é desenvolvida especificamente para grafos de proveniência no modelo PROV-DM. As consultas de proveniência, geralmente, são aplicadas a uma grande quantidade de grafos. Contudo, cada grafo é formado por um pequeno número de vértices e arestas. Além disso, estes grafos possuem similaridades entre si, pois são resultantes de variações dos *workflows*.

Para analisar o tempo de processamento das consultas faz parte desta tese responder consultas tradicionais de proveniência de duas formas: baseadas nos vários grafos de proveniência resultantes das execuções dos *workflows* e baseadas somente no grafo sumário. No primeiro caso, para responder as consultas é preciso percorrer todos os grafos de proveniência. Mesmo que um único grafo seja percorrido em pouco tempo, ainda persiste a complexidade de gerenciar a repetição da consulta para cada grafo e de agrupar os resultados obtidos. Isto porque os SGBD de grafos são projetados para

gerenciar um único grafo volumoso e não um conjunto de grafos independentes. Utilizar um único grafo para representar conjuntos de grafos exige o emprego de certos artifícios por parte do usuário. No segundo caso, para responder as consultas é preciso percorrer um único grafo sumário, ao invés de uma grande coleção de grafos. Desta forma, o tempo de processamento das consultas é reduzido e os mesmos resultados corretos são fornecidos. No entanto, para realizar as consultas, o usuário precisa estar ciente das características do grafo sumário.

Desta forma, foi realizada uma avaliação experimental com aplicação da sumarização em quatro bases de testes, constituídas por quantidades diferentes de grafos de proveniência. Para cada base de testes foi gerada uma base do sumário. Na avaliação, consultas de proveniência foram executadas sobre as bases de testes e as respectivas bases dos sumários geradas, para analisar os tempos de processamento em ambos os casos. Os resultados obtidos na avaliação experimental mostraram uma redução entre 41,97% e 99,99% dos tempos de processamentos das consultas aplicadas nas bases dos sumários, em relação às mesmas consultas aplicadas nas bases de testes. Além disso, a sumarização reduziu em aproximadamente 99% os tamanhos das bases de testes, considerando a quantidade de vértices e arestas. Esta redução varia de acordo com as características do conjunto de grafos de proveniência. Se todos os grafos possuírem uma variedade relativamente pequena de atividades (entidades), havendo muitos grafos com as mesmas atividades (entidades), a redução é significativa, como é o caso apresentado nesta tese. Se, ao contrário, as atividades (entidades) variarem muito de grafo para grafo, a redução será pequena, podendo até ser nula, no caso de atividades (entidades) nunca presentes em mais de uma execução de *workflow*. As próximas seções definem o problema tratado nesta tese, sua hipótese geral, os objetivos pretendidos, as principais contribuições obtidas e sua organização.

1.1 Problema

O problema abordado nesta tese consiste na ausência de soluções para realizar consultas de linhagem sobre conjuntos de grafos de proveniência de modo eficiente (com curto tempo de resposta) retornando um ou mais subgrafos, ou um conjunto de grafos (no lugar de conjunto de vértices). Tal problema foi considerado como um desafio em (Aggarwal e Wang, 2010) e ainda hoje permanece em aberto. Conforme

discutido, SGBD relacionais e de grafos (como implementados atualmente) não proporcionam o desempenho necessário.

A abordagem de Cuervas-Vicenttín *et al.* (2014) com o Neo4j, é a que mais se aproxima do problema endereçado nesta tese, entretanto eles se concentram no problema de processamento de consultas a dados de proveniência que precisam percorrer grafos com grande volume de vértices (na ordem de 10 mil vértices). No entanto, trabalhos publicados na literatura voltada à proveniência, em especial a de *workflows* científicos, como nos eventos *International Provenance and Annotation Workshop Series*[8] (IPAW) e *Usenix Workshop on the Theory and Practice of Provenance*[9] (TaPP), mostram que dados de proveniência gerados por SGWfC tendem a ser formados por grandes volumes de pequenos grafos, com, aproximadamente, 20 vértices cada um. Essa evidência também pode ser constatada em execuções de diversos *workflows* científicos com os sistemas Chiron (Ogasawara *et al.,* 2013), SciCumulus (Oliveira *et al.,* 2011), Pegasus (Deelman *et al.,* 2005) e Swift (Gadelha *et al.,* 2013). Alguns, podem ser observados em casos reais de aplicações científicas na área de bioinformática com em (Ocanã *et al.*, 2011b) e na área de óleo e gás. Outros, são evidenciados na execução do *workflow* de astronomia Montage (Deelman, 2010), um *workflow* usado como benchmark para avaliar SGWfC e há ainda os *workflows* utilizados na série de *Provenance Challenges*[10].

1.2 Hipótese

Partindo da premissa de que a melhor forma de armazenar e consultar grafos é utilizar um SGBD de grafos, a hipótese geral desta tese é a de que, mesmo utilizando SGBD de grafos para armazenar e consultar os dados de proveniência, persiste o problema de consultar o grande volume de grafos produzidos em um experimento científico computacional. Outra hipótese desta tese é que aplicando uma das formas de compactação, desenvolvida para grafos volumosos, nos vários grafos de proveniência, é possível obter um único grafo com volume de dados reduzido e, assim, também reduzir o tempo de processamento de consultas.

8 http://www.ipaw.info/
9 https://www.usenix.org/conferences
10 http://twiki.ipaw.info/bin/view/Challenge/

A compactação deve ser feita sem perda de dados, para que os cientistas possam validar seus experimentos sem serem induzidos a erros. Para que seja possível responder a consultas de proveniência o grafo compactado deve preservar as estruturas (arestas) dos grafos originais. Para tornar as consultas mais simples e reduzir seu tempo de processamento, elas devem ser feitas utilizando unicamente o grafo compactado, sem que seja necessário consultar os diversos grafos gerados ou desfazer a compactação.

1.3 Objetivos

O objetivo desta tese é desenvolver uma solução para reduzir o tempo de processamento de consultas feitas sobre um conjunto de diversos grafos de proveniência gerados durante várias execuções de *workflows* científicos em um experimento científico computacional.

A solução proposta consiste em gerar um único grafo compactado que represente todos os grafos gerados durante um experimento científico, de tal modo que suas redundâncias sejam armazenadas uma única vez e suas diferenças sejam evidenciadas. Assim, o volume de dados no grafo compactado será reduzido, melhorando o tempo de processamento das consultas, de acordo com as hipóteses descritas. Além disso, as consultas podem ser feitas somente com base no grafo compactado, eliminando a necessidade de consultar os vários grafos de proveniência originais.

1.4 Contribuições

As principais contribuições desta tese são:

- Desenvolvimento de uma técnica de sumarização, que se baseia nas similaridades entre os grafos de proveniência. Esta técnica gera um único grafo sumário, sem vértices e nem arestas redundantes. O grafo sumário pode ser usado para responder consultas de proveniência e melhorar o tempo de processamento das mesmas.
- Possibilidade de escolha de qualquer atributo dos vértices para realizar a sumarização. Os vértices que representam atividades tendem a se repetir nos grafos de proveniência. Os que representam entidades, no entanto, tendem a apresentar maior variação. Neste caso, agrupar vértices correspondentes pode gerar um grafo sumário tão volumoso quanto o conjunto de grafos de proveniência. O atributo usado na sumarização deve ser escolhido pelo

usuário e, idealmente, deve ser o que mais se repete nos tipos de vértices dos grafos, para que o grafo sumário seja o mais compactado possível.

- Desenvolvimento de uma técnica de sumarização para atributos dos vértices e seus valores, com o objetivo de evitar repetições no grafo sumário. Isto porque atividades (entidades) correspondentes no conjunto de grafos de proveniência podem ter atributos e valores iguais, ou valores distintos para um mesmo atributo.
- Aplicação de uma técnica de compressão na lista de cada atributo "*execução*". Isto porque o atributo "*execução*" contém a lista de todas as execuções do *workflow* a que supervértices e superarestas pertencem. Se um supervértice (superaresta) pertencer a milhares de execuções, este atributo pode ter diversas entradas em sua lista. Desta forma, consultas que façam busca neste atributo podem ter seu tempo de processamento comprometido. A técnica de compressão consiste na codificação de lacunas entre os números da lista do atributo "*execução*", com o objetivo de reduzir seu tamanho e melhorar o tempo de processamento das consultas.
- Desenvolvimento da operação inversa do sumário, para mostrar que os grafos de proveniência podem ser reconstruídos a partir do sumário e sem perda de dados.

1.5 Organização

Esta tese está organizada da seguinte forma. O Capítulo 2 apresenta o conceito de proveniência, o modelo de proveniência PROV-DM e os principais padrões de consultas de proveniência. O Capítulo 3 discute as formas de armazenamento e de consulta dos grafos de proveniência encontradas na literatura e, também, apresenta uma comparação entre o uso de SGBD relacionais e o uso de SGBD de grafos. O Capítulo 4 descreve e analisa trabalhos encontrados na literatura relacionados às técnicas de compactação de grafos (sumarização, compressão e fatoração). O Capítulo 5 apresenta o SGProv, suas principais definições, como o grafo sumário é gerado a partir de um conjunto de grafos de proveniência e apresenta um exemplo de utilização do SGProv. O Capítulo 6 descreve o desenvolvimento do SGProv, seu algoritmo, a compressão do atributo "*execução*" e a obtenção dos grafos originais a partir do grafo sumário gerado pelo SGProv. O Capítulo 7 apresenta a avaliação experimental que foi feita com o

SGProv, as consultas de proveniência que foram realizadas nos grafos de proveniência e no grafo sumário e discute os resultados obtidos. Por último, o Capítulo 8 apresenta as conclusões desta tese e os trabalhos futuros.

Capítulo 2 Proveniência, Modelo PROV e Consultas

Este capítulo aborda o conceito de proveniência e sua importância para os experimentos científicos, descreve o modelo de dados de proveniência PROV-DM, uma das especificações do PROV para interoperabilidade entre os Sistemas de Gerência de Workflows Científicos (SGWfC) e apresenta os principais padrões de consultas de proveniência.

2.1 Proveniência

Os experimentos científicos computacionais podem ser modelados como um conjunto de atividades e um fluxo de dados entre elas. Cada atividade pode ser um programa de computador que processa um conjunto de dados de entrada e produz outro conjunto de dados de saída, gerando o fluxo de dados dos chamados *workflows* científicos. Um workflow científico pode ser definido como a especificação formal de um processo que representa os passos a serem executados em um determinado experimento científico. Estes passos são normalmente associados à seleção de dados, análise e visualização (Silva *et al.,* 2010).

Um *workflow* científico pode ser executado inúmeras vezes no contexto de um único projeto, gerando uma grande quantidade de dados intermediários e finais de interesse do usuário. Os Sistemas de Gerência de *Workflows* Científicos (SGWfC) têm o objetivo de automatizar a construção e execução de tais *workflows*, possibilitando ao cientista realizar uma espécie de programação em alto nível, através do encadeamento de programas que seguem um determinado fluxo e lógica (Taylor *et al.,* 2007).

Os SGWfC registram o fluxo de dados e atividades do *workflow*, capturando os dados de proveniência, que contêm o histórico de derivação dos dados de um produto, incluindo seu repositório de dados original, produtos intermediários e os programas que foram aplicados para produzi-lo (Freire *et al.,* 2008). Existem dois tipos de proveniência: prospectiva e retrospectiva. Proveniência prospectiva se refere à especificação de uma atividade e relaciona os passos que devem ser seguidos para gerar o resultado do experimento. A proveniência prospectiva é utilizada, por exemplo, para visualização da cadeia de atividades a ser executada por um *workflow* científico. Proveniência retrospectiva se refere às atividades executadas, bem como informações sobre o ambiente usado para a produção dos resultados (Freire *et al,.* 2008). A partir da

base de proveniência retrospectiva, os cientistas podem obter informações sobre a produção, a interpretação e a validação do resultado científico alcançado em um experimento (Silva *et al.,* 2014). Esta tese trata a proveniência retrospectiva, portanto, quando é feita uma referência à proveniência, significa unicamente proveniência retrospectiva.

A Figura 3 ilustra a proveniência prospectiva capturada como a definição de um *workflow* que produz dois tipos de dados: um histograma e uma fotografia. A proveniência retrospectiva (mostrada nas caixas brancas) contém informações sobre os dados de entrada e saída de cada atividade, o usuário que a executa e o início e término da execução (Freire *et al*., 2008).

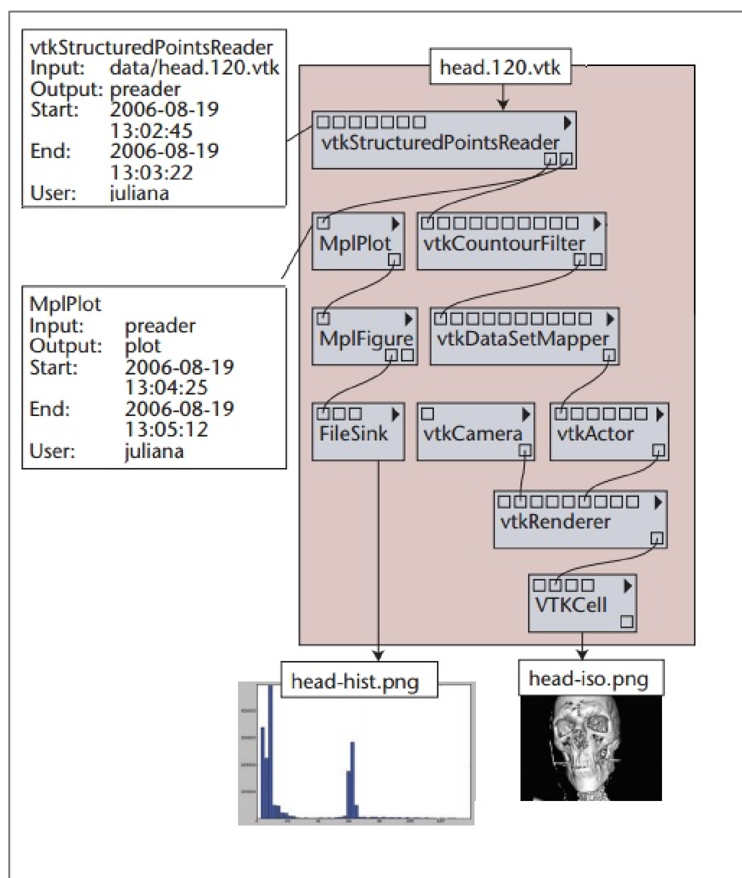


Figura 3. Proveniência prospectiva e retrospectiva (Freire *et al.,* 2008)

Dados de proveniência associados a uma execução de *workflow* formam naturalmente um grafo direcionado e acíclico (DAG). Desta forma, cada execução de um workflow gera um grafo de proveniência, que contém o histórico da derivação dos dados durante cada execução. Após algum tempo, a base de proveniência passa a ser composta por muitos grafos e necessita de espaço de armazenamento considerável. Em cada grafo os vértices representam programas (atividades) ou dados (entidades),

possuem esquemas variáveis e que, muitas vezes, não são previamente conhecidos. As arestas indicam a linhagem ou caminho de derivação dos dados. Consultar a base de proveniência, portanto, implica na manipulação de um grande volume de grafos. Estas características fazem com que a habilidade de armazenar e consultar os dados de proveniência ainda seja um desafio significativo (Aggarwal e Wang, 2010). A Figura 4 ilustra uma base de proveniência, com o destaque de um dos grafos que pertence a esta base.

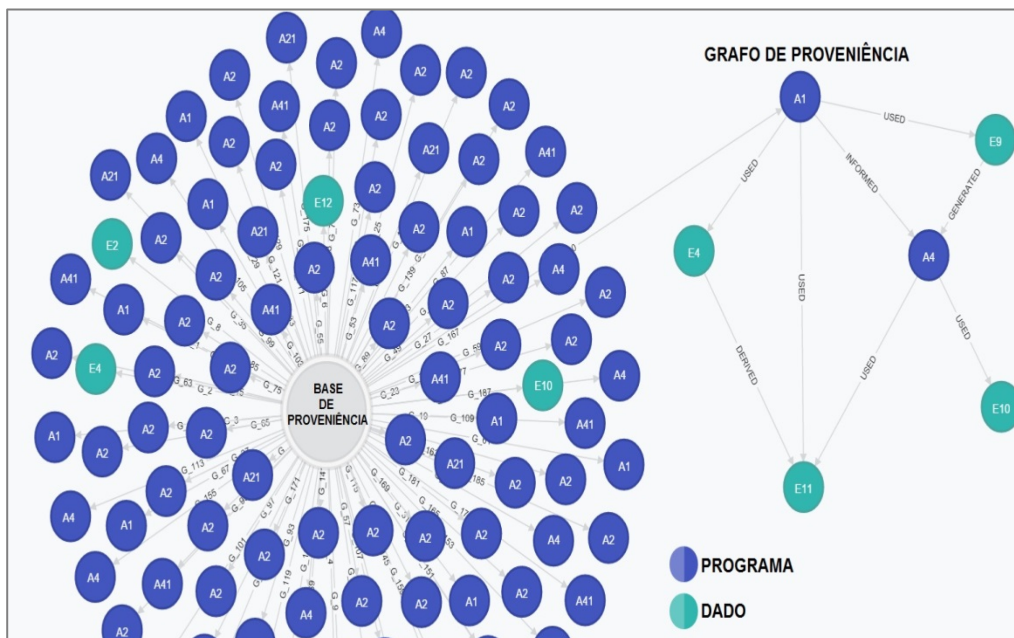


Figura 4. Base de proveniência.

Conforme a importância da proveniência aumenta, se faz necessário um modelo de representação de seus dados que permita a interoperabilidade entre diferentes SGWfC. Para suprir esta necessidade, várias desafios acadêmicos sobre proveniência foram iniciados, sendo o primeiro deles em 2006, durante o primeiro *International Provenance and Annotation Workshop*[11]. O resultado deste esforço foi a criação de um modelo de representação de dados de proveniência chamado de *Open Provenance Model*[12] (OPM). O OPM se encontra em sua versão final (1.1).

O OPM foi projetado para atender aos seguintes requisitos: permitir que as informações sejam trocadas entre diversos sistemas de proveniência, como os SGWfC; permitir aos desenvolvedores construir e compartilhar ferramentas que operam sobre

11 http://www.ipaw.info/
12 http://openprovenance.org/

um modelo de proveniência; definir proveniência de forma precisa; apoiar uma representação digital de proveniência para qualquer fim, produzida por sistemas de computador ou não; possibilitar a coexistência de múltiplos níveis de descrição para uma única execução de um workflow; e definir um conjunto de regras que identificam as inferências válidas que podem ser feitas em uma representação de proveniência (Soares 2013).

O OPM é a base para a definição de um conjunto de especificações, o PROV, proposto pelo *World Wide Web Consortium* (W3C), que se encontra em desenvolvimento. Uma destas especificações, PROV-DM, traz uma proposta de modelo de dados para proveniência. A próxima seção descreve o PROV-DM.

2.2 PROV-DM

O PROV é um conjunto ou família de especificações ligadas aos dados de proveniência que visam especificar aspectos como estrutura, formas de serialização, ontologias, entre outros. Em abril de 2013, o PROV passou a ser uma recomendação do *World Wide Web Consortium* (W3C). Uma visão geral das especificações é dada pelo PROV-Overview[13]. O PROV-DM[14] é o modelo de dados do PROV, sendo a base conceitual para a família de especificações. As estruturas centrais do PROV-DM descrevem o uso e a produção de entidades (dados) por atividades (programas), que podem ser influenciados de várias formas por agentes. Estas estruturas e suas relações são ilustradas pela Figura 5.

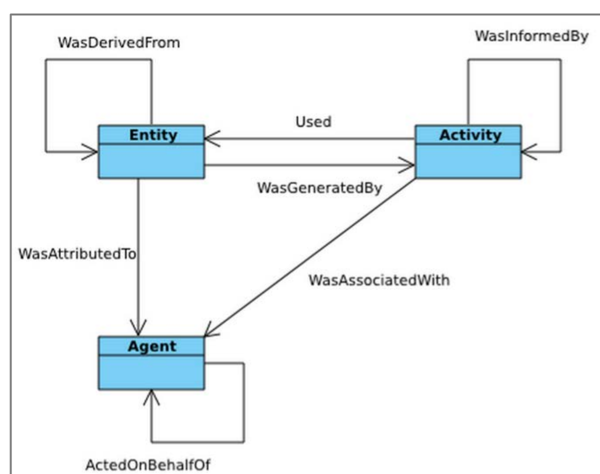


Figura 5. Estruturas centrais do PROV-DM.

[13] http://www.w3.org/TR/prov-overview/
[14] http://www.w3.org/TR/prov-dm/

O PROV-DM define uma entidade como algo físico, digital ou conceitual, podendo ser real ou imaginário. Uma atividade é definida como algo que ocorre durante um período de tempo e age sobre ou com entidades, podendo transformar, consumir, modificar, utilizar, gerar, processar ou realocá-las. Um agente é definido como algo que tem alguma responsabilidade sobre a ocorrência de uma atividade ou a existência de uma entidade.

Atividades e entidades estão associadas de duas formas diferentes: atividades produzem entidades e atividades utilizam entidades. A relação *WasGeneratedBy* se refere à produção de uma nova entidade por uma atividade. Esta entidade não existia antes da sua geração e se torna disponível para utilização após a mesma. A relação *Used* se refere ao começo da utilização de uma entidade por uma atividade.

A geração de uma entidade por uma atividade e a sua subsequente utilização por outra atividade é denominada comunicação (*WasInformedBy*). Desta forma, a relação *WasInformedBy* se refere à troca de alguma entidade, não especificada, por duas atividades, onde uma atividade usa alguma entidade que foi produzida por outra atividade.

A relação *WasDerivedFrom* se refere à transformação de uma entidade em outra nova entidade, à atualização de uma entidade resultando em uma nova entidade, ou à construção de uma nova entidade com base em outra já existente.

Os agentes podem estar relacionados a entidades, atividades e outros agentes. A relação *WasAttributedTo* se refere à atribuição de uma entidade para um agente. A relação *WasAssociatedWith* se refere à atribuição da responsabilidade por uma atividade ao agente, indicando que o agente exerce um papel nesta atividade. A relação *ActedOnBehalfOf* se refere à delegação da autoridade e responsabilidade para um agente (por ele mesmo ou outro agente) para realizar atividade específica, enquanto outro agente tem a responsabilidade sob o resultado do trabalho delegado ao primeiro agente.

As bases de proveniência tratadas nesta tese são compostas por vários grafos representados segundo o PROV-DM. O grafo sumário produzido após a aplicação do SGProv em cada base de proveniência também é representado no mesmo modelo. Desta forma, a sumarização proposta nesta tese tem ciência da semântica do modelo dos grafos utilizados. A nomenclatura usada no resto desta tese é a mesma do modelo PROV-DM, onde programas são referenciados como atividades e dados como

entidades. Esta tese aplica consultas nos grafos originais da base de proveniência produzidos nas execuções de um workflow científico e no grafo sumário resultante. Estas consultas são descritas a seguir.

2.3 Consultas de Proveniência

Na literatura, são encontrados trabalhos que abordam os principais padrões de consultas de proveniência. O trabalho de Woodman *et al.* (2011) propõe o e-Science Central, um sistema que armazena e executa versões atuais e antigas de *workflows,* e que também, armazena os dados de proveniência originados nestas execuções. Aquele trabalho define um conjunto básico de consultas que todo sistema de proveniência deve responder. A consulta mais genérica, considerada a linha de base das consultas de proveniência, retorna todas as dependências diretas e indiretas de um dado. Por exemplo, "*Q1. Encontre todas as dependências diretas e indiretas do dado X*". O retorno de *Q1* é um subgrafo do grafo de proveniência que inclui todos os caminhos que partem de *X,* ou que chegam até ele. Esta consulta pode ter duas variações: a busca por dependências anteriores, como, por exemplo, "*Q2. Encontre todos os dados usados para gerar o dado X*", ou por dependências posteriores, como, por exemplo, "*Q3. Encontre todos os dados que são derivados a partir do dado X*". *Q2* é usada para explicar a presença do dado *X* no resultado obtido. *Q3* é usada para analisar o impacto da influência direta ou indireta do dado *X* no resultado obtido.

O trabalho de Gadelha e Mattoso (2011) propõe oito padrões de consultas de proveniência, que podem ser usados como base para a especificação e desenvolvimento de um sistema de dados de proveniência. Estes padrões são baseados nas consultas propostas na série *Provenance Challenges*, nas consultas de proveniência encontradas na literatura, como as propostas em (Woodman *et al.,* 2011), e nas consultas especificadas em colaboração com os usuários científicos do SGWfC Swift[15]. São eles:

1. Atributo de entidade: consultas que procuram atributos de uma entidade.
2. Relacionamento de um passo: consultas a entidades envolvidas em um relacionamento direto, como, por exemplo, consumo ou produção de um arquivo por um processo.

[15] http://swift-lang.org/main/

3. Relacionamento de múltiplos passos: consultas a entidades envolvidas em um número arbitrário de relacionamentos. É o caso onde a consulta geralmente retorna todas as dependências de dados diretas e indiretas de uma entidade, e, para isto, é necessário executar uma operação de travessia nos grafos de proveniência.
4. Correspondência entre grafos de proveniência: consultas para determinar similaridade entre grafos de linhagem. Elas podem incluir uma combinação dos outros três padrões citados acima.
5. Executar resumos: consultas que retornam atributos específicos de uma execução (por exemplo, memória utilizada) ou das entidades que o *workflow* contém (processos e seus respectivos conjuntos de dados de entrada e saída).
6. Executar comparações: consultas que comparam múltiplas execuções de *workflows* (vários grafos de proveniência) em relação a algum atributo para analisar como o mesmo variou entre elas.
7. Executar correlações: consultas que correlacionam atributos de várias execuções de *workflows* (vários grafos de proveniência) diferentes.
8. Resumo de execuções do mesmo workflow: consultas que agregam dados de um conjunto de execuções de um mesmo workflow. Por exemplo, consulta que retorna a média da precisão de um modelo computacional após um determinado número de execuções.

Um conjunto de consultas de proveniência foi definido nesta tese para a comparação dos seus tempos de processamentos quando aplicadas nos vários grafos da base de proveniência e no grafo sumário produzido pelo SGProv. Este conjunto de consultas aborda a maioria dos padrões citados nesta seção e são descritas na seção 7.1.

Na literatura, são encontradas várias formas de armazenar e consultar os dados de proveniência. Estas formas são descritas e analisadas no capítulo a seguir.

Capítulo 3 Armazenamento de Dados de Proveniência

Este capítulo discute as formas de armazenamento e de consulta de grafos de proveniência encontradas na literatura e apresenta uma comparação entre o uso de SGBD relacionais, que é a forma mais comumente utilizada pelos trabalhos relacionados, e o uso de SGBD de grafos, que é a forma utilizada nesta tese.

Existem basicamente quatro abordagens para armazenar e consultar os grafos de proveniência: SGBD relacional; SGBD de grafos; Híbrida: linguagem de consulta para grafos com armazenamento relacional; e SGBD XML. Estas abordagens são discutidas a seguir.

3.1 SGBD Relacional

A abordagem mais comum para tratar o problema de armazenamento e consulta de dados de proveniência é utilizar um SGBD relacional como em (Huahai e Singh, 2008) e (Anand *et al.,* 2012). Um grafo de proveniência, geralmente, é representado em um SGBD relacional através de um esquema que contém duas relações: uma para vértices e outra para arestas. As consultas são feitas usando SQL. Entretanto, para recuperar os caminhos entre os vértices é preciso fazer junções entre relações repetidas vezes, o que pode comprometer o tempo de processamento das consultas.

Consultas de maior complexidade devem ser implementadas através de procedimentos armazenados no SGBD ou externamente, com algoritmos desenvolvidos em linguagens convencionais e acessando o banco de dados, o que pode ser um problema para usuários não familiarizados com programação. Como as bases de dados de proveniência são muito volumosas e, geralmente, são formadas por uma grande quantidade de vértices e arestas, o tempo de processamento das consultas pode ficar ainda mais comprometido, como é descrito em Soares (2013). Além disso, os SGBD relacionais são baseados em esquemas de dados predefinidos e não flexíveis. Entretanto, dados de proveniência possuem estruturas bem flexíveis, muitas vezes, não conhecidas *a priori* e necessitam de mudanças constantes (Holland *et al.*, 2008).

3.2 SGBD de Grafos

Um SGBD de grafo representa os dados armazenados diretamente como um grafo. Para fazer consultas aos dados de um grafo é necessário percorrer seus caminhos. Os SGBD de grafos utilizam grafos com vértices, arestas e propriedades (atributos de vértices e arestas) para representar e armazenar informações. Os vértices e arestas possuem estruturas flexíveis, ou seja, podem possuir atributos arbitrários, não possuindo esquema predefinido e podendo ser alterados facilmente. Para percorrer os caminhos dos grafos utilizam algoritmos clássicos da literatura, como busca em largura *(Breadth-First Search)*, busca em profundidade (*Depth-First Search*) e busca do menor caminho entre dois vértices (*e.g*. algoritmo de *Djikstra)*.

A partir da análise dos trabalhos descritos em Vicknair *et al.* (2010) e Soares (2013), Woodman *et al.* (2011) e Cuervas-Vicenttín *et al.* (2014) esta tese parte da premissa de que utilizar SGBD de grafos para armazenar os grafos de proveniência e de uma linguagem de consulta para grafos para realizar consultas de proveniência aos grafos armazenados é a melhor alternativa. O SGBD de grafos utilizado nesta tese é o Neo4j juntamente com sua linguagem de consulta Cypher, que são descritos a seguir.

3.2.1 Neo4j

O Neo4j é um SGBD orientado a grafos desenvolvido principalmente em Java, sendo de código aberto, com documentação e suporte acessíveis. Ele permite a criação de grafos com elementos (vértices e arestas) sem esquemas de dados predefinidos. É indicado para aplicações que tenham como característica a exploração e a descoberta de caminhos complexos entre seus dados, que são geralmente vistos como uma rede, como acontece com os grafos de proveniência. Os dados são armazenados em vértices, que contêm propriedades (atributos). Como em qualquer grafo, os vértices são interligados por arestas, que também possuem propriedades, formando estruturas complexas. Ao contrário dos grafos clássicos, porém, o Neo4j suporta a criação de grafos multi-relacionais (Rodriguez e Neubauer 2011). Tais grafos são caracterizados por possuírem arestas de diferentes tipos, cada uma representando uma forma de relacionamento diferente entre os vértices. Desta forma, entre um mesmo par de vértices podem existir várias arestas, sendo uma de cada tipo.

