

PROBLEMA DE ALCANÇABILIDADE EM
GRAFOS MUITO GRANDES: APLICAÇÃO,
COMPLEXIDADE E ALGORITMOS

RODRIGO FERREIRA DA SILVA

**PROBLEMA DE ALCANÇABILIDADE EM
GRAFOS MUITO GRANDES: APLICAÇÃO,
COMPLEXIDADE E ALGORITMOS**

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais - Departamento de Ciência da Computação. como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

ORIENTADOR: SEBASTIÁN ALBERTO URRUTIA

Belo Horizonte
Fevereiro de 2019

© 2019, Rodrigo Ferreira da Silva
Todos os direitos reservados

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Silva, Rodrigo Ferreira da.

S586p Problema de alcançabilidade em grafos muito grandes: aplicação, complexidade e algoritmos / Rodrigo Ferreira da Silva. — Belo Horizonte, 2019. xxi, 81p.: il.; 29 cm.

Tese (doutorado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Sebastián Alberto Urrutia

1. Computação – Teses. 2. Teoria de grafos. 3. Alcançabilidade em grafos. 4. Ordenação topológica. I. Orientador. II Título.

CDU 519.6*62(043)



UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO


FOLHA DE APROVAÇÃO


Problema de Alcançabilidade em Grafos Muito Grandes: Aplicação,
Complexidade e Algoritmos

RODRIGO FERREIRA DA SILVA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROF. SEBASTIÁN ALBERTO URRUTIA - Orientador
Departamento de Ciência da Computação - UFMG


PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG


PROF. LOIC PASCAL GILLES CERF
Departamento Ciência da Computação - UFMG


PROF. VINÍCIUS FERNANDES DOS SANTOS
Departamento de Ciência da Computação - UFMG


PROF. FLÁVIO KEIDI MIYAZAWA
Instituto de Computação - UNICAMP


PROF. UÉVERTON DOS SANTOS SOUZA
Instituto de Computação - UFF

Belo Horizonte, 15 de fevereiro de 2019.

Agradecimentos

Primeiramente, gostaria de agradecer a minha esposa Denise e minha filha Gabriela, por terem me incentivado durante todo o meu curso. Agradeço aos meus pais, Valquir e Fátima, que abdicaram de inúmeras coisas e momentos para me permitir chegar até aqui.

Ao iniciar um curso de Doutorado, me senti inseguro já que deveria necessariamente produzir uma inovação científica para conseguir defender a minha tese. É uma responsabilidade que realmente incomoda já que muitas vezes o caminho que trilhamos em uma pesquisa pode não ter resultados positivos. O papel do orientador é justamente tentar guiar o aluno para caminhar na trilha correta. Por isso, tenho que agradecer ao Professor Sebastián Urrutia por ter conduzido a minha orientação de forma brilhante desde o primeiro até o último dia do curso. Agradeço aos Professores Vinícius Fernandes e Raquel Prates por também terem me apoiado durante esse caminho.

O Professor Sebastián também me concedeu a oportunidade de fazer um período sanduíche de três meses no *Molde University College* na Noruega, sob supervisão do Professor Lars Magnus Hvattum. Agradeço ao Professor Lars por ter me ajudado a avançar na pesquisa durante esse período e também por ter permitido que eu colaborasse em outros projetos junto a outros renomados pesquisadores.

O desafio de vencer a primeira fase da qualificação me juntou a outros alunos que fazem pesquisa em outras áreas da Computação. Provavelmente, se não fosse a temida qualificação, eu não teria conhecido o Filipe, Júlio, Adriano, Paulo, Bruno, Heber e Alex. Não posso deixar de mencionar, ainda, os amigos do LaPo, que também me ajudaram nessa caminhada.

Agradeço aos funcionários da Secretaria do DCC, em especial à Sônia, Linda e Sheila, que estiveram sempre de prontidão para me ajudar em várias situações. Agradeço de forma mais ampla a todos os membros do DCC por manterem um curso com nível máximo de excelência. Por fim, agradeço ao DCC e CAPES pela minha bolsa de pesquisa durante o curso.

*“Nós somos o que repetidamente fazemos.
Excelência, então, não é um ato, mas um hábito.”*
(Aristóteles)

Resumo

Dados um grafo direcionado acíclico $G = (V, E)$ e dois vértices quaisquer $u, v \in V$, o problema de alcançabilidade consiste em responder se a partir de u é possível alcançar v percorrendo as arestas do grafo. Para grafos muito grandes, com milhões de vértices, é impraticável realizar uma busca no grafo a cada consulta ou armazenar o fecho transitivo completo já que o espaço necessário é da ordem de $O(|V|^2)$. Abordagens intermediárias geram índices para efetuar cortes negativos e positivos durante a execução das consultas. Neste trabalho, formalizamos o Problema de Dominância Fraca Unilateral que é utilizado informalmente por abordagens da literatura para geração de índices de boa qualidade, e provamos que o problema é NP-Completo. Identificamos algumas oportunidades para melhoria nas principais abordagens para o problema. Abordagens atuais podem utilizar apenas um número pequeno de ordenações topológicas no seu índice. Propomos uma nova abordagem escalável, chamada LYNX, que pode utilizar um número grande ordenações topológicas do grafo como índice de corte negativo, sem degradar o desempenho para cada ordenação topológica incluída. Uma estratégia similar é aplicada ao índice de corte positivo. Além disso, o LYNX propõe que o tamanho do índice possa ser configurado pelo usuário e também permite um controle sobre a proporção entre o índice de corte negativo e positivo dependendo do padrão de consulta esperado. Mostramos através de experimentos computacionais que o LYNX supera a abordagem que é considerada o estado-da-arte em termos de tempo de consulta usando o mesmo tamanho de índice com grafos de alto índice de alcançabilidade.

Palavras-chave: Teoria de Grafos, Alcançabilidade em Grafos, Grafos Muito Grandes, Ordenação Topológica.

Abstract

Given a directed acyclic graph $G = (V, E)$ and two vertices $u, v \in V$, the reachability problem is to answer if it is possible to reach v from u by transversing the edges of the graph. For very large graphs, with millions of vertices, it is impracticable to search the graph for each query and it is also impracticable to store the transitive closure of the graph since it is $O(|V|^2)$ in terms of space. Then, scalable approaches aim to generate good indexes used to prune the search during its execution in the graph. In this work, we formalize the One-Sided Weak Dominance Drawing Problem which is already used informally in the literature to construct this kind of indexes and we also prove that this problem is NP-Complete. Next, we identify opportunities for improvement on main approaches for the problem. Recent approaches to this problem can use only a small number of topological sorts on their indexes. We propose a novel scalable approach called LYNX that uses a large number of topological sorts of the graph as a negative cut index without degrading the query time. A similar strategy is applied regarding the positive cut index. In addition, LYNX proposes a user-defined index size that enables the user to control the ratio between negative and positive cut depending on the expected query pattern. We show by computational experiments that LYNX outperforms the state-of-the-art approach in terms of query-time using the same index-size for graphs with high reachability ratio.

Keywords: Graphs Theory, Reachability in Graphs, Very Large Graphs, Topological Sorting.

Lista de Figuras

1.1	Exemplo de grafo direcionado acíclico (DAG).	2
1.2	Arestas exploradas em uma consulta de alcançabilidade através do DFS.	2
1.3	Arestas exploradas em uma consulta de alcançabilidade através do BFS.	3
1.4	Relação de compromisso entre o tempo de consulta e tamanho do índice [Yildirim et al., 2010], com a complexidade do tempo de consulta modificada de $O(V + E)$ para $O(E)$	3
2.1	Exemplo de DAG e intervalos gerados para o índice de corte negativo pelo GRAIL.	13
2.2	Exemplo de DAG e filtro de nível gerado pelo GRAIL.	14
2.3	Exemplo de DAG e intervalos para o índice de corte positivo gerados pelo GRAIL.	15
2.4	Exemplo de DAG G (esquerda) e permutação aleatória $\pi = (7, 11, 8, 6, 3, 0, 2, 1, 10, 4, 9, 5)$ (direita)	22
2.5	Exemplo de DAG G (esquerda) e atribuição de um número do conjunto $\{1, 2, 3, 4, 5\}$ a cada vértice segundo uma função hash (direita)	23
3.1	Grafo direcionado acíclico apresentado como exemplo em [Yildirim et al., 2010].	27
3.2	DAG e intervalos gerados para o índice de corte negativo pelo GRAIL.	29
4.1	Exemplo de DAG e respectivo Desenho de Dominância.	31
4.2	Exemplo de execução do algoritmo descrito no Lema 1.	38
4.3	Instância do OSCM-4-Star.	40
4.4	Instância OSCM-4-Star da Figure 4.3 com os centros divididos.	42
4.5	Grafo G gerado a partir de uma instância do OSCM-4-Star com 3 estrelas.	42
5.1	Exemplo de caso patológico da heurística <i>Maximum-Rank</i>	50
5.2	Grafo coroa (<i>crown graph</i>) com seis vértices.	53

5.3	Grafo de ordenações topológicas do grafo coroa apresentado na Figura 5.2.	53
5.4	Aplicação da estratégia de pulos.	55
5.5	Aplicação da estratégia de pulos no grafo de ordenações topológicas do grafo coroa.	57
5.6	Tempo de consulta do BFL em função do valor do parâmetro s	64
5.7	Tempo de processamento (s) do LYNX à medida em que são inseridas novas ordenações topológicas	65
5.8	Tamanho do índice (MB) do LYNX à medida em que são inseridas novas ordenações topológicas	66
5.9	Tempo de consulta (ms) do LYNX à medida em que são inseridas novas ordenações topológicas	66
5.10	Comparação entre FELINE, BFL e LYNX em relação ao tempo de consulta com o crescimento do número de arcos e R-ratio.	72

Lista de Tabelas

2.1	Rótulos IP para o DAG G da Figura 2.4 com $k = 5$	22
2.2	Rótulos BFL para o DAG G da Figura 2.5.	23
5.1	Comparação do número de falsos positivos gerados pelas abordagens exatas para o WDD e OSWDD, e a heurística <i>MaximumRank</i> utilizada pela FELINE.	48
5.2	Comparação do tempo de execução das abordagens exatas para o WDD e OSWDD	49
5.3	Instâncias de grafos reais para <i>benchmark</i>	62
5.4	Instâncias de grafos sintéticos para <i>benchmark</i>	63
5.5	Cortes negativos com a utilização da estratégia de pulos	64
5.6	Tempo de consulta (ms) do LYNX à medida em que são inseridas novas ordenações topológicas	67
5.7	Tempo de execução (ms) para 1 milhão de consultas aleatórias em instâncias de grafos reais	68
5.8	Tempo de execução (ms) para 1 milhão de consultas balanceadas, 500 mil positivas e 500 mil negativas, em instâncias de grafos reais	68
5.9	Tempo de pré-processamento (s) em instâncias de grafos reais	69
5.10	Utilização de memória (MB) em instâncias de grafos reais	69
5.11	Tempo de pré-processamento (s) para instâncias de grafos sintéticos	70
5.12	Utilização de memória (MB) para instâncias de grafos sintéticos	71
5.13	Tempo de execução (ms) para 1 milhão de consultas aleatórias para instâncias de grafos sintéticos	71

Sumário

Agradecimentos	vii
Resumo	xi
Abstract	xiii
Lista de Figuras	xv
Lista de Tabelas	xvii
1 Introdução	1
1.1 Motivação	1
1.2 Principais Contribuições deste Trabalho	4
1.3 Organização do Restante da Proposta de Tese	5
2 Abordagens Escaláveis para Alcançabilidade em Grafos	7
2.1 Definições Preliminares	7
2.2 GRAIL	11
2.2.1 Índice de Corte Negativo	12
2.2.2 Filtro de Nível	14
2.2.3 Índice de Corte Positivo	15
2.2.4 Consulta de Alcançabilidade	16
2.3 FERRARI	16
2.3.1 Índice de Corte Positivo	16
2.3.2 Poda Baseada em Sementes	17
2.3.3 Filtro de Nível e Topológico	17
2.4 FELINE	18
2.4.1 Índice de Corte Negativo	19
2.5 HD-GDD	20

2.6	IP	21
2.7	BFL	23
3	Equivalência entre Conjunto de Intervalos e Duas Ordenações Topológicas	25
3.1	Equivalência entre Representações	26
3.2	GRAIL com Ordenações Topológicas	27
3.3	FELINE com Intervalos	28
3.4	Comparação entre Índices da FELINE e GRAIL	28
4	Desenho de Dominância Fraca Unilateral	31
4.1	Introdução	31
4.2	Desenho de Dominância Fraca	32
4.2.1	Formulação Matemática para o WDD	33
4.3	FELINE e Desenho de Dominância Fraca	34
4.4	Desenho de Dominância Fraca Unilateral	35
4.4.1	Formulação Matemática para o OSWDD	35
4.5	Complexidade do OSWDD	36
4.5.1	Lema Preliminar	36
4.5.2	OSWDD é NP-Completo	39
4.6	Simplificação da Prova de NP-Completude do KNN_{TP}	43
4.7	Um limite inferior dependente da dimensão para o OSWDD	44
5	Nova Abordagem para Consultas de Alcançabilidade em Grafos Grandes	47
5.1	Motivação	47
5.1.1	Comparação entre WDD, OSWDD e <i>MaximumRank</i>	48
5.1.2	Caso Patológico da Heurística <i>Maximum-Rank</i>	50
5.1.3	Eficiência do Índice do HD-GDD	50
5.1.4	Oportunidades para Melhoria do IP e BFL	51
5.2	LYNX	51
5.2.1	Grafo de Ordenações Topológicas	52
5.2.2	Estratégia de Pulos	54
5.2.3	Índice de Corte Negativo	55
5.2.4	Índice de Corte Positivo	58
5.2.5	Consulta de Alcançabilidade	60
5.2.6	Otimizações Adicionais	60
5.3	Resultados Computacionais	61

5.3.1	Ambiente Computacional	61
5.3.2	Instâncias de <i>Benchmark</i>	61
5.3.3	Configuração de Parâmetros	62
5.3.4	Eficiência da Estratégia de Pulos	64
5.3.5	Eficiência do LYNX com a Inclusão de Ordenações Topológicas	65
5.3.6	Resultados para Grafos Reais	67
5.3.7	Resultados para Grafos Sintéticos	72
6	Conclusões e Trabalhos Futuros	75
6.1	Trabalhos Futuros	76
6.1.1	Combinar BFL com LYNX	76
6.1.2	Instâncias Reais com Maior Número de Vértices e Arcos	76
6.1.3	Solução Escalável Combinada com SCARAB	76
6.1.4	Algoritmo Aproximativo para o OSWDD	77
6.1.5	Aplicação de Metaheurísticas para o WDD e OSWDD	77
	Referências Bibliográficas	79

Capítulo 1

Introdução

Neste trabalho, tratamos do problema de alcançabilidade em grafos muito grandes. É um problema clássico, bem resolvido para grafos de pequeno e médio porte através de algoritmos de busca, como busca em profundidade ou busca em largura, em tempo linear em relação ao tamanho do grafo.

Na próxima seção, apresentamos a motivação para este trabalho e os desafios ao se tratar deste problema para grafos com milhões de vértices e dezenas de milhões de arestas.

1.1 Motivação

Seja $G = (V, E)$ um grafo direcionado acíclico (ou DAG, do inglês *Directed Acyclic Graph*), onde V é o conjunto de vértices e E o conjunto de arestas. Definimos uma consulta de alcançabilidade entre dois vértices $u, v \in V$ como responder se é possível alcançar v a partir de u percorrendo as arestas do grafo. Caso v seja alcançável a partir de u dizemos que $u \rightsquigarrow v$ e, caso u não alcance v , dizemos que $u \not\rightsquigarrow v$.

A Figura 1.1 apresenta um exemplo de um DAG. Neste grafo, podemos observar que $1 \rightsquigarrow 8$, que é um exemplo de consulta de alcançabilidade com resposta positiva. Porém, $1 \not\rightsquigarrow 6$, o que é um exemplo de consulta com resposta negativa.

Os algoritmos clássicos de busca em profundidade (ou DFS, do inglês *Depth-First Search*) [Tarjan, 1972] ou busca em largura (ou BFS, do inglês *Breadth-First Search*) podem ser utilizados para responder uma consulta de alcançabilidade em um DAG. No exemplo da Figura 1.2, utilizando o DFS para a consulta de alcançabilidade entre os vértices 1 e 8, são exploradas as arestas $(1, 3)$, $(1, 4)$, $(4, 7)$ e $(4, 8)$, apresentadas em vermelho.

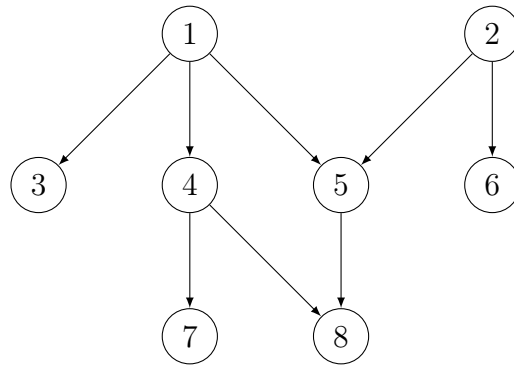


Figura 1.1. Exemplo de grafo direcionado acíclico (DAG).

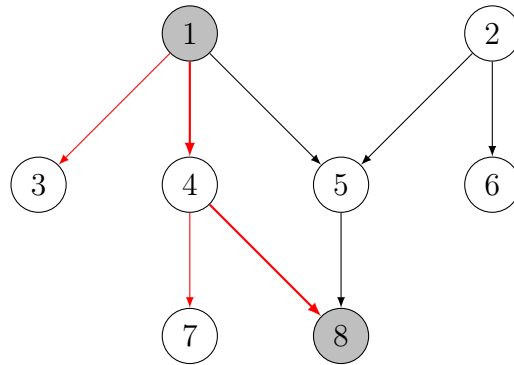


Figura 1.2. Arestas exploradas em uma consulta de alcançabilidade através do DFS.

Se o BFS for utilizado, para a mesma consulta entre os vértices 1 e 8, serão exploradas primeiramente arestas $(1, 3)$, $(1, 4)$, $(1, 5)$, e em seguida as arestas $(4, 7)$ e $(4, 8)$, apresentadas em vermelho na Figura 1.3. Em termos de esforço computacional utilizado para realizar a pesquisa, para este caso o BFS explora uma aresta a mais que o DFS para responder a mesma consulta. No entanto, para uma consulta de alcançabilidade entre os vértices 1 e 5, o DFS explora 5 arestas, enquanto o BFS explora apenas 3 arestas.

No pior caso, os algoritmos de busca DFS e BFS percorrem todas as arestas do DAG, com complexidade $\mathcal{O}(|E|)$. É importante ressaltar que esses algoritmos possuem complexidade $\mathcal{O}(|V| + |E|)$ na literatura [Cormen et al., 2009], porém, quando utilizados para uma consulta de alcançabilidade, não precisam percorrer todos os vértices do grafo, se limitam aos vértices alcançáveis através do vértice origem.

No contexto deste trabalho, são analisadas consultas em grafos muito grandes, com milhões de vértices e dezenas de milhões de arestas, e ainda milhões de consultas

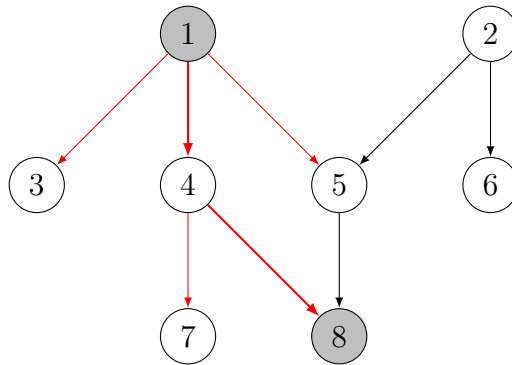


Figura 1.3. Arestas exploradas em uma consulta de alcançabilidade através do BFS.

a serem realizadas, o que torna inviável na prática correr o risco de percorrer todo o grafo a cada consulta. Então, outra abordagem que surge como alternativa é computar o fecho transitivo completo do DAG e armazená-lo em memória primária ou secundária para consulta em tempo constante.

Para grafos pequenos, a estratégia de armazenar o fecho transitivo é promissora. Porém, se tomarmos como exemplo um grafo com 100 milhões de vértices, o fecho transitivo teria que armazenar 10^{16} entradas. Considerando que cada entrada consome pelo menos 1 bit, o espaço mínimo necessário seria de mais de 1 petabyte (1.136 terabytes) de dados, o que pode ser inviável considerando ainda o tempo de processamento para sua geração, que tem complexidade de $\mathcal{O}(|V||E|)$ executando, por exemplo, uma busca DFS ou BFS a partir de cada vértice.

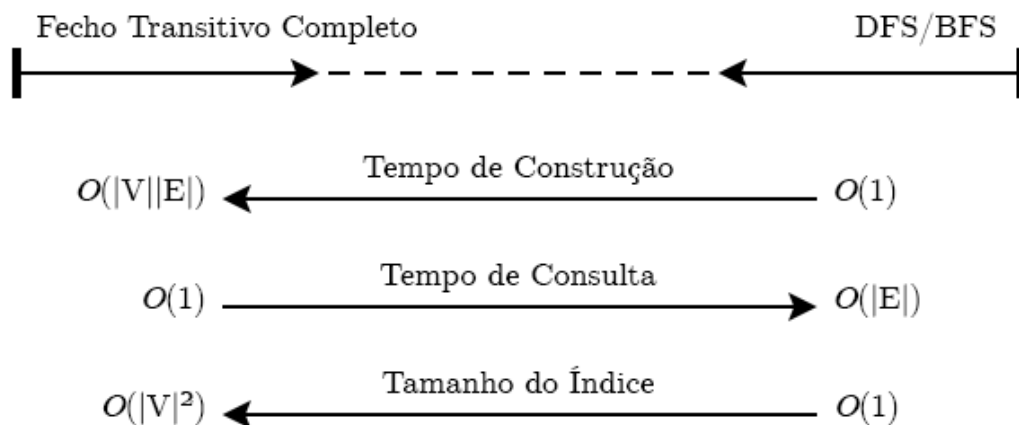


Figura 1.4. Relação de compromisso entre o tempo de consulta e tamanho do índice [Yildirim et al., 2010], com a complexidade do tempo de consulta modificada de $\mathcal{O}(V + E)$ para $\mathcal{O}(E)$

Conforme foi mostrado, as duas estratégias são impraticáveis para grafos muito grandes, seja pelo tempo gasto para cada consulta no pior caso (estratégia de busca no grafo a cada solicitação) ou pelo tempo de processamento e uso de espaço quadrático para gerar e armazenar o fecho transitivo completo. Com isso, as propostas escaláveis para resolver consultas de alcançabilidade estão entre essas duas abordagens extremas, apresentadas na Figura 1.4. Essas propostas intermediárias incluem a geração de índices que possibilitam cortes diversos durante a busca no grafo. As principais abordagens escaláveis para o problema presentes na literatura são apresentadas no Capítulo 2.

1.2 Principais Contribuições deste Trabalho

- Na maioria das abordagens utilizadas por algoritmos para consultas de alcançabilidade em grafos grandes, os índices são representados por intervalos simples. A FELINE [Velo et al., 2014] propõe uma abordagem que utiliza duas ordenações topológicas do DAG para representar o índice que realiza um corte negativo durante as consultas. No entanto, iremos mostrar que as representações por intervalos ou por ordenações topológicas têm a mesma expressividade, isto é, dada uma representação por um conjunto de intervalos, é possível traduzí-la em duas ordenações topológicas e também o contrário, dadas duas ordenações topológicas de um grafo é possível representá-las através de um conjunto de intervalos. Com isso, é possível traduzir as representações dos índices de várias abordagens (e.g. [Yildirim et al., 2012; Anand et al., 2013; Velo et al., 2014; Li et al., 2017]) em uma única representação.
- A heurística utilizada pela FELINE e proposta em [Kornaropoulos, 2012] se propõe a resolver o problema de Desenho de Dominância Fraca (WDD) [Kornaropoulos & Tollis, 2012b] (*Weak Dominance Drawing*) para a construção de um índice composto de duas ordenações topológicas. O WDD consiste em encontrar duas ordenações topológicas de um DAG que maximizem o número de inversões entre os pares de vértices, produzindo um índice de corte negativo eficiente para responder consultas. No entanto, a heurística de fato resolve outro problema mais simples que é criado quando uma das coordenadas (ou ordenações topológicas) é fixada. Para este último problema, que é o problema de fato resolvido pelo algoritmo proposto em [Kornaropoulos, 2012] e utilizado pela FELINE, nada se sabe sobre a sua complexidade. Neste trabalho, formalizamos a descrição deste problema e apresentamos uma prova formal de NP-Completeness. Além disso, como subproduto desta prova, simplificamos a prova de NP-Completeness do *nea-*

rest neighbor Kendall tau para uma ordenação parcial e uma ordenação total, o que fortalece os resultados obtidos em [Brandenburg et al., 2012].

- São expostas evidências importantes de que existe um grande espaço para melhorias nas abordagens atuais para o problema de alcançabilidade em grafos muito grandes. Mostramos um caso patológico da heurística usada pela FELINE e limitações das abordagens HD-GDD [Li et al., 2017], IP [Wei et al., 2018] e BFL [Su et al., 2017].
- Propomos uma nova abordagem para o problema, chamada de LYNX, que introduz diversas melhorias sobre as propostas atuais para o problema. O LYNX segue a representação da FELINE que utiliza ordenações topológicas no seu índice de corte negativo. No entanto, ao contrário do HD-GDD, o desempenho das consultas não se degrada ao incluir uma nova ordenação topológica no índice.
- Apresentamos experimentos computacionais com instâncias de grafos reais e sintéticos que mostram que o LYNX supera as abordagens atuais para o problema para grafos com alto índice de alcançabilidade.

1.3 Organização do Restante da Proposta de Tese

Os próximos capítulos deste trabalho estão organizados da maneira que se segue.

No Capítulo 2 é feita uma revisão bibliográfica sobre as principais abordagens escaláveis para o problema de alcançabilidade em grafos muito grandes. São abordados o GRAIL [Yildirim et al., 2010], que foi a primeira abordagem escalável para o problema descrita na literatura, e as principais abordagens subsequentes, FERRARI [Anand et al., 2013], FELINE [Velooso et al., 2014], HD-GDD [Li et al., 2017], IP [Wei et al., 2018] e BFL [Su et al., 2017].

As principais abordagens na literatura para o problema de alcançabilidade representam seus índices por um conjunto de intervalos ou duas ordenações topológicas. No Capítulo 3, mostramos que essas duas representações são equivalentes e que é possível transformar uma representação na outra.

No Capítulo 4, formalizamos o Problema de Dominância Fraca Unilateral que é utilizado informalmente na literatura para a geração de índice de corte negativo. Concluimos também sobre a NP-Completeness da sua versão de decisão e, para isso, provamos um lema sobre permutações gerais que pode ser útil em outros contextos.

Em seguida, no Capítulo 5, propomos uma nova abordagem para o problema de alcançabilidade em grafos muito grandes, chamada LYNX, e mostramos o seu desempenho na solução do problema em grafos reais e sintéticos.

