UNIVERSIDADE FEDERAL DO PARANÁ

GEOFFREY ALBERTO VITORIO MARTINS

MANUTENÇÃO DE CAMINHOS MÍNIMOS EM GRAFOS DINÂMICOS

CURITIBA

2012

UNIVERSIDADE FEDERAL DO PARANÁ

GEOFFREY ALBERTO VITORIO MARTINS

MANUTENÇÃO DE CAMINHOS MÍNIMOS EM GRAFOS DINÂMICOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação no curso de Mestrado em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. André Luiz Pires Guedes.

CURITIBA

2012

M386m Martins, Geoffrey Alberto Vitorio
Manutenção de caminhos mínimos em grafos dinâmicos / Geoffrey Alberto Vitorio Martins. – Curitiba, 2012.
64f. : il. color. ; 30 cm.

Dissertação (mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Ciência da Computação, 2012.

Orientador: André Luiz Pires Guedes.
Bibliografia: p. 63-64.

1. Teoria dos Grafos. 2. Algoritmos paralelos. I. Universidade Federal do Paraná. II. Guedes, André Luiz Pires. III. Título.

CDD: 511.5

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Geoffrey Alberto Vitorio Martins, avaliamos o trabalho intitulado, *"MANUTENÇÃO DE CAMINHOS MÍNIMOS EM GRAFOS DINÂMICOS"*, cuja defesa foi realizada no dia 30 de agosto de 2012, às 17:00 horas, no Auditório do Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 30 de agosto de 2012.

Prof. Dr. André Luiz Pires Guedes
DINF/UFPR – Orientador

Prof. Dr. Murilo Vicente Gonçalves da Silva
UTFPR – Membro Externo

Prof. Dr. André Vignatti
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

Dedico este trabalho à memória de meus pais, que não estão mais entre nós, ao Prof. André Guedes que sempre insistiu em me mostrar que o trabalho tinha potencial, mesmo quando eu mesmo não conseguia enxergá-lo e aos grandes amigos do curso de Ciência da Computação da UFPR, por me darem os melhores anos da minha vida.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. André Luiz Pires Guedes, pelo acompanhamento, confiança, orientação, amizade e persistência.

Ao curso de Mestrado de Ciência da Computação da Universidade Federal do Paraná pela oportunidade e todo o apoio recebido.

Aos meus grandes amigos Alexsandro Oliveira e Marcel Leite, pelas discussões acaloradas e de alto nível, que sempre me fizeram recobrar a esperança no mundo.

Aos meus falecidos pais, que de formas particulares sempre acreditaram no meu sucesso e nunca deixaram de apoiar e investir, mesmo quando tudo apontava para o exato oposto. Espero tê-los orgulhado.

“*Eu tenho a força, cavaleiro de Jedi. Então vem popozuda...”*

DeFalla

RESUMO

Esse trabalho faz uma análise crítica detalhada de alguns algoritmos de manutenção e consulta em grafos dinâmicos como o algoritmo de Even e Shiloach, o algoritmo de Henzinger e King e em especial o algoritmo de Roditty e Zwick, um algoritmo totalmente dinâmico com propriedades interessantes como critérios para a reinicialização de suas estruturas ou o uso de paralelismo em suas consultas.

Além de apresentar o algoritmo e sua análise inicial, uma análise crítica é feita examinando diversos aspectos do algoritmo original, onde são propostas modificações de forma a introduzir melhorias de precisão e desempenho, assim como melhores maneiras de escolher seus conjuntos aleatórios, modificações em suas condições de reinício para aumentar as chances de que o caminho mínimo seja encontrado ou ainda inclusão de funcionalidade adicional.

Palavras-Chave: Grafos, Grafos Dinâmicos, Alcançabilidade, Conectividade, Caminhos Mínimos, Manutenção de Caminhos Mínimos em grafos, Grafos direcionados.

ABSTRACT

This work provides a detailed critical analysis of some dynamic graph algorithm, targeting maintenance and queries of shortest paths. Analyzed algorithms includes the Even-Shiloach algorithm, the Henzinger-King Algorithm and as the main focus, the algorithm of Roditty and Zwick, a fully-dynamic all-pairs algorithm with a number of interesting characteristics such as the existence of certain conditions in which a full reinitialization of the structures is performed or the use of parallelism in its query processes.

Besides presenting an introduction to the algorithms and the initial analysis, a critical analysis of the Roditty-Zwick algorithm is presented, pointing the strong and weak spots and highlighting some of the more interesting characteristics of the algorithm. Afterwards, a number of changes are proposed, such as better ways to choose its random vertex sets, modifications in the restart conditions or even the addition of new functionality.

Keywords: Dynamic Graphs, Reachability, Connectivity, Shortest Paths, Dynamic Graph Maintenance, Directed Graphs.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Um dos aspectos mais interessantes de teoria dos grafos é a proximidade de alguns problemas teóricos com problemas do mundo real. De fato, o estudo de vários tipos de caminhos em grafos (por exemplo, Caminhos Eulerianos e Ciclos Hamiltonianos) encontram aplicações e auxiliam no desenvolvimento de soluções e tomada de decisão de problemas reais.

Enquanto grafos são tradicionalmente estudados como objetos estáticos, grafos dinâmicos representam melhores modelos do mundo real, onde mudanças que acontecem com o passar do tempo demandam a busca por soluções para que propriedades importantes dos grafos sejam mantidas após uma sequência de alterações e maneiras para melhor entender o efeito que tais alterações têm no grafo em si.

As primeiras experiências com grafos dinâmicos vieram da ideia de aplicar teoria dos grafos a sistemas sociais, onde se reconheceu que grafos deveriam mudar com o passar do tempo de forma a melhor refletir a realidade das interações sociais em grupos e organizações (SILJAK, 2006). Sistemas sociais são sistemas complexos que mapeiam interações em grupos de pessoas seja por existência de relações ou ainda o nível de importância de cada relação, porém a dinâmica social que influencia tais relações exige uma estrutura adaptável que seja capaz de refletir corretamente as mudanças em tais relações, com o intuito de ser capaz de responder perguntas sobre estas relações como grupamentos sociais, centralidade, distância entre indivíduos, etc. Posteriormente o conceito foi expandido de forma a contemplar outros modelos de relações dinâmicas que possam ser mapeadas em problemas da vida real.

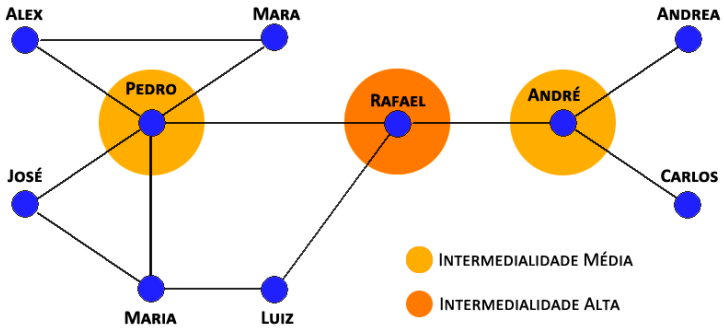

Figura 1 - Sistema social representado em um grafo.

A Figura 1 demonstra um sistema social simples destacando uma propriedade chamada centralidade de intermedialidade (do inglês: *Betweeness Centrality*) entre indivíduos do grupo. Note que uma única mudança nas relações entre indivíduos pode alterar totalmente o balanço de tal propriedade, indicando que esta relação deve ser observada através do tempo.

A motivação inicial deste trabalho se deu examinando conceitos de redes móveis ad-hoc em que dispositivos conectam-se uns aos outros por proximidade e o movimento desses dispositivos causa mudanças nas conexões e por consequência nas rotas para que os pacotes de dados chegassem ao destino. O algoritmo aplicado a resolver o problema deve ser capaz de responder perguntas sobre o estado da conectividade entre dispositivos, ou ainda melhores rotas para os pacotes de dados. Adicionalmente, diferentes dispositivos possuem diferentes capacidades ou alcances de transmissão e recepção, que são análogos às direções e pesos de arestas em grafos direcionados como demonstrado na Figura 2. Este problema, dadas as variáveis envolvidas, só pode ser corretamente estudado quando se consideram grafos dinâmicos direcionados.

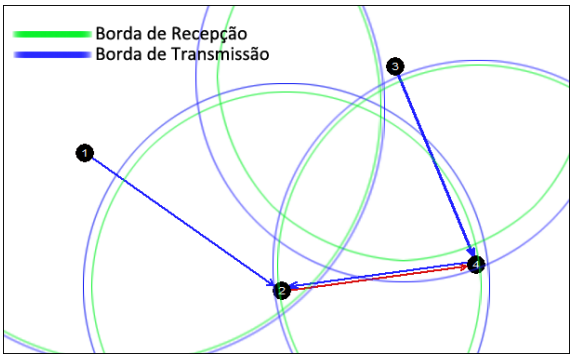

Figura 2 - Estações móveis e suas capacidades de transmissão e recepção.

Existem diversas maneiras de promover mudanças em um grafo, seja por retirada ou inserção de arestas ou vértices, ou ainda mudanças nos pesos de arestas. Este trabalho se propõe a estudar a manutenção de caminhos mínimos em grafos dinâmicos direcionados sem considerar pesos nas arestas, e que por sua vez são afetados por inserções e remoções de arestas através da análise de algoritmos relevantes para o tema.

Um algoritmo projetado para lidar com grafos dinâmicos tem usualmente três componentes básicas distintas: Inicialização, Atualização (ou manutenção) e Consultas.

A Inicialização prepara as estruturas de dados para receberem a informação do grafo e quaisquer informações adicionais que sejam necessárias no processo de obtenção das soluções. A Atualização é responsável por refletir as mudanças no grafo em suas estruturas à medida que acontecem e manter todos os dados necessários à solução do problema. As Consultas são perguntas feitas à estrutura de dados de forma a responder sobre uma determinada propriedade do grafo em certo estado de execução do algoritmo.

O custo de cada uma dessas componentes é diretamente dependente da estrutura de dados. A estrutura de dados mais simples para armazenar um grafo é uma matriz. Embora esta estrutura seja simples e favorável para operações de atualização, é desfavorável para operações de consulta, por exigir uma total varredura da estrutura para a resposta de consultas simples. Na medida em que a estrutura mais simples é alterada para favorecer operações de consulta, a atualização e manutenção da estrutura se tornam mais complicadas. Visando o equilíbrio de custos entre essas duas operações, a relação entre atualizações e consultas é o que define qual o tipo de algoritmo mais adequando para a solução de um determinado problema.

Neste trabalho o problema a ser resolvido é a manutenção de caminhos mínimos em grafos dinâmicos direcionados. Os problemas consideram os grafos sem pesos nas arestas, com inserções e remoções de arestas, mas não será considerada a inserção ou remoção de vértices, embora isto possa ser alcançado indiretamente removendo-se todas as arestas incidentes a um dado vértice. As operações de consulta relevantes para o problema são as consultas

de distância mínima e alcançabilidade. O trabalho trata principalmente do algoritmo de Roditty e Zwick (RODITTY; ZWICK, 2011), um algoritmo publicado em 2011 e que demonstra os melhores índices de eficiência dentre os algoritmos inicialmente pesquisados e por possuir características interessantes como o uso de processos paralelos para obter ganhos em tempo de processamento. O algoritmo de Roditty e Zwick faz uso de outros dois algoritmos de grafos dinâmicos para manutenção de suas estruturas internas, o algoritmo de Even e Shiloach (EVEN; SHILOACH, 1981) e o algoritmo de Henzinger e King (HENZINGER; KING, 1995).

Este trabalho mostra um detalhamento do algoritmo de Roditty e Zwick e seus algoritmos dependentes, e faz uma análise crítica de seus processos, mostrando pontos fortes e fracos, e sugerindo pequenas modificações para diminuir a incidência de erros e melhorar escolhas aleatórias.

No Capítulo 2, são apresentadas definições e notações usadas comumente neste trabalho.

O Capítulo 3 apresenta o algoritmo de Even e Shiloach, um algoritmo decremental para problemas parcialmente dinâmicos que é capaz de responder consultas de conectividade em árvores ou grafos com rapidez e com estruturas de simples manutenção.

O Capítulo 4 apresenta o algoritmo de Henzinger e King, um algoritmo decremental Monte Carlo para manutenção de grafos dinâmicos que usa o algoritmo de Even e Shiloach para manutenção de suas estruturas, mas com um esquema de consulta otimizado que consegue resultados significativamente melhores em consultas de alcançabilidade.

No Capítulo 5 é apresentado o algoritmo de Roditty e Zwick, tema principal deste trabalho. Este algoritmo utiliza os algoritmos de Even e Shiloach e Henzinger e King para manutenção de suas estruturas, mas utiliza-se de um esquema de busca com processos paralelos e parâmetros que indicam a necessidade de um total reinício das estruturas de dados uma vez que elas se tornam cada vez mais difíceis de manter com o passar do tempo. É um

algoritmo poderoso que mantém a conectividade de maneira eficaz e elegante, porém com certa probabilidade de erro.

O Capítulo 6 traz uma análise de vários pontos do algoritmo de Roditty e Zwick, com especial atenção a cada um dos diferentes passos de seu processo de busca, entendendo os fatores que determinam que um resultado venha de cada uma das partes de uma consulta. Também examina aspectos das escolhas aleatórias feitas pelo algoritmo, condições para reinicialização de suas estruturas e propõe mudanças no algoritmo de forma a mitigar chances de pior caso ou aumentar probabilidades de sucesso.

O Capítulo 7 é um resumo do trabalho e seus resultados, concluindo a análise dos algoritmos dinâmicos e abrindo possibilidades para trabalhos futuros que possam dar continuidade a este estudo.

2 DEFINIÇÕES

Este capítulo visa introduzir definições e conceitos que serão utilizados neste trabalho.

