Aplicação de Métodos de Busca Guiada na Exploração do Espaço de Estados de Gramáticas de Grafos

Alexandro Souza Ramos
Maria Claudia Silva Boeres
Eduardo Zambon
UFES - Universidade Federal do Espírito Santo
Departamento de Informática
Av. Fernando Ferrari, s/n - Goiabeiras - 29.060-900 - Vitória - ES - Brasil
alexandroramos@icloud.com
{boeres,zambon}@inf.ufes.br

RESUMO

O GROOVE é uma ferramenta de transformação de grafos utilizada para criar e verificar modelos baseados em grafos. Para tal verificação, é realizada a exploração exaustiva do espaço de estados onde todos os possíveis estados relativos a um modelo são gerados. Contudo, existem importantes classes de sistemas que geram um espaço de estados muito grande ou até mesmo infinito, inviabilizando o uso da exploração exaustiva. Neste trabalho, é proposto um algoritmo de busca guiada que utiliza uma métrica de distância entre grafos para tomar decisões ao caminhar na busca pelo estado objetivo. Os experimentos mostram que a busca guiada é bem eficiente quando comparado com a exploração exaustiva, permitindo a redução de quase 95% no número de estados explorados.

PALAVRAS CHAVE. Busca Guiada, Métricas de Distância, GROOVE.

ABSTRACT

GROOVE is a general purpose graph transformation toolset used for system modelling that performs model checking. In model checking, an exhaustive exploration mechanism is used where all possible states of a model are generated. However, there are important classes of systems that have extremely large or even infinite state spaces and therefore cannot be (fully) explored using an exhaustive exploration. This paper presents a guided search algorithm that uses a metric for distance between graphs to make decisions when walking to find a goal state. Experimental results indicate that guided search leads to nearly a 95% reduction of the explored states in comparison to an exhaustive exploration.

KEYWORDS. Guided Search, Distance Metrics, GROOVE.

1. Introdução

Ao longo dos anos, a tecnologia vem transformando a forma de atuação de empresas, entidades, governos e pessoas. A utilização de sistemas de computação é cada vez maior, sendo necessário garantir o correto funcionamento destes, uma vez que comportamentos inesperados podem gerar graves consequências. A medida que esses sistemas evoluem e tornam-se cada vez mais complexos, é necessário ter mecanismos eficientes para avaliar seus comportamentos. Os *métodos formais* reúnem técnicas baseadas em formalismos matemáticos para a especificação, desenvolvimento e verificação de sistemas.

Em muitos casos, o uso de métodos formais requer que um sistema seja descrito através de um modelo. Assim, pode-se utilizar a *verificação de modelos* (*model checking*) [Baier et al., 2008] do sistema para explorar todos os possíveis comportamentos, garantindo através de um rigor matemático, que o seu funcionamento se mantém coerente com as especificações de projeto, sem erros ou inconsistências.

Em geral, os sistemas em questão são compostos por objetos que possuem relação entre si, sendo facilmente representados por um grafo, onde os nós descrevem os objetos e os arcos descrevem as relações entre os objetos. Além disso, os objetos podem ser criados ou excluídos e suas relações podem mudar, criando um cenário adequado para a *transformação de grafos* [Rozenberg, 1997].

A transformação de grafos tem sido defendida como um formalismo flexível, adequado para modelar sistemas complexos e com configurações dinâmicas. O GROOVE [Rensink, 2004], [Ghamarian et al., 2012] é uma ferramenta de transformação de grafos que permite 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.

A *exploração do espaço de estados* é o processo responsável por enumerar os possíveis estados relativos a um modelo. No GROOVE, cada estado é representado por um grafo e as transições entre estados correspondem a transformação de grafos. Atualmente a exploração é realizada de forma exaustiva, ou seja, a partir do estado inicial, todos os estados sucessores possíveis são gerados de acordo com o modelo e armazenados no espaço de estados. Isso gera um importante fonte de informação sobre o sistema que pode ser utilizada para análise posterior.

A exploração exaustiva pode ser aplicada em diversos problemas, mas seu uso é inviabilizado no domínio de sistemas que possuem um espaço de estados muito grande ou infinito. São exemplos, modelos de sistemas que manipulam estrutura de dados na memória ou redes de comunicação de grande escala [Zambon, 2013].

Diante desse cenário, em oposição à exploração exaustiva, surge a busca guiada, um algoritmo capaz de explorar parcialmente o espaço de estados. A partir do estado inicial é investigado a presença ou ausência de um caminho até um ou mais estados objetivos. Um requisito essencial de uma busca guiada é a métrica, utilizada para definir a ordem de armazenamento e exploração dos estados. A métrica calcula uma distância quantitativa entre o estado atual e o estado objetivo.

Este trabalho apresenta a implementação e experimentação de um algoritmo de busca guiada no GROOVE. Os experimentos mostram que esse algoritmo é bem eficiente quando comparado com a exploração exaustiva, permitindo a redução de quase 95% no número de estados explorados.