Por fim, no Capítulo 6, delineamos as conclusões sobre este trabalho e enumeramos as possíveis linhas para a extensão da pesquisa apresentada nesta Tese.

Capítulo 2

Abordagens Escaláveis para Alcançabilidade em Grafos

Neste capítulo, apresentamos uma revisão bibliográfica sobre as principais abordagens escaláveis na literatura para o problema de alcançabilidade em grafos muito grandes.

Começamos, na Seção 2.1, introduzindo algumas notações e definições que serão úteis para o restante deste trabalho. Nas seções posteriores, apresentamos em sequência o GRAIL [Yildirim et al., 2010, 2012], FERRARI [Anand et al., 2013], FELINE [Velo et al., 2014], HD-GDD [Li et al., 2017], IP [Wei et al., 2018] e BFL [Su et al., 2017], e discutimos sobre as diferenças entre essas abordagens.

2.1 Definições Preliminares

Definição 1 (Grafo não-direcionado). *Um grafo não-direcionado $G = (V, E)$ é definido por um conjunto finito V de vértices interconectados dois a dois por arestas, representadas pelo conjunto $E = \{\{u, v\} \in E \mid u, v \in V\}$. Neste trabalho consideramos apenas grafos não-direcionados simples, isto é, que não possuem laços (arestas que começam e terminam no mesmo vértice) e não possuem arestas paralelas.*

Definição 2 (Grafo direcionado). *Um grafo direcionado $G = (V, A)$ é definido por um conjunto finito V de vértices e um conjunto de pares ordenados E que representam arcos (arestas direcionadas) $\{(u, v) \in A \mid u, v \in V\}$ que conectam certos pares de vértices. Neste trabalho consideramos apenas grafos direcionados simples, isto é, que não possuem laços (arcos que começam e terminam no mesmo vértice) e não possuem arcos paralelos.*

Definição 3 (Grafo direcionado acíclico). *Um grafo direcionado acíclico (ou DAG, do inglês Directed Acyclic Graph) é um grafo direcionado tal que não existe caminho não vazio que parte e termina em um mesmo vértice.*

Definição 4 (Grafo direcionado transposto). *O transposto de um grafo direcionado $G = (V, A)$ é representado por $G^T = (V, A^T)$ tal que $A^T = \{(u, v) \mid (v, u) \in A\}$. Isto é, possui o mesmo conjunto de vértices com todos os arcos invertidos em comparação aos arcos de G .*

Definição 5 (Sucessores e Predecessores). *O conjunto de sucessores de um vértice u é representado por $Suc(u) = \{v \in V \mid (u, v) \in A\}$ e o conjunto de predecessores é representado por $Pre(u) = \{v \in V \mid (v, u) \in A\}$*

Definição 6 (Grau de entrada e saída). *Em um grafo direcionado, o grau de entrada de um vértice é definido por $indeg(u) = |Suc(u)|$ e o grau de saída de um vértice é definido por $outdeg(u) = |Pre(u)|$.*

Definição 7 (Fontes e Sumidouros). *Seja um grafo direcionado $G = (V, E)$, uma fonte é um vértice $f \in V$ se $indeg(f) = 0$. Um vértice s é um sumidouro quando $outdeg(s) = 0$.*

Definição 8 (Distância em um grafo). *Seja um grafo $G = (V, E)$ e dois vértices $u, v \in V$. A distância entre v e u , ou $dist(u, v)$, é definida como o comprimento (número de arestas) do menor caminho entre u e v em G .*

Definição 9 (Excentricidade). *Em um grafo $G = (V, E)$, a excentricidade $ecc(v)$ de um vértice $v \in V$ é dada pela máxima distância entre o vértice v e outro vértice qualquer de G , ou seja, $ecc(v) = \max_{u \in V} (dist(v, u))$. Para um grafo desconectado, todos os vértices possuem excentricidade infinita. A excentricidade máxima é chamada de largura ou diâmetro ($diam(G)$) do grafo.*

Definição 10 (Ordenação topológica). *Uma ordenação topológica t de um DAG $G = (V, E)$ é uma permutação de V tal que para cada aresta $(u, v) \in E$, $t(u) < t(v)$, onde $t(u)$ representa a posição de u na ordenação topológica t .*

Definição 11 (Busca em Largura ou BFS, do inglês *Breadth-First Search*). *Dado um grafo $G = (V, E)$ e um vértice $s \in V$, a busca em largura explora sistematicamente as arestas de G até descobrir cada vértice alcançável por s , segundo o Algoritmo 1. O algoritmo calcula e armazena no vetor d todas as menores distâncias em número de arestas entre o vértice s e os vértices acessíveis a partir dele. Por sua vez, o vetor π armazena o predecessor de cada vértice e , a partir desse vetor, é possível construir*

uma árvore representada por $G_\pi = (V_\pi, E_\pi)$, onde $V_\pi = \{v \in V \mid \pi[v] \neq NIL\} \cup \{s\}$ e $E_\pi = \{(\pi[v], v) \mid v \in V_\pi - \{s\}\}$.

Algoritmo 1: Busca em largura (BFS) a partir de s [Cormen et al., 2009]

```

1   $BFS(G, s)$ 
2  for cada vértice  $u \in V(G) - \{s\}$  do
3       $cor[u] \leftarrow \text{BRANCO}$ 
4       $d[u] \leftarrow \infty$ 
5   $cor[s] \leftarrow \text{CINZA}$ 
6   $d[s] \leftarrow 0$ 
7   $Q \leftarrow \text{FilaVazia}()$ 
8  Adicionar( $Q, s$ )
9  while  $Q \neq \emptyset$  do
10      $u \leftarrow \text{Remove}(Q)$ 
11     for cada  $v \in \text{Suc}(u)$  do
12         if  $cor[v] = \text{BRANCO}$  then
13              $cor[v] \leftarrow \text{CINZA}$ 
14              $d[v] \leftarrow d[u] + 1$ 
15              $\pi[v] \leftarrow u$ 
16             Adicionar( $Q, v$ )
17      $cor[u] \leftarrow \text{PRETO}$ 

```

Definição 12 (Busca em Profundidade ou DFS, do inglês *Depth-First Search*). Dado um grafo $G = (V, E)$, na busca em profundidade as arestas são exploradas a partir do vértice v mais recentemente descoberto, segundo o Algoritmo 2. O subgrafo predecessor gerado a partir de π é representado por $G_\pi = (V, E_\pi)$, onde $E_\pi = \{(\pi[v], v) \mid v \in V \wedge \pi[v] \neq NIL\}$, e forma uma floresta primeiro na profundidade composta por várias árvores primeiro na profundidade. As arestas em E_π são chamadas arestas de árvore.

Definição 13 (Consulta de alcançabilidade). Seja o DAG $G = (V, E)$ e $u, v \in V$, uma consulta de alcançabilidade, representada por $u \stackrel{?}{\rightsquigarrow} v$, tem por objetivo responder se é possível alcançar v a partir de u percorrendo as arestas do grafo. Representamos por $u \not\rightsquigarrow v$ quando a partir de u não é possível alcançar v e por $u \rightsquigarrow v$, caso contrário, isto é, quando é possível alcançar v a partir de u .

Definição 14 (Vértices alcançáveis e inalcançáveis). Utilizamos a notação de alcançabilidade para representar todos os vértices que podem alcançar um vértice u do grafo como $In(u) = \{v \in V \mid v \rightsquigarrow u\}$ e o conjunto de todos os vértices alcançados por u como $Out(u) = \{v \in V \mid u \rightsquigarrow v\}$.

Algoritmo 2: Busca em profundidade (DFS) [Cormen et al., 2009]

```

1 DFS(G)
2   for cada vértice  $u \in V(G) - \{s\}$  do
3      $cor[u] \leftarrow \text{BRANCO}$ 
4      $\pi[u] \leftarrow \text{NIL}$ 
5      $tempo \leftarrow 0$ 
6     for cada vértice  $u \in V(G)$  do
7       if  $cor[u] = \text{BRANCO}$  then
8         DFS-VISIT( $u$ )
9 DFS-VISIT( $u$ )
10   $cor[u] \leftarrow \text{CINZA}$ 
11   $tempo \leftarrow tempo + 1$ 
12  for cada vértice  $v \in \text{Suc}(u)$  do
13     $\pi[v] \leftarrow u$ 
14    DFS-VISIT( $v$ )
15   $cor[u] \leftarrow \text{PRETO}$ 
16   $f[u] \leftarrow tempo$ 
17   $tempo ++$ 

```

Definição 15 (Índice de alcançabilidade (R-ratio)). *O índice de alcançabilidade (R-ratio) de um DAG $G = (V, E)$ é a porcentagem de consultas positivas entre consultas envolvendo todos os possíveis pares de vértices $(u, v) \in V \times V$. Como é impraticável computá-lo para grafos muito grandes, é feita uma estimativa utilizando um número grande de consultas aleatórias e computando o número de consultas positivas entre elas.*

Definição 16 (Fecho transitivo). *O fecho transitivo de um grafo $G = (V, E)$ é uma relação binária transitiva representada por $tr(G) = \{(u, v) \mid u, v \in V \wedge u \rightsquigarrow v\}$.*

Definição 17 (Índice de corte negativo). *Seja um DAG $G = (V, E)$, um índice de corte negativo $I_N(G)$ é definido como um subconjunto de todos os pares de vértices $(u, v) \in V \times V$ tal que u não alcança v , ou seja $I_N(G) \subseteq \{(u, v) \mid \forall u, v \in V, u \not\rightsquigarrow v\}$. O uso de um índice negativo durante uma pesquisa de alcançabilidade permite então diminuir o número de vértices explorados durante a busca no grafo, desconsiderando vértices que, de acordo com o índice, certamente não possuem caminho até o vértice destino.*

Definição 18 (Índice de corte positivo). *De forma semelhante ao índice de corte negativo, podemos definir um índice de corte positivo $I_P(G)$ como sendo um subconjunto de todos os pares de vértices $(u, v) \in V \times V$ tal que u alcança v , ou seja $I_P(G) \subseteq$*

$\{(u, v) \mid \forall u, v \in V, u \rightsquigarrow v\}$. Ou seja, $I_P(G)$ é um subconjunto do fecho transitivo de G . Dessa forma, o índice de corte positivo permite terminar uma busca em um vértice intermediário a partir do qual certamente se alcança o vértice destino.

Definição 19 (Conjunto parcialmente ordenado, ou poset, ou ordenação parcial, do inglês *partially ordered set*). Um poset é um par $\mathcal{P} = (X, P)$ onde X é um conjunto e P é uma relação binária reflexiva, anti-simétrica e transitiva em X . Chamamos X de conjunto base enquanto P é uma ordenação parcial em X .

Definição 20 (Pares comparáveis e incomparáveis). Seja $\mathcal{P} = (X, P)$ um poset e $x, y \in X$ com $x \neq y$. Dizemos que x e y são comparáveis quando $x < y$ ou $y < x$ em P . Caso contrário, x e y são incomparáveis.

Definição 21 (Cadeias e anti-cadeias). Um poset $\mathcal{P} = (X, P)$ é chamado de cadeia (ordenação linear ou também ordenação total) quando todos os pares distintos de X são comparáveis em P . Uma anti-cadeia se define quando todos os pares distintos de X são incomparáveis em P .

Definição 22 (Extensão linear). Sejam P e Q ordenações parciais sobre o mesmo conjunto X . Chamamos Q de extensão de P se $P \subseteq Q$. Se Q é uma ordenação linear de X , então Q é uma extensão linear de P .

Definição 23 (Dimensão de um poset). A dimensão de um poset $\mathcal{P} = (X, P)$, denotada por $\dim(X, P)$, é o menor número inteiro positivo t para o qual existe uma família $\mathcal{R} = \{L_1, L_2, \dots, L_t\}$ de extensões lineares de P tal que $P = \cap \mathcal{R} = \cap_{i=1}^t L_i$.

Definição 24 (Dimensão de um DAG). A dimensão de um DAG $G = (V, E)$, denotada por $\dim(G)$, é o menor número inteiro positivo s para o qual existe uma família $\mathcal{R} = \{t_1, t_2, \dots, t_s\}$ de ordenações topológicas de G tal que $tr(G) = \cap \mathcal{R} = \cap_{i=1}^s t_i$. Computar a dimensão de um DAG $G = (V, E)$ é o mesmo que computar a dimensão de um poset $\mathcal{P} = (X, P)$ onde $X = V$ e $P = tr(G)$.

2.2 GRAIL

GRAIL [Yildirim et al., 2010, 2012] é conhecido por ser a primeira abordagem escalável para o problema de alcançabilidade em grafos muito grandes. A cada consulta de alcançabilidade, o GRAIL dispara uma busca no grafo que tem como base o algoritmo de busca em profundidade (DFS). Durante essa busca, a exploração de certos caminhos

é eliminada através do uso de índices de corte positivo e negativo, o que acelera significativamente a resposta da consulta. A seguir, serão apresentados os índices utilizados pelo GRAIL.

2.2.1 Índice de Corte Negativo

GRAIL propõe um índice baseado em intervalos para cortes negativos com n conjuntos de intervalos, onde cada conjunto de intervalos é chamado de L^i , $1 \leq i \leq n$ e n é um parâmetro do algoritmo. Durante a construção do conjunto de intervalos L^i , a cada vértice $u \in V$ é associado um intervalo $L_u^i = [s_u^i, e_u^i]$, tal que se $L_v^i \not\subseteq L_u^i$, então $u \not\rightarrow v$. No entanto, caso $L_v^i \subseteq L_u^i$, ou seja, $s_u^i \leq s_v^i \leq e_v^i \leq e_u^i$, então nada pode ser afirmado sobre a alcançabilidade de v a partir de u .

O índice de corte negativo é gerado segundo o Algoritmo 3 que é executado para gerar cada conjunto de intervalos L^i . Em sua versão padrão, uma ordenação aleatória σ dos vértices é gerada a priori e, a cada passo do algoritmo, os vértices são visitados de acordo com essa ordenação, da esquerda para a direita. Em seguida, para aumentar a diferença entre os conjuntos de intervalos, é gerado um novo conjunto de intervalos utilizando o mesmo algoritmo, porém, visitando os vértices na ordem reversa de σ . A função $Suc(x)$ retorna todos os sucessores de x em G e a função $\min\{s_c^i : c \in Suc(x)\}$ retorna infinito caso o conjunto $Suc(x)$ esteja vazio.

Algoritmo 3: GRAIL: construção do índice de corte negativo

```

1 Labeling( $G, i, \sigma$ )
2    $r \leftarrow 1$  //variável global, ranking do vértice
3    $Roots \leftarrow \{n \in V(G) \mid indeg(n) = 0\}$ 
4   for  $x \in Raizes$ , na ordem que aparece em  $\sigma$  do
5      $\lfloor$  Visit( $x, i, G, \sigma$ )
6 Visit( $x, i, G, \sigma$ )
7   if  $x$  já foi visitado then
8      $\lfloor$  return
9   for  $y \in Suc(x)$ , na ordem que aparece em  $\sigma$  do
10     $\lfloor$  Visit( $y, i, G, \sigma$ )
11     $r_c^* \leftarrow \min\{s_c^i : c \in Suc(x)\}$ 
12     $L_x^i \leftarrow [\min(r, r_c^*), r]$ 
13     $r \leftarrow r + 1$ 

```

É possível observar que o algoritmo de geração do índice de corte negativo possui uma estrutura muito semelhante ao algoritmo de busca em profundidade (DFS),

alternando a ordem com que os vértices são visitados segundo a ordenação σ fornecida como parâmetro.

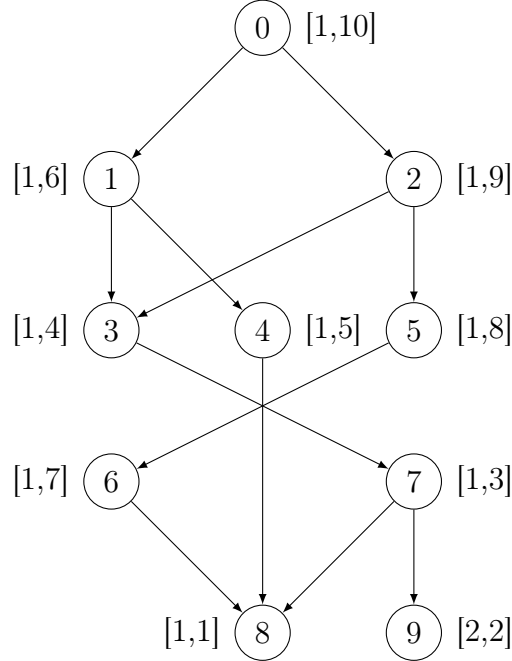


Figura 2.1. Exemplo de DAG e intervalos gerados para o índice de corte negativo pelo GRAIL.

A Figura 2.1 apresenta um exemplo de DAG e respectivo conjunto de intervalos gerado de acordo com o Algoritmo 3. Neste exemplo, o algoritmo percorre o grafo iniciando pelo do vértice 0 com $r = 1$. Recursivamente, o algoritmo visita os vértices 1, 3, 7 e 8. O vértice 8 é um sumidouro, logo $L_8^i = [1, 1]$ já que $r = 1$ e não possui nenhum descendente. Neste ponto, r é incrementado e o vértice 9 é visitado gerando o intervalo $L_9^i = [2, 2]$ e, após finalizar o vértice 9, r é incrementado novamente. Na sequência, os vértices 7 e 3 são visitados gerando os intervalos $L_7^i = [1, 3]$ e $L_3^i = [1, 4]$. O algoritmo retorna ao vértice 1 e visita o vértice 4. Neste ponto, $r = 5$, porém $r_c^* = 1$, já que $L_8^i = [1, 1]$ e 8 é descendente de 4. Então, $L_4^i = [1, 5]$ e r é incrementado. O algoritmo continua a sua execução dessa forma até finalizar a construção dos intervalos para os vértices restantes.

Note que o conjunto de intervalos gerado mantém a propriedade $L_v^i \not\subseteq L_u^i \rightarrow u \not\prec v$, para todo $u, v \in V$. Por exemplo, $L_9^i \not\subseteq L_8^i$, $L_4^i \not\subseteq L_3^i$ e $L_2^i \not\subseteq L_1^i$. Logo, pelo índice de corte negativo, $9 \not\prec 4$, $4 \not\prec 3$ e $2 \not\prec 1$, o que pode ser avaliado em tempo constante, sem percorrer o grafo. No entanto, $L_4^i \subseteq L_2^i$ é um falso-positivo do índice, já que $2 \not\prec 4$.

2.2.2 Filtro de Nível

Como se trata de um DAG, os vértices podem ser agrupados em níveis, de forma que os sumidouros estejam no primeiro nível, e os vértices restantes nos níveis superiores dos seus descendentes. O nível de um vértice u pode ser calculado como:

$$l_u = \begin{cases} 1 & \text{se } u \text{ é sumidouro} \\ 1 + \max_{v \in \text{Suc}(u)}(l_v) & \text{c.c.} \end{cases}$$

Dessa forma, se $l_u \leq l_v$, isto é, se v estiver em um nível igual ou superior a u , então $u \not\rightsquigarrow v$. Com isso, o filtro de níveis pode ser utilizado durante a busca no grafo para não explorar caminhos a partir de vértices que certamente não irão chegar ao destino. Note que, durante a busca no grafo em uma consulta de alcançabilidade, quando se passa de um vértice para o seu sucessor o nível sempre diminui, logo com o uso do índice a busca visita apenas vértices até o nível l_v .

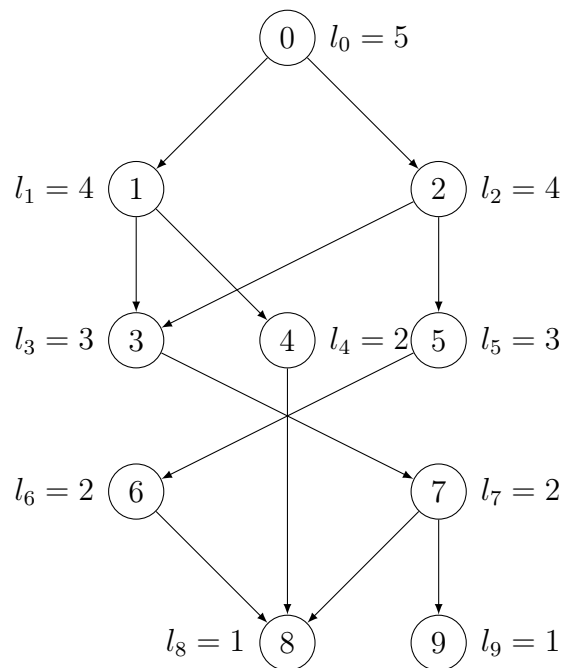


Figura 2.2. Exemplo de DAG e filtro de nível gerado pelo GRAIL.

A Figura 2.2 apresenta um exemplo de grafo e o nível calculado para cada vértice. Os sumidouros 8 e 9 possuem nível igual a 1. Na sequência, temos l_4 , l_6 e l_7 igual a 2. Em seguida, os vértices 3 e 5 recebem o nível igual a 3. O vértice 1 está ligado aos vértices 4 e 3, então, recebe nível igual a 4. O vértice 2 está conectada aos vértices 3 e 5, então também recebe nível igual a 4. Por fim, o vértice 0, a única fonte do grafo, recebe nível igual a 5.

Utilizando o grafo e os níveis computados na Figura 2.2, podemos concluir em tempo constante que $6 \not\rightsquigarrow 2$, já que $l_6 \leq l_2$. Ou ainda, que $7 \not\rightsquigarrow 6$ já que $l_7 = l_6$.

2.2.3 Índice de Corte Positivo

De forma semelhante ao índice de corte negativo, GRAIL define um índice de corte positivo $IP(G)$ que é um subconjunto do fecho transitivo de G .

GRAIL constrói o índice de corte positivo baseado nas arestas de árvore do grafo geradas durante a execução do algoritmo DFS. Durante o caminhar no grafo, são gerados intervalos com o tempo de início e finalização do processamento de cada vértice de forma que, para quaisquer dois vértices $u, v \in V$, se $IP_v \subseteq IP_u$, então $u \rightsquigarrow v$.

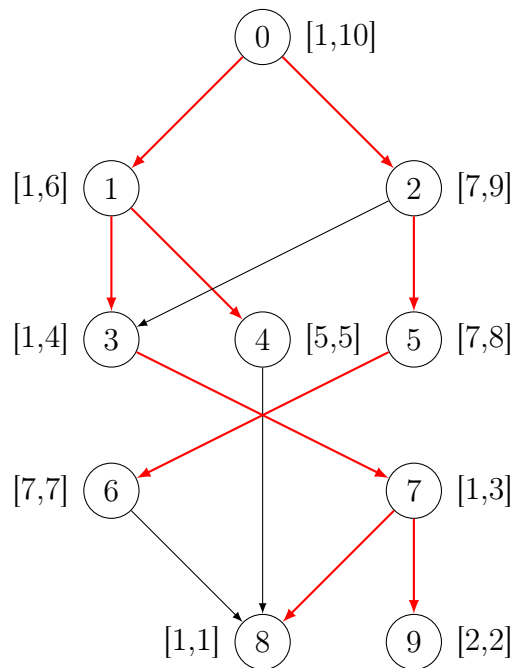


Figura 2.3. Exemplo de DAG e intervalos para o índice de corte positivo gerados pelo GRAIL.

A Figura 2.3 apresenta um exemplo de índice de corte positivo gerado para o DAG tomado como exemplo, na qual as arestas de árvore estão marcadas em vermelho. Utilizando o índice apresentado na Figura 2.3, é possível concluir em tempo constante e sem consultar o grafo que $1 \rightsquigarrow 9$ e $2 \rightsquigarrow 6$, já que $IP_9 \subseteq IP_1$ e $IP_6 \subseteq IP_2$.

2.2.4 Consulta de Alcançabilidade

O Algoritmo 4 apresenta a consulta de alcançabilidade do GRAIL com a utilização dos índices propostos. Na linha 2 é aplicado o corte positivo, isto é, com o uso do índice IP é possível estabelecer a priori para certos pares de vértices se u alcança v . O filtro de nível é utilizado na linha 4. Na linha 6, para cada conjunto de intervalos L^i , é aplicado o índice de corte negativo. Por fim, caso não seja possível podar a busca no vértice u , o DFS continua a partir dos seus descendentes.

Algoritmo 4: GRAIL: consulta de alcançabilidade

```

1 Reachable( $u, v, G$ )
2   if  $IP_v \subseteq IP_u$  then
3     return True //  $u \rightsquigarrow v$ 
4   if  $l_u \leq l_v$  then
5     return False //  $u \not\rightsquigarrow v$ 
6   for  $i \leftarrow 1$  até  $n$  do
7     if  $L_v^i \not\subseteq L_u^i$  then
8       return False //  $u \not\rightsquigarrow v$ 
9   for  $c \in Suc(u)$  do
10    if Reachable( $c, v, G$ ) then
11      return True
12  return False

```

2.3 FERRARI

FERRARI [Anand et al., 2013] é uma proposta que tem como objetivo melhorar o desempenho de consultas de alcançabilidade com o foco em um ponto fraco do GRAIL, o desempenho em consultas positivas. Para tanto, FERRARI propõe um índice principal de corte positivo, baseado em cobertura do DAG por árvores, juntamente com o índice de filtro de nível proposto pelo GRAIL.

2.3.1 Índice de Corte Positivo

Ao contrário do GRAIL, que tem o foco no índice de corte negativo, a FERRARI propõe um índice de corte positivo mais abrangente, baseado no algoritmo para cobertura de DAG por árvores proposto em [Agrawal et al., 1989]. Esse problema possui uma subestrutura “optimal” [Anand et al., 2013], o que permite a sua computação através de

um algoritmo de programação dinâmica que, no entanto, tem complexidade proibitiva para grafos muito grandes ($O(|V||E|)$).

Dessa forma, é proposta uma heurística mais simples que não permite uma aproximação garantida, no entanto, desempenha bem na prática, conforme apresentado nos resultados obtidos pela abordagem.

Seja um DAG $G = (V, E)$ e $\tau : V \rightarrow \{1, 2, \dots, n\}$ uma ordenação topológica de G . A posição de um vértice u em τ , dada por $\tau(u)$, representa um limite superior para o número de predecessores de u em G . Para cada vértice u com o conjunto de predecessores $Prec(u)$, é selecionada a aresta do vértice $p \in Prec(u)$ com a posição máxima na ordenação topológica para a inclusão na árvore.

Em seguida, dada uma cobertura por árvores T de G , são associados intervalos $I_T(v)$ a cada vértice de G através de um caminharmento DFS em T . Por fim, para garantir a restrição de no máximo k (provisão de espaço definida pelo usuário) intervalos por vértice, os intervalos são combinados através de um algoritmo guloso que é uma heurística para o algoritmo proposto em [Anand et al., 2013] e gera resultados sub-ótimos, porém, com bons resultados na prática.

2.3.2 Poda Baseada em Sementes

A eficiência do algoritmo de alcançabilidade depende diretamente do número de vértices que são expandidos durante a busca no grafo. Neste contexto, vértices com grau de saída muito alto trazem um custo maior para a busca já que o número de consultas recursivas é maior.