Para consultar os dados armazenados o Neo4j disponibiliza um conjunto de funcionalidades (*The Traversal Framework*) para navegar no grafo e identificar caminhos de acordo com a ordem dos vértices. Índices podem ser criados sobre as propriedades dos vértices e arestas, com o objetivo de facilitar consultas de seleção. Uma limitação do Neo4j, e da maioria dos SGBD de grafos, está no fato de ele ser projetado para gerenciar um único grafo por base de dados, mesmo que este seja bastante extenso. A manipulação de um conjunto de grafos, como é o caso do problema abordado nesta tese, exige artifícios por parte do usuário do SGBD.

3.2.2 **Cypher**

Cypher é uma linguagem declarativa de consulta a grafos desenvolvida pelos mesmos criadores do Neo4j. Desta forma, esta linguagem expressa *o que* se quer obter do grafo e não como fazê-lo. A linguagem Cypher é baseada na estrutura do SQL. As cláusulas básicas utilizadas pela consulta são:

- *Match*: padrão baseado em expressões regulares, que descreve subgrafos que se pretende encontrar no grafo armazenado na base de dados.
- *Where*: restrições ou filtros a serem aplicados sobre os subgrafos que correspondem ao padrão.
- *Return*: especifica os dados a serem retornados pela consulta.

Por exemplo, considerando o grafo a seguir, que representa pessoas e suas relações de amizade.

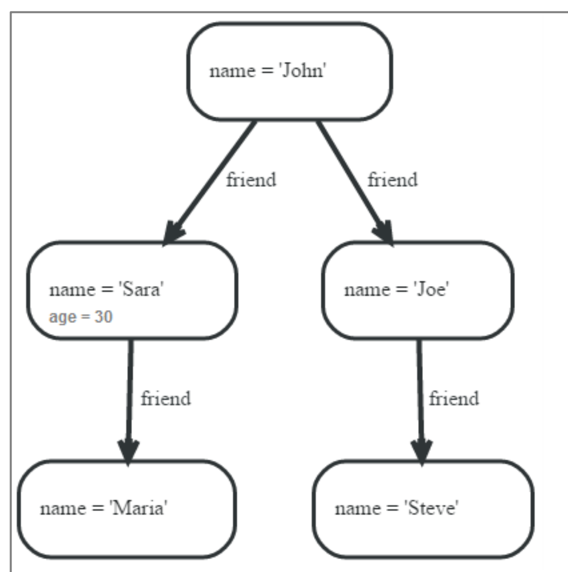


Figura 6. Grafo de pessoas e seus amigos.

A consulta “*encontre os amigos diretos de John*”, teria como resultado “*Sara*” e “*Joe*” e seria escrita da seguinte forma em Cypher.

```
MATCH (n {name: 'John'})-[:friend]->(f)
RETURN n.name, f.name
```

Na consulta *n* e *f* são identificadores dados aos vértices dos subgrafos que correspondem ao padrão. O vértice *n* deve ter o atributo “*name = John”* e a aresta entre os dois vértices deve ser do tipo “*friend*”. Os vértices devem ser escritos entre parênteses e as arestas entre colchetes. As arestas do grafo devem se especificadas entre os símbolos “–“ e “->”.

Se consulta fosse, “*encontre os amigos diretos de John que têm idade igual a 30 anos*”, teria como resultado “*Sara*” e seria escrita da seguinte forma em Cypher.

```
MATCH (n {name: 'John'})-[:friend]->(f)
WHERE f.age = 30
RETURN n.name, f.name
```

Esta consulta é igual a anterior, mas na cláusula *WHERE* é definida uma restrição sobre o valor do atributo “*age*” do vértice que representa o amigo de John.

A consulta “*encontre todos os amigos dos amigos de John*”, teria como resultado “*Maria*” e “*Steve*” e seria escrita da seguinte forma em Cypher.

```
MATCH (n {name: 'John'})-[:friend]->()-[:friend]->(f)
RETURN n.name, f.name
```

Esta consulta, a partir do vértice com *“name = John”,* percorre duas arestas do tipo *friend* para encontrar os amigos dos amigos de John. Por isso, na consulta em Cypher a aresta *friend* é escrita duas vezes.

As consultas em Cypher podem ser executadas através interface Web do Neo4j, que também permite a visualização dos grafos armazenados, ou através da API Java do Neo4j, que é a alternativa utilizada nesta tese. Há API para outras linguagens, que não são utilizadas nesta tese.

3.3 Abordagem Híbrida

Outra abordagem encontrada na literatura para tratar o armazenamento e consulta de grafos é o uso de uma linguagem de consulta adequada para os grafos

mapeada sobre SGBD relacionais ou outros modelos de dados. Como exemplos desta abordagem híbrida é possível citar a Lorel (Abiteboul *et al.,* 1996), voltada para XML, a SPARQL (Harting e Heese, 2007), voltada para grafos RDF, e algumas voltadas para grafos de proveniência, como a PQL (Holland *et al.,* 2008), que foi desenvolvida a partir da Lorel, a QLP (Anand *et al.,* 2009; Anand *et al.* 2010), a OPQL (Huahai e Singh, 2008) e a ProQL (Karvounarakis *et al.,* 2010).

As consultas aos grafos de proveniência são feitas nas linguagens propostas, mas internamente são convertidas e executadas em SQL. Esta abordagem híbrida gera uma sobrecarga grande na construção destes mapeamentos grafo-relações-grafo e as relações resultantes da consulta relacional, por sua vez, precisam ser mapeadas para um grafo que seja compreensível pelo usuário, mantendo a harmonia com a linguagem em que a consulta foi especificada.

3.4 SGBD XML

Ao contrário do modelo relacional, o modelo XML apresenta relacionamentos por meio de caminhos em grafos. XQuery é a linguagem usada para fazer consultas ao documento XML. Embora a estrutura de um documento XML possa ser representada como um grafo ou uma árvore, a navegação do processamento de consultas da XQuery assume um conjunto de árvores a ser percorrido. Uma consulta XQuery começa na raiz de um documento XML, percorre seus elementos e termina nas folhas. Assim, o XML utilizado para consultas XQuery não representa os dados de proveniência de forma natural, porque os dados de proveniência formam um grafo mais geral do que uma árvore, de acordo com Holland *et al.* (2008).

O trabalho desenvolvido em Holland *et al.* (2008) teve como objetivo adaptar documentos XML e linguagens de consultas XML para tratar grafos de proveniência. Entretanto, o trabalho concluiu que, embora seja possível fazê-lo, foi necessário criar uma linguagem muito diferente da XQuery para realizar consultas complexas aos dados de proveniência. Um dos problemas identificados foi o de que, na XQuery, os caminhos são compostos pelos nomes dos objetos. Entretanto, para a especificação de consultas sobre os dados de proveniência, os caminhos devem ser compostos pelos nomes dos relacionamentos entre objetos e os objetos devem ser identificados por seus atributos e posição no grafo. Isso acontece porque os objetos normalmente são designados por códigos de difícil leitura. Uma alternativa é armazenar os dados de proveniência em um

repositório XML e usar referências cruzadas do XML para criar os grafos. Para que seja possível visitar todos os vértices do grafo usando a XQuery é necessário programar funções que o façam. Isso pode ser um problema para usuários não familiarizados com programação.

Conforme analisado, utilizar um SGBD relacional, a abordagem híbrida e utilizar SGBD XML são soluções que levam muito tempo para analisar um grande volume de dados em uma consulta analítica. Dentre todas as abordagens analisadas, utilizar um SGBD de grafos e uma linguagem de consulta para grafos parece ser a melhor alternativa. A seguir é feita uma comparação do uso de SGBD relacionais, que é a alternativa mais utilizada encontrada na literatura, e o uso de SGBD de grafos, que é a alternativa utilizada nesta tese.

3.5 Comparação entre SGBD Relacional e SGBD de Grafo

Os SGBD relacionais têm sido estudados por muitas décadas e, atualmente, constituem a forma de gerenciamento de dados mais utilizada. Estes sistemas, geralmente, são muito eficientes, mas devem ser analisados com cuidado no que diz respeito ao armazenamento de grafos. Neste caso, as consultas geram a necessidade de execução de muitas junções recursivas entre relações para que seja possível percorrer os caminhos dos grafos, comprometendo o tempo de processamento das consultas. O trabalho de Vicknair *et al.* (2010) faz uma comparação entre o desempenho do SGBD relacional MySQL e o SGBD de grafos Neo4j no tempo de processamento das consultas a grafos de proveniência. A Tabela 1 mostra os tamanhos das bases de dados utilizadas no trabalho.

Tabela 1. Tamanhos das bases de dados (Vicknair *et al.* 2010).

Base de Dados	Vértices	Tipo de Dados	Tamanho MySQL	Tamanho Neo4j
1000int	1000	Int	0.232M	0.428M
5000int	5000	Int	0.828M	1.7M
10000int	10000	Int	1.6M	3.2M
100000int	100000	Int	15M	31M
1000char8k	1000	8K Char	18M	33M
5000char8k	5000	8K Char	87M	146M
10000char8k	10000	8K Char	173M	292M
100000char8k	100000	8K Char	1700M	2900M
1000char32k	1000	32K Char	70M	85M
5000char32k	5000	32K Char	504M	406M
10000char32k	10000	32K Char	778M	810M
100000char32k	100000	32K Char	6200M	7900M

Como mostra a Tabela 1, foram criadas doze bases de dados e doze grafos correspondentes. Cinco destas bases armazenam dados do tipo primitivo inteiro e as demais do tipo caractere.

As consultas foram divididas em dois tipos: estruturais (consultam a estrutura do grafo) e aos dados (consultam o conteúdo de propriedades específicas armazenadas nos vértices). Estas consultas são descritas a seguir.

3.5.1 Consultas Estruturais

As consultas que percorrem as estruturas dos grafos foram as seguintes.

E1: Encontre todos os vértices órfãos, ou seja, aqueles que não possuem arestas para outros vértices.

E2: Navegue no grafo até a profundidade 4 e conte o número de vértices.

E3: Navegue no grafo até a profundidade 128 e conte o número de vértices.

A Tabela 2 apresenta os resultados das consultas.

Tabela 2. Resultados das consultas estruturais, em milissegundos (Vicknair *et al.,* 2010).

Base de Dados	MySQL E2	Neo4j E2	MySQL E3	Neo4j E3	MySQL E1	Neo4j E1
1000int	38.9	2.8	80.4	15.5	1.5	9.6
5000int	14.3	1.4	97.3	30.5	7.4	10.6
10000int	10.5	0.5	75.5	12.5	14.8	23.5
100000int	6.8	2.4	69.8	18.0	187.1	161.8
1000char8K	1.1	0.1	21.4	1.3	1.1	1.1
5000char8K	1.0	0.1	34.8	1.9	7.6	7.5
10000char8K	1.1	0.6	37.4	4.3	14.9	14.6
100000char8K	1.1	6.5	40.9	13.5	187.1	146.8
1000char32K	1.0	0.1	12.5	0.5	1.3	1.0
5000char32K	2.1	0.5	29.0	1.6	7.6	7.5
10000char32K	1.1	0.8	38.1	2.5	15.1	15.5
100000char32K	6.8	4.4	39.8	8.1	183.4	170.0

As consultas estruturais **E2** e **E3** executadas no Neo4j foram bem mais rápidas do que as executadas no MySQL. Tal comportamento é observado porque os SGBD relacionais não foram desenvolvidos para navegar em grafos. A consulta **E1** obteve tempos que foram, na maioria das vezes, menores no MySQL, pois o Neo4j percorre os grafos com base nos seus caminhos e esta consulta procura por vértices que não possuem arestas.

3.5.2 Consultas aos Dados

As consultas **D1** e **D2** foram aplicadas às bases de inteiros e a **D3** às bases de caracteres.

D1: Conte o número de vértices cujo atributo "*x*" é igual a um determinado valor.

D2: Conte o número de vértices cujo atributo "*x*" é menor do que um determinado valor.

D3: Conte o número de vértices cujo atributo "*x*" contém uma determinada cadeia de caracteres (com comprimento entre 4 e 8 caracteres).

A Tabela 3 apresenta os resultados das consultas **D1** e **D2**.

Tabela 3. Resultados das consultas D1 e D2, em milissegundos (Vicknair *et al.*, 2010).

Base de Dados	Rel D1	Neo D1	Rel D2	Neo D2
1000int	0.3	33.0	0.0	40.6
5000int	0.4	24.8	0.4	27.5
10000int	0.8	33.1	0.6	34.8
100000int	4.6	33.1	7.0	43.9

As consultas **D1** e **D2** foram bem mais rápidas no MySQL. Isso porque, o Neo4j trata todos os atributos como texto e as consultas procuram por valores inteiros e a conversão é demorada.

A consulta **D3**, que procura por uma cadeia de caracteres no valor do atributo, apresenta valores interessantes, que são apresentados na Tabela 4.

Tabela 4. Resultado da consulta D3, em milissegundos, onde d é o tamanho da cadeia de caracteres procurada (Vicknair *et al.*, 2010).

Base de Dados	Rel	Neo	Rel	Neo	Rel	Neo	Rel	Neo	Rel	Neo
	d=4	d=4	d=5	d=5	d=6	d=6	d=7	d=7	d=8	d=8
1000char8k	26.6	35.3	15.0	41.6	6.4	41.6	11.1	41.6	15.6	36.3
5000char8k	135.4	41.6	131.6	41.8	112.5	36.5	126.0	33.0	91.9	41.6
10000char8k	301.6	38.4	269.0	41.5	257.8	41.5	263.1	42.6	249.9	41.5
100000char8k	3132.4	41.5	3224.1	41.5	3099.1	42.6	3077.4	41.8	2834.4	36.4
1000char32k	59.5	41.5	41.6	42.6	30.9	41.5	31.9	41.4	31.9	35.4
5000char32k	253.4	42.3	242.9	41.5	229.4	35.3	188.5	38.5	152.0	41.5
10000char32k	458.4	36.3	468.8	41.6	468.3	41.6	382.1	41.5	298.8	36.3
100000char32k	3911.3	41.4	4859.1	33.3	6234.8	37.3	4703.3	41.5	2769.6	41.5

O tempo de processamento da consulta **D3** foi muito menor no Neo4j do que no MySQL. Com 1000 vértices, os resultados foram parecidos e o MySQL obteve

melhores resultados, mas conforme o número de vértices aumentou, o Neo4j foi muitas vezes mais rápido.

3.5.3 Considerações Sobre a Comparação entre SGBDs

No exemplo apresentado, os dois sistemas apresentaram bom desempenho. O Neo4j teve um tempo de processamento muito melhor do que o MySQL nas consultas estruturais e nas que procuraram cadeias de caracteres nos conteúdos dos dados, principalmente conforme aumentava a quantidade de vértices nos grafos. O pior tempo de processamento do Neo4j foi obtido nas consultas a valores inteiros nos conteúdos de dados, devido à necessidade de conversão de inteiros para caracteres, que são os tipos de dados dos atributos no Neo4j.

Como as consultas de proveniência são, geralmente, estruturais, utilizar o Neo4j parece ser mais adequado do que um SGBD relacional. Entretanto, mesmo utilizando SGBD de grafos para armazenar os dados de proveniência, persiste o problema de consultar o grande volume de dados de proveniência produzidos em um experimento científico computacional. Por exemplo, analisando os dados da Tabela 2 é possível observar que, nas consultas realizadas no Neo4j, os melhores tempos de processamento foram obtidos nos grafos com menor número de vértices, independente do tamanho dos dados armazenados nestes vértices. Isto indica que, mesmo utilizando um SGBD de grafos, manipular um conjunto com muitos grafos de proveniência pode comprometer o tempo de processamento das consultas de proveniência.

Aplicando uma técnica de compactação de grafos nos vários grafos de proveniência gerados em um experimento, é possível reduzir o volume de dados e, assim, tornar as consultas mais simples e reduzir seus tempos de processamento. A compactação deve ser feita sem perda de dados, para que os cientistas possam validar seus experimentos sem serem induzidos a erros. Para que seja possível responder a consultas de proveniência, o grafo compactado deve preservar a topologia dos grafos originais. Para tornar as consultas mais simples e reduzir seu tempo de processamento, elas devem ser feitas utilizando unicamente o grafo compactado, sem que seja necessário consultar os diversos grafos gerados, ou desfazer a compactação. As técnicas de compactação de grafos encontradas na literatura são discutidas no próximo capítulo.

Capítulo 4 Técnicas de Compactação de Grafos

Este capítulo descreve e analisa trabalhos relacionados às técnicas de compactação de grafos (sumarização, compressão e fatoração) a fim de identificar como estas técnicas são utilizadas e qual delas pode ser aplicada nesta tese para a compactação de grafos de proveniência com o objetivo de melhorar o tempo de processamentos das consultas.

4.1 Sumarização de Grafos

A sumarização consiste em agrupar dados de acordo com uma ou mais caraterísticas em comum. O problema da sumarização de grafos tem sido estudado em várias áreas de pesquisa com o objetivo de reduzir seu volume, a fim de que possam ser armazenados em memória principal, tornando possível a aplicação dos algoritmos clássicos encontrados na literatura. Mesmo que a aplicação não seja limitada pela memória, a representação sumarizada fornece mais eficiência para as consultas aos grafos. Entretanto, a maior parte dos estudos são relacionados à sumarização de grafos Web, com o objetivo de otimizar o espaço dos relacionamentos entre as várias páginas existentes (Aggarwal e Wang, 2010).

A sumarização pode ser feita com ou sem perdas em relação ao grafo original, dependendo das necessidades da aplicação. Para algumas aplicações, pode não ser necessário recriar exatamente o grafo original, sendo suficiente uma reconstrução aproximada. Em relação aos grafos de proveniência, a sumarização deve ser feita sem perda de dados, para que os cientistas possam validar seus experimentos sem a indução de erros. Alguns trabalhos são propostos na literatura com o objetivo de gerar representações sumarizadas de grafos de vários domínios de aplicação. Estes trabalhos são apresentados a seguir.

4.1.1 Sumarização de Grafo com Erro Limitado

O objetivo do trabalho de (Navlakha *et al.,* 2008) é construir um grafo sumarizado bem menor e mais simples de ser visualizado do que o original. A representação sumarizada de um grafo é composta por duas partes: um grafo sumário e uma lista de arestas de correção.

O grafo sumário representa a estrutura do grafo original em alto nível. No sumário, os vértices do grafo são agrupados em supervértices e suas arestas são agrupadas em superarestas. Para cada supervértice criado é mantida uma lista de vértices do grafo original correspondentes a ele. O agrupamento é feito de acordo com a teoria clássica de grafos, por exemplo: procura sumarizar áreas densas do grafo; se existe um subgrafo completo e bipartido, então os dois núcleos podem ser agrupados em dois supervértices e as arestas substituídas por uma superaresta entre os supervértices; e um clique pode ser agrupado em um único supervértice com um auto-relacionamento.

Para o agrupamento também são consideradas as similaridades das estruturas dos vértices e das arestas. Por exemplo, são agrupados em um supervértice os vértices que possuem arestas para um mesmo conjunto de vértices. Este conjunto de vértices também é agrupado em outro supervértice. As arestas existentes entre os dois supervértices criados são grupadas em uma única superaresta.

O grafo original pode ser recuperado expandindo o grafo sumário. Entretanto, pode ser que nem todas as arestas criadas no sumário existam no grafo original e vice-versa. Uma lista de correção contém as arestas que precisam ser aplicadas ao grafo sumário de modo a possibilitar a reconstrução do grafo original. As arestas que existem no grafo original contêm o sinal soma (+) e devem ser adicionadas ao sumário, já as que não existem contêm o sinal subtração (–) e devem ser excluídas do mesmo. Um exemplo da técnica de sumarização proposta em (Navlakha *et al.,* 2008) é apresentado a seguir.

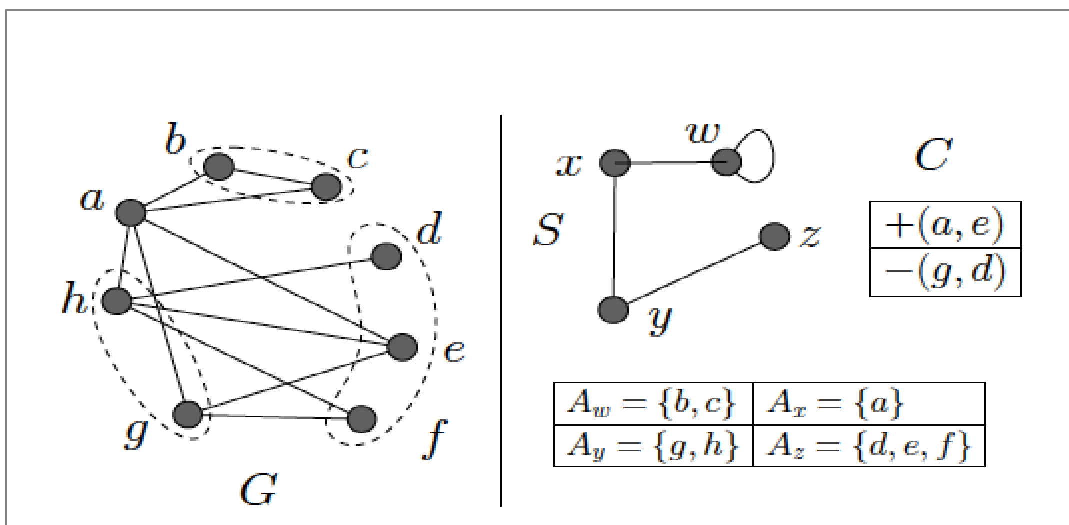


Figura 7. Grafo original (G), sumário (S) e lista de arestas de correção (C) (Navlakha *et al.,* 2008).

A Figura 7 apresenta o grafo original (*G*). Os vértices envolvidos por linhas pontilhadas são agrupados em supervértices para formar o grafo sumário (*S*). Para cada supervértice de (*S*) é mantida uma lista dos vértices correspondentes em *G* (A_w, A_y, A_x e A_z). A lista de arestas de correção é representada por *C*.

A reconstrução do grafo original (*G*) é feita a partir do grafo sumário (*S*). O exemplo a seguir mostra a reconstrução do vértice ***g*** e seus vizinhos.

1º) Achar o supervértice que contém ***g*** = ***y***.

2º) Criar arestas de ***g*** para todos os vértices das listas dos supervértices dos vizinhos de ***y***. Os vizinhos de ***y*** são ***x*** *{a}* e ***z*** *{d, e, f}*. Assim, são criadas as arestas *{(g, a), (g, d), (g, e), (g, f)}* (Figura 8).

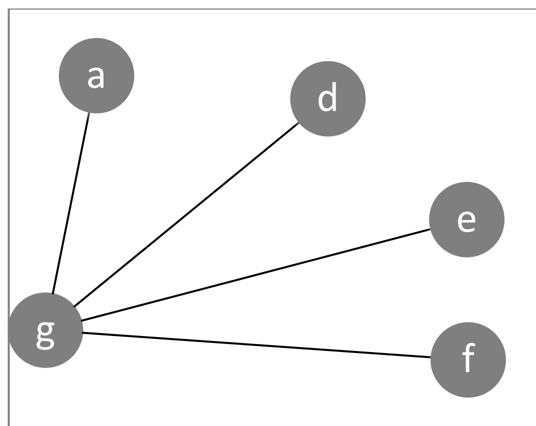


Figura 8. Expansão das arestas de y.

3º) Aplicar a lista C: incluir +(a, e) e excluir –(g, d) (Figura 9).

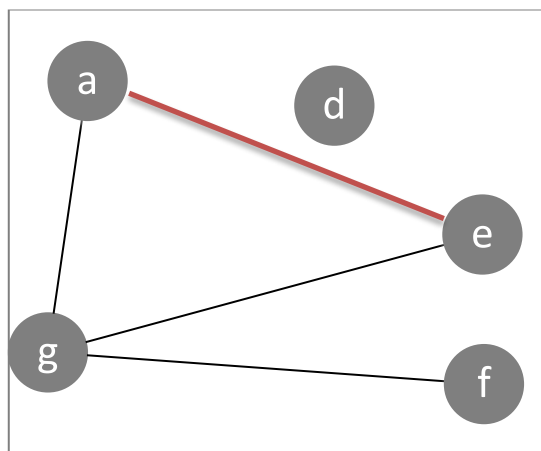


Figura 9. Aplicando lista de arestas de correção.

O grafo sumário pode ser construído sem ou com perdas, caso para o usuário seja aceitável certo grau de erro na recuperação dos dados originais, com o objetivo de melhorar a compactação em detrimento da precisão.

Esta abordagem de sumarização é aplicada a grafos genéricos, onde a estrutura dos dados armazenados nos vértices não é conhecida. Desta forma, a sumarização é feita com base na estrutura (arestas) dos grafos e em conceitos da teoria clássica de grafos (cliques, áreas densas, etc.). Embora reduza o volume do grafo original, pode não ser uma boa técnica de sumarização para melhorar o tempo de processamento das consultas. Para realizar as consultas é preciso reconstruir o grafo original com base na lista de arestas de correção, pois são inseridas no grafo sumário arestas que não existem no grafo original e são removidas do grafo sumário arestas que existem no grafo original.

4.1.2 **Sumarização Iterativa de Grafos**

O objetivo do trabalho de (Yu *et al.,* 2010) é construir um grafo compactado a partir de um grafo original com tamanho mais reduzido, mas que forneça os dados necessários para a aplicação. No grafo sumário cada vértice representa um conjunto de vértices do grafo original e cada aresta representa as conexões entre dois conjuntos de vértices. As técnicas propostas no trabalho contemplam grafos não direcionados, mas podem ser estendidas para grafos direcionados.

Cada vértice do grafo sumário é chamado de grupo ou supervértice e corresponde a uma parte do conjunto original de vértices. Cada aresta do sumário é chamada grupo de arestas ou superarestas e representa as conexões entre dois supervértices. Uma superaresta só existe se cada vértice de um supervértice tiver pelo menos uma aresta com um vértice do outro supervértice.

A Figura 10.a mostra um exemplo da técnica de sumarização proposta em *(*Yu *et al.,* 2010) para um grafo de redes sociais, onde os vértices representam estudantes e existem dois tipos de arestas: amigos e colegas de turma. Cada vértice possui um conjunto de atributos (gênero, departamento, etc.). Alguns estudantes são só amigos ou só colegas de turma, outros são as duas coisas ao mesmo tempo. O grafo sumário correspondente é apresentado na Figura 10.b. Este grafo contém quatro grupos de estudantes e as arestas entre os mesmos. Em cada supervértice, os estudantes possuem o mesmo conjunto de atributos com os mesmos valores. Por exemplo, possuem o mesmo gênero e estão no mesmo departamento. Os supervértices estão relacionados a outros supervértices através de superarestas (amigos, colegas de turma, ou as duas coisas). O sumário possui informações sobre os vértices e arestas do grafo original.

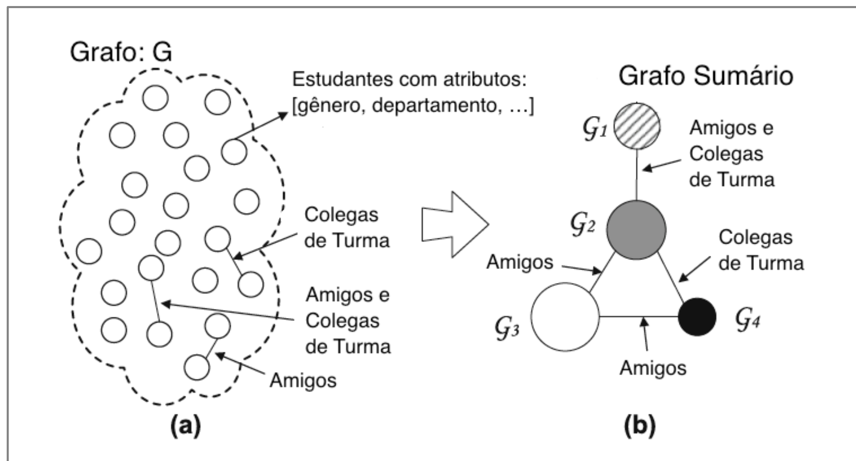


Figura 10. **Grafo original (a) e Grafo Sumário (b) (Yu *et al.,* 2010).**

A técnica de sumarização proposta em (Yu *et al*., 2010) é baseada na agregação de componentes de um grafo através de dois algoritmos. O primeiro é o SNAP (*Summarization by Grouping Nodes on Attribute and Pairwise Relationships*), que produz um grafo sumário a partir do grafo original agrupando vértices, com base nos seus atributos e arestas. O grafo sumário produzido pelo SNAP deve satisfazer três requisitos:

- Homogeneidade de atributo: Todos os vértices de um supervértice devem ter os mesmos atributos e estes devem possuir os mesmos valores.
- Homogeneidade de aresta: Para os pares de supervértices que possuem uma superaresta entre eles, cada vértice de um supervértice deve ter pelo menos uma aresta com um vértice do outro supervértice. Por exemplo, na Figura 10.b cada estudante (vértice) do supervértice *G2* é amigo de pelo menos um estudante do supervértice *G3*, colega de turma de algum estudante de *G4* e tem pelo menos um amigo e um colega de turma em *G1*.
- Compactação máxima: Pode existir mais de uma forma de agrupamento de vértices e arestas que satisfaçam os requisitos anteriores. Entretanto, o grafo sumário deve ter o tamanho menor possível, dentro dos requisitos de homogeneidade.

O algoritmo SNAP produz o agrupamento de vértices com base nos seus atributos e arestas. Entretanto, pode produzir um grande número de pequenos supervértices, ou até um vértice que não pertença a nenhum supervértice. O grafo sumário produzido pode, portanto, ser tão volumoso quanto o grafo original comprometendo o tempo de processamento das consultas. A alternativa usada é deixar que os usuários controlem o tamanho do sumário, escolhendo o nível de detalhes

desejado. Esta é a abordagem utilizada pelo segundo algoritmo, k-SNAP. Este permite que o usuário escolha o nível de detalhe do grafo sumário através do uso de técnicas OLAP (*On-line Transactional Processing*) para aumentar o nível de detalhe (*drill-down*) ou reduzir o nível de detalhe (*roll-up*).

O algoritmo k-SNAP relaxa o requisito de homogeneidade para as arestas, permitindo que nem todos os vértices de um supervértice tenham arestas com vértices de outro supervértice vizinho para que haja uma superaresta entre eles. Entretanto, a homogeneidade para atributo é mantida. Os usuários podem controlar o número de supervértices do sumário especificando um valor para o parâmetro *k*. É possível que existam várias maneiras de agrupar os vértices no tamanho *k*, mas nem todas estas maneiras podem ter a qualidade desejada. O objetivo do k-SNAP é encontrar um grafo sumário, de tamanho *k* e com a melhor qualidade possível. Para isso, é feita a medição da qualidade do sumário gerado, analisando o quanto ele é diferente do sumário "ideal", gerado pelo SNAP.