O restante do artigo é organizado da seguinte forma: na seção 2 são apresentadas os conceitos preliminares sobre a transformação de grafos; na seção 3 são ilustradas as estratégias de exploração dos espaço de estados; a seção 4 apresenta as métricas de distância; a base de dados experimentais é apresentada na seção 5; na seção 6 são discutidos os resultados computacionais. Finalmente a seção 7 destaca as conclusões e trabalhos futuros.

2. Conceitos Preliminares

Nesta seção são apresentados os conceitos preliminares e definições utilizadas neste trabalho.

2.1. Grafos

O GROOVE realiza transformações em grafos simples e rotulados. Dessa forma, os grafos de interesse deste trabalho possuem as mesmas características.

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

- *N_G é um conjunto finito de nós;*
- *$A_G \subseteq N_G \times Rot \times N_G$ é um conjunto finito de arcos.*

Para cada arco $a = \langle s, l, t \rangle \in A_G$, está associado um *nó-origem* s, um *rótulo* l e um *nó-destino* t, denotados por $\mathsf{src}(a)$, $\mathsf{lab}(a)$ e $\mathsf{tgt}(a)$, respectivamente. A definição do conjunto de arcos A_G caracteriza o grafo como sendo *direcionado*. A associação de um rótulo ao arco caracteriza o grafo como sendo *rotulado*.

2.2. Transformação de Grafos

A Transformação de Grafos é uma técnica para criar um novo grafo usando as condições descritas em uma regra que é aplicada ao grafo original.

Definição 2.2 (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.*

A Figura 1, adaptada de [Zambon, 2013], mostra um exemplo de transformação de grafos. O grafo L é induzido pelos vértices $1, 2$ e 3 de G (função m). Obter a função m é o problema de correspondência de grafos, conhecido como casamento de subgrafos [Rensink e Kuperus, 2009]. No restante do texto, assume-se que a função $match$ resolve esse problema. O subgrafo de G correspondente a L é então substituído por uma cópia de R gerando o novo grafo H.

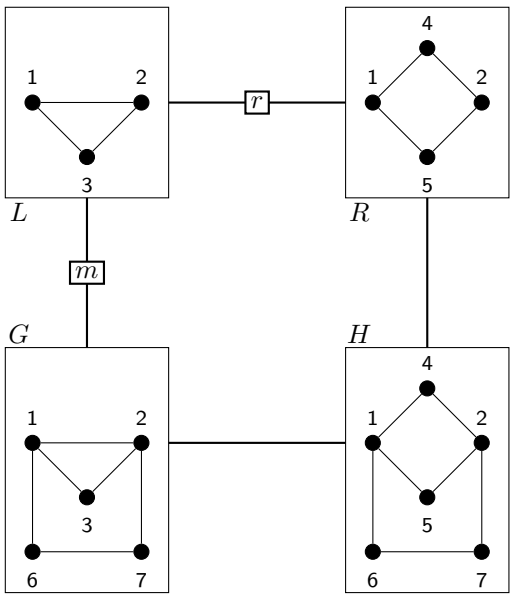


Figura 1: Exemplo de uma transformação de grafo (adaptado de [Zambon, 2013]).

O método de transformação de grafos utilizado no GROOVE é conhecido como *transformação algébrica* [Rozenberg, 1997]. Como este trabalho não tem foco no estudo do processo de

transformação de grafos em si, os detalhes de como uma transformação é realizada não são discutidos aqui. Para a pesquisa deste artigo, é suficiente assumir que existe um processo de transformação de grafos que gera os novos estados durante a exploração. Esse processo será denotado aqui pela função $transformar$.

2.3. Sistema de Transição de Grafos

O conjunto de todos os estados gerados pelas operações de transformação de grafos é denominado $\mathcal{S}$ e o conjunto de todas as transições aplicadas sobre os estados é chamado de $\rightarrow$. Esses dois conjuntos formam um *Sistema de Transição de Grafos* (STG).

3. Exploração do Espaço de Estados

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.

Atualmente a exploração no GROOVE é realizada de forma exaustiva, isto é, todos os estados possíveis são explorados a partir da gramática. Contudo, existem modelos de sistemas de determinados domínios que geram um espaço de estados muito grande ou até mesmo infinitos, inviabilizando o uso da exploração exaustiva. Diante desse cenário, em oposição a exploração exaustiva, é proposto neste trabalho aplicação de métodos de busca guiada, capaz de explorar parcialmente o espaço de estados.

3.1. Exploração Exaustiva

O algoritmo de exploração exaustiva do espaço de estados do GROOVE constrói um STG a partir de uma gramática de grafos para exploração exaustiva, e é apresentado no Algoritmo 1. Neste algoritmo, $\mathcal{N}$ representa o conjunto de estados *novos*, que ainda serão explorados.