No primeiro passo de construção do índice é selecionado um subconjunto $S \subset V$ contendo os $|S|$ vértices com maior grau e, para cada vértice $v \in V$ são computados os conjuntos $In(v) = \{s \in S | s \rightsquigarrow v\}$ e $Out(v) = \{s \in S | v \rightsquigarrow s\}$.

Uma vez associados, os conjuntos são utilizados durante o processamento de consultas sendo que, para uma consulta $u \stackrel{?}{\rightsquigarrow} v$ em G , se $Out(u) \cap In(v) \neq \emptyset$, então $u \rightsquigarrow v$. Se existe uma semente s tal que $s \in In(u)$ e $s \notin In(v)$, isto é, a semente s alcança u mas não alcança v , a consulta pode ser terminada com uma resposta negativa.

2.3.3 Filtro de Nível e Topológico

Da mesma forma que o GRAIL, a FERRARI utiliza um filtro de nível que fornece um corte negativo durante o processamento da consulta de alcançabilidade. Além disso, também é utilizado um filtro de corte negativo topológico mantido através de uma

ordenação topológica τ de G de forma que para uma consulta $u \overset{?}{\rightsquigarrow} v$, se $\tau(v) < \tau(u)$, então $u \not\rightsquigarrow v$.

2.4 FELINE

FELINE [Velooso et al., 2014] é uma abordagem para alcançabilidade em grafos grandes que se baseia no GRAIL e tem como principal diferencial a representação e o processo de construção do índice de corte negativo.

O algoritmo para a construção do índice de corte negativo foi inspirado pelo Problema do Desenho de Dominância [Eades & Whitesides, 1994], ou *Dominance Drawing* (DD), que consiste em obter duas ordenações topológicas, t_X e t_Y , de um grafo G de dimensão igual a dois tais que, dados dois vértices quaisquer u e v , se u aparece antes de v nas duas ordenações topológicas, então u alcança v . Dessa forma, é possível desenhar os vértices do grafo em um plano, a partir das coordenadas $(t_X(v), t_Y(v))$ onde a posição de $X(v) = t_X(v)$ e a posição de $Y(v) = t_Y(v)$.

A partir desse desenho é possível identificar visualmente a alcançabilidade entre quaisquer dois vértices, dado que se u alcança v , então $X(u) < X(v)$ e $Y(u) < Y(v)$. Caso contrário, isto é, se $X(u) > X(v)$ ou $Y(u) > Y(v)$, é possível afirmar que u não alcança v . Com isso, se for necessário, por exemplo, estabelecer quais são todos os vértices alcançáveis a partir de um vértice v , basta observar apenas o quadrante superior à $X(v)$ e $Y(v)$.

A aplicação do Desenho de Dominância está fortemente relacionada ao conceito de dimensão do grafo ($dim(G)$) que é definida como o menor valor de k para o qual um Desenho de Dominância com a dimensão k pode ser obtido [Eades & Whitesides, 1994]. O problema de decisão equivalente foi provado ser NP-Completo em [Brightwell & Massow, 2013].

Uma direção natural para a extensão desse problema é tentar obter um Desenho de Dominância de um grafo G de dimensão qualquer no plano com a relaxação da restrição $X(u) < X(v) \wedge Y(u) < Y(v) \rightarrow u \rightsquigarrow v$, minimizando então o número de violações. Seguindo essa linha, foi proposto em [Kornaropoulos & Tollis, 2012b] o problema de Desenho de Dominância Fraca, ou *Weak Dominance Drawing* (WDD), que consiste em, dado um grafo direcionado acíclico $G = (V, E)$, determinar duas ordenações topológicas que minimizem o número de falsos positivos. Dadas duas ordenações topológicas $t_X = [x_1, x_2, \dots, x_{|V|}]$ e $t_Y = [y_1, y_2, \dots, y_{|V|}]$ de G , um falso-positivo é caracterizado para dois vértices $v, u \in V$, quando u não alcança v em G , e porém $t_X(u) < t_X(v)$ e $t_Y(u) < t_Y(v)$. Com isso, se observadas apenas as duas ordenações

topológicas, poderia-se pensar que u alcança v , o que não é verdade.

Outra forma de enxergar o problema é considerando o conjunto interseção entre t_X e t_Y , definido como $I(t_X, t_Y) = \{(u, v) | t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)\}$ ([Kornaropoulos & Tollis, 2012b]). Quanto menor o número de falsos-positivos, menor a interseção entre as duas ordenações topológicas.

A versão de decisão do problema consiste em, dado um grafo G e um número inteiro positivo k , estabelecer se existem duas ordenações topológicas tais que o número de falsos-positivos seja no máximo k . O WDD foi provado ser NP-Completo em [Kornaropoulos & Tollis, 2012b].

2.4.1 Índice de Corte Negativo

Para computar o seu índice, FELINE utiliza uma heurística para o WDD. Inicialmente é gerada uma ordenação topológica do grafo, t_X , e esta é fornecida como entrada para a geração de t_Y . O Algoritmo 5 é utilizado para a geração de t_Y , chamado de *Maximum-Rank* [Kornaropoulos & Tollis, 2012a] que, a cada iteração, escolhe o vértice fonte com o ranking ou posição máxima em t_X , e o remove do grafo.

Algoritmo 5: Maximum-Rank

Input: G, t_X
Output: t_Y

- 1 Inicializa S_G ;
- 2 $t_Y \leftarrow Nil$;
- 3 **for** $i \leftarrow 1$ **to** n **do**
- 4 $u \leftarrow maxRank_{t_X}(S_G)$;
- 5 $G \leftarrow G - u$;
- 6 Atualiza S_G ;
- 7 $t_Y[i] = u$;
- 8 **end**

O conjunto S_G é inicializado contendo todos os vértices fonte do grafo G . A ordenação topológica t_Y , na qual será armazenado o resultado do algoritmo, é inicialmente vazia. Em seguida, a cada iteração o algoritmo escolhe o vértice com o ranking máximo no conjunto S_G através da função $maxRank_{t_X}(S_G)$. Essa função retorna o vértice de $u \in S_G$ que está na maior posição na ordenação topológica X . O vértice selecionado é removido do grafo, juntamente com as suas arestas incidentes e também removido de S_G . O conjunto S_G é atualizado incluindo todos os vértices fonte do grafo G resultantes com a remoção do vértice u . O vértice u é adicionado ao final da ordenação topológica t_Y .

É importante ressaltar que a escolha do vértice $u \in S_G$ a ser inserido em t_Y realizada a cada iteração é localmente ótima entre todos os vértices de S_G pois a posição escolhida para o vértice em Y não gera falsos positivos entre os pares de S_G [Kornaropoulos & Tollis, 2012a], já que os vértices $S_G - \{u\}$ serão adicionados posteriormente em t_Y . A complexidade final do algoritmo é $O(|V| \log |V| + |E|)$ [Velo et al., 2014].

2.5 HD-GDD

Li et al. [Li et al., 2017] propõem uma abordagem chamada *High Dimensional Graph Dominance Drawing* ou HD-GDD para o problema de alcançabilidade em grafos muito grandes. Essa abordagem se baseia na FELINE [Velo et al., 2014] com um índice de corte negativo d -dimensional baseado em ordenações topológicas, ao contrário do índice bi-dimensional proposto pela FELINE.

A primeira dimensão, ou ordenação topológica, é construída da mesma forma que ocorre na FELINE, através da criação de uma ordenação topológica utilizando DFS. As dimensões seguintes são construídas através de uma adaptação da heurística *Maximum-Rank* proposta pela FELINE, porém, com a utilização das seguintes regras para determinar a ordem com a qual os vértices fonte são escolhidos a cada passo:

Comparação das coordenadas Se $\forall i \in [1, d], t_i(u) < t_i(v)$, então v será escolhido para ser visitado antes de u na lista de fontes para a construção da dimensão $d + 1$. Caso nenhum ou mais de um vértice satisfaça essa condição, a próxima regra é utilizada.

Somas das coordenadas A fonte f com o maior $\sum_{i=1}^d t_i(f)$ é escolhida para ser visitada primeiro na lista de fontes para a construção da dimensão $d + 1$. Em caso de empate, a próxima regra é utilizada.

Desvio padrão das coordenadas A fonte f com o maior desvio de coordenadas nas dimensões anteriores é escolhida primeiro.

Mínimo das coordenadas anteriores Quando todas as condições anteriores não forem suficientes para determinar qual vértice deverá ser escolhido primeiro na lista de fontes, a fonte com a menor coordenada nas ordenações topológicas já construídas é escolhida.

O número de falsos positivos cai à medida em que o número de ordenações topológicas aumenta, porém, a proporção pela qual o número de falsos positivos diminui, decresce a cada dimensão acrescentada. Por este motivo, essa abordagem funciona bem

somente para um número pequeno de dimensões (geralmente até 5) considerando que, a cada passo de uma consulta de alcançabilidade, uma nova dimensão que efetivamente não introduza um corte negativo para o vértice sendo analisado, introduz mais carga durante o seu processamento.

2.6 IP

Wei et al. [2018] propuseram uma abordagem que usa rotulagem aleatória dos vértices chamada de IP, baseada em permutações independentes [Broder, 1997]. O algoritmo utiliza o fato de que se $u \rightsquigarrow v$, então $In(u) \subseteq In(v)$ e $Out(v) \subseteq Out(u)$. Dessa forma, se $In(u) \not\subseteq In(v)$ ou $Out(v) \not\subseteq Out(u)$, então $u \not\rightsquigarrow v$.

Ao iniciar, o algoritmo gera permutações aleatórias dos vértices utilizando o algoritmo KS (*Knuth shuffle*) [Knuth, 1997]. O algoritmo é não-tendencioso, isto é, qualquer permutação dos vértices tem chances iguais de ser gerada.

Para cada permutação π , o algoritmo gera dois conjuntos, $\mathcal{L}_{in}(u)$ e $\mathcal{L}_{out}(u)$, para cada vértice $u \in V$. O conjunto $\mathcal{L}_{in}(u)$ representa o conjunto dos k menores números $\pi(v)$ tal que $v \in In(u)$. Similarmente, $\mathcal{L}_{out}(u)$ é o conjunto dos k menores números $\pi(v)$ tal que $v \in Out(u)$. O índice é gerado para todos os vértices do grafo em tempo $O(k(|V| + |A|))$ com o tamanho do índice com no máximo $2k|V|$.

A seguir, apresentamos um exemplo extraído de [Wei et al., 2018] para a geração do índice IP. A Figura 2.4 apresenta à esquerda o grafo G usado como neste exemplo. Para cada vértice de G é atribuído uma posição na permutação aleatória π , como mostrado à direita na figura. Por fim, a Tabela 2.1 mostra a geração dos conjuntos \mathcal{L}_{out} e \mathcal{L}_{in} para cada vértice do grafo, usando $k = 5$. Por exemplo, temos $In(v5) = \{v0, v1, v5\}$ e $Out(v5) = \{v5, v7, v8, v10, v11\}$. Com isso, $\mathcal{L}_{in}(v5) = \{\pi(v0), \pi(v1), \pi(v5)\} = \{7, 11, 0\}$ e $\mathcal{L}_{out}(v5) = \{\pi(v5), \pi(v7), \pi(v8), \pi(v10), \pi(v11)\} = \{0, 1, 5, 9, 10\}$. Se $|\mathcal{L}_{in}(v5)|$ ou $|\mathcal{L}_{out}(v5)|$ fosse maior que k , seriam preservados nesses conjuntos somente os menores k elementos.

Um consulta de alcançabilidade $u \overset{?}{\rightsquigarrow} v$ pode ser podada negativamente em um vértice intermediário w sempre que $\min(\mathcal{L}_{in}(v)) > \min(\mathcal{L}_{in}(u)) \vee \min(\mathcal{L}_{out}(v)) > \min(\mathcal{L}_{out}(u))$.

O IP também considera dois índices adicionais. O primeiro índice é composto por duas versões do filtro de níveis proposto pelo GRAIL [Yildirim et al., 2012]. A primeira versão é computada exatamente conforme feito no GRAIL e a segunda versão computada utilizando o grafo transposto. Com isso, o nível inicial igual a 1 é atribuído às raízes e o restante dos níveis são calculados sucessivamente. Assim duas informações

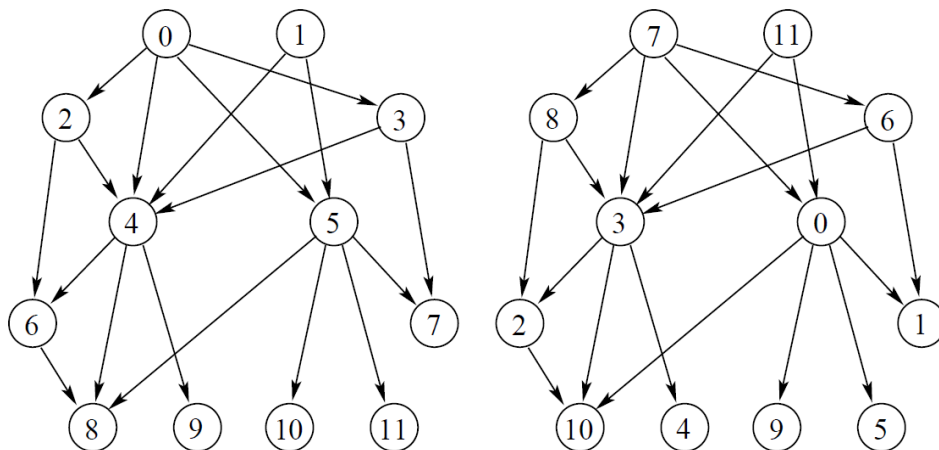


Figura 2.4. Exemplo de DAG G (esquerda) e permutação aleatória $\pi = (7, 11, 8, 6, 3, 0, 2, 1, 10, 4, 9, 5)$ (direita)

Tabela 2.1. Rótulos IP para o DAG G da Figura 2.4 com $k = 5$.

Vértice	\mathcal{L}_{out}	\mathcal{L}_{in}
v_0	{0,1,2,3,4}	{7}
v_1	{0,1,2,3,4}	{11}
v_2	{2,3,4,8,10}	{7,8}
v_3	{1,2,3,4,6}	{6,7}
v_4	{2,3,4,10}	{3,6,7,8,11}
v_5	{0,1,5,9,10}	{0,7,11}
v_6	{2,10}	{2,3,6,7,8}
v_7	{1}	{0,1,6,7,11}
v_8	{10}	{0,2,3,6,7}
v_9	{4}	{3,4,6,7,8}
v_{10}	{9}	{0,7,9,11}
v_{11}	{5}	{0,5,7,11}

topológicas diferentes do grafo são registradas, que também podem fornecer cortes negativos diferentes.

O segundo índice adicional é gerado através da geração do fecho transitivo de vértices do grafo com alto grau de saída. Esse índice efetivo quando o grafo possui vértices em que o grau de saída é discrepante dos demais fazendo com que o tempo de consulta seja afetado negativamente ao passar por esses vértices. Com esse índice, a consulta é podada positivamente ou negativamente assim que encontrado um vértice com alto grau de saída.

2.7 BFL

O *Bloom filter Labeling* (BFL) [Su et al., 2017] propõe outro método de rotulação aleatória para o seu índice. Similarmente ao IP, o BFL explora o fato de que se $In(u) \not\subseteq In(v)$ or $Out(v) \not\subseteq Out(u)$, então $u \not\rightarrow v$.

No BFL, cada vértice é mapeado a um número do conjunto $\{1, 2, \dots, s\}$ por uma função *hash* $g(\cdot)$. O índice relacionado a um vértice u é gerado como um subconjunto de $\{1, 2, \dots, s\}$ tal que $\mathcal{L}_{out}(u) = \bigcup_{(u,w) \in A} \mathcal{L}_{out}(w) \cup \{g(u)\}$.

A Figura 2.5 apresenta um exemplo de grafo (à esquerda) e uma atribuição de números aleatórios a cada vértice considerando $s = 5$ (à direita).

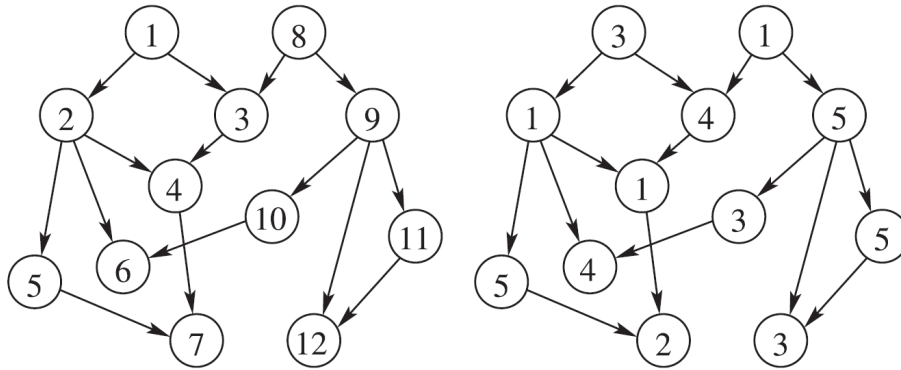


Figura 2.5. Exemplo de DAG G (esquerda) e atribuição de um número do conjunto $\{1, 2, 3, 4, 5\}$ a cada vértice segundo uma função hash (direita)

Tabela 2.2. Rótulos BFL para o DAG G da Figura 2.5.

Vértice	\mathcal{L}_{out}	\mathcal{L}_{in}
v_1	$\{1, 2, 3, 4, 5\}$	$\{3\}$
v_2	$\{1, 2, 4, 5\}$	$\{1, 3\}$
v_3	$\{1, 2, 4\}$	$\{1, 3, 4\}$
v_4	$\{1, 2\}$	$\{1, 3, 4\}$
v_5	$\{2, 5\}$	$\{1, 3, 5\}$
v_6	$\{4\}$	$\{1, 3, 4, 5\}$
v_7	$\{2\}$	$\{1, 2, 3, 4, 5\}$
v_8	$\{1, 2, 3, 4, 5\}$	$\{1\}$
v_9	$\{3, 4, 5\}$	$\{1, 5\}$
v_{10}	$\{3, 4\}$	$\{1, 3, 5\}$
v_{11}	$\{3, 5\}$	$\{1, 5\}$
v_{12}	$\{3\}$	$\{3\}$

Em uma consulta $u \overset{?}{\rightsquigarrow} v$, se $\mathcal{L}_{out}(v) \not\subseteq \mathcal{L}_{out}(w)$, então $w \not\rightsquigarrow v$ e a consulta por ser podada negativamente em um vértice intermediário w . Caso contrário, $u \rightsquigarrow v$ ou o par é um falso-positivo.

O BFL usa um vetor de bits para representar os subconjuntos de $\{1, 2, \dots, s\}$ relacionados a cada vértice. Então, as operações de união para computar cada subconjunto \mathcal{L}_{out} do índice podem ser feitas eficientemente através da execução de operações bit-a-bit entre os vetores. Para valores de s pequenos, isso é feito em uma operação. Essa mesma estratégia é aplicada durante a execução da consulta para verificar se $\mathcal{L}_{out}(v) \not\subseteq \mathcal{L}_{out}(w)$ em um vértice intermediário w e podar a consulta.

Além do índice principal de corte negativo, o BFL utiliza um índice de corte positivo similar ao GRAIL, baseado em intervalos o que ajuda a responder de forma mais eficiente consultas que terminam com $u \rightsquigarrow v$.

Capítulo 3

Equivalência entre Conjunto de Intervalos e Duas Ordenações Topológicas

A maioria das abordagens (e.g. [Yildirim et al., 2012], [Anand et al., 2013] e [Li et al., 2015]) utiliza conjuntos de intervalos como índice para inferir sobre a alcançabilidade entre dois vértices em um DAG. Nas abordagens mencionadas, para um DAG $G = (V, E)$, dados dois vértices $u, v \in V$ e um conjunto de intervalos $L = \{L_1, L_2, \dots, L_{|V|}\}$, u não alcança v se $L_u \not\subseteq L_v$. Quando $L_u \subseteq L_v$, não é possível inferir sobre a alcançabilidade já que pode se tratar de um falso-positivo.

Conforme apresentado no Capítulo 2, a FELINE utiliza duas ordenações topológicas para o mesmo fim. Seja $G = (V, E)$ um DAG e t_X e t_Y duas ordenações topológicas desse DAG, onde $t_X(u)$ representa a posição do vértice u em t_X . Dados dois vértices $v, u \in V$, u não alcança v se $t_X(v) < t_X(u)$ ou $t_Y(v) < t_Y(u)$. Se $t_X(u) < t_X(v)$ e $t_Y(u) < t_Y(v)$ não é possível inferir sobre a alcançabilidade, isto é, de forma semelhante à abordagem por intervalos, este caso pode representar um falso-positivo.

Neste capítulo, será mostrado que as duas representações, seja por um conjunto de intervalos ou por duas ordenações topológicas, são equivalentes e que a partir de uma das representações é possível construir a outra através de um algoritmo simples com complexidade linear.

3.1 Equivalência entre Representações

A seguir apresentaremos dois teoremas que, de forma construtiva, mostram que existe uma equivalência entre as representações baseadas em intervalos e baseadas em ordenações topológicas.

Teorema 1. *Seja $G = (V, A)$ um DAG e seja $L = \{L_1, L_2, \dots, L_{|V|}\}$ um conjunto de intervalos tal que $L_u = [s_u, f_u]$, $f_i \neq f_j, \forall i, j \in V, i \neq j$ e $L_v \not\subseteq L_u \rightarrow u \not\rightsquigarrow v$. Existem duas ordenações topológicas de G , t_X e t_Y tais que $L_v \not\subseteq L_u \rightarrow t_X(v) < t_X(u) \vee t_Y(v) < t_Y(u)$.*

Demonstração. A ordenação topológica t_X é construída pela ordenação dos vértices em ordem crescente em relação a s_u e, em caso de empate, ordenando os vértices de forma decrescente em relação a f_u . Por sua vez, t_Y é construído pela ordenação dos vértices de forma decrescente em relação a f_u .

Se $L_v \not\subseteq L_u$, temos então três cenários:

1. $s_v \leq s_u \leq f_u < f_v$, o que implica em $t_X(v) < t_X(u) \wedge t_Y(v) < t_Y(u)$;
2. $s_u \leq s_v \leq f_u < f_v$, o que implica em $t_Y(v) < t_Y(u)$;
3. $s_v < s_u \leq f_v < f_u$, o que implica em $t_X(v) < t_X(u)$.

Para provar que t_X e t_Y são ordenações topológicas de G , observe que se $u \rightsquigarrow v$, então $L_v \subseteq L_u$ o que implica em $s_u \leq s_v \leq f_v < f_u$ e logo $t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)$. \square

Teorema 2. *Seja $G = (V, A)$ um DAG e t_X e t_Y duas ordenações topológicas de G . Existe um conjunto de intervalos $L = \{L_1, L_2, \dots, L_{|V|}\}$ tal que $L_v \not\subseteq L_u \rightarrow u \not\rightsquigarrow v$ e $t_X(v) < t_X(u) \vee t_Y(v) < t_Y(u) \rightarrow L_v \not\subseteq L_u$.*

Demonstração. Para cada $L_u = [s_u, f_u] \in L$ definimos $s_u = t_X(u)$ e $f_u = 2 \times |V| - t_Y(u) + 1$.

Dessa forma, se $t_X(v) < t_X(u)$, então $s_v < s_u$, portanto, $L_v \not\subseteq L_u$; se $t_Y(v) < t_Y(u)$, então $f_u < f_v$ e novamente $L_v \not\subseteq L_u$.

Para mostrar que $L_v \not\subseteq L_u \rightarrow u \not\rightsquigarrow v$ observe que $u \rightsquigarrow v$ implica em $t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)$ o que significa que $s_u < s_v$ e $f_v < f_u$ e consequentemente $L_v \subseteq L_u$. \square

As demonstrações dos Teoremas 1 e 2 apresentam dois algoritmos para a transformação entre as representações por intervalos e ordenações topológicas. Com isso, é

possível fazer uma tradução entre as representações propostas para as abordagens de índices para o problema de alcançabilidade.

3.2 GRAIL com Ordenações Topológicas

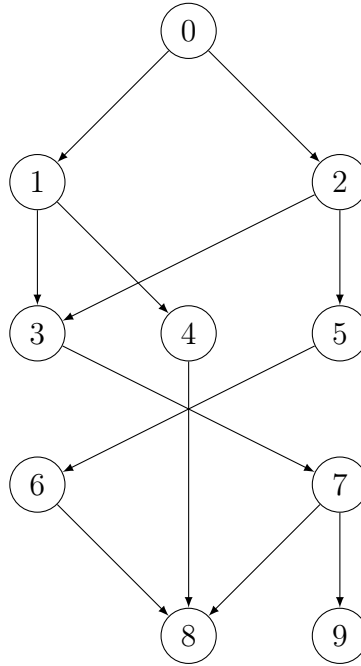


Figura 3.1. Grafo direcionado acíclico apresentado como exemplo em [Yildirim et al., 2010].

A partir da execução do algoritmo proposto pelo GRAIL usando instância apresentada na Figura 3.1 como entrada, temos o seguinte conjunto de intervalos $L = \{L_0 = [1, 10], L_1 = [1, 6], L_2 = [1, 9], L_3 = [1, 4], L_4 = [1, 5], L_5 = [1, 8], L_6 = [1, 7], L_7 = [1, 3], L_8 = [1, 1], L_9 = [2, 2]\}$.

Usando o algoritmo proposto no Teorema 1, temos as ordenações topológicas correspondentes $t_X = (0, 2, 5, 6, 1, 4, 3, 7, 8, 9)$ e $t_Y = (0, 2, 5, 6, 1, 4, 3, 7, 9, 8)$. Nessa representação utilizada pelo GRAIL temos 15 falsos-positivos, são eles: $(2, 1)$, $(2, 4)$, $(4, 3)$, $(4, 7)$, $(4, 9)$, $(5, 1)$, $(5, 3)$, $(5, 4)$, $(5, 7)$, $(5, 9)$, $(6, 1)$, $(6, 3)$, $(6, 4)$, $(6, 7)$ e $(6, 9)$. Entre t_X e t_Y existe apenas o falso-positivo entre o par $(8, 9)$.

Executando a heurística *Maximum-Rank* utilizada pela FELINE com t_X como entrada, teríamos como resultado a ordenação topológica $t_Y = (0, 1, 4, 2, 3, 7, 9, 5, 6, 8)$, com apenas 3 falsos-positivos, são eles: $(4, 3)$, $(4, 7)$ e $(9, 8)$. Este simples exemplo mostra uma grande diferença entre as abordagens em relação o número de falsos-positivos

gerados, utilizando o mesmo espaço de armazenamento. Além disso, a heurística utilizada pela FELINE é localmente ótima [Kornaropoulos, 2012; Veloso et al., 2014], isto é, não há como aumentar o número de inversões fazendo apenas uma troca entre vértices adjacentes (*swap*) em t_Y . Considerando a solução apresentada pelo GRAIL, a simples inversão do par (4, 2) ou do par (9, 5) em t_Y já produz uma solução com menos falsos-positivos.