4.1.3 Sumarização de Grafo Orientada a Descoberta

O trabalho de (Zhang *et al.,* 2010) propõe soluções para duas limitações do algoritmo k-SNAP (item 4.1.2). A primeira se refere à falta de eficiência do k-SNAP quando os vértices de um grafo possuem atributos com muitos valores distintos. Assim, a restrição de homogeneidade de atributo pode produzir um sumário com tantos supervértices quanto o grafo original. A segunda se refere à falta de um mecanismo automático que produza um pequeno conjunto de sumários relevantes para o usuário. Isso ocorre porque é possível que existam vários agrupamentos diferentes quando se relaxa o requisito de homogeneidade de aresta. Portanto, em cada execução do algoritmo um sumário diferente pode ser produzido. Cabe ao usuário escolher, entre as soluções, a mais relevante.

Como solução para a primeira limitação, o trabalho de (Zhang *et al.,* 2010) propõe uma categorização dos atributos, baseada em seus valores e nas arestas dos vértices do grafo, para reduzir o número de supervértices no grafo sumário. Entretanto obter uma divisão de categorias adequadas não é uma tarefa trivial. Uma solução poderia ser usar o conhecimento do domínio de especialistas para criar estas categorias, mas na prática isso não é sempre possível, especialmente se os dados mudam frequentemente, como no caso de aplicações de redes sociais. Uma alternativa mais

adequada é ter uma forma automática de fazê-lo, explorando as similaridades dos valores dos atributos e das arestas entre os vértices do grafo. A Figura 11 mostra um exemplo de categorização de atributos com base em um grafo complexo onde os vértices são artigos que possuem o atributo “*número de citações*”.

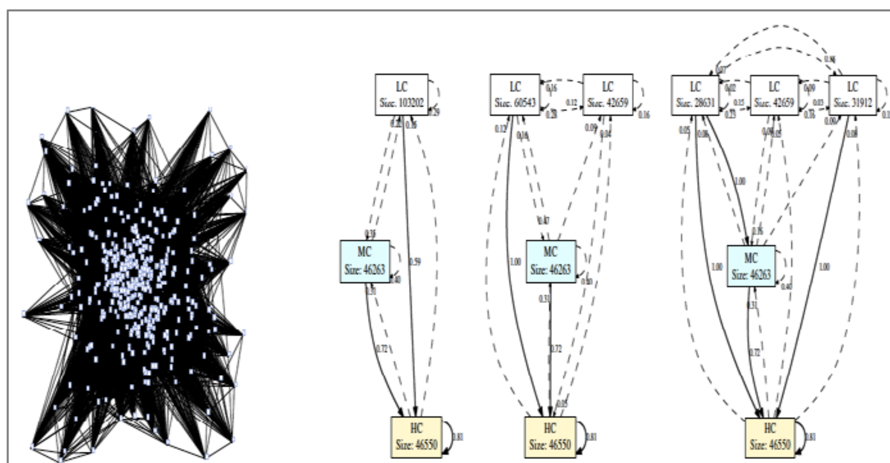


Figura 11. Grafo original de artigos e três exemplos de sumários com categorização de atributos (Zhang *et al*., 2010).

Na figura, o grafo possui 196.015 artigos, conectados por 823.432 arestas e o atributo citado possui 423 valores distintos. Assim, o grafo sumário terá ao menos 423 supervértices. A categorização proposta divide os atributos em três grupos: LC (número de citações<=3), MC (4<= número de citações <=8) e HC (número de citações >=9). Com a categorização, o grafo sumário resultante é mais compacto e mais simples facilitando o entendimento por parte dos usuários. Os vértices no sumário podem ser agrupados pelas categorias dos seus atributos e também pelas arestas entre seus vértices. Assim, vários sumários podem ser gerados. Por exemplo, a Figura 11 mostra três tipos de sumário que podem ser gerados com 3, 4 e 5 supervértices. As conexões em negrito indicam superarestas fortes (mais de 50% de participação dos vértices) e aquelas em pontilhado indicam superarestas fracas.

A solução proposta para a segunda limitação é uma medida para avaliar a relevância de uma sumarização. O objetivo é medir a relevância das sumarizações possíveis e automaticamente identificar um pequeno conjunto de grafos sumários que possa ser útil para o usuário. Esta medida considera somente superarestas fortes, pois estas representam um padrão de relacionamento frequente entre os vértices de dois supervértices no grafo original. Um sumário é considerado relevante quando possui três características: diversidade, cobertura e concisão. A diversidade está relacionada com a existência de superarestas fortes entre supervértices com atributos de valores diferentes,

pois fornece mais informações sobre o grafo original. A cobertura está relacionada à existência de mais supervértices que participam de superarestas fortes, pois resulta em um sumário de melhor compreensão. Por último, a concisão está relacionada com a existência de poucos supervértices e de superarestas fortes, pois resulta em um sumário mais resumido, de visualização e compreensão mais fáceis.

4.1.4 **Sumarização de Grafo Homogênea**

A técnica de sumarização produz um grafo sumário e pode ser feita de duas formas:

- Restritiva: Para existir uma superaresta entre dois supervértices, cada vértice de um supervértice deve ter pelo menos uma aresta com um vértice do outro supervértice.
- Flexível: Para existir uma superaresta entre dois supervértices deve existir pelo menos uma aresta entre um par de vértices, onde cada um pertença a um supervértice envolvido na relação.

A abordagem restritiva pode produzir um grafo sumário tão volumoso quanto o original. A abordagem flexível pode ter várias formas de agrupamento diferentes, por isso o desafio é medir a qualidade da sumarização. Além disso, nas duas abordagens os vértices dos supervértices devem conter o mesmo conjunto de atributos e com os mesmos valores. Neste caso, trabalhar com atributos que possuem grande variedade de valores pode gerar um grafo sumário com muitos supervértices.

O trabalho de (Liu e Yu, 2011) propõe três algoritmos para fazer a sumarização. O primeiro gera uma sumarização homogênea exata, baseada na abordagem restritiva. O segundo gera uma sumarização homogênea aproximada, baseada na abordagem flexível. O terceiro gera uma sumarização homogênea aproximada otimizada, que procura analisar todas as formas de agrupamento possíveis e identificar qual delas é a que possui melhor qualidade, ou seja, menos diferenças com a sumarização gerada pela abordagem restritiva.

A sumarização homogênea exata é baseada na abordagem restritiva, onde um supervértice é homogêneo exato se: os valores dos atributos dos vértices de um supervértice são iguais; vértices de um mesmo supervértice têm conjuntos de supervértices vizinhos iguais; e vértices de um mesmo supervértice têm números de

arestas iguais para cada supervértice vizinho. Este tipo de agrupamento produz a melhor sumarização de um grafo volumoso em termos de homogeneidade. Contudo, o tamanho do grafo sumário pode ser tão grande quando o do grafo original, principalmente se os vértices tiverem muitos atributos e com valores distintos.

Para produzir um grafo sumário mais compacto é preciso relaxar os requisitos descritos acima, que é a abordagem da sumarização homogênea aproximada. Um supervértice é homogêneo aproximado se: os valores dos atributos dos vértices de um supervértice são similares; vértices de um mesmo supervértice têm conjuntos de supervértices vizinhos similares; vértices de um mesmo supervértice têm números de arestas similares para um supervértice vizinho. Podem existir várias formas de agrupamento que satisfaçam estas restrições, mas o objetivo é achar a melhor delas. A sumarização homogênea aproximada otimizada mede a qualidade de uma sumarização proposta analisando o quanto esta é similar à sumarização exata.

4.2 Considerações Sobre a Sumarização

A sumarização tem como objetivo obter uma representação resumida (sumário) de um grafo volumoso, que contenha todos os dados do grafo original e seja mais adequada para visualização e análise. Para gerar o grafo sumário é preciso determinar uma ordem ou forma de agrupar vértices e arestas, com base nas suas similaridades ou igualdades. Os trabalhos apresentados aplicam a sumarização em grafos genéricos, onde a estrutura e semântica dos grafos não são conhecidas. Por isso, se concentram em agrupar vértices de acordo com seus conjuntos similares de arestas com outros vértices e de acordo com seus atributos, que devem ser iguais e possuir os mesmos valores. Entretanto, a maioria destes trabalhos não considera que os vértices e arestas dos grafos podem possuir vários atributos e com valores distintos. Neste cenário, o grafo sumário produzido pode ser tão volumoso quanto o grafo original. Além disso, em alguns dos trabalhos apresentados é preciso refazer os grafos da base original para responder as consultas de proveniência, como descrito no item 4.1.1. A Tabela 5 apresenta um resumo das técnicas de sumarização apresentadas, com suas principais características, vantagens e desvantagens.

Tabela 5. Resumo da Sumarização

	Sumarização	Agrupa dados de acordo com uma ou mais características em comum.		
	Com Erro Limitado	Iterativa	Orientada à Descoberta	Homogênea
Agrupamento	Cliques, áreas densas do grafo, subgrafo completo e bipartido. Vértices com arestas para um mesmo conjunto de vértices.	Vértices que possuem um atributo em comum e com o mesmo valor são agrupados em um supervértice. Dois supervértices possuem uma superaresta entre eles se todos os vértices do primeiro possuírem pelo menos uma aresta com um vértice do segundo.	Categorização de atributos baseada nos valores dos atributos dos vértices e nas suas arestas.	Exata: Vértices de um supervértice devem possuir atributos em comum, com os mesmos valores e o mesmo número de arestas para supervértices vizinhos. Aproximada: Vértices de um supervértice devem possuir atributos em comum, com valores similares e número similar de arestas para supervértices vizinhos.
Vantagens	Grafo sumário e lista de arestas de correção.	Grafo sumário com homogeneidade de vértices e arestas.	Grafo sumário com número de supervértices mais reduzido do que o produzido pela iterativa.	Duas abordagens para a sumarização. A exata produz um sumário com homogeneidade de vértices e arestas. A aproximada produz um sumário mais reduzido.
Desvantagens	Insere no sumário arestas que não existem no grafo original e remove arestas que existem. É preciso refazer o grafo original para responder as consultas.	Pode produzir um grafo sumário tão volumoso quanto o grafo original, se os atributos possuírem muitos valores distintos.	Dificuldade em obter uma divisão dos atributos em categorias. Depende do conhecimento de um especialista.	A exata pode produzir um grafo sumário tão volumoso quanto o grafo original. A aproximada pode gerar várias formas de sumarização.

4.3 Compressão de Grafos

A compressão tem como objetivo reduzir o número de bytes necessários para codificar o grafo original. Comprimir dados significa também retirar redundâncias, considerando que, geralmente, os dados contêm informações registradas repetidas vezes que podem ou precisam ser eliminadas para reduzir o seu espaço de armazenamento. O objetivo é armazenar os dados repetidos uma única vez, fazendo uma referência a ele sempre que voltar a aparecer. Por exemplo, a sequência "*AAAAAA*" ocupa 6 bytes, mas poderia ser representada pela sequência "*6A*", que ocupa 2 bytes, economizando espaço.

A compressão, geralmente, cria dicionários para catalogar os dados. O dicionário pode ser elaborado, por exemplo, como uma lista numerada, onde se atribui um número para cada dado repetido. Assim, é possível referenciar um dado pelo seu respectivo número, ao invés de repeti-lo com o conteúdo original. O dicionário deve ser armazenado juntamente com os dados comprimidos, para que seja possível restabelecer os dados originais. Esta abordagem pode comprometer as consultas de proveniência, pois para consultar os dados do grafo original é preciso refazê-lo aplicando o dicionário de dados.

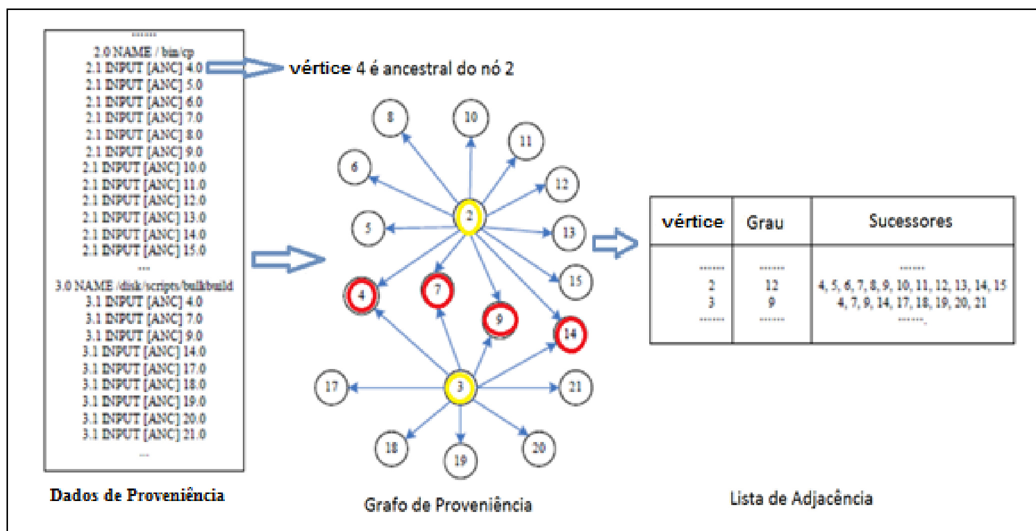
A maioria dos trabalhos encontrados na literatura sobre compressão de grafos refere-se a grafos Web. O trabalho apresentado em (Xie *et al.,* 2011) aplica técnicas de compressão Web para grafos de proveniência. Para isso, o trabalho considera que grafos de proveniência e grafos Web possuem estruturas similares e características em comum.

Os grafos web são direcionados, seus vértices são as URLs e suas arestas são as ligações entre as páginas. Os algoritmos de compressão de grafos web exploram suas características como:

- Localidade: Poucas ligações vão além do domínio URL. Assim, a grande maioria tende a apontar para páginas próximas.
- Similaridade: Páginas que não estão muito distantes umas das outras tem grande probabilidade de possuírem vizinhos em comum.
- Consecutividade: Os números dos vértices sucessores de uma página estão em ordem sequencial.

Segundo (Xie *et al.,* 2011), os grafos de proveniência possuem estrutura e características semelhantes aos grafos web. A Figura 12 mostra o mapeamento dos dados de proveniência para uma lista de adjacências que representa o grafo de proveniência. Esta figura também ilustra que o grafo de proveniência possui as características dos grafos web: similaridade, localidade e consecutividade.

- Similaridade: Os vértices 2 e 3 são similares, pois possuem os vértices vizinhos sucessores em comum 4, 7, 9 e 14.
- Localidade: Os sucessores do vértice 2 estão entre 4 e 15, enquanto os sucessores do no 3 estão entre 4 e 21. Isso acontece porque vértices que são usados como entrada para um processo, geralmente, estão num mesmo diretório e, por isso, possuem números próximos.
- Consecutividade: Os sucessores dos vértices 2 e 3 não numerados de 4 a 15 e de 17 a 21. Esta numeração é atribuída pelo sistema de captura de proveniência à medida que o grafo é gerado.



vértice	Grau	Sucessores
......	......	......
2	12	4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
3	9	4, 7, 9, 14, 17, 18, 19, 20, 21
......	......	......

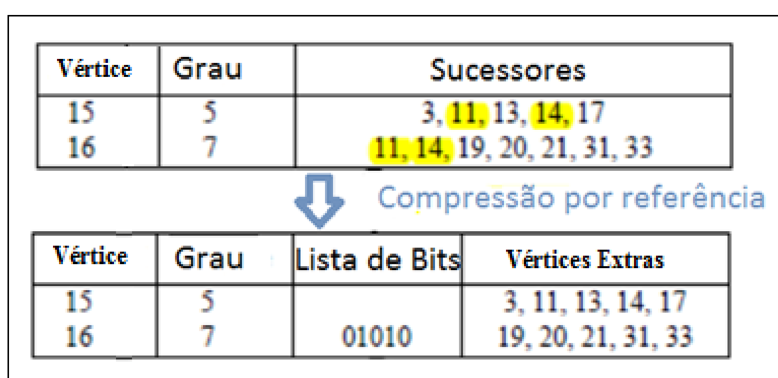
Figura 12. Dados de proveniência com grafo e lista de adjacência correspondentes (Xie *et al.,* 2011)

A técnica de compressão de grafos web descrita em (Xie *et al.,* 2011) aplica três passos básicos:

1. Compressão por referência: A lista de sucessores de um vértice é codificada usando como referência uma lista similar de sucessores de outro vértice. O objetivo é reduzir o volume evitando codificar dados duplicados.
2. Achar números consecutivos: Codifica números consecutivos armazenando o menor número da sequência e a quantidade de números que pertencem à mesma.

3. Codificar lacunas: Codifica lacunas entre sucessores de um vértice, ao invés de codificar os próprios sucessores.

Na compressão por referência o primeiro passo é achar um vértice Y cuja lista de sucessores seja mais similar à lista de sucessores de um vértice X, que será usado como parâmetro. A lista de sucessores de Y é codificada em duas partes: uma lista de bits, que identifica sucessores em comum entre X e Y, e vértices extras, que são os sucessores de Y que não são sucessores de X. A Figura 13 mostra um exemplo da compressão por referência, onde o vértice 15 (X) é usado como parâmetro para o vértice 16 (Y) e estes possuem listas similares de vértices sucessores.



Vértice	Grau	Sucessores
15	5	3, 11, 13, 14, 17
16	7	11, 14, 19, 20, 21, 31, 33

Vértice	Grau	Lista de Bits	Vértices Extras
15	5		3, 11, 13, 14, 17
16	7	01010	19, 20, 21, 31, 33

Figura 13. Primeiro passo da compressão por referência (Xie *et al.,* 2011).

Os vértices 15 e 16 possuem os sucessores 11 e 14 em comum. Após a aplicação da compressão por referência, o vértice 16 é codificado a partir do vértice 15 em duas partes: uma lista de bits e uma lista de vértices extras. A lista de bits identifica quais sucessores do vértice 15 também são sucessores do vértice 16. Esta lista contém um bit para cada sucessor de 15, onde 0 representa que 16 não possui o sucessor de 15 correspondente à posição, e 1, que possui. No exemplo, a lista de bits de 16 possui 1 nas posições correspondentes aos números 11 e 14 da lista de sucessores de 15. A lista de vértices extras contém o resto dos vértices sucessores de 16 que não são sucessores de 15.

O segundo passo procura números consecutivos na lista de sucessores de 15 e dos vértices extras de 16. Estes vértices são codificados armazenando o menor número da sequência e a quantidade de números que ela possui. A Figura 14 ilustra o segundo passo.

	Grau	Lista de Bits	Vértices Extras
15	5		3, 11, 13, 14, 17
16	7	01010	19, 20, 21, 31, 33

Achar números consecutivos

Vértice	Grau	Lista de Bits	Menor Num	Qtd	Vértices Extras
15	5		13	2	3, 11, 17
16	7	01010	19	3	31, 33

Figura 14. Segundo passo da compressão por referência (Xie *et al.*, 2011).

No exemplo, a lista de sucessores de 15 possui a sequência 13 e 14. Assim, são armazenados o menor número da sequência e sua quantidade de números. A lista de vértices extras contém os vértices restantes. Já a lista de sucessores de 16 possui a sequência 19, 20 e 21. Da mesma forma, são armazenados o menor número, a quantidade de números da sequência e os vértices extras.

O terceiro passo consiste em codificar as lacunas entre os sucessores restantes (vértices extras). Sejam X1, X2, X3,..., Xn sucessores de X, em ordem crescente, a codificação é feita como: X1 - X, X2 - X1,..., Xn - Xn-1. A Figura 15 ilustra o terceiro passo.

Vértice	Grau	Lista de Bits	Menor Num	Qtd	Vértices Extras
15	5		13	2	3, 11, 17
16	7	01010	19	3	31, 33

Codificar lacunas

Vértice	Grau	Lista de Bits	Menor Num	Qtd	Vértices Extras
15	5		13	2	-12, 8, 6
16	7	01010	19	3	15, 2

Figura 15. Terceiro passo da compressão por referência (Xie *et al.*, 2011).

No exemplo, aplicando as regras de codificação de lacunas para o vértice 15 do exemplo, obtém-se:

$X = 15$, $X_1 = 3$, $X_2 = 11$ e $X_3 = 17$

Calcular:

$X_1 – X = 3 – 15 = -12$

$X_2 - X_1 = 11 – 3 = 8$

$X_3 – X_2 = 17 – 11 = 6$

O trabalho de (Xie *et al.*, 2011) descreve duas abordagens para aplicar a compressão web em grafos de proveniência. A primeira está relacionada com a busca de listas de sucessores similares e a segunda com a codificação de lacunas entre sucessores de um vértice.

a) Lista de Referência Identificada pelos Nomes dos Vértices

A técnica de compressão de grafos web busca entre as listas de sucessores dos vértices a mais similar a de um determinado vértice. Contudo, achar listas similares não é uma tarefa fácil, especialmente se o grafo possuir muitos vértices.

As bases de proveniência, geralmente, armazenam os nomes dos vértices. Além disso, estes nomes tendem a se repetir na base de dados, pois os processos que geram os dados de proveniência são executados várias vezes, com parâmetros de entrada distintos. Assim, vértices com o mesmo nome, geralmente, possuem lista de sucessores similares.

Uma abordagem para buscar listas de sucessores similares em grafos de proveniência é utilizar como referência, para um determinado vértice, a lista de outro vértice com o mesmo nome. A Figura 16 ilustra um exemplo desta abordagem.

W=3
W=3

Vértice	Nome	Sucessores
...	...	...
15	/bin/cp	19, 21, 32
16	/bin/bash	4, 9, 13, 17
17	/bin/su	19, 20, 23
18	/usr/bin	3, 11, 13, 14, 17
19	/bin/hostname	3, 10, 13, 17
20	/sbin/consoletype	4, 8, 9, 11
21	/meminfo	4, 8, 11, 15
22	/usr/bin/id	5, 7, 11, 12
23	/usr/bin	3, 11, 13, 14, 18
24	/bin/sed	4, 6
25	/usr/bin	3, 11, 13, 14, 19
...	...	...

Figura 16. Buscar lista de sucessores similares usando como referência para um vértice a lista de outro vértice com o mesmo nome (Xie *et al.*, 2011).

Na figura, quando o vértice 18 aparece pela primeira vez o algoritmo de compressão procura uma lista de sucessores similares nos três vértices existentes (15, 16 e 17). A lista mais similar é a do vértice 16, que é escolhida como referência. Entretanto, quando o vértice 23 é incluído, como este possui o mesmo nome do vértice

18, a lista escolhida como referência é a do vértice 18. O mesmo ocorre quando o vértice 25 é inserido no grafo.

b) Codificar Lacunas entre Sucessores de Vértices Diferentes

Os algoritmos de compressão de grafos web exploram a localidade dos mesmos e codificam lacunas entre os sucessores de um vértice. Em relação aos grafos de proveniência, normalmente, seus vértices possuem um único sucessor. Uma alternativa é codificar as lacunas entre sucessores de vértices diferentes. A Figura 17 mostra uma comparação entre estas duas abordagens.

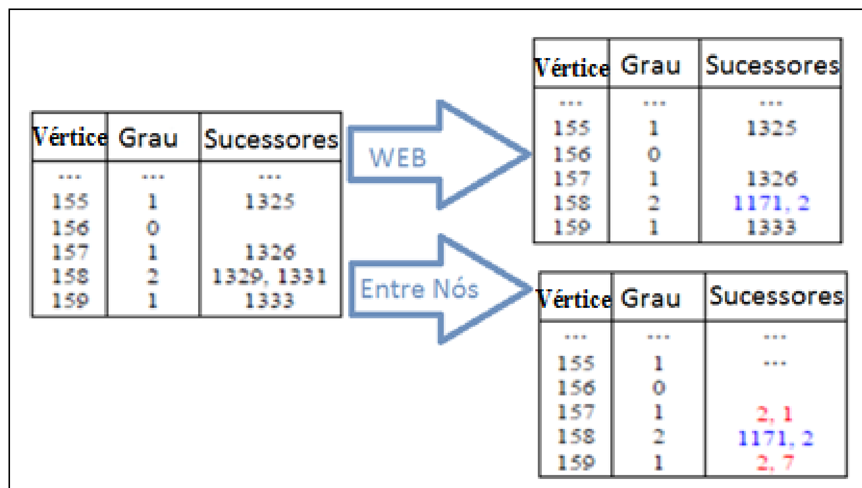


Figura 17. Codificar lacunas entre os sucessores de um único vértice e entre sucessores de vértices diferentes (Xie *et al.*, 2011).

A compressão web codifica somente os sucessores do vértice 158. Onde:

158
1329 – 158 = 1171
1331 – 1329 = 2

A abordagem proposta para grafos de proveniência também codifica os sucessores dos vértices 157 e 159. Onde:

157	159
157 – 155 = 2	159 – 157 = 2
1326 – 1325 = 1	1333 – 1326 = 7

A Tabela 6 apresenta um resumo da técnica de compressão, com suas principais características, vantagens e desvantagens.

Tabela 6. Resumo da Compressão.

	Compressão	Reduz o número de bytes necessários para codificar o grafo original, substituindo informações repetidas por códigos e criando um dicionário de correspondência entre o código criado e a informação original.
Vantagens	Reduz o espaço de armazenamento do grafo original e elimina informações repetidas no mesmo.	
Desvantagens	Pode gerar um custo extra para as consultas, uma vez que é preciso fazer cálculos para decodificar o grafo comprimido para obter os dados originais e responder as consultas.	

4.4 Redução de Grafos

A redução de grafos utiliza técnicas para reduzir o espaço de armazenamento dos dados de proveniência, identificando e subtraindo dados comuns a todos os grafos de proveniência. Um exemplo da aplicação de redução para grafos de proveniência é encontrado em (Chapman *et al.,* 2008). O trabalho propõe técnicas que incluem um modelo de proveniência e dois tipos de algoritmos de redução: fatoração e herança de proveniência. O primeiro reduz o espaço de armazenamento removendo registros e vértices de proveniência duplicados. O segundo é uma otimização do primeiro e procura similaridades estruturais numa parte dos dados (herança estrutural) ou entre dados de um tipo específico (herança de predicado).

O modelo de proveniência proposto em (Chapman *et al.,* 2008) é composto pelos seguintes itens:

- Manipulação: Recebe um ou mais conjuntos de dados como entrada e produz um conjunto de dados como saída. Um workflow é modelado como um grafo direcionado onde cada vértice representa uma manipulação. Uma manipulação pode ser, por exemplo, uma função ou um programa completo. Uma consulta pode ser uma manipulação ou uma árvore de manipulações. Por exemplo, uma seleção recebe um conjunto de dados como entrada e executa uma ou mais manipulações para gerar os dados de saída com base em algum critério.
- Registro de Proveniência: Armazena os dados de entrada e as manipulações realizadas nele para produzir um novo item.

- Vértice de Proveniência: Consiste em uma manipulação simples, seus dados de entrada e parâmetros que fazem parte da proveniência armazenada para um determinado dado.
- Componente do Vértice de Proveniência: Uma manipulação simples, dados de entrada ou parâmetros que formam parte de um vértice de proveniência.
- Proveniência de Instância: Proveniência armazenada e associada a um determinado dado.
- Proveniência do Conjunto de Dados: Proveniência armazenada e associada a todo o conjunto de dados.
- Proveniência Armazenada: Repositório com todos os registros de proveniência. Inclui a proveniência de todo o conjunto de dados e dos dados individuais.

4.4.1 **Fatoração**

Em uma base de dados muito volumosa alguns itens devem possuir proveniência idêntica ou similar. A fatoração consiste em identificar estas similaridades entre os dados para que sejam armazenados somente uma vez. São propostos três tipos de fatoração: básica, de vértices e de argumentos (Chapman *et al.,* 2008).

- Fatoração Básica

Identifica registros de proveniência idênticos e armazena uma única cópia. O algoritmo percorre a base de dados e separa os registros de proveniência duplicados dos dados. Na base de dados original são introduzidos ponteiros para os respectivos registros de proveniência da nova base. A Figura 18 mostra um exemplo de fatoração básica.

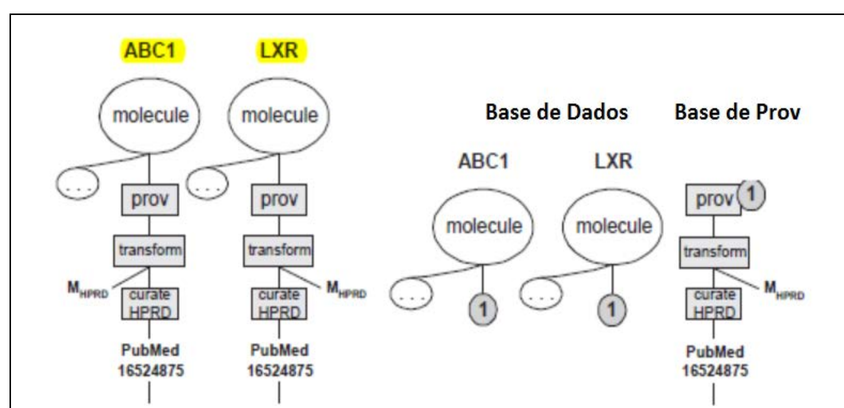


Figura 18. Exemplo de fatoração básica (Chapman *et al.*, 2008).

Na Figura 18, as moléculas ABC1 e LXR são ilustradas com sua proveniência. O algoritmo verifica que os registros de proveniência são idênticos e os substitui por um ponteiro (1) para o registro na base de proveniência.

- Fatoração de Vértices

É aplicada quando dois itens possuem registros de proveniência distintos, mas com vértices em comum. A fatoração remove os vértices iguais e armazena somente uma cópia de cada vértice na base de proveniência. Na base de dados são armazenados ponteiros para os respectivos vértices. Por exemplo, supondo os seguintes registros de proveniência:

P0 = A -> B -> C
P1 = A -> X -> C

Após a fatoração básica, estes dois registros são armazenados na base de proveniência. Entretanto, aplicando a fatoração de vértices é possível fazer a seguinte redução:

P0 e P1 = A -> (B OU X) -> C

Um ponteiro é armazenado na base de dados para indicar se a proveniência do item é P0 ou P1 e se o vértice presente é B ou X. Entretanto, para fazer esta redução é preciso determinar que os vértices A e C de P0 e P1 são iguais e que os vértices B e X são similares.

Assim, dois vértices são iguais se:

- Pertencem a uma mesma manipulação.
- Os parâmetros das manipulações são iguais.

Dois vértices são similares em relação a uma determinada função de similaridade. Estas funções devem ser escritas por especialistas que tenham conhecimento da base de dados. As funções de similaridade definem uma relação binária entre vértices e possuem as seguintes propriedades de equivalência:

- Reflexiva: Cada vértice de proveniência é similar a si próprio.

- Simétrica: Se um vértice N1 é similar a um vértice N2, então N2 é similar a N1.
- Transitiva: Se N1 é similar a N2 e N2 é similar a N3, então N1 é similar a N3.