Definição 3.1 (Gramática de Grafos para Exploração Exaustiva) *Uma* gramática de grafos para exploração exaustiva $\mathcal{G} = \langle \mathcal{R}, G_0 \rangle$ *é uma tupla composta por um conjunto de regras de transformação de grafos* $\mathcal{R}$ *e um grafo inicial* G_0.

Algoritmo 1: Exploração Exaustiva do Espaço de Estados

Entrada: gramática $\mathcal{G} = \langle \mathcal{R}, G_0 \rangle$
Saída: $STG = \langle \mathcal{S}, \rightarrow \rangle$

```
1  S := {G_0}, → := ∅, N := {G_0};
2  enquanto N ≠ ∅ faça
3      escolha G ∈ N ;          // qual G é escolhido depende da estratégia de busca
4      N := N \ {G};
5      para r ∈ R, m ∈ match(r, G) faça
6          H := transformar(r, m, G);
7          se novo(H, S) então                                   // se H ∉ S
8              S := S ∪ {H};
9              N := N ∪ {H};
10         fim
11         → := → ∪ {G -r-> H};
12     fim
13 fim
```

Na linha 1, o conjunto de estados gerados e o conjunto dos estados que ainda serão explorados são inicializados com o grafo inicial da gramática $\mathcal{G}$. O conjunto de transições é inicializado como vazio.

O critério de escolha e inserção de um grafo em $\mathcal{N}$ são destacados nas linhas 3 e 9. Este trabalho utiliza o critério de *busca em largura (breadth-first search)*, que gera de forma sistemática todos os estados. A fim de garantir que os estados serão explorados uma única vez, o conjunto $\mathcal{N}$

é implementado como uma fila, onde um grafo G, que será transformado, é retirado do início do conjunto $\mathcal{N}$ e o grafo transformado H é inserido no final.

A linha 5 lida com a correspondência entre as regras de transformação e o estado que está sendo explorado. A função $match(r, G)$ computa todos os casamentos de subgrafos possíveis entre o grafo L de r e o grafo G. A função $transformar(r, m, G)$ na linha 6 aplica a regra de transformação e gera o novo grafo H.

Na linha 7, o procedimento $novo(H, \mathcal{S})$ é responsável por verificar se o grafo H já existe no conjunto $\mathcal{S}$ de todos os estados explorados, evitando que um estado duplicado seja inserido. Finalmente, na linha 11, a transição executada na linha 6 é inserida no conjunto de transições $\rightarrow$.

3.2. Busca Guiada

A busca guiada é uma forma de caminhamento pelo espaço de estados que usa alguma métrica de distância do estado atual para o estado objetivo, com intuito de decidir qual o próximo estado será explorado.

Definição 3.2 (Gramática de Grafos para Busca Guiada) *Uma* gramática de grafos para busca guiada $\mathcal{G} = \langle \mathcal{R}, G_0, G_f \rangle$ *é uma tupla composta por um conjunto de regras de transformação de grafos* $\mathcal{R}$*, um grafo inicial* G_0 *e um grafo objetivo* G_f*.*

A exploração usando a busca guiada, que constrói um STG a partir de uma gramática de grafos para busca guiada pode ser traduzida no Algoritmo 2, onde $\mathcal{N}$ representa o conjunto de estados *novos*, que ainda serão explorados. O conjunto $\mathcal{T}$ armazena temporiamente os estados transformados, ordenados de acordo com o cálculo da métrica.

Algoritmo 2: Busca Guiada no Espaço de Estados

Entrada: gramática $\mathcal{G} = \langle \mathcal{R}, G_0, G_f \rangle$
Saída: $STG = \langle \mathcal{S}, \rightarrow \rangle$

```
1  S := {G_0}, → := ∅, N := {G_0}, T := ∅, encontrado := falso;
2  enquanto (N ≠ ∅) e (¬ encontrado) faça
3      escolha G ∈ N ;          // qual G é escolhido depende da implementação de N
4      N := N \ {G};
5      para r ∈ R, m ∈ match(r, G) faça
6          H := transformar(r, m, G);
7          se novo(H, S) então                                   // se H ∉ S
8              S := S ∪ {H};
9              T := T ∪ {H};
10             se H = G_f então
11                 encontrado := verdadeiro;
12             fim
13         fim
14         → := → ∪ {G -r-> H};
15     fim
16     N := N ∪ T;
17     T := ∅
18 fim
```

A linha 1 do Algoritmo 2 inicializa o sistema de transição e o conjunto de estados a serem explorados. Além disso o conjunto $\mathcal{T}$ é inicializado como vazio e a variável que indica quando o estado objetivo foi encontrado, recebe o valor falso.

Na linha 2 destaca-se a variável $encontrado$ como a condição de parada do algoritmo. O critério de escolha em $\mathcal{N}$ depende da estratégia de busca guiada implementada. As linhas 4, 5, 6, 7,

8 e 14 são descritas da mesma forma que no Algoritmo 1, e por isso não serão discutidas novamente aqui.

