ESTUDO EXPERIMENTAL DA APLICAÇÃO DE ISOMORFISMO DE GRAFOS NO PROBLEMA DE VERIFICAÇÃO DE MODELOS

Alessandra S. Anyzewski
Maria Claudia S. Boeres
Eduardo Zambon
UFES – Universidade Federal do Espírito Santo
Departamento de Informática
Av. Fernando Ferrari, s/n – Goiabeiras – 29.060–900 – Vitória – ES – Brasil
ale.anyzewski@gmail.com
{boeres, zambon}@inf.ufes.br

RESUMO

Este trabalho propõe uma nova forma de resolver a etapa de isomorfismo de grafos presente na ferramenta GROOVE, usada para verificação de modelos baseados em grafos. O algoritmo de isomorfismo é um dos mais importantes entre os processos do GROOVE, de forma que, ganhos de performance nessa etapa impactam bastante o desempenho da ferramenta. Ao longo da análise de um modelo, o GROOVE compara quanto a isomorfismo cada novo grafo gerado com o conjunto de grafos computados previamente. São testados aqui os filtros de não-isomorfismo: critérios que identificam não-isomorfismos a um baixo custo computacional, e que restringem o número de buscas diretas por isomorfismo a serem executadas. Os experimentos mostram que essa nova abordagem é bastante promissora, capaz de detectar perto de 100% dos não-isomorfismos nas instâncias de teste, com tempos de execução consideravelmente mais baixos quando comparados ao algoritmo original do GROOVE.

PALAVRAS CHAVE. Isomorfismo de Grafos, Verificação de Modelos, GROOVE. Área de classificação principal: Teoria e Algoritmos em Grafos.

ABSTRACT

This work presents a new way of solving the graph isomorphism stage in GROOVE, a tool for graph-based model checking. The graph isomorphism algorithm is one of the most important among GROOVE's processes, such that improvements in this stage improve the tool performance overall. During a model analysis, GROOVE compares with respect to isomorphism each new graph generated with the previously computed graph set. This paper presents non-isomorphism filters: criteria that identifies non-isomorphism with low computational cost, and that restricts the number of direct searches to be executed. The experiments show that this new approach is very promising, and is capable of detecting almost 100% of the non-isomorphisms in the test instances, with execution times considerably lower when compared to GROOVE's original algorithm.

KEYWORDS. Graph Isomorphism, Model Checking, GROOVE. Main Area: Graph Theory and Algorithms.

1. Introdução

O isomorfismo de grafos conforme definido em [Fortin, 1996] é um problema clássico da Teoria de Grafos, que consiste em determinar se é possível definir um mapeamento entre os vértices de dois grafos de forma que sejam respeitadas as conexões definidas por seus arcos. Esse problema dispõe de diversos algoritmos na literatura que o solucionam com bons desempenhos, tais como o *Nauty* [McKay et al., 1981], o *Traces* [McKay e Piperno, 2014] e o *Saucy* [Darga et al., 2008], por exemplo.

Uma variação desse problema consiste em determinar se um grafo é isomorfo a algum dos elementos de um conjunto composto por grafos sabidamente não-isomorfos entre si. Combinar os elementos de um conjunto de tamanho n dois a dois para compará-los gera um custo da ordem de $O(n^2)$. Desta forma, com o intuito de reduzir o custo dessa busca no conjunto de grafos procura-se estabelecer um *certificado* para os grafos, isto é, um número inteiro que represente o grafo, de forma que o isomorfismo seja buscado somente entre pares de grafos com o mesmo certificado [Rensink, 2007]. Porém, o cálculo de um *certificado* que discrimine bem os grafos de um conjunto pode ser oneroso computacionalmente.

Tal problema, conforme descrito, é recorrente em sistemas como o GROOVE (GRaph-based Object-Oriented VErification) – [Rensink, 2004], [Ghamarian et al., 2012] – ferramenta de verificação de modelos (*model checking*) baseados em grafos. A verificação de modelos é uma das formas de realizar a verificação formal de sistemas, cujo objetivo é garantir com formalismo matemático que o funcionamento de um sistema se mantém coerente com suas especificações de projeto. A verificação de sistemas é especialmente interessante quando executada sobre sistemas automáticos críticos, que ofereçam grande risco em caso de falhas. O GROOVE é capaz de modelar e analisar modelos de forma automatizada, e um dos principais objetivos no seu desenvolvimento é processar modelos cada vez maiores com um custo computacional cada vez menor. Denomina-se *exploração do espaço de estados* o processo de enumerar os possíveis estados relativos a um modelo. No GROOVE, cada estado é representado por um grafo.

O algoritmo de isomorfismo é um dos mais importantes entre os processos presentes no GROOVE [Ghamarian et al., 2010], de forma que, ganhos de performance nessa etapa impactam o desempenho da ferramenta como um todo. Grafos isomorfos são reduzidos a um só ao longo da exploração do espaço de estados por um processo chamado *redução por simetria* [Rensink, 2007], [Rensink, 2010].

O GROOVE utiliza a abordagem por *certificados* como padrão, conforme descrito em [Rensink, 2007], [Rensink, 2010] e [Anyzewski, 2016]. O objetivo deste trabalho é propor um novo algoritmo para melhorar a performance do módulo de isomorfismo do GROOVE e, para isso, são investigados filtros de não-isomorfismos: critérios que identificam não-isomorfismos a um baixo custo computacional. Os experimentos mostram que essa nova abordagem é bastante promissora, capaz de detectar perto de 100% dos não-isomorfismos nas instâncias de teste, com tempos de execução consideravelmente mais baixos quando comparados ao algoritmo original do GROOVE.