O algoritmo percorre a base de dados. Quando encontra um vértice de proveniência faz a sua busca na base de proveniência. Se não existir, é inserido na mesma. Quando todos os vértices são visitados as similaridades são procuradas na base de proveniência. Assim, se um vértice N1 é igual ou similar a um vértice N2, se tem o mesmo ancestral e o mesmo descendente, os registros dos dois vértices são reduzidos a um só. O ponteiro da base de dados deve ser atualizado para referenciar os vértices envolvidos na redução.

Um grafo de proveniência pode conter uma longa cadeia de vértices similares. Para que duas cadeias sejam similares devem possuir:

- Mesmo comprimento.
- Os primeiros vértices iguais.
- Os últimos vértices iguais.
- Vértices intermediários similares de acordo com a função utilizada.

A Figura 19 ilustra dois registros de proveniência com tamanhos diferentes e, por isso, não são combinados pela fatoração de vértices. Entretanto, é possível fazer esta combinação indicando o vértice 2 como opcional.

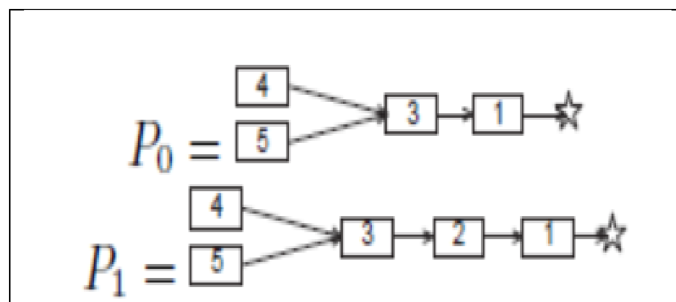


Figura 19. Registro de proveniência com tamanhos diferentes (Chapman *et al.,* 2008).

A Figura 20 mostra a proveniência armazenada sem a fatoração de vértices e com a fatoração de vértices, sendo que a segunda opção obtém a redução dos dados. O ponteiro da base de dados deve indicar quando um vértice é opcional ou não para o item a ele associado.

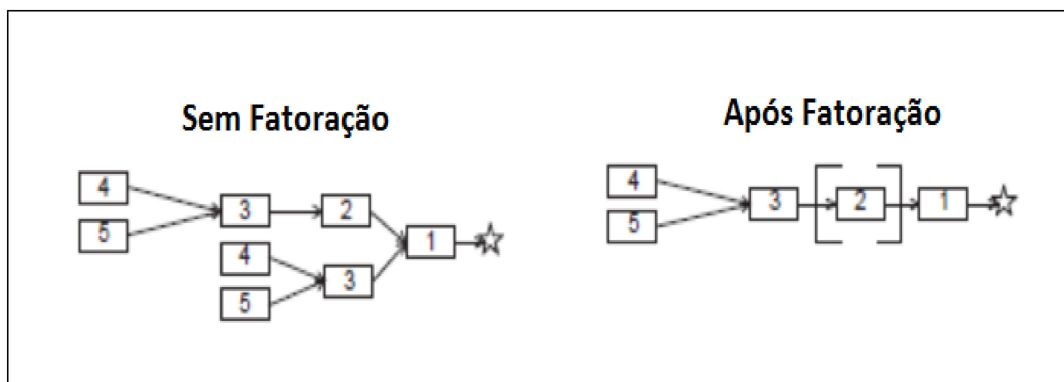


Figura 20. Exemplo da figura 19 com e sem fatoração de vértices (Chapman *et al.,* 2008).

- Fatoração de Argumentos

Pequenas diferenças entre os vértices podem limitar o uso da fatoração. Para evitar que isso ocorra, a fatoração de argumentos procura identificar componentes que não se repetem na base de dados (argumentos), para que os vértices possam ser fatorados. Um argumento é um componente que existe no grafo de proveniência, mas não se repete mais do que um determinado número de vezes estabelecido pelo usuário. Na Figura 21 os registros de proveniência das moléculas são iguais, exceto pelo valor de PubMed.

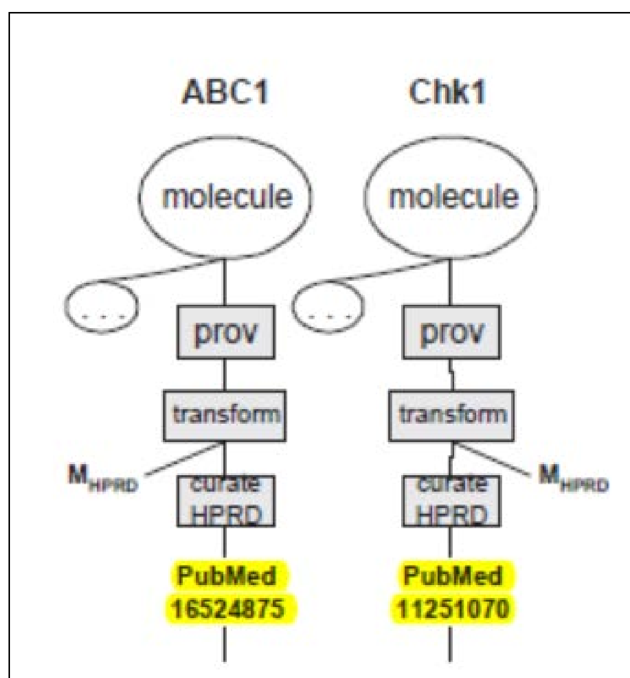


Figura 21. Exemplo de registros de proveniências iguais, mas com o argumento PubMed diferente (Chapman *et al.,* 2008).

O algoritmo envolve duas buscas na base de dados. A primeira identifica os argumentos. Para isso, é contabilizado o número de vezes que cada componente apareceu. Os componentes com uma ocorrência, por exemplo, são identificados como

argumentos. O segundo passo aplica a fatoração de vértices, gera a base de proveniência com uma única cópia de cada vértice e insere os respectivos ponteiros nas bases de dados originais. Os argumentos permanecem na base de dados. A Figura 22 ilustra a fatoração de argumentos.

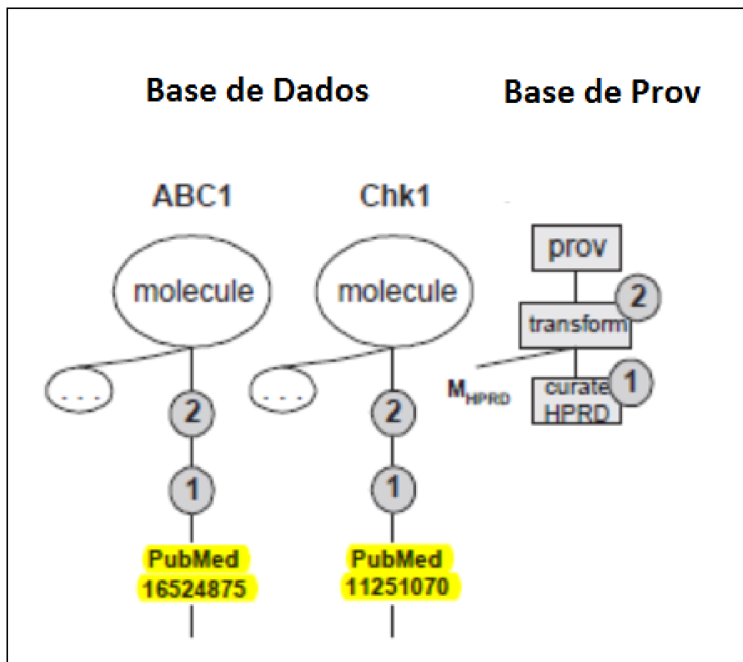


Figura 22. Fatoração de argumentos (Chapman *et al.* 2008).

4.4.2 **Herança de Proveniência**

Consiste em uma otimização da fatoração, onde as similaridades são procuradas em uma parte dos dados (herança de estrutura) ou entre os dados de proveniência associados a itens de um determinado tipo (herança de predicado). Quando um item herda proveniência não é necessário armazená-la, pois o mecanismo de herança irá instanciar corretamente os dados necessários (Chapman *et al.*, 2008).

- Herança de Estrutura

Normalmente itens que possuem relacionamento estrutural (pais-filhos ou ascendente-descendente) compartilham informações de proveniência. Entretanto, para herdar a proveniência do antecessor por estrutura, o descendente deve possuir proveniência idêntica à do seu antecessor. Quando um dado não possuir proveniência associada na base de dados resultante, pode-se assumir que está é a mesma do seu antecessor. A Figura 23 mostra um exemplo de dados de uma molécula e sua proveniência associada, e também, os mesmos dados após a aplicação da herança de estrutura.

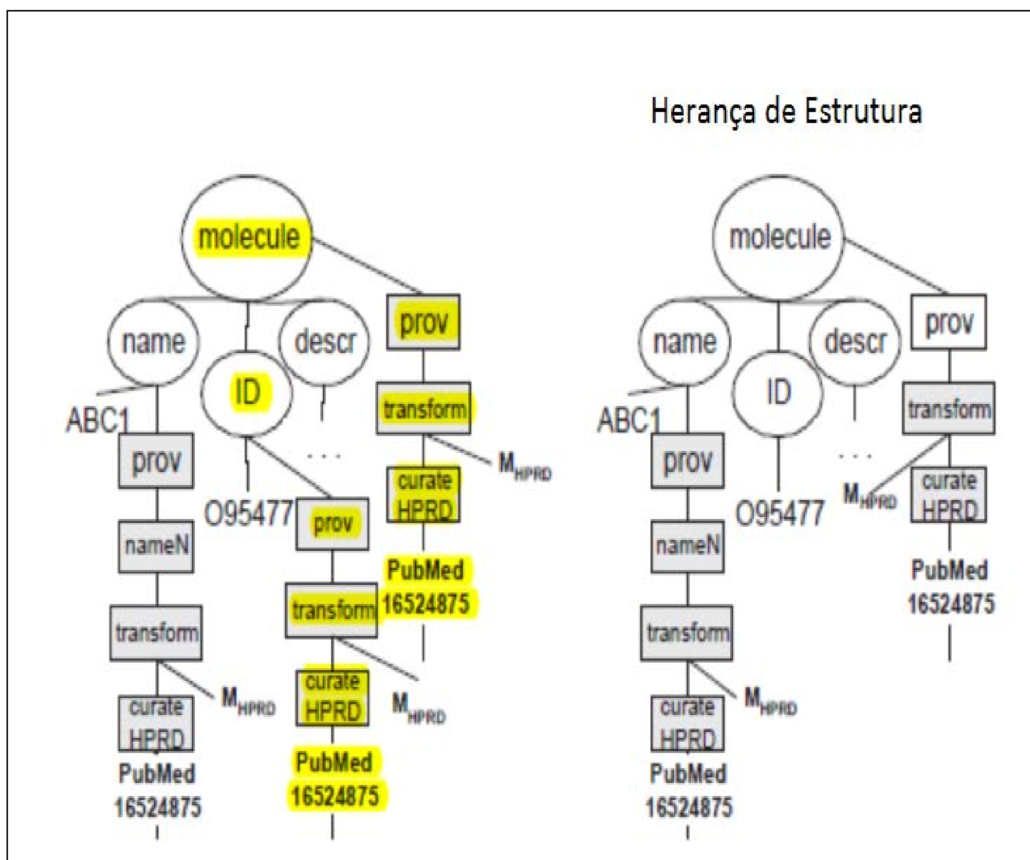


Figura 23. Herança de Estrutura (Chapman *et al.,* 2008).

Na figura à esquerda, o item "*ID*" compartilha os mesmos dados de proveniência do seu antecessor (*molecule*). Ao invés de armazenar todos os dados de proveniência o algoritmo armazena somente os que são diferentes dos seus antecessores. No exemplo, o item "*name*" possui dados de proveniência diferentes do seu antecessor (*molecule*). A figura à direita mostra os dados reduzidos após a aplicação da herança de estrutura.

- Herança de Predicado

A base de dados é dividida em grupos que contenham registros de proveniência idênticos usando como parâmetro um ou mais predicados definidos pelo usuário. Para isso, o usuário deve ter conhecimento da base de dados. Por exemplo, um predicado definido pode ser "*dados com o mesmo nome*". Assim, é possível armazenar um único registro de proveniência para todos os dados da base que possuam o mesmo nome. A Figura 24 mostra a base de dados original e a base de dados de proveniência após a aplicação da herança de predicado de dados com o mesmo nome.

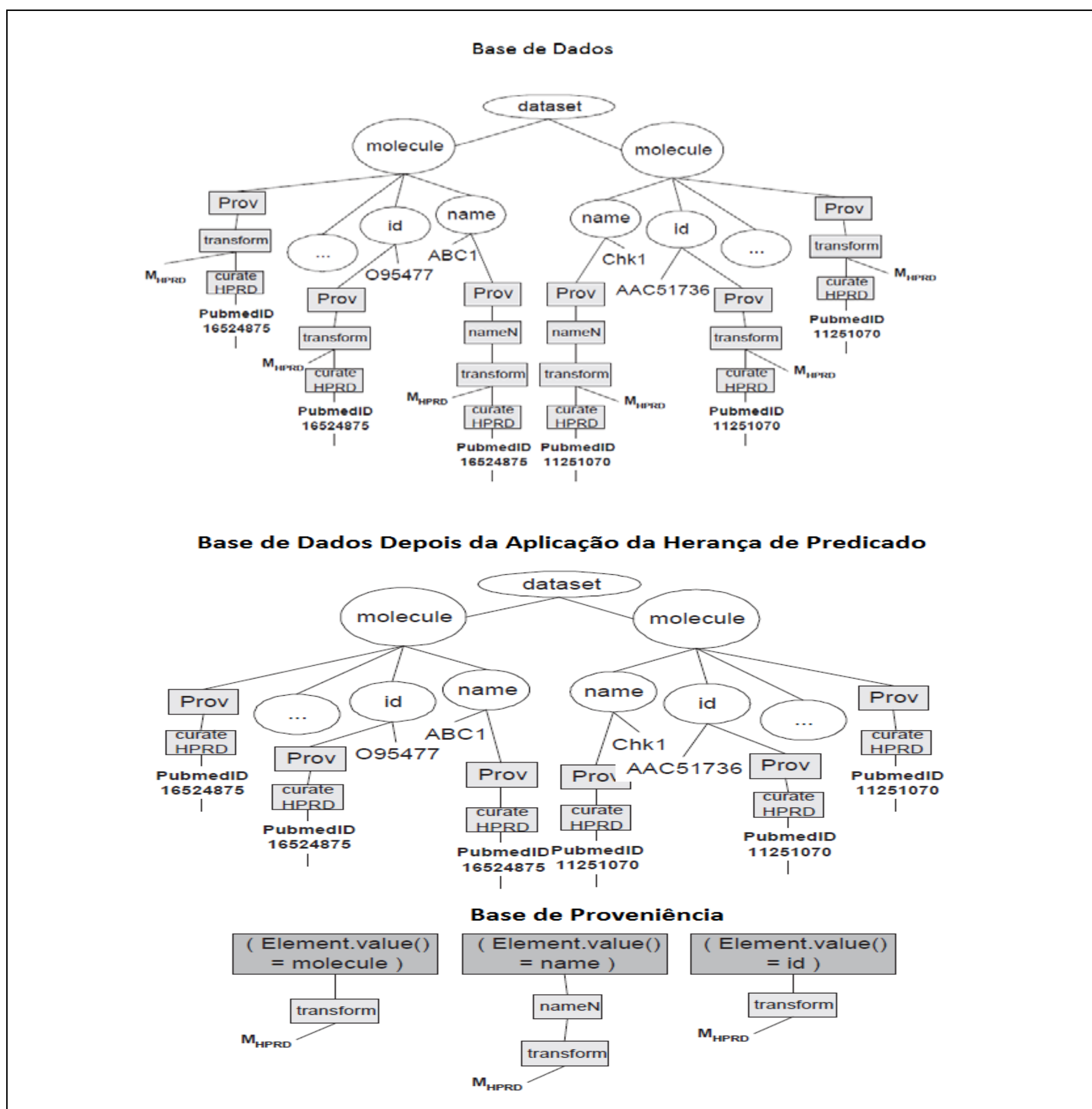


Figura 24. Herança de predicado de dados com mesmo nome (Chapman *et al.*, 2008).

O algoritmo percorre a base de dados duas vezes. O primeiro passo identifica os componentes dos registros de proveniência que são comuns a todos os dados que satisfazem cada predicado utilizado.

Se um dado satisfaz um predicado e na base de proveniência ainda não existe o registro predicado-proveniência, ele é criado. Este registro contém todos os componentes de proveniência do dado. Se o registro já existir, são procurados e removidos os componentes do registro que não fazem parte da proveniência do dado.

No fim do primeiro passo, a base de proveniência contém um conjunto de pares predicado-proveniência. Cada par só irá existir se todos os dados que satisfazem o predicado possuírem um subconjunto não vazio de componentes de proveniência em

comum. O segundo passo armazena o resto da proveniência que não foi herdada pelo predicado.

A Tabela 7 apresenta um resumo da técnica de redução, com suas principais características, vantagens e desvantagens.

Tabela 7. Resumo da Redução

	Redução	Reduz o espaço de armazenamento dos grafos de proveniência identificando e subtraindo dados comuns a todos os grafos.
Vantagens	Modelo de proveniência e dois algoritmos de redução: fatoração e herança. Reduz o espaço de armazenamento dos grafos originais e elimina dados repetidos nos mesmos.	
Desvantagens	Cria uma nova base de proveniência onde são armazenados os dados repetidos nos grafos. Introduz ponteiros na base original para os respectivos dados da nova base de proveniência. Para consultar os grafos é preciso consultar as duas bases de proveniência podendo comprometer o tempo de processamento das consultas.	

4.5 Conclusões do Capítulo

Analisando os trabalhos propostos na literatura referentes à compactação de grafos é possível identificar que a compressão tem como objetivo reduzir o volume dos grafos eliminando dados duplicados. Isso é feito através da sua substituição de dados repetidos na base original por códigos e da criação de uma tabela de correspondência dos códigos com os dados originais. A compressão pode gerar um custo extra para as consultas, uma vez que é preciso fazer cálculos para decodificar o grafo comprimido para obter os dados originais e responder as consultas.

A redução utiliza técnicas para diminuir o espaço de armazenamento dos dados de proveniência subtraindo dados repetidos e armazenando-os em uma nova base de proveniência. Para responder as consultas de proveniência persiste a necessidade de consultar os grafos da base original e da nova base de proveniência criada, podendo comprometer a sua eficiência.

A sumarização tem como objetivo obter uma representação resumida (sumário) de um grafo volumoso, que contenha todos os dados do grafo original e seja mais adequada para a sua visualização e análise. Para gerar o grafo sumário é preciso determinar uma ordem ou forma de agrupar vértices similares. Em alguns trabalhos que

aplicam a técnica de sumarização é preciso refazer os grafos da base original para responder as consultas de proveniência, como no descrito no item 4.1.1. Os trabalhos encontrados na literatura relativos à sumarização são aplicados para grafos genéricos, onde a estrutura e semântica dos grafos não são conhecidas. Por isso, se concentram em agrupar vértices de acordo com seus conjuntos similares de arestas com outros vértices e com seus atributos, que devem ser iguais e possuir os mesmos valores. Entretanto, a maioria destes trabalhos não considera que os vértices e arestas dos grafos podem possuir vários atributos e com valores distintos. Neste cenário, o grafo sumário produzido pode ser tão volumoso quanto o grafo original.

Analisando as técnicas de compactação de grafos, esta tese escolheu aplicar a técnica de sumarização para grafos de proveniência, pois não é preciso decodificar o grafo resultante, como na compressão, e também, não é preciso consultar mais de uma base de proveniência, como na redução. O objetivo da técnica de sumarização aplicada é de gerar um grafo sumário que represente os grafos da base original, mas com volume reduzido, e que contenha todas as informações necessárias sobre os grafos originais para responder as consultas de proveniência. Somente o grafo sumário é usado para responder as consultas sem que seja preciso refazer os grafos da base original. O SGProv, mecanismo de sumarização proposto nesta tese é a presentado no próximo capítulo.

Capítulo 5 O Mecanismo de Sumarização SGProv

Esta tese adota com o conceito de experimento científico como sendo um conjunto de simulações computacionais, em especial execuções de *workflows* científicos. Os *workflows* científicos são uma abstração para modelar o fluxo de programas e de dados em um experimento científico, onde os programas representam algum algoritmo ou método que deve ser aplicado ao longo do processo de experimentação (Barker e van Hemert, 2008). Uma das atividades principais na condução do experimento científico é realizar a análise dos dados produzidos após as diversas execuções dos *workflows*. Uma das análises principais corresponde a consultas sobre estas execuções. Assume-se que os resultados das execuções dos *workflows* são representados como grafos de proveniência, que indicam o caminho de derivação dos dados até o resultado da execução. Portanto, os grafos são direcionados e acíclicos. Nesta tese, um único experimento científico pode estar relacionado a um conjunto de grafos de proveniência. Propiciar a análise do experimento através destes conjuntos de grafos é o principal objetivo desta tese. Assim, foi desenvolvido o SGProv, um mecanismo de sumarização para grafos de proveniência, visando ao alto desempenho de consultas a respeito de um experimento científico. Este capítulo descreve as definições adotadas no SGProv, explica como o grafo sumário é gerado a partir dos grafos de proveniência e apresenta um exemplo de aplicação do mecanismo.

5.1 Definições

O SGProv sumariza um conjunto de grafos de proveniência, gerado a partir das execuções de *workflows* científicos. No SGProv, um grafo é definido da seguinte forma.

Definição 1 (Grafo): Um grafo G é definido como $G = (V, E, A, T)$, onde $V=\{v_1, v_2,..., v_n\}$ é o conjunto de seus vértices, $E=\{e_1, e_2,..., e_m\} \subseteq (VxV)$ é o conjunto de suas arestas direcionadas, $A=\{a_1, a_2,..., a_p\}$ é o conjunto de atributos associados a seus vértices e/ou arestas e $T=\{t_1, t_2,..., t_q\}$ é o conjunto de tipos de suas arestas. Cada vértice $v_i \in V$ tem pelo menos um atributo $a_x \in A$. O valor de a_x no vértice v_i é representado por $Val(v_i, a_x)$. Cada aresta $e_j \in E$ possui um único tipo t_k definido por $Tipo(e_j)$, onde $t_k \in T$. O valor de um atributo a_y pertencente a uma aresta e_j é representado por $\text{Val}(e_j, a_y)$. Como as arestas são direcionadas, $Origem(e_j)$ e $Destino(e_j)$ representam respectivamente os vértices de origem e de destino da aresta e_j.

Um grafo de proveniência é um tipo especial de grafo definido da seguinte forma.

Definição 2 (Grafo de Proveniência): Um grafo de proveniência G_p é definido como $G_p = (V_p, E_p, A_p, T_p)$, onde os vértices V_p representam os elementos do grafo (atividades ou entidades) e as arestas E_p representam a linhagem dos vértices. Um atributo "*elem*" $\in A_p$ deve existir em todos os vértices. Se um vértice v_{pi} representa uma atividade, $Val(v_{pi}, elem) =$ *"atividade"*. Por outro lado, se v_{pi} representa uma entidade, $Val(v_{pi}, elem) =$ *"entidade"*. Vértices que representam atividades ou entidades também devem possuir um atributo *"cod"* $\in A_p$, que consiste no seu atributo identificador. O conjunto A_p também contém outros elementos dos grafos de proveniência, como, por exemplo, nomes das atividades. O conjunto T_p representa os tipos de arestas, que podem ser *WasInformedBy*, *WasDerivedFrom*, *Used*, *WasGeneratedBy*, *WasAttributedTo*, *WasAssociateWith* ou *Acted OnBehalfOf*.

Grafos de proveniência distintos podem ter vértices ou arestas correspondentes, que são definidos da seguinte forma.

Definição 3 (Vértices Correspondentes): Dois vértices v_{pi} and v_{pj}, pertencentes a grafos de proveniência diferentes, são correspondentes se eles representam um mesmo tipo de elemento, $Val(v_{pi}, elem) = Val(v_{pj}, elem)$, e possuem o mesmo valor para o atributo identificador, $Val(v_{pi}, cod) = Val(v_{pj}, cod)$.

Definição 4 (Arestas Correspondentes): Duas arestas e_{pi} e e_{pj}, pertencentes a grafos de proveniência diferentes, são correspondentes se elas possuem o mesmo tipo, $Tipo(e_{pi}) = Tipo(e_{pj})$, e possuem o mesmo vértice de origem e de destino, $(Origem(e_{pi}) = Origem(e_{pj}))$ e $(Destino(e_{pi}) = Destino(e_{pj}))$.

5.2 Grafo Sumário

O SGProv baseia-se na premissa de que vértices que representam um mesmo tipo de atividade frequentemente pertencem a vários grafos de proveniência. Esta premissa é baseada nas características dos experimentos científicos computacionais, onde atividades são executadas repetidamente, com diferentes combinações de parâmetros, em variações de um mesmo *workflow*. Por outro lado, vértices que representam entidades, geralmente, possuem maior variação nestes grafos, pois correspondem a entradas e saídas das atividades, que tendem a variar de execução para

execução. Além disso, atividades e entidades podem ter atributos iguais, algumas vezes com valores também iguais, em diferentes grafos de proveniência. Com base nestas características e com o objetivo de reduzir o volume do conjunto dos grafos de proveniência, a fim de obter melhor desempenho no processamento das consultas, a abordagem usada no SGProv é reduzir o máximo possível repetições de elementos nos grafos de proveniência. Uma vez que a sumarização é uma abordagem bastante usada na literatura para reduzir grafos muito volumosos com base nas similaridades de seus elementos, o SGProv aplica a sumarização em um conjunto de grafos de proveniência com o mesmo objetivo. Assim, o SGProv vértices e arestas correspondentes de um conjunto de grafos de proveniência são agrupados produzindo um único grafo sumário, que contém todos os vértices e arestas dos grafos originais, mas sem redundância. No SGProv, um grafo sumário é definido da seguinte forma.

Definição 5 (Grafo Sumário): Um grafo sumário G_s é definido como $G_s = (V_s, E_s, A_s, T_s)$. Cada vértice $v_s \in V_s$ é chamado de supervértice e cada aresta $e_s \in E_s$ é chamada de superaresta.

Dado um conjunto de grafos de proveniência $\{G_{p1} = (V_{p1}, E_{p1}, A_{p1}, T_{p1}), ..., G_{px} = (V_{px}, E_{px}, A_{px}, T_{px})\}$ a ser sumarizado, supervértices contêm conjuntos de vértices correspondentes e superarestas contêm conjuntos de arestas correspondentes. Estes são definidos da seguinte forma:

Definição 6 (Supervértice): Um supervértice $v_{si} = \{v_{p1}, v_{p2},..., v_{pn}\} \subseteq (V_{p1} \cup V_{p2} \cup ... \cup V_{px})$, onde $(Val(v_{p1}, elem) = Val(v_{p2}, elem) = ... = Val(v_{pn}, elem) =$ *"atividade"* (ou *"entidade"*) e $Val(v_{p1}, cod) = Val(v_{p2}, cod) = ... = Val(v_{pn}, cod))$.

Definição 7 (Superaresta): Uma superaresta conecta dois supervértices v_{si} e v_{sj} se existe uma aresta $e_k = (v_{pm}, v_{pn}) \in (E_{p1} \cup E_{p2} \cup ... \cup E_{px})$, onde $v_{pm} \in v_{si}$ e $v_{pn} \in v_{sj}$. Desta forma, uma superaresta e_{si} é definida como $e_{si}=\{e_{p1}, e_{p2},..., e_{pm}\} \subseteq (E_{p1} \cup E_{p2} \cup ... \cup E_{px})$, onde $\{Origem(e_{p1}), Origem(e_{p2}),..., Origem(e_{pm})\} \subseteq Origem(e_{si})$, $\{Destino(e_{p1}), Destino(e_{p2}),..., Destino(e_{pm})\} \subseteq Destino(e_{si})$ e $Tipo(e_{p1}) = Tipo(e_{p2}) = ... = Tipo(e_{pm})$.

Cada vértice (aresta) que pertença a um único grafo de proveniência G_{pk} é incluído no sumário como um supervértice (superaresta) que contém um único vértice (aresta). Alguns supervértices e superarestas possuem um atributo "*execução*", que consiste na lista de execuções do *workflow* em que estiveram presentes. Este atributo propicia a recuperação dos caminhos dos grafos originais para responder as consultas de

proveniência com base somente no grafo sumário. Supervértices e superarestas que contêm vértices e arestas de um mesmo grafo de proveniência possuem valores correspondentes em seu atributo "*execução*". A ausência deste atributo em um supervértice (superaresta) indica que estes possuem vértices (arestas) presentes em todos os grafos de proveniência. A Figura 25 apresenta um exemplo de como o SGProv agrupa vértices e arestas. A partir dos grafos de proveniência G_{p1} e G_{p2}, o grafo sumário G_s é gerado, onde: o supervértice v_{s1} é criado, pois $Val(v_{10}, elem) = Val(v_{20}, elem) =$ *"atividade"* e $Val(v_{10}, cod) = Val(v_{20}, cod) =$ *"A4"*; o supervértice v_{s2} é criado, pois $Val(v_{11}, elem) = Val(v_{21}, elem) =$ *"entidade"* e $Val(v_{11}, cod) = Val(v_{21}, cod) =$ *"E11"*; e a superaresta e_{s12} é criada, pois existem as arestas (e_1, e_2) entre os vértices v_{s1} e v_{s2}, onde $\{Origem(e_1), Origem(e_2)\} \subseteq Origem(e_{s12})$, $\{Destino(e_1), Destino(e_2)\} \subseteq Destino(e_{12})$ e $Tipo(e_1) = Tipo(e_2) =$ *"Used"*.