A seção 2.1 apresenta definições relativas a grafos. A seção 2.2 detalha definições sobre diferentes tipos de buscas em grafos. A seção 2.3 traz definições sobre diferentes problemas em grafos dinâmicos e por fim, a seção 2.4 contém definições sobre problemas aleatorizados, manutenibilidade, análise amortizada e esparsificação.

2.1 GRAFOS

Vértices (ou nós, ou nodos) e Arestas são as unidades fundamentais de um grafo. O conjunto de todos os vértices de um grafo é aqui chamado de conjunto *V*, e o conjunto de todas as arestas de um grafo é aqui denominado conjunto *E*.

Um Grafo G = (*V*, *E*) é um par de conjuntos de forma que os elementos de E são subconjuntos com dois elementos de V. Uma aresta entre os vértices *u* e *v* (em que a ordem dos vértices não importa) é denotada por {*u*,*v*}. Um grafo é uma representação abstrata de um conjunto de objetos onde alguns pares de objetos (vértices) são conectados por arestas. A forma usual de representação de um grafo é desenhando pontos que representam os vértices e unindo tais pontos por linhas ou curvas, representando as arestas, como mostra a Figura 3 (DIESTEL, 2000). Um Grafo Ponderado é um grafo com uma função $w: E \rightarrow \mathbb{R}$ associando pesos às arestas.

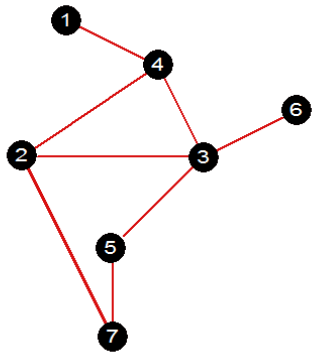

Figura 3 - Exemplo de grafo não direcionado.

Um Grafo Direcionado (ou digrafo) é um grafo cujas arestas possuem direções associadas a elas. (DIESTEL, 2000). Uma aresta que vai do vértice *u* ao vértice *v* é denotada por (*u*,*v*). A Figura 4 representa um grafo direcionado.

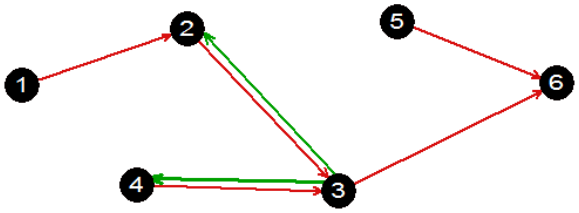

Figura 4 - Exemplo de grafo direcionado.

Um Subgrafo de um grafo G é um grafo cujo conjunto de vértices é um subconjunto do conjunto de vértices V e o conjunto de arestas é um subconjunto do conjunto E, ou seja, cuja relação de adjacência é um subconjunto de G restrita a esse subconjunto (SZWARCFITER, 1988).

Um Caminho em um grafo é uma sequência de vértices tal que de cada vértice existe uma aresta para o próximo vértice da sequência. Um caminho pode ser infinito, porém um caminho finito sempre terá um vértice inicial e um vértice final (DIESTEL, 2000).

Um grafo é dito Conexo se existe ao menos um caminho entre qualquer par de vértices. Em grafos direcionados, se o vértice *u* pode ser alcançado por ao menos um caminho partindo do vértice *v*, o vértice *u* é dito Alcançável a partir de *v*. Alcançabilidade é a propriedade de um vértice de poder alcançar ou

ser alcançado por outros vértices. Se em um grafo direcionado existe ao menos um caminho entre qualquer par de vértices, este grafo direcionado será denominado Fortemente Conexo.

Um Ciclo é um caminho que tem o mesmo vértice como inicial e final. A escolha do vértice inicial ou final de um ciclo é arbitrária. Um ciclo em um grafo direcionado é chamado Ciclo Direcionado. Um grafo Acíclico é um grafo que não possui ciclos em sua estrutura. (DIESTEL, 2000)

O Fecho Transitivo de um grafo G é definido como o grafo G* = (V, E*), onde E* = {(*i*,*j*) : existe um caminho do o vértice *i* até o vértice *j* em G}. O fecho transitivo também pode ser visto como a matriz de todos os caminhos de qualquer comprimento em G.

O conceito de fecho transitivo é especialmente útil para responder consultas de alcançabilidade, pois após a construção do fecho esta consulta pode ser respondida facilmente em tempo linear, bem como as buscas em profundidade e largura. O fecho transitivo também pode ser usado para encontrar as componentes conexas de um grafo.

Uma Componente Conexa de um grafo é um subgrafo maximal conexo de G (DIESTEL, 2000). A Figura 5 mostra um exemplo de um grafo com três componentes conexas distintas, representadas pelas cores de suas arestas.

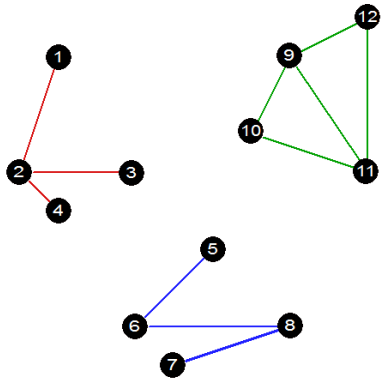

Figura 5 - Um grafo com três componentes conexas distintas.

Uma Árvore é um grafo conexo e acíclico. Grafos acíclicos, porém não conexos são denominados Floresta. Floresta também pode ser entendida como uma união disjunta de árvores (SZWARCFITER, 1988) como demonstrado na

Figura 6. Um subgrafo H de um grafo F que contenha todos os vértices de G e seja uma árvore é chamado de Árvore Geradora de G. (EVEN, 1979)

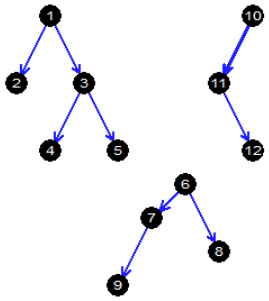

Figura 6 - Uma floresta, ou união disjunta de árvores.

Grafos dinâmicos são grafos que estão sujeitos a mudanças discretas em sua estrutura. Uma mudança em um grafo pode ser uma simples inserção ou remoção de aresta ou mudanças em suas propriedades como peso de arestas. (DEMETRESCU; FINOCCHI; ITALIANO, 2001)

Alcançabilidade (do inglês: *Reachability*) é a habilidade de ir de um dado vértice a outro em um grafo direcionado. A alcançabilidade em um grafo não direcionado é trivial, bastando encontrar as componentes conexas, o que pode ser feito em tempo linear considerando o tamanho total do grafo ($|V| + |E|$). Um grafo direcionado é Unilateralmente Conexo quando, para qualquer par de vértices, ao menos um vértice é alcançável do outro (DAINTITH, 2008).

2.2 BUSCAS EM GRAFOS

Uma Busca em Largura (do Inglês: *Breadth-First Search* ou BFS) é um algoritmo que realiza um percurso ou travessia num grafo começando por um vértice denominado raiz e explorando todos os vértices vizinhos. Então, para cada um desses vértices mais próximos, explora os seus vértices vizinhos inexplorados e assim por diante, até que se encontre o alvo da busca (KNUTH, 1968). O algoritmo de busca em largura utiliza uma fila como estrutura interna.

Busca em Profundidade (do Inglês: *Depth-First Search* ou DFS) é um algoritmo que efetua uma busca em um grafo e explora o mais longe possível

antes de retornar ao último ponto de escolha e tentar o próximo caminho alternativo. O algoritmo de busca em profundidade utiliza uma pilha como estrutura auxiliar, ou pode ser implementado com recursividade.

2.3 PROBLEMAS EM GRAFOS DINÂMICOS

Os problemas em grafos dinâmicos podem ser classificados em: Totalmente Dinâmicos, se as atualizações incluem inserções e remoções irrestritas de arestas, ou, Parcialmente Dinâmicos, se apenas uma das operações de inserção ou remoção é permitida. Se apenas inserções forem permitidas, o problema é chamado Incremental. Se apenas remoções são permitidas, é chamado Decremental. (EPPSTEIN; GALIL; ITALIANO, 1999)

Problemas em grafos dinâmicos também podem ser classificados quanto à utilização de um vértice raiz ou não. Problemas que são focados na solução do ponto de vista de um único vértice e/ou focado em um único ponto de partida são chamados de "*single source*". Um exemplo de problema *single source* é o "*Single Source Shortest Path*" (SSSP) que é a geração do caminho mínimo a partir de um vértice para todos os outros vértices de um grafo.

O problema de redes *ad-hoc* mencionado na introdução deste trabalho é um problema *single source*, pois cada um dos dispositivos integrantes da rede calcula a topologia do grafo e as rotas para entrega de pacotes de acordo com sua própria visão da rede, o que é equivalente a cada vértice manter suas próprias estruturas contendo sua própria visão do grafo.

Problemas que se propõem a encontrar soluções para todos os vértices (ou pares de vértices) são chamados "*all-pairs*". Um exemplo de problema all-pairs é o "*All Pairs Shortest Path*" (APSP), onde os caminhos mínimos entre todos os pares de vértices são calculados. O problema APSP pode ser visto como uma generalização do problema SSSP.

Os problemas envolvendo sistemas sociais brevemente descritos na introdução deste trabalho são exemplos de problemas "*all-pairs*", pois o

resultado que se deseja obter é uma visão do todo, e não a visão de cada indivíduo no grupo.

2.4 DEFINIÇÕES GERAIS

Dois tipos principais de métodos estatísticos para algoritmos são referenciados na literatura: O Método Monte Carlo e o Método Las Vegas. No método de Monte Carlo se baseia em amostragens aleatórias massivas para obter resultados numéricos e pode gerar soluções com probabilidade de falha arbitrariamente pequena. Já o Método de Las Vegas propões reduzir a quantidade de erros gerada pelo método Monte Carlo utilizando funções para distribuição de probabilidades que fazem o algoritmo se concentrar nas áreas com maior probabilidade de contribuição na resposta final (HROMKOVIC, 2002)

No caso de problemas de decisão, essencialmente, existem dois tipos de algoritmo de Monte Carlo: algoritmos com erro unilateral (do inglês: *one-sided error*) e algoritmos com erro bilateral (do inglês: *two-sided error*). Um algoritmo Monte Carlo tem erro unilateral, se a probabilidade de fracasso é zero, para pelo menos uma das saídas Sim ou Não. Se a probabilidade de falha é não nula para as saídas Sim e Não, o algoritmo possui erro bilateral. (HROMKOVIC, 2002)

O método Las Vegas sempre assegura uma solução correta para o problema considerado. Entretanto, ela nem sempre é obtida dentro de um tempo de execução suportável. Essa abordagem é interessante para aqueles problemas cujo tempo esperado de processamento seja suficientemente pequeno (ou polinomial) para a aplicação em questão. No método de Las Vegas, executamos sucessivas repetições do mesmo algoritmo até que uma solução correta seja encontrada.

Manutenibilidade (do inglês: *Maintainability*) é a capacidade de um grafo associado a um algoritmo de manter uma determinada propriedade do grafo dinâmico enquanto este sofre alterações em suas arestas ou vértices. Um exemplo é o problema APSP dinâmico clássico, que recalcula todos os

caminhos mínimos entre todos os pares de vértices toda vez que uma aresta é retirada ou inserida.

Análise Amortizada é uma forma de análise de algoritmos apresentada inicialmente por Robert Tarjan (TARJAN, 1985) que leva em consideração toda a sequência de operações de um programa. Ela permite que se estabeleçam limites para o pior caso de desempenho independentemente das entradas, por observação de todas as operações. Este método é baseado na ideia de que enquanto certas operações de um programa podem ser dispendiosas em recursos, elas podem não ocorrer em uma frequência alta o suficiente para comprometer todo o programa, de forma que o número de operações menos onerosas será de longe superior às mais dispendiosas em longo prazo, mudando o balanço de complexidade ao longo de várias iterações. É particularmente útil justamente por garantir o desempenho do pior caso ao invés de fazer suposições sobre o estado do programa. (FIEBRINK, 2012)

Esparsificação (do inglês: *Sparsification*) é uma técnica para dinamizar algoritmos de grafos dinâmicos. É uma técnica no estilo “dividir e conquistar” que permite reduzir a dependência do número de arestas de um grafo de forma que o tempo usado para manter uma determinada propriedade de um grafo se torne equivalente a manter a mesma propriedade em um grafo esparso. (EPPSTEIN, *et al.*, 1997)

3 ALGORITMO DE EVEN E SHILOACH

O algoritmo de Shimon Even e Yossi Shiloach (EVEN; SHILOACH, 1981) é um algoritmo decremental capaz de manter os primeiros *k* níveis de uma árvore de caminhos mínimos de origem única (SSSP). Este algoritmo pode ser aplicado para grafos direcionados, não direcionados ou árvores e sua principal característica é possuir tempo de execução no caso médio da ordem de **O(k.m)**, onde *m* é o número inicial de arestas no grafo.

O algoritmo utiliza uma tabela para os vértices que armazena o nome da componente conexa a qual cada vértice pertence. Dessa forma, quaisquer consultas de conectividade neste algoritmo podem ser respondidas em tempo **O(1)** apenas por comparação dos nomes das componentes conexas.

Outra característica importante deste algoritmo é a utilização de processos paralelos para varredura do grafo ou árvore, executando buscas em largura e em profundidade. Note que isto não se refere à computação paralela, mas sim a um intercalamento de subprocessos (dados dois processos A e B sendo A = {a1, a2, a3...} e B={b1, b2, b3}, o processo intercalado resulta em {a1, b1, a2, b2, a3, b3,...}.). Quando implementado computacionalmente, nada impede que computação paralela seja utilizada.

