Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Banco de Dados em Grafos: Uma Conversão Para Restrições Pseudo-Booleanas

Autor: Gabriel Marques Tiveron
Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF
2022

Gabriel Marques Tiveron

Banco de Dados em Grafos: Uma Conversão Para Restrições Pseudo-Booleanas

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB
Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF
2022

Agradecimentos

Agradeço aos meus pais, Luis e Roselene, por sempre terem me apoiado e se dedicado a mim, em toda a minha caminhada acadêmica e pessoal, com o suporte para que eu pudesse focar nos meus estudos. Agradeço, também, aos meus irmãos, Letícia e Daniel, por terem me acompanhado e aconselhado em todos os momentos em que necessitei. Agradeço a minha namorada, Isabelle, por ter sempre me dado forças e apoio para crescer em minha vida.

Agradeço ao professor Bruno, um professor excepcional, que sempre demonstra entusiasmo em suas explicações, por ter me orientado e corroborado com a conquista dessa minha etapa acadêmica.

Resumo

Com o avanço da tecnologia e da necessidade de aramazenamento de dados, é imprescindível para empresas a utilização de bancos de dados. Os bancos de dados (BDs) possuem diversas abordagens, como, por exemplo o modelo relacional. Dentre as abordagens, destaca-se a orientada a grafos, usada em empresas de grande porte, como o LinkeIn.

O BD orientado a grafos possui uma vantagem de utilização para a empresa LinkedIn, pois tem o modelo de armazenamento em um formato único, o qual permite atualizações em tempo constante, bem como possibilita a realização de consultas específicas, de muito interesse e com maior facilidade.

Tendo em vista tal sistema, o objetivo desse trabalho é transcrever um grafo, bem como suas consultas, para um modelo de restrições pseudo-Booleanas e, após aplicado em um resolvedor, verificar o desempenho e consistência do modelo proposto.

Palavras-chave: Banco de dados em Grafo; Consultas; SAT; Pseudo-Boolean; Lógica proposicional; Modelagem SAT.

Lista de ilustrações

Figura 1 – Representação de um grafo com sujeito, predicado e objeto 10
Figura 2 – Exemplo de grafo direcionado 11
Figura 3 – Exemplo de grafo não direcionado (ou bidirecionado) 11
Figura 4 – Caminho entre i e z 12
Figura 5 – Representação da formula CNF em grafo 15
Figura 6 – Conexões de primeiro grau de j destacadas 18
Figura 7 – Diagrama com o esquema criado no Alg. 3.2 19
Figura 8 – Exemplo de grafo 22
Figura 9 – Relação da quantidade de variáveis e restrições como crescimento dos grafos 29
Figura 10 – Criação de restrições em relação ao tempo 30
Figura 11 – Desempenho em consultas do tipo FOAF por tempo 31
Figura 12 – Menor caminho pelo método de restrições 31
Figura 13 – Diagrama de armazenamento em VList 43
Figura 14 – Exemplo da aplicação de hash map em um objeto genérico 44

Lista de tabelas

Tabela 1 – Tabela verdade da operação condicional ou implicação 14
Tabela 2 – Tabela verdade da operação bicondicional 14
Tabela 3 – Tabela verdade da fórmula CNF descrita na Eq. 2.2 15
Tabela 4 – Relação entre restrições e grafos 28
Tabela 5 – Índices de ligação de arestas 44

Lista de abreviaturas e siglas

BD	Banco de dados
BDG	Banco de dados em Grafos
BDR	Banco de dados Relacional
BFS	Busca em largura
CNF	Forma Normal Conjuntiva ou *Conjunctive Normal Form*
DFS	Busca em profundidade
F	Falso
ID	Índice
LIFO	*Last in, First Out* ou Último a entrar, primeiro a sair
SAT	Satisfatibilidade ou Satisfatível
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Linguagem de Consulta Estruturada ou *Structured Query Language*
UnB	Universidade de Brasília
UNSAT	Não Satisfatível
V	Verdadeiro
VList	Lista de Vetores

Sumário

1 INTRODUÇÃO . . . 9

Introdução . . . 9

2 REFERENCIAL TEÓRICO . . . 11

2.1 Grafos . . . 11

2.1.1 Busca . . . 12

2.1.2 Caminho . . . 12

2.1.3 Amigo de amigo - FOAF . . . 12

2.2 SAT . . . 12

2.2.1 Lógica Proposicional . . . 12

2.2.1.1 Leis de De Morgan . . . 13

2.2.1.2 Tabela Verdade . . . 14

2.2.1.3 Forma Normal Conjuntiva (CNF) . . . 14

2.2.2 Satisfatibilidade . . . 14

2.2.2.1 Avaliando Resolução SAT . . . 16

2.2.2.2 Pseudo-Booleano . . . 16

3 TRABALHOS CORRELATOS . . . 17

3.1 *On Modeling Connectedness in Reductions from Graph Problems to Extended Satisfiability* . . . 17

3.2 LIquid . . . 18

3.2.1 Estrutura . . . 19

3.2.1.1 Consultas em LIquid . . . 19

4 IMPLEMENTAÇÃO . . . 22

4.1 Visão Geral . . . 22

4.2 Formulação . . . 22

4.2.1 Operador Choose-H . . . 23

4.2.2 Restrições . . . 24

4.3 Consultas . . . 26

4.3.1 Conexões de primeiro grau . . . 26

4.3.2 Caminho . . . 26

4.3.3 FOAF . . . 27

4.4 Enumeração de modelos . . . 27

4.5 Experimentos . . . 28

4.5.1 Casos de teste 28
4.5.2 Resultados 30

5 CONCLUSÃO 32

REFERÊNCIAS 33

ANEXOS 34

ANEXO A – CASO DE EXEMPLO 35

ANEXO B – LIQUID 42
B.0.1 Indexação 42
B.0.2 Lista de Vetores 43
B.0.3 Hash Map 43
B.0.4 Visão geral 44
B.0.5 Consultas 44
B.0.6 Datalog 46

1 Introdução

A utilização de bancos de dados (BDs) se tornou imprescindível para as empresas e qualquer fornecedor de serviços digitais mediante o crescimento da demanda por armazenamento de dados para diversas finalidades, como servidores de empresas ou governamentais (YU, 2008). Dessa forma, ao longo do tempo, surgiram diversos modelos de SGBD's, como o orientado a objeto, o relacional, o baseado em grafos, dentre outros. Cada modelo criado tem uma proposta de regras de negócio para o armazenamento dos dados, as quais são otimizadas conforme cada caso particular. O SGBD explorado nesse trabalho será o baseado em grafos.

O SGBD baseado em grafos, utilizado em empresas de grande porte, como LinkedIn, AirBus e Lufthansa, possuem casos de uso extremamente importantes e abrangentes, para o acompanhamento de encomendas digitais por toda sua frota aérea, no caso da AirBus e Lufthansa, e para gerênciamento de conexões entre usuários e empresas, como é o caso do LinkedIn. Alguns deles são descritos em *Enterprise Analytics using Graph Database and Graph-based Deep Learning* (HENNA; IEEE; KALLIADAN, 2015) Além disso, os SGBD's em grafos são focados em: processamento de dados altamente conectados, possuir flexibilidade no modelo de dados utilizado pelos grafos e possuir um bom desempenho para leituras locais, fazendo uma travessia no grafo (POKORNý, 2015).

Um dos pontos de importância de um BD em grafo é a eficiência das consultas. A pesquisa de conexão em primeiro grau é relativamente simples, pois só é necessário acessar as ligações de um nó. Já as consultas de conexões de segundo grau, ou conexões de conexões, podem trazer mais dificuldade para o processamento, devido à grandeza de relações dos nós (MEYER et al., 2019). Para entender como as consultas são realizadas, primeiramente, é necessário compreender como as informações são armazenadas: os nós são estruturas que não necessariamente obedecem a um padrão. As ligações, por sua vez, são consolidadas da seguinte forma: primeiramente são definidos o sujeito (que efetua a ação), o predicado (a descrição da conexão) e o objeto. Assim é possível criar as relações de um nó com outro, como pode-se observar na Fig. 1.

As consultas em SGBD's em grafos, especificamente LIquid, são feitas baseando-se em satisfatibilidade de condições (YANG; YAO, 2019), ou seja, são determinadas condições a serem obedecidas por nós e arestas. Cita-se, abaixo, um exemplo de aplicação das condições a serem obedecidas:

> Selecione Nós que possuam o Atributo $X = W$ e possuam ligações do tipo Y ou do tipo Z;

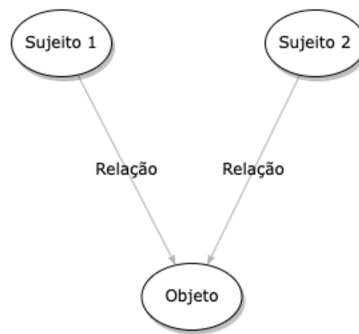


Figura 1 – Representação de um grafo com sujeito, predicado e objeto

Os sistemas de satisfatibilidade possuem uma sequência de passos para a formação de equações que variam o resultado entre verdadeiro e falso. Dessa forma, é possível exemplificar a formulação do referido tipo de problema, como descrito na Eq. 1.1, onde n é o numero de conjuntos de restrições e m e j são restrições quaisquer, as quais não se limitam a apenas duas. Por fim, k é o valor de comparação. Caso a soma de m_1 e j_1 for menor que k_1, o valor resultante da expressão será falso.

$$\sum_{i=1}^{n} m_i + j_i >= k_i \tag{1.1}$$

Tendo em vista esse modelo de formatação, pode-se traduzir as restrições de consulta de um BD em grafos para o referido tipo de restrição. Sendo assim, após a formulação das restrições, pode-se aplicá-las a um resolvedor com a finalidade de obter os casos do grafo os quais obedecem às regras definidas. Tal modelo de formatação é o objeto de estudo do presente trabalho. Nesse sentido, pode-se verificar uma o desempenho das consultas obtidas por meio do referido método.

Este trabalho tem por objetivo realizar consultas em um banco de dados em grafos utilizando uma instância SAT. Para isso, foram realizados estudos e desenvolvimentos de ferramentas com a finalidade de se obter resultados e, posteriormente, suas análises.

Dessa forma, o trabalho está dividido da seguinte maneira: a Seção 2 abordará o referencial teórico, onde serão trabalhados os temas de grafos, formulações e resolvedores SAT. A Seção 3 tratará de trabalhos correlatos, os quais trazem um arcabolso teórico e motivação. Por fim, a Seção 4 analisará a implementação do modelo proposto, bem como os resultados.

2 Referencial Teórico

2.1 Grafos

Como é explicado em *Inductive graphs and functional graph algorithms* (ERWIG, 2001), a visão predominante de um grafo é um bloco monolítico, onde um grafo (G) é descrito como G = (V, E), onde (V), são um conjunto de vértices, e (E) é um subconjunto de V em V,

$$E \subseteq V \times V$$

onde a estrutura que um nó representa pode variar de acordo com a necessidade. Grafos podem ser direcionados ou não direcionados. Nos direcionados, as arestas indicam um nó de origem e outro de destino, impossibilitando o caminho na direção inversa na mesma aresta, Fig. 2. Já nos grafos não direcionados, ou bidirecionais, a aresta indica uma conexão entre ambos os nós independente de direcionamento, podendo ser feito um caminho de um para o outro, por meio da mesma aresta. Vide Fig. 3.