O restante do artigo é organizado da seguinte forma: na seção 2 são apresentadas as definições e conceitos preliminares, necessários ao entendimento do trabalho; na seção 3 são definidas as instâncias utilizadas para a realização dos experimentos; a seção 4 apresenta a metodologia dos experimentos realizados, a apresentação dos resultados obtidos e sua análise. Finalmente, na seção 5 são destacadas as conclusões e os trabalhos futuros.

2. Definições e Conceitos Preliminares

Nesta seção são apresentados os insumos teóricos utilizados neste trabalho.

2.1. Conceitos de Grafos

O GROOVE opera sobre grafos do tipo direcionado e rotulado, portanto, esse é o tipo de grafo utilizado neste trabalho.

Definição 2.1 (Grafo Direcionado e Rotulado) *Seja Rot um universo finito de rótulos. Um* grafo direcionado e rotulado *é uma tupla $G = \langle V_G, A_G \rangle$ onde*

- V_G *é um conjunto finito de vértices (ou nós);*
- $A_G \subseteq V_G \times Rot \times V_G$ *é um conjunto finito de arcos (ou arestas).*

Para cada arco $a = \langle s, l, t \rangle \in A_G$, está associado um *nó-fonte* s, um *rótulo* l e um *nó-sumidouro* t, denotados por $\mathsf{src}(a)$, $\mathsf{lab}(a)$ e $\mathsf{tgt}(a)$, respectivamente. A presença de arcos caracteriza o grafo como sendo do tipo *direcionado*. A associação de um rótulo ao arco caracteriza o grafo como sendo *rotulado*. Grafos direcionados e rotulados podem ser relacionados entre si por um morfismo de grafos.

Definição 2.2 (Morfismo de Grafos) *Dados dois grafos G e H, um morfismo entre eles é uma função $m : (V_G \cup A_G) \to (V_H \cup A_H)$, tal que*

- $m(V_G) \subseteq V_H; \quad m(A_G) \subseteq A_H;$
- $\mathsf{src}_H \circ m = m \circ \mathsf{src}_G; \quad \mathsf{tgt}_H \circ m = m \circ \mathsf{tgt}_G;$ *e*
- $\mathsf{lab}_H \circ m = \mathsf{lab}_G$.

Isto é, m mantém coerentes as associações entre nós-fonte, nós-sumidouro e rotulações. Escreve-se $m : G \to H$ para indicar que m é um morfismo de G em H. Um isomorfismo é um tipo especial de morfismo de grafos.

Definição 2.3 (Isomorfismo de Grafos) *Dado dois grafos G e H, um isomorfismo entre G e H, denotado $G \simeq H$, é um morfismo $m : G \to H$ tal que m é uma função bijetora.*

Os grafos a serem comparados quanto a isomorfismo são gerados por um processo de transformação de grafos, que é o processo fundamental de funcionamento do GROOVE.

2.2. Transformação de Grafos

Um grafo pode ter sua estrutura modificada através da aplicação de regras de transformação de grafos. O GROOVE executa transformações de grafos para explorar o espaço de estados de um modelo, gerando os grafos que serão comparados quanto a isomorfismo.

Definição 2.4 (Transformação de Grafos) *A transformação de grafos é uma técnica declarativa baseada em regras do tipo $r : L \to R$, onde L e R são grafos relacionados por um morfismo. Transformar um grafo consiste em aplicar uma regra r sobre um* grafo hospedeiro *G (grafo que sofrerá a transformação). Ao encontrar um subgrafo de G correspondente a L, ele será substituído por uma cópia de R, de acordo com a regra r, formando um novo grafo H.*

Um exemplo de transformação de grafo pode ser visto na figura 1 adaptada de [Zambon, 2013]. Na figura, o grafo L é correspondente aos vértices $1, 2$ e 3 de G (função m). Encontrar m é um problema conhecido como casamento de subgrafos (*subgraph matching*) [Rensink e Kuperus, 2009]. O subgrafo de G correspondente a L é substituído então pelo grafo R conforme a regra r, formando o grafo H.

Neste trabalho, não é preciso conhecer detalhes do processo (algoritmo) de transformação de grafos que gera os grafos a serem analisados entre si quanto ao isomorfismo. No caso do GROOVE, o método de transformação de grafos utilizado é conhecido como *transformação algébrica*, cujos detalhes são omitidos aqui mas podem ser vistos em [Rozenberg, 1997].

2.3. Sistema de Transição de Grafos

O conjunto $\mathcal{S}$ de todos os estados gerados pelas operações de transformação de grafos e o conjunto $\to$ de todas as transições aplicadas sobre os estados formam um *Sistema de Transição de Grafos* (STG). Neste trabalho tratamos de STGs de tamanhos finitos. A construção dos conjuntos que compõem um STG é chamada de *exploração de espaço de estados*. O ponto de partida para a exploração é a gramática de grafos a ser explorada.