O algoritmo de Roditty e Zwick (RODITTY; ZWICK, 2011) que é o foco deste trabalho, bem como o algoritmo decremental de Henzinger e King utilizam-se do algoritmo de Even e Shiloach para uma rápida manutenção de conjuntos de conectividade.

Na seção 3.1 é descrita a versão do algoritmo de Even e Shiloach aplicada a árvores. A seção 3.2 trata de Even e Shiloach aplicado a grafos. As duas versões são utilizadas por outros algoritmos descritos neste trabalho.

3.1 ALGORITMO DE EVEN E SHILOACH EM ÁRVORES

Dado um grafo G=(V,E), onde V é o conjunto de vértices e E é o conjunto de arestas, e este grafo não possui ciclos, então este grafo é uma única árvore ou é uma floresta com *n* árvores. Uma vez que o algoritmo aqui descrito é decremental, é possível afirmar que a remoção de uma aresta qualquer transforma o grafo em uma floresta com *n+1* árvores.

O algoritmo utiliza uma tabela que armazena para cada vértice o nome da componente conexa a que este vértice pertence. Esta tabela de vértices e componentes conexas é atualizada da seguinte forma: Cada vez que uma aresta ***e*** que liga os vértices *x* e *y* é removida da árvore T, duas varreduras são disparadas, a partir dos vértices *x* e *y* com o objetivo de percorrer totalmente as duas componentes conexas criadas pela remoção desta aresta. Este processo de varredura pode ser feito em largura ou profundidade. Quando uma dessas varreduras termina, o processo geral é paralisado e a tabela é atualizada com um novo nome de componente conexa para todos os vértices visitados pela varredura que finalizou. Os vértices da outra varredura permanecem com o mesmo nome anterior, logo somente a menor das componentes conexas resultantes recebe um novo nome.

Como a tabela de vértices e componentes conexas é atualizada a cada remoção de aresta, o tempo de resposta para uma consulta do tipo "*os vértices a e b pertencem à mesma componente conexa?*" pode ser respondida em tempo constante apenas por uma simples comparação de nomes na tabela.

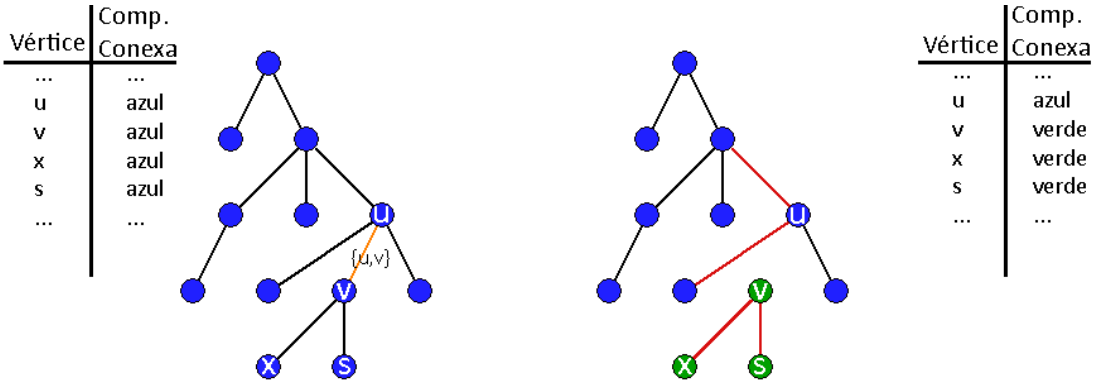

Figura 7 - Remoção de uma aresta (*u*,*v*) e reorganização das componentes conexas pelo algoritmo de Even e Shiloach

Na Figura 7, quando a aresta é removida, buscas em largura são iniciadas a partir dos vértices *u* e *v*. Como o processo que parte do vértice *v* termina primeiro, todos os vértices alcançados pela busca em largura passam a pertencer a uma nova componente conexa (verde). O processo de busca em largura na outra componente conexa é então finalizado.

Note que este algoritmo assume que os vértices são homogêneos, ou seja, tem o mesmo poder computacional. O algoritmo pode ser facilmente adaptado para trabalhar com vértices/arestas com diferentes pesos, mas isto afeta diretamente o tempo de execução assintótico.

3.2 ALGORITMO DE EVEN E SHILOACH EM GRAFOS

Da mesma forma que descrito na seção anterior, o algoritmo Even e Shiloach aplicado a grafos também mantém uma tabela de vértices, que indica para cada um deles o nome da componente conexa a qual o vértice pertence, logo uma consulta sobre dois vértices quaisquer pertencerem ou não a mesma componente conexa é respondida apenas por comparação dos nomes na tabela. Porém a manutenção da tabela de componentes conexas torna-se mais complicada para grafos e é aqui descrita:

O algoritmo utiliza-se novamente de processos paralelos. O primeiro processo (processo A) checa se a remoção de uma aresta quebra uma

componente conexa e o segundo (processo B) verifica se a remoção da aresta não quebra a componente conexa a que ela pertence.

O processo A utiliza o algoritmo de busca em profundidade e analogamente ao método usado em árvores, dispara a busca a partir dos vértices *x* e *y* da aresta removida e. Cada uma das buscas tem por objetivo encontrar o outro vértice, e os dois processos param se:

a) Um dos processos alcançou o outro vértice, e neste caso eles ainda estão na mesma componente conexa.
b) Um dos processos finaliza sem encontrar o outro vértice, e neste caso eles não estão mais na mesma componente conexa.

Embora o resultado (a) possa dizer com certeza que os dois vértices ainda pertencem a uma mesma componente conexa, este resultado não é levado em conta até a execução do processo B. Isto se deve ao fato de que o processo B, por utilizar busca em largura possui uma estrutura que deve ser mantida.

No caso de um dos subprocessos do processo A finalizar sem encontrar o outro vértice (resultado (b)), a componente conexa original foi quebrada, então todos os vértices da busca que foi finalizada recebem um novo nome de componente conexa e a tabela é atualizada, de forma análoga a execução do algoritmo em árvores.

O processo B utiliza o algoritmo de busca em largura, logo é necessário um processo para a inicialização da estrutura de dados deste algoritmo: Um vértice aleatório r é escolhido, e o algoritmo de busca em largura é executado a partir dele, armazenando vértices em níveis. O nível N_0 contém apenas o vértice ***r***. Todos os vértices de distância i para o vértice *r* estão no nível N_i. Se o grafo G não é conexo, um novo vértice *s* ainda não utilizado é escolhido e o algoritmo de busca em largura é reiniciado. Este vértice *s* é colocado no nível N_1 e uma aresta artificial é introduzida para conectar *r* e *s*. O objetivo das arestas artificiais é manter todas as componentes conexas na mesma estrutura da busca em largura, e não tem outro propósito. As arestas artificiais somente são utilizadas no processo B.

A estrutura tem as seguintes propriedades:

1) Um vértice v no nível N_i, i>0 tem ao menos uma aresta que o conecta a algum vértice em N_{i-1}. Se existe apenas uma aresta assim, ela é possivelmente uma aresta artificial, porém se existe mais de uma, nenhuma delas é artificial.
2) Um vértice *v* no nível N_i pode ter qualquer número de arestas que o conectem a outros vértices no nível N_i e com vértices no nível N_{i+1} ou N_{i-1}, mas não pode ter arestas que conectem a outros níveis que não sejam N_{i+1}, N_i, e N_{i-1}.

Denominam-se conjuntos **α(v)**, **β(v)** e **γ(v)** os conjuntos de arestas que conectam com N_{i-1}, N_i e N_{i+1}, respectivamente.

Quando uma aresta **(*u*,*v*)** é excluída, o processo B considera os seguintes dois casos:

1. **Os vértices *u* e *v* estão no mesmo nível.** Neste caso a remoção da aresta não muda as componentes conexas. A aresta é simplesmente excluída dos conjuntos **β(u)** e **β(v)** e o processo B (e também o processo A) é paralisado.
2. **Os vértices *u* e *v* estão em níveis diferentes**. Assumindo que $u \in N_{i-1}$ e $v \in N_i$. A aresta é removida de γ(u) e α(v).
   2.1. Se o novo **α(v)** não é vazio, então as componentes não se alteram. O processo B é paralisado.
   2.2. Se o novo **α(v)** é vazio, então o vértice *v* "desce" ao menos um nível, e isso pode causar uma sequência de atualizações nos vértices dos níveis inferiores. É então usada uma fila Q, na qual colocamos vértices cujo nível deve ser mudado. O vértice v é colocado na fila Q e os seguintes passos são aplicados:

      a. Se a fila Q está vazia, este procedimento acaba e os processos A e B são paralisados.
      b. Seja *w* o primeiro elemento de Q. Remova *w* de Q.
      c. Remova *w* do seu nível atual, e coloque-o no próximo nível (de N_j para N_j+1)

d. Para cada aresta e'=(*w*,*w'*) em β(*w*), remova e' de β(*w'*) e ponha em γ(*w'*)
e. α(*w*) ← β(*w*)
f. Para cada aresta e'=(*w*,*w'*) em γ(*w*) remova e' de α(*w*) e ponha em β(*w*). Se o novo α(*w*) é vazio, ponha *w'* em Q.
g. β(*w*) ← γ(*w*). γ(*w*) ← ∅.
h. Se α(*w*) é vazio, ponha *w* em Q.
i. Retorne ao passo (a).

Se a remoção de uma aresta não quebrar nenhuma componente conexa e o caso for 2.2, então em algum ponto o procedimento será paralisado. Se a remoção quebrar uma componente, o procedimento não será paralisado, porém o Processo A vai reconhecer essa condição e paralisará o processo.

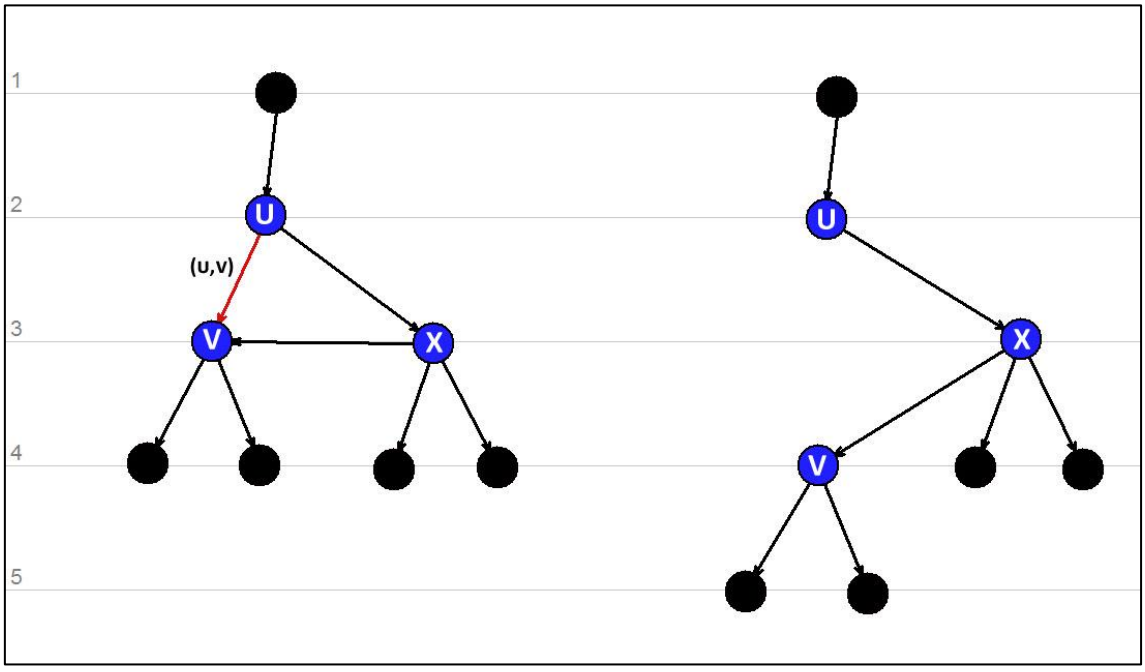

Figura 8 - Caso 2.2 do Algoritmo de Even e Shiloach.

Na Figura 8, a remoção da aresta (*u*,*v*) força o vértice v a descer de nível e provoca uma "avalanche" em todos os seus vértices filho.

3.3 COMPLEXIDADE DE EVEN E SHILOACH

Quando aplicado a árvores, o algoritmo de Even e Shiloach consegue responder consultas em tempo O(1), pois a resposta é obtida por simples comparação de entradas na tabela. Para atualização da tabela devido a uma remoção de aresta, o seguinte cálculo é aplicado: Uma vez que o número de vértices da nova componente não excede o número de vértices de sua contraparte, cada vértice pode pertencer a uma componente renomeada no máximo $\lfloor \log |V| \rfloor$ vezes, onde |V| é o número de vértices total da árvore. Se associarmos um custo às operações de varredura, levamos em conta a aresta que diretamente chega ao vértice cujo nome da componente conexa foi alterado, e em paralelo uma aresta pertencente à outra componente, então cada vértice é levado em consideração também $\lfloor \log |V| \rfloor$ vezes, tendo como resultado a complexidade de $O(|V| \log |V|)$, para todo o processo de atualização das tabelas.

Quando aplicado a grafos, temos que levar em conta as complexidades dos processos A e B descritos. Analogamente ao método utilizado em árvores, podemos inferir que a complexidade do Processo A é O(|E|log|E|), pois a operação decisiva para a complexidade é executada nas arestas, e não nos vértices.

No Processo B, segundo o artigo de Even e Shiloach (EVEN; SHILOACH, 1981) a complexidade é da ordem de $O(|V| \cdot |E|)$, portanto a complexidade resultante do algoritmo de Even e Shiloach aplicado a grafos é $O(|V| \cdot |E|)$.