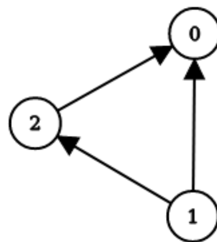


Figura 2 – Exemplo de grafo direcionado

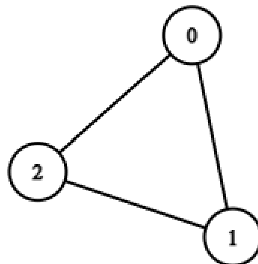


Figura 3 – Exemplo de grafo não direcionado (ou bidirecionado)

2.1.1 Busca

O grafo é comumente utilizado em diversos casos. Alguns algoritmos essenciais são os de inserção ou deleção de um novo nó/aresta e de buscas dentro do grafo. Os algoritmos de busca são, no geral, formas de atravessar o grafo procurando por um nó ou aresta em específico. Tais algoritmos baseiam-se em sua essência no caminho que pode ser alcançado por cada nó.

2.1.2 Caminho

Como ilustrado na Fig. 4, um nó tem conexões com outros nós e, ao realizar uma travessia entre esses nós, tem-se um caminho formado entre dois nós. Esse caminho não necessariamente é único, podendo haver diferença de quantidade de nós visitados entre um e outro.

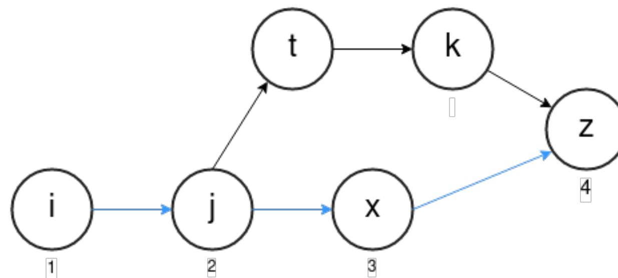


Figura 4 – Caminho entre i e z

2.1.3 Amigo de amigo - FOAF

Amigo de amigo ou *Friend of a friend (FOAF)*, é uma consulta em grafo que visa abranger todas as conexões de segundo grau de um nó específico, ou seja, todos os nós que têm um vizinho em comum.

2.2 SAT

Primeiramente, para entender o princípio da satisfatibilidade, é necessário conhecer a lógica proposicional e outras propriedades que serão descritas nesta seção.

2.2.1 Lógica Proposicional

Uma lógica proposicional é um sistema onde fórmulas representam sentenças declarativas, ou proposições, que podem ser verdadeiras (V) ou falsas (F), mas nunca ambas (RIBAS, 2015).

As proposições são atômicas e, juntas, podem formar sentenças compostas com o auxílio de conectivos lógicos. Um exemplo de proposição é:

Está chovendo.

Essa afirmação só pode assumir os valores V ou F. Já um exemplo de senteça composta é:

Se está chovendo, então o céu está nublado.

Essa afirmação utiliza um conectivo lógico chamado implicação representada pelo símbolo ($\Rightarrow$) onde, se uma proposição é verdadeira, então a seguinte também será. Além desse conectivo também têm-se os conectivos de: bicondicional ($\Leftrightarrow$), negação ($\neg$), conjunção ($\wedge$) e, por fim, disjunção ($\vee$). Sendo assim, uma sentença é formada por uma ou mais sentenças e conectivos.

Uma *fórmula*, que é um conjunto de proposições simples ou compostas, segue um conjunto de regras (RIBAS, 2015). Essas são:

- Todo átomo é uma fórmula;
- Se α é uma fórmula, então $(\neg\alpha)$ também é uma fórmula;
- $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \Rightarrow \beta)$ e $(\alpha \Leftrightarrow \beta)$ serão fórmulas se, e somente se, α e β forem fórmulas;
- Todas as fórmulas são geradas pela aplicação das regras anteriores.

2.2.1.1 Leis de De Morgan

Como descreve (KRIVOCHEN, 2018) em *De Morgan's laws and NEG-raising: a syntactic view*, as cláusulas ou conjunto de sentenças e operadores lógicos também tem algumas propriedades que devem ser observadas. As Leis de De Morgan descrevem que:

- $(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta$
- $(\alpha \wedge \beta) = \neg\alpha \vee \neg\beta$

Ou seja, um operador lógico $\wedge$ pode ser decrito como um operador $\vee$ executando as devidas negações às suas proposições. Além disso, vale notar que tal fato ocorre se, e somente se, $\neg\alpha \vee \neg\neg\alpha = V$.

2.2.1.2 Tabela Verdade

Uma forma de visualizar a resolução de fórmulas proposicionais é por meio da tabela verdade. Essa tabela descreve todas as combinações possíveis de uma dada condição ou conjunto de sentenças e operações. As Tab. 1 e 2 exemplificam tabelas verdade dos operadores condicional e bicondicional.

α	β	$\alpha \Rightarrow \beta$
F	F	V
F	V	V
V	F	F
V	V	V

Tabela 1 – Tabela verdade da operação condicional ou implicação

α	β	$\alpha \Leftrightarrow \beta$
F	F	V
F	V	F
V	F	F
V	V	V

Tabela 2 – Tabela verdade da operação bicondicional

2.2.1.3 Forma Normal Conjuntiva (CNF)

A Forma Normal Conjuntiva, segundo Briere (2009), é descrita como uma fórmula onde as operações entre proposições são apenas disjunções e tais conjuntos são relacionados entre sí por conjunções. Desse modo, as fórmulas devem obecer esse pressuposto e têm o formato descrito na Eq. 2.1.

$$(\alpha \vee \beta) \wedge (\delta \vee \gamma) \wedge ... \wedge (\nu \vee \kappa) \tag{2.1}$$

A representação da CNF também pode ser feita por meio de um grafo, onde a raiz representa o operador de conjunção ($\wedge$) e, logo em seguida, os operadores de disjunção ($\vee$), seguidos ou não por operadores de negação, a depender da fórmula em questão. Por fim, as folhas representam os literais. A Fig. 5 representa um formato em grafos para a Eq. 2.2.

$$(\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \tag{2.2}$$

2.2.2 Satisfatibilidade

Para que um problema possa ser considerado e resolvido como SAT, ele deve, primeiramente, estar na sua CNF. Dessa maneira, é necessário que se apliquem as leis de

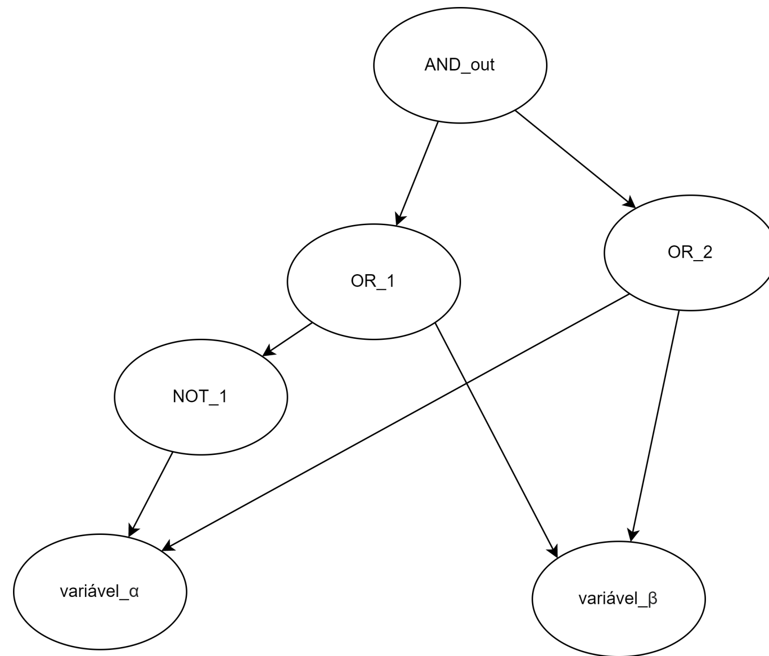


Figura 5 – Representação da formula CNF em grafo

Fonte: Adaptação de Ribas (2015)

De Morgan, negações ou operações distributivas para que, então, a fórmula passe a estar em sua CNF.

SAT é um problema considerado NP-completo, ou seja, pode ser verificada em tempo polinomial. Esses problemas consideram que as proposições podem assumir valores entre Verdadeiro e Falso, e avaliam se há uma combinação possível para que o resultado do conjunto de restrições se torne verdadeiro. Sendo assim, pode-se analisar cada caso por meio de uma tabela verdade e verificar se o conjunto possui uma combinação possivel (SAT), ou não (UNSAT).

Considerando a formula descrita no diagrama da Fig. 5, temos a tabela verdade descrita na Tab. 3.

α	β	$(\alpha \vee \neg\beta) \wedge (\alpha \vee \beta)$
F	F	F
F	V	F
V	F	V
V	V	V

Tabela 3 – Tabela verdade da fórmula CNF descrita na Eq. 2.2

Podemos afirmar, então, que o problema é satisfatível, desde que α seja verdadeiro. Dessa forma, como há uma combinação possível para que o conjunto de restrições seja verdadeiro, pode-se classificar esse problema como SAT.

2.2.2.1 Avaliando Resolução SAT

Para realizar um passo-a-passo da resolução feita na Tab. 3, deve-se, primeiramente, definir os valores das variáveis α e β. Considerando que os valores foram definidos como $f1 = \alpha : V, \beta : F$, então a Eq. 2.3 descreve a resolução da fórmula.

$$\begin{gathered}(\alpha \lor \neg\beta) \land (\alpha \lor \beta) \\ (V \lor \neg F) \land (V \lor F) \\ (V \lor V) \land (V) \\ V \land V \\ V\end{gathered} \tag{2.3}$$

2.2.2.2 Pseudo-Booleano

Os sistemas pseudo-Booleanos são bem próximos ao modelo SAT, diferenciando-se apenas pelo fato das variáveis serem condicionadas em sistemas onde valores não se prendem a zeros e uns. Um exemplo é observado na Eq. 2.4.

$$[\sum_{i=1}^{n} x_i] <= w_k \tag{2.4}$$

No caso da Eq. 2.4, x é uma variável booleana, podendo assumir os valores de 0 ou 1. Já w obedece o conjunto $\{\forall t \in w \in \mathbb{Z} \mid 0 \leq t \leq n\}$.

3 Trabalhos correlatos

Como trabalhos correlatos, cita-se dois de extrema importância, sendo eles: On Modeling Connectedness in Reductions from Graph Problems to Extended Satisfiability Oliveira et al. (2012) - que estrutura uma modelagem na qual o presente trabalho se baseia; e Nosecond indexing of graph data with hash maps and vlists. Proceedings of the 2019 International Conference on Management of Data Meyer et al. (2019) - o qual expõe a problemática da qual o vigente trabalho se propõe explorar.