3.3 FELINE com Intervalos

Conforme apresentado na seção anterior, para o exemplo de DAG da Figura 3.1 e usando a ordenação topológica $t_X = (0, 2, 5, 6, 1, 4, 3, 7, 8, 9)$ como entrada, a heurística *Maximum-Rank* produz a ordenação topológica $t_Y = (0, 1, 4, 2, 3, 7, 9, 5, 6, 8)$.

Podemos então, utilizando o algoritmo proposto na demonstração do Teorema 2, obter os intervalos correspondentes às ordenações topológicas t_X e t_Y . O resultado da aplicação do algoritmo é $L = \{L_0 = [1, 20], L_1 = [5, 19], L_2 = [2, 17], L_3 = [7, 16], L_4 = [6, 18], L_5 = [3, 13], L_6 = [4, 12], L_7 = [8, 15], L_8 = [9, 11], L_9 = [10, 14]\}$.

Os mesmos falsos positivos (4, 3), (4, 7) e (9, 8) continuam presentes na representação por intervalos, já que $L_3 \subset L_4$ e $L_7 \subset L_4$ e, no entanto, $4 \not\leftrightarrow 3$ e $4 \not\leftrightarrow 7$.

3.4 Comparação entre Índices da FELINE e GRAIL

Considerando que as representações são equivalentes em termos de expressividade e espaço de armazenamento, então é possível fazer uma comparação mais justa entre diferentes algoritmos para geração de índices.

O GRAIL utiliza múltiplos conjuntos de intervalos no seu índice. Esses conjuntos de intervalos são construídos pela simples alteração da ordem de visitação dos vértices, definida através de uma ordenação aleatória dos vértices para cada conjunto de intervalos. Porém, o algoritmo de geração do índice é limitado pois não permite a construção de todas as combinações de intervalos possíveis (ou todas as ordenações topológicas do grafo), já que está associada a um caminharmento no grafo.

Em contrapartida, a heurística *Maximum-Rank* permite gerar qualquer ordenação topológica do grafo, alterando apenas a escolha de qual fonte entrará na ordenação topológica em cada iteração, escolha esta que é feita de forma gulosa com intenção de inverter o maior número de pares de vértices nas duas ordenações. Por este motivo, a heurística *Maximum-Rank* tem um papel fundamental no bom desempenho de consulta

da FELINE com um ganho significativo em comparação ao GRAIL, e é o que diferencia as duas abordagens.

No entanto, apesar do índice de corte negativo da FELINE gerar melhores resultados que o do GRAIL, vamos mostrar esses resultados não são estritamente melhores a partir de um exemplo. Primeiramente, para comparar as duas abordagens, definimos uma maneira de gerar os índices na qual os vértices do grafo devem ser visitados por ambas as abordagens em uma mesma ordem pré-definida.

Seja um DAG $G = (V, E)$ e uma permutação σ de V . Seja o conjunto de intervalos de corte negativo $L = \{L_1, L_2, \dots, L_{|V|}\}$ gerados pelo algoritmo do GRAIL visitando os vértices na ordem definida por σ . Seja também as ordenações topológicas t_X e t_Y geradas pelo algoritmo da FELINE, t_X através de um caminharmento no grafo por DFS visitando os vértices na ordem definida por σ , e t_Y pela heurística *MaximumRank* com t_X fornecida como entrada. Queremos mostrar através de um exemplo que, para todo par de vértices (u, v) em G , $L_v \not\subset L_u \not\Rightarrow t_X(v) < t_X(u) \vee t_Y(v) < t_Y(u)$.

Seja $G = (V, E)$ o DAG representado na Figura 3.2 e $\sigma = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$. Os intervalos gerados pelo algoritmo do GRAIL visitando os vértices na ordem definida por σ também estão representados na Figura.

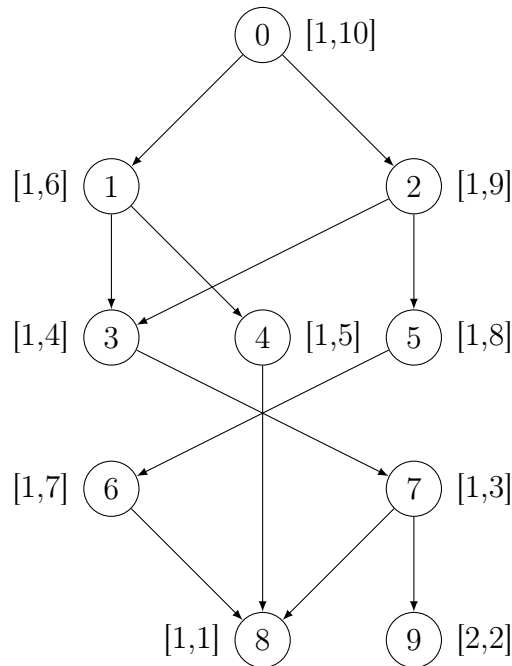


Figura 3.2. DAG e intervalos gerados para o índice de corte negativo pelo GRAIL.

A primeira ordenação topológica gerada pela FELINE utiliza o tempo de finalização do DFS na ordem natural de visitação dos vértices, que é justamente de-

finida pela permutação σ . Com isso temos $t_X = (0, 2, 5, 6, 1, 4, 3, 7, 9, 8)$. Após a utilização da heurística *Maximum-Rank* fornecendo G e t_X como entrada, temos $t_Y = (0, 1, 4, 2, 3, 7, 9, 5, 6, 8)$.

O índice de corte negativo da FELINE, representado por t_X e t_Y , apresenta o falso-positivo $(9, 8)$, pois os vértices estão na mesma posição relativa em t_X e t_Y . Os vértices 8 e 9 são incomparáveis, isto é, $8 \not\prec 9$ e $9 \not\prec 8$. Os intervalos produzidos pelo GRAIL não possuem o falso-positivo $(9, 8)$, já que $L_8 \not\subset L_9$ e $L_9 \not\subset L_8$.

O exemplo apresentado pode ser estendido acrescentando-se uma cadeia de vértices conectados em sequência com origem no vértice 8. Dessa forma, o índice da FELINE pode ser tão pior que o índice do GRAIL quanto se queira com relação ao número de falsos-positivos.

Capítulo 4

Desenho de Dominância Fraca Unilateral

4.1 Introdução

Os resultados apresentados neste Capítulo foram publicados em [da Silva et al., 2018].

Um dos problemas recorrentes na área de Desenho de Grafos (*Graphs Drawing*) é representar um grafo direcionado acíclico (*Directed Acyclic Graph* ou DAG) no Plano Cartesiano de forma que, a partir da posição dos vértices no desenho, seja possível inferir sobre a alcançabilidade entre eles. Esse tipo de representação é chamado de Desenho de Dominância (*Dominance Drawing*).

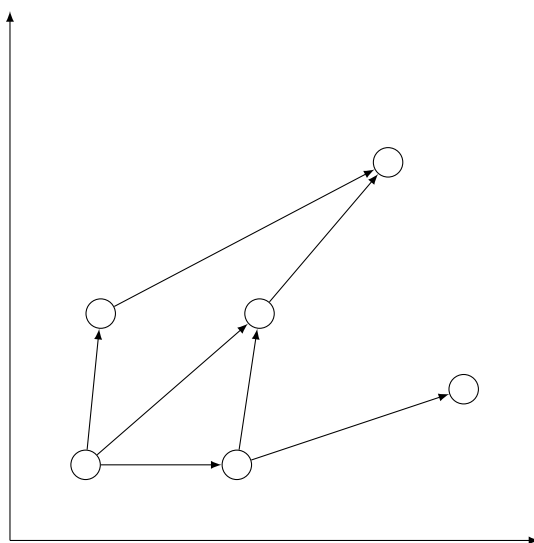


Figura 4.1. Exemplo de DAG e respectivo Desenho de Dominância.

A Figura 4.1 mostra um exemplo de DAG que possui Desenho de Dominância, e sua alcançabilidade pode ser inferida a partir do desenho apresentado.

É possível enxergar o Desenho de Dominância de um grafo como uma ordenação parcial R dos vértices. Seja $G = (V, E)$ um grafo direcionado acíclico e dois vértices $u, v \in V$. Se $X(u) \leq X(v) \wedge Y(u) \leq Y(v)$, então, $u \prec_R v$ ou $(u, v) \in R$. Caso contrário, $u \not\prec v$ ou $(u, v) \notin R$. Se $u \not\prec_R v \wedge v \not\prec_R u$, então u e v são incomparáveis. Através desta representação, esse problema também pode ser formulado como encontrar duas extensões lineares, $Ext_X(R)$ e $Ext_Y(R)$, de forma que $Ext_X(R) \cap Ext_Y(R) = R$. Isto só é possível quando $dim(R) \leq 2$.

Outra maneira de enxergar esse problema consiste em encontrar duas ordenações topológicas, t_X e t_Y de G , tal que $t_X \cap t_Y = tr(G)$ onde $tr(G)$ representa o fecho transitivo de G . A interseção entre duas ordenações topológicas é o conjunto de todos os pares ordenados de vértices (u, v) tais que u e v estão na mesma posição relativa em t_X e t_Y ou ainda, mais formalmente, $I(t_X, t_Y) = \{(u, v) | t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)\}$.

4.2 Desenho de Dominância Fraca

Em 2011, foi proposto o Problema de Desenho de Dominância Fraca (*Weak Dominance Drawing* ou WDD) [Kornaropoulos & Tollis, 2011] que consiste em, dado um DAG $G = (V, E)$ e um inteiro positivo C , responder se existe uma coleção de duas ordenações topológicas, t_X e t_Y de G , tais que sua interseção tenha cardinalidade c ou menor.

Dados dois vértices $u, v \in V$, se existir um caminho de u para v em G , então $t_X(u) < t_X(v)$ e $t_Y(u) < t_Y(v)$. No entanto, o fato de que $t_X(u) < t_X(v)$ e $t_Y(u) < t_Y(v)$ não é suficiente para determinar se existe uma caminho de u para v . No caso em que $t_X(u) < t_X(v)$, $t_Y(u) < t_Y(v)$ e não existe um caminho de u para v , dizemos que o par ordenado (u, v) representa um falso positivo.

Dessa forma, se dois vértices $u, v \in V$ estão na mesma posição relativa em t_X e t_Y (assumindo sem perda de generalidade que $t_X(u) < t_X(v)$ e $t_Y(u) < t_Y(v)$) então não é possível concluir sobre a alcançabilidade de u para v . Neste caso, o par ordenado (u, v) faz parte da interseção entre t_X e t_Y , que é definida como $I(t_X, t_Y) = \{(u, v) | t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)\}$.

Se $u, v \in V$ estiverem em ordem invertida em t_X e t_Y (assumindo sem perda de generalidade que $t_X(u) < t_X(v)$ e $t_Y(u) > t_Y(v)$), u não alcança v e também v não alcança u . Podemos definir o conjunto de pares invertidos ou inversões entre t_X e t_Y como $\bar{I}(t_X, t_Y) = \{(u, v) | t_X(u) < t_X(v) \wedge t_Y(u) > t_Y(v)\}$.

A versão de decisão do WDD pode ser escrita como:

(WDD) DESENHO DE DOMINÂNCIA FRACA

ENTRADA: Um grafo direcionado acíclico $G = (V, E)$ e um inteiro positivo $c \geq 0$.

PERGUNTA: Existem duas ordenações topológicas t_X e t_Y de G tal que $|I(t_X, t_Y)| \leq c$?

WDD foi provado ser NP-Completo por Kornaropoulos & Tollis [2012b]. O caso especial onde é necessário decidir se existem duas ordenações topológicas as quais a interseção é exatamente o fecho transitivo de G é possível resolver através de um algoritmo polinomial [Langley, 1995]. Por outro lado, mesmo essa versão se torna NP-Completa se um número k de ordenações topológicas for considerado, para qualquer $k \geq 3$ [Yannakakis, 1982].

4.2.1 Formulação Matemática para o WDD

Propomos, então, uma formulação de programação inteira para o WDD. As formulações apresentadas a seguir foram publicadas em [da Silva & Urrutia, 2016].

Sejam as variáveis \mathcal{X}_{ij} e \mathcal{Y}_{ij} correspondentes às ordenações topológicas t_X e t_Y , respectivamente, definidas como a seguir:

$$1. \mathcal{X}_{ij} = \begin{cases} 1 & \text{se } t_X(i) < t_X(j) \\ 0 & \text{c.c.} \end{cases}$$

$$2. \mathcal{Y}_{ij} = \begin{cases} 1 & \text{se } t_Y(i) < t_Y(j) \\ 0 & \text{c.c.} \end{cases}$$

Com esta definição, propomos a seguinte formulação para o WDD:

$$\text{Minimizar } \sum_{i=1}^{|V|} \sum_{j=i+1}^{|V|} \mathcal{X}_{ij} \mathcal{Y}_{ij} + (1 - \mathcal{X}_{ij})(1 - \mathcal{Y}_{ij})$$

sujeito a:

$$\mathcal{X}_{ij} = 1 \quad \forall (i, j) \in A \quad (4.1)$$

$$\mathcal{X}_{ij} + \mathcal{X}_{ji} = 1 \quad \forall i, j \in V, i \neq j \quad (4.2)$$

$$\mathcal{X}_{ij} + \mathcal{X}_{jk} + \mathcal{X}_{ki} \leq 2 \quad \forall i, j, k \in V, i \neq j \neq k \quad (4.3)$$

$$\mathcal{Y}_{ij} = 1 \quad \forall (i, j) \in A \quad (4.4)$$

$$\mathcal{Y}_{ij} + \mathcal{Y}_{ji} = 1 \quad \forall i, j \in V, i \neq j \quad (4.5)$$

$$\mathcal{Y}_{ij} + \mathcal{Y}_{jk} + \mathcal{Y}_{ki} \leq 2 \quad \forall i, j, k \in V, i \neq j \neq k \quad (4.6)$$

$$\mathcal{X}_{ij}, \mathcal{Y}_{ij} \in \{0, 1\} \quad \forall i, j \in V, i \neq j \quad (4.7)$$

A função objetivo minimiza o número de interseções entre \mathcal{X} e \mathcal{Y} na qual $\mathcal{X}_{ij} = 1$ e $\mathcal{Y}_{ij} = 1$, ou simetricamente $\mathcal{X}_{ij} = 0$ e $\mathcal{Y}_{ij} = 0$. As restrições (1) fixam a ordem relativa dos pares de vértices para os quais existem arcos no grafo. As restrições (2) e (3) garantem que não existirão ciclos na ordenação total \mathcal{X} . As restrições (4), (5) e (6) tem as mesmas funções das restrições (1), (2) e (3), porém, relativas às ordenação topológica \mathcal{Y} .

O modelo acima é não-linear, já que inclui o produto $\mathcal{X}_{ij}\mathcal{Y}_{ij}$ na sua função objetivo. Pode ser linearizado através da introdução das variáveis \mathcal{Z}_{ij} , onde $\mathcal{Z}_{ij} = 1$ se $\mathcal{X}_{ij} = 1$ e $\mathcal{Y}_{ij} = 1$, e $\mathcal{Z}_{ij} = 0$ caso contrário. O modelo resultante é apresentado a seguir:

$$\text{Minimizar} \quad \sum_{i=1}^{|V|} \sum_{j=1, j \neq i}^{|V|} \mathcal{Z}_{ij}$$

sujeito a:

$$\mathcal{Z}_{ij} \geq \mathcal{X}_{ij} + \mathcal{Y}_{ij} - 1 \quad \forall i, j \in V, i \neq j \quad (4.8)$$

$$\mathcal{X}_{ij} = 1 \quad \forall (i, j) \in A \quad (4.9)$$

$$\mathcal{X}_{ij} + \mathcal{X}_{ji} = 1 \quad \forall i, j \in V, i \neq j \quad (4.10)$$

$$\mathcal{X}_{ij} + \mathcal{X}_{jk} + \mathcal{X}_{ki} \leq 2 \quad \forall i, j, k \in V, i \neq j \neq k \quad (4.11)$$

$$\mathcal{Y}_{ij} = 1 \quad \forall (i, j) \in A \quad (4.12)$$

$$\mathcal{Y}_{ij} + \mathcal{Y}_{ji} = 1 \quad \forall i, j \in V, i \neq j \quad (4.13)$$

$$\mathcal{Y}_{ij} + \mathcal{Y}_{jk} + \mathcal{Y}_{ki} \leq 2 \quad \forall i, j, k \in V, i \neq j \neq k \quad (4.14)$$

$$\mathcal{X}_{ij}, \mathcal{Y}_{ij}, \mathcal{Z}_{ij} \in \{0, 1\} \quad \forall i, j \in V, i \neq j \quad (4.15)$$

4.3 FELINE e Desenho de Dominância Fraca

Conforme apresentado no Capítulo 2, FELINE [Veloso et al., 2014] é uma abordagem inovadora baseada no Desenho de Dominância Fraca [Kornaropoulos & Tollis, 2012b]. Utiliza o fato de que, dados dois vértices $v, u \in V$ e uma ordenação topológica t , se $t(v) < t(u)$ então u não alcança v .

O seu índice é representado por duas ordenações topológicas, t_X e t_Y , que são geradas em um estágio inicial de construção. Para melhorar o desempenho do índice, a cardinalidade do conjunto de interseções $I(t_X, t_Y)$ precisa ser reduzida ao mínimo possível, o que significa resolver a versão de otimização do WDD. Para cada consulta de alcançabilidade entre u e v , esse índice é utilizado como um corte negativo enquanto o grafo é percorrido por meio de DFS, isto é, a busca é podada sempre que $t_X(z) > t_X(v)$

ou $t_Y(z) > t_Y(v)$ para qualquer z alcançado durante a execução do DFS de u para v .

No primeiro estágio de construção do índice, t_X é gerado utilizando DFS. Então, FELINE utiliza a heurística Maximum-Rank, apresentada na Seção 2.4, para construir t_Y .

Claramente, a heurística Maximum-Rank se propõe a resolver um problema diferente do WDD, já que a primeira ordenação topológica é fixada no início da execução do algoritmo, o que reduz drasticamente o espaço de busca de soluções. Para este novo problema, até onde vai nosso conhecimento, nenhum algoritmo polinomial é conhecido e sua complexidade computacional até este trabalho estava em aberto.

4.4 Desenho de Dominância Fraca Unilateral

Neste trabalho, propomos então o problema do Desenho de Dominância Fraca Unilateral (*One-Sided Weak Dominance Drawing* ou OSWDD) que é uma especialização do WDD onde é fornecida a ordenação topológica t_X como entrada para o problema. Com isso, é necessário obter apenas uma das ordenações topológicas, t_Y , que minimize $I(t_X, t_Y) = \{(u, v) | t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)\}$.

A versão de decisão do problema pode ser descrita como:

(OSWDD) Desenho de Dominância Fraca Unilateral

ENTRADA: Um grafo direcionado acíclico $G = (V, E)$, um ordenação topológica t_X de G e um inteiro positivo $k \geq 0$.

PERGUNTA: Existe uma ordenação topológica t_Y de G tal que $|I(t_X, t_Y)| \leq k$?

4.4.1 Formulação Matemática para o OSWDD

A seguir, apresentamos uma formulação de programação inteira para o OSWDD. A formulação é similar à proposta para o WDD, porém, fixamos a ordenação topológica t_X o que significa tornar constantes as variáveis \mathcal{X}_{ij} .

$$\text{Minimizar } \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} x_{ij} \mathcal{Y}_{ij} + (1 - x_{ij})(1 - \mathcal{Y}_{ij})$$

sujeito a:

$$\mathcal{Y}_{ij} = 1 \quad \forall (i, j) \in A \quad (4.16)$$

$$\mathcal{Y}_{ij} + \mathcal{Y}_{ji} = 1 \quad \forall i, j \in V, i \neq j \quad (4.17)$$

$$\mathcal{Y}_{ij} + \mathcal{Y}_{jk} + \mathcal{Y}_{ki} \leq 2 \quad \forall i, j, k \in V, i \neq j \neq k \quad (4.18)$$

$$\mathcal{Y}_{ij} \in \{0, 1\} \quad \forall i, j \in V, i \neq j \quad (4.19)$$

4.5 Complexidade do OSWDD

Nesta seção iremos apresentar uma prova que conclui sobre a NP-Compleitude do OSWDD. Para tanto, a seguir, será introduzido o Lema 1 que se aplica a permutações quaisquer.

4.5.1 Lema Preliminar

Na prova a seguir, usamos o conceito de posições à esquerda ou à direita de um certo elemento em uma permutação. Dizemos que u está à esquerda de v na permutação σ se $\sigma(u) < \sigma(v)$, e está à direita caso contrário.

Lema 1. *Sejam $\mathcal{F} = \{S(1), S(2), \dots, S(n)\}$ uma partição de um conjunto S com $S(i) = \{a_1(i), a_2(i), \dots, a_{m_i}(i)\}$. Dada uma permutação σ de S tal que $\sigma(a_r(i)) < \sigma(a_s(i))$, $\forall i \in \{1 \dots n\}, \forall r, s, 1 \leq r < s \leq m_i$, e uma permutação τ de S tal que $\tau(a_r(i)) > \tau(a_s(i))$, $\forall i \in \{1 \dots n\}, \forall r, s, 1 \leq r < s \leq m_i$, existe uma permutação ω de S tal que $\omega(a_r(i)) > \omega(a_s(i))$, $\forall i \in \{1 \dots n\}, \forall r, s, 1 \leq r < s \leq m_i$ na qual todos os elementos do mesmo conjunto $S(i)$ aparecem consecutivamente em ω e $|\bar{I}(\sigma, \omega)| \leq |\bar{I}(\sigma, \tau)|$, ou seja, o número de inversões entre σ e ω é menor ou igual ao número de inversões entre σ e τ .*

Demonstração. Seja $a_r(i), a_{r+1}(i) \in S(i)$. Seja $B_\sigma(a_r(i), a_{r+1}(i))$ os elementos de S entre $a_r(i)$ e $a_{r+1}(i)$ em σ , e $B_\tau(a_{r+1}(i), a_r(i))$ os elementos entre $a_{r+1}(i)$ e $a_r(i)$ em τ . Note que não existem elementos de $S(i)$ em $B_\sigma(a_r(i), a_{r+1}(i))$ ou em $B_\tau(a_{r+1}(i), a_r(i))$.

Vamos dividir os elementos de $B_\tau(a_{r+1}(i), a_r(i))$ em três conjuntos: $T_1 = \{a_t(j) \in B_\tau(a_{r+1}(i), a_r(i)) | \sigma(a_t(j)) < \sigma(a_r(i))\}$; $T_2 = \{a_t(j) \in B_\tau(a_{r+1}(i), a_r(i)) | a_t(j) \in B_\sigma(a_r(i), a_{r+1}(i))\}$; e $T_3 = \{a_t(j) \in B_\tau(a_{r+1}(i), a_r(i)) | \sigma(a_t(j)) > \sigma(a_{r+1}(i))\}$.

Observe que para cada elemento $a_t(j) \in T_2 \cup T_3$ o par $(a_r(i), a_t(j))$ pertence a $\bar{I}(\sigma, \tau)$; para cada elemento $a_t(j) \in T_1$ o par $(a_t(j), a_r(i))$ não pertence a $\bar{I}(\sigma, \tau)$; para cada elemento $a_t(j) \in T_1 \cup T_2$ o par $(a_t(j), a_{r+1}(i))$ pertence a $\bar{I}(\sigma, \tau)$; para cada elemento $a_t(j) \in T_3$ o par $(a_{r+1}(i), a_t(j))$ não pertence a $\bar{I}(\sigma, \tau)$.

A partir de τ vamos definir uma nova permutação τ' colocando $a_r(i)$ na próxima posição à direita de $a_{r+1}(i)$ e deslocando cada elemento em $B_\tau(a_{r+1}(i), a_r(i))$ uma posição para a direita. Note que cada par $(a_t(j), a_r(i))$ tal que $a_t(j) \in T_1$ pertence a $\bar{I}(\sigma, \tau')$, cada par $(a_r(i), a_t(j))$ tal que $a_t(j) \in T_2 \cup T_3$ não pertence a $\bar{I}(\sigma, \tau')$ e todos os outros pares de elementos em S pertencem a $\bar{I}(\sigma, \tau')$ se e somente se eles pertencem a $\bar{I}(\sigma, \tau)$. Então, $|\bar{I}(\sigma, \tau')| = |\bar{I}(\sigma, \tau)| + |T_1| - |T_2| - |T_3|$.

Similarmente, vamos definir a permutação τ'' colocando $a_{r+1}(i)$ na posição à esquerda de $a_r(i)$ e deslocando cada elemento em $B_\tau(a_{r+1}(i), a_r(i))$ uma posição para a esquerda. Então, por argumento similar, $|\bar{I}(\sigma, \tau'')| = |\bar{I}(\sigma, \tau)| - |T_1| - |T_2| + |T_3|$.

Se $|T_1| > |T_3|$, então $|\bar{I}(\sigma, \tau'')| < |\bar{I}(\sigma, \tau)|$ já que $-|T_1| - |T_2| + |T_3| < 0$. Por outro lado, se $|T_3| > |T_1|$, então $|\bar{I}(\sigma, \tau')| < |\bar{I}(\sigma, \tau)|$ já que $|T_1| - |T_2| - |T_3| < 0$. Finalmente, se $|T_1| = |T_3|$, então $|\bar{I}(\sigma, \tau'')| = |\bar{I}(\sigma, \tau')| \leq |\bar{I}(\sigma, \tau)|$ desde que $-|T_1| - |T_2| + |T_3| = |T_1| - |T_2| - |T_3| = -|T_2| \leq 0$.

Agora introduzimos um algoritmo que iterativamente define novas permutações a partir de τ da forma descrita a seguir. Enquanto existe pelo menos um conjunto $S(i)$ no qual os elementos não estão consecutivos em τ e considerando cada conjunto $S(i)$ em uma ordem determinada, escolha dois elementos não consecutivos de $S(i)$ e substitua τ por τ' toda vez que $|T_1| \leq |T_3|$ e por τ'' caso contrário. A cada iteração o número de inversões entre σ e τ decresce ou um elemento do conjunto $S(i)$ é colocado à esquerda da sua posição anterior. Então, em algum momento o algoritmo termina com cada conjunto $S(i)$ com todos os elementos consecutivos já que quando os elementos de $S(i)$ estão consecutivos, o algoritmo nunca o separa novamente.

Com o algoritmo descrito acima é possível obter ω tal que $\omega(a_r(i)) > \omega(a_s(i))$, $\forall i \in \{1 \dots n\}, \forall r, s, 1 \leq r < s \leq m_i$ no qual todos os elementos de cada conjunto $S(i)$ aparecem consecutivamente em ω e $|\bar{I}(\sigma, \tau)| \geq |\bar{I}(\sigma, \omega)|$. \square

Informalmente, o Lema 1 diz que se σ e τ são (a) permutações sob o mesmo conjunto S , (b) a família $\mathcal{F} = \{S(1), \dots, S(n)\}$ é uma partição de S e (c) cada subpermutação de τ induzida por um elemento de \mathcal{F} está invertida em σ em relação a τ , então existe uma permutação ω com as mesmas propriedades que τ tal que os elementos de cada $S(i)$ ocorrem consecutivamente em ω .