A linha 9 trata da inserção de cada grafo transformado H no conjunto $\mathcal{T}$. Para cada grafo H é computado um valor com base em uma métrica em relação ao grafo objetivo G_f e esse critério é utilizado para ordenar os estados em $\mathcal{T}$. A linha 10 verifica se o grafo transformado H é igual ao grafo objetivo G_f, e em caso afirmativo, o valor *verdadeiro* é atribuído a $encontrado$, indicando que o estado objetivo foi encontrado.

Caso o grafo objetivo ainda não tenha sido encontrado, todos os elementos do conjunto temporário $\mathcal{T}$ são incluídos no conjunto de estados a serem explorados $\mathcal{N}$ (linha 16). Finalmente na linha 17, o conjunto $\mathcal{T}$ é esvaziado.

3.2.1. Caminhamento Aleatório

O caminhamento aleatório é capaz de explorar parcialmente o espaço de estados, através de uma busca "às cegas" (em contraste com uma busca guiada que utiliza uma métrica de distância). Uma vez que o caminhamento aleatório não calcula nenhuma informação adicional sobre os estados, tudo que o algoritmo pode fazer é gerar sucessores e distinguir um estado objetivo de um estado não-objetivo.

Apesar de não haver nenhum critério para guiá-lo, esse caminhamento pode ser implementado com base no Algoritmo 2. Os conjuntos $\mathcal{T}$ e $\mathcal{N}$ são implementados como listas lineares. A escolha de G em $\mathcal{N}$ é feita de forma arbitrária e com a mesma probabilidade, através da geração de um número aleatório entre 0 e o tamanho de $\mathcal{N}$ que determina o índice do estado que será removido da lista para exploração. Não há prioridade para inserção de um grafo transformado em $\mathcal{T}$.

O caminhamento aleatório foi implementado neste trabalho para ser utilizado como uma base de comparação com a busca guiada. A hipótese considerada aqui é de que um simples caminhamento aleatório pelo espaço de estados não é uma boa forma para se buscar o grafo objetivo, uma vez que não há nenhuma garantia de que o caminho aleatório que é construído é o mais eficiente. Além disso, devido à sua característica inerente, o caminhamento aleatório pode gerar resultados drasticamente diferentes em execuções distintas. Essa hipótese foi confirmada pelos resultados experimentais apresentados na seção 6.1.

3.2.2. Caminhamento Guloso

Neste trabalho, o caminhamento guloso é o método utilizado na busca guiada. A ideia básica é computar todos os casamentos de subgrafos possíveis de uma regra de transformação em um grafo, calcular o valor da métrica em cada um deles e selecionar o que possui menor valor de métrica, e a partir daí expandir os próximos estados, até encontrar ao estado objetivo.

Na Figura 2 é apresentado um exemplo. Considerando o Algoritmo 2, o conjunto $\mathcal{N}$ é implementado como uma pilha, e $\mathcal{T}$ como uma fila de prioridades. Inicialmente o sistema de transição está vazio e o grafo inicial da gramática $\mathcal{G}$ inicializa o conjunto de estados a serem explorados. O estado **s0** é retirado da pilha e todos os casamentos de subgrafos possíveis são computados. Para cada casamento, ocorre a transformação baseada na regra **r**, gerando um novo estado. A seguir, calcula-se o valor da distância do novo estado para o estado objetivo, denotado por **m**, e o novo estado é inserido em $\mathcal{T}$, usando-se como prioridade as distâncias calculadas. No exemplo, nenhum dos novos estados **s1**, **s2** e **s3** são iguais ao grafo objetivo, então eles são removidos do início de $\mathcal{T}$ e inseridos em $\mathcal{N}$. Dessa forma, o primeiro elemento de $\mathcal{N}$ é o que possui o melhor valor de distância. Esse processo é realizado com todos os estados até que o estado objetivo seja encontrado.