3.1 *On Modeling Connectedness in Reductions from Graph Problems to Extended Satisfiability*

Devido aos esforços nas áreas de SAT e MaxSAT, o interesse na redução de diversos problemas NP para (Max)SAT tem crescido, principalmente problemas complexos da Teoria de Grafos (OLIVEIRA et al., 2012). Para alcançar os objetivos descritos no trabalho, é apresentado um operador para a escolha de Nós no grafo. Dessa forma, é descrito o operador **Choose-H**.

Seja $\mathbb{B} = \{0, 1\}$ um conjunto de valores onde 1 corresponde a verdadeiro e 0 a falso. Sendo assim, pode-se considerar $H \subseteq \{0, \cdots, k\}$ como um conjunto finito, positivo e de inteiros. Dessa forma define-se o operador **Choose-H** (C_H):

$$C_H(\, f_1, f_2, \cdots, f_k \,) = \begin{cases} 1, se \sum_{i=1}^{k} f_i \in H \\ \qquad 0, c.c. \end{cases} \tag{3.1}$$

Desse modo, o valor será verdadeiro apenas se o número de argumentos verdadeiros for igual a H. Assim, o operador $C_{\{0,2\}}(\, f_1, \cdots, f_k \,)$ será verdadeiro se, e somente se, houver zero ou dois argumentos verdadeiros.

Esse operador possui algumas propriedades importantes a serem destacadas, sendo elas: $(\, f_1 \wedge \cdots \wedge f_k \,) = C_k(\, f_1, \cdots, f_k \,)$, $(\, f_1 \vee \cdots \vee f_k) = C_{\{0,\cdots,k\}}(\, f_1, \cdots, f_k \,)$, $\neg f_1 = C_0(\, f_1 \,)$ e, por fim, $\neg C_H(f_1, \cdots, f_k) = C_{\{0,\cdots,k\}\setminus H}(f_1, \cdots, f_k)$.

Além do operador, também é de suma importância conhecer a notação $\mathcal{N}$, que denota o conjunto de Nós vizinhos. Um exemplo pode ser observado na Fig. 6 o $\mathcal{N}(j) = \{i, t, x\}$.

Uma das soluções propostas é a de encontrar um caminho entre dois Nós arbitrários u e w. Para tal, é preciso que o conjunto de vizinhos $\mathcal{N}$ seja interpretado de tal forma que

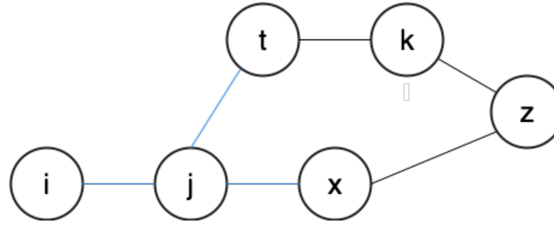


Figura 6 – Conexões de primeiro grau de j destacadas

$\mathcal{N}(v_i) = \{v_0, \cdots, v_k\}$ tal que os valores assumidos por cada Nó do conjunto está contido em $\mathbb{B}$.

Sendo assim, a modelagem que restringe o conjunto para resultar no caminho entre u e w é descrito segundo a fórmula:

$$f_p(G, v_u, v_w) = C_1(\mathcal{N}(v_u)) \wedge C_1(\mathcal{N}(v_w)) \wedge C_{\{0,2\}}(\mathcal{N}(v_1)) \wedge C_{\{0,2\}}(\mathcal{N}(v_2)) \wedge \cdots \wedge C_{\{0,2\}}(\mathcal{N}(v_m))$$

onde G representa o grafo, v_u e v_w representam os Nós v e w e, por fim, $v_1, v_2, \cdots, v_m$ representam os outros Nós do grafo fora u e w.

Dessa forma, a restrição criada possibilita que apenas as variáveis que fazem parte de um caminho qualquer, se existente, entre u e w sejam verdadeiras, permitindo a solução do problema em questão.

3.2 LIquid

Diferentemente do Banco de Dados Relacional (BDR), o qual, no geral, tem uma estrutura tabular de relacionamentos, os Bancos de Dados em Grafos (BDG's) possuem uma estrutura única, como visto na Fig. 1. Essa estrutura é facilmente reproduzida em código.

```
1 struct node {
2     vector<edge> edges;
3 };
4
5 struct edge {
6     node* subject;
7     node* predicate;
8     node* object;
9 };
```

Algoritmo 3.1 – Estrutura de dado de um grafo

Como representado no Alg. 3.1, pode-se observar uma lista de arestas, onde cada aresta é composta por três elementos: o sujeito, que executa uma ação; o predicado, que representa o tipo de relação; e o objeto, que sofre a ação. Um exemplo possível pode

ser a relação de conhecimento, onde se uma pessoa A (sujeito) conhece (predicado) um determinado conteúdo (objeto). Essa relação será representada a partir do preenchimento dos dados no código citado.

3.2.1 Estrutura

Um exemplo de criação de uma base pode ser notada no Alg. 3.2. Essa sequência de passos gera um esquema no formato observado na Fig. 7

O algoritmo supracitado define relações visando o princípio já comentado de sujeito, predicado e objeto, onde o sujeito de uma das relações é o ator, o predicado é *atua em* e o objeto é o filme. Dessa mesma forma, ocorrem as relações, ou arestas, em um banco de dados que se baseia em grafos.

```
1 CREATE (LOTR:Movie {title: 'Senhor dos Aneis: A sociedade do Anel',
    released: 2001})
2 CREATE (peter:Person:Director {name: 'Peter Jackson', born: 1961})
3 CREATE (viggo:Person:Actor {name: 'Viggo Mortensen', born: 1958})
4 CREATE (viggo)-[:ACTED_IN {roles: ['Aragorn']}]->(LOTR)
5 CREATE (peter)-[:DIRECTED]->(LOTR)
```

Algoritmo 3.2 – Exemplo de esquema de banco de dados em grafos

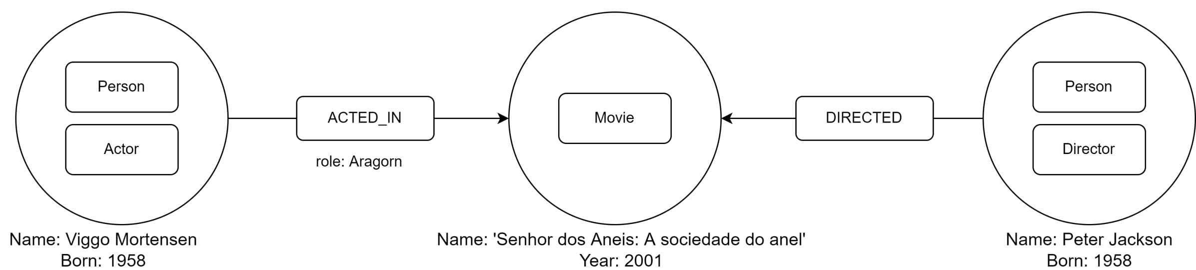


Figura 7 – Diagrama com o esquema criado no Alg. 3.2

Fonte: Autor

A referida formatação rege praticamente todos os relacionamentos descritos segundo o banco descrito por Meyer et al. (2019). Dessa forma, fica visível os fatos de que todo relacionamento é de primeiro grau, bem como o fato de que fazer sua travessia tem um custo constante.

3.2.1.1 Consultas em LIquid

Enquanto em BDR é necessário realizar *joins* em diversas tabelas por meio de chaves estrangeiras, nos BDG's, tendo um index inicial, se faz uma travessia no grafo até o(s) nó(s) desejável(is). Para tal, utiliza-se uma notação parecida com a de datalog:

```
1 Vertice ( x, "relacao-X", y ),
```

```
2 Vertice ( y, "relacao-y", "no qualquer" )?
```

Algoritmo 3.3 – Consulta LIquid

Diferentemente dos datalogs, essa consulta é traduzida para um grafo de restrições. Tais restrições são mapeadas para uma *hydra*, que é basicamente uma coluna, onde são listados os possíveis valores.

$$vértice(x, "relacao - X", y) \rightarrow [(x_1, y_1), ...] \tag{3.2}$$

$$vértice(y, "relacao - y", "noqualquer") \rightarrow [(y_1), ...] \tag{3.3}$$

Sendo assim, ao verificar que um nó qualquer tem uma relação do tipo y com o nó especificado, basta iterar sobre suas conexões até que se encontre um x possível.

Algoritmo 1: Busca em um grafo por predicados de um sujeito / objeto

```
while edges in y do
    if edge.type is soughtRelation then
        addToResponse(edge)
    end
end
```

Por fim, é possível notar que a estrutura geral de um banco de dados baseado em grafos difere-se do modelo relacional principalmente por quatro pontos principais, esses sendo:

- Todas as relações são de primeiro grau. A forma de expressar relações com a tríplice já citada de sujeito, predicado e objeto torna as relações mais palpáveis. Enquanto o modelo relacional, possuindo mais de uma tabela e relacionamentos de muitos para muitos passam a ter de realizar *joins* tornando muitas vezes algo de dificil leitura.
- Travessia de arestas de forma rápida e tempo constante. Como o endereço de acesso para o próximo nó é armazenado no nó vigente a travessia se torna eficiente e possui tempo constante.
- Resultado de consultas é um subgrafo, em contrapartida ao resultado tabular visto no modelo mais comum de banco de dados, o modelo em questão retorna uma estrutura de grafo que obedece as condições impostas pela consulta e pode ter seu resultado visualizado em formato tabular ou não.
- Evolução de esquema em tempo constante. Como os atributos de um dado nó é feito por meio de uma aresta entre ambos, a adição ou deleção tem custo constante e é mais simples que um *ALTER TABLE* realizado em bancos de dados relacionais.

Um estudo mais detalhado acerca do Banco de Dados LIquid pode ser observado no Anexo B.

4 Implementação

4.1 Visão Geral

Para interpretar um grafo em restrições pseudo-Booleanas, é necessário, primeiramente, que seja definida a forma com que serão tratados cada elemento dessa estrutura de dado. Para exemplificar essa transcrição será utilizado o grafo da Fig. 8.

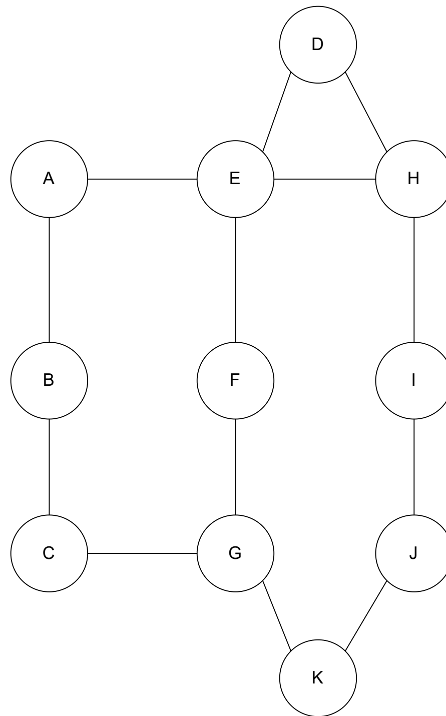


Figura 8 – Exemplo de grafo

Para a implementação, serão considerados grafos bidirecionais. Sendo assim, pode-se separar cada nó e cada aresta para serem mapeados posteriormente. O objetivo desse mapeamento é tornar possível a execução de consultas e interpretação dos grafos por meio de restrições pseudo-Booleanas.