A Figura 4.2 ilustra a execução do algoritmo descrito no Lema 1, que será detalhada a seguir. Cada subconjunto $S(i)$ é representado por uma cor distinta. A

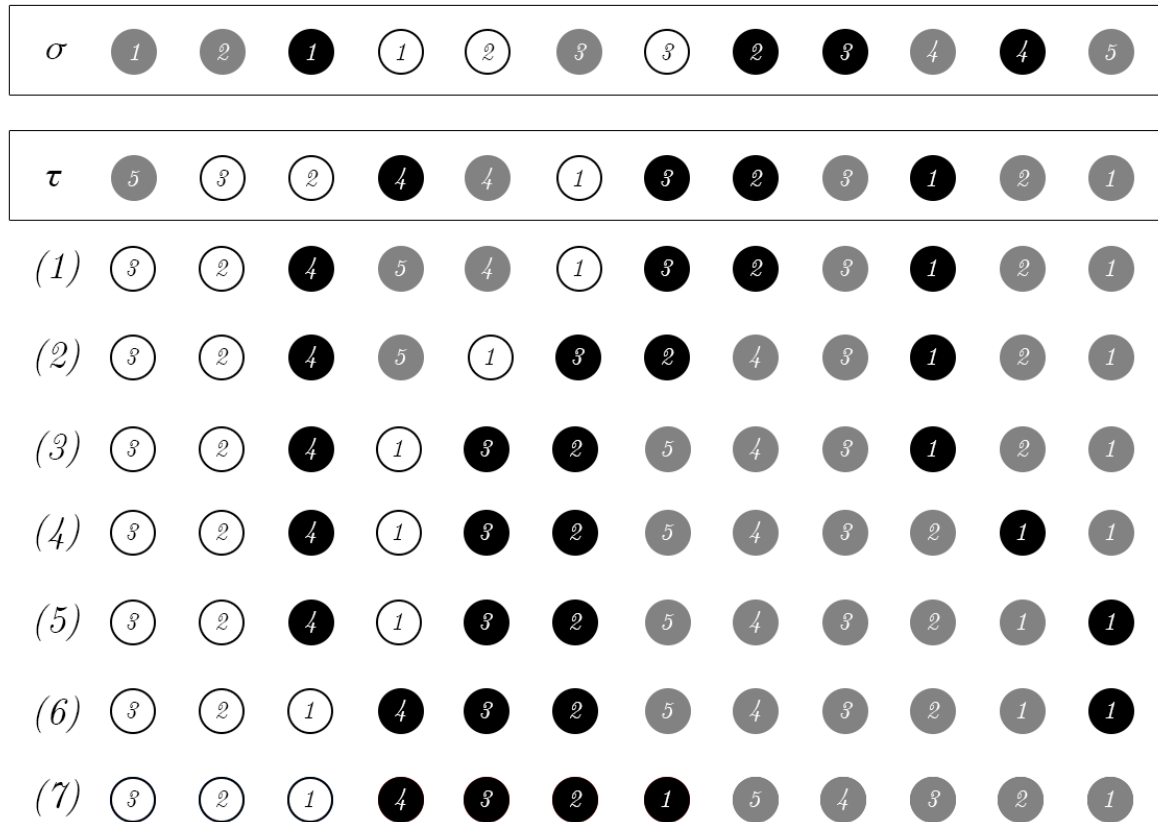


Figura 4.2. Exemplo de execução do algoritmo descrito no Lema 1.

permutação original τ possui $|\bar{I}(\sigma, \tau)| = 54$.

Permutação τ Os elementos em cinza são considerados primeiro, iniciando pelos elementos 5 e 4. Neste caso, T_1 é composto pelos elementos 2 e 3 brancos, T_2 é composto pelo elemento 4 preto e T_3 é vazio. Como $|T_1| > |T_3|$, o elemento 5 cinza é movimentado para a posição mais à esquerda do elemento 4 da mesma cor gerando a permutação (1) com o número de inversões reduzido em 3.

Permutação (1) São considerados os elementos cinza 4 e 3 com $|T_1| = 1$, $|T_2| = 2$ e $|T_3| = 0$. Como $|T_1| > |T_3|$, o elemento 4 cinza é movimentado para a posição mais à esquerda do elemento 3 gerando a permutação (2), reduzindo o número de inversões em 3.

Permutação (2) A movimentação do elemento 4 cinza criou uma lacuna entre os elementos 5 e 4. Com isso, consideramos novamente os elementos 5 e 4 com $|T_1| = 3$, $|T_2| = 0$ e $|T_3| = 0$. Como $|T_1| > |T_3|$, o elemento 5 é movimentado para a posição mais à esquerda do elemento 4, gerando a permutação (3).

Permutação (3) Consideramos agora os elementos cinza 3 e 2 com $|T_1| = 0$, $|T_2| = 1$ e $|T_3| = 0$. Neste caso, $|T_1| = |T_3| = 0$ e, segundo o algoritmo proposto, o elemento 2 cinza é movimentado para a posição mais à direita do elemento 3 da mesma cor, resultado na permutação (4).

Permutação (4) Consideramos agora os elementos cinza 2 e 1 com $|T_1| = 0$, $|T_2| = 0$ e $|T_3| = 1$. Neste caso, $|T_1| < |T_3|$, então, o elemento 1 cinza é movimentado para a posição mais a esquerda do elemento 2 da mesma cor, resultado na permutação (5).

Permutação (5) Neste passo todos os elementos em cinza estão consecutivos e $|\bar{I}(\sigma, \tau_{(5)})| = 43$. São considerados agora os elementos brancos, começando por 2 e 1, já que os elementos 3 e 2 já estão consecutivos em (5). Neste caso, $|T_1| = 0$, $|T_2| = 0$ e $|T_3| = 1$, então movimentamos o elemento 1 para a posição mais à direita do elemento 2 branco.

Permutação (6) Agora, como os elementos cinzas e brancos já estão consecutivos, consideramos os elementos 2 e 1 de cor preta, que são os únicos não consecutivos. Para esses elementos, $|T_1| = 2$, $|T_2| = 1$ e $|T_3| = 2$. Como $|T_1| = |T_3|$, segundo o algoritmo proposto, o elemento 1 é movimentado para a posição mais à direita do elemento 2.

Permutação (7) No final da execução, todos os elementos da mesma cor estão consecutivos e $|\bar{I}(\sigma, \tau_{(7)})| = 41$.

4.5.2 OSWDD é NP-Completo

A formulação proposta para OSWDD considera a minimização de $I(t_X, t_Y)$. Uma formulação equivalente do OSWDD pode ser obtida considerando a maximização de $\bar{I}(t_X, t_Y)$. Esse novo conjunto é o complemento do conjunto de interseções $I(t_X, t_Y)$ sob o conjunto de pares de vértices ordenados nos quais $t_X(u) < t_X(v)$ e, como consequência, maximizar sua cardinalidade implica em minimizar a cardinalidade de $I(t_X, t_Y)$.

Note que se, em vez de considerar t_X considerarmos o conjunto invertido, o qual chamamos de \bar{t}_X , então o OSWDD é equivalente a minimizar a cardinalidade de $\bar{I}(\bar{t}_X, t_Y)$. Então, uma nova definição do problema de decisão pode ser definida como:

(OSWDD) Desenho de Dominância Fraca Unilateral

ENTRADA: Um grafo direcionado acíclico $G = (V, E)$, uma permutação de V , que

chamamos de t_X^- , na qual o seu reverso t_X é uma ordenação topológica G e um inteiro não negativo k .

PERGUNTA: Existe uma ordenação topológica t_Y de G tal que o $|\bar{I}(t_X^-, t_Y)| \leq k$?

Em Brandenburg et al. [2012] é mostrado que a versão de decisão do problema *nearest neighbor Kendall tau* para uma ordenação total e outra parcial (KNN_{TP}) é NP-completa. Esse problema considera uma ordenação parcial κ e uma ordenação total σ de um dado conjunto e tem por objetivo encontrar um extensão linear de κ que tem o menor número de inversões em comparação com σ .

KNN_{TP} é similar ao OSWDD. Eles são diferentes ao passo que, enquanto KNN_{TP} não considera nenhuma relação entre κ e σ além de serem definidos sob o mesmo conjunto, no OSWDD o grafo G (i.e. a ordem parcial) e t_X^- (i.e. a ordenação total) são definidas tal que o reverso de t_X^- é uma ordenação topológica de G . Então, o OSWDD pode ser visto como um caso especial de KNN_{TP} . Dessa forma, o fato de KNN_{TP} ser NP-Completo não implica que o OSWDD seja também NP-Completo.

A seguir provamos que a versão de decisão do OSWDD é NP-Completa por uma transformação polinomial a partir do *One-Sided Crossing Minimization 4 Star* [Muñoz et al., 2002].

Teorema 3. *OSWDD é NP-Completo.*

Demonstração. O problema *One-Sided Crossing Minimization 4 Star* (OSCM-4-Star) consiste em posicionar vértices de um conjunto de n 4-stars (um vértice central conectado a quatro folhas) em duas linhas paralelas. As folhas são posicionadas na linha superior e os centros na linha inferior. A Figure 4.3 mostra um exemplo de instância do OSCM-4-Star.

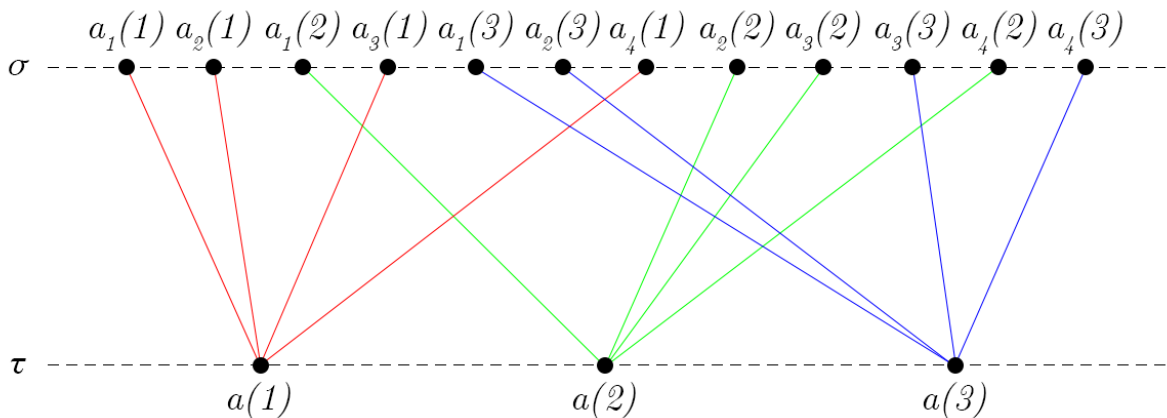


Figura 4.3. Instância do OSCM-4-Star.

Dada uma ordenação fixa de folhas na linha superior, o problema consiste em encontrar uma permutação dos centros na linha inferior tal que o número de cruzamentos é minimizado. A versão de decisão desse problema foi provada ser NP-Completa em [Eades & Whitesides, 1994] e pode ser definida como:

(OSCM-4-Star) *One Sided Crossing Minimization 4 Star*

ENTRADA: Um conjunto de centros $S = \{a(1), a(2), \dots, a(n)\}$, uma permutação σ sob o conjunto de folhas $A = \cup_{i \in \{1 \dots n\}} A(i)$, onde $A(i) = \{a_1(i), a_2(i), a_3(i), a_4(i)\}$ e um inteiro positivo k' .

PERGUNTA: Existe uma permutação τ dos centros em S tal que a cardinalidade do conjunto de cruzamentos $\{(a_r(i), a_s(j)) | \sigma(a_r(i)) < \sigma(a_s(j)), \tau(a(j)) < \tau(a(i)), i, j \in \{1 \dots n\}, s, r \in \{1 \dots 4\}\}$ não é maior que k' ?

Podemos assumir que, sem perda de generalidade, $\sigma(a_1(i)) < \sigma(a_2(i)) < \sigma(a_3(i)) < \sigma(a_4(i)), i \in \{1 \dots n\}$. Em qualquer outro caso, é possível associar novos rótulos aos vértices correspondentes em cada conjunto $A(i)$ para que essa condição seja mantida.

Para reduzir o problema de minimização de cruzamentos OSCM-4-Star para OSWDD, introduzimos primeiramente uma formulação alternativa para o OSCM-4-Star. Nessa formulação, dividimos cada centro $a(i) \in S$ em $S(i) = \{a_1(i), a_2(i), a_3(i), a_4(i)\}$. Cada um dos quatro vértices de $S(i)$ é conectado apenas a uma das folhas em $A(i)$ e recebe o mesmo rótulo do vértice ao qual está conectado, como é apresentado na Figura 4.4. Neste ponto, o conjunto S e a permutação σ são compostos pelos mesmos elementos. Dessa forma, podemos reformular o OSCM-4-Star como:

(OSCM-4-Star) *One Sided Crossing Minimization 4 Star*

ENTRADA: Um conjunto $S = \cup_{i \in \{1 \dots n\}} A(i)$, onde $A(i) = \{a_1(i), a_2(i), a_3(i), a_4(i)\}$, a permutação σ sob os elementos do conjunto S e um inteiro positivo k' .

PERGUNTA: Existe uma permutação τ dos elementos no conjunto S tal que os elementos em $A(i), i \in \{1 \dots n\}$ são consecutivos e a cardinalidade do conjunto de cruzamento $\{(a_r(i), a_s(j)) | \sigma(a_r(i)) < \sigma(a_s(j)), \tau(a_s(j)) < \tau(a_r(i)), i, j \in \{1 \dots n\}, s, r \in \{1 \dots 4\}\}$ não é maior que k' ?

Agora, introduzimos uma função que toma como entrada uma instância da formulação alternativa do OSCM-4-Star e retorna uma instância do OSWDD. A função de transformação recebe a permutação σ de um conjunto $S = \cup_{i \in \{1 \dots n\}} A(i)$, onde

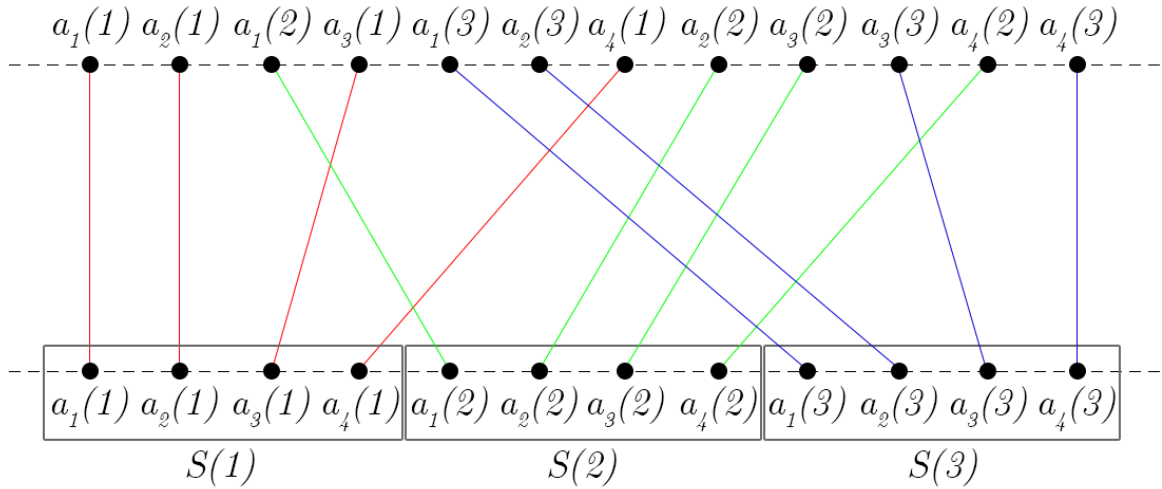


Figura 4.4. Instância OSCM-4-Star da Figure 4.3 com os centros divididos.

$A(i) = \{a_1(i), a_2(i), a_3(i), a_4(i)\}$ e $\sigma(a_1(i)) < \sigma(a_2(i)) < \sigma(a_3(i)) < \sigma(a_4(i))$, e um inteiro k' correspondente a uma instância do OSCM-4-Star. E retorna um DAG G , a permutação \bar{t}_X de V tal que o seu reverso t_X é uma ordenação topológica de G , e um inteiro k , correspondente a uma instância do OSWDD.

A função constrói o DAG $G = (V, \vec{A})$ da seguinte forma. Primeiro, faz $V = S$. Então, $\vec{A} = \cup_{i \in \{1 \dots n\}} \vec{A}(i)$ onde cada conjunto $\vec{A}(i)$ possui 3 arcos, $\vec{A}(i) = \{(a_4(i), a_3(i)), (a_3(i), a_2(i)), (a_2(i), a_1(i))\}$. A Figura 4.5 mostra um grafo G gerado de uma instância do OSCM-4-Star com 3 estrelas. Note que G é um conjunto de caminhos direcionados disjuntos, onde cada um é induzido por um subconjunto $A(i)$ de V .

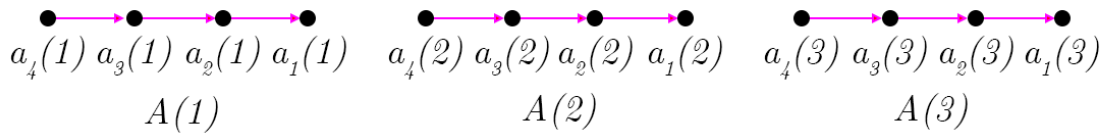


Figura 4.5. Grafo G gerado a partir de uma instância do OSCM-4-Star com 3 estrelas.

A permutação \bar{t}_X sob V é definida como igual a σ e a constante $k = k' + 6n$, onde n é o número de centros.

Note que o reverso de \bar{t}_X , que é t_X , é uma ordenação topológica de G já que cada aresta $(u, v) \in \vec{A}$ implica em $\sigma(u) > \sigma(v)$ que implica em $\bar{t}_X(u) > \bar{t}_X(v)$ que implica em $t_X(u) < t_X(v)$.

Em seguida, mostramos que uma instância do OSCM-4-Star recebida pela função de transformação tem uma solução “sim” se e somente se a instância do OSWDD

retornada pela função tem uma solução “sim”.

Caso a instância do OSCM-4-Star dada para a função tenha uma solução “sim”, então existe uma permutação τ de elementos em S tal que os elementos em $A(i), i \in \{1 \dots n\}$ são consecutivos e o número de cruzamentos entre τ e σ é igual a $q \leq k'$. Então, a permutação τ pode ser transformada para uma ordenação topológica t_Y de G através da inversão da ordem a qual os elementos de cada subconjunto de $A(i)$ aparecem em τ . A cardinalidade de $\bar{I}(t_X, t_Y)$ é $q + 6n \leq k' + 6n = k$. A adição do termo $6n$ vem da inversão da ordem dos elementos de cada subconjunto de $A(i)$, e esta inversão não gera novos cruzamentos entre elementos de diferentes $A(i)$. Como resultado, a instância do OSWDD obtida pela função tem solução “sim”.

Se a instância do OSWDD retornada pela função tem uma solução “sim”, então existe uma ordenação topológica t_Y de G tal que $|\bar{I}(t_X, t_Y)| = q \leq k$. Se os elementos de cada $A(i)$ aparecem consecutivamente em t_Y , então, através da inversão da ordem dos elementos em cada conjunto $A(i)$ é possível obter uma permutação τ na qual os elementos de $A(i)$ aparecem consecutivamente e na qual o número de cruzamentos entre τ e σ é igual a $q - 6n \leq k - 6n = k'$. O que significa que a instância do OSCM-4-Star recebida pela função tem solução “sim”. Se os elementos de cada $A(i)$ não estiverem consecutivos em t_Y , pelo Lema 1 sabemos que existe uma outra ordenação topológica de G com no máximo o mesmo número de inversões entre t_X e t_Y e na qual os elementos de cada $A(i)$ aparecem consecutivos. Dessa forma, o argumento anterior mostra que, também neste caso, existe uma solução “sim” para a instância do OSCM-4-Star recebida pela função.

Por fim, OSWDD está em NP já que é possível verificar que t_Y é uma ordenação topológica de G em $O(|E|)$ ($\forall(a, b) \in E, t_Y(a) < t_Y(b)$), e o número de inversões $\bar{I}(t_X, t_Y)$ pode ser computado em $O(|V|^2)$ comparando cada par de vértices em V . Então, OSWDD é NP-completo. \square

4.6 Simplificação da Prova de NP-Completeness do KNN_{TP}

A prova apresentada na seção anterior utiliza uma transformação polinomial a partir do OSCM-4-Star, que é o mesmo problema utilizado na redução proposta em [Brandenburg et al., 2012] para mostrar a NP-completeness de KNN_{TP} . É importante notar que OSWDD é um caso particular do KNN_{TP} no qual o reverso de σ é uma extensão linear de κ . Por essa razão, toda instância de OSWDD é uma instância válida do KNN_{TP} , mas o contrário não é verdade. Isso significa que a NP-completeness de OSWDD implica

na NP-completude do KNN_{TP} .

É importante mencionar também que a prova do Teorema 3 foi inspirada na prova presente em [Brandenburg et al., 2012]. No entanto, a nossa prova baseada no Lema 1 é significativamente mais simples em comparação à prova anterior, na qual o tamanho do conjunto S é desnecessariamente aumentado e, por isso, vários lemas são necessários para mostrar a sua corretude.

Além disso, o resultado apresentado neste trabalho é mais forte na medida em que em [Brandenburg et al., 2012] é mostrado que o KNN_{TP} é NP-Completo mesmo se a ordem parcial for composta por n caminhos de 12 vértices cada. Nossa prova mostra que é NP-Completo mesmo se a ordem parcial for composta por n caminhos de 4 vértices, e a ordenação parcial restrita ao reverso de uma extensão linear da ordenação parcial.

4.7 Um limite inferior dependente da dimensão para o OSWDD

A seguir, introduzimos um limite inferior (primal) para o OSWDD dependendo da dimensão do DAG G , considerando a versão do OSWDD cujo problema é maximizar $\bar{I}(t_X, t_Y)$. Visto que computar a dimensão de um DAG é NP-completo [Yannakakis, 1982], não há aplicação prática para dimensões $d > 2$. No entanto, se a dimensão do DAG for conhecida, o limite inferior garante a existência de uma solução para o problema com uma certa qualidade.

Seja um DAG $G = (V, A)$ e uma ordenação topológica t_X de G , podemos definir o grafo $H_{G, t_X} = (V, A \cup A')$ com $A' = \{(u, v) \in V \times V \mid \{u, v\} \in Z_G \wedge t_X(u) > t_X(v)\}$, onde $Z_G = \{\{u, v\} \mid u \not\prec v \wedge v \not\prec u\}$. O conjunto de arcos A' contém todos os pares ordenados (u, v) que são incomparáveis em G e invertidos em relação a t_X . Note que $|A'| = |Z_G|$.

Considere o seguinte problema: dado um DAG $G = (V, A)$ e uma ordenação topológica t_X de G , computar o conjunto máximo $D \subseteq A'$ tal que o subgrafo $W = (V, A \cup D)$ de H_{G, t_X} seja acíclico. Este problema é equivalente a resolver o OSWDD, conforme será mostrado a seguir.

Note que não existem pares incomparáveis em W . Para qualquer par incomparável $\{u, v\}$ em G , ou (u, v) ou (v, u) está em A' . Já que D é maximal, qualquer arco de A' não presente em D deveria gerar um ciclo em W se for incluído em D . Dessa forma, existe um caminho em W de u para v ou de v para u , e portanto u e v não

são incomparáveis. Como consequência disse, o grafo W tem uma única ordenação topológica t_Y .

Note também que t_Y é uma ordenação topológica de G . Mais ainda, todo par ordenado (u, v) representado por um arco em D está invertido em t_Y em relação a t_X . Já que D é tão grande quanto possível, a ordenação topológica t_Y maximiza $\bar{I}(t_X, t_Y)$ sobre todas as ordenações topológicas de G . Então, computar D é equivalente a resolver o OSWDD.

Seja d a dimensão de G . Então, pela definição de dimensão de um grafo, existe um conjunto T de d ordenações topológicas $\{t_1, \dots, t_d\}$ de G tal que, para cada par ordenado de vértices incomparáveis (u, v) de G , existe pelo menos uma ordenação topológica $t_i \in T$ tal que $t_i(u) < t_i(v)$.

O Algoritmo 6 calcula a ordenação topológica t_Y de G com $|\bar{I}(t_X, t_Y)| \geq |Z_G|/d$ conforme mostramos abaixo. As linhas 1 a 3 associam um conjunto vazio de arcos D_i a cada uma das d ordenações topológicas em T . Na linha 4, o conjunto A' é definido. Então, nas linhas 5 e 6, para cada $(u, v) \in A'$, obtemos as ordenações topológicas t_i de G nas quais o arco (u, v) é um arco para frente, que vai de um vértice em uma posição menor para outro em uma posição maior em t_i , e adicionamos este arco ao conjunto associado na linha 7.

Note que cada arco $(u, v) \in A'$ é sempre incluído em pelo menos um conjunto devido à definição de T . Visto que todos os arcos em A apontam para frente em $t_i, i \in \{1, \dots, d\}$, qualquer grafo com um conjunto de vértices V e conjunto de arcos $A \cup D_i$ será acíclico. Nas linhas 10, 11, 12 e 13 é selecionado o maior conjunto D_i para tornar-se D , é computado o DAG com os arcos do DAG original G e os pertencentes a D , e retornada uma ordenação topológica deste DAG.

Teorema 4. *Seja t_Y a ordenação topológica retornada pelo Algoritmo 6, então $|\bar{I}(t_X, t_Y)| \geq |Z_G|/d$.*

Demonstração. Já que cada arco $(u, v) \in A'$ está incluído em pelo menos um conjunto D_i , a soma de todos os arcos de cada conjunto é pelo menos $|A'|$ e o conjunto máximo entre todos os subconjunto possui pelo menos $|A'|/d = |Z_G|/d$ arcos. Cada um desses arcos impõe uma ordem entre um par de arcos incomparáveis em G invertidos em relação a t_X para qualquer ordenação topológica computada do DAG. Dessa forma, $|\bar{I}(t_X, t_Y)| \geq |Z_G|/d$. \square

O Algoritmo 6 mostra que é sempre possível obter uma solução para o OSWDD com $|\bar{I}(t_X, t_Y)| \geq |Z_G|/d$. Infelizmente, não é possível computar tal solução com o

Algoritmo 6: Algoritmo para obter a solução para o OSWDD com $\bar{I}(t_X, t_Y) \geq |Z_G|/d$.

Input: DAG G de dimensão d , Ordenação Topológica t_X de G ,
 $T = \{t_1, \dots, t_d\}$ ordenações topológicas de G

Output: Ordenação topológica t_Y de G

```

1 for  $i \leftarrow 1$  to  $d$  do
2   |  $D_i \leftarrow \emptyset$ 
3 end
4  $A' \leftarrow \{(u, v) \mid \{u, v\} \in Z_G \wedge t_X(u) > t_X(v)\}$ 
5 foreach  $(u, v) \in A'$  do
6   | foreach  $t_i \in T$  tal que  $t_i(u) < t_i(v)$  do
7     |  $D_i \leftarrow D_i \cup \{(u, v)\}$ 
8     | end
9 end
10  $D \leftarrow$  maior conjunto  $D_1, \dots, D_d$ 
11  $W \leftarrow (V, A \cup D)$ 
12  $t_Y \leftarrow$  ordenação topológica de  $W$ 
13 return  $t_Y$ 

```

Algoritmo 6 em tempo polinomial, a menos que $P = NP$ já que computar a dimensão d e o conjunto T é NP-difícil [Yannakakis, 1982].