4 ALGORITMO DE HENZINGER E KING

Em 1995, Monika Hauch Henzinger e Valerie King apresentaram em "*Fully Dynamic Biconnectivity and Transitive Closure*" (HENZINGER; KING, 1995) um algoritmo para lidar com problemas de biconectividade em grafos totalmente dinâmicos, que era exponencialmente mais rápido que soluções propostas anteriormente. Este algoritmo apresentava uma abordagem diferente do que se usava até então em termos de alcançabilidade. No mesmo artigo foi proposta uma variação do algoritmo principal que lida apenas com remoções de arestas (decremental), e faz uso do algoritmo de Even e Shiloach para manutenção interna de suas sub-árvores.

O algoritmo *Henzinger e King,* é Monte Carlo para as consultas, ou seja, consegue garantir a corretude de respostas afirmativas do tipo "*sim, o vértice u pode ser alcançado a partir do vértice v*", porém para respostas negativas existe uma probabilidade de erro de $O(1/n^c)$, onde c é uma constante previamente escolhida, relacionada ao grau de precisão desejado do algoritmo.

O tempo amortizado de atualização da estrutura de dados do algoritmo devido à remoção de uma aresta é de $O(n \log^2 n)$, e o tempo amortizado de consulta de alcançabilidade é de $O\left(\frac{n}{\log n}\right)$.

O algoritmo decremental de Henzinger e King é um dos algoritmos utilizados ativamente pelo algoritmo de Roditty e Zwick (RODITTY; ZWICK, 2011), objeto principal deste estudo.

Na seção 4.1 é apresentado o algoritmo de Henzinger e King e na seção 4.2 é apresentada a complexidade deste algoritmo.

4.1 DESCRIÇÃO DO ALGORITMO DE HENZINGER E KING

Com base no algoritmo de Even e Shiloach, descrito em detalhes no capítulo anterior, o seguinte teorema é apresentado:

Teorema 1: O conjunto de todos os vértices alcançáveis a partir de um vértice específico, por um caminho cuja distância é igual ou menor a n em um grafo dinâmico decremental direcionado pode ser mantido em tempo O(m.n), onde n é o número de vértices e m o número de arestas.

A operação que consiste em uma aplicação do teorema 1 (e por consequência, do algoritmo de Even e Shiloach em um subgrafo) será aqui chamada de "manutenção". Esta operação mantém os caminhos mínimos para todos os vértices alcançáveis por uma distância menor ou igual a **k**, partindo de (ou convergindo para) um vértice **v**, e será denotada por **mantenha(v,k).**

Convencionamos aqui que o conjunto de vértices que podem ser alcançados a partir de um vértice *v*, por uma distância menor ou igual a k será chamado de **out(*v*,k)**. O conjunto de vértices que alcançam o vértice v percorrendo uma distância menor ou igual a k será chamado de **in(*v*,k)**. A Figura 9 demonstra a ideia em um grafo direcionado:

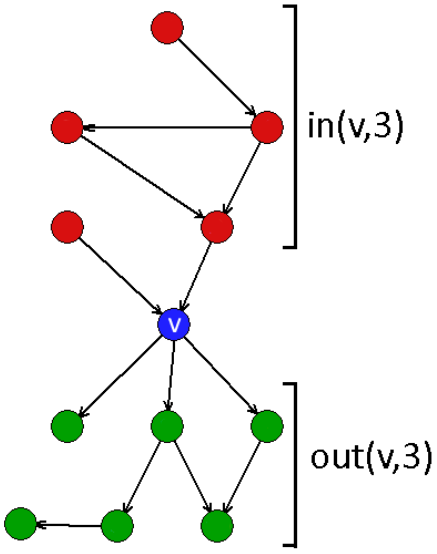

Figura 9 - Os conjuntos *in(v,k)* e *out(v,k)*, a partir do vértice *v* e com distância 3.

4.1.1 ***Inicialização***

Dado um grafo G=(V,E), seja **r** um parâmetro escolhido pelo usuário e limitado a 2 ≤ r ≤ |V|. Para cada i = 1,...,$\log r$, escolha aleatoriamente um conjunto de min{O(2^ilog n),n} vértices distintos S_i.

Para cada vértice distinto x, mantenha out(x, $n/2^i$) e in(x,$n/2^i$) para cada *i*, tal que:

a) $x \in S_i$, e
   a. Conjunto Out(x) = $U_{\{i,x \in Si\}}$ out(x, $n/2^i$), e
   b. Conjunto In(x) = $U_{\{i,x \in Si\}}$ in(x,$n/2^i$)

Finalmente, para cada vértice u ϵ V, mantenha out(u,*n*/*r*) e in(u,*n*/*r*). Todas estas operações de manutenção devem ser efetuadas cada vez que uma aresta for removida. Os conjuntos aqui descritos formam a estrutura de dados do algoritmo de Henzinger e King.

4.1.2 ***Consultas de Alcançabilidade***

Para responder uma consulta de alcançabilidade de v a partir de u:

1) Primeiro teste se *v* ∈ out(*u*,n/r). Se sim, a resposta é sim;
2) Se não, teste se para cada vértice distinto *x*, u ∈ In(*x*) e v ∈ Out(*x*). Se para algum vértice *x* as duas repostas forem positivas, a resposta é sim;
3) Caso nenhuma correspondência seja encontrada, a resposta é não.

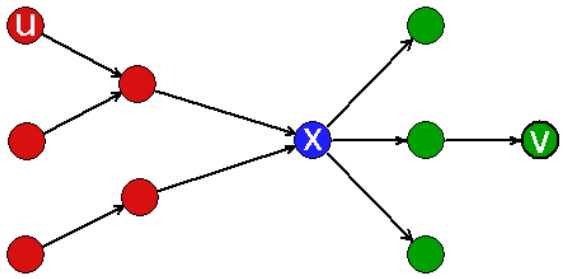

Figura 10 - O segundo teste de alcançabilidade do algoritmo de Henzinger e King.

Na Figura 10, O vértice *u* pode alcançar o (está conexo ao) vértice *v*, pois u ϵ In(*x*) e v ϵ Out(*x*). Note que a recíproca não é verdadeira para grafos direcionados.

Se o caminho mínimo entre (*u*,*v*) tem distância menor ou igual a n/r, então a consulta será respondida corretamente. Por outro lado, se o caminho mínimo entre (*u*,*v*) é maior do que n/r, a resposta é informada corretamente com alta probabilidade.

4.2 COMPLEXIDADE DE HENZINGER E KING

O tempo total de atualização da estrutura por remoção de uma aresta é $O\left(mn \log n \log r + \frac{n^2 m}{r}\right)$, que amortizado resulta em $O\left(n \log^2 n + \frac{n^2}{r}\right)$. O tempo de consulta é proporcional ao número de vértices distintos dados por $O(min\{r \log n\,, n\})$.

Se considerarmos $r = \frac{n}{\log^2 n}$, o tempo amortizado de atualização da estrutura será da ordem de $O(n \log^2 n)$ e o tempo amortizado de consulta é de $O(n \log n)$.

5 ALGORITMO DE RODITTY E ZWICK

O algoritmo criado por Liam Roditty e Uri Zwick (RODITTY; ZWICK, 2011) é um algoritmo para caminhos mínimos em grafos totalmente dinâmicos, entre todos os vértices. O algoritmo mistura conceitos dos algoritmos de Henzinger e King (HENZINGER; KING, 1995) e Even e Shiloach (EVEN; SHILOACH, 1981), porém adicionando ideias interessantes de manutenibilidade como a reinicialização das estruturas do algoritmo se determinadas condições são alcançadas e parâmetros que regulam a precisão dos resultados obtidos. Este algoritmo consegue resultados semelhantes ou melhores que o algoritmo de Henzinger e King e consegue um tempo de consulta O(t) com alta probabilidade, em que t é um parâmetro dado ao algoritmo, análogo ao parâmetro *r* do algoritmo de Henzinger e King.

O algoritmo de Roditty e Zwick trabalha em fases. No início de cada fase o grafo atual G=(V,E) é passado pelo algoritmo decremental de Henzinger e King (HENZINGER; KING, 1995) e a estrutura inicial deste algoritmo decremental é criada. Esta estrutura será usada apenas durante a fase atual de execução, sendo totalmente reconstruída na fase subsequente. No artigo “High-Probability Parallel Transtive Closure Algorithms” (ULLMAN; YANNAKAKIS, 1991), Jeffrey Ullman e Mihalis Yannakakis apresentaram uma propriedade de conjuntos aleatórios de vértices em relação a caminhos no grafo, que é descrita no teorema abaixo:

> ***TEOREMA 2****: Seja G = (V,E) um grafo direcionado com n vértices. Seja* $\mathbf{1} \leq \boldsymbol{k} \leq \boldsymbol{n}$ *e seja S um subconjunto de V obtido pela seleção aleatória de maneira independente de um conjunto de vértices de tamanho* $\frac{(cn\,ln\,n)}{k}$*. Se p é um caminho em G com tamanho pelo menos k, então a probabilidade de ao menos um dos vértices de p pertencer a S é de 1-*n^{-c}*.*

Seguindo este teorema, um conjunto de vértices S do grafo inicial G é escolhido, com tamanho $\frac{(\mathrm{cn\,ln}\,n)}{\mathrm{k}}$, onde k é um parâmetro de precisão do algoritmo a ser discutido posteriormente. Cada um destes vértices distintos será chamado de **pivô**. O algoritmo então emprega o algoritmo de busca em largura padrão para construir árvores de e para todos os vértices do conjunto

S. Para um vértice $w \in S$, chamaremos de $\mathbf{T_{in}(w)}$ a árvore de caminhos mínimos até o vértice w, e de $\mathbf{T_{out}(w)}$ a árvore de caminhos mínimos a partir do vértice w.

Um segundo subconjunto de vértices é criado para conter os vértices que terão arestas adicionadas incidindo aos mesmos. Este subconjunto C é inicializado sem vértices e esvaziado ao final de cada fase do algoritmo.

Nas seções 5.1 a 5.3 são descritas as operações de inserção e remoção de arestas e consulta de distância do algoritmo. A seção 5.4 apresenta a descrição formal do algoritmo enquanto a complexidade é apresentada na seção 5.5

5.1 INSERÇÃO DE ARESTAS

Um conjunto E' de arestas deve ser inserido, sendo que todas elas têm uma das extremidades ligada a um dado vértice $w \in V$. Este vértice w será aqui chamado "centro de inserção".

Se o número de elementos do conjunto C for maior ou igual que um parâmetro t (que também será discutido em detalhes posteriormente), a fase atual de execução do algoritmo é encerrada e todas as estruturas de dados e conjuntos auxiliares são reiniciados.

Se $|C| < t$, o vértice w é adicionado ao conjunto C. Uma vez o vértice w adicionado ao conjunto C, os primeiros k níveis das árvores de caminho mínimo de e para todos os vértices pertencentes ao conjunto C são calculados e/ou mantidos. Estas árvores são geradas ou mantidas com o algoritmo de Even e Shiloach, descrito em detalhes no capítulo 3. Para um vértice $w \in C$, chamaremos de $\check{T}_{in}(w)$ as árvores de caminho mínimo geradas até o vértice w e te $\check{T}_{out}(w)$ as árvores de caminho mínimo que foram geradas a partir do vértice w.

Note que a diferença entre as árvores T_{in} e $\check{T}_{in}$ (e também T_{out} e $\check{T}_{out}$) está no fato de que as árvores $\check{T}$ são mantidas pelo algoritmo decremental de Even e

Shiloach (EVEN; SHILOACH, 1981) enquanto as árvores T são totalmente recalculadas a cada operação de atualização.

Note também que a estrutura de dados do algoritmo decremental de Henzinger e King (HENZINGER; KING, 1995) inicializada no começo da fase atual de execução do algoritmo não é afetada pela inserção de arestas.

5.2 REMOÇÃO DE ARESTAS

Um conjunto E' de arestas deve ser removido, sendo que todas elas têm uma das extremidades ligada a um dado vértice $w \in V$. Este vértice w será aqui chamado "centro de remoção". A remoção do conjunto E' tem três passos:

1) As arestas de E' são removidas da estrutura de dados decremental do algoritmo de Henzinger e King, que foi inicializada no início da fase atual.
2) Para cada vértice *w* pertencente ao conjunto C, mantenha as árvores $\check{T}_{in}(w)$ e $\check{T}_{out}(w)$.
3) Para cada vértice *v* pertencente ao conjunto S, reconstrua as árvores $T_{in}(v)$ e $T_{out}(v)$.

5.3 CONSULTAS EM RODITTY E ZWICK

5.3.1 Consulta de distância entre vértices

O algoritmo de Roditty e Zwick é capaz de responder a consultas de distância, sendo que *dist(u,v)* representa a distância em peso ou número de arestas do vértice *u* até o vértice *v* na versão atual do grafo G. Uma consulta de distância segue 3 passos distintos, executados paralelamente.

O primeiro passo é uma consulta direta à estrutura do algoritmo decremental de Henzinger e King, definida no início da fase atual. Esta estrutura é atualizada em todas as operações de remoção de arestas (mas não

é alterada em operações de inserção). A resposta desta consulta será aqui chamada de R1.

Levando em conta que todas as operações de inserção são monitoradas pela estrutura de dados decremental, é seguro dizer neste momento que a resposta para **dist(*u*,*v*) ≤ R1**. Se nesta fase até o momento o conjunto C é vazio ou ainda se existe um caminho mínimo de *u* até *v* que não usa nenhuma aresta inserida na fase atual de execução do algoritmo, então **dist(*u*,*v*) = R1**. Neste caso, não há necessidade de calcular os passos seguintes.