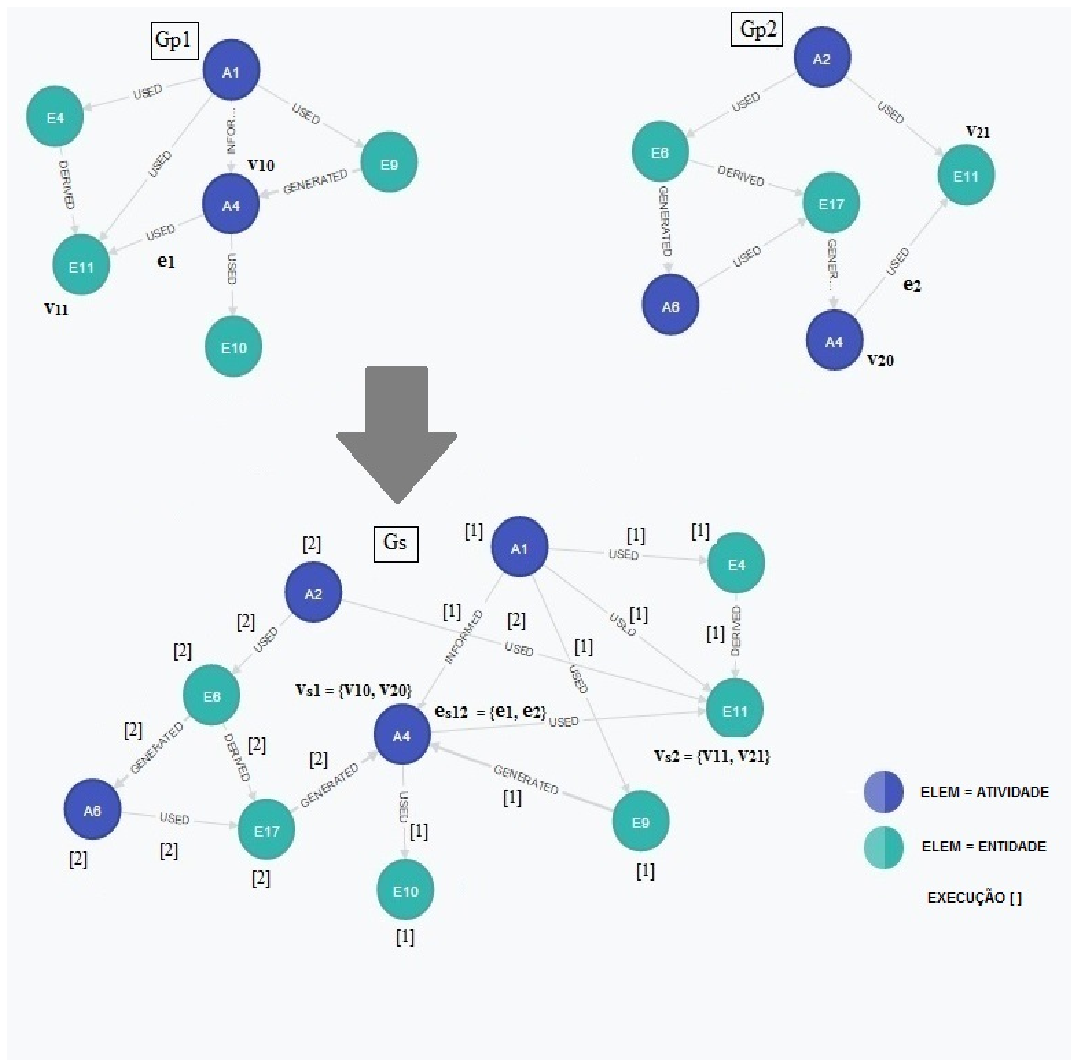


Figura 25. Exemplo de sumarização do SGProv.

Em G_s, $A_s=\{a_{s1}, a_{s2},..., a_{sp}\}$ é o conjunto de atributos "*elem*" e "*cod*" associados aos supervértices, e também, os atributos das superarestas. Uma vez que os atributos dos vértices e seus respectivos valores podem se repetir nos grafos de proveniência, estes são armazenados em um grafo de atributos para eliminar redundâncias no grafo sumário. Um grafo de atributos é definido da seguinte forma.

Definição 8 (Grafo de Atributos): Um grafo de atributos G_a é um subgrafo do grafo sumário e é definido como $G_a = (V_a, E_a, A_a, T_a)$, onde $V_a \subseteq V_s$, $E_a \subseteq E_s$, $A_a \subseteq A_s$ e $T_a \subseteq T_s$. Cada vértice $v_a \in V_a$ é chamado de supervértice do grafo de atributos e cada aresta $e_a \in E_a$ é chamada de superaresta do grafo de atributos, que são definidos da seguinte forma.

Definição 9 (Supervértice do Grafo de Atributos): Um supervértice do grafo de atributos representa um conjunto de atributos dos vértices dos grafos de proveniência, que possuem o mesmo nome, ou um conjunto de valores iguais destes atributos. Se um vértice $v_{ai} \in V_a$ representa um atributo, $Val(v_{ai}, elem) =$ *"atributo"*. Se um vértice representa um valor de um atributo, $Val(v_{ai}, elem) =$ *"valor"*. "*Cod*" é propriamente o nome ou valor do atributo. Assim, *(*$Val(v_{ai}, cod) =$ *nome do atributo)* ou *(*$Val(v_{ai}, cod) =$ *valor do atributo).*

Definição 10 (Superaresta do Grafo de Atributos): Uma superaresta pode ser dos tipos "*atrib*" ou "*val*" $\in T_a$. Uma superaresta do tipo "*atrib*" conecta um supervértice que possui um conjunto de nomes de atributos com supervértices que contêm conjuntos de todos os seus possíveis valores. Assim, se $e_{aj} \in E_a$, e $Tipo(e_{aj}) =$ *"atrib"*, então *(*$Origem(e_{aj}) =$ *nome do atributo)* e *(*$Destino(e_{aj}) =$ *valor do atributo)*. Uma superaresta do tipo "*val*" conecta valores dos atributos com os respectivos supervértices do grafo sumário, que representam uma atividade ou entidade, a que pertencem. Assim, se $e_{aj} \in E_a$ e $\text{Tipo}(e_{aj}) =$ "*val*", então $Origem(e_{aj}) =$ *valor do atributo* e $Destino(e_{aj}) =$ *supervértice do sumário (atividade ou entidade)*. Algumas arestas do tipo "*val*" possuem um atributo "*execução*", que consiste na lista de execuções do *workflow* em que o valor do seu vértice de origem esteve presente em um atributo do seu vértice de destino (atividade ou entidade). A ausência do atributo "*execução*" nestas arestas indica que o valor do atributo e o atributo correspondente estiveram presentes, nos vértices a que pertencem, em todos os grafos de proveniência.

No grafo de atributos, os atributos e os valores dos vértices dos grafos de proveniência são armazenados uma única vez, sem redundância. A Figura 26 apresenta um exemplo de alguns supervértices do grafo sumário da Figura 25 e seu grafo de atributos correspondente.

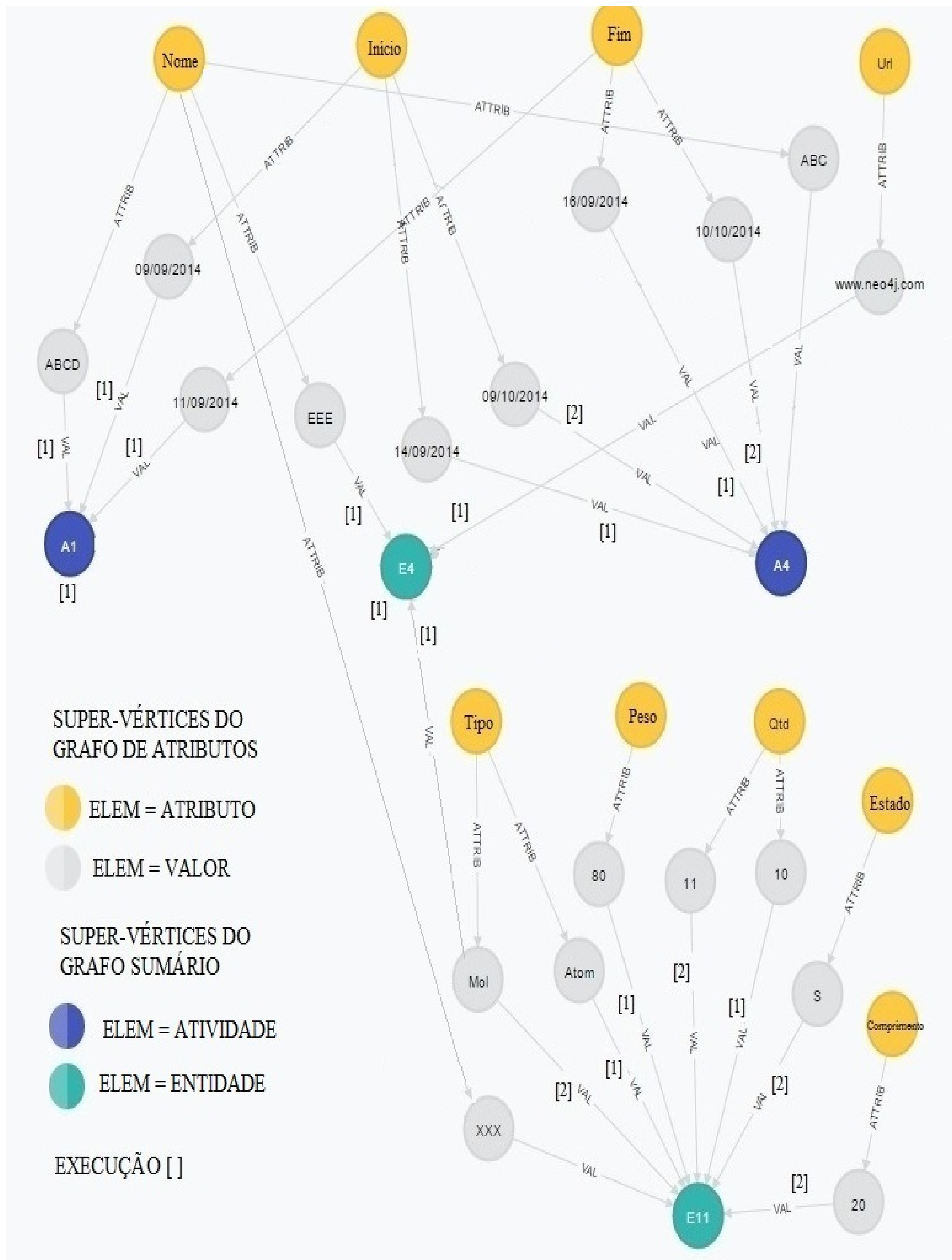


Figura 26. Exemplo de grafo de atributos.

Na Figura 26, os supervértices do grafo de atributo que representam os nomes dos atributos possuem "*cod*" com os valores: "*Nome*", "*Início*", "*Fim*", "*Url*", "*Tipo*", "*Peso*", "*Qtd*", "*Estado*" e "*Comprimento*". Estes supervértices possuem o atributo "*elem = atributo*" (ver legenda do grafo) e são conectados com todos os seus valores por arestas do tipo "*atrib*". Os supervértices do grafo de atributos que representam valores dos atributos possuem seu atributo "*elem = valor*" e são conectados aos supervértices do grafo sumário que representam atividades ou entidades *("cod = A1", "cod = E4", "cod = A4"* e *"cod = E11")* por arestas do tipo "*val*". A atividade "*A1*" e a entidade "*E4*" estão presentes no grafo de proveniência G_{p1} ("*execução=1*"), portanto todos os seus valores de atributos possuem "*execução = 1*" nas suas arestas do tipo "*val*". A atividade "*A4*" e a entidade "*E11*" estão presentes em G_{p1} e G_{p2} ("*execução = 2*"), portanto possuem valores de atributos com "*execução = 1*" e "*execução = 2*". *A4* e *E11* também possuem atributos presentes nos dois grafos de proveniência, G_{p1} e G_{p2}. São eles "*nome = ABC"* e "*nome = XXX"* respectivamente. Portanto, não possuem o atributo "*execução*".

O grafo sumário representa um conjunto de grafos de proveniência resultantes das execuções de um mesmo *workflow*, mas sem redundâncias. Assim, deve ser possível reconstruir os grafos de proveniência a partir do sumário, sem perda de dados, aplicando a operação inversa do sumário. A sumarização é baseada no agrupamento de vértices e arestas correspondentes, portanto a operação inversa da sumarização consiste na "expansão" do grafo sumário (Navlakha *et al.,* 2008) . A operação inversa é definida da seguinte forma.

Definição 11 (Operação Inversa do Sumário): Para cada supervértice $v_{si} \in V_s$, criar vértices do conjunto $\{V_{p1} \cup V_{p2} \cup ... \cup V_{px}\}$ e para cada superaresta $e_{si} \in E_s$, adicionar arestas do conjunto $(E_{p1} \cup E_{p2} \cup ... \cup E_{px})$. A base para aplicar a operação inversa do sumário consiste em consultar no sumário supervértices e superarestas que possuem o mesmo valor para o atributo "*execução*". Assim, para cada valor de "*execução*" é criado um grafo de proveniência.

A hipótese desta tese é que, aplicando o mecanismo de sumarização proposto é possível reduzir o volume de dados de um conjunto de grafos e, assim, melhorar o tempo de processamento das consultas de proveniência. Além disso, com a sumarização a análise dos resultados das consultas se torna mais simples. As consultas podem ser elaboradas com base em um único grafo e produzem resultados sem redundâncias. Sem

a sumarização, seria necessário consultar inúmeros grafos de proveniência e analisar todos os resultados produzidos. Espera-se ainda que ao usar um sistema nativo de armazenamento e consultas de grafos, as consultas aplicadas ao grafo sumário sejam ainda mais eficientes. A seguir, é apresentado um exemplo da aplicação do SGProv.

5.3 Exemplo de Aplicação do SGProv

O SGProv é executado de modo iterativo. O conjunto de grafos de entrada CG_p é sumarizado de modo cumulativo até que este conjunto seja todo varrido de modo exaustivo. Em cada iteração o SGProv recebe como entrada o grafo sumário produzido na iteração anterior e o próximo grafo do conjunto CG_p e produz um novo grafo sumário. Na primeira iteração, como ainda não foi produzido nenhum grafo sumário, o primeiro grafo do conjunto CG_p é considerado o grafo sumário desta iteração e grafo de atributos é inicializado com os atributos deste grafo de proveniência. Na iteração seguinte, o SGProv recebe o grafo sumário produzido na iteração anterior e o próximo grafo do conjunto CG_p e produz um novo grafo sumário. O procedimento continua até que todo o conjunto CG_p tenha sido processado e o grafo sumário final tenha sido produzido. O exemplo, a seguir, ilustra o funcionamento do SGProv. No exemplo, os grafos de proveniência estão no formato da interface Web do Neo4j para visualização e consulta de grafos e seguem a representação do modelo PROV-DM. No exemplo, os vértices dos grafos representam atividades e entidades e as arestas entre eles são dos tipos: *WasInformedBy,* que indica a comunicação entre atividades, onde a atividade de destino usa dados gerados pela atividade de origem; *Used,* que indica a utilização de uma entidade por uma atividade; *WasGeneratedBy*, que indica a produção de uma nova entidade por uma atividade; e *WasDerivedFrom,* que indica a transformação, atualização ou construção de uma nova entidade a partir de outra entidade.

Seja um conjunto de grafos de proveniência de entrada $CG_p = \{G_{p0}, G_{p1}, G_{p2}\}$, onde cada vértice dos grafos possui dois atributos "*elem*" (ver legenda dos grafos) e "*cod*" (valores dentro dos vértices). A Figura 27 ilustra a primeira iteração do exemplo, onde os grafos G_{p0} e o G_{p1} são sumarizados gerando o grafo sumário G_{s1}. Nesta iteração o grafo G_{p0} é considerado o grafo G_{s0}, pois ainda não foi produzido nenhum grafo sumário.

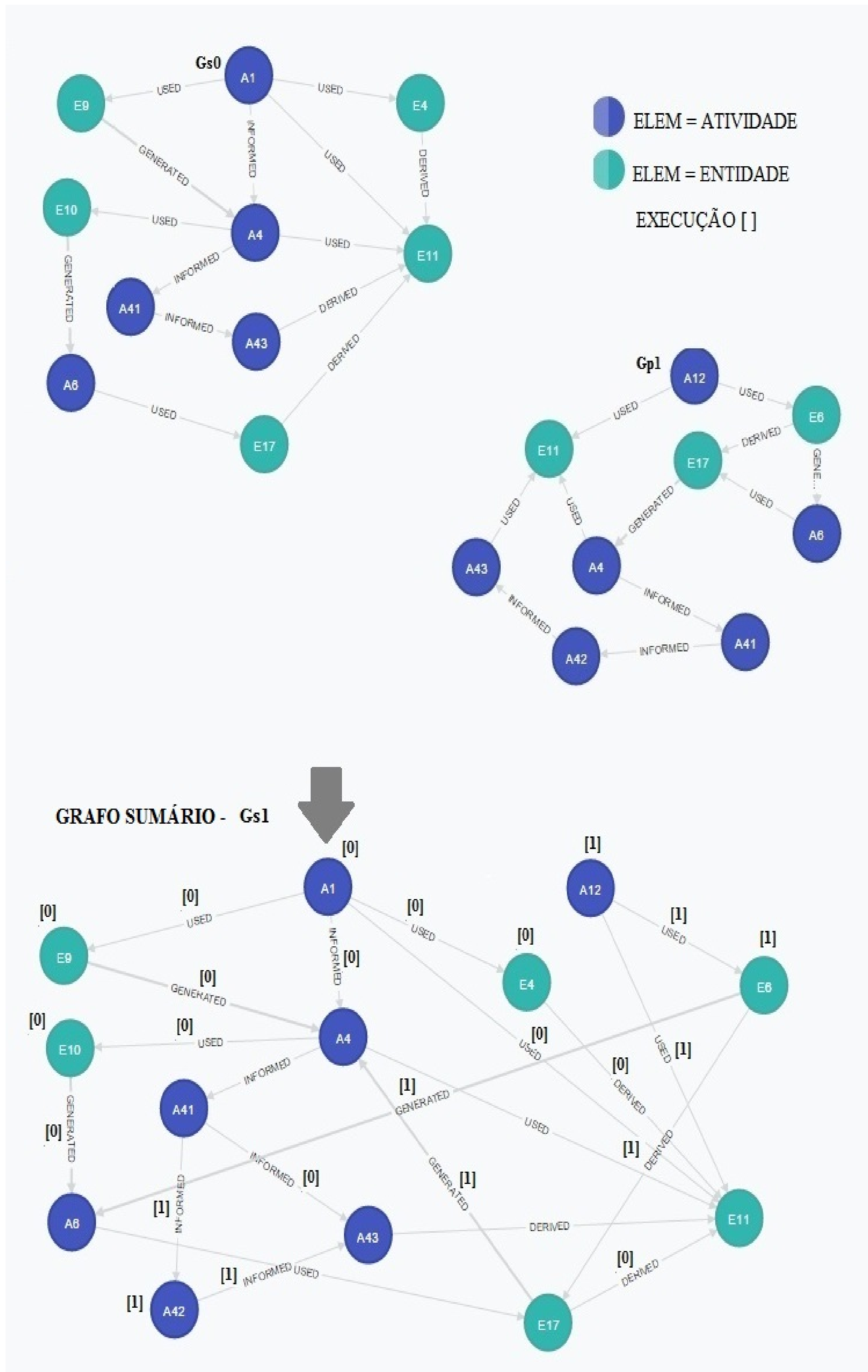


Figura 27. Primeira iteração do SGProv.

No exemplo, o SGProv procede da seguinte forma. O SGProv começa percorrendo os dois grafos procurando: vértices e arestas correspondentes entre G_{p1} e G_{s0} para inseri-los no novo grafo sumário G_{s1}. Na primeira iteração (Figura 27), os vértices correspondentes nos dois grafos representam as atividades com os valores dos atributos "*cod*" iguais a *"A4"*, *"A6"*, *"A41"* e *"A43"*, e também, os vértices que representam as entidades com os valores dos atributos "*cod*" iguais a *"E11"* e *"E17"*. Como são comuns aos dois grafos, estes vértices são inseridos em G_{s1} sem o atributo "*execução*". Arestas destes vértices que são correspondentes nos dois grafos também são inseridas em G_{s1} sem o atributo "*execução*". Por exemplo, a aresta entre os vértices *"A4"* e *"A41"* é comum aos dois grafos. As arestas dos vértices correspondentes, que existem em G_{p1}, mas não têm correspondentes em G_{s0} são inseridas em G_{s1} com o atributo "*execução = 1*", pois pertencem a execução *1* (G_{p1}). Por exemplo, a aresta entre os vértices *"E17"* e *"A4"* existe em G_{p1}, mas não tem correspondente em G_{s0}. Por conseguinte, as arestas que existem em G_{s0}, mas não têm correspondentes em G_{p1} são inseridas em G_{s1} com o atributo "*execução = 0*", pois pertencem a execução *0* (G_{s0}). Por exemplo, a aresta entre os vértices *"E17"* e *"E11"* existe em G_{s0}, mas não tem correspondente em G_{p1}.

Em seguida, os vértices de G_{p1} que não têm correspondentes em G_{s0} e suas arestas são inseridos em G_{s1} com o atributo "*execução = 1*". Por exemplo, os vértices *"A12"* e *"E6"*, assim como a aresta entre eles, existem em G_{p1}, mas não têm correspondentes em G_{s0}. Por último, os vértices de G_{s0} que não têm correspondentes em G_{p1} e suas arestas são inseridos em G_{s1} com o atributo *"execução = 0"*. Por exemplo, os vértices *"A1"* e *"E9"*, assim como a aresta entre eles, existem em G_{s0}, mas não têm correspondentes em G_{p1}.

Em cada passo da iteração, o SGProv também compara os atributos do grafo de atributos de G_{s0} com os atributos dos vértices de G_{p1}, produzindo um novo grafo de atributos em G_{s1}. O SGProv sempre verifica se os atributos e valores já existem no grafo de atributos de G_{s1} antes de inseri-los, para evitar redundâncias no sumário.

Os vértices correspondentes nos dois grafos têm seus atributos e respectivos valores comparados. Atributos iguais com valores iguais são inseridos em G_{s1} e são associados aos respectivos supervértices de G_{s1} através da aresta do tipo "*val*" sem o atributo "*execução*", pois pertencem a ambas execuções (G_{s0} *e* G_{p1}). Se atributos iguais têm valores distintos, ambos os valores são inseridos em G_{s1} e na aresta do tipo "*val*" é

adicionado o atributo *"execução = 0"* se o valor pertencer a G_{s0} ou *"execução = 1"* se pertencer a G_{p1}. Finalmente, atributos não comuns aos dois vértices e seus valores são inseridos no grafo de atributos de G_{s1} e na aresta do tipo *"val"* também é adicionado o atributo *"execução = 0"* se o valor pertencer a G_{s0} ou *"execução = 1"* se pertencer a G_{p1}. A Figura 28 mostra um exemplo de grafo de atributos.

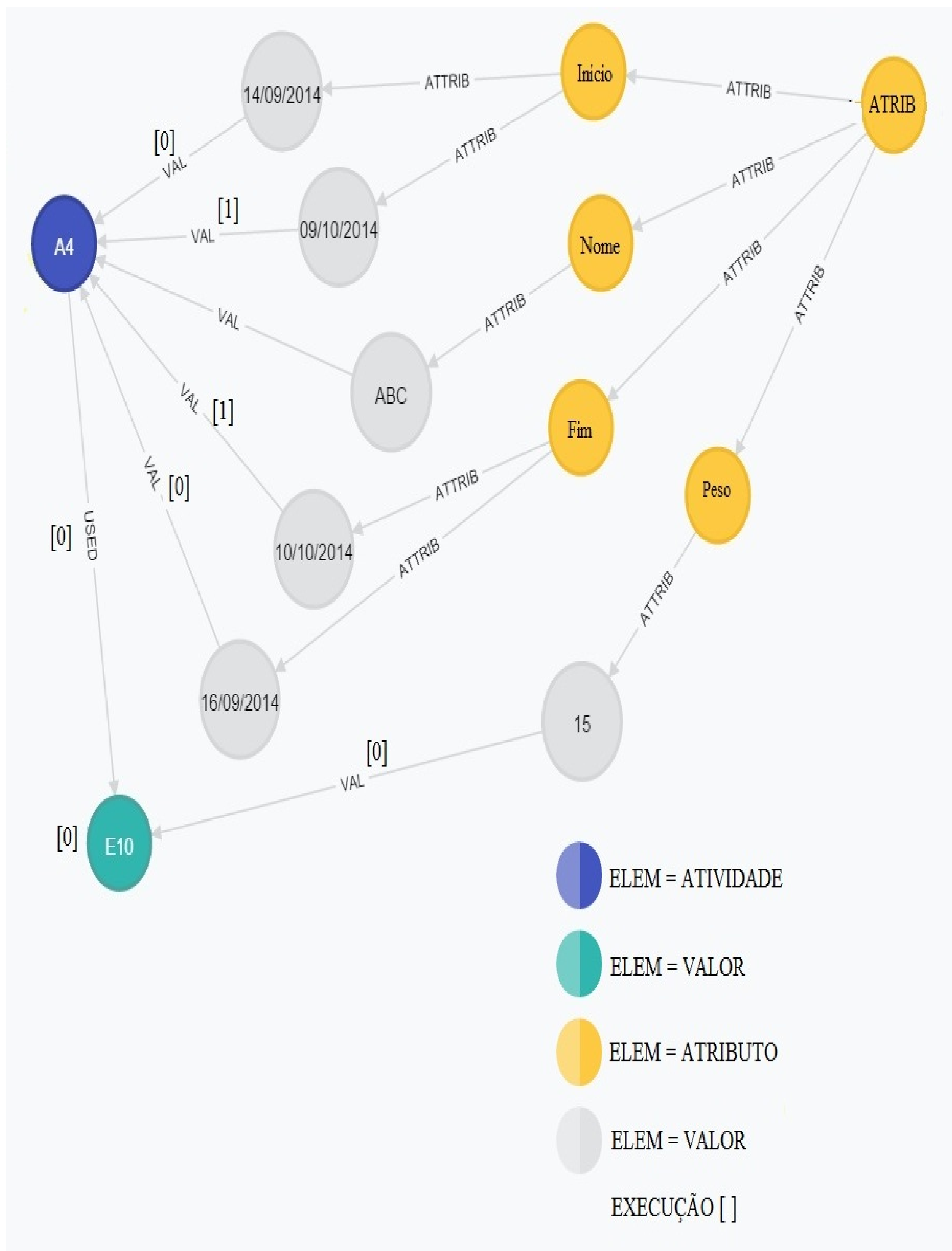


Figura 28. Parte do grafo de atributos de G_{s1}.

O SGProv trata da mesma forma os vértices de G_{p1} sem correspondentes em G_{s0}, assim como os vértices de G_{s0} sem correspondentes em G_{p1}. No primeiro caso, os novos atributos e valores são inseridos em G_{s1} e nas arestas do tipo "*val*" são adicionados atributos "*execução =1*". No segundo caso, nas arestas do tipo "*val*" são adicionados atributos "*execução =0*".

A Figura 28 apresenta os atributos da atividade *"A4"*, que pertence aos grafos G_{s0} e *Gp1*, e da entidade *"E10"*, que pertence ao grafo G_{s0}. A atividade *"A4"* em G_{s0} possui os atributos: "*Início = 14/09/2014*" e "*Fim = 16/09/2014*", que contêm o atributo "*execução = 0*" em suas arestas do tipo "*val*". Possui também o atributo "*Nome = ABC*", que não contém o atributo "*execução*" na sua aresta do tipo "*val*" e, portanto, pertence a todas as execuções (G_{s0} e G_{p1}). A atividade *"A4"* em G_{p1} possui os atributos: "*Início = 09/10/2014*", "*Fim = 10/10/2014*" e "*Nome = ABC*". A entidade *"E9"* em G_{s0} possui um único atributo, "*Peso = 15*".

No final desta iteração, o grafo sumário G_{s1}, incluindo seu grafo de atributos, é gerado contendo todos os vértices, arestas, atributos e valores dos dois grafos G_{s0} e G_{p1}, mas sem redundâncias. A Figura 29 ilustra a segunda iteração do exemplo, onde o grafo G_{s1}, produzido na iteração anterior, e o próximo grafo do conjunto de entrada, G_{p2}, são sumarizados gerando o grafo G_{s2}.

O SGProv procede como na iteração anterior, comparando o grafo G_{p2} com o grafo G_{s1}. Em G_{s2}, os vértices que representam as atividades *"A4"* e *"A6"*, e também a entidade *"E11"*, pertencem a todas as execuções, portanto não possuem o atributo "*execução*". Os vértices *"E6"* e *"E9"* são comuns a G_{s1} e G_{p2}, entretanto o vértice *"E9"* pertence somente às execuções *0* e *2*, enquanto o vértice *"E6"* pertence às execuções *1* e *2*. As arestas em G_{s2} também têm o valor de seu atributo "*execução*" atualizado. Por exemplo, a aresta entre os vértices *"E6"* e *"A6"* é comum ao grafo G_{s1} ("*execução = 1*") e ao grafo G_{p2}. No grafo G_{s2}, esta aresta tem seu atributo "*execução*" atualizado com o valor *2* (G_{p2}). Após todos os grafos do conjunto de entrada serem analisados, o SGProv é finalizado.

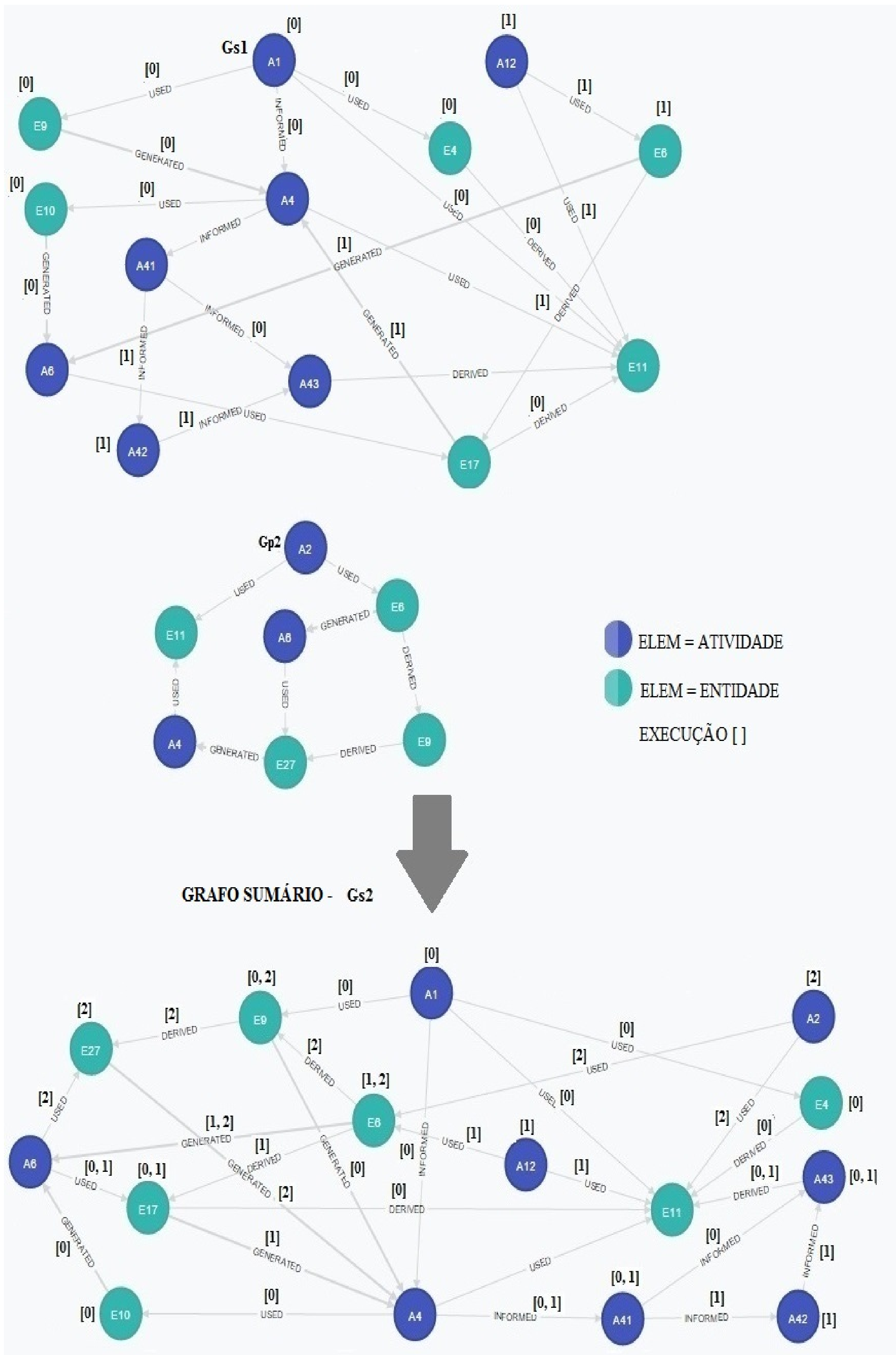


Figura 29. Segunda iteração do mecanismo de sumarização.