Em contrapartida, é possível checar se $d = 2$ e, em caso afirmativo, computar o conjunto correspondente T em tempo polinomial [Langley, 1995]. Dessa forma, o Algoritmo 6 se torna uma 1/2-aproximação para o OSWDD sempre que G for de dimensão 2. Brandenburg et al. [Brandenburg et al., 2012] apresentam um algoritmo de 2-aproximação para KNN_{TP} . No entanto, note que o problema de minimização KNN_{TP} e o problema de maximização OSWDD são relacionados como problemas de decisão, mas não em termos de aproximação.

Capítulo 5

Nova Abordagem para Consultas de Alcançabilidade em Grafos Grandes

Neste Capítulo, propomos uma nova abordagem para o problema de alcançabilidade em grafos muito grandes. Parte do trabalho referente à construção dessa nova abordagem foi elaborada em colaboração com o Professor Lars Magnus Hvattum, no Molde University College na Noruega, em um período sanduíche durante o curso de Doutorado.

Começamos pela Seção 5.1 que mostra que muito ainda pode ser feito para melhorar as abordagens existentes na literatura para o problema. Na seção seguinte, é proposto o LYNX, que é uma proposta de solução para o problema de alcançabilidade em grafos muito grandes. Em seguida, na Seção 5.3, são apresentados os resultados computacionais obtidos pelo LYNX em comparação com as principais abordagens da literatura e feita uma discussão sobre os resultados alcançados.

5.1 Motivação

No Capítulo 4, formalizamos o problema tratado pela heurística proposta pela FELINE, chamado de Problema de Desenho de Dominância Fraca Unilateral (OSWDD). Neste problema, uma ordenação topológica t_X é fornecida como entrada e resta encontrar uma outra ordenação topológica t_Y que minimize a cardinalidade do conjunto de interseções $I(t_X, t_Y) = \{(u, v) | t_X(u) < t_X(v) \wedge t_Y(u) < t_Y(v)\}$. No mesmo capítulo também foi apresentada uma prova de que a versão de decisão do OSWDD é NP-Completa e um limite inferior para o OSWDD dependente da dimensão do grafo. Parte do estudo apresentado nesta seção está presente no trabalho publicado em [da Silva & Urrutia, 2016].

5.1.1 Comparação entre WDD, OSWDD e *MaximumRank*

A seguir, mostraremos que existe uma grande lacuna entre os resultados ótimos para instâncias do WDD, os resultados ótimos para o OSWDD e os resultados alcançados pela heurística utilizada pela FELINE.

Para ser possível produzir uma comparação entre os problemas, os modelos de Programação Linear Inteira para o WDD e OSWDD apresentados no Capítulo 4 foram implementados com a utilização do IBM CPLEX 12.6 e executados em uma máquina com processador Intel Core i5 de 2.8 GHz e 12 GB de memória RAM. Além disso, para fins de comparação, foi executada a heurística *MaximumRank* proposta em [Kornaropoulos & Tollis, 2012a] utilizada pela FELINE [Velo et al., 2014]. A primeira ordenação topológica foi gerada através da execução de DFS, tanto para o OSWDD, quanto para a heurística *MaximumRank* da FELINE.

Tabela 5.1. Comparação do número de falsos positivos gerados pelas abordagens exatas para o WDD e OSWDD, e a heurística *MaximumRank* utilizada pela FELINE.

Vértices	Arestas	FPs WDD	FPs OSWDD	FPs <i>MaximumRank</i>
10	25	1 (1,0%)	3 (3,0%)	3 (3,0%)
20	62	1 (0,2%)	6 (1,5%)	7 (1,8%)
30	86	1 (0,1%)	12 (1,3%)	20 (2,2%)
40	108	5 (0,3%)	21 (1,3%)	29 (1,8%)
50	132	14 (0,6%)	80 (3,2%)	89 (3,6%)
60	177	22 (0,7%)	222 (6,2%)	331 (9,2%)
70	235	38 (0,7%)	326 (6,7%)	335 (6,8%)
80	258	57 (0,8%)	374 (5,8%)	691 (10,8%)
90	341	64 (0,7%)	562 (6,9%)	1.167 (14,4%)
100	442	91 (0,9%)	660 (6,6%)	1.647 (16,5%)

A Tabela 5.1 apresenta os resultados alcançados durante a execução dos algoritmos usando um grafo direcionado acíclico criado a partir da instância Arxiv¹, que é uma instância de um grafo real que representa um arquivo de impressões eletrônicas de artigos científicos nos campos da matemática, física, ciência da computação, biologia quantitativa, finança quantitativa e estatística, que podem ser acessados via Internet. A instância possui um total de 6.000 vértices e 66.707 arestas. Porém, para esse teste, foram utilizadas instâncias reduzidas que contêm apenas uma fração desse grafo, para que fosse possível executar os algoritmos exatos. Para gerar as instâncias reduzidas,

¹Mais informações em arxiv.org

foram selecionados os primeiros N vértices do grafo e mantidas as arestas originais entre eles.

Além do número absoluto de potenciais falsos positivos (pares de vértices incomparáveis não invertidos nas duas ordenações topológicas), é apresentado também um percentual em relação ao número total de inversões.

Comparando-se as soluções ótimas geradas para o WDD e para OSWDD, é possível observar que, conforme o tamanho da instância aumenta, o número de potenciais falsos-positivos para o WDD é significativamente inferior ao número de potenciais falsos-positivos para o OSWDD. Neste caso particular, para instâncias acima de 60 vértices, o número de falsos positivos é pelo menos 6 vezes menor. Isso ocorre pois para resolver o WDD é possível escolher quaisquer duas ordenações topológicas do grafo enquanto no OSWDD uma das ordenações é fixa.

A heurística gera resultados significativamente distantes do limite mínimo possível, que é o resultado para a abordagem exata implementada no OSWDD. Além disso, essa diferença percentual aumenta com o tamanho das instâncias, isto é, o resultado da heurística piora com o aumento do tamanho das instâncias. Isso mostra que existe um grande espaço para melhorias em termos da qualidade da solução.

Tabela 5.2. Comparação do tempo de execução das abordagens exatas para o WDD e OSWDD

Vértices	Arestas	Tempo WDD (s)	Tempo OSWDD (s)
10	25	1,40	0,12
20	62	1,25	0,25
30	86	1,15	0,44
40	108	16,13	1,11
50	132	80,76	1,77
60	177	131,10	1,71
70	235	449,90	2,86
80	258	3.839,90	5,16
90	341	5.604,30	7,17
100	442	23.167,86	9,92

Na Tabela 5.2 é apresentado o tempo de execução para resolução dos modelos e execução da heurística para as mesmas instâncias anteriores. O tempo de execução do WDD cresce rapidamente com o tamanho da instância enquanto que para o OSWDD o tempo cresce em uma velocidade bastante inferior. Para todas as instâncias computadas, a heurística *MaximumRank* foi executada em menos de 1 milissegundo.

5.1.2 Caso Patológico da Heurística *Maximum-Rank*

A Figura 5.1 apresenta um caso patológico da heurística *Maximum-Rank* utilizada pela FELINE. Seja o DAG $G = (V, E)$ apresentado na Figura 5.1 e uma ordenação topológica $t_X = (1, 2, 3, 4, 5)$ de G , o resultado da execução da heurística é a ordenação topológica $t_Y = (2, 1, 3, 4, 5)$ que introduz a inversão apenas o par $(1, 2)$ em comparação com t_X . No entanto, neste cenário a melhor solução é $t_Y = (1, 3, 4, 2, 5)$ que introduz duas inversões entre os pares $(2, 3)$ e $(2, 4)$.

Este simples exemplo pode ser estendido para mostrar que a razão entre o valor da solução obtida pela heurística e a produzida pela solução ótima pode ser arbitrariamente grande, através da introdução de novos vértices em G aumentando a cadeia entre os vértices 1 e 4, e também introduzindo esses mesmos vértices em t_X entre 2 e 3.

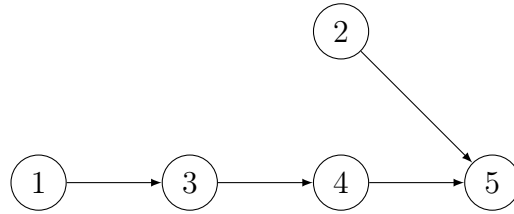


Figura 5.1. Exemplo de caso patológico da heurística *Maximum-Rank*

5.1.3 Eficiência do Índice do HD-GDD

Conforme apresentado no Capítulo 2, a abordagem HD-GDD [Li et al., 2017] introduz uma heurística que estende a abordagem proposta pela FELINE acrescentando mais ordenações topológicas ao seu índice de corte negativo. Com mais ordenações topológicas, o número de inversões é potencialmente maior, produzindo um índice que realiza mais podas na árvore de caminhamento de uma consulta de alcançabilidade.

No entanto, a introdução de novas ordenações topológicas implica em mais comparações ao se avaliar cada vértice na árvore de busca. Então, caso o número de novos cortes negativos acrescentados por uma nova ordenação topológica no índice não seja suficientemente grande, o tempo de consulta pode não ser reduzido ou ainda aumentado, que seria o caso em que o número de cortes não fosse suficiente para compensar o tempo de avaliação do índice para cada vértice.

Um caso simples que mostra claramente quando a introdução de novas ordenações topológicas somente degrada o desempenho da consulta é a criação do índice para um DAG que é uma árvore direcionada. Neste caso, o índice utilizado pela FELINE

com duas ordenações topológicas não terá falsos positivos, e na abordagem HD-GDD novas ordenações topológicas seriam acrescentadas sem produzir nenhum novo corte e aumentando o tempo de consulta.

5.1.4 Oportunidades para Melhoria do IP e BFL

Apesar da eficiência comprovada das abordagens IP [Wei et al., 2018] e BFL [Su et al., 2017], que superam as abordagens anteriores, melhorias consideráveis foram introduzidas somente no índice de corte negativo. O índice de vértices de alto grau do IP pode ser visto como um índice de corte positivo modesto, já que o número de vértices selecionados é em geral relativamente pequeno em comparação com o tamanho do grafo (apenas vértices com grau de saída maior que 100). Isso implica que seu desempenho pode se degradar para consultas positivas.

O BFL é considerado o estado-da-arte para o problema de alcançabilidade em DAGs muito grandes. Ele utiliza um índice de corte positivo similar ao GRAIL, no entanto, com apenas um conjunto de intervalos criados a partir de um único caminho no grafo.

Dessa forma, pouco esforço tem sido feito para se construir um índice que seja adequado para tratar consultas positivas mesmo que elas possam representar um parte significativa de consultas em cenários reais.

5.2 LYNX

Propomos, então, o LYNX (reachabiLitY iNdeX for very large graphs), uma nova abordagem escalável para consultas de alcançabilidade em grafos muito grandes. O LYNX tem por objetivo melhorar os índices propostos nas abordagens anteriores, proporcionando cortes mais eficientes durante o caminhamento no grafo.

O nome LYNX (lince, em português) foi criado durante o período sanduíche na Noruega e representa um animal típico da região sendo também um felino, mostrando sua relação próxima com a FELINE.

O índice negativo é baseado no proposto pela FELINE, no entanto, introduzimos um novo método para melhorar a geração do índice, chamado de estratégia de pulos, que trabalha junto com a heurística *Maximum-Rank* da FELINE. Além disso, é usado um conjunto maior de ordenações topológicas sem degradar o desempenho para cada ordenação introduzida no índice como ocorre no HD-GDD.

A mesma estratégia utilizada no índice de corte negativo é também utilizada no índice de corte positivo para permitir uma família maior de conjuntos de intervalos

sem degradar o desempenho. Com isso, é possível melhorar significativamente a estratégia utilizada no GRAIL, que também é usada extensivamente em quase todas as abordagens posteriores para o problema (e.g. Anand et al. [2013], Veloso et al. [2014], Li et al. [2017], Su et al. [2017]).

Trabalhando junto com as estratégias brevemente descritas acima, o LYNX propõe um abordagem flexível na qual o usuário pode definir o quanto de memória poderá ser usado pelo índice de alcançabilidade e também em qual proporção essa alocação será dividida entre os índices de corte negativo e positivo. Então, o índice pode ser melhorado utilizando o conhecimento do usuário sobre o conjunto de dados e sobre o padrão de consulta.

A seguir, introduziremos o grafo de ordenações topológicas que será útil na definição do índice de corte negativo do LYNX.

5.2.1 Grafo de Ordenações Topológicas

Seja o DAG $G = (V, E)$, podemos definir o grafo de ordenações topológicas de G como o grafo não direcionado $G^{TS} = (V^{TS}, E^{TS})$. O conjunto de vértices V^{TS} representa conjunto de todas as ordenações topológicas de G , ou ainda, $V^{TS} = \{\text{permutação } p \text{ de } V \mid (u, v) \in E \rightarrow p(u) < p(v)\}$. Por sua vez, $E^{TS} = \{(s, t) \mid |\bar{I}(s, t)| = 1\}$, ou seja, existe uma aresta entre as ordenações topológicas s e t se o número de inversões entre elas é igual a 1, isso quer dizer que apenas dois vértices adjacentes estão invertidos nas duas ordenações, que representa um movimento de troca simples entre dois vértices (*jump* ou *swap*).

Esse grafo pode ser muito grande já que um DAG pode possuir um número exponencial de ordenações topológicas. Se tomarmos, por exemplo, um DAG sem arestas, qualquer permutação dos vértices é uma ordenação topológica.

A Figura 5.2 apresenta um exemplo de grafo coroa (*crown graph*) com seis vértices. No grafo de ordenações topológicas correspondente, apresentado na Figura 5.3, as ordenações topológicas $t_1 = (1, 2, 3, 4, 5, 6)$ e $t_2 = (1, 3, 2, 4, 5, 6)$ são adjacentes, pois diferem somente pelo par invertido $(2, 3)$.

O grafo de ordenações topológicas apresentado na Figura 5.3 possui 48 vértices e 96 arestas, 12 vértices com grau 3, 24 vértices com grau 4 e 12 vértices com grau 5. A largura ou diâmetro do grafo (a distância mínima entre os vértices mais distantes) é igual a 8, o que significa que existem pelo menos duas ordenações topológicas onde o número de inversões entre elas é igual a 8.

Seja uma ordenação parcial κ de V representada pelo fecho transitivo de G , ou $tr(G)$. Uma ordenação topológica de G é uma extensão linear de κ e, por sua vez,

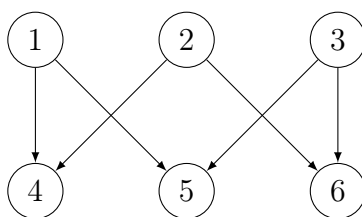


Figura 5.2. Grafo coroa (*crown graph*) com seis vértices.

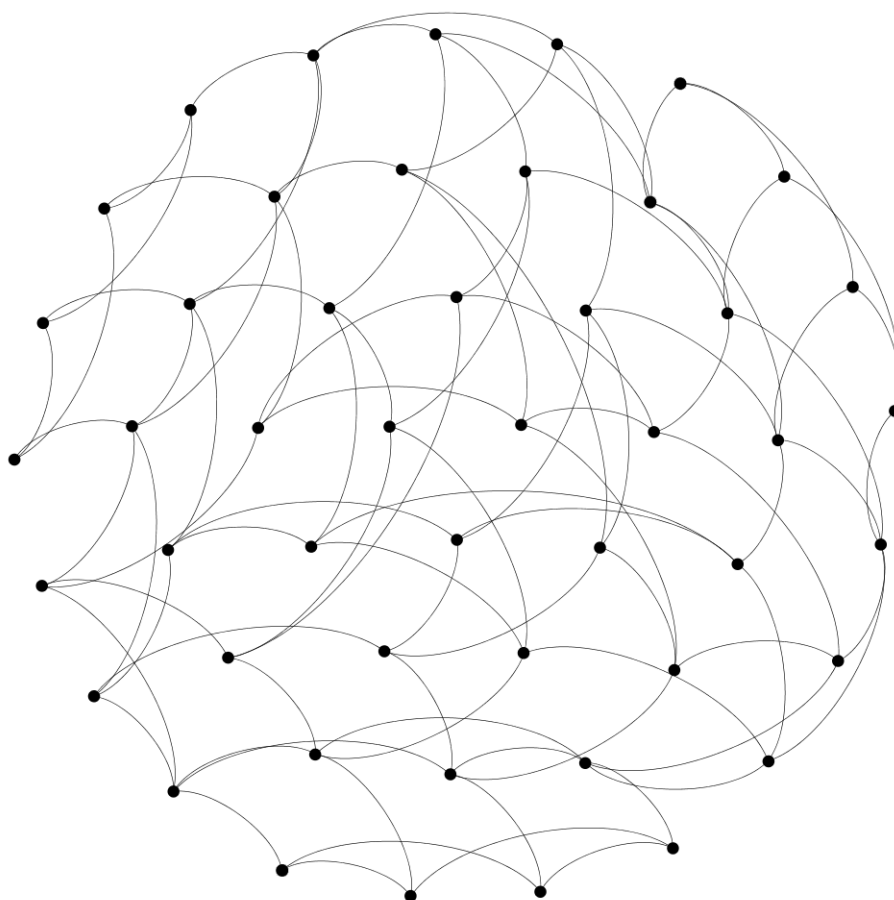


Figura 5.3. Grafo de ordenações topológicas do grafo coroa apresentado na Figura 5.2.

o referente grafo de ordenações topológicas também representa o grafo de extensões lineares de κ .

Do ponto de vista do grafo de ordenações topológicas, o WDD corresponde a encontrar um par diametral (p_1, p_2) em G^{TS} .

Corneil et al. [2002] mostram um algoritmo aproximado para computar o diâmetro

de grafos com ciclos induzidos de tamanho não maiores que k . O algoritmo proposto executa dois procedimentos BFS consecutivos: o primeiro iniciando de um vértice arbitrário e armazenando o último vértice visitado u ; o segundo iniciando de u . Seja v o último vértice visitado no segundo BFS, então o par retornado pelo procedimento é (u, v) . Os autores mostram que a distância entre u e v é no mínimo $diam(G) - \lfloor k/2 \rfloor - 2$.

Dessa forma, o diâmetro de G^{TS} pode ser aproximado pela cota mínima $diam(G^{TS}) - 5$ já que, como mostrado por Massow [2009], o espaço de ciclos de G^{TS} é gerado apenas por 4-ciclos e 6-ciclos. No entanto, executar BFS em G^{TS} é impraticável já que o número de vértice do grafo, i.e. o número de ordenações topológicas de G , pode ser exponencialmente grande.

Executar BFS em G^{TS} para obter um vértice com distância máxima do vértice inicial é equivalente a resolver uma instância do OSWDD, que é NP-difícil, o que foi mostrado no capítulo anterior.

5.2.2 Estratégia de Pulos

Conforme apresentado no Capítulo 4, a heurística *Maximum-Rank* não se propõe a resolver o WDD. Visto que uma das ordenações topológicas é fixada, o problema que abordado pela *Maximum-Rank* pode ser visto como encontrar o vértice mais longe possível de um outro vértice em G^{TS} , o que corresponde a resolver o OSWDD. Independentemente da qualidade da heurística, a qualidade da solução (número de inversões entre t_X e t_Y) é limitado pela excentricidade do vértice t_X em G^{TS} .

Então, propomos um heurística que é baseada na ideia do algoritmo proposto por Corneil et al. [Corneil et al., 2002] que utiliza duas execuções de BFS para obter uma aproximação para pares diametrais em G^{TS} conforme apresentado na Seção 5.2.1. No entanto, como G^{TS} pode ser exponencialmente grande, nós usamos a heurística *Maximum-Rank* em G como uma alternativa ao BFS em G^{TS} .

Iniciamos a estratégia de pulos gerando uma ordenação topológica de G aleatória t_X . Para tanto, usamos um algoritmo similar à heurística *Maximum-Rank*, substituindo a função *maxRank* por uma escolha aleatória das fontes presentes na lista. Note que a ordenação topológica t_X representa um vértice em G^{TS} .

Em seguida, usamos a heurística *Maximum-Rank* para gerar t_Y . Neste ponto temos duas ordenações topológicas, t_X e t_Y , a primeira criada sem nenhum critério de otimização (aleatória) e a segunda gerada com a tentativa de maximizar a distância da primeira em G^{TS} .

Neste ponto, a heurística é executada novamente, usando t_Y como entrada e recebendo t'_X como resultado. Se, ao contrário de utilizar a heurística *Maximum-Rank*,

utilizássemos uma abordagem exata, a distância entre t_X e t_Y seria certamente menor ou igual que a distância entre t_Y e t'_X . No entanto, já que utilizamos uma heurística, não existe garantia com relação a esse resultado, mas, através de experimentos verificamos que a heurística gera soluções de boa qualidade usando a estratégia de pulos. Este processo é ilustrado na Figura 5.4.

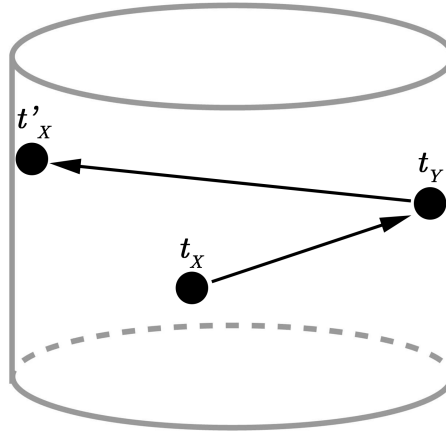


Figura 5.4. Aplicação da estratégia de pulos.

É importante notar que t'_X é sempre um ótimo local considerando a vizinhança de *swap* entre vértices na ordenação topológica. A cada iteração, a heurística *Maximum-Rank* mantém registro de todas as fontes do DAG e a fonte com maior posição em t'_Y é removida do grafo. Então, se u estiver ao lado de v em t'_X tal que $t'_X = (\dots, u, v, \dots)$, então $u \rightsquigarrow v$ ou eles estarão invertidos em t'_Y .

5.2.3 Índice de Corte Negativo

Conforme apresentado no Capítulo 2, dado um DAG $G = (V, E)$, um índice de corte negativo $I_N(G)$ é definido como um subconjunto de todos os pares de vértices $(u, v) \in V \times V$ tal que u não alcança v , ou seja $I_N(G) \subseteq \{(u, v) | \forall u, v \in V, u \not\rightsquigarrow v\}$.

O algoritmo proposto pelo HD-GDD para consulta de alcançabilidade verifica cada ordenação topológica do índice para cada vértice visitado durante a execução do DFS. Conforme ressaltado, essa técnica permite apenas o uso de poucas ordenações topológicas (tipicamente 5). A cada ordenação extra introduzida, o número de corte aumenta, porém, o tempo de verificação em cada vértice cresce, e em certos casos se torna maior que o benefício introduzido pelos cortes extras.

O índice de corte negativo do LYNX armazena um determinado número de ordenações topológicas que são geradas usando a estratégia de pulos, com a qual o par de ordenações topológicas t'_X e t_Y é adicionado ao índice. O primeiro par t'_X e t_Y é gerado através da criação de t_X com um algoritmo DFS como ponto de partida, e os pares seguintes com ordenações topológicas t_X aleatórias como descrito na Seção 5.2.2.

Para cada vértice u , o LYNX armazena um identificador para a ordenação topológica onde u tem a maior posição, definida como th_u , e a ordenação topológica onde o vértice tem a posição menor, definido como tl_u , entre todas as ordenações geradas. Isso pode ser feito em $O(|V|N)$ onde N é o número de ordenações geradas.

Com este procedimento, podemos podar a consulta eficientemente usando o índice proposto verificando somente essas duas ordenações topológicas para cada vértice explorado. Para uma consulta $u \overset{?}{\rightsquigarrow} v$, ou cada vértice intermediário u , se $th_u(u) > th_u(v)$ ou $tl_v(u) > tl_v(v)$, a consulta pode ser podada neste ponto.

Visto que u tem a posição máxima em th_u em comparação com todas as outras ordenações topológicas geradas, mais vértices estão à esquerda de u em th_u e a consulta $u \overset{?}{\rightsquigarrow} v$ pode ser podada u se $th_u(v) < th_u(u)$. Similarmente, tl_v tem grande probabilidade para podar uma consulta como um índice de corte negativo para uma consulta para v , já que v tem posição mínima em tl_v .

A seguir, apresentamos um exemplo de geração do índice de corte negativo para o grafo da Figura 5.2. Para este exemplo, definimos que o índice terá 4 ordenações topológicas. Dessa forma, geraremos 2 pares de ordenações topológicas utilizando a estratégia de pulos, cada par iniciando o processamento usando 2 diferentes permutações aleatórias dos vértices do grafo. São elas: $p_1 = (4, 6, 5, 2, 1, 3)$ e $p_2 = (2, 4, 3, 1, 6, 5)$.

A primeira ordenação topológica aleatória do grafo será gerada usando a heurística *Maximum-Rank*, fornecendo DAG e uma permutação aleatória como entrada. Iniciamos a construção da primeira ordenação topológica t_1 com a permutação p_1 , sendo $S_G = \{1, 2, 3\}$, que são as fontes do grafo. Como 3 é o vértice em S_G que está mais à direita em p_1 , adicionamos 3 em t_1 e o removemos do grafo. Com isso temos $t_1 = (3)$ e agora $S_G = \{1, 2\}$ com a remoção do vértice 3 do grafo. Como 1 é o vértice em S_G que está mais à direita em p_1 , adicionamos 1 em t_1 e removemos 1 do grafo. Esse procedimento é repetido até que $S_G = \emptyset$. No final da execução da heurística, temos $t_1 = (3, 1, 2, 5, 6, 4)$.

A heurística *Maximum-Rank* é aplicada novamente usando t_1 como entrada e obtendo $t_2 = (2, 1, 4, 3, 6, 5)$. Por fim, como estabelecido pela estratégia de pulos, aplicamos mais uma vez a heurística *Maximum-Rank* usando t_2 como entrada e obtendo $t_3 = (3, 1, 5, 2, 6, 4)$ como saída. Por fim, adicionamos t_2 e t_3 ao índice de corte negativo. A Figura 5.5 apresenta essas três ordenações topológicas destacadas no grafo de

ordenações topológicas de G , evidenciando os saltos aplicados na estratégia de pulos.