O segundo passo consiste em achar um caminho mínimo de *u* a *v* que passe através de um dos centros de inserção do conjunto C, ou seja, um caminho mínimo que tenha sido reduzido entre um dos vértices da consulta (*u* ou *v*) e um dos vértices no conjunto C por conta de uma aresta adicionada em uma operação de inserção. Isto pode ser obtido da seguinte forma:

Para cada vértice *w* em C, verifica-se se a árvore $\check{T}_{in}(w)$ contém o vértice *u* (ou seja, se existe um caminho do vértice *u* até o vértice *w*, e este caminho tem tamanho ≤ k). Em caso afirmativo, o algoritmo checa se a árvore $\check{T}_{out}(w)$ contém o vértice *v* (ou seja, se existe um caminho partindo do vértice *w* até o vértice *v*, e este caminho tem tamanho ≤ k). Se as duas respostas são afirmativas para um vértice *w*, é criada uma possível resposta $\mathbf{R2_w}$, definida por: **dist(*u*,*w*) + dist(*w*,*v*)**. Uma resposta $R2_w$ está limitada por $\lceil 2k \rceil$, porém só é garantida até a distância k.

Se o vértice *u* não está contido em $\check{T}_{in}(w)$ e/ou o vértice *v* não está contido em $\check{T}_{out}(w)$, o vértice é desconsiderado e o algoritmo move para o próximo vértice em C.

A resposta "final" **R2** é obtida pela **mínima** distância obtida dentre todas as respostas $R2_w$ calculadas com sucesso (ou seja, com o vértice u contido em $\check{T}_{in}(w)$ e o vértice *v* contido em $\check{T}_{out}(w)$). Assim como no passo 1, é seguro dizer que **dist(*u*,*v*) ≤ R2**.

O terceiro passo consiste em encontrar um caminho mínimo de *u* a *v* que passe por um dos vértices do conjunto S. De maneira análoga ao segundo passo, para cada vértice *w* em S, a árvore $T_{in}(w)$ é checada para verificar a

presença do vértice *u*. Em caso afirmativo, a árvore $T_{out}(w)$ é checada pela presença do vértice *v*, gerando possíveis repostas $\mathbf{R3_w}$. A resposta final **R3** é a mínima resposta $\mathbf{R3_w}$ encontrada.

Dessa forma, a resposta final para **dist(*u*,*v*) = min(R1,R2,R3).**

5.3.2 Consulta de alcançabilidade

O Algoritmo de Roditty e Zwick da forma em que foi originalmente apresentado não possui uma consulta de alcançabilidade. Uma proposta original de extensão do algoritmo é apresentada na seção 6.4.

5.4 DESCRIÇÃO FORMAL DE RODITTY E ZWICK

O algoritmo de Roditty e Zwick é descrito formalmente na Figura 11 reproduzindo a forma em que foi apresentado originalmente em “On Dynamic Shortest Paths Problems” **(RODITTY; ZWICK, 2011)**. Esta definição formal será estendida posteriormente.

Inicializa(G,k,t):

Inicializa-HK(G,t)
$C \leftarrow \varnothing$
$S \leftarrow$ Random(V, (cn ln n)/k)
Cria-Arvores(S)

Remover (E’):

$E \leftarrow E - E'$
Remove-HK(E’)
Para cada $v \in C$
Remove-Arvore($\check{T}_{in}(v)$, E’,k)
Remove-Arvore($\check{T}_{out}(v)$, E’,k)
Cria-Arvores(S)

Inserir (E’,v):

$E \leftarrow E \cup E'$
Se $|C| \geq t$ então Inicializa (G,k, t)
$C \leftarrow C \cup \{v\}$
Inicializa-Arvores($\check{T}_{in}(v)$, E, k)
Inicializa-Arvores($\check{T}_{out}(v)$, E, k)
Cria-Arvores(S)

Cria-Árvores(S):

Para cada $w \in S$
BuscaLargura($T_{in}(w)$,E)
BuscaLargura($T_{out}(w)$,E)

Dist(u,v):

R1 $\leftarrow$ Consulta-HK(u,v)
R2 $\leftarrow \min_{w \in C}$ Busca-Arvore($\check{T}_{in}(w)$,u) + Busca-Arvore($\check{T}_{out}(w)$,v)
R3 $\leftarrow \min_{w \in C}$ Busca-Arvore($T_{in}(w)$,u) + Busca-Arvore($T_{out}(w)$,v)
Retorna min(R1, R2, R3)

Figura 11 - Descrição formal do algoritmo de Roditty e Zwick

O algoritmo é inicializado pela chamada **Inicializa(G,k,t)**, onde G=(V,E) é o grafo inicial e k e t são os parâmetros de precisão do algoritmo. Esta chamada de inicialização é efetuada no início de cada fase do algoritmo, e representa uma reinicialização total de todas as estruturas, de forma não recursiva.

Uma inserção de um conjunto de arestas E’ em que todas elas incidem a um vértice v será efetuada por uma chamada a **Inserir (E’,*v*)**.

Uma remoção de um conjunto de arestas E’ é realizado chamando **Remover(E’)**.

Uma chamada a **Cria-Arvores(S)** é usada para recriar as árvores de caminho mínimo de e para todos os vértices do conjunto S.

A chamada **Inicializa-HK(G,t)** faz a inicialização da estrutura de dados mantida pelo algoritmo de Henzinger e King (HENZINGER; KING, 1995). A função **Consulta-HK** faz a consulta de distância entre vértices à estrutura do algoritmo de Henzinger e King.

A chamada **Random(V,$c\,n\,\ln n/k$)** cria o conjunto de vértices distintos escolhidos com alta probabilidade segundo Ullmann e Yannakakis (ULLMAN; YANNAKAKIS, 1991).

A chamada a **Cria-Árvores(S)** constrói as árvores de caminho mínimo $T_{in}(w)$ e $T_{out}(w)$ para todo w no conjunto S.

A chamada **Inicializa-Arvores($\check{T}_{in}(v)$, E, k)** inicializa a manutenção decremental dos primeiros k níveis de uma árvore de caminhos mínimos $\check{T}_{in}$ para o vértice v. Essa árvore é atualizada no caso de uma remoção de um conjunto E' de arestas quando **Remove-Arvore($\check{T}_{in}(v)$, E',k)** é chamada. As chamadas para árvores $\check{T}_{out}(v)$ são processadas de forma análoga.

As chamadas **BuscaLargura($T_{in}(w)$,E)** são usadas para criar árvores estáticas utilizando o algoritmo padrão de busca em largura. Distâncias nestas árvores são encontradas pelas chamadas **Busca-Arvore($T_{in}(w)$,*u*)** e seus equivalentes T_{out}.

Por fim, a função de consulta **Dist(*u*,*v*)** responde a distância entre um dado par de vértices (*u*,*v*), em um determinado estado de execução do algoritmo.

5.5 COMPLEXIDADE EM RODITTY E ZWICK

A complexidade das operações de inserção ou remoção do algoritmo de Roditty e Zwick é da ordem de $O(\frac{mn^2 \log n}{t^2} + km + \frac{mn \log n}{k})$, amortizado. E por

sua vez a complexidade de uma consulta de distância é da ordem de O($t + \frac{n \log n}{k}$) no pior caso.

Cada resultado obtido pelo algoritmo é correto com probabilidade de pelo menos $1 - 2n^{-c}$. Se o valor escolhido para a constante k for de $(n \log n)^{1/2}$ e $(n \log n)^{1/2} \leq t \leq n^{3/4}(\log n)^{1/4}$, então o tempo amortizado de atualização do algoritmo é de O $\left(\frac{mn^2 \log n}{t^2}\right)$.

A prova desses resultados pode ser verificada no artigo original de Roditty e Zwick (RODITTY; ZWICK, 2011).

É importante mencionar que a constante c é uma determinante do grau de precisão do algoritmo, pois influencia no número de vértices a serem escolhidos para o conjunto S é limitada por $0 < c \leq \frac{k}{\log n}$. Como c é uma constante arbitrária, ela é ignorada pelo cálculo de complexidade do algoritmo.

6 ANÁLISE CRÍTICA DO ALGORITMO DE RODITTY E ZWICK

O algoritmo de Roditty e Zwick possui várias componentes e a interação entre essas componentes não é trivial. Portanto é necessário que se faça uma análise da necessidade e importância de cada uma dessas componentes e de como elas se integram.

Adicionalmente, são propostas extensões no algoritmo original de forma a melhorar seu desempenho, qualidade do resultado e funcionalidade.

Na seção 6.1 será feita uma pequena análise de cada uma das três componentes que podem ser respostas na consulta de distância entre vértices. A seção 6.2 trata da escolha de vértices pivôs e possíveis problemas da abordagem totalmente aleatória. A seção 6.3 trata dos efeitos do reinício do algoritmo nas estruturas e respostas, e a seção 6.4 propõe uma consulta de alcançabilidade como extensão do algoritmo original e explora possíveis problemas desta nova consulta.

6.1 PROCESSOS DA CONSULTA DE DISTÂNCIA ENTRE VÉRTICES

O algoritmo de Roditty e Zwick mantém várias estruturas ao mesmo tempo, como parte dos algoritmos secundários que utiliza para manutenção ou com o intuito de facilitar consultas. O capítulo anterior mostrou que este algoritmo executa três diferentes processos para a consulta de distâncias entre vértices e o menor resultado dentre os três obtidos é considerado a resposta final. Uma análise de cada um dos três processos é apresentada a seguir como forma de melhor compreender a necessidade da existência de cada um deles e as situações em que a resposta final vem de cada um dos três processos.

Para as seções 6.1.1 a 6.1.3, assuma uma consulta dist(u,v) de acordo com o algoritmo formalizado na seção 5.4, e que existe um caminho P de u a v, cujo tamanho é igual à distância de u a v.

6.1.1 Resposta R1 – Consulta direta à estrutura de Henzinger e King

A resposta R1 do algoritmo de Roditty e Zwick é obtida de uma consulta direta à estrutura árvores mantida pelo algoritmo de Henzinger e King no grafo G. O algoritmo de Henzinger e King é um algoritmo decremental, ou seja, só vai atualizar suas estruturas em caso de remoção de arestas, ignorando as mudanças no grafo causadas por inserções de arestas. Este algoritmo tem a vantagem de ter baixo tempo de consulta, porém sua natureza decremental se torna uma desvantagem quando inserções de arestas afetam o caminho P, pois o algoritmo não fica ciente de tais inserções. Consideremos os seguintes casos:

a) O caminho P não contém nenhuma aresta inserida desde a última inicialização:
   - P pode ser retornado por R1 ou $|R1| = |P|$
b) O caminho P contém arestas inseridas desde a última inicialização:
   - P não é retornado por R1, porém $|R1| \geq |P|$. Isto se deve ao fato de que mesmo com arestas inseridas, ainda pode existir um caminho alternativo com o mesmo tamanho de P, como mostra a Figura 12.

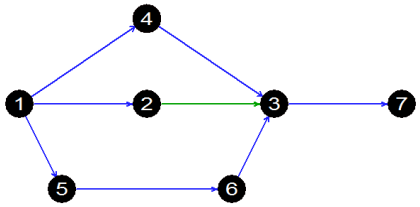

Figura 12 - Resposta R1 não se altera após inserção.

A Figura 12 mostra a inserção da aresta (2,3). Considere um caminho entre os vértices 1 e 7. Apesar da resposta R1 não conseguir enxergar a aresta inserida e utilizá-la para a obtenção da resposta, existe outro caminho de mesma distância, equivalente ao caminho P.

Quanto mais inserções de arestas, maior o número de caminhos que passarão a usar estas novas arestas, e por consequência direta, são menores as chances de P ser retornado por R1.

6.1.2 Resposta R2 – Buscas nas árvores dinâmicas dos pontos de inserção

A segunda possível resposta do algoritmo de Roditty e Zwick vem de árvores criadas nos Pontos de Inserção, ou seja, nos vértices que recebem novas arestas. O método consiste em manter um par de árvores $Ť_{in}$ e $Ť_{out}$ com o algoritmo dinâmico de Even e Shiloach.

Um ponto interessante a mencionar, é que embora as árvores sejam mantidas tendo como origem cada um dos pontos de inserção, o algoritmo de Even e Shiloach é decremental, ou seja, só vai processar a remoção de arestas em suas árvores, ignorando as inserções. A Figura 13 demonstra isso de forma mais clara:

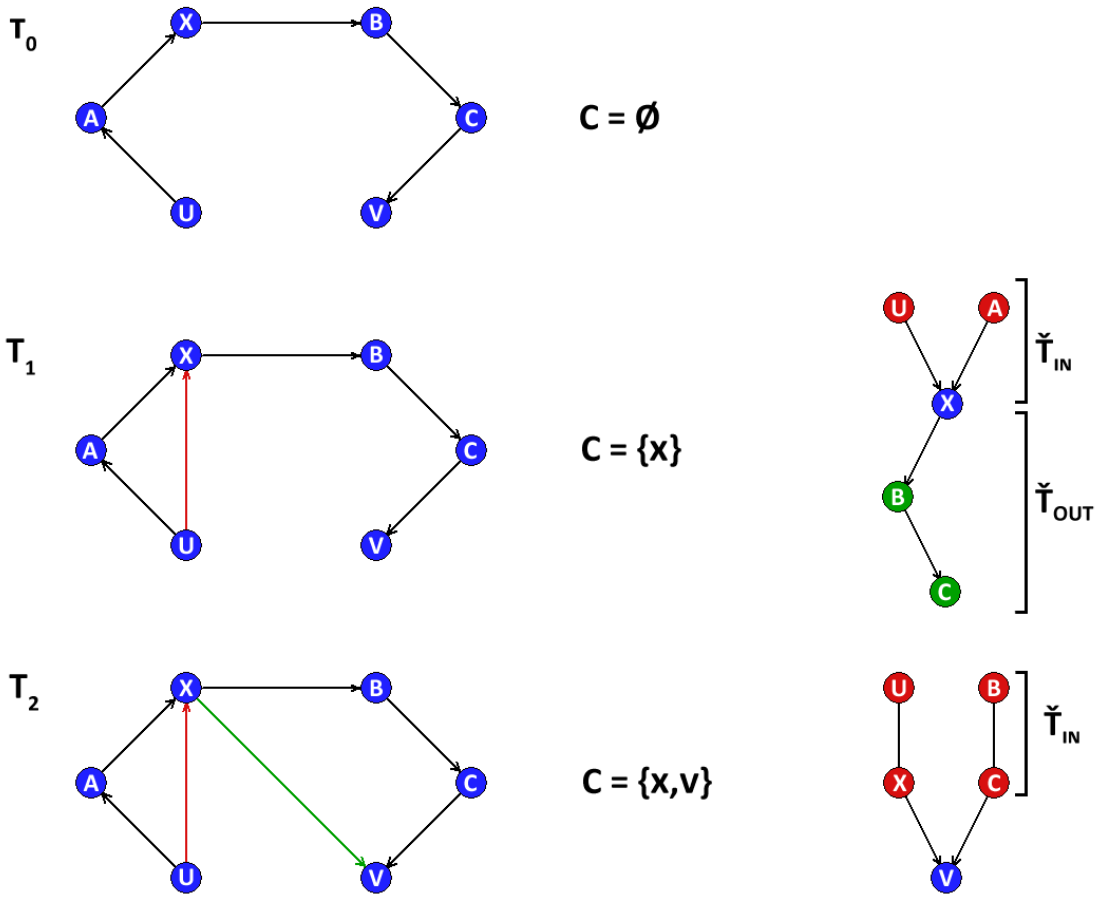

Figura 13 - Inserções de arestas afetando a Resposta R2 para *k*=2.