Analisando o exemplo, é possível verificara eficiência do SGProv. Na Figura 27, G_{s1} foi produzido a partir da sumarização de G_{p0} e G_{p1}. Estes grafos juntos possuem 19 vértices e 25 arestas. Por outro lado, G_{s1} tem 13 vértices e 21 arestas. Assim, houve uma redução de 31,5% dos vértices e de 16% das arestas, com o custo da introdução de 24 valores de atributo ("*execução*"). Na Figura 29, G_{s2} foi produzido a partir da sumarização dos grafos G_{s1} e G_{p2}. Se a sumarização não tivesse sido aplicada, o volume total de dados corresponderia à soma dos grafos originais G_{p0}, G_{p1} e G_{p2}. Estes três grafos juntos possuem 26 vértices e 33 arestas, enquanto G_{s2} possui 15 vértices e 27 arestas. Assim, vértices e arestas tiveram uma redução de 42,3% e 18% respectivamente, com o custo da introdução de 37 valores de atributo ("*execução*"). A compressão do atributo "*execução*" é discutida no item 6.2 do próximo capítulo. Desta forma, após a aplicação de cada iteração do mecanismo de sumarização, se os grafos de proveniência do conjunto CG_p possuírem muitos vértices e arestas correspondentes, a redução do volume de dados se tornará mais significativa. Após a aplicação de *n-1* iterações do mecanismo de sumarização para *n* grafos de proveniência de entrada, obtém-se um grafo sumário, que contém todos os vértices e arestas dos grafos de entrada originais, mas sem redundâncias. No exemplo, consultando apenas o grafo G_{s2} é possível fazer consultas aos grafos G_{p0} (G_{s0}), G_{p1} e G_{p2}, sem a necessidade de percorrer cada grafo individualmente. Para isso, verifica-se o valor do atributo "*execução*" dos vértices e das arestas do grafo G_{s2}. Assim, os vértices e arestas que não possuem este atributo pertencem a todas as execuções e os que possuem valor igual a: *0* pertencem ao grafo G_{p0} (G_{s0}); *1* pertencem ao grafo G_{p1}; *2* pertencem ao grafo G_{p2}.

Capítulo 6 Desenvolvimento do SGProv

Uma vez que as definições e a intuição de uso do SGProv são apresentadas no capítulo 5, este capítulo apresenta na seção 6.1 o algoritmo do SGProv que transforma as definições do grafo sumário em estruturas de dados e métodos que realizam a sumarização. A seção 6.2 explica como é feita a compressão do atributo "*execução*" que pertence a supervértices e superarestas do grafo sumário. Finalmente, a seção 6.3 detalha como os grafos de proveniência originais são reconstruídos a partir do grafo sumário.

6.1 Algoritmo

Algoritmo: SGProv

```
      Algoritmo SGProv (CGp, Gs)
      Entrada: CGp: { Gp0, Gp1, …, Gpm-1 }, where Gpi=(Vpi, Epi, Api, Tpi), i = 0...m-1 // conjunto de grafos de proveniência
                sum_atrib //atributo usado na sumarização
      Saída:    Gs= (Vs, Es, As, Ts) //grafo sumério
                Ga = (Va, Ea, Aa, Ta) //grafo de atributos
      nExec:=1 // número da execução
      gVértice := [] // lista de vértices do grafo de proveniência não encontrados no sumário
      sVértice := [] // lista de vértices do sumário não encontrados no grafo de proveniência
1     Início
2     Gs := CGp [0], achouVértice := 0
3     Ga := CriaGrafoAtributos()
4     Enquanto nExec < CGp .tamanho faça
5         G := CGp [nExec]  // G = (Vg, Eg, Ag, Tg)
6         achouVértice := 0
7         Para vg ∈ G.Vg do // vg →vértice do grafo proveniência // G.Vg → conjunto de vértices do grafo de proveniência faça
8             Para vs ∈ Gs .Vs do // vs → vértice do sumário // Gs.Vs → conjunto de vertices do sumário faça
9                 Se (Val(vg, elem) = Val(vs, elem) ) and (Val(vg, sum_atrib) = Val(vs, sum_atrib)) então
10                    Val(vs, achou) := 1
11                    Se (∃ vs.execução) então
12                        Val(vs, execução [++]) := nExec
13                    Fim se
14                    achouVértice :=1
15                    AtualizaArestas(G, Gs, vg,, vs)
16                    AtualizaAtributos(G, Ga, vg,, vs)
17                Fim se
18            Fim para
19            Se achouVértice= 0 então
20                gVértice[++] := vg
21            Fim se
22        Fim para
23        Para vs ϵ Gs .Vs faça
24            Se Val(vs, achou) = 1 então
25                Remover(vs.achou)
26            Fim se
27            Senão
28                sVértice[++] := vs
29            Fim senão
30        Fim para
31        Para vg ϵ gVértice faça
```

```
32              Criar(v_s)
33              Val(v_s, sum_atrib) := Val(v_g, sum_atrib)
34              Val(v_s, elem) := Val(v_g, elem)
35              Inserir(v_s, G_s)
36              Val(v_s, execução) := nExec
37              Inserir(v_g.A_g, G_a) // v_g.A_g → atributos do vértice do grafo de proveniência
38              Para (v_g, v_gDest) ϵ E_g faça
39                  v_sDest := v_gDest
40                  Inserir((v_s, v_sDest), G_s)
41                  Val((v_s, v_sDest), execução) := nExec
42              Fim para
43              Para (v_gOrig, v_g) ϵ E_g faça
44                  v_sOrig := v_gOrig
45                  inserir((v_sOrig, v_s), G_s)
46                  Val((v_sOrig, v_s), execução) := nExec
47              Fim para
48          Fim para
49          Para v_s ∈ sVértice faça
50              Se (∄ v_s.execução) então
51                  Val(v_s, execução) := [0, ..., nExec -1]
52              Fim se
53              Para (v_s, v_sDest) ∈ E_s faça
54                  Se (∄ (v_s, v_sDest). execução) então
55                      Val((v_s, v_sDest), execução) := [0, ..., nExec -1]
56                  Fim se
57              Fim para
58              Para (v_sOrig, v_s) ∈ E_s faça
59                  Se(∄ (v_s, v_sOrig). execução) então
60                      Val((v_sOrig, v_s), execução) := [0, ..., nExec -1]
61                  Fim se
62              Fim para
63              Para (v_a, v_s) ∈ E_a faça
64                  Se (∄ (v_a, v_s). execução) então
65                      Val((v_a, v_s), execução) := [0, ..., nExec-1]
66                  Fim se
67              Fim para
68          Fim para
69          nExec = nExec+1
70          Limpar(gVértice)
71          Limpar(sVértice)
72      Fim enquanto
73      Fim
```

Função: Atualiza Arestas

```
        Função AtualizaArestas(G, G_s, v_g, v_s)
        Entrada:  G //grafo de proveniência
                  G_s //grafo sumário
                  v_g //vértice do grafo de proveniência
                  v_s //vértice do grafo sumário
                  sum_atrib //atributo usado na sumarização
74      Início
75          achouArestaD:=0, achouArestaO:=0
76          Para (v_g, v_gDest) ∈ G.E_g faça // G.E_g → conjunto de arestas do grafo de proveniência
77              Para (v_s, v_sDest) ∈ G_s.E_s faça // G_s.E_s → conjunto de arestas do sumário
78                  Se(Val(v_gDest, elem) = Val(v_sDest, elem)) e (Val(
                    v_gDest, sum_atrib) = Val(v_sDest, sum_atrib)) então
79                      Se(∃ (v_s, v_sDest).execução) então
80                          Val((v_s, v_sDest), execução[++]) := nExec
81                      Fim se
82                      achouArestaD :=1
83                      Val((v_s, v_sDest), achou) := 1
```

84 Parar
85 **Fim se**
86 **Fim para**
87 **Se** (achouArestaD= 0) e
88 (∃ {v ϵ $G_s.V_s$ | Val(v, sum_atrib) = Val(v_{gDest}, sum_atrib) e Val(v.elem) = Val(v_{gDest}, elem)}) **então**
89 v_{sDest} := procurar (v, $G_s.V_s$)
90 Inserir((v_s, v_{sDest}), G_s)
91 Val((v_s, v_{sDest}), execution) := nExec
92 **Fim se**
93 **Fim para**
94 **Para** (v_{gOrig}, v_g) ∈ G.E_g **faça**
95 **Para** (v_{sOrig}, v_s) ∈ $G_s.E_s$ **faça**
96 **Se** (Val(v_{gOrig}, elem) = Val(v_{sOrig}, elem)) e (Val(v_{gOrig}, sum_attrib) = Val(v_{sOrig}, param)) **então**
97 **Se** (∃ (v_s, v_{sOrig}).execução) **então**
98 Val((v_s, v_{sOrig}), execução[++]:= nExec
99 **Fim se**
100 achouArestaO :=1
101 Val((v_{sOrig}, v_s), achou) := 1
102 Parar
103 **Fim se**
104 **Fim para**
105 **Se** (achouArestaO = 0) e
106 (∃ {v ϵ $G_s.V_s$ | Val(v, sum_atrib) = Val(v_{gOrig}, sum_atrib) e Val(v.elem) = Val(v_{gOrig}, elem)) **então**
107 v_{sOrig} := procurar (v, $G_s.V_s$)
108 Inserir((v_{sOrig}, v_s), G_s)
109 Val((v_{sOrig}, v_s), execução):= nExec
110 **Fim se**
111 **Fim para**
112 **Para** (v_s, v_{sDest}) ∈ $G_s.E_s$ **faça**
113 **Se** Val((v_s, v_{sDest}), achou) = 1 **então**
114 Remover((v_s, v_{sDest}).achou)
115 **Fim se**
116 **Senão**
117 **Se** (∄ (v_s, v_{sDest}). execução) **então**
118 Val((v_s, v_{sDest}), execução) := [0, ..., nExec -1]
119 **Fim se**
120 **Fim senão**
121 **Fim para**
122 **Para** (v_{sOrig}, v_s) ∈ $G_s.E_s$ **faça**
123 **Se** Val((v_{sOrig}, v_s), achou) = 1 **então**
124 Remover((v_{sOrig}, v_s). achou)
125 **Fim se**
126 **Senão**
127 **Se** (∄ (v_{sOrig}, v_s). execução) **então**
128 Val((v_{sOrig}, v_s), execução) := [0, ..., nExec -1]
129 **Fim se**
130 **Fim senão**
131 **Fim para**
132 **Fim**

Função: Atualiza Atributos

Função AtualizaAtributos(G, G_a, v_g, v_s)
Entrada: G // grafo de proveniência
G_a // grafo de atributos
v_g // vértice do grafo de proveniência
v_s // vértice do grafo sumário
133 **Início**
134 $v_s.A_s$ = {(v_{an0}, v_{av0}), (v_{an1}, v_{av1}), (v_{an2}, v_{av2}),…, (v_{anx}, v_{avx})} ∈ G_a // conjunto de atributos de v_s
135 // v_{an} → nomes dos atributos do vértice v_s
136 // v_{av} → valores dos atributos do vértice v_s
137 $v_g.A_g$ = {$atributo_0$, $atributo_1$, $atributo_2$,..., $atributo_y$} ∈ G.A_g // conjunto de atributos de v_g

```
138        achouS:=0
139        achouA:=0
140        Para (v_g.atributo) ∈ v_g.A_g faça
141            Para (v_an, v_av) ∈ v_s.A_s faça
142                Se((v_g.atributo) = Val(v_an, cod)) e (Val(v_g, atributo) = Val(v_av, cod)) então
143                    Val((v_av, v_s), achou) := 1
144                    achouS := 1
145                    Se(∃ (v_av, v_s).execução) então
146                        Val((v_av, v_s), execução[++]:= nExec
147                    Fim se
148                Fim se
149                Se ((v_g.atributo) = Val(v_an, cod)) e (Val(v_g, atributo) != Val(v_av, cod)) então
150                    Inserir(v_av := Val(v_g, atributo), G_a)
151                    Criar(v_av, v_s)
152                    Val((v_av, v_s), execução) := nExec
153                    Val((v_av, v_s), achou) := 1
154                    achouS := 1
155                Fim se
156            Fim para
157            Se (achouS = 0) então
158                Para (v_an, v_av) ∈ G_a faça
159                    Se((v_g.atributo) = Val(v_an, cod)) e (Val(v_g, atributo) = Val(v_av, cod)) então
160                        Criar(v_av, v_s)
161                        Val((v_av, v_s), achou) := 1
162                        Val((v_av, v_s), execução:= nExec
163                        achouA := 1
164                    Fim se
165                    Se ((v_g.atributo) = Val(v_an, cod)) e (Val(v_g, atributo) != Val(v_av, cod)) então
166                        Inserir(v_av := Val(v_g, atributo), G_a)
167                        Criar(v_av, v_s)
168                        Val((v_av, v_s), execução) := nExec
169                        Val((v_av, v_s), achou) := 1
170                        achouA := 1
171                    Fim se
172                Fim para
173            Fim se
174            Se (achouS = 0) e (achouA = 0) então
175                Inserir(v_an := v_g.atributo, G_a)
176                Inserir(v_av := Val(v_g, atributo), G_a)
177                Criar(v_av, v_s)
178                Val((v_av, v_s), execução) := nExec
179                Val((v_av, v_s), achou) := 1
180            Fim se
181            achouS := 0
182            achouA := 0
183        Fim para
184        Para (v_an, v_av) ∈ v_s.A_s faça
185            Se (Val(v_av, v_s), achou) = 1) então
186                Remover((v_av, v_s).achou)
187            Fim se
188            Senão
189                Se (∄ (v_av, v_s).execution) então
190                    Val(v_av, v_s), execution) := [0, ..., nExec-1]
191                Fim se
192            Fim senão
193        Fim para
194    Fim
```

O algoritmo SGProv recebe como entrada um conjunto de grafos de proveniência (CG_p), que são resultantes das execuções de *workflows* científicos em um

experimento científico computacional, e o atributo (*sum_atrib*) que deve ser usado para agrupar os vértices durante a sumarização. A complexidade do algoritmo não afeta o desempenho das consultas de proveniência, pois após a execução de um *workflow*, os dados de proveniência não sofrem atualização. Desta forma, o grafo sumário é gerado, a partir dos vários grafos da base de proveniência, uma única vez e é armazenado pelo SGProv. O grafo sumário resultante é utilizado para responder as consultas de proveniência sem que seja necessário executar novamente o SGProv. O SGProv é, portanto, um processo executado de forma independente que prepara os dados para serem utilizados pelas consultas de proveniência.

O SGProv é executado iterativamente e, em cada iteração, percorre, compara e sumariza o grafo sumário (G_s) produzido na iteração anterior e o próximo grafo de proveniência do conjunto de entrada (CG_p), produzindo como saída um novo grafo sumário. Este servirá de entrada para a próxima iteração, até que todo o conjunto de grafos de entrada seja comparado. O algoritmo, portanto, é executado em m-1 iterações, onde m é o número total de grafos de proveniência do conjunto original de entrada. Ao término de todas as iterações, é produzido o grafo sumário final.

O algoritmo procede da seguinte forma, na primeira iteração, como ainda não foi criado nenhum grafo sumário, o primeiro grafo do conjunto de entrada é considerado o sumário inicial (linha 2) e seu respectivo grafo de atributos é inicializado (linha 3). Em cada iteração (*nExec)* é selecionado o próximo grafo de proveniência do conjunto CG_p (linha 5).

O algoritmo inicialmente procura vértices correspondentes (v_g e v_s) entre G e G_s (Capítulo 5, 5.1, definição 3). O algoritmo pode também procurar vértices que possuam os mesmos valores para os atributos "*elem*" e "*sum_atrib*", onde este último pode ser qualquer atributo que deve ser utilizado na sumarização, não necessariamente o atributo "*cod*". Isto porque, se algum outro atributo, diferente de "*cod",* aparecer com maior frequência nos vértices dos grafos de proveniência, ele deve ser utilizado na sumarização para obter um grafo sumário mais reduzido (linhas 7 -22). Se o correspondente de v_g é achado em G_s, o v_s correspondente é marcado com o atributo "*achou = 1*" (linha 10). Este atributo só é usado durante a execução do SGProv para marcar os vértices do grafo sumário que existiam também no grafo de proveniência em uma determinada execução. Se v_s possui o atributo "*execução*", este é atualizado com a execução corrente (linhas 11-13). Em seguida, a função **Atualiza Arestas** (linha 15)

compara as arestas de v_g e v_s e faz as atualizações necessárias em G_s (linhas 74-132). Esta função começa procurando as arestas de saída de v_g em v_s (linhas 77-93). Arestas correspondentes têm seus atributos "*execução*" atualizados em G_s, com o número da execução corrente (linhas 79-81). Arestas de v_g sem correspondentes em v_s são inseridas em G_s juntamente com o atributo "*execução*" (linhas 87-92). O mesmo procedimento é feito para as arestas de entrada de v_g e v_s (linhas 94-111). Por último, a função procura as arestas de saída e de entrada correspondentes entre v_s e v_g. As arestas de v_s sem correspondentes em v_g têm seus atributos "*execução*" atualizados, com todas as execuções anteriores menos a atual (linhas 112-131).

Em seguida, a função **Atualiza Atributos** (linha 16) compara os atributos de v_g e v_s e faz as atualizações necessárias em G_a (linhas 133-194). Esta função começa procurando os atributos de v_g em v_s (linhas 140-156). Ao encontrar atributos iguais com valores iguais, atualiza o atributo "*execução*" da aresta que conecta o valor do atributo com v_s (linhas 142-148). Ao encontrar atributos iguais, mas com valores diferentes, insere o valor do atributo em G_a e o conecta a v_s. A esta nova aresta é adicionado o atributo "*execução*" com o valor da execução corrente (linhas 149-155). Se não acha o atributo de v_g em v_s, procura o atributo nos outros vértices do grafo G_a de G_s (linhas 157-173). Se acha atributos iguais com valores iguais, conecta o valor do atributo a v_s e insere o atributo "*execução*" nesta aresta (linhas 159-164). Se acha atributos iguais, mas com valores diferentes, insere o valor do atributo em G_a e o conecta a v_s. A esta nova aresta é adicionado o atributo "*execução*" com o valor da execução corrente (linhas 165-171). Se não acha o atributo em v_s e em G_a, insere o atributo e seu valor em G_a, conecta o valor do atributo a v_s e insere o atributo "*execução*" a esta nova aresta (linhas 174-180). Por fim, atualiza o atributo execução dos atributos e valores que existem em v_s e que não existem em v_g (linhas 184-193).

Após as execuções das funções **Atualiza Arestas** e **Atualiza Atributos** dos vértices correspondentes entre *G* e G_s, o algoritmo continua da seguinte forma. Vértices de G sem correspondentes em G_s são inseridos na lista *gVértice* (linhas 19-21). O algoritmo continua até que todos os vértices de *G* sejam comparados com os vértices de G_s. Na segunda fase do algoritmo, este procura por vértices marcados (achou=1) em G_s e os remove. Vértices que não contêm esta marcação são inseridos na lista *sVértice* (linhas 23-30). Esta lista contém todos os vértices de G_s sem correspondentes em *G*. Em seguida, o algoritmo insere os vértices da lista *gVértice* e suas arestas em G_s e seus

atributos e valores em G_a (linhas 31-48). Os vértices da lista *sVértice*, suas arestas, atributos e valores têm seus atributos "*execução*" atualizados com todas as execuções anteriores menos a atual (linhas 49-68). A variável *nExec* é incrementada (linha 69) e as listas *gVértice* e *sVértice* são reiniciadas (linhas 70-71). O algoritmo continua até que tenham sido processados todos os grafos do conjunto CG_p, produzindo no final o grafo sumário resultante G_s e seu grafo de atributos G_a.

6.2 Compressão do Atributo Execução

O atributo "*execução*" consiste em uma lista dos identificadores das execuções dos *workflows* a que supervértices e superarestas do grafo sumário pertencem. Este atributo é importante para a reconstrução dos grafos de proveniência originais a partir do sumário e para as consultas de proveniência. Assim, dois supervértices v_{s1} and v_{s2} pertencem a uma mesma execução do *workflow* se $\exists\{x \in Val(A_s, execution) \mid x \in Val(v_{s1}, execution) \text{ and } x \in Val(v_{s2}, execution)\}$. A ausência do atributo "*execução*" em um supervértice ou superaresta indica que este pertence a todas as execuções. Entretanto, se um supervértice ou superaresta pertencer a 49.000 execuções do total de 50.000, por exemplo, seu atributo "*execução*" terá uma lista com identificadores de 1 a 49.000. Neste caso, consultas de proveniência que utilizem este atributo podem ter seu desempenho comprometido como, por exemplo, "*procure todos os supervértices e superarestas que pertencem à execução 100*". Outro problema é a dificuldade de visualização dos resultados das consultas pelos cientistas.

Com base nestas questões, o SGProv faz a compressão dos valores do atributo "*execução*" codificando lacunas entre estes números (Xie *et al.* 2011) e cria novos atributos com as lacunas de execução identificadas. A Figura 30 mostra um exemplo do grafo sumário antes da compressão e a Figura 31 mostra o resultado obtido após sua aplicação.

Para realizar a compressão do atributo, o SGProv procura inicialmente lacunas com diferença de 1 unidade entre seus valores, em seguida lacunas com diferença de 2 unidades e assim sucessivamente, até que todos os identificadores de execução do grafo sumário tenham sido analisados. Para cada lacuna identificada é criado um novo atributo com dois valores, o início e o fim da lacuna. O novo atributo é adicionado ao supervértice ou superaresta do sumário e o atributo "*execução*" original é removido.

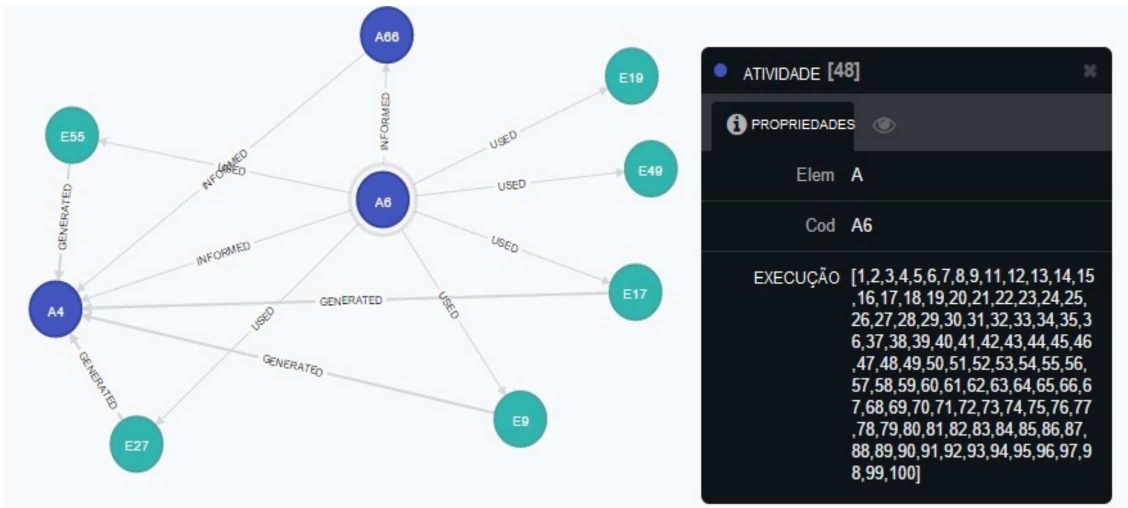


Figura 30. Grafo sumário antes da compressão do atributo "*execução*".

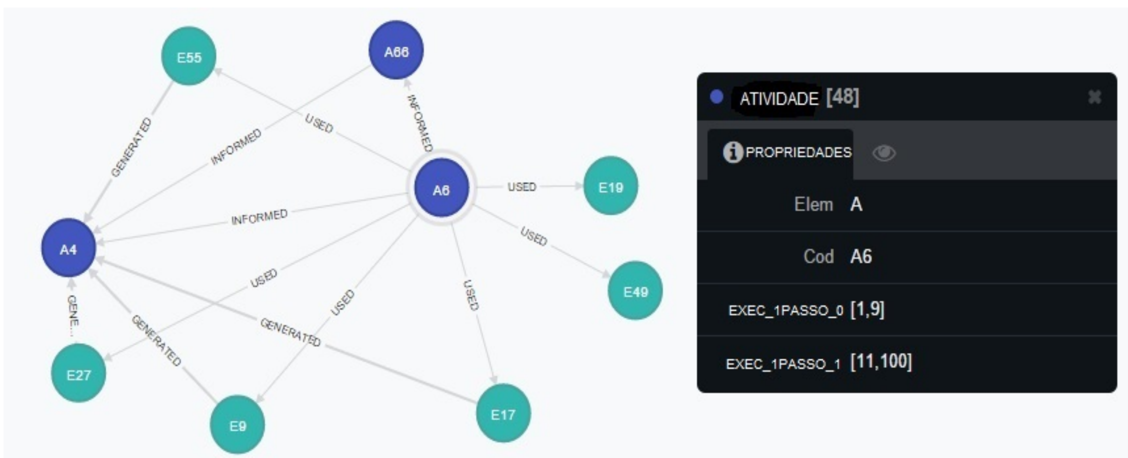


Figura 31. Grafo sumário após a compressão do atributo "*execução*".

Por exemplo, a Figura 30 mostra o atributo "*execução*" antes da compressão do supervértice que representa a atividade *"A6"*. Seu atributo "*execução*" tem lacunas com diferença de 1 unidade entre 1 e 9 e também entre 11 a 100. A Figura 31 mostra os atributos de *"A6"* após a otimização. Dois novos atributos foram criados para as lacunas identificadas: "*Exec_1Passo_0 [1, 9]*" e "*Exec_1Passo_1 [11, 100]*". No atributo "*Exec_1Passo_0 [1, 9]*", "*Exec_*" indica que é um atributo "*execução*", "*1Passo*" indica que a diferença entre a lacuna identificada é de 1 unidade, "*_0*" indica que é a primeira lacuna criada para esta diferença e "*[1, 9]*" indica que a lacuna começa em 1 e vai até 9 ([1, 2, 3, 4, 5, 6, 7, 8, 9]). No atributo "*Exec_1Passo_1 [11, 100]*", "*_1*" indica que é a segunda lacuna criada para esta diferença e "*[11, 100]*" indica que a lacuna começa em 11 e vai até 100 ([11, 12, 13, 14, ..., 100]).

Desta forma, o SGProv reduz consideravelmente a quantidade de valores armazenados em cada atributo "*execução*", criando um grafo sumário mais reduzido do

que o produzido antes da otimização. Além disso, analisando as duas figuras acima é possível notar que a visualização destes atributos na Figura 31 é mais simples do que na Figura 30.

6.3 Obtenção dos Grafos Originais a Partir do Sumário

Na literatura são encontrados trabalhos que tratam a sumarização de grafos com ou sem perda de dados, conforme foi descrito no Capítulo 4, item 4.1. Em relação aos experimentos científicos, a sumarização deve ser feita sem perda de dados, para que os cientistas possam analisar e validar os resultados de seus experimentos. Para mostrar que a sumarização foi feita sem a perda de dados, deve ser possível reconstruir os grafos de proveniência originais a partir do grafo sumário aplicando a este a operação inversa da sumarização. A sumarização é feita a partir do agrupamento de vértices e arestas, a operação inversa é feita expandindo os supervértices e superarestas do sumário (Navlakha *et al.,* 2008).

Enquanto o sumário representa um conjunto de grafos de proveniência de forma resumida, mas contendo todos os vértices, arestas e atributos dos grafos originais, o atributo "*execução*" permite que os grafos de proveniência sejam reconstruídos a partir do sumário. Isto porque este atributo identifica a qual ou quais execuções um vértice, aresta ou atributo pertence. A reconstrução é feita, portanto, consultando os supervértices e superarestas do sumário e o atributo "*execução*" de cada um para criar os respectivos grafos de proveniência. O exemplo a seguir mostra como é feita a operação inversa do sumário.

A Figura 32 mostra um grafo sumário e seu grafo de atributos obtidos a partir da sumarização de dois grafos de proveniência. A operação inversa começa consultando os vértices do sumário, seus atributos e valores, para a quantidade de execuções que geraram o sumário final. Na figura os valores para os atributos "*execução*" são *0* e *1*. Consultando o sumário para o valor de "*execução = 0",* obtêm-se: *A1*, "*nome: ABC*"; *A2*, "*completo: sim*" e "*tempo: 2s*"; *E1*, "*estado: S*", *E2*, "*url: www.test.org*"; e *E3*, sem atributos. Os vértices *A1* e *E1* pertencem a todos os grafos de proveniência e por isso não possuem o atributo "*execução*". A partir deste resultado, a operação inversa reconstrói o grafo de proveniência *0* usado na sumarização criando estes vértices, como ilustra a Figura 33.