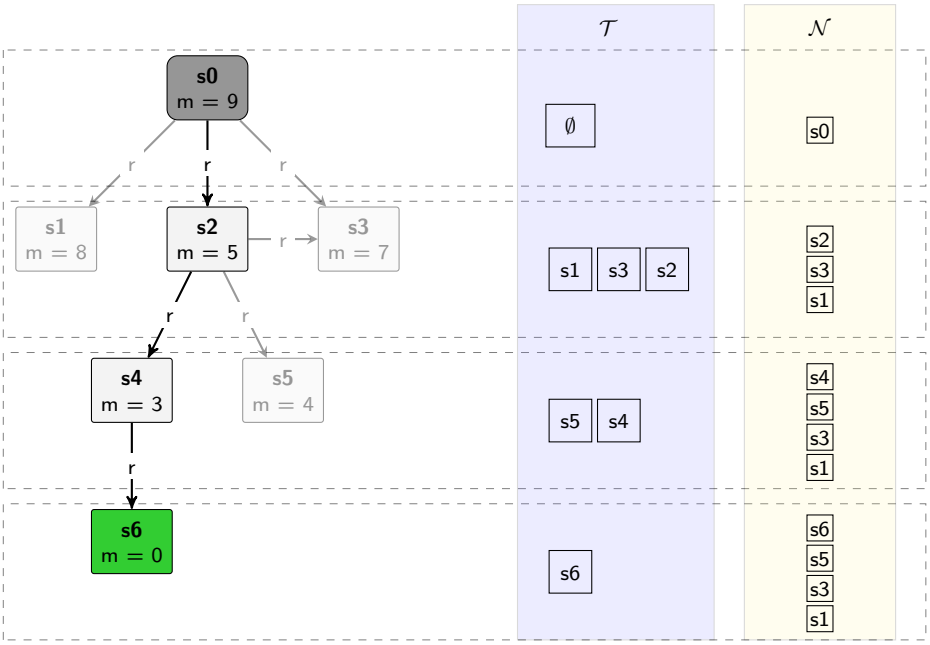


Figura 2: Busca Guiada com Caminhamento Guloso.

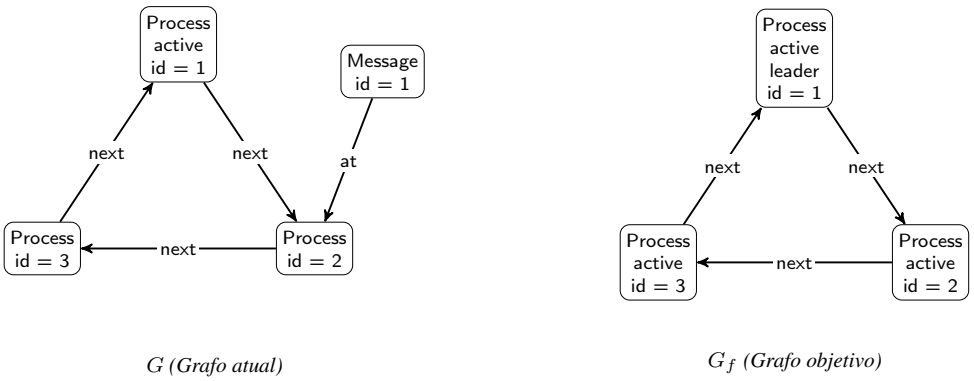
4. Métricas de Distância

A distância entre dois grafos G_1 e G_2 é representada por $d(G_1, G_2)$ e mede a dissemelhança entre um conjunto de características de cada um deles. Quanto maior for a distância, menor será a semelhança entre os grafos.

A métrica de distância deve satisfazer, no mínimo, as seguintes propriedades:

- $d(G_1, G_2) >= 0$ (não-negativa);
- $d(G_1, G_1) = 0$; e
- $d(G_1, G_2) = d(G_2, G_1)$ (simetria).

Com o objetivo de exemplificar o cálculo das métricas que serão apresentadas a seguir, foram selecionados dois grafos, gerados na exploração da gramática *leader election*, apresentada na seção 5. O grafo G é um estado intermediário e G_f é o estado objetivo. Os rótulos contidos nos nós do grafo são arcos rotulados do tipo laço, e foram representados visualmente dessa forma para facilitar o entendimento do exemplo.



G (Grafo atual) *G_f (Grafo objetivo)*

Figura 3: Grafos da gramática *leader election*

4.1. Contagem Simples

Na contagem simples, o conceito da distância entre dois grafos baseia-se unicamente na diferença entre o número de nós e arcos *não-laço* de cada grafo. Considerando o exemplo da Figura 3 temos:

	G	G_f
nº de nós	4	3
nº de arcos não-laço	4	3

O valor total da métrica é dado pela soma do módulo da diferença entre nós e arcos não-laço. Assim, no exemplo acima $d(G, G_f) = 2$.

4.2. Contagem de Rótulos

A contagem de rótulos tem um conceito parecido com a contagem simples, porém ao invés de contar nós e arcos não-laço, são contados os rótulos destes (incluindo arcos laço).

	G	G_f
Process	3	3
Message	1	0
active	1	3
leader	0	1
next	3	3
at	1	0
id = 1	2	1
id = 2	1	1
id = 3	1	1

O valor total da métrica é dado pela soma do módulo da diferença entre os rótulos dos grafos. Assim, no exemplo acima $d(G, G_f) = 6$.