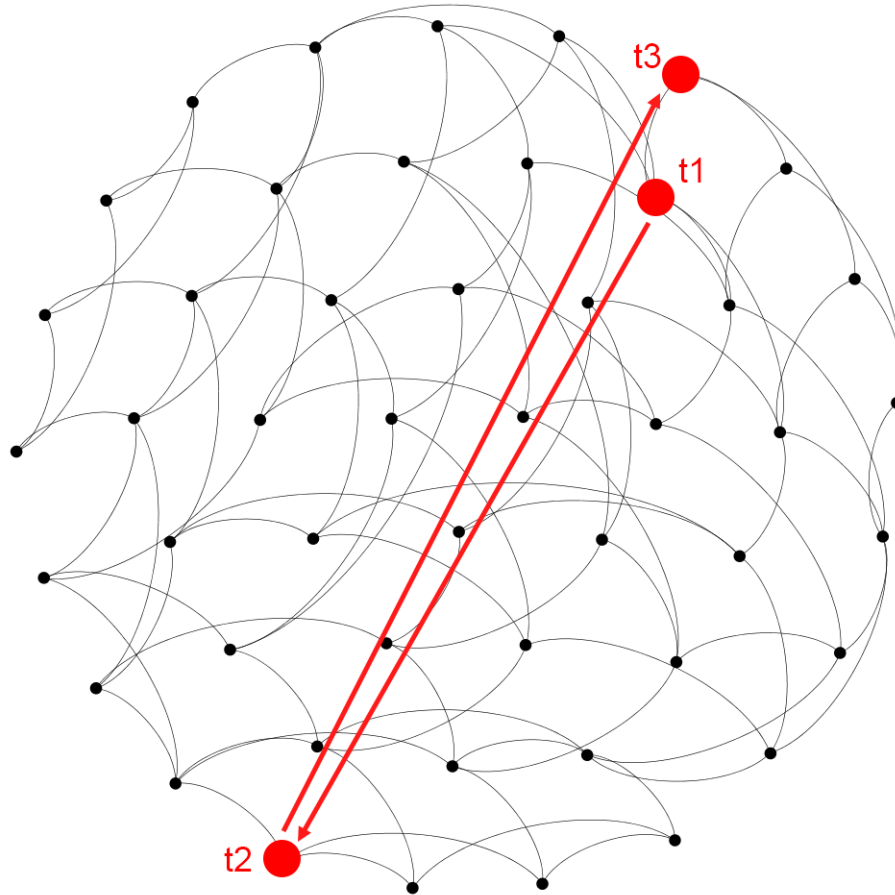


Figura 5.5. Aplicação da estratégia de pulos no grafo de ordenações topológicas do grafo coroa.

Repetindo esse mesmo processo usando a permutação $p_2 = (2, 4, 3, 1, 6, 5)$, temos as ordenações topológicas resultantes $t_4 = (1, 3, 5, 2, 6, 4)$, $t_5 = (2, 3, 6, 1, 4, 5)$ e $t_6 = (1, 3, 5, 2, 4, 6)$, nessa ordem.

Com o descarte das ordenações topológicas t_1 e t_4 , temos o seguinte índice de corte negativo:

- $t_2 = (2, 1, 4, 3, 6, 5)$
- $t_3 = (3, 1, 5, 2, 6, 4)$
- $t_5 = (2, 3, 6, 1, 4, 5)$
- $t_6 = (1, 3, 5, 2, 4, 6)$

Para cada vértice, é necessário definir em qual ordenação ele aparece mais à esquerda (tl) e mais à direita (th). Em caso de empate, mantemos a primeira ordenação topológica encontrada como resultado. A lista abaixo apresenta os resultados para cada vértice:

Vértice 1 $tl_1 = t_6$ e $th_1 = t_5$

Vértice 2 $tl_2 = t_2$ e $th_2 = t_3$

Vértice 3 $tl_3 = t_3$ e $th_3 = t_2$

Vértice 4 $tl_4 = t_2$ e $th_4 = t_3$

Vértice 5 $tl_5 = t_3$ e $th_5 = t_2$

Vértice 6 $tl_6 = t_5$ e $th_6 = t_6$

Para a consulta $1 \overset{?}{\rightsquigarrow} 3$, por exemplo, examinamos as ordenações topológicas onde o vértice 1 aparece mais à direita (th_1) e o vértice 3 aparece mais à esquerda (tl_3), que são respectivamente as ordenações t_5 e t_3 . Como $t_5(3) < t_5(1)$, já podemos responder negativamente à consulta sem mesmo examinar a ordenação t_3 ou o grafo.

5.2.4 Índice de Corte Positivo

Assim como no GRAIL, o índice de corte positivo é composto por P conjuntos de intervalos construídos por caminhamentos DFS em G . A fonte pela qual o DFS é iniciado e a ordem de avaliação dos vértices descendentes é aleatória para todos os conjuntos gerados.

No GRAIL, cada conjunto de intervalos no índice é avaliado para cada vértice explorado durante o caminhamento em uma consulta de alcançabilidade. Com isso, temos um problema similar ao problema do índice de corte negativo do HD-GDD, onde, no GRAIL, um novo conjunto de intervalos introduzido no índice de corte positivo pode degradar o desempenho se o número de cortes adicionais não compensarem a avaliação extra em cada vértice na execução da consulta.

Propomos uma estratégia onde o tamanho do intervalo (i.e. número de vértices dentro de um intervalo ou sucessores) é armazenado para cada vértice durante o caminhamento DFS executado durante sua geração. O Algoritmo 7 mostra como cada conjunto de intervalos é gerado e como é computado o número de sucessores para cada vértice dentro deste intervalo.

Algoritmo 7: Algoritmo para geração de um intervalo para o índice de corte positivo.

```

1 PositiveCutSet(G, Sucessores, IP)
2   Raizes  $\leftarrow \{r \mid (v, r) \notin A(G), \forall v \in V\}$ 
3   for cada vértice u  $\in V$  do
4     Visitado[u]  $\leftarrow$  Falso
5     Sucessores[u]  $\leftarrow$  0
6     tempo  $\leftarrow$  0
7     Embaralha(Raizes)
8     for cada vértice u  $\in$  Raizes do
9        $\lfloor$  PositiveCutSet-Visit(u, G, tempo, IP)
10 PositiveCutSet-Visit(u, G, tempo, IP)
11   Visitado[u]  $\leftarrow$  Verdadeiro
12   tempo  $\leftarrow$  tempo + 1
13   i  $\leftarrow$  tempo
14   Embaralha(Suc(u))
15   for cada vértice v  $\in$  Suc(u) do
16     if Visitado[v] = False then
17        $\lfloor$  PositiveCutSet-Visit(v, G, tempo, IP)
18   Sucessores[u]  $\leftarrow$  tempo - i
19   IP[u]  $\leftarrow$  [i, tempo]
20   tempo ++

```

Com a informação gerada pelo algoritmo apresentado, é possível armazenar para cada vértice u um identificador para o conjunto de intervalos onde o vértice possui o maior número de sucessores entre todos os outros conjuntos, definido como p_u . Como p_u é o intervalo onde u tem mais sucessores, existe a maior chance de que um outro vértice v qualquer esteja entre os seus sucessores entre todos os outros intervalos gerados e, por fim, maior chance de produzir um corte positivo para este vértice.

Durante o caminhamento DFS, para cada consulta de alcançabilidade $u \overset{?}{\rightsquigarrow} v$, para cada vértice intermediário w explorado, se $p_w(v) \subseteq p_w(w)$ então o caminhamento é imediatamente interrompido e a consulta é respondida positivamente.

Com o uso do índice proposto, maximizamos as chances de produzir um corte positivo para cada vértice explorado durante a execução da consulta de alcançabilidade, sem penalizar o tempo de execução com a verificação de vários intervalos para cada vértice.

5.2.5 Consulta de Alcançabilidade

O Algoritmo 8 mostra a função usada pelo LYNX para responder consultas de alcançabilidade. O índice de corte negativo é verificado na linha 4 para podar a consulta se a posição de u for maior que a de v em th_u ou tl_v . Caso contrário, se a consulta não for podada, o índice de corte positivo é usado para responder a consulta positivamente se $p_u(v) \subseteq p_u(u)$. Senão, a consulta continua visitando os vizinhos de u .

Algoritmo 8: Consulta de Alcançabilidade do LYNX

```

1 Reachable( $u, v, G$ )
2   if  $u = v$  then
3     | return True //  $u \rightsquigarrow v$ 
4   if  $th_u(u) > th_u(v) \vee tl_v(u) > tl_v(v)$  then
5     | return False //  $u \not\rightsquigarrow v$ 
6   if  $p_u(v) \subseteq p_u(u)$  then
7     | return True //  $u \rightsquigarrow v$ 
8   for  $s \in Suc(u)$  do
9     | if  $s$  não estiver sido visitado then
10    | | if Reachable( $s, v, G$ ) then
11    | | | return True
12  return False

```

5.2.6 Otimizações Adicionais

Em tempo de pré-processamento, construímos dois vetores booleanos que registram os vértices que são fontes ou sumidouros do grafo. Isso pode ser feito em tempo linear através da verificação de todas as arestas do grafo. Então, este índice pode ser usado antes de examinar os índices de corte negativo e positivo durante a consulta de alcançabilidade. A consulta pode ser podada sempre que $u \neq v$ e u for um sumidouro ou v for uma fonte do grafo.

É importante ressaltar que essa verificação não adiciona nenhum corte negativo durante a pesquisa. No entanto, ele melhora o desempenho da consulta para todos os conjuntos de dados testados pois fontes e sumidouros representam uma porção considerável dos vértices. Além disso, quando essa condição é verificada, a pesquisa pode ser podada sem mesmo checar índices e listas de adjacência.

5.3 Resultados Computacionais

5.3.1 Ambiente Computacional

Os experimentos foram executados em uma máquina com processador Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz, 16 GB de memória principal e sistema operacional Linux Ubuntu 18.04. Os códigos-fonte da FELINE e do BFL foram gentilmente cedidos pelos autores Veloso et al. [Veloso et al., 2014] e Su et al. [Su et al., 2017], respectivamente. Todos os códigos foram escritos em C++ e foram compilados com o gcc versão 8.2.0.

5.3.2 Instâncias de *Benchmark*

O conjunto de instâncias é dividido entre grafos reais e grafos sintéticos. As instâncias de grafos reais são instâncias de *benchmark* propostas na literatura em [Yildirim et al., 2012; Anand et al., 2013; Veloso et al., 2014; Wei et al., 2014], apresentadas na Tabela 5.3.

Começando pelos grafos pequenos, temos a instância *arXiv* (www.arxiv.org) que é um arquivo de impressões eletrônicas de artigos científicos nos campos da matemática, física, ciência da computação, biologia quantitativa, finanças e estatística. O grafo *go* é um subgrafo da instância *go-uniprot*, que será abordado mais à frente. O *pubmed* é um grafo construído a partir do PubMed Central (PMC) que é um arquivo de periódicos biomédicos e de ciências do U.S. National Institutes of Health's National Library of Medicine (NIH/NLM). A *citeseer* é um grafo de uma rede de citações bibliográfica.

Partindo para os grafos grandes, com milhões de vértices, temos a instância *citeseerx* (citeseerx.ist.psu.edu) que, assim como o *citeseer*, é uma rede de citação bibliográfica, na qual o grau de saída de vértices que não são sumidouros está entre 10 e 30. A instância *go-uniprot* é um grafo de termos de Ontologias de Genes com arquivos de anotações do banco de dados universal de proteínas UniProt (www.uniprot.org). Os grafos *uniprotenc22m*, *uniprotenc100m*, e *uniprotenc150m* são subgrafos do grafo RDF completo do UniProt. O *cit-Patents* (snap.stanford.edu) inclui todas as citações de patentes concedidas pelos Estados Unidos entre 1975 e 1999. A instância *govwild* é um grafo RDF construído a partir de govwild.hpi-web.de e transformado para o DAG correspondente. O *yago* é um DAG construído a partir de um RDF de uma base de conhecimento desenvolvida pelo Max Planck Institute (MPI) for Computer Science. O *twitter* é um DAG da rede social Twitter coletado por [Cha et al., 2010]. Por fim, o grafo *web-uk* é um grafo web coletado por [Boldi et al., 2008] no domínio *.uk*.

Tabela 5.3. Instâncias de grafos reais para *benchmark*

Instância	V	E	Fontes	Sumidouros	d_{avg}
arXiv	6.000	66.707	961	624	11,1
go	6.793	13.361	64	3.087	2,0
pubmed	9.000	40.028	2.609	4.702	4,4
citeseer	693.947	312.282	613.497	381.665	0,5
uniprotenc_22m	1.595.444	1.595.442	556.158	2	1,0
cit-Patents	3.774.768	16.518.947	515.785	1.685.423	4,4
citeseerx	6.540.401	15.011.260	567.151	5.740.712	2,3
go_uniprot	6.967.956	34.770.235	6.946.003	286	5,0
govwild	8.022.880	23.652.610	1.302.461	5.189.465	2,9
uniprotenc_100m	16.087.295	16.087.293	14.598.960	2	1,0
yago	16.375.503	25.908.132	3.003.181	13.372.794	1,6
twitter	18.121.168	18.359.487	3.138.961	16.383.480	1,0
web-uk	22.753.644	27.221.332	10.826.445	16.136.119	1,2
uniprotenc_150m	25.037.600	25.037.598	21.650.057	2	1,0

Utilizamos instâncias sintéticas de benchmark para mostrar como a performance em relação ao tempo de consulta é afetada com o crescimento do número de arcos do grafo. Este conjunto de instâncias, apresentado na Tabela 5.4 consiste em 20 DAGs com 10 milhões de vértices com a média de grau de saída dos vértices variando entre 1 e 20. É importante notar que uma instância sintética com mais arcos não é uma extensão de outra com menos arcos.

O algoritmo proposto por [Yildirim et al., 2012] foi usado para gerar essas instâncias sintéticas. Ele começa criando uma permutação aleatória dos vértices. Então, para cada arco a ser adicionado, dois vértices são selecionados aleatoriamente e um arco é criado do vértice mais à esquerda para o mais à direita da permutação gerada. Dessa forma, com os arcos em uma única direção, garante-se que o grafo gerado não conterá ciclos.

5.3.3 Configuração de Parâmetros

Nesta seção, nós discutimos sobre a configuração de parâmetros para o BFL e LYNX para os próximos experimentos. A FELINE não tem parâmetros para serem configurados. Su et al. [2017] apresentam um estudo para encontrar os melhores valores para dois parâmetros do BFL: o tamanho do intervalo s e o número de intervalos d . Eles mostram que $s = 160$ e $d = 10$ são bons valores para esses parâmetros considerando a instância `cit-Patents` como exemplo.

Durante o início da comparação com o LYNX, começamos com esses valores men-

Tabela 5.4. Instâncias de grafos sintéticos para *benchmark*

Instância	V	E	Fontes	Sumidouros	d_{avg}
syn-10M-1	10M	10M	4.323.454	4.324.154	1
syn-10M-2	10M	20M	2.453.746	2.454.584	2
syn-10M-3	10M	30M	1.662.624	1.662.826	3
syn-10M-4	10M	40M	1.250.418	1.248.915	4
syn-10M-5	10M	50M	998.482	999.924	5
syn-10M-6	10M	60M	833.717	833.532	6
syn-10M-7	10M	70M	714.819	714.501	7
syn-10M-8	10M	80M	624.543	625.513	8
syn-10M-9	10M	90M	554.809	555.576	9
syn-10M-10	10M	100M	500.272	499.565	10
syn-10M-11	10M	110M	454.230	455.104	11
syn-10M-12	10M	120M	416.427	416.591	12
syn-10M-13	10M	130M	384.510	383.928	13
syn-10M-14	10M	140M	357.077	356.487	14
syn-10M-15	10M	150M	333.423	333.060	15
syn-10M-16	10M	160M	312.126	311.704	16
syn-10M-17	10M	170M	294.052	293.889	17
syn-10M-18	10M	180M	277.365	277.886	18
syn-10M-19	10M	190M	262.677	263.392	19
syn-10M-20	10M	200M	250.239	249.401	20

cionados, no entanto produzindo apenas resultados modestos para o BFL. Mostramos na Figura 5.6 que se aumentarmos o valor de s , o desempenho geral do BFL também melhora e o tempo de consulta é reduzido para a instância *cit-Patents*. Note que a instância *cit-Patents* é a mesma instância usada como *benchmark* de parâmetros em [Su et al., 2017].

Então, utilizamos $s = 1,280$ para o BFL visto que este é o valor máximo de s que não excede a memória principal total disponível no ambiente de teste para todas as instâncias utilizadas. Nós também observamos que a diferença entre a memória usada pelo BFL e a memória estimada mostrada na saída do algoritmo é muito maior do que o esperado para a representação do grafo na memória, o que sugere que essa medida não é precisa. Por esta razão, nós usamos o uso total de memória como medida para comparar todas as abordagens. Para todos os testes, configuramos a memória máxima do LYNX para a mesma quantidade utilizada pelo BFL (com uma possível pequena variação) para cada instância testada.

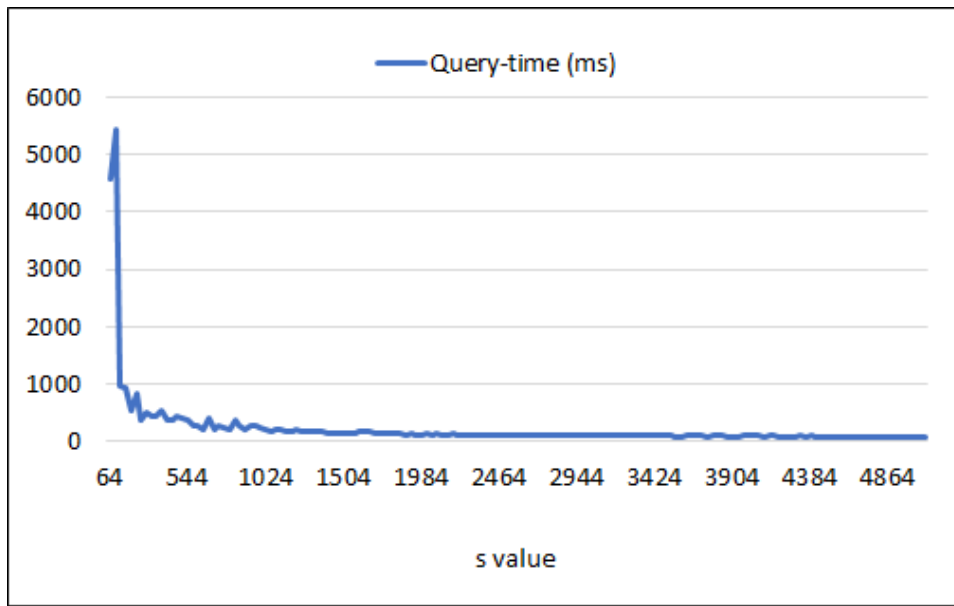


Figura 5.6. Tempo de consulta do BFL em função do valor do parâmetro s

Tabela 5.5. Cortes negativos com a utilização da estratégia de pulos

Instância	Cortes Negativos sem Pulos	Cortes Negativos com Pulos
arXiv	597.761	606.196
go	726.032	753.032
pubmed	731.600	768.131
citeseer	955.619	974.621
uniprotenc_22m	990.420	999.932
cit-Patents	661.565	686.280
citeseerx	800.270	812.599
go_uniprot	997.803	997.904
govwild	911.708	911.769
uniprotenc_100m	967.640	996.989
yago	980.250	980.250
twitter	853.233	854.114
web-uk	847.728	848.030
uniprotenc_150m	951.356	998.352

5.3.4 Eficiência da Estratégia de Pulos

Nesta seção, analisamos a eficiência da estratégia de pulos proposta para o LYNX. Para tanto, modificamos o LYNX para trabalhar semelhantemente à FELINE, utilizando somente duas ordenações topológicas em seu índice. A partir disso, criamos duas versões do código, sendo que em uma das versões incluímos a estratégia de pulos proposta pelo LYNX.

A Tabela 5.5 apresenta o número de cortes negativos no início da consulta para

1 milhão de consultas. Cada corte negativo feito no início da consulta evita percorrer os vértices do grafo. Os resultados mostram que consistentemente a estratégia de pulos aumenta o número de cortes negativos para todos os grafos reais do conjunto de *benchmark*, com exceção da instância *yago* onde o número se manteve mesmo.

Destacamos os resultados para a instância *uniprotenc_150m*, que é a maior do conjunto de testes em número de vértices, para a qual 95,1% das consultas foram resolvidas através do índice construído sem a estratégia pulos, passando para 99,8% das consultas utilizando a estratégia de pulos.

5.3.5 Eficiência do LYNX com a Inclusão de Ordenações Topológicas

Executamos um experimento para verificar o comportamento do LYNX em relação ao tempo de consulta, tempo de geração e tamanho do índice à medida em que novas ordenações topológicas são adicionadas no índice. Seleccionamos a instância *cit-Patents* do conjunto de instâncias reais por se tratar de uma das instâncias mais difíceis do conjunto. As ordenações topológicas foram inseridas aos pares no índice, utilizando a estratégia de pulos.

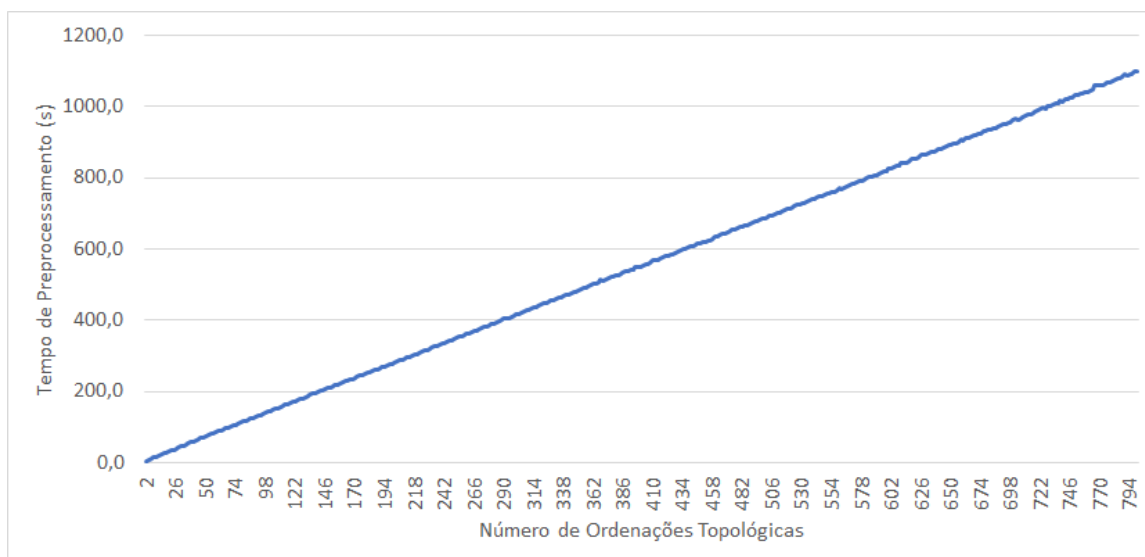


Figura 5.7. Tempo de pré-processamento (s) do LYNX à medida em que são inseridas novas ordenações topológicas

A Figura 5.7 mostra o comportamento do tempo de pré-processamento à medida em que são inseridas ordenações topológicas no índice. Como é possível observar, o tempo é linear em relação ao número de ordenações topológicas inseridas. O tamanho

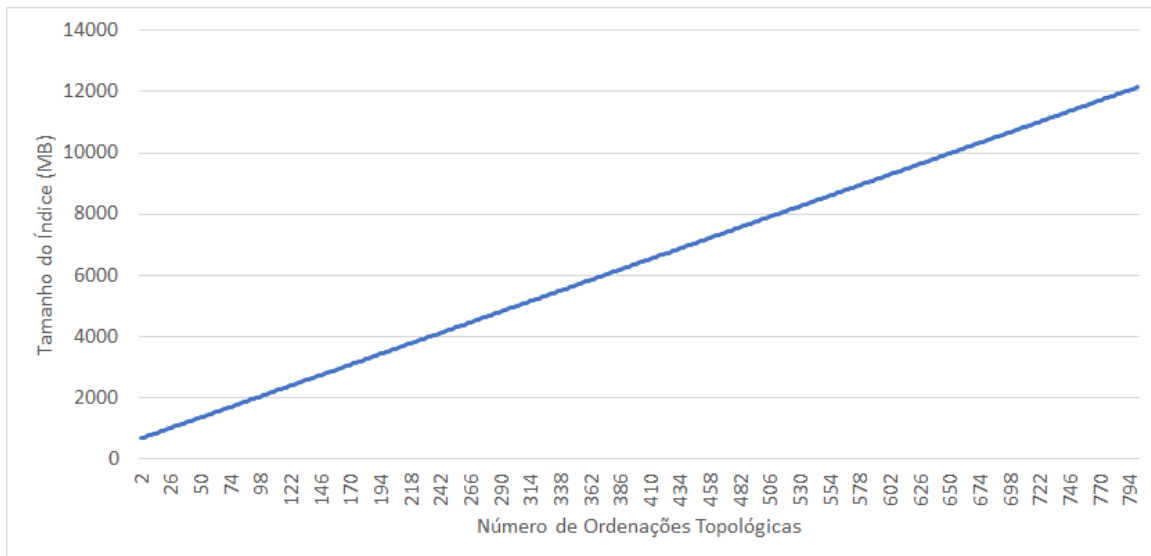


Figura 5.8. Tamanho do índice (MB) do LYNX à medida em que são inseridas novas ordenações topológicas

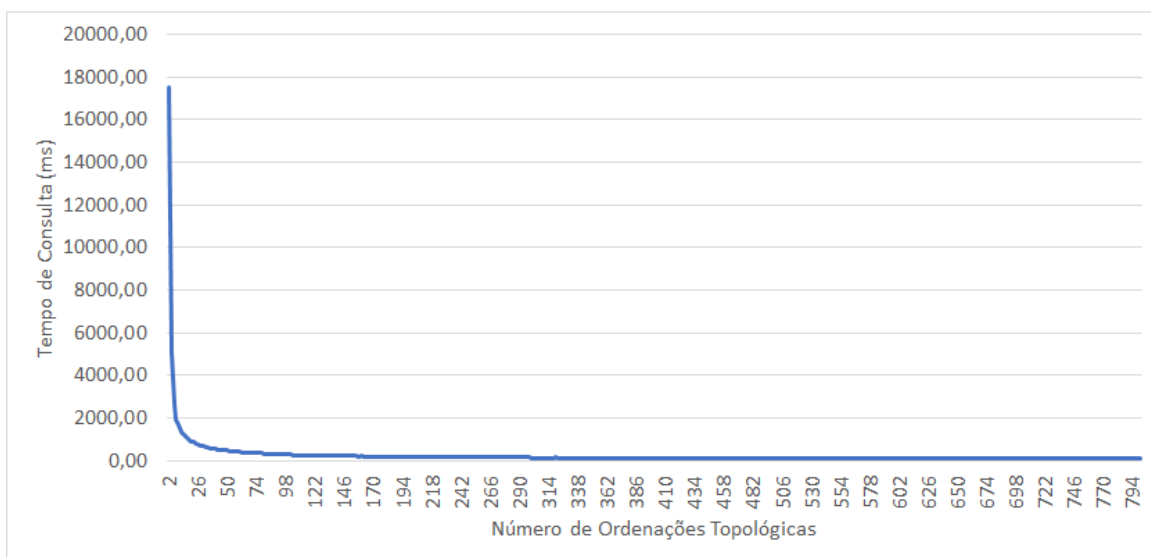


Figura 5.9. Tempo de consulta (ms) do LYNX à medida em que são inseridas novas ordenações topológicas

do índice também aumenta linearmente em relação ao número de ordenações topológicas, como pode ser observado na Figura 5.8.