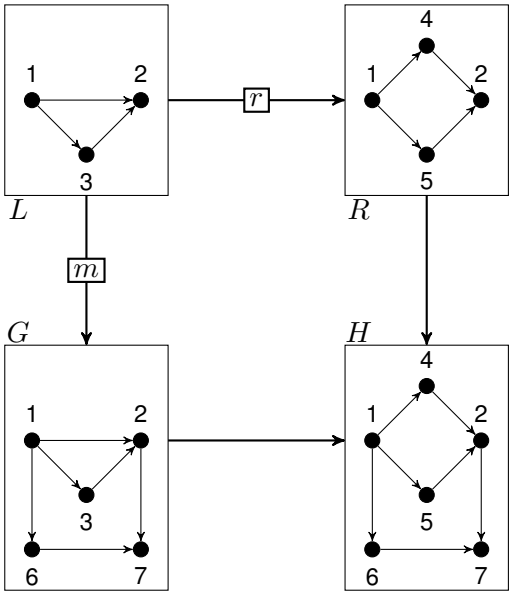

Figura 1: Exemplo de uma transformação de grafo (adaptado de [Zambon, 2013]).

Definição 2.5 (Gramática de Grafos) *Uma* gramática de grafos $\mathcal{G} = \langle \mathcal{R}, G_0 \rangle$ *é uma tupla composta por um conjunto de regras de transformação de grafos* $\mathcal{R}$ *e um grafo inicial* G_0.

A exploração do espaço de estados que constrói um STG a partir de uma gramática de grafos pode ser descrita pelo algoritmo 1, onde $\mathcal{N}$ representa o conjunto de estados *novos*, ainda a serem explorados.

Entrada: gramática $\mathcal{G} = \langle G_0, \mathcal{R} \rangle$
Saída: $STG = \langle \mathcal{S}, \rightarrow \rangle$
1 $\mathcal{S} := \emptyset,\ \rightarrow := \emptyset,\ \mathcal{N} := \{G_0\}$;
2 **enquanto** $\mathcal{N} \neq \emptyset$ **faça**
3 **para** *cada* $G \in \mathcal{N}$ **faça**
4 $\mathcal{N} := \mathcal{N} \setminus \{G\}$;
5 **para** $r \in \mathcal{R},\ m \in match(r, G)$ **faça**
6 $H := transformar(r, m, G)$;
7 **se** $novo(H, \mathcal{S})$ **então**
8 $\mathcal{S} := \mathcal{S} \cup \{H\},\ \mathcal{N} := \mathcal{N} \cup \{H\}$;
9 **fim**
10 $\rightarrow := \rightarrow \cup \{G \xrightarrow{r} H\}$;
11 **fim**
12 **fim**
13 **fim**

Algoritmo 1: Exploração do espaço de estados

A linha 1 inicializa o sistema de transformação com conjuntos vazios, e inicializa o conjunto de estados *novos* $\mathcal{N}$ com o grafo inicial. Enquanto houverem estados a serem explorados, o algoritmo escolhe um deles na linha 3 e o remove do conjunto $\mathcal{N}$ na linha 4, de acordo com a estratégia de exploração. A estratégia adotada (por exemplo, busca em largura ou busca em profundidade) não tem influência sobre o conjunto de estados que será encontrado ao fim da exploração já que a busca é exaustiva.