4.2 Formulação

O mapeamento de variáveis foi definido para que sejam supridas as propostas do trabalho. Nesse sentido, o mapeamento possibilita a transcrição genérica de um grafo para restrições SAT. Desse modo, essa transcrição permite a criação de consultas em grafos, com a adição de outras restrições, com a finalidade de encontrar conexões de primeiro grau, caminho entre dois nós quaisquer e, por fim, amigos de amigos (FOAF).

Como citado anteriormente, cada vértice e cada aresta será mapeado para variáveis SAT e terão seus valores representados no domínio de $\mathbb{B}$. Bem como citado em Oliveira et al. (2012), será utilizado o operador **Choose-H** para realizar a seleção ou não dos nós.

4.2.1 Operador Choose-H

O operador **Choose-H** descreve o comportamento de escolha de vértices dado um nó qualquer. Sendo assim, temos a notação padrão para esse operador, a qual é dada pelo simbolo C, seguido de um número, ou conjunto de números, os quais especificam a quantidade de arestas que devem ser escolhidas para que esse operador seja valorado como verdadeiro. Caso contrário, será valorado como falso.

Dessa forma, é possível descrever o operador de duas formas principais: quando há apenas um número e quando há um conjunto de números. Para o primeiro caso, onde há apenas um número, temos o operador C_n, onde $n \in \mathbb{N}$. Ao representá-lo em restrições pseudo-Booleanas, temos as Eq. 4.1 e 4.2,

$$\sum_{i=0}^{N} \neg a_i \geq N - n \tag{4.1}$$

$$\sum_{i=0}^{N} a_i \geq n \tag{4.2}$$

onde N indica o total de conexões do vértice em questão, a_i representa uma aresta do nó que está sendo mapeado e, por fim, n descreve o número de arestas que devem ser valoradas como verdadeiras.

Já para o caso onde há um conjunto de restrições, é necessário atribuir uma nova variável para atuar como um *switch* entre as duas restrições. Dessa forma, podemos descrever um operador $C_{\{t,n\}} \therefore t, n \in \mathbb{N}$, onde pode haver t ou n arestas verdadeiras no conjunto. Sendo assim, temos as Eq. 4.3 e 4.4.

$$\forall \quad k \in \{t, n\} \quad (\sum_{i=0}^{N} a_i) + N \times \neg\sigma_k \geq \alpha_k \tag{4.3}$$

$$\forall \quad k \in \{t, n\} \quad (\sum_{i=0}^{N} \neg a_i) + N \times \sigma_k \geq 2 \times N - \alpha_k \tag{4.4}$$

A semântica das variáveis se mantêm, porém, com o acréscimo das variáveis de *switch*, dada pelo símbolo σ_k, bem como do acréscimo de demarcação, dada pelo símbolo α_k, onde $\alpha \in \{t, n\}$. A variável σ faz uma ponte entre os dois conjuntos de restrições, ou seja, torna ambos os casos verdadeiros para que o conjunto de solução possa ser satisfatível. Já a variável α, representa a demarcação de qual caso está sendo tratado, seja de t ou de

n. Sendo assim, é necessário que, para cada número do conjunto, haja as duas equações descritas acima.

Tendo em vista a descrição desse operador, no contexto do presente trabalho, será utilizado apenas a representação de C_n, onde há apenas uma opção de aresta a ser escolhida.

4.2.2 Restrições

Cada vértice possui quatro variáveis correspondentes a si, as quais definem sua categoria. Cada nó pode possuir apenas uma categoria. Além disso, cada aresta possui uma variável que é valorada como 1, caso seja utilizada, e como 0, caso não seja utilizada.

Para que os nós obedeçam essas categorias, são geradas as Eq. 4.5 a 4.18 para cada vértice do grafo.

$$C_{0_v} + P_v + I_v + F_v \geq 1 \tag{4.5}$$

$$\neg C_{0_v} + \neg P_v + \neg I_v + \neg F_v \geq 3 \tag{4.6}$$

Primeiramente, cada vértice do grafo será classificado entre quatro categorias: não escolhido (C_0), passagem (P), inicial (I) e final (F), o que pode ser observado nas Eq. 4.5 e 4.6, onde, para cada vértice será possível definir apenas uma dessas categorias como verdadeira.

$$C_{0_v} + \sum_{i=0}^{E|v|} x_{v_i} \geq 1 \tag{4.7}$$

$$E|v|\neg C_{0_v} + \sum_{i=0}^{E|v|} \neg x_{v_i} \geq E|v| \tag{4.8}$$

As Eq. 4.7 e 4.8 descrevem que, ao categorizar um vértice como C_0, todas as arestas que incluem esse nó devem ser valorados como falso (0). Nessas equações, para cada vértice é gerado um conjunto de restrições, onde C_0 é uma das possíveis categorizações do nó, $E|v|$ é o grau do vértice, ou seja, a quantidade de arestas que ele possui e, por fim, x_{v_i} representa uma aresta do vértice v que tem seu domínio também em $\mathbb{B}$.

$$\neg I_v + \sum_{i=0}^{E|v|} x_{v_i} \geq 1 \tag{4.9}$$

$$E|v|\neg I_v + \sum_{i=0}^{E|v|} \neg x_{v_i} \geq E|v| - 1 \tag{4.10}$$

$$\neg F_v + \sum_{i=0}^{E|v|} x_{v_i} \geq 1 \tag{4.11}$$

$$E|v|\neg F_v + \sum_{i=0}^{E|v|} \neg x_{v_i} \geq E|v| - 1 \tag{4.12}$$

Já as Eq. 4.9 e 4.10 descrevem a definição de um nó categorizado como inicial (I). Sendo assim, ao identificar um nó como inicial, deve-se garantir que uma aresta será utilizada - assumir o valor verdadeiro, visto que desse nó deverá seguir para um nó futuro através de uma das arestas. Essas equações mantém seu comportamento para atender igualmente a categoria de vértice final (F), na qual deverá ser selecionada uma aresta por onde se chegará ao final.

$$2\neg P_v + \sum_{i=0}^{E|v|} x_{v_i} \geq 2 \tag{4.13}$$

$$E|v|\neg P_v + \sum_{i=0}^{E|v|} \neg x_{v_i} \geq E|v| - 2 \tag{4.14}$$

A última categoria abordada segue a mesma lógica das anteriores, porém acarreta na seleção de duas arestas, pois é necessário que seja definida uma aresta por onde se chegará ao vértice em questão e outra aresta para a saída ao próximo vértice. Sendo assim, as Eq. 4.13 e 4.14 exploram a restrição descrita.

Por fim, são restringidas as quantidades de vértices por categorias no grafo de forma geral. Dessa forma, temos as Eq. 4.15 e 4.16 que limitam os vértices iniciais para apenas um, o que pode ser aplicado igualmente para os nós finais, como nas Eq. 4.17 e 4.18.

$$\sum_{i=0}^{|V|} I_v \geq 1 \tag{4.15}$$

$$\sum_{i=0}^{|V|} \neg I_v \geq |V| - 1 \tag{4.16}$$

$$\sum_{i=0}^{|V|} F_v \geq 1 \tag{4.17}$$

$$\sum_{i=0}^{|V|} \neg F_v \geq |V| - 1 \tag{4.18}$$

4.3 Consultas

Com as equações descritas na Seção 4.2 pode-se, então, representar um grafo genérico qualquer e passar a realizar as consultas.

4.3.1 Conexões de primeiro grau

As ligações de primeiro grau são representadas por todos os nós que possuem uma aresta direta com o vértice em questão. Traduzindo para a linguagem de restrições criadas na seção anterior, temos que há um nó inicial, o qual se pretende encontrar as conexões, e um nó final, que é a conexão. Dessa forma, podemos definir um conjunto de restrições para adicionar às restrições do grafo, seja u um vértice qualquer do qual se pretende encontrar as conexões de primeiro grau. Assim, temos que:

$$I_u \geq 1 \tag{4.19}$$

$$\sum_{i=0}^{|V|} P_i \geq 0 \tag{4.20}$$

$$\sum_{i=0}^{|V|} \neg P_i \geq N \tag{4.21}$$

A Eq. 4.19 define u como o nó inicial, enquanto as Eq. 4.20 e 4.21 restringem o sistema para que não haja nenhum nó de passagem, o que implica no fato de que todos os nós na vizinhança de u sejam passíveis de serem categorizados com o tipo final (F).

4.3.2 Caminho

Dado dois vértices u e w, pode-se traçar um caminho entre ambos ao definir um nó inicial (u) e um nó final (w). Sendo assim, é possível transcrever essas restrições. Vide Eq. 4.19, que define o nó inicial, e 4.22, que define o nó final.

$$F_w \geq 1 \tag{4.22}$$

Tendo essa estrutura feita em restrições, temos que, para encontrar o menor caminho entre dois pontos, apenas adicionar a Eq. 4.23 para que seja buscado o menor caminho possível.

$$minimize : \sum_{i=0}^{N} P_i \tag{4.23}$$

Dessa forma, o total de nós (N) será restringido, para que o menor número de vértices seja categorizado como de passagem (P) considerando apenas grafos sem pesos nas arestas.

4.3.3 FOAF

Semelhante à conexão de primeiro grau, a consulta de amigos de amigos apenas se difere por possuir um nó de passagem entre o vértice de início e o de final. Dessa forma, é definido o nó inicial como na Eq. 4.19 e, por meio das Eq. 4.24 e 4.25, define-se que deve haver um, e apenas um, nó de passagem.

$$\sum_{i=0}^{|V|} P_v \geq 1 \tag{4.24}$$

$$\sum_{i=0}^{|V|} \neg P_v \geq |V| - 1 \tag{4.25}$$

Essa modelagem do problema permite que, de forma simples, sejam adicionados mais vértices intermediários, sem a necessidade da adição de um *for* aninhado, por exemplo, que seria a ação necessária em um modelo tradicional.

Uma exemplificação do passo-a-passo da implementação, juntamente com as consultas, pode ser observada no Anexo A.

4.4 Enumeração de modelos

Com as equações definidas na Seção 4.2, e com o acréscimo das equações definidas na Seção 4.3, podemos chegar a uma solução, seja de conexões de primeiro grau, caminho ou conexões de amigos de amigos. Porém, chegar à apenas uma solução, geralmente não é suficiente. Para tal, temos o conceito de enumerar os modelos que são gerados.

A enumeração de modelos consiste em gerar diversas combinações de valoração das variáveis até cobrir um número pré-definido de tentativas ou até cobrir todas as combinações possíveis. Dessa forma, utilizamos o resolvedor *CLASP* (GEBSER et al., 2007) com parâmetros específicos, os quais serão melhor apresentados abaixo, para atender as demandas do presente trabalho.