Por fim, o comportamento do tempo de consulta em relação ao número de ordenações topológicas é apresentado na Figura 5.9. Os resultados totais para o tempo de consulta, tamanho do índice e tempo de pré-processamento também são apresentados de forma resumida na Tabela 5.6. Ao contrário do HD-GDD, à medida em que novas ordenações topológicas são introduzidas no índice do LYNX, o tempo de consulta di-

Tabela 5.6. Tempo de consulta (ms) do LYNX à medida em que são inseridas novas ordenações topológicas

Ordenações Topológicas	Índice (MB)	Preprocessamento (s)	Tempo de Consulta (ms)
2	656	5,1	17491,40
4	685	8,0	5070,33
6	713	10,5	2636,61
8	742	13,2	1963,98
10	771	16,5	1676,95
20	915	30,2	930,94
30	1059	45,0	698,12
40	1203	58,4	547,51
50	1347	73,4	504,35
60	1491	88,8	429,32
70	1635	101,7	371,34
80	1779	115,3	341,85
90	1923	128,5	315,88
100	2067	142,4	289,04
150	2787	210,5	225,85
200	3507	277,6	185,34
250	4227	346,6	164,11
300	4947	415,5	151,15
350	5667	483,2	140,71
400	6387	550,7	129,42
450	7107	617,3	121,45
500	7827	686,7	115,50
550	8547	755,2	111,70
600	9267	825,2	106,28
650	9987	892,2	102,60
700	10707	960,8	98,62
750	11427	1033,0	97,18
800	12147	1100,3	94,22

minui passando de 17 segundos para duas ordenações, para 94,22 milissegundos para 800 ordenações. No entanto, para 800 ordenações, o tamanho do índice cresce para 12 GB em memória e o tempo necessário para a sua geração passa para 18 minutos.

5.3.6 Resultados para Grafos Reais

A Tabela 5.9 apresenta o tempo de preprocessamento da FELINE, BFL e LYNX para instâncias de grafos reais. Note que LYNX precisa de mais tempo que a FELINE e BFL para criar o seu índice devido ao maior tamanho do índice em comparação com a FELINE e a algoritmos mais sofisticados envolvidos. No entanto, o tempo ainda é razoável considerando grafos com milhões de vértices e arcos que irão ser preprocessados uma única vez e consultados repetidamente.

O uso de memória da FELINE, BFL e LYNX é apresentado na Tabela 5.10. Visto que FELINE não possui nenhum parâmetro para ser configurado, o tamanho do índice é fixo e utiliza apenas duas ordenações topológicas com o índice de corte negativo. Para o BFL e FELINE, o tamanho do índice aumenta com o aumento do número de

Tabela 5.7. Tempo de execução (ms) para 1 milhão de consultas aleatórias em instâncias de grafos reais

Instância	R-ratio	FELINE	BFL	LYNX	$\frac{LYNX}{BFL}$
arXiv	15,517%	508,89	173,94	71,16	40,91%
go	0,241%	74,84	38,67	17,26	44,64%
pubmed	0,644%	51,94	34,03	18,27	53,69%
citeseer	0,000%	44,69	23,46	14,97	63,80%
uniprotenc_22m	0,000%	28,21	22,48	11,47	51,03%
cit-Patents	0,042%	3.688,13	65,99	307,63	466,18%
citeseerx	0,234%	95,10	29,11	32,74	112,46%
go_uniprot	0,001%	27,34	21,76	17,55	80,65%
govwild	0,008%	82,56	27,88	38,14	136,80%
uniprotenc_100m	0,000%	27,87	25,51	27,26	106,86%
yago	0,000%	29,78	24,33	24,95	102,57%
twitter	7,375%	49,67	28,74	33,37	116,11%
web-uk	8,982%	158,89	49,18	52,69	107,14%
uniprotenc_150m	0,000%	28,15	27,21	30,60	112,44%
		(soma) 4.896,06	(soma) 592,29	(soma) 698,06	(média geom.) 91,38%

Tabela 5.8. Tempo de execução (ms) para 1 milhão de consultas balanceadas, 500 mil positivas e 500 mil negativas, em instâncias de grafos reais

Instância	FELINE	BFL	LYNX	$\frac{LYNX}{BFL}$
arXiv	757,66	292,93	129,79	44,31%
go	98,68	87,59	39,08	44,62%
pubmed	225,91	153,68	85,83	55,85%
citeseer	129,63	119,31	105,76	88,64%
uniprotenc_22m	50,03	43,72	59,58	136,29%
cit-Patents	20.299,93	1.079,06	2.702,66	250,47%
citeseerx	1.291,46	552,37	472,83	85,60%
go_uniprot	255,91	278,81	131,99	47,34%
govwild	387,09	190,04	166,67	87,70%
uniprotenc_100m	109,93	90,74	87,47	96,40%
yago	121,44	77,06	108,46	140,76%
twitter	121,25	73,00	146,21	200,30%
web-uk	243,93	194,30	152,17	78,32%
uniprotenc_150m	121,68	103,07	93,54	90,75%
	(soma) 24.214,53	(soma) 3.335,65	(soma) 4.482,04	(média geom.) 90,02%

Tabela 5.9. Tempo de pré-processamento (s) em instâncias de grafos reais

Instance	FELINE	BFL	LYNX
arXiv	0,00	0,00	14,10
go	0,00	0,00	10,91
pubmed	0,00	0,00	13,15
citeseer	0,23	0,09	41,92
uniprotenc_22m	0,44	0,15	64,87
cit-Patents	3,62	2,23	152,86
citeseerx	3,12	1,55	167,01
go_uniprot	3,01	1,60	216,85
govwild	3,46	1,76	212,78
uniprotenc_100m	5,32	1,72	425,83
yago	8,69	2,45	458,98
twitter	4,55	1,75	435,59
web-uk	6,09	2,50	520,04
uniprotenc_150m	9,22	2,93	673,38

Tabela 5.10. Utilização de memória (MB) em instâncias de grafos reais

Instância	FELINE	BFL	LYNX
arXiv	49	700	640
go	48	698	640
pubmed	49	700	640
citeseer	172	1.007	1.004
uniprotenc_22m	375	1.396	1.396
cit-Patents	903	2.548	2.564
citeseerx	1.356	3.714	3.732
go_uniprot	1.559	4.101	4.075
govwild	1.740	4.480	4.464
uniprotenc_100m	3.127	7.797	7.754
yago	3.222	8.053	8.085
twitter	3.427	8.616	8.745
web-uk	4.449	10.922	10.909
uniprotenc_150m	4.849	11.776	11.856

Tabela 5.11. Tempo de processamento (s) para instâncias de grafos sintéticos

Instância	FELINE	BFL	LYNX
syn-10M-1	7,58	3,22	314,00
syn-10M-2	10,05	4,91	376,19
syn-10M-3	11,23	6,27	405,51
syn-10M-4	12,11	7,61	426,92
syn-10M-5	12,95	8,93	447,52
syn-10M-6	13,72	10,17	472,29
syn-10M-7	14,43	11,43	492,01
syn-10M-8	15,11	12,39	516,34
syn-10M-9	15,71	13,66	539,03
syn-10M-10	16,49	14,80	566,45
syn-10M-11	17,35	16,24	606,11
syn-10M-12	18,06	17,03	637,76
syn-10M-13	18,58	18,10	669,48
syn-10M-14	19,40	19,20	708,53
syn-10M-15	20,25	20,29	731,35
syn-10M-16	20,70	21,33	762,31
syn-10M-17	21,32	22,65	817,96
syn-10M-18	22,00	23,40	846,62
syn-10M-19	22,68	24,55	881,66
syn-10M-20	23,45	25,54	904,65

vértices e arcos do grafo. Limitamos o uso de memória do LYNX ao mesmo utilizado pelo BFL para cada instância testada com possivelmente uma pequena variação.

A Tabela 5.7 apresenta o tempo médio de consulta para 10 execuções de um milhão de consultas aleatórias. BFL e LYNX superam a FELINE para todas as instâncias testadas. Os resultados obtidos pelo LYNX são melhores em média que os obtidos pelo BFL. No entanto, o BFL alcançou um tempo menor de consulta para 6 das 14 instâncias testadas. É importante ressaltar que para a maioria das instâncias o número de consultas positivas entre as consultas aleatórias (R-ratio) é menor que 1%.

Na Tabela 5.8 são apresentados os resultados médios para 10 execuções de consultas balanceadas, isto é, 500 mil consultas negativas e 500 mil consultas positivas. Com esse teste pretendemos analisar o desempenho do LYNX em comparação com o BFL e FELINE em um cenário mais realista com um conjunto maior de consultas positivas. O LYNX é melhor nos resultados em média novamente. O LYNX tem melhores resultados para 10 das 14 instâncias.

Tabela 5.12. Utilização de memória (MB) para instâncias de grafos sintéticos

Instância	FELINE	BFL	LYNX
syn-10M-1	1.947	5.090	4.815
syn-10M-2	2.095	5.240	5.031
syn-10M-3	2.207	5.372	5.258
syn-10M-4	2.311	5.522	5.415
syn-10M-5	2.417	5.683	5.583
syn-10M-6	2.526	5.852	5.827
syn-10M-7	2.635	6.027	5.994
syn-10M-8	2.753	6.208	6.241
syn-10M-9	2.878	6.392	6.422
syn-10M-10	2.994	6.577	6.604
syn-10M-11	3.114	6.760	6.859
syn-10M-12	3.224	6.936	7.107
syn-10M-13	3.328	7.108	7.275
syn-10M-14	3.441	7.284	7.519
syn-10M-15	3.559	7.463	7.687
syn-10M-16	3.685	7.649	7.811
syn-10M-17	3.813	7.843	8.127
syn-10M-18	3.948	8.043	8.321
syn-10M-19	4.083	8.241	8.590
syn-10M-20	4.211	8.440	8.703

Tabela 5.13. Tempo de execução (ms) para 1 milhão de consultas aleatórias para instâncias de grafos sintéticos

Instância	R-ratio	FELINE	BFL	LYNX	$\frac{LYNX}{BFL}$
syn-10M-1	0,00%	80,07	34,27	37,04	108,07%
syn-10M-2	0,00%	283,27	44,45	52,57	118,26%
syn-10M-3	0,00%	990,78	47,28	110,10	232,86%
syn-10M-4	0,00%	4.143,29	60,28	359,48	596,39%
syn-10M-5	0,01%	19.189,83	223,53	1.513,31	677,00%
syn-10M-6	0,12%	88.920,91	1.480,11	6.729,79	454,68%
syn-10M-7	0,57%	371.138,62	9.023,79	28.471,64	315,52%
syn-10M-8	1,66%	1.011.163,88	33.194,64	78.998,78	237,99%
syn-10M-9	3,47%	2.039.423,75	86.928,96	164.553,57	189,30%
syn-10M-10	5,56%	3.094.120,25	171.151,09	255.431,69	149,24%
syn-10M-11	7,80%	4.143.705,25	284.854,66	346.045,91	121,48%
syn-10M-12	9,97%	5.044.193,00	437.610,60	432.520,21	98,84%
syn-10M-13	12,05%	5.816.364,50	607.268,01	508.844,34	83,79%
syn-10M-14	14,00%	6.793.176,50	811.775,72	562.137,75	69,25%
syn-10M-15	15,80%	7.713.948,50	1.019.217,07	616.853,78	60,52%
syn-10M-16	17,43%	8.062.516,00	1.235.536,72	661.500,98	53,54%
syn-10M-17	19,06%	8.315.207,00	1.478.750,13	731.334,65	49,46%
syn-10M-18	20,39%	9.561.146,00	1.698.724,03	772.635,99	45,48%
syn-10M-19	21,67%	9.918.577,00	1.925.852,27	772.956,71	40,14%
syn-10M-20	22,97%	9.804.729,00	2.185.029,81	856.852,79	39,21%

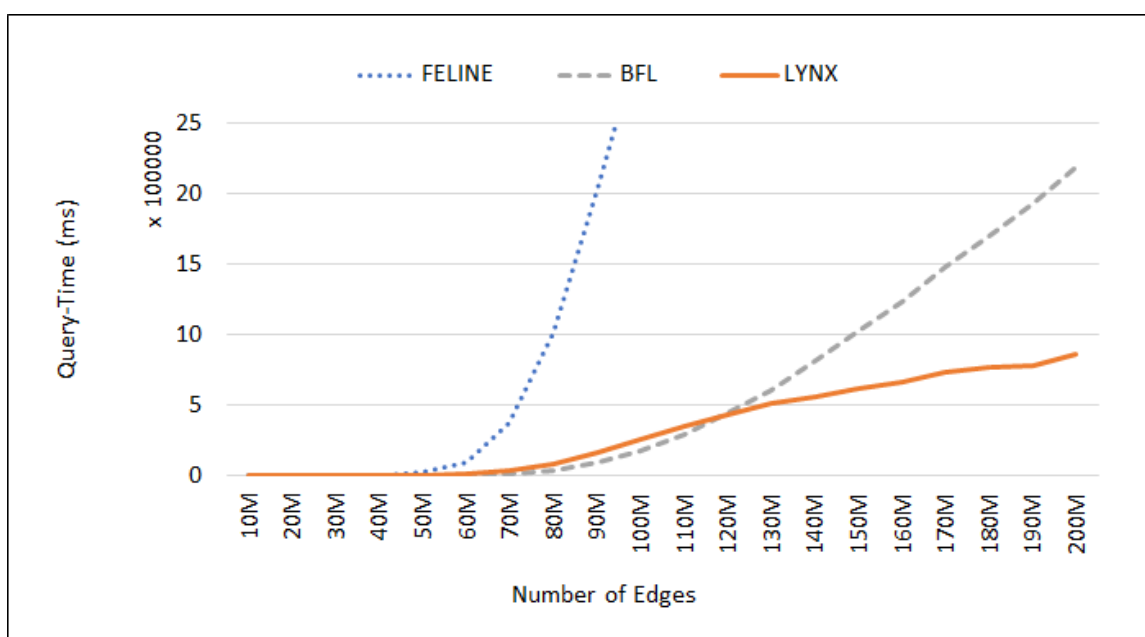


Figura 5.10. Comparação entre FELINE, BFL e LYNX em relação ao tempo de consulta com o crescimento do número de arcos e R-ratio.

5.3.7 Resultados para Grafos Sintéticos

A Tabela 5.11 apresenta o tempo de pré-processamento para as instâncias sintéticas com 10 milhões de vértices e média do grau dos vértices entre 1 e 20. O tempo de pré-processamento do LYNX é maior que a FELINE e BFL. No entanto, o tempo cresce lentamente com o crescimento do número de arcos. De fato, enquanto o BFL aumenta o tempo de processamento de 3,22 segundos para a menor instâncias para 25,54 para a maior instância (um aumento de 7,9 vezes), o tempo de pré-processamento do LYNX aumenta de 314,0 para 904,65 segundos para as mesmas instâncias (um aumento de 2,9 vezes).

O uso de memória para as instâncias sintéticas é apresentado na Tabela 5.12. Novamente, FELINE utiliza menos memória que BFL e LYNX. A memória necessária pelo BFL aumenta com o número de arcos para um s fixo. A memória do LYNX é limitado pelo utilizado pelo BFL para cada instância.

A Tabela 5.13 apresenta o tempo de execução para 1 milhão de consultas aleatórias. O tempo de consulta da FELINE cresce mais rápido que o BFL e LYNX, o que pode ser observado graficamente na Figura 5.10. Para instâncias de grafo com grau de vértice médio até 11, com um pequeno R-ratio (número de consultas positivas entre as aleatórias), BFL supera o LYNX. No entanto, quando o número de arcos cresce e também o R-ratio, o LYNX supera claramente o BFL e a diferença fica cada vez maior.

Para a instância `syn-10-20`, com R-ratio de 22.97%, LYNX roda em menos de 40% do tempo utilizado pelo BFL.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho, avançamos na teoria e prática do problema de alcançabilidade em grafos muito grandes. Mostramos que a representação de índices por um conjunto de intervalos tem a mesma expressividade que a representação utilizando duas ordenações topológicas. Além disso, apresentamos algoritmos que podem realizar a transformação de uma representação para a outra.

Propomos a formalização do Problema de Dominância Fraca Unilateral (OSWDD), utilizado por abordagens escaláveis para a geração de índices que auxiliam na resposta de consultas de alcançabilidade. Mostramos que o OSWDD é uma especialização do WDD e que o número de inversões resultado de uma solução do OSWDD depende da ordenação topológica fornecida como entrada. Concluímos sobre a NP-Completeness do OSWDD e, para isso, provamos um Lema em permutações gerais que pode ser útil em outros contextos. A mesma é ainda uma prova mais simples sobre a NP-Completeness do *nearest neighbor Kendall tau* em comparação com a prova encontrada na literatura [Brandenburg et al., 2012].

Apresentamos o LYNX, que é uma nova abordagem para o problema de alcançabilidade em grafos grandes e mostramos sua eficiência comparando resultados computacionais com a literatura, com um destaque para resultados positivos a favor do LYNX para grafos com alto R-ratio.

O LYNX permite ao usuário um controle sobre a quantidade de memória disponível para o seu índice, inclusive sobre a proporção do índice em relação a cortes positivos e negativos. Com isso, caso o usuário tenha algum conhecimento antecipado sobre o conjunto de dados a ser consultado, isso pode ser usado em favor do algoritmo, resultado em maior eficiência e um tempo de consulta menor.

Por fim, o LYNX mostrou-se uma abordagem robusta para os grafos sintéticos utilizados nos experimentos, mantendo os bons resultados para grafos com grande número

de vértices e principalmente arcos. A introdução de arcos aumenta potencialmente o R-ratio do grafo, e nos casos de maior R-ratio o LYNX supera o estado-da-arte na literatura.

6.1 Trabalhos Futuros

6.1.1 Combinar BFL com LYNX

Neste trabalho, mostramos que o LYNX supera o BFL para grafos com alto índice de alcançabilidade. Para grafos com baixo índice de alcançabilidade, os resultados do LYNX e BFL ficam muito próximos para a maioria das instâncias testadas. Porém, para certas instâncias, o BFL supera o LYNX.

Portanto, pode ser interessante avaliar para quais grafos o BFL se comporta melhor que o LYNX e decidir, antes da geração do índice, qual a melhor abordagem para o grafo sendo avaliado. Isso possibilitaria combinar as duas abordagens e utilizar cada uma na situação em que se comporta melhor.

6.1.2 Instâncias Reais com Maior Número de Vértices e Arcos

Existe uma carência latente por instâncias reais de DAGs com grande número de vértices e arcos. O perfil das instâncias reais atuais, extremamente esparsas e com R-ratio muito baixo, pode favorecer certas abordagens e não retratam muito bem a finalidade do algoritmo que é ter bom desempenho quando o número de vértices e arcos cresce.

Dessa forma, uma linha importante a ser seguida nos próximos passos deste trabalho é descobrir e formatar instâncias reais de DAGs maiores que as atuais, o que representa um grande desafio, porém seus resultados podem gerar um grande benefício para esta área de pesquisa.

6.1.3 Solução Escalável Combinada com SCARAB

Assim como outras abordagens para o problema, o LYNX também pode ser combinado com *frameworks* como o SCARAB [Jin et al., 2012] e Interval-Index [Li et al., 2015] que permitem reduzir o tamanho do grafo a ser consultado criando um espinha dorsal que contém informações topológicas suficientes do grafo para permitir consultas de alcançabilidade. Com o uso de *frameworks* deste tipo pode ser possível escalar o algoritmo para outro patamar, para grafos com bilhões de vértices e arestas. Atualmente, as abordagens presentes na literatura para o problema de alcançabilidade propõem soluções para grafos com milhões de vértices, então tratar grafos na escala de bilhões de

vértices ainda é um tópico em aberto e que pode ser explorado na continuidade deste trabalho.

6.1.4 Algoritmo Aproximativo para o OSWDD

As heurísticas propostas para o OSWDD, apesar de eficientes na prática, não trazem nenhuma garantia teórica quanto ao seu resultado final. Uma linha para o prosseguimento desta pesquisa é propor algoritmos aproximativos para o problema, que forneçam garantias quanto ao número mínimo de inversões entre a ordenação topológica fornecida como entrada e a recebida como resultado do seu processamento.

6.1.5 Aplicação de Metaheurísticas para o WDD e OSWDD

Conforme apresentado nos resultados obtidos, a LYNX permite gerar um índice de boa qualidade em comparação com as outras abordagens presentes na literatura. Apesar disso, muito ainda pode ser melhorado na etapa de construção dos índices de alcançabilidade. Em sistemas reais seria factível gerarmos um índice inicial e, em seguida, em paralelo às consultas, possivelmente em outro ambiente computacional, realizar iterações adicionais para melhorar o índice gerado. Uma vez obtido um índice mais eficiente, o anterior pode ser substituído e as consultas podem ser respondidas mais rapidamente.

Dessa forma, outra proposta de continuidade deste trabalho é produzir uma heurística iterativa que proporcione melhorias contínuas no índice que possam ser executadas a longo prazo. Neste caso, heurísticas de memória adaptativa [Glover & Laguna, 1997; Glover, 2005] aparecem como uma opção interessante já que permitiriam “aprender” quais pares de vértices são mais importantes para serem invertidos nas ordenações topológicas.

Referências Bibliográficas

- Agrawal, R.; Borgida, A. & Jagadish, H. V. (1989). Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Rec.*, 18(2):253--262. ISSN 0163-5808.
- Anand, A.; Seufert, S.; Bedathur, S. & Weikum, G. (2013). Ferrari: Flexible and efficient reachability range assignment for graph indexing. Em *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pp. 1009--1020, Washington, DC, USA. IEEE Computer Society.
- Boldi, P.; Santini, M. & Vigna, S. (2008). A large time-aware web graph. *SIGIR Forum*, 42(2):33--38. ISSN 0163-5840.
- Brandenburg, F. J.; Gleißner, A. & Hofmeier, A. (2012). Comparing and aggregating partial orders with kendall tau distances. Em *WALCOM: Algorithms and Computation: 6th International Workshop, WALCOM 2012, Dhaka, Bangladesh, February 15-17, 2012. Proceedings*, pp. 88--99, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Brightwell, G. R. & Massow, M. (2013). Diametral pairs of linear extensions. *SIAM J. Discrete Math.*, 27(2):634--649.
- Broder, A. (1997). On the resemblance and containment of documents. Em *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pp. 21--, Washington, DC, USA. IEEE Computer Society.
- Cha, M.; Haddadi, H.; Benevenuto, F. & Gummadi, K. P. (2010). Measuring user influence in twitter: The million follower fallacy. Em *in ICWSM 10: Proceedings of international AAAI Conference on Weblogs and Social*.
- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edição. ISBN 0262033844, 9780262033848.

- Corneil, D. G.; Dragan, F. F. & Köhler, E. (2002). On the power of bfs to determine a graphs diameter. Em Rajsbaum, S., editor, *LATIN 2002: Theoretical Informatics*, pp. 209--223, Berlin, Heidelberg. Springer Berlin Heidelberg.
- da Silva, R. F. & Urrutia, S. (2016). Um estudo sobre a aplicação de ordenações topológicas no problema de alcançabilidade em grafos grandes. Em *Anais do XLVIII Simpósio Brasileiro de Pesquisa Operacional*, Vitória, ES.
- da Silva, R. F.; Urrutia, S. & dos Santos, V. F. (2018). One-sided weak dominance drawing. *Theoretical Computer Science*. ISSN 0304-3975.
- Eades, P. & Whitesides, S. (1994). Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361 – 374. ISSN 0304-3975.
- Glover, F. (2005). *Adaptive Memory Projection Methods for Integer Programming*, pp. 425--440. Springer US, Boston, MA.
- Glover, F. & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publisher, Boston, Dordrecht, London.
- Jin, R.; Ruan, N.; Dey, S. & Xu, J. Y. (2012). Scarab: Scaling reachability computation on large graphs. Em *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pp. 169--180, New York, NY, USA. ACM.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Semi-numerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-89684-2.
- Kornaropoulos, E. M. (2012). Dominance Drawing of Non-Planar Graphs. Dissertação de mestrado, University of Crete, Heraklion, Greece.
- Kornaropoulos, E. M. & Tollis, I. G. (2011). Weak dominance drawings and linear extension diameter. *CoRR*, abs/1108.1439.
- Kornaropoulos, E. M. & Tollis, I. G. (2012a). Overloaded orthogonal drawings. Em *Proceedings of the 19th International Conference on Graph Drawing*, GD'11, pp. 242--253, Berlin, Heidelberg. Springer-Verlag.
- Kornaropoulos, E. M. & Tollis, I. G. (2012b). Weak dominance drawings for directed acyclic graphs. Em *Graph Drawing - 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, pp. 559--560.

- Langley, L. J. (1995). Recognition of orders of interval dimension two. *Discrete Applied Mathematics*, 60:257--266.
- Li, F.; Yuan, P. & Jin, H. (2015). *Interval-Index: A Scalable and Fast Approach for Reachability Queries in Large Graphs*, pp. 224--235. Springer International Publishing, Cham.
- Li, L.; Hua, W. & Zhou, X. (2017). Hd-gdd: high dimensional graph dominance drawing approach for reachability query. *World Wide Web*, 20(4):677--696. ISSN 1573-1413.
- Massow, M. (2009). *Linear Extension Graphs and Linear Extension Diameter*. Tese de doutorado, TU Berlin, Berlin.
- Muñoz, X.; Unger, W. & Vrt' o, I. (2002). *One Sided Crossing Minimization Is NP-Hard for Sparse Graphs*, pp. 115--123. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Su, J.; Zhu, Q.; Wei, H. & Yu, J. X. (2017). Reachability querying: Can it be even faster? *IEEE Transactions on Knowledge and Data Engineering*, 29(3):683--697.
- Tarjan, R. (1972). Depth first search and linear graph algorithms. *SIAM JOURNAL ON COMPUTING*, 1(2).
- Veloso, R. R.; Cerf, L.; Meira Jr., W. & Zaki, M. J. (2014). Reachability queries in very large graphs: A fast refined online search approach. Em *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pp. 511--522.
- Wei, H.; Yu, J. X.; Lu, C. & Jin, R. (2014). Reachability querying: An independent permutation labeling approach. *Proc. VLDB Endow.*, 7(12):1191--1202. ISSN 2150-8097.
- Wei, H.; Yu, J. X.; Lu, C. & Jin, R. (2018). Reachability querying: An independent permutation labeling approach. *The VLDB Journal*, 27(1):1--26. ISSN 1066-8888.
- Yannakakis, M. (1982). The complexity of the partial order dimension problem. *SIAM. J. on Algebraic and Discrete Methods*, 3(3):351--358.
- Yildirim, H.; Chaoji, V. & Zaki, M. J. (2010). Grail: Scalable reachability index for large graphs. *Proc. VLDB Endow.*, 3(1-2):276--284. ISSN 2150-8097.
- Yildirim, H.; Chaoji, V. & Zaki, M. J. (2012). Grail: A scalable index for reachability queries in very large graphs. *The VLDB Journal*, 21(4):509--534. ISSN 1066-8888.