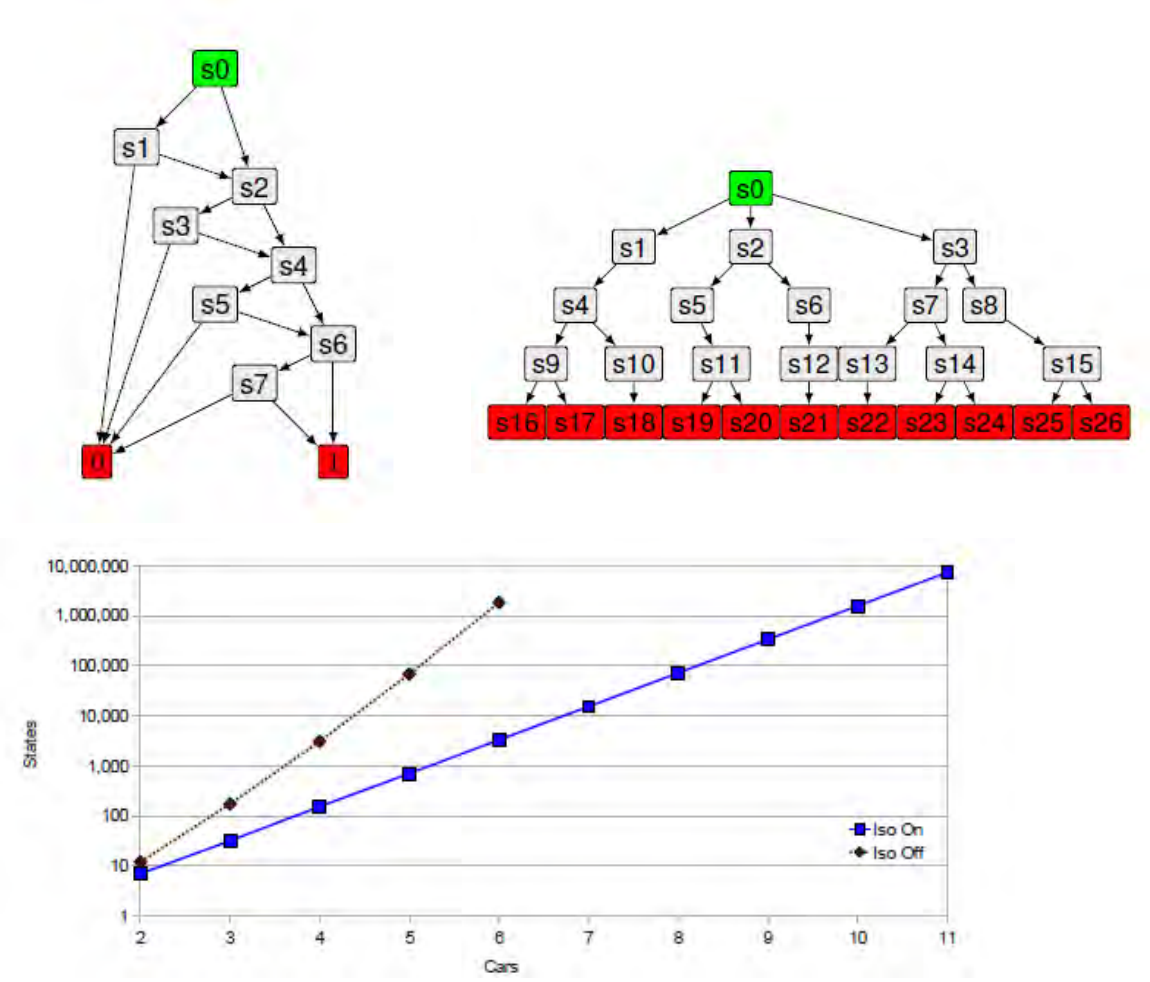

Figura 2: O efeito da redução por simetria no GROOVE (gráfico extraído de [Ghamarian et al., 2010]).

A linha 5 trata de encontrar as correspondências entre as regras de transformação e o estado que está sendo explorado (G). Ao aplicar a regra r em G é gerado o estado H (linha 6), o qual é checado quanto a isomorfismo em relação ao conjunto $\mathcal{S}$ na linha 7, pelo método $novo(H, \mathcal{S})$. Na linha 10, a transição executada na linha 6 é armazenada no conjunto de transições $\rightarrow$.

Uma gramática de grafos gera um STG que representa todas as possibilidades de estados e transições do modelo a partir de um estado inicial, desde que esteja condicionada a ter em seu conjunto $\mathcal{S}$ apenas um representante de cada conjunto de grafos isomorfos (conforme verificado pelo método $novo$ na linha 7 do algoritmo 1). Como o STG representa todas as possibilidades de aplicações das regras na exploração de sua respectiva gramática, dizemos que a gramática foi explorada de forma *exaustiva*.

Na exploração de um espaço de estados, uma etapa essencial é checar, para cada novo estado gerado, se ele é ou não isomorfo a cada um dos estados gerados previamente. Comparar o novo grafo com cada um dos anteriores geraria um custo computacional da ordem de $O(|\mathcal{S}|^2)$. Uma vez que $\mathcal{S}$ é, em geral, um conjunto da ordem de milhões de estados, esse custo pode ser muito alto e qualquer melhora na performance tem um grande impacto no desempenho do GROOVE. Daí a importância de escolher bem a estratégia de detecção de isomorfismos durante essa exploração.

O impacto da redução por simetria pode ser visto na figura 2. Na parte superior da figura há uma ilustração de como fica a exploração de um espaço de estados com e sem a redução de simetria respectivamente, onde cada quadrado representa um estado. Assumindo que há apenas

Tabela 1: Características das Gramáticas de Teste.

Gramáticas	Média arestas	Média nós	Total de Grafos	Número de Grafos Não-Isomorfos
Append	22.4	25.4	4325	1720
Ad-hoc	10.9	4.9	2448	117
Gossip	2.8	8	1923	55
Filósofos	3.7	8	972	247
Bad_tic_tac_toe	8.2	19	6046	6046

duas classes de grafos isomorfos nos estados finais da gramática (em vermelho na figura), é possível ver a redução no número de estados gerados/explorados quando se usa a verificação de isomorfismo.

A parte inferior da figura 2 mostra um gráfico com o número de estados obtidos na exploração de uma gramática em função do tamanho do grafo inicial, com e sem a redução por simetria (verificação de isomorfismo). Pode-se notar dos exemplos a importância de manter a redução por simetria no processo de exploração, já que manter mais estados pode acarretar em um grande uso da memória, a ponto de até mesmo inviabilizar a exploração da gramática.

Os filtros de não-isomorfismo são critérios simplificados de diferenciação entre dois grafos, criados com o objetivo de detectar não-isomorfismos a um baixo custo computacional. A proposta desta pesquisa é determinar quão bem os filtros atuam na redução do número de buscas diretas por isomorfismo a serem executadas sobre os conjuntos de grafos gerados na exploração do espaço de estados.

3. Base de Dados Experimental e Descrição dos Filtros

3.1. Gramáticas de Teste

A base de grafos para testes foi gerada com as gramáticas utilizadas por [Rensink, 2007], [Rensink, 2010] e [Anyzewski, 2016]. São elas:

Append modela a inserção de um elemento no fim de uma estrutura de dados do tipo fila de tamanho 5, quando há 3 invocações de inserção concorrentes.

Ad-hoc é o modelo de uma rede *ad-hoc*, que consiste de um conjunto de 4 nós conectados, mas sem a presença de um servidor central. Cada nó só consegue consultar as informações de seus 2 vizinhos mais próximos.