Na Figura 13, as marcas de tempo T_1 e T_2 representam duas inserções sequenciais em um grafo. A inserção da aresta (*u*,*x*) em T_1 coloca o vértice *x* no conjunto C, e criando árvores $\check{T}_{in}(x)$ e $\check{T}_{out}(x)$ para o vértice *x*. Como k=2, não existe caminho entre *u* e *v* na união das árvores $\check{T}_{in}(x)$ e $\check{T}_{out}(x)$, pois o caminho mínimo entre *u* e *v* no grafo em T_1 tem tamanho maior que *k*. No tempo T_2, a aresta (*x*,*v*) é inserida e uma árvore $\check{T}_{in}(v)$ é criada para o vértice *v* ($\check{T}_{out}(v)$ só contém o vértice *v*). Nesse ponto, as árvores conseguem enxergar o caminho P entre *u* e *v*. Note que ao fazer a inserção da aresta (*x*,*v*) as árvores do tempo T_1 relativas ao vértice *x* não sofrem alteração, pois o algoritmo de Even e Shiloach, usado para manutenção das árvores é decremental. Formalizando:

> ***TEOREMA 3:*** *Se P contém arestas inseridas e |P| ≤ k, então P está contido na união das árvores $\check{T}_{in}$ e $\check{T}_{out}$ do último vértice de P a entrar no conjunto C.*

Demonstrando o Teorema 3: Se P contém arestas inseridas, cada uma destas arestas tem pelo menos um vértice em C. Logo em P tem pelo menos um vértice de C. Seja w o último vértice de P a entrar em C. Sejam *P1* e *P2* as partes de *P* antes e depois de *w*, respectivamente, incluindo w. Ou seja, $P1 = (u, x1, x2, \dots, xi, w)$ e $P2 = (w, y1, y2, \dots, yj, v)$. Como $|P| \leq k$, temos que $|P1| \leq k$ e $|P2| \leq k$. Nenhuma das arestas em *P1* e *P2* foi inserida depois que $\check{T}_{in}(w)$ e $\check{T}_{out}(w)$ foram inicializadas. Como $\check{T}_{in}(w)$ e $\check{T}_{out}(w)$ suporta remoções, então o caminho *P1* está em $\check{T}_{in}(w)$ e o caminho *P2* está em $\check{T}_{out}(w)$. Logo *P* está contido na união de $\check{T}_{in}(w)$ e $\check{T}_{out}(w)$.

A Resposta R2 tem um rápido tempo de consulta, porém tem a limitação de que as distâncias d entre os vértices devem ser menores ou iguais a k para a garantia da resposta. Apesar disso, distâncias $k < d \leq 2k$ podem ser encontradas, mas dependem diretamente do posicionamento do centro de inserção entre os vértices *u* e *v*. Respostas com caminhos de distância superior a 2*k* nunca serão encontradas pela Resposta R2.

6.1.3 Resposta R3 – Buscas nas árvores estáticas dos pivôs

A terceira possível resposta para o algoritmo de Roditty e Zwick é gerada pelas árvores de busca em largura T_{in} e T_{out} geradas em cada um dos pivôs selecionados no início da execução da fase atual do algoritmo. Uma boa escolha desse conjunto aleatório de vértices seguindo o Teorema 2 faz com que exista uma alta probabilidade de que um caminho P entre dois vértices quaisquer passe por pelo menos um vértice do conjunto de pivôs, o conjunto S.

O grau de precisão da resposta R3 é diretamente dependente do grau de precisão escolhido para o algoritmo, do número de pivôs escolhidos e sua disposição no grafo.

Como as árvores T_{in} e T_{out} são geradas utilizando o algoritmo de busca em largura para todos os outros vértices alcançáveis do grafo a partir de cada um dos pivôs, então o caminho mais curto entre *u* e *v* associado a este pivô é do mesmo tamanho que P. Se não existe caminho entre *u* e *v*, uma resposta negativa também é definitiva. Porém existem dois casos onde a resposta R3 apresenta falhas:

a) Se nenhum caminho mínimo entre *u* e *v* passa por qualquer dos pivôs, porém existe ao menos um caminho P’ não mínimo que passe por um dos pivôs. Então a resposta R3 será um caminho maior que P.
b) Se nenhum caminho entre *u* e *v* passa por qualquer dos pivôs. Neste caso R3 não poderá encontrar um caminho, mesmo que ele exista. Apesar de extremo este caso é possível, pois podem existir regiões no grafo que não são alcançáveis pelas árvores associadas aos pivôs que são obtidas pela resposta R3, ao menos até que o ciclo do algoritmo se reinicie. Este cenário extremo de pior caso será mais bem explorado na seção 6.4.2.

Como a resposta R3 garante as respostas positivas, mas não consegue efetivamente garantir respostas negativas, esta resposta é do estilo Monte Carlo. Outros fatores podem influenciar positiva ou negativamente a resposta, como a disposição dos pivôs, o número de pivôs selecionados para o conjunto S. A seção 6.2 demonstrará melhor a questão da disposição dos pivôs.

6.2 ESPARSIBILIDADE DOS PIVÔS

Na resposta R3 do algoritmo de Roditty e Zwick, é parte do algoritmo selecionar um conjunto aleatório de vértices (aqui chamados de pivôs) para compor o conjunto S. Os vértices desse conjunto são aleatoriamente escolhidos a cada início de ciclo do algoritmo como descrito no início do capítulo 5.

Analisando brevemente possíveis disposições dos vértices pivôs no desenho geral do grafo, é possível conjecturar que um conjunto de pivôs mais esparso pode dar mais chances de que um caminho P qualquer passe por um dos pivôs. Por outro lado, uma má distribuição de pivôs que cause uma concentração de pivôs em uma determinada área do grafo afeta negativamente a probabilidade de que um caminho que não passe por esta área de ser corretamente identificado pela resposta R3.

A Figura 14 demonstra o problema de um conjunto de pivôs que desfavorece a obtenção da resposta R3. Considere os vértices em verde como sendo o conjunto aleatório de pivôs selecionado para um determinado ciclo de execução do algoritmo. Caminhos entre pares de vértices como (8,23) ou (15,9) podem não ser encontrados por conta de uma má distribuição dos pivôs. Por outro lado uma distribuição mais esparsa pode permitir que um número maior de caminhos mínimos sejam encontrados.

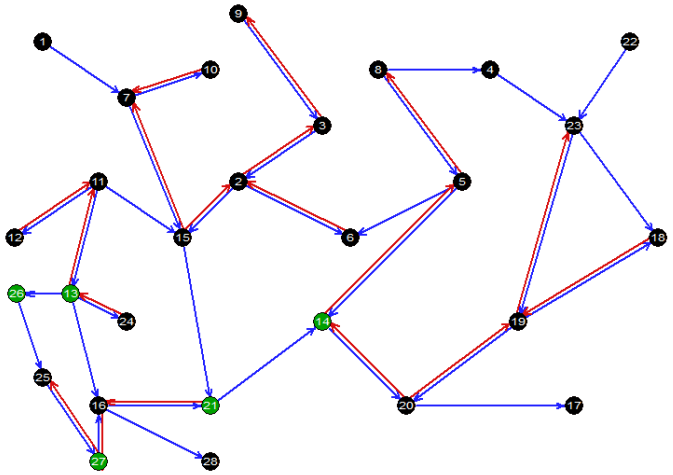

Figura 14 - Uma escolha aleatória desfavorável de pivôs para R3

Embora existam diversos estudos sobre como extrair versões mais esparsas de um grafo como o *Sparsification* de Eppstein e Frederickson (EPPSTEIN, *et al.*, 1997) que podem representar saídas inteligentes para a melhor escolha dos vértices do conjunto S, este trabalho não vai se aprofundar nos pormenores de uma escolha direcionada de pivôs. Ao invés disso, este trabalho propõe um algoritmo simples que pode ser implementado como uma extensão da busca em largura feita no início de cada ciclo de Roditty e Zwick, projetada para não impactar a complexidade das buscas, mas ao mesmo tempo impedir que vértices adjacentes ou muito próximos sejam ambos pivôs, resultando em árvores T_{in} e T_{out} de um vértice sejam muito semelhantes às do outro, efetivamente diminuindo a variabilidade do conjunto de árvores e por consequência as probabilidades de um caminho ser encontrado.

6.2.1 Seleção esparsa de Pivôs

Uma ideia simples de influência na seleção aleatória de pivôs é forçar a esparsibilidade da escolha dos próximos pivôs marcando os vizinhos dos vértices já escolhidos até certa profundidade *d*, de forma a impedir sua seleção como um novo pivô. Este mecanismo é simples o suficiente para ser implementado como extensão da criação das árvores de busca em largura efetuadas no ato da escolha de cada pivô sem necessariamente afetar a complexidade deste processo. A vantagem direta é prevenir que uma escolha pobre de pivôs (como por exemplo todos os pivôs juntos concentrados em uma parte do grafo) possa influir negativamente no desempenho do algoritmo.

Para tornar esse processo parte das buscas em largura é necessária uma modificação no algoritmo de Roditty e Zwick. Na inicialização do algoritmo os vértices do conjunto S são escolhidos todos de uma vez, e uma chamada a Cria-Árvores(S) encarrega-se da criação das árvores T_{in} e T_{out} para cada um dos vértices escolhidos. Na versão modificada, os vértices de S são escolhidos um a um e a cada escolha de um novo vértice como pivô *w*, as árvores $T_{in}(w)$ e $T_{out}(w)$ são geradas no momento da escolha.

Escolhido um determinado vértice w como pivô, o algoritmo de busca em largura é lançado a partir de *w* para todos os vértices alcançáveis do grafo G (criação da árvore T_{in}). Para cada vizinho de *w* detectado até a profundidade *d*, o algoritmo retira estes vértices do conjunto de possíveis escolhas para os próximos pivôs, efetivamente impedindo que estes sejam selecionados no futuro. O processo se repete na criação da árvore T_{out}, e por fim outro pivô é selecionado dentre os vértices restantes até que todos os pivôs necessários sejam escolhidos ou terminar a disponibilidade de vértices para escolha.

Também é importante mencionar que ao menos um pivô deve existir em cada componente conexa do grafo G. A seção 6.4 mostrará uma situação de pior caso do algoritmo de Roditty e Zwick relacionada com a falta de pivôs em componentes conexas e possíveis soluções para o problema.

6.3 CONDIÇÕES DE REINÍCIO

O algoritmo de Roditty e Zwick toma um parâmetro em sua inicialização que define um limite que ao ser alcançado determina o abandono de sua estrutura atual e uma nova inicialização, efetivamente refazendo todas as escolhas aleatórias do algoritmo e contornando o problema da degradação de desempenho devido ao aumento do custo de manutenção da estrutura após um determinado número de mudanças no grafo.

Um gráfico aproximado que monitora o desempenho do algoritmo de Roditty e Zwick é mostrado na Figura 15:

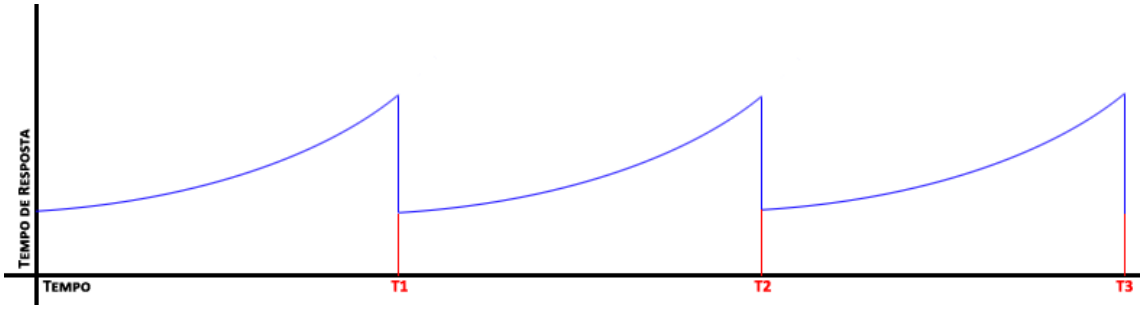

Figura 15 - Gráfico aproximado de desempenho de Roditty e Zwick, baseado em simulações do algoritmo.