5. Base de Dados Experimentais

Para a realização dos testes de exploração exaustiva, caminhamento aleatório e busca guiada, foram selecionadas quatro gramáticas que modelam problemas de domínios distintos:

- **leader election**: modela um protocolo de comunicação que visa, através de trocas de mensagens entre nós de uma rede, eleger um líder pela comparação de seus números identificadores. As especificações são encontradas em [Ghamarian e Zambon, 2009].
- **car platoon**: modela parte de um protocolo de comunicação sem fio entre um pelotão de carros em rodovias, que trocam mensagens para coordenar as ações dentro do pelotão, como por exemplo, a junção de um novo carro ao pelotão. Este modelo é apresentado com detalhes na seção 5.1.
- **philosophers**: problema clássico do jantar dos filósofos. Esse modelo exemplifica problemas e soluções encontrados na programação concorrente. Detalhes precisos do modelo são encontrados em [Varró, 2004].
- **append**: modela a inserção de um elemento no fim de uma estrutura de dados fila de tamanho 5, com um determinado número de invocações de inserção concorrentes. Quanto maior o número de invocações, maior é o espaço de estados da gramática. Mais detalhes podem ser vistos em [Rensink et al., 2004].

Na seção seguinte, as regras da gramática *car platoon* são apresentadas. Na apresentação visual de um regra utilizada neste trabalho (que é tomada a partir GROOVE), os grafos R e L são combinados em um único grafo constituído de quatro tipos de elementos, mostrados na Figura 4.

Os ***readers*** representam um padrão que deve estar presente no grafo hospedeiro, para que a regra seja aplicável. Os ***erasers*** representam os nós e arcos que serão removidos. Os ***creators*** identificam os nós e arcos que serão adicionados e os ***embargoes*** representam subpadrões que devem estar ausentes no grafo hospedeiro, para que a regra possa ser aplicada.

(a) Readers (b) Erasers (c) Creators (d) Embargoes

Figura 4: Elementos visuais de uma regra

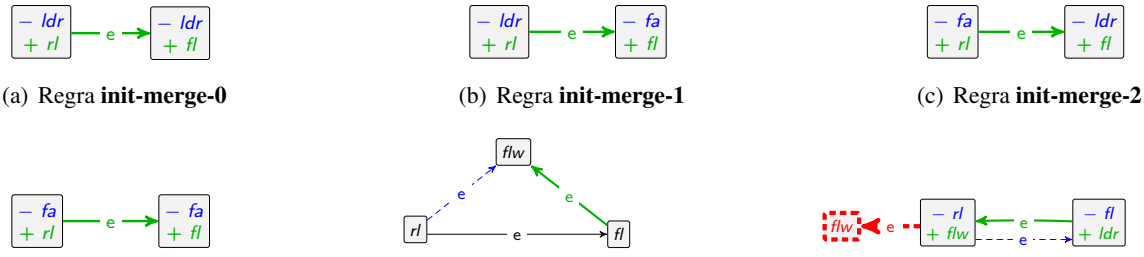
5.1. Gramática *car platoon*

Esta gramática foi adaptada de [Bauer, 2006] por [Zambon, 2013], e seu modelo criado no GROOVE é apresentado na Figura 5. Os carros são representados por nós, que podem assumir diferentes estados dentro do pelotão. Esses estados são representados através de um rótulo do nó. Um carro que não está no pelotão tem o estado de *free agent* (rótulo *fa*).

Quando um pelotão é formado, o primeiro carro está rotulado como *leader* (rótulo *ldr*), e os demais carros do pelotão como *followers* (rótulo *flw*). A comunicação entre os carros no pelotão é representado pelos arcos com rótulo *e*. Esta gramática modela o processo de junção de dois pelotões.

Dois carros rotulados como *leader* de pelotões diferentes se aproximam para iniciar a junção. Tomemos como exemplo a regra mostrada na Figura 5(a). O *leader* de um pelotão muda o estado para tornar-se um *front leader* (rótulo *fl*) enquanto o outro para tornar-se um *rear leader* (rótulo *rl*). A junção também pode ocorrer entre um *leader* e um *free agent* ou dois carros *free agent*. Essas situações são modeladas nas regras de junção na Figura 5(b-d).

Uma vez que os carros *front leader* e *rear leader* são definidos, o *rear leader* transfere seus carros *followers*, um a um, para o *front leader*. A transferência de um seguidor é modelada pela regra da Figura 5(e). Quando o *rear leader* não possui mais *followers*, muda seu estado para *follower* e junta-se ao pelotão liderado pelo *front leader*, que muda seu estado para *leader*. Esse processo é modelado pela regra da Figura 5(f), terminando a junção dos pelotões.



(a) Regra **init-merge-0** (b) Regra **init-merge-1** (c) Regra **init-merge-2**

(d) Regra **init-merge-3** (e) Regra **pass-1** (f) Regra **ldr2flw**

Figura 5: Gramática *car platoon*

6. Resultados Computacionais

Os resultados dos testes são mostrados nas Tabelas 1 e 2. Foram executados testes usando três formas de caminhamento pelo espaço de estados: caminhamento exaustivo usando BFS, caminhamento aleatório e guloso. O caminhamento guloso utilizou duas métricas: contagem simples e contagem de rótulos.