Gossip modela o problema "*gossiping girls*" descrito em [Hurkens, 2000]. Todos os grafos gerados ao longo da exploração tem o mesmo número de nós, correspondente às 4 garotas a fazer fofocas entre si.

Filósofos é o modelo para o conhecido problema do jantar dos filósofos. O número de nós dos grafos gerados é fixo (o dobro do número de filósofos, neste caso, 4), o que gera uma grande simetria nesse problema.

Bad tic-tac-toe é o jogo da velha em um tabuleiro 3x3. O número de nós é fixo mas o problema não tem simetria, já que, nesse caso, os campos de marcação do jogo da velha são diferenciados entre si. Este é um caso com muita similaridade entre as instâncias em termos de estrutura, mas totalmente diferentes em relação aos rótulos. Esse caso foi criado para este trabalho, a fim de propor um problema de difícil solução para os filtros de não-isomorfismos.

As características das gramáticas em relação ao número médio de arcos, número médio de nós, número total de grafos gerados pela exploração do espaço de estados e número de grafos não-isomorfos no conjunto são apresentadas na tabela 1 para cada uma das gramáticas. Podemos notar que a gramática *Gossip* é proporcionalmente a que tem mais grafos isomorfos, enquanto a *Bad tic-tac-toe* não contém nenhum isomorfo (graças à diferenciação dos rótulos referentes a cada uma das posições do jogo da velha).

Para ilustrar o número de atributos presentes nas instâncias de grafos tratadas usualmente no GROOVE, é apresentado o exemplo da figura 3, proveniente da exploração da gramática de

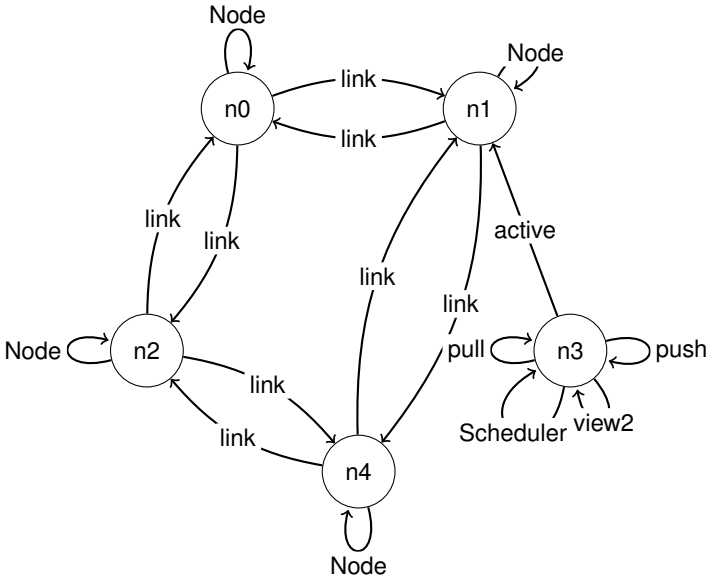

Figura 3: Exemplo de grafo da gramática *Ad-hoc*.

teste *Ad-hoc*. Essa gramática é, dentre as testadas, uma das que produz grafos com maior grau de similaridade entre os nós, mas ainda assim, é possível notar que há diversos invariantes presentes no grafo que podem ajudar a diferenciar os nós entre si (por exemplo, a distância de cada nó ao nó rotulado como *Scheduler*), o que ajuda a diferenciar os grafos entre si, descartando a hipótese de eles serem isomorfos.

O algoritmo 2 sintetiza a metodologia de execução dos testes realizados com os filtros.

Entrada: $\mathcal{S}$:= grafos explorados na gramática *sem* verificação de isomorfismo
para $G \in \mathcal{S}$ **faça**
  para $G' \in \mathcal{S}$ *tal que* $G' \neq G$ **faça**
    se G *e* G' *ainda não foram comparados quanto a isomorfismo* **então**
      $podemSerIso(G, G')$;
    fim
  fim
fim

Algoritmo 2: Contexto de execução dos filtros.

Na metodologia de testes empregada, cada par de grafos do conjunto de grafos relativo a uma gramática foi submetido à detecção de isomorfismo, para todas as gramáticas. Isso significa que, para um conjunto de dados contendo $|\mathcal{S}|$ grafos, foram feitos $\frac{|\mathcal{S}|\times(|\mathcal{S}|-1)}{2}$ testes (combinação de $|\mathcal{S}|$ grafos 2 a 2). O valor de $|\mathcal{S}|$ para cada gramática corresponde à quarta coluna da tabela 1.

Na base de dados de teste já há a informação de isomorfismo relativa a todas as combinações de grafos, e assim, os resultados das execuções do método $podemSerIso$ são comparados com os resultados esperados nas análises de desempenho apresentadas na seção 4, de resultados computacionais. A implementação do método $podemSerIso$ varia conforme o filtro de não-isomorfismo a ser executado, cujas descrições seguem na próxima seção.

3.2. Descrição dos Filtros

Foram propostos seis tipos de filtros, a serem detalhados a seguir. Todos recebem como entrada um par de grafos a serem comparados.