A utilização do *CLASP* foi feita com o auxílio das opções de `model`, que quantificam o número de modelos resultantes da solução formulada. Já a opção `quiet`, trata do que será mostrado como resultado. Por fim, a opção `thread`, realiza a paralelização do resolvedor.

Sendo assim, a execução, para a resolução do conjunto de restrições descrito nas equações acima relatadas, do resolvedor *CLASP* é realizada da seguinte maneira:

```
1 clasp 0 --quiet=2 -t 12 file_name.pbs
```

onde o 0 representa a opção de modelos, o qual, por padrão, é 1, e faz com que sejam realizadas todas as possibilidades de solução possíveis. Já a opção `-quiet=2`, faz com que a exibição da solução mostre apenas quantos modelos foram enumerados como SAT, juntamente com o tempo de solução, número de *threads*, dentre outras informações. Logo após, a *tag* `-t 12` passa a informação de quantas *threads* serão utilizadas ao ser executado o resolvedor.

No trabalho em questão foram utilizadas 12 *threads*, pela utilização de memória que não permitiu a utilização das 16 *threads* disponíveis no processador. Por fim, é definido o nome do arquivo onde o resolvedor buscará o conjunto de soluções que será resolvido, `file_name.pbs`.

4.5 Experimentos

Para a realização dos experimentos, foi utilizado o resolvedor *CLASP* (GEBSER et al., 2007), versão 3.3.4 (libpotassco version 1.1.0), e foram gerados grafos com conexões aleatórias com a finalidade de extrair as métricas para cada caso e comparar o desempenho e eficiência do modelo proposto. Sendo assim, foram feitas consultas de amigos de amigos em grafos diversos para analisar o desempenho.

4.5.1 Casos de teste

A Tab. 4 mostra os dados dos casos utilizados para gerar as restrições e, posteriormente, extrair as métricas de desempenho.

Vértices	Total Arestas	Arestas/vértice	Variáveis	Restrições
100	4.450	89	4.850	1.004
500	50.500	202	52.500	5.004
750	84.000	224	87.000	7.504
1.000	136.000	272	140.000	10.004
10.000	1.375.000	275	1.415.000	100.004

Tabela 4 – Relação entre restrições e grafos

Vale notar que a relação entre o crescimento no número de variáveis e restrições se dá de forma linear, obedecendo as Eq. 4.26, para as variáveis, e 4.27, para as restrições,

$$N_{vars} = 4 \times |V| + |E| \tag{4.26}$$

$$N_{restrictions} = 10 \times |V| + 4 \tag{4.27}$$

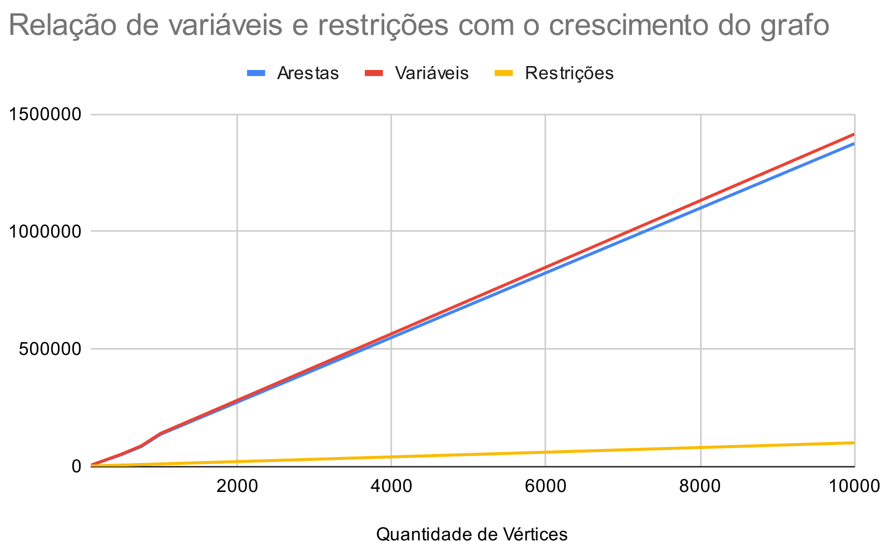


Figura 9 – Relação da quantidade de variáveis e restrições como crescimento dos grafos

onde $|V|$ é a quantidade total de vértices e $|E|$ é a quantidade total de arestas.

Esses grafos foram criados aleatoriamente levando em conta a quantidade média de conexões de primeiro grau que a maior parte das pessoas possuem no LinkedIn, Anciaux (2018), onde 28% possuem entre 0 e 300 conexões de primeiro grau. Desse modo, para cada caso, exceto o de 100 vértices, onde foi utilizado o alcance de 80 a 90, foi escolhido aleatoriamente um número entre 200 e 300, que definiu a quantidade de arestas que cada vértice possuiria.

A geração dos grafos foi feita com o auxílio da biblioteca NetworkX, da linguagem Python, utilizando a função de gerar grafos normalizados, ou seja, com arestas de número padrão, definida aleatoriamente como citado anteriormente.

A máquina de testes, utilizada para gerar as métricas de análise, é uma máquina dedicada e confiável para a realização do *benchmark* do sistema desenvolvido. A máquina segue as seguintes especificações:

- AMD Ryzen 2700 Eight-Core Processors
- 32 GB de RAM
- 112 GB de memória em disco
- SO - Ubuntu 20.04 LTS

Tendo em vista esse grafo gerados, foi possível mapear de forma genérica cinco casos diferentes. Como pode-se observar na Fig. 10, o mapeamento generico do grafo leva um tempo considerável para ser realizado. Já a adição das restrições de consulta, citadas

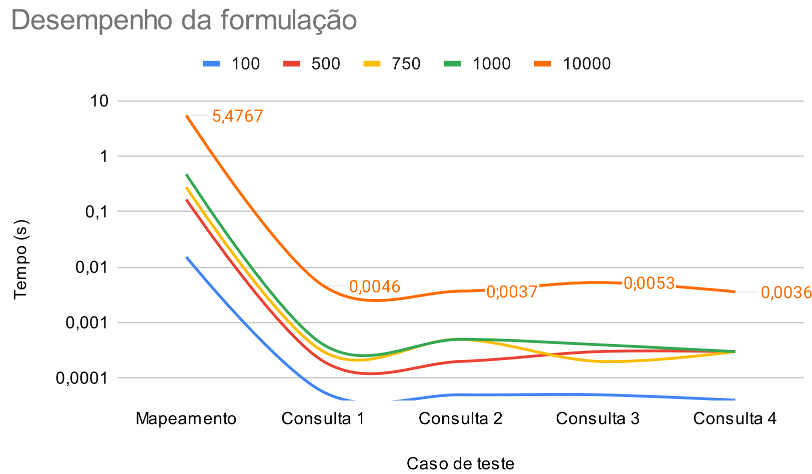


Figura 10 – Criação de restrições em relação ao tempo

na Seção 4.3, tem uma queda relevante no tempo, visto que o modelo proposto para realizá-las torna essa operação trivial.

4.5.2 Resultados

Após a implementação do modelo, foram extraídos os resultados de desempenho. Dentre os resultados extraídos, cita-se o tempo de geração de conversão do *input* para restrições pseudo-Booleanas e a adição de consultas para o grafo em questão. Dessa forma, a título de comparação, foi implementada uma solução tradicional, Alg. 2, a qual corrobora para a interpretação dos resultados, bem como para ter uma margem devida com a finalidade de quantificar a eficiência do modelo proposto.

Algoritmo 2: Método Tradicional

Data: v
Result: $FOAF(v)$
for $u \in E_v$ **do**
  for $w \in E_u$ **do**
    $r \leftarrow w$;
  end
end
return r

Sendo assim, após a execução do modelo em grafos de grandezas variadas, temos os resultados obtidos pelo modelo proposto em comparação ao modelo tradicional apresentado na Fig. 11. Observa-se que o desempenho do modelo tradicional possui uma eficiência maior em relação as restrições SAT utilizando o resolvedor CLASP.

Pode-se observar que o tempo que leva para gerar as consultas é irrisório, porém o CLASP não é otimizado para esse tipo específico de consultas, visto que é um resolvedor

genérico de conjunto de restrições SAT. Dessa forma, sugere-se como trabalho futuro a criação de um resolvedor que interprete as variáveis de forma tal que proporcione um melhor desempenho.

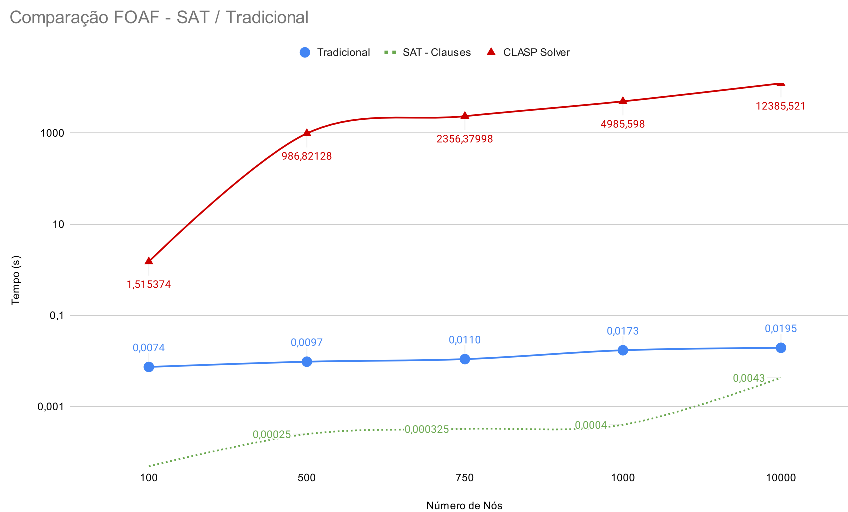


Figura 11 – Desempenho em consultas do tipo FOAF por tempo

Já para o caso de consultas de menor caminho em um grafo, temos como resultado a Fig. 12. Nela, pode-se perceber que há um desempenho melhor do que nas consultas de amigos de amigos. Tal fato ocorre pois não é necessário realizar a enumeração de modelos, somente é necessário que se encontre o modelo classificado como "ótimo", aquele que minimiza os nós categorizados como passagem.