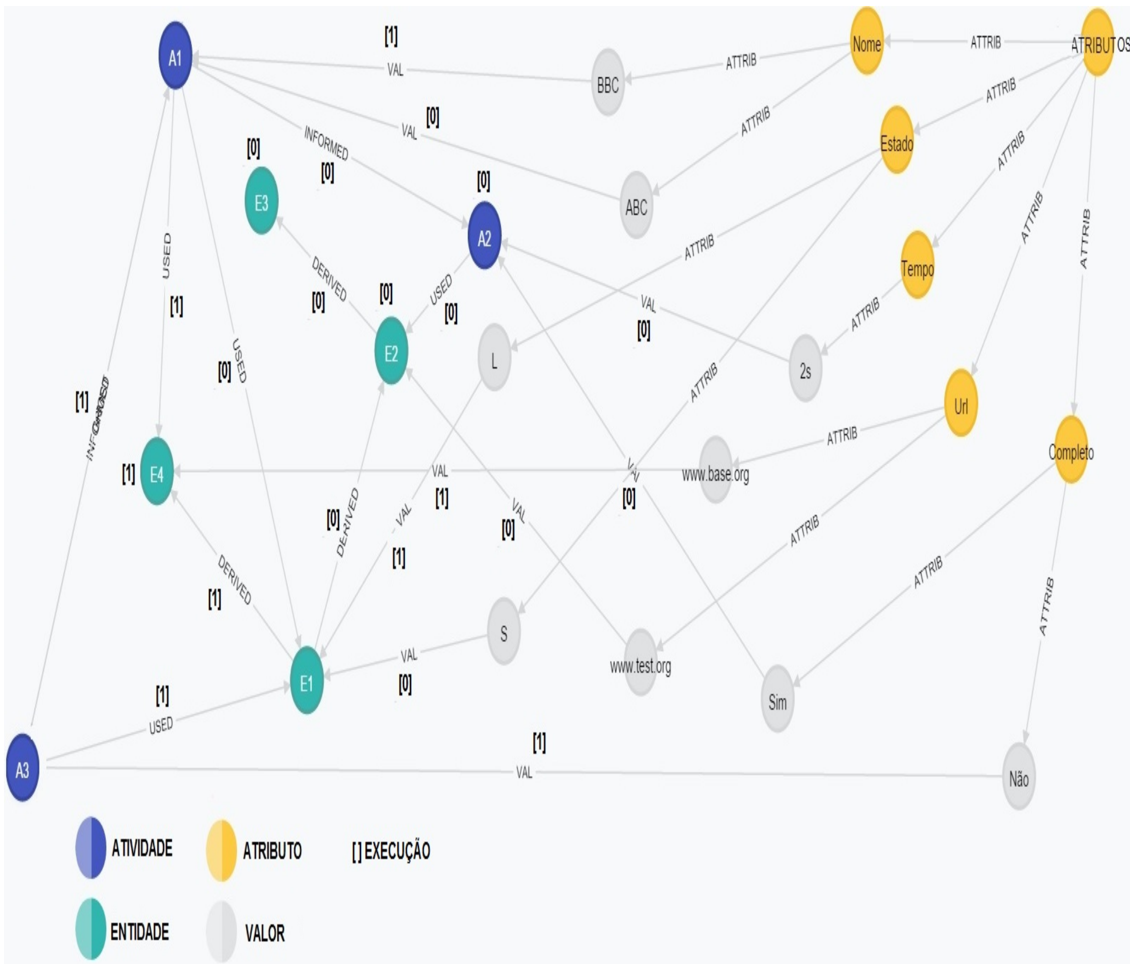


Figura 32. Grafo sumário e seu grafo de atributos.

Figura 33. Vértices do grafo de proveniência 0 (G_{p0}).

Em seguida, a operação inversa consulta as arestas do sumário, que possuem o valor do atributo "*execução = 0*". Consultando a Figura 32, obtêm-se: *(A1, A2), (A1, E1), (A2, E2), (E1, E2) e (E2, E3)*. Estas arestas são inseridas no grafo de proveniência *0* da Figura 33, como ilustra a Figura 34.

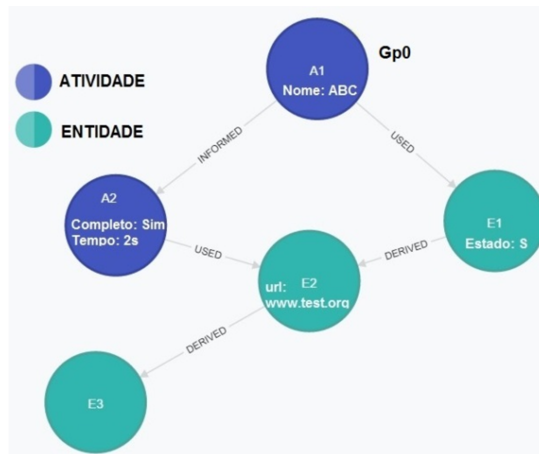


Figura 34. Grafo de proveniência 0 (G_{p0}).

A operação inversa procede da mesma forma para reconstruir o grafo de proveniência *1*, mas desta vez consultando o sumário para o valor de "*execução = 1*". No final da operação inversa são reconstruídos todos os grafos de proveniência usados na sumarização, sem perda de dados (Figura 35). O capítulo seguinte apresenta a avaliação experimental do SGProv, que inclui a aplicação da operação inversa no exemplo apresentado neste capítulo.

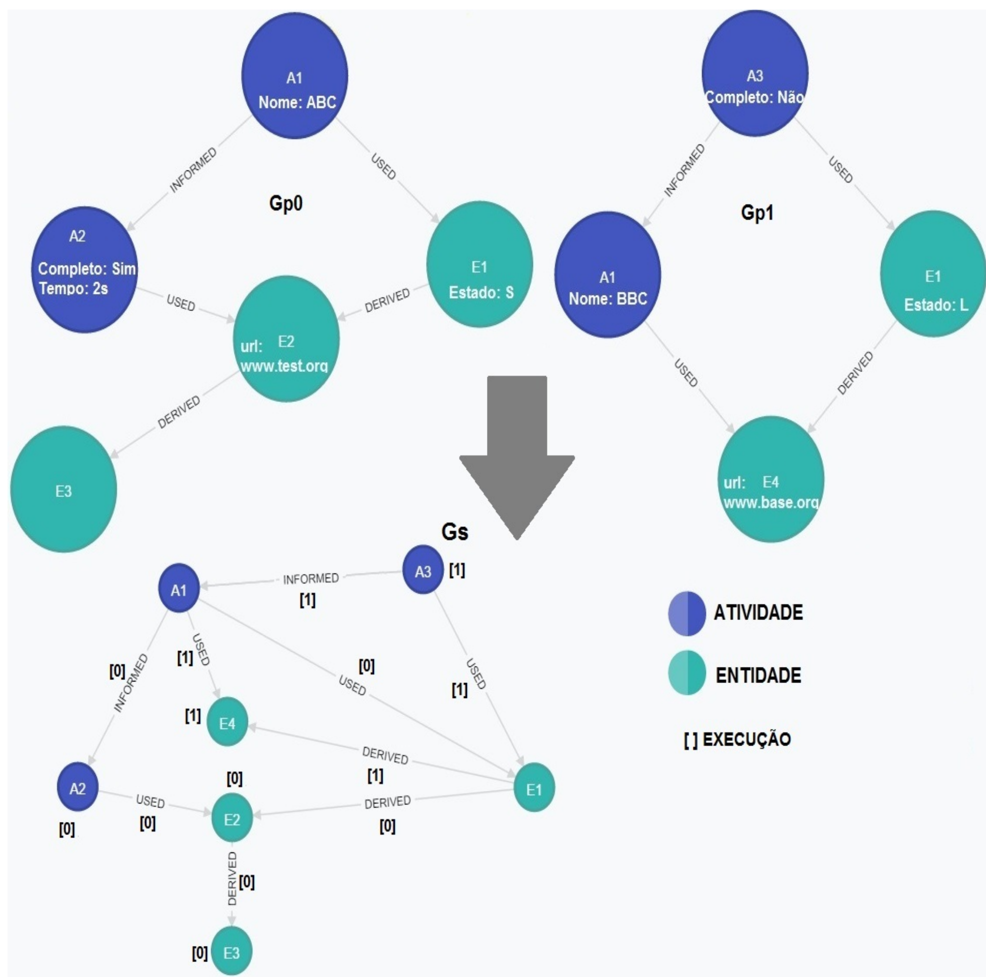


Figura 35. Grafo de proveniência 0 e 1 (G_{p0} e G_{p1}) e grafo sumário (G_s).

Capítulo 7 Avaliação Experimental

O SGProv foi implementado na linguagem Java. Para gerenciar a base de dados, o SGBD utilizado foi o Neo4j 2.1.5, que é orientado a grafos. As consultas de proveniência foram elaboradas em Cypher, linguagem nativa do Neo4j, e submetidas através da API Java. Os experimentos foram realizados em um computador com processador Intel Core i7, 2.4 GHz, 8GB de RAM e HD de 750 GB com 5400 rpm. A Figura 36 ilustra quatro exemplos de grafos de proveniência gerados para a avaliação experimental.

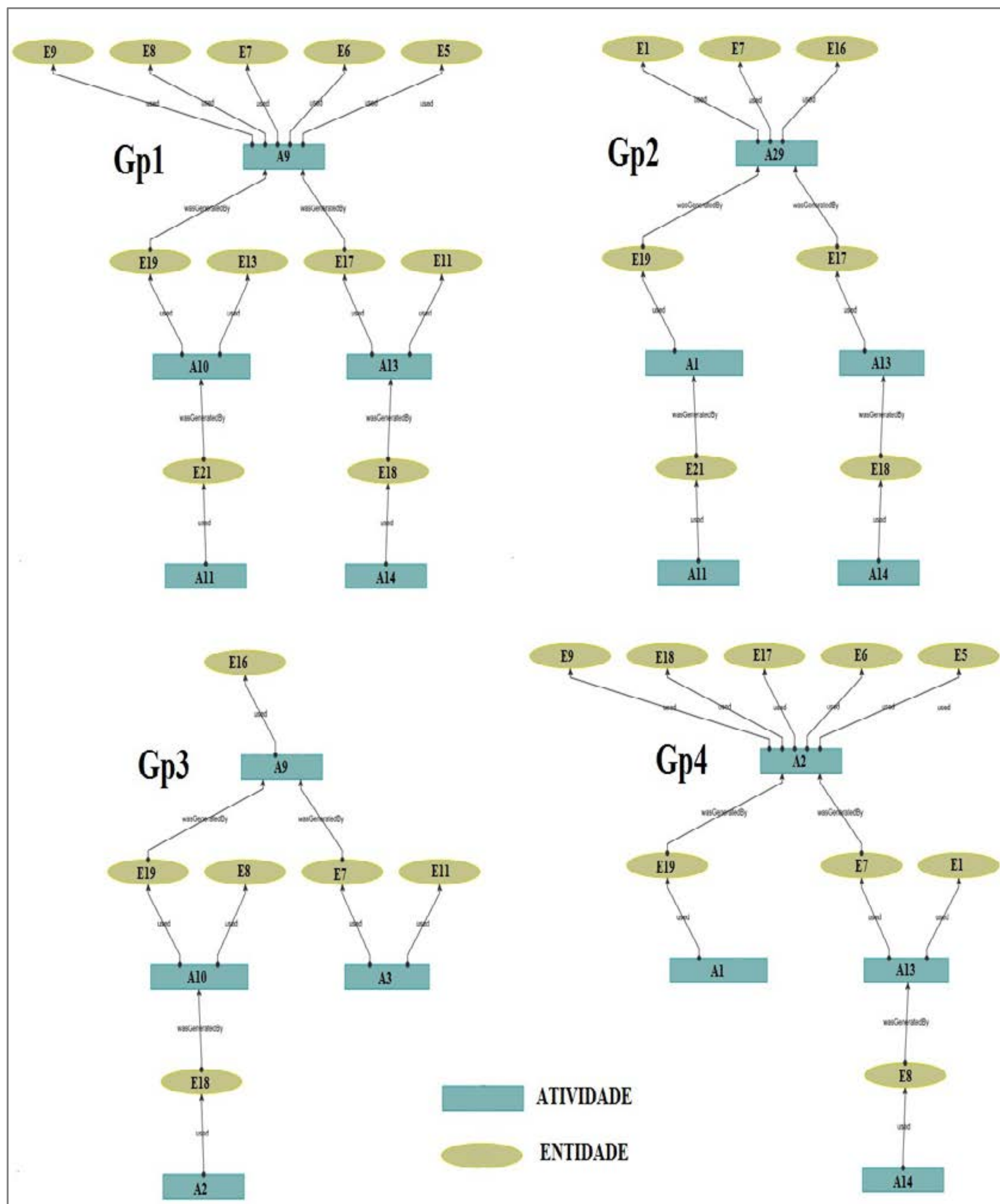


Figura 36. Exemplos de grafos de proveniência gerados para a avaliação experimental.

Para avaliar o desempenho do SGProv foram criadas quatro bases de testes com quantidades diferentes de grafos de proveniência (1.000, 5.000, 10.000 e 50.000 grafos) gerados a partir de execuções de *workflows* sintéticos, baseados na especificação PROV-DM e nos trabalhos de Cuervas-Vicenttín *et al.* (2014) e Chunhyeok *et al.* (2013). Alguns grafos possuem estruturas similares, mas não idênticas, e outros possuem estruturas bastante diferentes (Figura 36). O SGProv foi aplicado nas bases de testes, gerando uma base do sumário para cada base de testes.

Nas bases de testes, um vértice representa uma atividade ou entidade usada pelos cientistas nos seus experimentos. Uma mesma atividade ou entidade pode pertencer a vários grafos de proveniência. Atividades e entidades podem ter atributos. Atividades correspondentes podem ter atributos diferentes ou valores distintos para um mesmo atributo, assim como as entidades. Vinte atividades, trinta entidades e quinze atributos diferentes, com grande variedade de valores, foram definidos para os *workflows* sintéticos, e, em cada execução, algumas atividades foram substituídas por outras similares, novas atividades foram incluídas ou outras foram excluídas.

A Tabela 8 mostra as características dos grafos de proveniência de cada base de testes, de cada grafo sumário e grafo de atributos produzido pelo SGProv, que são armazenados nas denominadas **Bases dos Sumários**, e a taxa aproximada da redução do volume de dados obtida com a sumarização. As bases de dados são gerenciadas pelo Neo4j.

Tabela 8. Características da base de testes e da base do sumário produzida pelo SGProv.

<table>
<tr><th rowspan="2"></th><th colspan="2">Base de Testes</th><th colspan="4">Base do Sumário</th><th colspan="2" rowspan="2">Taxa de Redução</th></tr>
<tr><th colspan="2">Grafo de Proveniência</th><th colspan="2">Grafo Sumário</th><th colspan="2">Grafo de Atributos</th></tr>
<tr><td>Qtd de Grafos</td><td>Vértices</td><td>Arestas</td><td>Vértices</td><td>Arestas</td><td>Atributos</td><td>Valores</td><td>Vértices</td><td>Arestas</td></tr>
<tr><td>1.000</td><td>6.510</td><td>8.013</td><td rowspan="4">50</td><td rowspan="4">100</td><td rowspan="4">15</td><td rowspan="4">1.000</td><td>99,23%</td><td>98,75%</td></tr>
<tr><td>5.000</td><td>32.510</td><td>40.013</td><td>99,84%</td><td>99,75%</td></tr>
<tr><td>10.000</td><td>68.010</td><td>80.113</td><td>99,92%</td><td>99,87%</td></tr>
<tr><td>50.000</td><td>323.510</td><td>398.263</td><td>99,98%</td><td>99,97%</td></tr>
</table>

Uma vez que, considerando todos os grafos, há apenas 20 atividades e 30 entidades diferentes, onde cada uma é representada por um vértice, o grafo sumário é composto por 50 vértices. Isto porque o SGProv agrupa vértices correspondentes que representam uma mesma atividade (ou entidade). Como as dependências (arestas) entre os vértices também podem se repetir em vários grafos de proveniência, o SGProv agrupa arestas correspondentes, ou seja, que possuem o mesmo tipo, e também, a

mesma atividade (ou entidade) de origem e de destino, produzindo um número menor de arestas no grafo sumário.

Em relação ao espaço de armazenamento em disco, as bases dos sumários ocupam muito mais espaço do que as bases de testes. Isto pode ser explicado pelo tamanho do atributo "*execução*" dos supervértices e superarestas do grafo sumário, que pode conter milhares de entradas em sua lista. Após realizar a compressão do atributo "*execução*" (seção 6.2), os tamanhos das bases dos sumários foram consideravelmente reduzidos. Neste experimento, uma base do sumário gerada pelo SGProv antes das aplicação da compressão do atributo "*execução*" é denominadas **Base do sumário sem Compressão**.

A Tabela 9 mostra os tamanhos em disco de cada base de testes, base do sumário sem compressão, base do sumário e a taxa aproximada da redução do espaço de armazenamento em disco obtida com a compressão.

Tabela 9. Tamanhos em disco das bases de dados.

	Base de Testes	Base do Sumário sem Compressão	Base do Sumário	Taxa de Redução	
				Base de Testes	Base do Sumário
1.000	11,8 MB	25,5 MB	8,24 MB	30,17%	67,69%
5.000	27,2 MB	764 MB	14,2 MB	47,79%	98,14%
10.000	52,6 MB	2,5 GB	19,3 MB	63,30%	99,25%
50.000	148 MB	74,9 GB	47,8 MB	67,70%	99,94%

Em seguida, na seção 7.1, são descritas as consultas de proveniência aplicadas na avaliação experimental. Na seção 7.2 são apresentados e avaliados os resultados obtidos nas consultas. A última seção deste capítulo (7.3) mostra a aplicação da operação inversa do sumário nas bases do sumário.

7.1 Consultas de Proveniência

Os cientistas, frequentemente, precisam consultar a base de proveniência para obter informações sobre seus experimentos científicos como, por exemplo, o relacionamento entre a produção e consumo dos dados pelos processos, ou o histórico de derivação dos dados e resultados intermediários obtidos. Desta forma, podem interpretar, entender, avaliar e validar os resultados de seus experimentos, e também, detectar possíveis erros (Gadelha e Mattoso, 2011).

Consultas típicas de proveniência, definidas por Woodman *et al.* (2011) e Gadelha e Mattoso (2011), que foram descritas no Capítulo 2, foram executadas sobre as bases de testes e as bases dos sumários para avaliar a variação do tempo de processamento de cada consulta em ambos os casos. As consultas foram formuladas na linguagem Cypher e foram submetidas ao Neo4j por programas Java através de sua API.

Algumas consultas possuem sintaxes diferentes, quando submetidas às bases de testes e às bases dos sumários, pois estas bases possuem estruturas diferentes. Nestes casos, a versão "a" é submetida às bases de testes e a versão "b", às bases dos sumários. Por exemplo, para consultar um atributo nas bases dos sumários é preciso percorrer o grafo de atributos, enquanto nas bases de testes os atributos são obtidos diretamente a partir dos vértices e arestas. Além disso, as consultas submetidas às bases dos sumários podem retornar o número da execução a que cada supervértice ou superaresta pertence bastando, para isso, incluir, na especificação dos dados retornados pelas consultas, o atributo "*execução*" e os atributos derivados deste após a compressão. A seguir são apresentadas as consultas utilizadas nos experimentos.

Consulta 1 (C1): Obter todas as atividades dos grafos e seus descendentes.

```
MATCH (n:activity)-[r:used|informed]->(m)
RETURN distinct n.cod, type r, m.cod
ORDER by n.cod
```

Esta consulta percorre todos os vértices dos grafos, retornando todas as atividades que produziram entidades para outras atividades e todas as entidades que foram consumidas por cada atividade. Este é um exemplo de consulta de linhagem, que descreve os caminhos de derivação dos dados e os resultados obtidos.

Consulta 2 (C2): Obter os descendentes diretos e indiretos da atividade "A4".

```
MATCH (n1:activity{cod:'A4'})-[r1:informed|used]->(n2)-[r2:informed|used|generated|derived]->(n3)
RETURN distinct n1.cod, type(r1), n2.cod as desc, type(r2), n3.cod as desc_desc
```

Esta é a consulta mais genérica de proveniência e é considerada uma linha de base de consultas que todos os sistemas gerenciadores de proveniência deveriam ser capazes de responder (*Woodman et al.,* 2011). A consulta retorna todos os descendentes diretos e indiretos da atividade que possui o atributo "*cod*" igual a "*A4*".

Consulta 3 (C3): Obter o menor caminho entre as atividades "A6" e "A4" e quais atividades consumiram entidades produzidas pela atividade "A4".

```
a) MATCH p1=shortestpath((n1:activity)-[r1*]->(n2:activity)),
          p2=(n2:activity{cod:'A4'})-[r:informed]->(n3:activity)
   WHERE (n1.cod='A6' and n2.cod='A4')
   WITH min(length(p1)) as min, collect(p1) as path, p2
   RETURN distinct min, path, nodes(p2) as consumed
```

```
b) MATCH p1=shortestpath((n1:activity{cod:'A6'})-[*]->(n2:activity{cod:'A4'})),
   p2=(n2:activity{cod:'A4'})-[r:informed]->(n3:activity)
   RETURN length(p1), nodes(p1) as path, nodes(p2) as consumed
```

Esta consulta procura o menor caminho de derivação entre as atividades "*A6*" e "*A4*". Em seguida, procura caminhos entre a atividade "*A4*" e outras atividades que consumiram entidades produzidas por "*A4*". A consulta retorna o tamanho do caminho entre as atividades "*A6*" e "*A4*", as atividades e arestas presentes neste caminho e as atividades de destino nos caminhos de "*A4*". Este é um caso em que duas sintaxes diferentes precisam ser usadas, de acordo com a base consultada.

Consulta 4 (C4): Obter as atividades e entidades que contribuíram para a geração da entidade "E9".

```
MATCH p= (e:entity{cod: 'E9'})-[:derived|generated*]->(n)
WITH distinct p
RETURN (extract(n in nodes(p)| n.cod)) as path, (extract(r in relationships(p)|r)) as rel
```

Esta consulta é usada para explicar como a entidade "*E9*" foi gerada durante o experimento. A consulta retorna todas as atividades e entidades que contribuíram diretamente ou indiretamente para gerar a entidade "*E9*". Esta consulta é muito importante para os cientistas que precisam rastrear a causa de algum erro encontrado na entidade resultante (Woodman *et al.* 2011).

Consulta 5 (C5): Obter as dados gerados e consumidos pela atividade "A6".

```
MATCH (e1:entity)-[r1:generated]->(a:activity{cod:'A6'})
RETURN distinct e1.cod as e, type(r1) as r
UNION
MATCH (a:activity{cod:'A6'})-[r2:used]->(e2:entity)
RETURN distinct e2.cod as e, type(r2) as r
```

Esta consulta procura entidades que foram geradas e consumidas pela atividade "*A6*". A consulta é útil para o cientista que precisa comparar os resultados produzidos por "*A6*" para cada entidade consumida. Esta consulta pode ser também aplicada em uma atividade similar a "*A6*" para comparar os resultados produzidos por cada atividade.

Consulta 6 (C6): Obter os atributos da atividade "A6".

```
a) MATCH (a:activity {cod: 'A6'})
   RETURN collect(a)
```

```
b) MATCH (at:atribute87)-[r1:attrib]->(v:values)-[r2:val]->(a:activity {cod: 'A6'})
   RETURN at.elem as atribute, v.elem as value
```

Esta consulta procura todos os atributos e seus valores que pertencem à atividade "*A6*". Esta consulta é importante para o experimento porque suas sintaxes são muito diferentes nas versões "*a*" e "*b*". A versão "*a*" obtém os atributos diretamente a partir dos vértices, enquanto a versão "*b*" precisa percorrer os grafos de atributos das bases dos sumários.

Consulta 7 (C7): Obter todas as atividades, entidades e arestas do grafo.

```
MATCH (n)-[r:informed|used|derived|generated]-(m)
RETURN distinct n, r, m
```

Esta consulta busca todos os vértices e arestas das bases de testes e das bases dos sumários. A consulta retorna todos os resultados produzidos em cada execução do *workflow*. Por isso, é importante para os cientistas que precisam comparar estes resultados.

7.2 Avaliação dos Resultados Obtidos nas Consultas

Cada consulta foi executada 20 vezes, os tempos das primeiras 10 execuções foram descartados e a média do tempo de processamento de cada uma foi calculada com base nas últimas 10 execuções. A Tabela 10 mostra a média do tempo de processamento obtido em cada consulta executada sobre as bases de testes e as bases dos sumários. O Neo4j permite que sejam criados índices para vértices, arestas e suas propriedades (atributos). Os tempos da Tabela 10 foram obtidos com a utilização de índices.

A Tabela 11 mostra a redução aproximada no tempo de processamento de cada consulta comparando os tempos obtidos na Tabela 10. Por exemplo, na base de testes com 1.000 grafos o tempo de processamento da consulta 1 (*C1*) foi 2.177 ms, enquanto que na base do sumário correspondente a mesma consulta durou 900 ms. Neste caso, a consulta *C1* teve uma redução de 58,65% no seu tempo de processamento.

Tabela 10. Média do tempo de processamento das consultas.

Consulta	1.000 grafos		5.000 grafos		10.000 grafos		50.000 grafos	
	Tempo (ms)		Tempo (ms)		Tempo (ms)		Tempo (ms)	
	Base de Testes	Base do sumário	Base de Testes	Base do sumário	Base de Testes	Base do sumário	Base de Testes	Base do sumário
C1	2.177	900	3.473	925	3.972	930	10.000	943
C2	2.335	923	3.623	935	4.483	967	10.666	995
C3.a	56.718	-	4.698.752	-	36.720.620	-	95.040.000	-
C3.b	-	1.000	-	1.026	-	1.088	-	1.117
C4	3.202	1.506	4.650	1.512	5.766	1.526	18.756	1.536
C5	2.832	1.012	4.198	1.039	4.797	1.046	14.443	1.059
C6.a	1.668	-	3.494	-	4.571	-	33.475	-
C6.b	-	968	-	1.448	-	1.863	-	2.233
C7	3.556	1.157	8.335	1.365	15.740	1.662	141.564	2.119

Tabela 11. Redução aproximada do tempo de processamento das consultas.

	Redução Aproximada			
	1.000 Grafos	5.000 Grafos	10.000 Grafos	50.000 Grafos
C1	58,65	73,36%	76,58%	90,57%
C2	60,47%	74,19%	78,42%	90,67%
C3	98,21%	99,99%	99,99%	99,99%
C4	52,97%	67,49%	73,53%	91,81%
C5	64,26%	75,25%	78,20%	92,66%
C6	41,97%	58,55%	59,24%	90,64%
C7	67,46%	80,05%	85,83%	97,63%

Todas as sete consultas executadas sobre as bases dos sumários tiveram um tempo de processamento médio bem inferior às suas equivalentes executadas sobre as bases de testes. Isto porque, o volume de cada base do sumário é muito menor do que o volume da base de testes correspondente. Além disso, as consultas podem ser respondidas usando somente as bases dos sumários, sem a necessidade de reconstruir os grafos de proveniência das bases de testes, ou de fazer a descompressão em memória do atributo "*execução*". Por outro lado, as consultas executadas sobre as bases de testes precisam percorrer cada grafo de proveniência desta base individualmente.

A consulta *C3* foi a que apresentou maior redução no seu tempo de processamento (99,99%). Esta redução pode ser explicada pela existência das atividades *A6* e *A4* em quase todos os grafos de proveniência das bases de testes. Assim, a consulta precisa procurar, em cada grafo, todos os caminhos entre *A6* e *A4* para achar o menor caminho entre estas atividades e também todas as arestas de saída de *A4*. Por outro lado,

nas bases dos sumários todas as atividades *A6* são agrupadas em um supervértice, assim com as atividades *A4*.

A consulta *C6* também apresentou grande redução no seu tempo de processamento (entre 41,97% e 90,64%), mesmo sendo preciso atravessar os grafos de atributos para responder esta consulta nas bases dos sumários. Esta redução pode ser explicada pela existência da atividade *A6* em quase todos os grafos de proveniência das bases de testes. Além disso, esta consulta retorna um grande volume de dados (atributos e valores repetidos) nas bases de testes, enquanto nas bases dos sumários os atributos e valores são armazenados uma única vez nos grafos de atributos.

Os tempos de processamentos de uma mesma consulta executada sobre as quatro bases do sumário foram aproximados. Por exemplo, a consulta *C1* obteve os tempos 900 ms, 925 ms, 930 ms e 943 ms, para 1.000, 5.000, 10.000 e 50.000 grafos respectivamente. Isto porque, independente do número de grafos de proveniência sumarizados, todas as bases do sumário geradas possuem o mesmo número de vértices (50) e arestas (100), como foi mostrado na Tabela 8.

A redução obtida no tempo de processamento das consultas aplicadas nas bases dos sumários (Tabela 11) evidencia o potencial do SGProv. Aumentando o volume de grafos nas bases de testes também aumentará o tempo de processamento das consultas executadas sobre esta base e mais significativa será a redução obtida com a utilização das bases dos sumários para a execução das consultas. Uma vez que o grafo sumário representa todos os grafos de proveniência da base de testes, mas sem dados redundantes, as consultas podem ser respondidas somente com base neste grafo. Nos experimentos, o tempo médio de geração do grafo sumário foi de 16 minutos. Mesmo que o tempo de geração do sumário seja maior do que o obtido no experimento, a redução obtida no tempo de processamento de consultas, como a *C3*, compensaria o tempo gasto com a sumarização. Além disso, o grafo sumário é gerado uma única vez e, em seguida, pode ser consultado diversas vezes, pois os dados de proveniência não sofrem atualizações.

As consultas foram executadas nas bases dos sumários geradas pelo SGProv após a compressão do atributo "*execução*". Se estas fossem executadas nas bases dos sumários sem compressão algumas consultas provavelmente teriam o tempo de processamento comprometido. Por exemplo, uma variação de *C1* (chama da de consulta

8 ou *C8*), que retorna todas as propriedades dos vértices, incluindo o atributo "*execução*" (Tabela 12).

Tabela 12. Tempo de processamento da consulta 8 (C8).