1. **Contagem de nós (NÓS):** Compara quantos nós tem os grafos. Se o número de nós é diferente, com certeza os grafos não são isomorfos.
2. **Contagem de arcos (ARS):** Análogo à contagem de nós.
3. **Contagem de rótulos (RÓTS):** Conta o número de ocorrências de cada um dos rótulos do grafo. São listados os diferentes rótulos relativos ao arcos de um dos grafos de entrada, sem repetição. O filtro itera sobre essa lista contandos quantos arcos têm esse rótulo para os dois grafos de entrada. Se para algum rótulo esses valores forem diferentes, os grafos não podem ser isomorfos.
4. **Graus dos vizinhos n-dist (GR):** Este filtro percorre os vizinhos de cada um dos nós, armazenando o somatório de todos os graus de saída e dos graus de entrada dos nós visitados. A implementação desse filtro é iterativa sobre as distâncias, percorrendo as distâncias 1 até n em relação aos vizinhos. Nos testes apresentados, consideram-se os valores de n = 1, 2, 3 e 4.
5. **Método das potências adaptado (POT):** Esse método foi apresentado por [Baroni, 2012] e consiste em atribuir um certificado inicial para cada nó (por exemplo, atribuir o valor 1 para todos os nós) e iterar sobre esses valores de forma que o novo certificado de nó seja a soma dos certificados de seus vizinhos.
   Aqui, como trata-se de grafos direcionados, são iterados os invariantes separadamente para os vizinhos-fonte e os vizinhos-sumidouro. Os invariantes de nós são inicializados com os valores dos graus de entrada e de saída. Os invariantes de vizinhos-fonte/sumidouro são somados compondo uma *potência de entrada* e uma *potência de saída*. Para que dois grafos possam ser isomorfos as potências devem ter o mesmo conteúdo.
6. **Sequência de rótulos dos vizinhos n-dist (SROT):** Esse filtro é análogo ao fitro de graus do item 4, porém, ao invés de percorrer os vizinhos armazenando as somas dos seus graus, ele percorre os arcos que levam aos vizinhos e armazena seus rótulos. Os rótulos associados a uma aresta a uma distância n dos nós iniciais são ordenados lexicograficamente em vetores correspondentes a cada nó. Se os dois grafos não possuem o mesmo conjunto de vetores de rótulos, eles não podem ser isomorfos.

A principal diferença entre o filtro de graus e o de potências é que o de graus, ao longo das iterações, busca os vizinhos mais distantes (2-dist, 3-dist, etc), enquanto o filtro de potências sempre consulta seus vizinhos 1-dist. Além disso o filtro de graus compõe um vetor para cada nó ao longo das iterações enquanto o filtro de potências sempre tem somente um inteiro associado ao nó. Outras discussões sobre os filtros, com detalhes das implementações e exemplos de suas execuções podem ser encontradas em [Anyzewski, 2016].

4. Resultados Computacionais

Os filtros foram testados individualmente e combinados, gerando filtros compostos. Os filtros foram executados para cada uma das gramáticas de teste em uma máquina Intel core i7, com 8GB de RAM. As implementações foram feitas em Java (linguagem em que foi desenvolvido o GROOVE), compiladas e executadas com o JDK 1.7.0_79. Os resultados foram comparados em termos de eficácia e de tempo computacional em milissegundos. A eficácia (expressa em percentagem) se refere ao número de pares não-isomorfos que foi detectado, em relação ao total de pares não-isomorfos presentes no conjunto de dados.

4.1. Filtros Individuais

Os resultados dos filtros individuais podem ser observados nas tabelas 2 e 3. Nas colunas estão listados os filtros enquanto nas linhas estão os tempos de execução em milissegundos e a eficácia em porcentagem, medidos para cada gramática.

- Mesmo sendo simples, os filtros podem atingir eficácias percentuais muito altas.

Tabela 2: Resultados dos filtros individuais (parte 1)

Grams	Res	NÓS	ARS	RÓTS	GR1	GR2	GR3	GR4
Append	t (ms)	363	325	54116	271600	826953	1475782	9350650
	efic (%)	91.19	97.49	97.90	99.92	99.94	99.95	99.97
Ad-hoc	t (ms)	108	113	7023	18555	55564	109727	167817
	efic (%)	6.46	76.94	61.35	96.61	99.09	99.46	99.46
Gossip	t (ms)	66	69	3590	17764	51862	82574	102648
	efic (%)	0	86.74	87.00	99.93	100	100	100
Filósofos	t (ms)	10	13	991	4217	12256	17143	20189
	efic (%)	0	74.55	98.37	84.39	84.39	84.39	84.39
Tic_tac	t (ms)	630	594	87383	246548	500747	649258	802128
	efic (%)	0	80.73	80.73	80.73	80.73	80.73	80.73

Tabela 3: Resultados dos filtros individuais (parte 2)

Grams	Res	SROT1	SROT2	SROT3	SROT4	POT1	POT2	POT3	POT4
Append	t (ms)	49433	1189288	1967869	2786872	343373	474513	620336	759188
	efic (%)	99.93	100	100	100	99.94	99.95	99.97	99.98
Ad-hoc	t (ms)	39132	103429	190004	281929	24385	32922	44110	54445
	efic (%)	99.25	99.97	100	100	99.07	99.44	99.44	99.44
Gossip	t (ms)	33117	76715	105769	133131	21420	30746	40467	50344
	efic (%)	99.93	100	100	100	100	100	100	100
Filósofos	t (ms)	7920	16098	19373	22080	4658	7606	10251	12866
	efic (%)	98.37	99.93	99.93	99.93	84.39	84.39	84.39	84.39
Tic_tac	t (ms)	399462	552541	682062	816702	269071	374513	483534	601721
	efic (%)	100	100	100	100	80.73	80.73	80.73	80.73