O computador utilizado possui processador Intel Core i5-3570, 8GB de RAM e Sistema Operacional Ubuntu 14.04 (x64). A implementação foi feita em Java (mesma linguagem utilizada pelo GROOVE), com o JDK 1.8.0_77 . A quantidade de estados gerados pelo caminhamento em BFS é utilizada como parâmetro para medir a eficiência das buscas.

6.1. Resultados Caminhamento Aleatório

Inicialmente, um algoritmo de caminhamento aleatório foi testado, pois sua implementação é simples e dispensa o cálculo de métricas para guiá-lo. Como o algoritmo lida com aleatoriedade, foram realizadas 1000 execuções para cada instância de cada gramática e calculado o desvio padrão para medir o quanto de variação existe em relação à média dos estados gerados. Os resultados podem ser observados na Tabela 1.

A primeira coluna da tabela lista as instâncias de quatro gramáticas utilizadas para teste. Elas estão organizadas em grupos ordenadas de forma crescente de tamanho. A segunda coluna apresenta a quantidade de estados gerados pelo caminhamento exaustivo (BFS). A terceira coluna apresenta a quantidade média de estados gerados pela busca aleatória, arrendondados para o inteiro mais próximo. O valor expresso em percentagem na quarta coluna mede de forma quantitativa a exploração do espaço de estados, dado pelo número total de estados médios gerados pelo caminhamento aleatório, em relação ao total de estados gerados pelo caminhamento exaustivo. A última coluna apresenta o cálculo do desvio padrão.

Ao observar os valores percentuais, nota-se que nas instâncias das gramáticas *leader election* e *philosophers* cerca de 77% do espaço de estados foi explorado. Já na gramática *append*, o percentual ficou em torno de 86% e a gramática *car platoon* apresentou o maior percentual de exploração, atingindo quase 100%. Esses valores indicam que para encontrar o estado objetivo, o caminhamento aleatório precisou explorar quase que totalmente o espaço de estados, degenerando praticamente para um caminhamento exaustivo.

Também é possível observar que, a medida que as instâncias crescem de tamanho, o desvio padrão aumenta, indicando uma grande variação na quantidade de estados gerados em cada execução em relação a média. Considerando a variação para mais e adicionando à média, obtém-se um valor de estados gerados próximos ou iguais aos gerados pela busca em largura.

Diante desses resultados, concluí-se que essa não é uma boa forma de caminhamento pelo espaço de estados, uma vez que o objetivo é explorar gramáticas que geram espaço de estados com grandes dimensões.

Tabela 1: Resultados do Caminhamento Aleatório

Gramática	BFS	Caminhamento Aleatório		
	Ger	Ger(Média)	%BFS	DesvPad
leader-election-3	32	28	87,63%	3,3
leader-election-4	144	115	79,98%	17,5
leader-election-5	803	622	77,48%	99,7
leader-election-6	5.383	4.205	78,12%	603,7
car-platoon-3	6	6	100%	0,0
car-platoon-4	13	12	94,59%	1,0
car-platoon-5	24	23	94,95%	1,5
car-platoon-6	48	45	94,43%	3,2
car-platoon-7	86	82	95,83%	4,4
car-platoon-8	160	155	96,85%	6,7
car-platoon-9	282	275	97,65%	8,4
car-platoon-10	500	492	98,41%	12,0
philosophers-3	27	21	77,32%	4,5
philosophers-4	86	63	72,93%	16,6
philosophers-5	275	210	76,31%	53,7
philosophers-6	982	731	74,42%	207,2
philosophers-7	3.499	2.596	74,20%	790,0
append-2	145	129	88,77%	13,0
append-3	3.185	2.714	85,22%	292,1

6.2. Resultados do Caminhamento Guloso

Como os resultados do caminhamento aleatório não foram satisfatórios, a segunda estratégia adotada foi caminhamento guloso combinado com as métricas de distância discutidas na seção 4. Os resultados podem ser observados na Tabela 2.

A primeira coluna da tabela lista as instâncias das quatro gramáticas utilizadas para teste. Elas estão organizadas em grupos ordenados de forma crescente de tamanho. A segunda coluna apresenta a quantidade de estados gerados pelo caminhamento exaustivo (BFS). Para cada métrica é apresentado o número de estados gerados e o percentual em relação ao BFS. Não são mostrados os tempos de execução dos testes pois o objetivo do experimento é caracterizar a eficiência (número de estados gerados) frente ao caminhamento BFS. De qualquer forma, a performance computacional de cada algoritmo pode ser inferida, pois é diretamente proporcional ao número de estados explorados.

Nas instâncias *leader-election-9* e *leader-election-10* não foi possível realizar os testes do BFS, pois a gramática apresenta crescimento exponencial e a memória disponível não foi suficiente para armazenar os estados gerados. No entanto, ao utilizar a busca gulosa, o número de estados gerados é pequeno, permitindo que o objetivo seja encontrado.