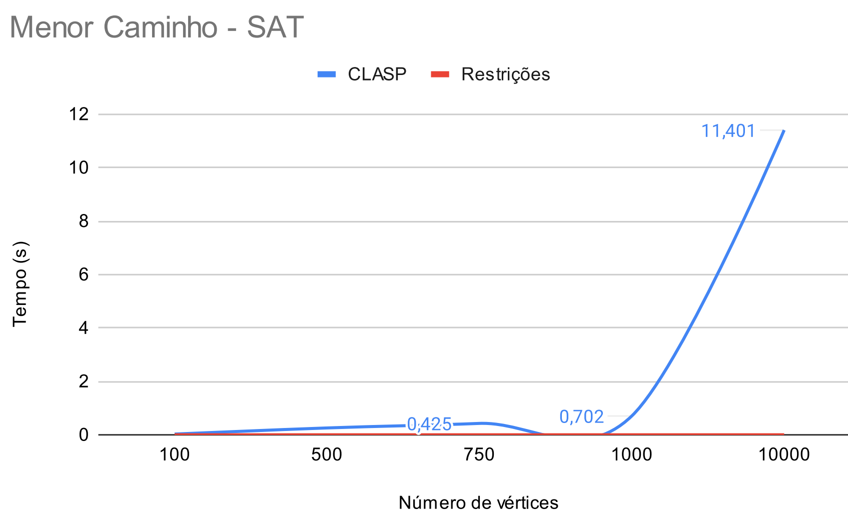


Figura 12 – Menor caminho pelo método de restrições

5 Conclusão

O presente trabalho propôs a interpretação de grafos por uma abordagem pseudo-Booleana de maneira tal que fosse possível interpretar um grafo genérico em um conjunto de restrições SAT. Propôs também, realizar consultas de mesmo modo como são feitas em bancos de dados em grafos, como a consulta de caminho entre vértices e conexões de primeiro e segundo grau. Sendo assim, mostrou-se possível efetuar esse modelo, realizando as consultas de forma geral e não se limitando apenas às consultas listadas, visto que, para gerar novas consultas, basta adicionar, ao conjunto, restrições que adequem o sistema ao resultado o qual se pretende obter.

Com a realização dos experimentos, o modelo mostrou-se consistente em suas respostas, porém com um tempo de solução elevado. Tal tempo é explicado pelo modo o qual o resolvedor foi utilizado, realizando um *enumerate*, ou seja, listando todas as combinações de variáveis do sistema até não encontrar mais soluções possíveis. Além disso, o número de variáveis tem um aumento considerável mediante o aumento de vértices, visto que cada vértice possui quatro variáveis de categoria.

Verificou-se, dessa forma, que o desempenho da abordagem ainda não supera o modelo tradicional. Porém, é possível otimizar a solução das restrições visto que, a cada operação de busca, pode-se negar as respostas encontradas anteriormente, reduzindo, assim, a quantidade de operações necessárias para se chegar à próxima solução. Sugere-se, dessa forma, que esse modelo de resolvedor seja trabalhado como um estudo futuro para que se analise o desempenho de um resolvedor focado na resolução de consultas em grafos, podendo adicionar no resolvedor uma restrição a cada iteração de sucesso na enumeração para que os nós categorizados como finais (F) sejam definidos nessa categoria apenas uma vez. Sugere-se, ainda, que sejam realizados estudos aprofundados acerca da forma de otimização do resolvedor para esses tipos de consultas, além de adaptar o modelo para solucionar consultas de menor caminho com pesos nas arestas.

Referências

ANCIAUX, L. *Interesting statistics on the number of LinkedIn connections per user!* 2018. Urlhttps://www.linkedin.com/pulse/interesting-statistics-number-linkedin-connections-per-lionel-anciaux. Citado na página 29.

BRIERE, A. Handbook of satisfiability: Volume 185 frontiers in artificial intelligence and applications. 2009. Citado na página 14.

ERWIG, M. Inductive graphs and functional graph algorithms. *Journal of Functional Programming*, Cambridge University Press PUB280 New York, NY, USA, v. 11, p. 467–492, 9 2001. ISSN 09567968. Disponível em: <https://dl.acm.org/doi/abs/10.1017/S0956796801004075>. Citado na página 11.

GEBSER, M. et al. clasp: A conflict-driven answer set solver. In: *LPNMR*. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 4483), p. 260–265. Citado 2 vezes nas páginas 27 e 28.

HENNA, S.; IEEE, S. M.; KALLIADAN, S. K. Enterprise analytics using graph database and graph-based deep learning. 2015. Citado na página 9.

KRIVOCHEN, D. G. De morgan's laws and neg-raising: a syntactic view. *Linguistic Frontiers*, Walter de Gruyter GmbH, v. 1, p. 112–121, 12 2018. Citado na página 13.

MEYER, A. C. A. R. Y. Y. S. et al. Nanosecond indexing of graph data with hash maps and vlists. *Proceedings of the 2019 International Conference on Management of Data*, ACM, v. 13, 2019. Disponível em: <https://doi.org/10.1145/3299869.> Citado 5 vezes nas páginas 9, 17, 19, 42 e 46.

OLIVEIRA, R. Tavares de et al. On modeling connectedness in reductions from graph problems to extended satisfiability. In: . [S.l.: s.n.], 2012. ISBN 978-3-642-34653-8. Citado 2 vezes nas páginas 17 e 23.

POKORNý, J. Graph databases: Their power and limitations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer, Cham, v. 9339, p. 58–69, 2015. ISSN 16113349. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-319-24369-6_5>. Citado 2 vezes nas páginas 9 e 46.

RIBAS, B. C. Um método de pré-processamento de fórmulas sat e pseudo-boolean baseado em técnicas de programação linear inteira mista. 2015. Disponível em: <https://acervodigital.ufpr.br/handle/1884/41122>. Citado 3 vezes nas páginas 12, 13 e 15.

YANG, B. A. R. A. A. C. J. C. I. C. P. L. S. M. A. R. S. S. Y.; YAO, J. *LIquid: The soul of a new graph database, Part 2 | LinkedIn Engineering*. 2019. Disponível em: <https://engineering.linkedin.com/blog/2020/liquid--the-soul-of-a-new-graph-database--part-2>. Citado 3 vezes nas páginas 9, 43 e 46.

YU, B. Database fundamentals. v. 5, 2008. Citado na página 9.

Anexos

ANEXO A – Caso de exemplo

Para exemplificar a utilização do modelo descrito, podemos considerar o grafo da Fig. 8. Dessa forma, para o arquivo de entrada temos a primeira linha contendo o número de vértices (V) e arestas (E) respectivamente. Em seguida são listados V vértices, posteriormente E pares de vértices que compõem as arestas. Com isso, temos o arquivo de entrada descrito no Alg. A.1.

```
11 13
a
b
c
d
e
f
g
h
i
j
k
a e
a b
b c
c g
g f
f e
g k
k j
j i
i h
h e
h d
d e
```