Consulta 8 (C8)	Obter todos os atributos das atividades dos grafos e dos seus descendentes.		
	MATCH (n:activity)-[r:used|informed]->(m) RETURN n, type(r), m order by n.cod		
Bases de Testes	Tempo (ms)		
	Sumário sem Compressão	Sumário	Redução Aproximada
1.000 grafos	1.265	930	26,48%
5.000 grafos	2.045	949	53,59%
10.000 grafos	4.002	965	75,89%
50.000 grafos	24.102	974	95,96%

Os tempos de processamento da consulta *C8* executada sobre as bases dos sumários sem compressão são consideravelmente maiores do que os tempos obtidos nas bases dos sumários. Isto pode ser explicado pela quantidade de dados que a consulta retorna, pois o atributo "*execução*" pode possuir muitos valores na sua lista. A Figura 37 mostra o resultado da consulta 8 (*C8)* para a atividade *A1* na base do sumário sem compressão ("*a*"), gerada pelo SGProv após a sumarização da base de testes com 1.000 grafos de proveniência, e na base do sumário ("*b*"). No item "*a"* da figura os valores do atributo "*execução*" foram abreviados para melhor visualização.

a) RESULTADO DA CONSULTA 8 APLICADA NA BASE DO SUMÁRIO SEM COMPRESSÃO

```
| n                                       | type(r)    |
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "USED"     | Node[3]{Elem:"E",Cod:"E4",Execution:[0]
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "USED"     | Node[6]{Elem:"E",Cod:"E11",Execution:[0,1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,21,22,23,...,1000]
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "USED"     | Node[4]{Elem:"E",Cod:"E9",Execution:[0,6|
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "INFORMED" | Node[48]{Elem:"A",Cod:"A6",Execution:[1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,21,22,23,24,...,1000]
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "INFORMED" | Node[2]{Elem:"A",Cod:"A4"
```

b) RESULTADO DA CONSULTA 8 APLICADA NA BASE DO SUMÁRIO

```
+-----------------------------------------------------------------------------------------------------------------------+
| n                                          | type(r)    | m                                                                 |
+-----------------------------------------------------------------------------------------------------------------------+
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "USED"     | Node[3]{Elem:"E",Cod:"E4",Execution:[0]}                          |
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "USED"     | Node[6]{Elem:"E",Cod:"E11",Exec_1Gap_0:[0,9],Exec_1Gap_1:[11,1000]} |
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "USED"     | Node[4]{Elem:"E",Cod:"E9",Execution:[0,6]}                        |
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "INFORMED" | Node[48]{Elem:"A",Cod:"A6",Exec_1Gap_0:[1,9],Exec_1Gap_1:[11,1000]} |
| Node[1]{Elem:"A",Cod:"A1",Execution:[0,7]} | "INFORMED" | Node[2]{Elem:"A",Cod:"A4"}                                        |
+-----------------------------------------------------------------------------------------------------------------------+
```

Figura 37. Resultado da consulta 8 na base do sumário sem compressão e na base do sumário.

7.3 Aplicação da Operação Inversa do Sumário

A operação inversa do sumário foi aplicada em todas as bases dos sumários sem compressão e os grafos de proveniência originais das bases de testes foram corretamente reconstruídos. Em seguida, as consultas de proveniência foram executadas sobre as bases de testes originais e as bases reconstruídas para comparar os resultados obtidos, que foram correspondentes. A operação inversa do sumário é baseada nas consultas em Cypher "*a*" e "*b*" da Figura 38 incorporadas em um programa Java.

```
a) MATCH (n) <-[v:VAL]-(value)<-[at:ATTRIB]-(attribute)
   WHERE (n.Elem='A' OR n.Elem='E') AND
             (nExec in n.Execution OR NOT HAS(n.Execution)) AND
             (nExec in v.Execution OR NOT HAS(v.Execution))
   RETURN n, attribute, value
   ORDER BY n.Cod

b) MATCH (n)-[r:USED|DERIVED|GENERATED|INFORMED]->(m)
   WHERE (nExec in n.Execution OR NOT HAS (n.Execution)) AND
            (nExec in r.Execution OR NOT HAS (r.Execution)) AND
            (nExec in m.Execution OR NOT HAS (m.Execution))
    RETURN n, r, m
```

Figura 38. Consultas usadas na operação inversa do sumário.

A consulta "*a*" retorna os supervértices, seus atributos e respectivos valores, que pertencem a uma execução igual ao valor passado em "*nExec*". A consulta "*b*" retorna todas as superarestas do sumário que pertencem execução "*nExec*". Estas duas consultas são executadas para todos os *n* valores de "*execução*" existentes no sumário e seus resultados são inseridos em um grafo de proveniência. Após *n* execuções destas consultas, todos os *n* grafos de proveniência são reconstruídos sem perda de dados.

A aplicação da operação inversa do sumário no experimento mostra empiricamente, que, a princípio, não há perda de informação com a sumarização dos grafos de proveniência realizada.

Capítulo 8 Conclusão

Esta tese apresenta o SGProv, um mecanismo que propõe aplicar a técnica de sumarização em grafos de proveniência gerados a partir das execuções de *workflows* científicos. O SGProv agrupa vértices e arestas correspondentes, que se repetem nos grafos de proveniência, gerando um único grafo sumário sem redundâncias e nem perdas de dados. A sumarização pode ser realizada com base em qualquer atributo de um vértice, que apareça mais frequentemente nos grafos de proveniência. SGProv também agrupa atributos de vértices e seus valores em um subgrafo do sumário, chamado de grafo de atributos. Neste grafo, os atributos e seus valores são armazenados uma única vez, eliminando redundâncias.

O grafo sumário tem como objetivo ser uma representação mais compacta de um conjunto de grafos de proveniência, reduzindo os tamanhos das bases de proveniência, em termos de diminuição do número de vértices e arestas. Por ser focado em eliminar redundâncias, quanto mais atividades, entidades e dependências entre elas (arestas) se repetirem nos grafos de proveniência, melhor será o resultado obtido com a sumarização em termos de redução no número de vértices e arestas. O sumário ótimo é obtido quando todos os grafos da base de proveniência forem iguais, em relação aos seus vértices e arestas. Se, por outro lado, os grafos de proveniência forem muito diferentes entre si, a sumarização pode produzir um grafo sumário tão volumoso quanto os grafos originais de proveniência (Figura 39).

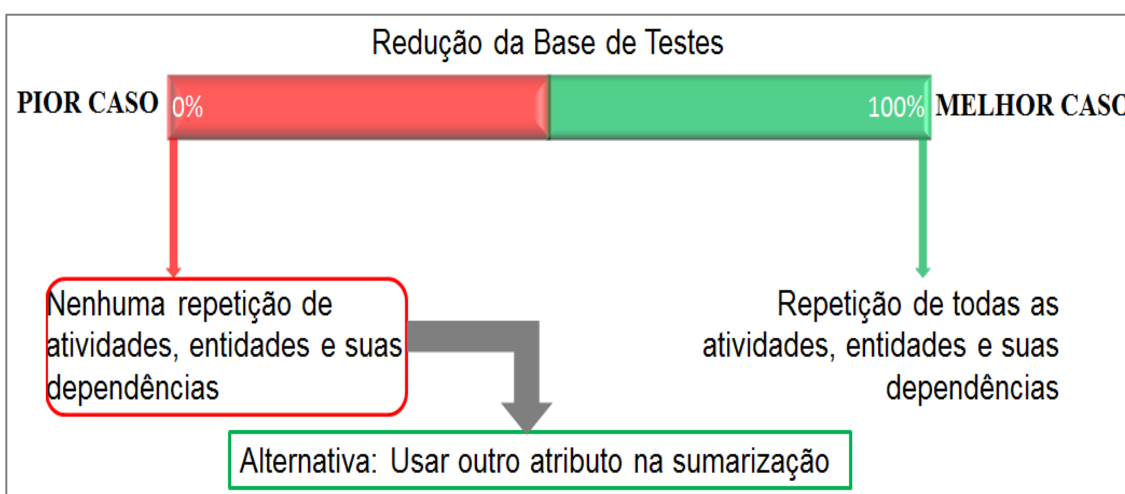


Figura 39. Redução das Bases de Testes obtida com o SGProv.

O SGProv permite que o atributo mais frequente nos vértices dos grafos de proveniência seja usado para determinar a equivalência destes vértices. Se não for possível determinar tal atributo, a sumarização deve ser feita com base em outras características dos vértices e das arestas. Por exemplo, pode-se agrupar vértices que possuem um mesmo conjunto de vértices vizinhos (Liu e Yu, 2011). Esta questão pode ser abordada em trabalhos futuros. Contudo, a principal característica dos grafos de proveniência resultantes de um mesmo experimento científico é a grande similaridade existente entre eles.

A sumarização proposta pelo SGProv traz um custo adicional correspondente à introdução do atributo "*execução*" em supervértices e superarestas do grafo sumário. Na avaliação experimental, após a compressão do atributo "*execução*", que codifica lacunas entre os números existentes na lista deste atributo, os tamanhos das bases dos sumários foram consideravelmente reduzidos em relação aos tamanhos das bases de testes (entre 30,17% e 67,70%) e das bases dos sumários sem compressão (entre 67,69% e 99,94%), em termos de redução do espaço de armazenamento em disco.

A sumarização também tem como objetivo reduzir o tempo de processamento das consultas de proveniência, uma vez que estas podem ser executadas sobre o grafo sumário, eliminando a necessidade de percorrer todos os grafos de proveniência produzidos em um experimento. Devido ao seu tamanho reduzido, o grafo sumário tem mais chances de poder ser totalmente carregado em memória principal. Devido aos mecanismos de *cache* do SGBD, tal característica tem o potencial de acelerar o tempo de execução de conjuntos de consultas sobre o mesmo grafo sumário. Os resultados obtidos na avaliação experimental mostraram uma redução entre 41,97% e 99,99% dos tempos de processamentos das consultas aplicadas nas bases dos sumários, em relação às mesmas consultas aplicadas nas bases de testes. A compressão do atributo "*execução*" contribuiu de maneira importante para reduzir o tempo de processamento de consultas que buscam este atributo como, por exemplo, a consulta *C8*. Isto porque, com a compressão o tamanho do atributo "*execução*" foi reduzido de *n* possíveis valores para atributos com intervalos contínuos delimitados por um par de valores.

Com as consultas utilizadas na avaliação experimental, foi confirmada a hipótese geral da tese de que usar funcionalidades de um SGBD de grafos para percorrer, recuperar e consultar dados de vários grafos de proveniência torna as consultas mais eficientes em relação, por exemplo, ao uso de um SGBD Relacional,

pois não envolve o processamento de muitas junções entre relações, para seguir a linhagem de derivação. Ao mesmo tempo, as consultas se tornam mais naturais devido ao SGBD usar a estrutura de dados nativa do grafo de proveniência. Com essas consultas, principalmente a *C3*, foi confirmada também a outra hipótese desta tese de que aplicar uma das formas de compactação, desenvolvida para grafos volumosos, nos vários grafos de proveniência, reduz o volume de dados e, assim, reduz também o tempo de processamento das consultas.

A operação inversa, realizada na avaliação experimental, mostrou que é possível reconstruir os grafos de proveniência originais a partir do grafo sumário, sem perda de dados. Esta operação pode ser usada, por exemplo, para que o cientista possa obter um determinado grafo de proveniência, caso seja necessário para a sua análise. A operação inversa do sumário se baseia em consultas feitas a vértices e arestas do grafo sumário que possuem um mesmo valor para o atributo "*execução*".

O SGProv considerou atividades, entidades e dependências correspondentes para agrupar vértices e arestas. Os grafos de proveniência no formato PROV-DM, além de atividades e entidades, possuem outros tipos de vértices como, por exemplo, agentes, e outros tipos de arestas. Além disso, as arestas também possuem atributos e seus valores. Estes e outros componentes também precisam ser considerados na sumarização, sendo possibilidades para trabalhos futuros.

O desempenho do Neo4j nas consultas de proveniência sem a aplicação do SGProv não foi satisfatório na avaliação experimental realizada. Idealmente o próprio SGBD deveria oferecer os mecanismos necessários para a compactação. Entretanto, para fazer uma análise mais ampla sobre a utilização dos SGBD de grafos para gerenciar grafos de proveniência e sobre a utilização de linguagens de consultas para grafos na realização de consultas de proveniência é preciso realizar testes comparativos com outros SGBD e outras linguagens de consultas, variando também o modelo de dados utilizado.

O desenvolvimento de uma biblioteca ou ferramenta para elaboração de consultas pelos cientistas é outra possibilidade de trabalho futuro. O Neo4j possui uma interface web para visualização dos grafos e realização de consultas. O trabalho descrito em Cuervas-Vicenttín *et al.* (2014) propõe uma ferramenta (PBase) que permite a visualização dos workflows, dos grafos de proveniência e a elaboração de consultas.

Entretanto, para usar estas ferramentas o cientista precisa ter conhecimento da linguagem de consultas Cypher.

Por último, um trabalho futuro importante, para que o usuário possa acompanhar a evolução da execução do workflow ao analisar a geração dos grafos de proveniência, seria viabilizar a submissão de consultas à base de grafos durante a execução do *workflow*. Para isso, seria necessário acoplar o SGProv à maquina de execução do workflow para que os dados de proveniência sejam armazenados e sumarizados de tempos em tempos, por exemplo.

Referências Bibliográficas

Abiteboul, S., Quass, D., McHugh, J., *et al.,* 1996, "The Lorel Query Language for Semistructured Data", *International Journal on Digital Libraries*, v. 1, pp. 68–88.

Abramson, D., Bethwaite, B., Enticott, C., *et al.*, 2011, "Parameter Exploration in Science and Engineering Using Many-Task Computing", *IEEE Transactions on Parallel and Distributed Systems*, v. 22, n. 6, pp. 960–973.

Aggarwal, C., Wang, H., 2010, *Managing and Mining Graph Data*. Springer.

Agrawal, R., Borgida, A., Jagadish, H. V., 1989, "Efficient Management of Transitive Relationships in Large Data and Knowledge Bases", *ACM SIGMOD Record*, v. 18, n. 2, pp. 253-262.

Amann, B., Constantin, C., Caron, C., *et al.*, 2013, "WebLab PROV: Computing Fine-Grained Provenance Links for XML Artifacts." In: *EDBT Workshop on Managing and Querying Provenance Data at Scale*, pp. 298–306. ACM.

Anand, M., 2010, *Managing Scientific Workflow Provenance*. D.Sc. thesis, University of California, USA.

Anand, M., Bowers, S., Altintas, I., *et al.,* 2010, "Approaches for Exploring and Querying Scientific Workflow Provenance Graphs." In: *3rd International Provenance and Annotation Workshop (IPAW 2010)*, NY, USA.

Anand, M., Bowers, S., Ludascher, B., 2009, "A Navigation Model for Exploring Scientific Workflow Provenance Graphs." In: *4th Workshop on Workflows in Support of Large-Scale Science (WORKS'09)*, NY, USA.

Anand, M., Bowers, S., Ludascher, B., 2010a, "Provenance Browser: Displaying and Querying Scientific Workflow Provenance Graphs." In: *26th IEEE International Conference on Data Engineering (ICDE'10)*, Long Beach, USA.

Anand, M., Bowers, S., Ludascher, B., 2010b, "Techniques for Efficiently Querying Scientific Workflow Provenance Graphs." In: *13th International Conference on Extending Database Technology (EDBT'10)*, NY, USA.

Anand, M., Bowers, S., Ludascher, B., 2012, "Database Support for Exploring Scientific Workflow Provenance Graphs." In: *24th International Conference on Scientific and Statistical Database Management*, Greece.

Angles, R., Gutierrez, C., 2005, "Querying RDF Data from a Graph Database Perspective." In: *2nd European Semantic Web Conference (ESWC'05)*, Greece.

Angles, R., Gutierrez, C., 2008. "Survey of Graph Database Models." *ACM Computing Surveys (CSUR).*

Balis, B., Bubak, M., Wach, J., 2007, "User-Oriented Querying Over Repositories of Data and Provenance." In: *IEEE International Conference on e-Science and Grid Computing*, India.

Barbosa, C., 2010, *HYDRA: Componentes Para o Paralelismo de Dados Em Experimentos Científicos*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

Barceló, P., 2013, "Querying Graph Databases." In: *32ND ACM SIGMOD-SIGACT-*

SIGART Symposium on Principles of Database Systems, NY, USA.

Barker, A., Van Hemert, J., 2008, "Scientific Workflow: A Survey and Research Directions." In: *Parallel Processing and Applied Mathematics*, v. 4967. Springer.

Belhajjame, K., Cheney, J., Corsar, D., *et al.*, 2013, PROV-O: The PROV Ontology. Disponível em: *http://www.w3.org/TR/2013/REC-prov-dm-20130430/.* Acesso em: 4 de set. 2015.

Boaventura Neto, P., 2010, *Grafos: Teorias, Modelos e Algoritmos*. 4 ed. Edgard Blucher.

Brito, R., 2010, "Bancos de Dados NoSQL X SGBDs Relacionais: Análise Comparativa." In: *III Congresso Tecnológico Da Info Brasil*, Fortaleza, Brasil.

Buneman, P, Khanna, S., Tan, W., 2001, "Why and Where: A Characterization of Data Provenance." In: *The 8th International Conference on Database Theory (ICDT'01)*, pp. 316–330, England.

Chapman, A., Jagadish, H., Ramanan, P., 2008, "Efficient Provenance Storage." In: *ACM SIGMOD International Conference on Management of Data*, Vancouver, Canada.

Cheah, Y., Plale, B., Kendall-Morwick, J., *et al.,* 2011, "A Noisy 10GB Provenance Database." In: *Business Process Management Workshops (BPM 2011).* Springer, pp. 370–381, France.

Chunhyeok, L., Shiyong, L., Artem, C., *et al.,* 2013, "Opql: Querying Scientific Workflow Provenance at the Graph Level", *Data Knowledge Engineering*, v. 88, pp. 37-59.

Cormen, T., Leiserson, C., Rivest, R., *et al.,* 2002, *Algoritmos Teoria e Prática*. 2 ed. Elsevier.

Costa, P., Braganholo, V., 2010, "Uma Nova Abordagem Para Consultas a Dados de Proveniência." In: *Simpósio Brasileiro de Banco de Dados (SBBD'10).* Belo Horizonte, Brasil.

Cuervas-Vicenttín, V., Luascher, B., Chirigati, F., *et al.,* 2014, "The PBase Scientific Workflow Provenance Repository", *International Journal of Digital Curation*, v. 9, n. 2, pp. 28–38.

Davidson, S., Freire, J., 2008, "Provenance and Scientific Workflows: Challenges and Opportunities". In: *Proceedings of the of the ACM SIGMOD International Conference on Management of Data*, pp. 1345-1350, NY, USA.

Davidson, S., Cohen-Boulakia, S., Eyal, A., *et al.,* 2008, "Provenance in ScientificWorkflow Systems." In: *Second International Provenance and Annotation Workshop (IPAW'08)*, Utah, USA.

Deelman, E., 2010, "Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments", *International Journal of High Performance Computing Applications*, v. 24, n. 3, pp. 284-298.

Deelman, E., Gannon, D., Shields, M., et al., 2009, "Workflows and e-Science: An Overview of Workflow System Features and Capabilities", *Future Generation Computer Systems*, v. 25, n. 5, pp. 528–540.

Deelman, E., Singh, G., Su, M., et al., 2005, "Pegasus: A Framework for Mapping

Complex Scientific Workflows Onto Distributed Systems", *Scientific Programming*, v. 13, n. 3, pp. 219–237.

El-Jaick, D., Mattoso, M., Lima, A. A. B., 2013, "SGProv: Mecanismo de Sumarização para Múltiplos Grafos de Proveniência". In: *Anais do 28º Simpósio Brasileiro de Banco de Dados (SBBD'13)*, Recife, PE, Brasil.

El-Jaick, D., Mattoso, M., Lima, A. A. B., 2014, "SGProv: Summarization Mechanism for Multiple Provenance Graphs", *Journal of Information and Data Management*, v. 5, n. 1, pp. 16-27.

Fan, W., Li, J., Wang, X., et al., 2012, "Query Preserving Graph Compression." In: *ACM SIGMOD/PODS*, Arizona, USA.

Freire, J., Koop, D., Santos, E., et al., 2008, "Provenance for Computational Tasks: A Survey", *Computing in Science and Engineering*, v. 10, n. 3, pp. 11–21.

Gadelha, L., Mattoso, M., 2011, "Provenance Query Patterns for Many-Task Scientific Computing." In: *3rd Usenix Workshop on the Theory and Practice of Provenance (TAPP'11)*, Greece.

Gadelha, L., Wilde, M., Mattoso, M., et al., 2013, "Provenance Traces of the Swift Parallel Scripting System", *EDBT/ICDT Workshops*, pp. 3325-326.

Gil, Y., Deelman, E., Ellisman, M., et al., 2007, "Examining the Challenges of Scientific Workflows", *Journal Computer*, v. 40, n. 12, pp. 24–32.

Gilbert, A., Levchenk, K., 2004, "Compressing Network Graphs." In: *Workshop Report Link Analysis and Group Detection (LinkKDD'04)*, Washington, USA.

Giugno, R., Shasha, D., 2002, "GraphGrep: A Fast and Universal Method for Querying Graphs." In: *16th International Conference on Pattern Recognition (ICPR'02)*, Quebec, Canada.

Guerra, G., Rochinha, F. A., Elias, R., et al., 2012, "Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine", *International Journal for Uncertainty Quantification*, v. 2, n. 1, pp. 53–71.

Guting, R., 1994, "GraphDB: Modeling and Querying Graphs in Databases." In: *Proceedings of the 20th Very Large Databases Conference (VLDB)*, Santiago, Chile.

Haichuan, S., Kitsuregawa, M., 2013, "Efficient Breadth-First Search on Large Graphs with Skewed Degree Distributions." In: *16th International Conference on Extending Database Technology (EDBT'13)*, pp. 311–322, Italy.

Harting, O., Heese, R., 2007, "The SPARQL Query Graph Model for Query Optimization." In: *4th European Semantic Web Conference (ESWC'07)*, Austria.

Holland, D., Braun, U., Maclean, M., 2008, "Choosing a Data Model and Query Language for Provenance." In: *12th International Provenance and Annotation Workshop (IPAW'08)*, Utah, USA.

Holzschuher, F., Peinl, R., 2015, "Querying a Graph Database–Language Selection and Performance Considerations", *Journal of Computer and System Sciences*.

Huahai, H., Singh, A., 2008, "Graphs-at-a-time: Query Language and Access Methods for Graph Databases." In: *ACM SIGMOD International Conference on*

Management of Data, NY, USA.

Islam, S., 2010, *Provenance, Lineage, and Workflows*. D.Sc. thesis, Brown University, Providence, USA.

Karvounarakis, G., Ives, Z., Tannen, V., 2010, “Querying Data Provenance.” In: *ACM SIGMOD/PODS,* Indianapolis, USA.

LeFevre, K., Terzi, E., 2010, “GraSS: Graph Structure Summarization”. In: *SIAM International Conference on Data Mining (SDM 2010)*, Ohio, USA.

Lim, C., Lu, S., Chebotko, A., et al., 2010, “Storing, Reasoning and Querying OPM Compliant Scientific Workflow Provenance Using Relational Databases”, *Future Generation Computer Systems*, v. 27, pp. 781–789.

Lim, C., Lu, S., Chebotko, A., et al., 2011, “OPQL: A First OPM-Level Query Language for Scientific Workflow Provenance.” In: *IEEE International Conference on Services Computing (SCC)*, pp. 136–143, Washington, USA.

Liu, Z., Yu, J., 2011, “On Summarizing Graph Homogeneously.” In: *Database Systems for Advanced Applications (DASFAA 2011)*, Hong Kong, China.

Lourenço, J., Cabral, B., Carreiro, p., et al., 2015, “Choosing the Right NoSQL Database for the Job: A Quality Attribute Evaluation”, *Journal of Big Data*. v. 2, n. 1, pp. 1-26.

Ma, S., Cao, Y., Fan, W., et al., 2012, “Capturing Topology in Graph Pattern Matching.” In: *VLDB Endowment*, v. 5, n. 4, Istanbul, Turquia.

Marinho, A., 2011, *ProvManager: Uma Abordagem para Gerenciamento de Proveniência de Experimentos Científicos*. Dissertação de M.Sc., Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil.

Mattoso, M., Werner, C., Travassos, G., et al., 2008, “Gerenciando Experimentos Científicos Em Larga Escala.” In: *XXVIII Congressos Da SBC - SEMISH - Seminário Integrado de Software e Hardware*, Pará, Brasil.

Mattoso, M., Werner, C., Travassos, G., et al., 2009, “Desafios No Apoio à Composição de Experimentos Científicos Em Larga Escala.” In: *XXXVI Seminário Integrado de Software e Hardware*, RS, Brasil.

Mattoso, M., Werner, C., Travassos, G et al., 2010, “Towards Supporting the Life Cycle of Large Scale Scientific Experiments.” In: *J. Business Process Integration and Management*, v. 5, n.1.

McGuiness, D., Michaelis, J., Moreau, L., 2008, *Provenance and Annotation of Data and Process*, Springer.

Missier, P., Byans, J., Gamble, C., et al, 2014, “ProvAbs: Model, Policy, and Tooling for Abstracting PROV Graphs”. In: *Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop*. Cologne, Germany.

Moreau, L., Clifford, B., Freire, J., et al., 2009, “The Open Provenance Model Core Specification (V1.1)”, *Future Generation Computer Systems*, Elsevier.

Moreau, L., Missier, P., 2012, The PROV Data Model and Abstract Syntax Notation. Disponível em: *www.w3c.org/TR/2012/WD-prov-dm-20120202*. Acesso em: 4 set. 2015.

Navlakha, S., Rastogi, R., Shrivastava, N., 2008, “Graph Summarization with Bounded Error.” In: *ACM SIGMOD International Conference on Management of Data*, Vancouver, Canada.

Ocaña, K. A. C. S., Oliveira, D., Dias, J., et al., 2011a, “Optimizing Phylogenetic Analysis Using SciHmm Cloud-based Scientific Workflow”. In: *Proceedings of the 2011 IEEE Seventh International Conference on eScience*, pp. 62-69, Washington, USA.

Ocaña, K. A. C. S., Oliveira, D., Ogasawara, et al., 2011b, “SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes”, *Advances in Bioinformatics and Computational Biology*, Springer, v. 6832, pp. 66–70, Heidelberg, Berlin.

Ogasawara, E., Dias, J., Oliveira, et al., 2011, “An Algebraic Approach for Data-Centric Scientific Workflows”. In: *Proceedings of the VLDB Endowment*, v. 4, n. 12, pp. 1328–1339, Seattle, USA.

Ogasawara, E., Dias, J., Silva, V., et al., 2013, “Chiron: a parallel engine for algebraic scientific workflows”, *Concurrency and Computation: Practice and Experience*. v. 25, n. 16, pp. 2327-2341.

Oliveira, D., Ocaña, K., Ogasawara, E., et al., 2011, “A Performance Evaluation of X-Ray Crystallography Scientific Workflow Using SciCumulus”, *IEEE CLOUD*, pp. 708-715.

Rangel, F., Silva, A., 2012, “A Máquina Virtual Java e a Otimização Inline: Um estudo de Caso”, *Revista Tecnológica*, v. 21. p. 103-118. Centro de Tecnologia da Universidade Estadual de Maringá-UEM, Brasil.

Robinson, I., Webber, J., Eifrem, E., 2013*, Graph Databases*. O'Reilly Media.

Rodriguez, M. e Neubauer, P., 2011, “A Path Algebra for Multi-Relational Graphs”. In: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops (ICDEW '11),* pp. 128-131, Hanover, Germany.

Roubtsov, V., 2003, “Watch Your HotSpot Compiler Go”. Disponível em: *http://www.javaworld.com/article/2077337/build-ci-sdlc/watch-your-hotspot-compiler-go.html*. Acesso em: 5 set. 2015.

Sadalage, P.J., Fowler, M., 2012, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional.

Santos, I., Dias, J., Oliveira, D., et al., 2013, “Runtime Dynamic Structural Changes of Scientific Workflows in Clouds”. In: *Proceedings of the International Workshop on Clouds and (eScience) Applications Management – CloudAM*, Dresden, Germany.

Shashani, A., Rotem, D., 2009, *Scientific Data Management Challenges, Technologies and Deployment*, CRCPress.

Silva, C., 2011, *Captura de Dados de Proveniência de Workflows Científicos Em Nuvens Computacionais*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

Silva, E., Ogasawara, E., Oliveira, D., et al., 2010, “Especificação Formal e Verificação de Workflows Científicos.” In: *XXX Congresso Da SBC - IV e-Science Workshop*, Minas Gerais, Brasil.

Silva, V., Oliveira, D., Mattoso M, 2014, SciCumulus 2.0: Um Sistema de Gerência de

Workflows Científicos para Nuvens Orientado a Fluxo de Dados. In: *Anais do 29º Simpósio Brasileiro de Banco de Dados (SBBD'14)*, Seção de Demos e Aplicações, Curitiba Paraná, Brasil.

Soares, R., 2013, *Estudo Comparativo entre Sistemas de Bancos de Dados de Grafos e Relacionais para a Gerência de Dados de Proveniência em Workflows Científicos*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

Starlinger, J., Davidson, S., Cohen-Boulakia, S., et al., 2015, "Effective and Efficient Similarity Search in Scientific Workflow Repositories", *Future Generation Computer Systems*.

Szwarcfiter, J.L, 1984, *Grafos e Algoritmos Computacionais*. 1 ed., Rio de Janeiro, Campus.

Szwarcfiter, J.L., 2003. *A Survey on Clique Graphs*. NY, USA, Springer.

Szwarcfiter, J.L., Markenzon, L., 2012, *Estruturas de Dados e Seus Algoritmos*. 3 ed., LTC.

Taylor, I., Deelman, E., Gannon, D., *et al.*, 2007, *Workflows for e-Science: Scientific Workflows for Grids*, Springer.

Teixeira, C., Silva, A., Meira, W., 2012, "Min-Hash Fingerprints for Graph Kernels: A Trade-off Among Accuracy, Efficiency and Compression." In: *Simpósio Brasileiro de Banco de Dados (SBBD'12)*, São Paulo, Brasil.

Tian, Y., Patel, J., 2010, "Interactive Graph Summarization", *Link Mining: Models, Algorithms, and Applications*, pp. 389-409, Springer.

Tylissanakis, G., Cotronis, Y., 2009, "Data Provenance and Reproducibility in Grid Based Scientific Workflows". In: *Workshop at the Grid and Pervasive Computing Conference*, Switzerland.

Vicknair, C., Macias, M., Zhao, Z., et al., 2010, "A Comparison of a Graph Database and a Relational Database". In: *48th ACM Southeast Conference*, Oxford, USA.

Wen, S., Fokoue, A., Srinivas, K., et al, 2015, "SQLGraph: An Efficient Relational-Based Property Graph Store". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, *ACM*, pp. 1887-1901.

Woodman, S., Hiden, H., Watson, P., 2011, "Achieving Reproducibility by Combining Provenance with Service and Workflow Versioning". In: *6th Workshop on Workflows in Support of Large-scale Science (WORKS'11), ACM*, pp. 127-136, NY, USA.

Xie, Y., Muniswamy-Reddy, K., Feng, D., et al., 2012, "A Hybrid Approach for Efficient Provenance Storage". In: *21st ACM International Conference on Information and Knowledge Management (CIKM'12)*, Maui, USA.

Xie, Y., Muniswamy-Reddy, K., Long, D., et al., 2011, "Compressing Provenance Graphs". In: *3rd Usenix Workshop on Theory and Practice of Provenance (TAPP'11)*, Greece.

Yu, P., Han, J., Faloutsos, C., et al., 2010, *Link Mining: Models, Algorithms, and Applications*. 2010 ed., Springer.

Zhang, N., Tian, Y., Patel, J., 2010, "Discovery-Driven Graph Summarization". In: *IEEE 26th International Conference on Data Engineering (ICDE)*, USA.