As gramáticas *leader election* e *append* não apresentam diferença no número de estados gerados para as duas métricas. Nesse caso, é preferível o uso da primeira métrica, que é mais simples de calcular, e portanto tem custo computacional menor. A gramática *philosophers* apresenta melhores resultados com a métrica de contagem de rótulos e a métrica de contagem simples apresenta bons resultados para a gramática *car-platoon*. Assim, não é possível afirmar que uma métrica é melhor do que a outra, sendo necessário uma análise mais profunda das características de cada gramática testada e avaliação de outras gramáticas, afim de obter respostas precisas sobre esse comportamento.

Tabela 2: Resultados da Busca Guiada

Gramáticas	BFS	Caminhamento Guloso			
		Contagem Simples		Contagem de Rótulos	
	Gen	Gen	% BFS	Gen	% BFS
leader-election-3	32	17	53,13%	17	53,13%
leader-election-4	144	31	21,53%	31	21,53%
leader-election-5	803	52	6,48%	52	6,48%
leader-election-6	5.383	82	1,52%	82	1,52%
leader-election-7	42.040	123	0,29%	123	0,29%
leader-election-8	373.282	177	0,05%	177	0,05%
leader-election-9	-	246	-	246	-
leader-election-10	-	332	-	332	-
philosophers-3	27	26	96,3%	13	48,15%
philosophers-4	86	67	77,91%	22	25,58%
philosophers-5	275	126	45,82%	32	11,64%
philosophers-6	982	374	38,09%	45	4,58%
philosophers-7	3.499	629	17,98%	59	1,69%
philosophers-8	13.006	1.073	8,25%	76	0,58%
philosophers-9	48.819	2.290	4,69%	94	0,19%
philosophers-10	186.194	1.773	0,95%	115	0,06%
car-platoon-3	6	6	100%	6	100%
car-platoon-4	13	9	69,23%	11	84,62%
car-platoon-5	24	15	62,5%	18	75%
car-platoon-6	48	19	39,58%	25	52,08%
car-platoon-7	86	25	29,07%	34	39,53%
car-platoon-8	160	29	18,13%	44	27,5%
car-platoon-9	282	35	12,41%	48	17,02%
car-platoon-10	500	39	7,8%	68	13,6%
append-2	145	30	20,69%	30	20,69%
append-3	3.185	149	4,68%	149	4,68%
append-4	103.225	2.639	2,56%	2.639	2,56%

Mesmo com métricas simples, foram atingidas reduções bastante significativas ao caminhar pelo espaço de estados, em comparação com o caminhamento exaustivo. Todas as gramáticas de teste apresentam redução de quase 95% no percentual de estados gerados nas maiores instâncias utilizadas. Outro resultado interessante é que a medida que as instâncias crescem, há uma tendência de redução no percentual de estados explorados. Essa redução indica que embora o tamanho do espaço de estados das gramáticas cresce de forma exponencial em relação ao tamanho das instâncias, a busca guiada é capaz de encontrar o estado objetivo explorando um número de estados que cresce apenas de forma linear.

Assim, conclui-se que a busca guiada é capaz de reduzir consideravelmente o número de estados visitados (explorados) ao se caminhar pelo espaço de estados, mostrando ser uma boa alternativa para solução de importantes classes de sistemas que geram um espaço de estados muito grande no GROOVE.

7. Conclusão e Trabalhos Futuros

Este artigo mostra as vantagens da aplicação de métodos de busca guiada no caminhamento do espaço de estados de gramáticas de grafos. Para isso é feita uma comparação entre o caminhamento exaustivo BFS, o caminhamento aleatório e os métodos de busca guiada.

Nos testes realizados, a aplicação de métodos de busca guiada, permitiu reduzir a exploração do espaço de estados em até 95% para as maiores instâncias testadas. É possível perceber que as reduções são maiores a medida que as instâncias crescem de tamanho.

Fica como trabalho futuro o estudo de outras métricas de distância e a implementação de novos algoritmos de busca, além da realização de experimentos com outras gramáticas de teste.

Referências

Baier, C., Katoen, J.-P., et al. (2008). *Principles of Model Checking*. MIT press Cambridge.

Bauer, J. (2006). *Analysis of communication topologies by partner abstraction*. Tese de doutorado, Universität des Saarlandes.

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*, p. 15–40.

Ghamarian, A. H. e Zambon, E. (2009). Verifying the leader election algorithm in GROOVE. *5th International Workshop on Graph-Based Tools*.

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

Rensink, A., Schmidt, Á., e Varró, D. (2004). Model checking graph transformations: A comparison of two approaches. p. 226–241.

Rozenberg, G. (1997). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World scientific.

Varró, D. (2004). Automated formal verification of visual modeling languages by model checking. *Software and Systems Modeling*, p. 85–113.

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