Algoritmo A.1 – Arquivo de entrada

Dessa forma, pode-se mapear as variáveis e compor o arquivo de restrições, Alg. A.2, onde as variáveis de x_1 a x_{44} representam os vértices, categorizados respectivamente em C_0, inicial, final e passagem, já de x_{45} até x_{57} correspondem as arestas.

```
* Restricoes de a
* Arestas do vertice
+1 x1 +1 x53 +1 x57 >= 1;
* C_0
+2 ~x1 +1 ~x53 +1 ~x57 >= 2;
```

```
6 * Tipo dos nos
7 +1 ~x1 +1 ~x2 +1 ~x3 +1 ~x4 >= 3;
8 +1 x1 +1 x2 +1 x3 +1 x4 >= 1;
9 * Define Inicial
10 +1 ~x2 +1 x53 +1 x57 >= 1;
11 +2 ~x2 +1 ~x53 +1 ~x57 >= 1;
12 +1 ~x3 +1 x53 +1 x57 >= 1;
13 +2 ~x3 +1 ~x53 +1 ~x57 >= 1;
14 * Define Passagem
15 +2 ~x4 +1 x53 +1 x57 >= 2;
16 +2 ~x4 +1 ~x53 +1 ~x57 >= 0;
17 * Restricoes de b
18 * Arestas do vertice
19 +1 x5 +1 x46 +1 x57 >= 1;
20 * C_0
21 +2 ~x5 +1 ~x46 +1 ~x57 >= 2;
22 * Tipo dos nos
23 +1 ~x5 +1 ~x6 +1 ~x7 +1 ~x8 >= 3;
24 +1 x5 +1 x6 +1 x7 +1 x8 >= 1;
25 * Define Inicial
26 +1 ~x6 +1 x46 +1 x57 >= 1;
27 +2 ~x6 +1 ~x46 +1 ~x57 >= 1;
28 +1 ~x7 +1 x46 +1 x57 >= 1;
29 +2 ~x7 +1 ~x46 +1 ~x57 >= 1;
30 * Define Passagem
31 +2 ~x8 +1 x46 +1 x57 >= 2;
32 +2 ~x8 +1 ~x46 +1 ~x57 >= 0;
33 * Restricoes de c
34 * Arestas do vertice
35 +1 x9 +1 x46 +1 x49 >= 1;
36 * C_0
37 +2 ~x9 +1 ~x46 +1 ~x49 >= 2;
38 * Tipo dos nos
39 +1 ~x9 +1 ~x10 +1 ~x11 +1 ~x12 >= 3;
40 +1 x9 +1 x10 +1 x11 +1 x12 >= 1;
41 * Define Inicial
42 +1 ~x10 +1 x46 +1 x49 >= 1;
43 +2 ~x10 +1 ~x46 +1 ~x49 >= 1;
44 +1 ~x11 +1 x46 +1 x49 >= 1;
45 +2 ~x11 +1 ~x46 +1 ~x49 >= 1;
46 * Define Passagem
47 +2 ~x12 +1 x46 +1 x49 >= 2;
48 +2 ~x12 +1 ~x46 +1 ~x49 >= 0;
49 * Restricoes de d
50 * Arestas do vertice
51 +1 x13 +1 x47 +1 x55 >= 1;
52 * C_0
```

```
53 +2 ~x13 +1 ~x47 +1 ~x55 >= 2;
54 * Tipo dos nos
55 +1 ~x13 +1 ~x14 +1 ~x15 +1 ~x16 >= 3;
56 +1 x13 +1 x14 +1 x15 +1 x16 >= 1;
57 * Define Inicial
58 +1 ~x14 +1 x47 +1 x55 >= 1;
59 +2 ~x14 +1 ~x47 +1 ~x55 >= 1;
60 +1 ~x15 +1 x47 +1 x55 >= 1;
61 +2 ~x15 +1 ~x47 +1 ~x55 >= 1;
62 * Define Passagem
63 +2 ~x16 +1 x47 +1 x55 >= 2;
64 +2 ~x16 +1 ~x47 +1 ~x55 >= 0;
65 * Restricoes de e
66 * Arestas do vertice
67 +1 x17 +1 x45 +1 x47 +1 x50 +1 x53 >= 1;
68 * C_0
69 +4 ~x17 +1 ~x45 +1 ~x47 +1 ~x50 +1 ~x53 >= 4;
70 * Tipo dos nos
71 +1 ~x17 +1 ~x18 +1 ~x19 +1 ~x20 >= 3;
72 +1 x17 +1 x18 +1 x19 +1 x20 >= 1;
73 * Define Inicial
74 +1 ~x18 +1 x45 +1 x47 +1 x50 +1 x53 >= 1;
75 +4 ~x18 +1 ~x45 +1 ~x47 +1 ~x50 +1 ~x53 >= 3;
76 +1 ~x19 +1 x45 +1 x47 +1 x50 +1 x53 >= 1;
77 +4 ~x19 +1 ~x45 +1 ~x47 +1 ~x50 +1 ~x53 >= 3;
78 * Define Passagem
79 +2 ~x20 +1 x45 +1 x47 +1 x50 +1 x53 >= 2;
80 +4 ~x20 +1 ~x45 +1 ~x47 +1 ~x50 +1 ~x53 >= 2;
81 * Restricoes de f
82 * Arestas do vertice
83 +1 x21 +1 x50 +1 x52 >= 1;
84 * C_0
85 +2 ~x21 +1 ~x50 +1 ~x52 >= 2;
86 * Tipo dos nos
87 +1 ~x21 +1 ~x22 +1 ~x23 +1 ~x24 >= 3;
88 +1 x21 +1 x22 +1 x23 +1 x24 >= 1;
89 * Define Inicial
90 +1 ~x22 +1 x50 +1 x52 >= 1;
91 +2 ~x22 +1 ~x50 +1 ~x52 >= 1;
92 +1 ~x23 +1 x50 +1 x52 >= 1;
93 +2 ~x23 +1 ~x50 +1 ~x52 >= 1;
94 * Define Passagem
95 +2 ~x24 +1 x50 +1 x52 >= 2;
96 +2 ~x24 +1 ~x50 +1 ~x52 >= 0;
97 * Restricoes de g
98 * Arestas do vertice
99 +1 x25 +1 x49 +1 x51 +1 x52 >= 1;
```

```
100 * C_0
101 +3 ~x25 +1 ~x49 +1 ~x51 +1 ~x52 >= 3;
102 * Tipo dos nos
103 +1 ~x25 +1 ~x26 +1 ~x27 +1 ~x28 >= 3;
104 +1 x25 +1 x26 +1 x27 +1 x28 >= 1;
105 * Define Inicial
106 +1 ~x26 +1 x49 +1 x51 +1 x52 >= 1;
107 +3 ~x26 +1 ~x49 +1 ~x51 +1 ~x52 >= 2;
108 +1 ~x27 +1 x49 +1 x51 +1 x52 >= 1;
109 +3 ~x27 +1 ~x49 +1 ~x51 +1 ~x52 >= 2;
110 * Define Passagem
111 +2 ~x28 +1 x49 +1 x51 +1 x52 >= 2;
112 +3 ~x28 +1 ~x49 +1 ~x51 +1 ~x52 >= 1;
113 * Restricoes de h
114 * Arestas do vertice
115 +1 x29 +1 x45 +1 x54 +1 x55 >= 1;
116 * C_0
117 +3 ~x29 +1 ~x45 +1 ~x54 +1 ~x55 >= 3;
118 * Tipo dos nos
119 +1 ~x29 +1 ~x30 +1 ~x31 +1 ~x32 >= 3;
120 +1 x29 +1 x30 +1 x31 +1 x32 >= 1;
121 * Define Inicial
122 +1 ~x30 +1 x45 +1 x54 +1 x55 >= 1;
123 +3 ~x30 +1 ~x45 +1 ~x54 +1 ~x55 >= 2;
124 +1 ~x31 +1 x45 +1 x54 +1 x55 >= 1;
125 +3 ~x31 +1 ~x45 +1 ~x54 +1 ~x55 >= 2;
126 * Define Passagem
127 +2 ~x32 +1 x45 +1 x54 +1 x55 >= 2;
128 +3 ~x32 +1 ~x45 +1 ~x54 +1 ~x55 >= 1;
129 * Restricoes de i
130 * Arestas do vertice
131 +1 x33 +1 x48 +1 x54 >= 1;
132 * C_0
133 +2 ~x33 +1 ~x48 +1 ~x54 >= 2;
134 * Tipo dos nos
135 +1 ~x33 +1 ~x34 +1 ~x35 +1 ~x36 >= 3;
136 +1 x33 +1 x34 +1 x35 +1 x36 >= 1;
137 * Define Inicial
138 +1 ~x34 +1 x48 +1 x54 >= 1;
139 +2 ~x34 +1 ~x48 +1 ~x54 >= 1;
140 +1 ~x35 +1 x48 +1 x54 >= 1;
141 +2 ~x35 +1 ~x48 +1 ~x54 >= 1;
142 * Define Passagem
143 +2 ~x36 +1 x48 +1 x54 >= 2;
144 +2 ~x36 +1 ~x48 +1 ~x54 >= 0;
145 * Restricoes de j
146 * Arestas do vertice
```

```
147 +1 x37 +1 x48 +1 x56 >= 1;
148 * C_0
149 +2 ~x37 +1 ~x48 +1 ~x56 >= 2;
150 * Tipo dos nos
151 +1 ~x37 +1 ~x38 +1 ~x39 +1 ~x40 >= 3;
152 +1 x37 +1 x38 +1 x39 +1 x40 >= 1;
153 * Define Inicial
154 +1 ~x38 +1 x48 +1 x56 >= 1;
155 +2 ~x38 +1 ~x48 +1 ~x56 >= 1;
156 +1 ~x39 +1 x48 +1 x56 >= 1;
157 +2 ~x39 +1 ~x48 +1 ~x56 >= 1;
158 * Define Passagem
159 +2 ~x40 +1 x48 +1 x56 >= 2;
160 +2 ~x40 +1 ~x48 +1 ~x56 >= 0;
161 * Restricoes de k
162 * Arestas do vertice
163 +1 x41 +1 x51 +1 x56 >= 1;
164 * C_0
165 +2 ~x41 +1 ~x51 +1 ~x56 >= 2;
166 * Tipo dos nos
167 +1 ~x41 +1 ~x42 +1 ~x43 +1 ~x44 >= 3;
168 +1 x41 +1 x42 +1 x43 +1 x44 >= 1;
169 * Define Inicial
170 +1 ~x42 +1 x51 +1 x56 >= 1;
171 +2 ~x42 +1 ~x51 +1 ~x56 >= 1;
172 +1 ~x43 +1 x51 +1 x56 >= 1;
173 +2 ~x43 +1 ~x51 +1 ~x56 >= 1;
174 * Define Passagem
175 +2 ~x44 +1 x51 +1 x56 >= 2;
176 +2 ~x44 +1 ~x51 +1 ~x56 >= 0;
177 * Restringe Inicial
178 +1 x2 +1 x6 +1 x10 +1 x14 +1 x18 +1 x22 +1 x26 +1 x30 +1 x34 +1 x38 +1
       x42>= 1;
179 +1 ~x2 +1 ~x6 +1 ~x10 +1 ~x14 +1 ~x18 +1 ~x22 +1 ~x26 +1 ~x30 +1 ~x34 +1
        ~x38 +1 ~x42 >= 10;
180 * Restringe Final
181 +1 x3 +1 x7 +1 x11 +1 x15 +1 x19 +1 x23 +1 x27 +1 x31 +1 x35 +1 x39 +1
       x43>= 1;
182 +1 ~x3 +1 ~x7 +1 ~x11 +1 ~x15 +1 ~x19 +1 ~x23 +1 ~x27 +1 ~x31 +1 ~x35 +1
        ~x39 +1 ~x43 >= 10;
```

Algoritmo A.2 – Arquivo de restrições

Com essas restrições, o grafo já está representado em uma instância SAT e basta adicionar as restrições de consulta para chegar ao resultado esperado. Portanto, pode-se adicionar as cláusulas descritas no Alg. A.4

```
1 * FOAF de a
```

```
+1 x2 >= 1;
+1 x4 +1 x8 +1 x12 +1 x16 +1 x20 +1 x24 +1 x28 +1 x32 +1 x36 +1 x40 +1
    x44  >= 1;
+1 ~x4 +1 ~x8 +1 ~x12 +1 ~x16 +1 ~x20 +1 ~x24 +1 ~x28 +1 ~x32 +1 ~x36 +1
     ~x40 +1 ~x44  >= 10;
```

Algoritmo A.3 – Restrições de consulta FOAF

Por fim, basta executar esse arquivo de restrições completo para o resolvedor, no contexto desse trabalho utilizamos o CLASP, os argumentos passados são, **0**, para listar todos os modelos enumerados, **–quiet=0** para detalhar as variáveis e, por fim **-t 12** para utilizar 12 threads.

```
clasp 0 --quiet=0 -t 12 _file_name_
```

Algoritmo A.4 – Comando para executar o resolvedor

Ao executar o comando obtemos a seguinte resposta:

```
c Answer: 1
v -x1 x2 -x3 -x4 -x5 -x6 -x7 x8 -x9 -x10 x11 -x12 x13 -x14 -x15 -x16 x17
v -x18 -x19 -x20 x21 -x22 -x23 -x24 x25 -x26 -x27 -x28 x29 -x30 -x31 -
    x32
v x33 -x34 -x35 -x36 x37 -x38 -x39 -x40 x41 -x42 -x43 -x44 -x45 x46 -x47
v -x48 -x49 -x50 -x51 -x52 -x53 -x54 -x55 -x56 x57
c Answer: 2
v -x1 x2 -x3 -x4 x5 -x6 -x7 -x8 x9 -x10 -x11 -x12 -x13 -x14 x15 -x16 -
    x17
v -x18 -x19 x20 x21 -x22 -x23 -x24 x25 -x26 -x27 -x28 x29 -x30 -x31 -x32
v x33 -x34 -x35 -x36 x37 -x38 -x39 -x40 x41 -x42 -x43 -x44 -x45 -x46 x47
v -x48 -x49 -x50 -x51 -x52 x53 -x54 -x55 -x56 -x57
c Answer: 3
v -x1 x2 -x3 -x4 x5 -x6 -x7 -x8 x9 -x10 -x11 -x12 x13 -x14 -x15 -x16 -
    x17
v -x18 -x19 x20 -x21 -x22 x23 -x24 x25 -x26 -x27 -x28 x29 -x30 -x31 -x32
v x33 -x34 -x35 -x36 x37 -x38 -x39 -x40 x41 -x42 -x43 -x44 -x45 -x46 -
    x47
v -x48 -x49 x50 -x51 -x52 x53 -x54 -x55 -x56 -x57
c Answer: 4
v -x1 x2 -x3 -x4 x5 -x6 -x7 -x8 x9 -x10 -x11 -x12 x13 -x14 -x15 -x16 -
    x17
v -x18 -x19 x20 x21 -x22 -x23 -x24 x25 -x26 -x27 -x28 -x29 -x30 x31 -x32
v x33 -x34 -x35 -x36 x37 -x38 -x39 -x40 x41 -x42 -x43 -x44 x45 -x46 -x47
v -x48 -x49 -x50 -x51 -x52 x53 -x54 -x55 -x56 -x57
s SATISFIABLE
c
c Models         : 4
```

Algoritmo A.5 – Resposta do resolvedor

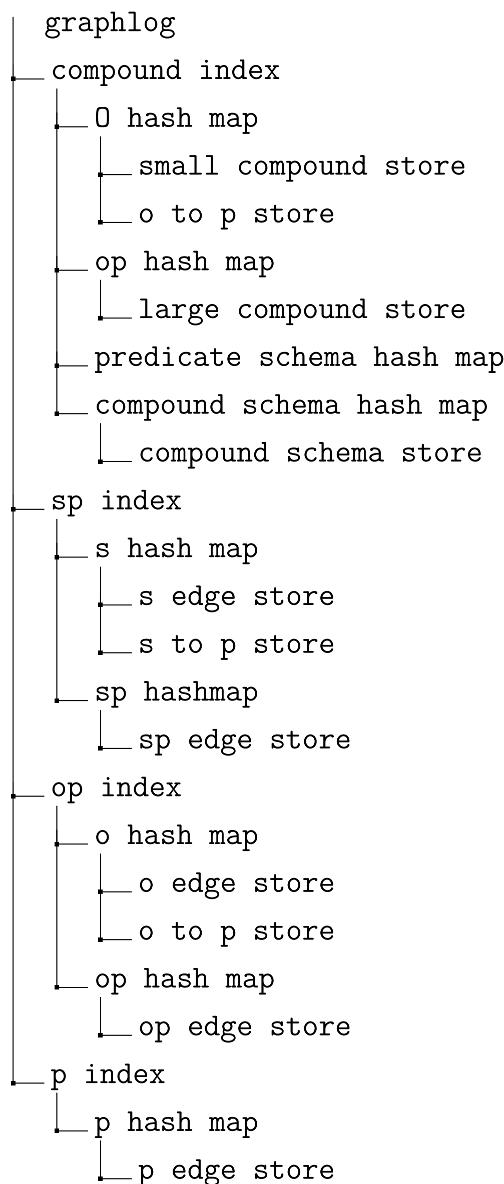
Interpretando a primeira resposta, temos, com valores verdadeiros, as variáveis:

- x_2: Vértice A como inicial;
- x_8: Vértice B como passagem;
- x_{11}: Vértice C como final;
- x_{13}, x_{17}, x_{21}, x_{25}, x_{29}, x_{33}, x_{37} e x_{41}: Vértices D a K como C_0;
- x_{46} e x_{57}: As arestas (A, B) e (B, C).