Na Figura 15, a linha azul mostra o desempenho do tempo de resposta do algoritmo de Roditty e Zwick. Os marcadores de tempo T1, T2 e T3 mostram os momentos em que, durante uma operação de inserção (função **Inserir (E',v)**) , o número de elementos do conjunto C alcança o parâmetro pré-estabelecido t, e as estruturas do algoritmo são reinicializadas.

A obtenção das respostas R1, R2 e R3 são diretamente influenciadas pelo número de inserções e remoções de arestas no grafo e todas as três respostas apresentam ganhos com este reinício programado das estruturas.

A resposta R1 que é obtida por uma consulta direta à estrutura do algoritmo de Henzinger e King é prejudicada por inserções de arestas, como visto na seção 6.1.1, logo o reinício da estrutura permite que as arestas inseridas sejam incorporadas à estrutura do algoritmo de Henzinger e King e sejam vistas como arestas "iniciais" do grafo.

O reinício do algoritmo esvazia o conjunto C, que contém todos os vértices que se tornaram pontos de inserção, e por consequência apaga também as árvores $\check{T}_{in}$ e $\check{T}_{out}$ que são mantidas dinamicamente para cada um destes vértices. Como cada vértice adicionado ao conjunto C representa um par adicional de árvores dinâmicas sendo mantido pelo algoritmo, o esvaziamento do conjunto C e remoção das árvores dinâmicas representa um ganho direto de desempenho ao algoritmo. Nesse ponto, por definição a resposta $R2 = \infty$ até que sejam realizadas novas inserções de arestas.

Para a resposta R3, o reinício da estrutura afeta diretamente o conjunto S de pivôs. O conjunto S atual é esvaziado e um novo conjunto com diferentes vértices é escolhido, para então serem criados os conjuntos de árvores de busca em largura T_{in} e T_{out} para cada novo elemento de S. Embora a mudança no conjunto não signifique necessariamente um ganho de desempenho, uma redistribuição dos pivôs permite uma readequação das probabilidades de um caminho entre dois vértices quaisquer de passar por ao menos um dos pivôs, como enunciado no Teorema 2. Adicionalmente, oferece uma oportunidade para a escolha de ao menos um pivô em cada componente conexa do grafo, satisfazendo os requisitos da seção 6.2.

Uma possibilidade totalmente nova para alterar as condições de reinício é substituir o parâmetro t constante por uma versão dinâmica, que se altera a cada ciclo de execução do algoritmo baseando-se na quantidade e qualidade de respostas encontradas. Uma das possíveis maneiras de implementar este método é obter estatísticas em tempo de execução sobre quantas respostas são encontradas por cada um dos métodos R1, R2 ou R3 e gerar uma lógica que permita ao algoritmo inferir quando a maioria das respostas a consultas deixam de vir de R1 e R2 (que possuem melhor grau de precisão na resposta).

Esta inteligência agregada ao algoritmo pode auxiliar na decisão de adiantar uma reorganização das estruturas ou ainda permanecer mais tempo com as estruturas atuais se estas estiverem favoráveis a sequencia de consultas que está sendo executada no momento.

Esta abordagem, embora interessante, não será explorada neste trabalho, sendo sugerida como uma possibilidade de trabalhos futuros.

6.4 CONSULTA DE ALCANÇABILIDADE PARA RODITTY E ZWICK

O algoritmo original apresentado por Roditty e Zwick possui consulta para distância entre vértices, porém não continha uma consulta para simples verificação da alcançabilidade entre vértices. Como o algoritmo em questão foi idealizado para ser utilizado em grafos que podem ter (ou vir a ter) mais de uma componente conexa, faz-se necessária a criação de uma consulta que possa atestar eficientemente se um dado vértice pode ser alcançado por outro seguindo os preceitos do algoritmo original.

Um ponto importante a mencionar é que em grafos direcionados, estar na mesma componente conexa não é condição suficiente para determinar alcançabilidade de um par de vértices, sendo também necessário um caminho com arestas direcionadas entre o vértice de origem e o vértice de destino.

Uma proposta de consulta de alcançabilidade complementar ao algoritmo de Roditty e Zwick é mostrada na Figura 16:

Conexo(u,v):
Se ∃ Consulta-HK(u,v) então **SIM**. senão:
Se ∃ Busca-Arvore($\check{T}_{in}$(w),u) + Busca-Arvore($\check{T}_{out}$(w),v) então **SIM**, senão
Se ∃ Busca-Arvore(T_{in}(w),u) + Busca-Arvore(T_{out}(w),v) então **SIM**, senão **NÃO**.

Figura 16 - Consulta de alcançabilidade para Roditty e Zwick

A função **Conexo(*u*,*v*)** determina uma forma de efetuar uma consulta simples de conectividade entre os vértices *u* e *v*. Inicialmente é feita uma consulta direta na estrutura do algoritmo decremental de Henzinger e King, definida no início da fase atual.

Se a resposta for positiva, os vértices definitivamente tem um caminho qualquer entre si. Caso a resposta seja negativa, lembrando que o algoritmo de Henzinger e King é Monte Carlo (ou seja, não consegue precisar respostas negativas), esta resposta é incerta e força a execução do passo seguinte.

Uma busca de conectividade análoga à da consulta de distância é efetuada para cada elemento do conjunto C (centros de inserção), verificando a

existência do vértice *u* em alguma das árvores $\check{T}_{in}$ e, caso encontrado, verifica a existência do vértice *v* na árvore $\check{T}_{out}$. Caso os dois vértices sejam encontrados, uma resposta afirmativa definitiva é retornada e a função é paralisada. Porém como as árvores $\check{T}_{in}$ e $\check{T}_{out}$ são mantidas dinamicamente com profundidade ≤ *k*, este passo é incapaz de prover uma resposta negativa definitiva se a distância (*u*,*v*) para todos os vértices do conjunto C for superior a *k*[1] , forçando a execução do passo seguinte.

O terceiro passo da consulta envolve uma busca nas árvores de busca em largura T_{in} para cada elemento do conjunto S (pivôs). Quando um dos processos obtiver uma resposta positiva onde uma árvore T_{in} contém o vértice *u* e uma T_{out} contém o vértice *v*, os dois processos são finalizados e a uma resposta afirmativa é retornada. Como as árvores T_{in} e T_{out} são geradas de cada elemento do conjunto S para todos os outros vértices do grafo G, um resultado negativo neste passo tem alta probabilidade de certeza em afirmar que não há caminho entre os vértices. A seção 6.4.2 mostra porque um resultado negativo da consulta não é definitivamente uma prova de não alcançabilidade entre os vértices.

6.4.1 Complexidade da Consulta de Alcançabilidade

A consulta de alcançabilidade aqui descrita foi projetada para reutilizar as estruturas de dados existentes e é uma derivação direta da consulta de distância entre vértices da seção 5.3.1, portanto a complexidade de ambas é semelhante e pode ser encontrada na seção 5.5.

Esta consulta de alcançabilidade é boa, considerando-se que é uma extensão de funcionalidade do algoritmo original. Se o intuito for apenas obter a alcançabilidade, outros métodos já conhecidos podem ser utilizados, como a obtenção do fecho transitivo do grafo direcionado e obter a alcançabilidade no grafo pelo isolamento de suas componentes conexas (AHO; GAREY; ULLMAN, 1972).

[1] Embora a consulta no passo 2 seja capaz de atestar entre dois vértices até a distância 2k, só é possível garantir que a distância será corretamente se $d(u,v) \leq k$. Para distâncias $k < d \leq 2k$ a distância pode ou não ser encontrada, mas depende diretamente do posicionamento do centro de inserção em relação aos vértices u e v.

6.5 PROBLEMAS DE ALCANÇABILIDADE

A consulta de alcançabilidade aqui definida, embora tenha sido desenhada para tirar o melhor proveito das estruturas existentes do algoritmo, expõe um pior caso do algoritmo de Roditty e Zwick, que envolve caminhos entre vértices que embora existam, não podem ser alcançados por nenhum dos três processos independentes de consulta.

Para melhor definir o problema e apontar possíveis correções, é necessário fazer algumas definições adicionais sobre o conjunto S de pivôs e suas relações com os caminhos no grafo.

Um vértice *x* **cobre** um par ordenado de vértices (*u*,*v*) se *x* está em um caminho qualquer de *u* a *v*. Se *x* está em um caminho mínimo de (*u*,*v*), então dizemos que *x* **cobre minimamente** (*u*,*v*).

Suponha que as árvores $T_{in}(x)$ e $T_{out}(x)$ foram calculadas. Pela definição do algoritmo, se *x* cobre (*u*,*v*), então *u* está $T_{in}(x)$ e *v* está em $T_{out}(x)$, ou seja, existe um caminho de *u* a *v* em $T_{in}(x) \cup T_{out}(x)$. Além disso, se *x* cobre minimamente (*u*,*v*), este caminho é mínimo.

O conjunto de pivôs S cobre um par ordenado de vértices (*u*,*v*) se algum vértice *x* em S cobre (*u*,*v*). Analogamente S cobre minimamente (*u*,*v*) se algum vértice *x* em S cobre minimamente (*u*,*v*).

Se S cobre (*u*,*v*) então o algoritmo é capaz de encontrar um caminho de *u* a *v* pela reposta R3, porém a resposta R3 só será mínima se S cobre minimamente (*u*,*v*).

Se S não cobre (*u*,*v*), mas existe um caminho P entre *u* e *v*, a resposta R3 é incapaz de encontrar um caminho, e a consulta de alcançabilidade vai retornar uma resposta incorreta.

Dessa forma, supondo que:

a) Exista um caminho de *u* a *v*, e

b) Que em todos os caminhos mínimos de *u* a *v* existe ao menos uma aresta inserida durante o ciclo atual de execução, e
c) Distância entre (*u*,*v*) > *k*

Temos duas situações interessantes distintas:

1) Se for desejado o caminho mínimo de *u* a *v*, S cobre (*u*,*v*) mas não cobre minimamente;
2) Se for desejado saber se *u* alcança *v*, mas S não cobre (*u*,*v*).

No caso 1, a resposta R1 não será mínima por conta das arestas inseridas, a resposta R2 pode não ser mínima, pois a distância entre os vértices supera a distância k que regula a profundidade das árvores dinâmicas a resposta R3 não será mínima, pois nenhum caminho mínimo de u a v pode ser encontrado nas árvores T_{in} e T_{out}.

No caso 2, a resposta R1 será um falso negativo, a resposta R2 pode ser um falso negativo (será certamente falso negativo se dist(*u*,*v*) >= 2*k*) e a resposta R3 será um falso negativo, pois nenhum caminho de *u* a *v* pode ser encontrado nas árvores T_{in} e T_{out}.

Este trabalho apresenta duas propostas distintas para tratamento das situações 1 e 2 apresentadas, descritas nas seções 6.5.1 e 6.5.2. Ambas as propostas envolvem modificações no conjunto de pivôs S de forma a promover pequenas mudanças no conjunto e melhorar as probabilidades de S cobrir minimamente todo o grafo.

6.5.1 Correção dinâmica do conjunto S

No momento da escolha do conjunto de vértices S, com tamanho $\frac{(cn \ln n)}{k}$, o algoritmo que faz a escolha dos vértices precisa garantir de alguma forma que todo vértice do conjunto V esteja sempre contido em ao menos uma árvore do tipo T_{in} e também em ao menos uma árvore do tipo T_{out}. Dessa forma, independente da distância entre os vértices consultados, a alcançabilidade

entre eles será corretamente identificada. Para solucionar esse problema é proposta uma modificação para a função **Cria-Árvores(S).**

A cada chamada à função **Cria-Árvores(S)**, antes de iniciar a criação das árvores de busca em largura, um par de conjuntos temporário L_{in} e L_{out} é criado, e são inicialmente vazios. Durante a criação das árvores de busca em largura T_{in} e T_{out} para cada vértice do conjunto S, cada vértice visitado pelo algoritmo de busca em largura para a criação de uma árvore T_{in} é colocado no conjunto L_{in}, e analogamente para cada vértice visitado pelo algoritmo de busca em largura para a criação de uma árvore T_{out}, este vértice é colocado no conjunto L_{out}. Quando o processo de criação das árvores de busca em largura for concluído, os conjuntos L_{in} e L_{out} devem ser comparados com o conjunto V.

Se existirem vértices que pertencem a V, mas não pertencem a L_{in} ou L_{out}, é um indicativo de vértices que não são cobertos pelo conjunto S.

Se forem encontrados vértices pertencentes a V, mas não a um dos conjuntos L_{in} ou L_{out}, um desses vértices deve ser escolhido aleatoriamente a partir de $V - (L_{in} \cup L_{out})$, e este vértice é adicionado ao conjunto S. O próximo passo é gerar as árvores T_{in} e T_{out} de busca em largura a partir deste vértice, atualizando os conjuntos L_{in} e L_{out} para todo vértice encontrado na geração das árvores de busca em largura. Ao final da geração de todas as árvores T_{in} e T_{out}, uma nova comparação entre os conjuntos L_{in}/L_{out} e V é feita de forma a procurar por vértices adicionais não cobertos, e o processo é então repetido até que $L_{in} = L_{out} = V$.