- Os três primeiros filtros (NÓS, ARS e RÓTS) tem uma, duas, ou mais ordens de grandeza a menos no custo computacional em relação aos outros filtros.
- O filtro de ARS é o que tem melhor relação eficácia/tempo computacional entre os filtros de contagem.
- Os filtros de graus e de potências tem eficácias parecidas, sendo o de potências um pouco mais eficiente. Porém em relação aos custos computacionais eles se comportam diferentemente. O filtro de GR1 tem custo menor em relação ao de POT1. Porém, o filtro de graus tem um custo crescendo a uma taxa aproximadamente exponencial conforme aumentamos a distância, enquanto o de potências tem taxa aproximadamente linear.
- O filtro mais eficiente entre eles é o de sequência de rótulos, mas ele é também o mais caro.
- Para os filtros que vão aumentando o alcance da análise (os que usam os vizinhos n-dist), podemos perceber que o 1-dist já atinge eficácias altas. Utilizar distâncias maiores aumenta bastante o custo computacional mas não chega a compensar em termos de eficácia.
- A diferença de proporção de não-isomorfos na base de grafos em cada gramática não chega a ter impacto na eficácia esperada para o filtro. São muito mais impactantes as características de estrutura das gramáticas, de forma que, as gramáticas que contém maior número de automorfismos (à parte a rotulação dos arcos) como a *Ad-hoc*, a *Phil* e especialmente a *Bad_tic_tac_toe* são as mais difíceis para a diferenciação por filtros.

Com essas observações, concluimos que o filtro SROT1 é o melhor filtro individual proposto, já que ele atinge quase 100% de eficácia para as variadas gramáticas de teste sem penalizar muito o tempo de execução.

4.2. Filtros Compostos

É interessante notar que não necessariamente os não-isomorfismos detectados por cada um dos filtros individuais, testados na seção anterior, tem interseção. Aqui são propostas combinações dos filtros testados individualmente a fim de melhorar seus desempenhos. São propostas sete

Tabela 4: Resultados dos filtros compostos

Gramáticas	Res	COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
Append	t (ms)	2041	9200	10385	13865	9633	10920	10207
	efic (%)	97.97	99.92	99.94	99.93	99.93	99.95	99.95
Ad-hoc	t (ms)	1783	6214	7528	11207	8163	8151	8116
	efic (%)	87.34	97.66	99.49	99.26	99.25	99.87	99.87
Gossip	t (ms)	771	3266	3658	5670	4510	4865	5140
	efic (%)	88.88	99.93	100	99.93	99.93	100	100
Filósofos	t (ms)	360	945	997	1444	1674	1699	2055
	efic (%)	98.37	98.37	98.37	98.37	98.37	98.37	98.37
Bad_tic_tac_toe	t (ms)	20982	74048	79377	101695	145208	139769	187797
	efic (%)	80.73	80.73	80.73	100	100	100	100

composições de filtros. Todos elas contém os três primeiros filtros dentre os listados na seção 3.2, já que esses filtros são muito mais rápidos de executar que os demais. Os filtros são executados na ordem em que são listados.

COMP 1: ARS + NÓS + RÓTS;
COMP 2: COMP 1 + GR1;
COMP 3: COMP 1 + POT1;
COMP 4: COMP 1 + SROT1;
COMP 5: COMP 1 + GR1 + SROT1;
COMP 6: COMP 1 + POT1 + SROT1;
COMP 7: COMP 1 + GR1 + POT1 + SROT1;

Os resultados dos testes dos filtros compostos estão apresentados na tabela 4. Podemos concluir que:

- O COMP1 é aproximadamente cinco vezes mais rápido que os outros, e responde bem para o caso em que os grafos gerados pela gramática são mais variados (gramática *Append*).
- Os filtros COMP4 a COMP7 detectam quase 100% dos não-isomorfismos em todas as gramáticas. A gramática *Bad_tic_tac_toe* é decisiva na escolha do melhor filtro, pois, diferenciar seus não-isomorfismos está muito ligado a mapear seus rótulos. Os filtros do 4 em diante contem o filtro sequência de rótulos, o que faz eles atingirem 100% de diferenciação para essa gramática.
- Dos últimos quatro filtros compostos, os mais rápidos são o COMP5 e o COMP6. A diferença entre eles é que o 5 usa o filtro de graus 1-dist e o 6 usa o de potências 1-dist. Já vimos nos testes individuais que, para o caso 1-dist, o critério de graus é em média mais rápido que o de potências.
- Os tempos de execução para todos os filtros compostos são bastante menores que os dos filtros individuais apresentados nas tabelas 2 e 3 (exceto os filtros de contagem).

Com base nessas observações, o filtro COMP5 é eleito como a melhor combinação para resolver o problema de não-isomorfismo entre as composições de filtros testadas.

4.3. Uso de Certificados como Filtros

Foi visto que os filtros são bem eficientes, e que podem evitar a maior parte dos cálculos de certificados que seriam feitos para determinar isomorfismos. Nesta seção busca-se checar como os filtros e os certificados se comparam em termos de tempo de execução. Os resultados da execução dos cálculos de certificados do GROOVE para as gramáticas de teste são mostrados na tabela 5.