ANEXO B – LIquid

B.0.1 Indexação

Como descrito em *Nanosecond Indexing of Graph Data With Hash Maps and VLists* (MEYER et al., 2019), os índices utilizados no BDG são armazenados como um *log offset* usando tempo virtual, e são descartados ID's com valores mais elevados. Um índice possui um *Hash Map* que guarda *offsets* para um log filtrado como um lista de tamanho variável, *vlist*. Na representação abaixo, é possível identificar a estrutura de armazenamento dos índices na estrutura do BDG, onde temos o sujeito (S), o par sujeito-predicado (SP), o objeto (O), o par objeto-predicado (OP) e, por fim, o predicado (P). A referida estrutura pode ser observada abaixo.



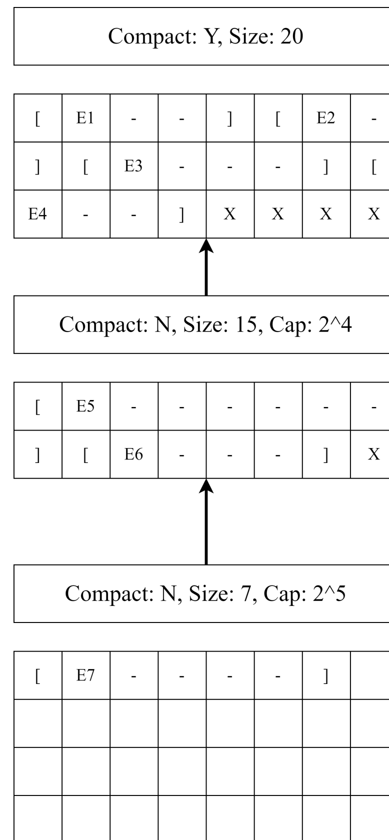


Figura 13 – Diagrama de armazenamento em VList

Fonte: Diagrama retirado de Yang e Yao (2019) Página 4 Figura 3.

B.0.2 Lista de Vetores

Uma *VList* se assemelha a um vetor, porém com um operação mais eficiente de acrescentar novos itens. Essa estrutura, em contrapartida, não necessita copiar dados antigos quando é necessário mais memória. Para isso, ela mantém um ponteiro que se refere à alocação anterior, Fig. 13. A cauda da *VList* possui um cabeçalho que contém o tamanho exato do último pedaço. Todos os outros pedaços contém informações acerca da sua capacidade de armazenamento em $\log_2$, como também quantos estão em uso e, por fim, um *offset* para o próximo segmento. As *VLists* são do tipo LIFO, onde a cabeça da lista é substituída pelo novo item adicionado a cada inserção, e este aponta para o próximo, até que se encontre a cauda da lista.

B.0.3 Hash Map

Um Hash Map é uma forma de armazenar dados de modo a separá-los por meio de *buckets*, como pode-se observar na Fig. 14. Tendo de anteamão a forma com que se aplicará a *hash*, é possível distribuir diversos dados em *buckets* finitos. Porém, ao aplicar essa função, pode haver conflitos, ou seja, haverá a possibilidade de ocorrer o fato de dois dados serem direcionados para um mesmo *bucket*. Dessa forma, poderá assumir uma *hash*

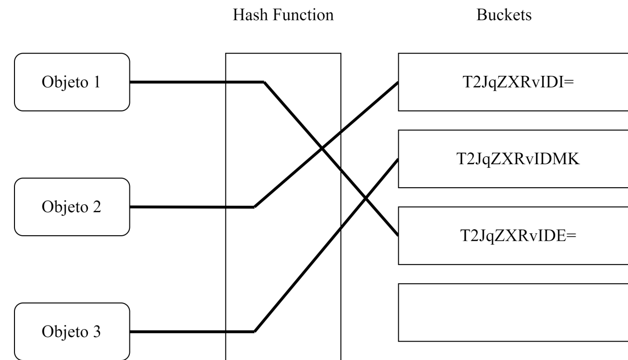


Figura 14 – Exemplo da aplicação de hash map em um objeto genérico

function como descrito na Eq. B.1. Se os dados atribuídos à essa função forem 3 e 30, os dois ocupariam a posição onde a resultante é 3.

$$B = N \bmod 9 \tag{B.1}$$

Para solucionar essa questão, utiliza-se a junção de *hashs* com outras estruturas de dados, como por exemplo a *VList* explicada anteriormente onde, ao ocorrer uma colisão, o último item a ser adicionado se torna a cabeça da lista e aponta para o anterior, e assim sucessivamente.

B.0.4 Visão geral

Tendo em vista os tópicos abordados anteriormente, é possível descrever como os dados são armazenados no BDG, Tab. 5. Nas entradas de dados, são verificadas apenas as últimas atualizações de cada conexão. Após certa quantidade de armazenamento utilizado, são descartadas as mais antigas. Possiblita-se, então, a concorrência de leitura e armazenamento.

Tipo de Index	Chave Hash Map	Entrada de dado
SP	Viggo	(Adição, atuou, Senhor dos aneis)
SP	Peter	(Adição, dirigiu, Senhor dos aneis)
P	dirigiu	(Adição, Peter, Senhor dos aneis)
P	atuou	(Adição, Viggo, Senhor dos aneis)

Tabela 5 – Índices de ligação de arestas

B.0.5 Consultas

O SQL é uma linguagem de consulta de SGBD muito comum e utilizada nos principais BDR como PostgreSQL, SQLServer e MySQL. É uma linguagem bem estruturada e atende bem aos requisitos de um BD.

Porém, quando se trata de um BDG, o SQL se torna de difícil leitura e pouco viável para utilização.

Note que no Alg. B.1 há auto união de tabelas, onde se realiza um *join* de uma tabela consigo mesma, para gerar o relacionamento. Tal fato gera queda de desempenho e traz resultados redundantes. Como pode-se observar no resultado obtido da consulta, há tuplas com valores repetidos, alterando apenas uma coluna. As consultas são feitas em um armazenamento ordenado, gerando uma complexidade $N \times \log N$, o que implica no fato de que, quanto maior o grafo, maior o tempo e memória gastos para a consulta ser realizada.

```
1  WITH g(a, b, c, d, e) AS (
2    WITH
3      e1(a, b) AS (
4        SELECT subject, object FROM edges WHERE subject=2
5          AND predicate=1),
6      e2(b, c) AS (
7        SELECT subject, object FROM edges WHERE predicate=1),
8      e3(c, d) AS (
9        SELECT subject, object FROM edges WHERE predicate=6),
10     e4(c, e) AS (
11       SELECT subject, object FROM edges WHERE predicate=7)
12   SELECT e1.a, e1.b, e2.c, e3.d, e4.e
13     FROM e1 JOIN e2 USING(b) JOIN e3 USING(c)
14     JOIN e4 USING(c))
15 SELECT va.literal AS a, vb.literal AS b, vc.literal AS c,
16     vd.literal AS d, ve.literal AS e
17   FROM
18     g, vertices AS va, vertices AS vb, vertices AS vc,
19       vertices AS vd, vertices AS ve
20   WHERE
21     g.a=va.vertex_id AND
22     g.b=vb.vertex_id AND
23     g.c=vc.vertex_id AND
24     g.d=vd.vertex_id AND
25     g.e=ve.vertex_id;
26
27  a  | b  | c  |  d   |    e
28 ----+----+----+------+--------
29  a1 | b1 | c1 | java | Oracle
30  a1 | b2 | c1 | java | Oracle
31  a1 | b1 | c1 | java | IBM
32  a1 | b2 | c1 | java | IBM
33  a1 | b1 | c1 | C++  | Oracle
34  a1 | b2 | c1 | C++  | Oracle
35  a1 | b1 | c1 | C++  | IBM
```

```
36 a1 | b2 | c1 | C++  | IBM
```

Algoritmo B.1 – Query SQL com seu respectivo resultado (YANG; YAO, 2019).

O BDG abordado em *Nanosecond Indexing of Graph Data With Hash Maps and VLists* (MEYER et al., 2019) é definido por uma capacidade de indexação de dados em grafos com desempenho superior ao da SQL na estrutura de grafos. O banco trabalhado apresenta um armazenamento no formato de datalog para o armazenamento e consultas.

B.0.6 Datalog

A estrutura do BDG é *index-free*, ou seja, não utiliza índices para referenciar os dados. Um BD com esse enfoque guarda em cada nó a referência dos adjacentes. Sendo assim, cada nó atua como um índice dos nós de sua vizinhança. Os índices atuam como um ponteiro indicando o local em memoria onde o próximo nó está (POKORNý, 2015).

Para o armazenamento de arestas, é necessário o sujeito (nó), predicado (tipo de relação) e objeto (nó). Cada operação realizada pode adicionar ou remover essas arestas, que são sempre armazenadas em logs (MEYER et al., 2019). Apenas a última operação será levada em consideração. Essas operações podem ser representadas em datalogs, como representa o Alg. B.2, onde o armazenamento é apenas entre relacionamentos, ou predicados.

```
1  relacao_comutativa(sujeito, objeto):-
2      relacao(sujeito, "relacao", objeto),
3      relacao(objeto, "relacao", sujeito).
4
5  relacao("no-1", "relacao", "no-2").
6  relacao("no-2", "relacao", "no-1").
7
8  | ?- relacao_comutativa("no-1", "no-2").
9
10 /* Verdadeiro */
```

Algoritmo B.2 – Representação Datalog de uma relação comutativa

A referida forma de armazenamento possui um problema quanto a relações N-árias, onde o resultado depende de N parâmetros, pois as arestas não têm um identificador e poderiam se obter várias de mesmo valor.