A nova função **Cria-Árvores(S)** é formalizada na Figura 17.

```
Cria-Árvores(S)
        L_in ←∅
        L_out ←∅
        Para cada w em S
                BuscaLargura(T_in(w),E)
                Coloque em L_in todos os vértices de T_in(w)
                BuscaLargura(T_out(w),E)
                Coloque em L_out todos os vértices de T_out(w)
        Fim
        Enquanto (L_in != V) | (L_out != V)
                Escolha x em ((V - L_in)∪(V - L_out))
                Faça S ← S ∪ {x}
                BuscaLargura(T_in(x),E)
                Coloque em L_in todos os vértices de T_in(x)
                BuscaLargura(T_out(x),E)
                Coloque em L_out todos os vértices de T_out(x)
        Fim
Fim
```

Figura 17 - A nova função Cria-Árvores(S)

Das duas situações de interesse descritas na seção 6.5, apenas a segunda é coberta por esta solução, ou seja, esta solução garante que S cobre todo vértice no grafo G, mas não pode garantir que S cobre minimamente quaisquer pares conexos de vértices. A consequência direta disso é que esta solução pode ser utilizada para verificação de alcançabilidade entre vértices, mas não resolve totalmente o problema de caminhos mínimos.

É importante mencionar que a adição do loop que preenche os conjuntos L_{in} e L_{out} pode impactar o tempo do algoritmo.

6.5.2 Adições ao conjunto S por inserções de arestas.

Um dos aspectos não explorados no algoritmo original é a relação entre o conjunto S de pivôs e o conjunto C de centros de inserção. A solução descrita na seção 6.5.1 é de fácil implementação no algoritmo e resolve o problema da cobertura do grafo, mas não garante uma solução que deixe o grafo minimamente coberto.

Uma segunda solução é aqui proposta, que lida melhor com a cobertura mínima do grafo à medida que inserções são feitas.

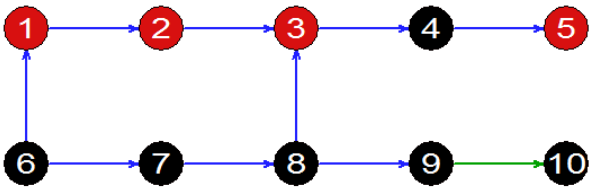

Figura 18 - O pior caso de Roditty e Zwick

Na Figura 18 considere os vértices em vermelho como pivôs, pertencentes ao conjunto S, e escolhidos ao início do ciclo atual. As arestas azuis são preexistentes no grafo e a aresta (9,10) em verde acabou de ser inserida. Deseja-se saber o caminho mínimo entre os vértices 6 e 10, considerando *k*=3. Considere que existe um caminho mínimo P entre os vértices 6 e 10. A Figura 19 mostra os caminhos mínimos T_{in} e T_{out} obtidos a partir de cada um dos pivôs disponíveis no grafo.

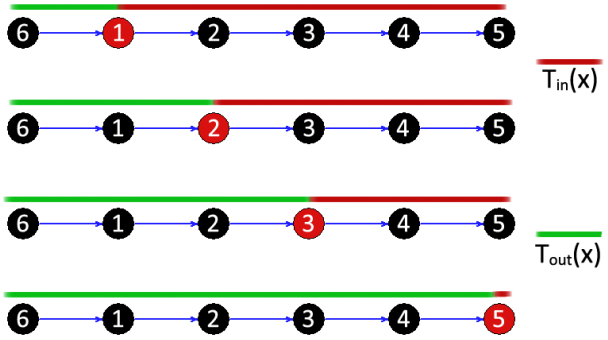

Figura 19 - Junções de T_{in} e T_{out} para todos os pivôs do grafo

Veja que devido ao posicionamento dos pivôs no grafo, nenhum deles é capaz de ver o caminho mínimo entre (6,10). Formalizando:

a) A resposta R1 é incapaz de retornar o caminho entre os vértices 6 e 10 devido a este caminho passar por uma aresta inserida;
b) A resposta R2 é incapaz de retornar o caminho entre os vértices 6 e 10 utilizando membros do conjunto C, pois k=3, e não existe um caso em que o vértice 6 esteja em uma árvore $\check{T}_{in}$ e o vértice 10 esteja em uma árvore $\check{T}_{out}$ de um membro do conjunto C (no caso, apenas o vértice 10).

c) A resposta R3 é incapaz de retornar o caminho entre 6 e 10 pois nenhum pivô tinha ao mesmo tempo o vértice 6 em uma árvore T_{in} e o vértice 10 em uma árvore T_{out}.

Esta é uma demonstração do pior caso do algoritmo de Roditty e Zwick, com respostas negativas falsas. Para resolver este problema, a seguinte modificação é proposta: Para cada vértice v em que ocorreu uma inserção de arestas, além de adicioná-lo ao conjunto C para que sejam mantidas árvores dinâmicas $\check{T}_{in}(v)$ e $\check{T}_{out}(v)$ a partir dele, também se adicionará o vértice v ao conjunto S, tornando-o também um pivô e imediatamente calculam-se as árvores $T_{in}(v)$ e $T_{out}(v)$. Esta alteração faz com que o menor caminho P esteja na união das árvores T_{in} e T_{out} do último centro de inserção.

Formalizando a ideia acima, apresentamos o Teorema 4:

TEOREMA 4: *Se calcularmos $T_{in}(y)$ e $T_{out}(y)$ do último centro de inserção y em P imediatamente após a inserção das arestas, então*
$\exists\ \boldsymbol{P'} \subseteq \boldsymbol{T_{in}}(\boldsymbol{y}) \cup \boldsymbol{T_{out}}(\boldsymbol{y})$ *tal que* $|\boldsymbol{P'}| = |\boldsymbol{P}|$.

Prova do Teorema 4: Como *y* é o último centro de inserção em P no momento do cálculo de $T_{in}(y)$ e $T_{out}(y)$, todas as arestas de P estão no grafo. Logo as duas buscas em largura resultam em P, ou outro caminho mínimo P' do mesmo comprimento de P.

7 CONCLUSÃO

O algoritmo de Roditty e Zwick se mostrou uma opção elegante e poderosa para a manutenção de grafos dinâmicos, com características interessantes como a utilização de processos paralelos em suas operações de consulta e a ideia de ciclos de reinício do algoritmo para compensação da degradação de desempenho e suas estruturas.

As vantagens do algoritmo de Roditty e Zwick residem principalmente no bom desempenho de suas consultas, aliado a um ótimo tempo de atualização de suas estruturas. A escolha de utilizar os algoritmos decrementais de Even e Shiloach e o de Henzinger e King como auxiliares tanto na manutenção de estruturas quanto como componentes da consulta assegurou baixos tempos de execução para operações individuais. Outra característica interessante é a possibilidade de regular o grau de precisão do algoritmo (com possível sacrifício de desempenho) por meio de variáveis.

Por outro lado, o algoritmo de Roditty e Zwick pode oferecer respostas imprecisas em determinados casos, e suas estruturas de dados requerem uma quantidade considerável de espaço em memória para serem mantidas. Escolhas aleatórias desfavoráveis em determinadas estágios de execução do algoritmo podem afetar diretamente o desempenho de uma fase, mas o mecanismo de reinício de estruturas busca compensar essas perdas.

Este trabalho mostra uma análise detalhada do algoritmo de Roditty e Zwick e seus algoritmos secundários, observando separadamente cada um dos seus subprocessos, e propondo modificações ao algoritmo original.

Uma das alterações aqui proposta visa evitar o pior caso de consulta de distância entre vértices, onde por uma distribuição desfavorável de pivôs, nenhuma das três alternativas é capaz de retornar uma resposta correta. Modificações pontuais na metodologia de escolha aleatória de pivôs e nos processos de manutenção da estrutura tornam possível contornar situações de pior caso até que aconteça o reinício das estruturas. Parte de essa alteração envolve uma determinada garantia de que os pivôs tenham certo grau de

esparsibilidade no grafo, então outra modificação é proposta: um algoritmo simples que aumenta as chances de escolha de pivôs de forma a induzir a escolha de um conjunto aleatório mais esparso.

Como adição ao algoritmo foi proposta uma consulta de simples alcançabilidade, que utiliza as estruturas de dados existentes e foram expostas vantagens e problemas desta consulta, bem como alternativas a sua utilização.

7.1 TRABALHOS FUTUROS

O algoritmo de Roditty e Zwick tem menos de um ano de publicação no tempo de escrita deste trabalho e ainda deixa margem para diversos tipos de pesquisa que não foram incluídos no escopo desta obra. Segue uma lista de possíveis linhas de pesquisas que podem originar-se a partir deste trabalho.

7.1.1 Testes de desempenho com casos reais

O algoritmo de Roditty e Zwick pode ser construído em código e executado com diferentes grafos e cenários de forma a gerar estatísticas de tempos de resposta e atualização das estruturas e confrontar seu desempenho teórico com o desempenho real.

Este exercício pode gerar resultados valiosos como melhores constantes de precisão, adaptações para induzir determinadas situações ao melhor caso ou ainda a identificação de situações que degradem o desempenho.

7.1.2 Dinamização das constantes determinantes de desempenho

Durante a análise inicial do algoritmo de Roditty e Zwick, consideramos por diversas vezes a transformação de determinadas constantes do algoritmo como a constante de precisão *c*, a constante de profundidade das árvores dinâmicas *k* ou ainda a constante *t* que determina o reinício das estruturas, de forma a adequar os valores ao tipo de grafo utilizado ou a determinadas

situações encontradas em tempo de execução. O propósito disso efetivamente é criar formas do o algoritmo “aprender” a balancear precisão e desempenho de acordo com o histórico de operações e resultados obtidos até o momento em sua execução.

Adicionando ao algoritmo a capacidade de coletar determinadas estatísticas em suas respostas ou ainda durante reorganizações de estruturas e utilizando tais estatísticas para mudar os valores das constantes de controle a cada ciclo pode melhorar ainda mais o desempenho do algoritmo em tempo de execução e ser ponto de partida para possíveis aumentos de precisão do algoritmo original.

7.1.3 Substituição de Algoritmos

Na manutenção do algoritmo de Henzinger e King existe espaço para a utilização de outros algoritmos secundários para a manutenção das árvores em si. No artigo original foi utilizado o algoritmo de Even e Shiloach, mas nada impede que este seja substituído por outros algoritmos de manutenção de caminhos mínimos em árvores dinâmicas, com possível ganho de desempenho.

7.1.4 Esparsificação para a escolha de pivôs

Conforme descrito no início do capítulo 6.2, o método *Sparsification* (EPPSTEIN, *et al.*, 1997) pode ser utilizado para obter-se uma versão mais esparsa de um grafo baseado em determinados critérios de “importância” dos vértices. O uso desta técnica para a obtenção de um conjunto mais esparso de pivôs que serão utilizados na resposta R3 pode resultar em maior precisão do algoritmo em menor tempo de execução.

REFERÊNCIAS

AHO, A.; GAREY, M. R.; ULLMAN, J. D. The Transitive Reduction of a Directed Graph. **SIAM Journal of Computing Sciences**. v. 1, p. 131-137, 1972.

DAINTITH, J. **A Dictionary of Computing**. 6a. [S.l.]: OUP Oxford, 2008.

DEMETRESCU, C.; FINOCCHI, I.; ITALIANO, G. F. **Dynamic Graphs**. Roma: Università di Roma, 2001.

DIESTEL, R. **Graph Theory**. 4a. [S.l.]: Springer, 2000.

EPPSTEIN, D. et al. Sparsification – A technique for speeding up dynamic graph algorithms. **Journal of Associated Computing and Mathematics.** p. 44:669–696, 1997.

EPPSTEIN, D.; GALIL, Z.; ITALIANO, G. F. Dynamic Graph Algorithms. **Algorithms and Theory of Computation Handbook**. 1999.

ESTRELLA-BALDERRAMA, A.; FOWLER, J.; KOBOUROV, S. G. GraphSET. **GraphSET - Simultaneous Embedding Tool** Disponível em: <http://graphset.cs.arizona.edu/>. Acessado em: 23/11/2011.

EVEN, S. **Graph Algorithms**. Maryland: Computer Science Press, 1979. 24 p.

EVEN, S.; SHILOACH, Y. An On-Line Edge-Deletion Problem. **Journal of the ACM**. p. 28(1):1-4, 1981.

FIEBRINK, R. Amortized Analysis Explained. **Princeton University** Disponível em: <http://www.cs.princeton.edu/~fiebrink/423/AmortizedAnalysisExplained_Fiebrink.pdf>. Acessado em: 17/06/2012.

GERLA, M.; BALTZER, J. C.; TSAI, J. T. C. Multiuser, Mobile, Multimedia Radio Network. **Wireless Networks**. p. 255–265, 1995.

HENZINGER, M. R.; KING, V. Fully Dynamic Biconnectivity and Transitive Closure. **Proc. of 36th FOCS**. p. 664-672, 1995.

HROMKOVIC, J. **Algorithms for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics.** London - Berlin - Heidelberg - New York: Springer Berlin Heidelberg, 2002.

KING, V.; G, S. **A Fully Dynamic Algorithm for Maintaining the Transitive Closure**. Proceedings of the 31st Annual Symposium on Theory of Computing - STOC'99. New York, NY, EUA: ACM. 1999. p. 492-498.

KNUTH, D. E. **The Art Of Computer Programming**. Boston: Addison-Wesley, v. 1, 1968.

RODITTY, L.; ZWICK, U. On Dynamic Shortest Path Problems. **Algorithmica Special Issue: European Symposium on Algorithms, Design and Analysis**. Volume 61, Setembro/2011.

SILJAK, D. D. **Dynamic Graphs**. International Conference on Hybrid Systems and Applications. Lafayette, LA: [s.n.]. 2006.

SZWARCFITER, J. L. **Grafos e Algoritmos Computacionais**. Rio de Janeiro: Campus, 1988.

TARJAN, R. E. Amortized Computational Complexity. **SIAM Journal of Algebraic and Discrete Methods**. v. 6, p. 306-318, 1985.

ULLMAN, J. D.; YANNAKAKIS, M. High Probability Parallel Transitive Closure Algorithms. **SIAM Journal of Computing**. p. 20:100-125, 1991.