Tabela 5: Resultados do uso dos certificados como filtros

Gramáticas	PR sQS	PR cQS
Append	139910	169770
Ad-hoc network	11984	12996
Gossip priorities	10469	28940
Filósofos	2029	2124
Bad_tic_tac_toe	152807	126167

Tabela 6: Relação entre os tempos computacionais do PR cQS e do COMP5

Gramáticas	tempo COMP5 / tempo PR cQS
Append	5,7%
Ad-hoc network	62,8%
Gossip priorities	15,6%
Filósofos	78,7%
Bad_tic_tac_toe	117,4%

A sigla *PR* representa o algoritmo *Partition Refiner* que é o adotado atualmente no GROOVE [Rensink, 2010]. *QS* indica quebra de simetria, e assim, *cQS* significa "com quebra de simetria". Todos os casos tiveram eficácia de 100% na detecção de não-isomorfismos. Dessa forma, as linhas contendo os valores de eficácias foram omitidas na tabela 5.

Podemos notar, comparando as tabelas 4 e 5, que os filtros têm muitas vezes tempos de execução mais baixos que qualquer um dos certificados e em algumas situações atingem até 100% de eficácia. A tabela 6 relaciona diretamente o tempo de execução do filtro composto que foi eleito o melhor (COMP5) com o algoritmo PR cQS do GROOVE. Nela pode-se notar que o filtro COMP5 tem performances melhores que o PR cQS para todas as gramáticas exceto a *Bad_tic_tac_toe*. Isso ocorre justamente devido às particularidades dessa gramática, de possuir grafos com várias estruturas idênticas, diferenciáveis apenas pela disposição dos rótulos. Mas pode-se observar que, por exemplo, que o filtro COMP4 é mais rápido que o PR cQS para a gramática *Bad_tic_tac_toe*, embora seja mais lento que o COMP5 para os demais casos de teste.

Os resultados apresentados na tabela 5 indicam que é possível melhorar (reduzir) consideravelmente o tempo de execução dos testes de não-isomorfismo substituindo o algoritmo atualmente implementado no GROOVE por um filtro composto. Com isso, valida-se a abordagem por filtros como sendo uma boa estratégia quando comparada à implementação atual do GROOVE.

5. Conclusões e Trabalhos Futuros

Conclui-se dos experimentos realizados neste trabalho que os filtros podem ser um artefato interessante na detecção de (não) isomorfismos do GROOVE. Há critérios mais simples que os certificados, capazes de eliminar a possibilidade de os grafos serem isomorfos, com um tempo computacional muito mais reduzido.

Uma possibilidade de trabalho futuro é integrar a implementação descrita aqui no código em produção do GROOVE. Atualmente essa integração depende de modificar a estrutura de armazenamento de grafos no GROOVE, já que, os certificados de grafo no código original, além de serem usados para determinação de isomorfismo são usados também como chave para o armazenamento dos novos grafos gerados. O código desenvolvido neste trabalho é *open source*, e está disponível para *download* em `https://sourceforge.net/p/groove-ale/code/`.

Outro importante trabalho futuro é aprofundar os estudos sobre os filtros, propondo novos critérios e novas combinações de filtros, realizando novos experimentos com outras gramáticas de teste.

Referências

Anyzewski, A. S. (2016). Estudo experimental da aplicação do algoritmo IVL na etapa de detecção de isomorfismos do GROOVE. Dissertação de mestrado, Universidade Federal do Espírito Santo.

Baroni, M. D. V. (2012). Um estudo da eficiência da autocentralidade no problema de isomorfismo de grafos. Dissertação de mestrado, Universidade Federal do Espírito Santo, Vitória - ES.

Darga, P. T., Sakallah, K. A., e Markov, I. L. (2008). Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th Annual Design Automation Conference*, p. 149–154. ACM.

Fortin, S. (1996). The graph isomorphism problem. Technical report, Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada.

Ghamarian, A. H., Mol, M., Rensink, A., e Zambon, E. (2010). Solving the topology analysis case study with GROOVE. Technical report, University of Twente.

Ghamarian, A. H., de Mol, M., Rensink, A., Zambon, E., e Zimakova, M. (2012). Modelling and analysis using GROOVE. *International journal on software tools for technology transfer*, 14(1): 15–40.

Hurkens, C. A. (2000). Spreading gossip efficiently. *Nieuw Archief voor Wiskunde*, 1:208–210.

McKay, B. D. et al. (1981). *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, US.

McKay, B. D. e Piperno, A. (2014). Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112.

Rensink, A. (2004). The GROOVE simulator: A tool for state space generation. In *Applications of Graph Transformations with Industrial Relevance*, p. 479–485. Springer.

Rensink, A. (2007). Isomorphism checking in GROOVE. *Electronic Communications of the EASST*, 1.

Rensink, A. (2010). Isomorphism checking for symmetry reduction. Technical report, Centre for Telematics and Information Technology, University of Twente.

Rensink, A. e Kuperus, J.-H. (2009). Repotting the geraniums: on nested graph transformation rules. *Electronic Communications of the EASST*, 18.

Rozenberg, G. (1997). *Handbook of Graph Grammars and Comp.*, volume 1. World scientific.

Zambon, E. (2013). *Abstract Graph Transformation Theory and Practice*. Tese de doutorado, Centre for Telematics and Information Technology, University of Twente.