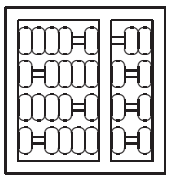

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Um Algoritmo Dinâmico para Árvore de Caminhos Mínimos

Daniel Felix Ferber *Arnaldo Vieira Moura*

Technical Report - IC-07-22 - Relatório Técnico

July - 2007 - Julho

Um Algoritmo Dinâmico para Árvore de Caminhos Mínimos

Daniel Felix Ferber Arnaldo Vieira Moura

31 de julho de 2007

Resumo

Propõe-se um algoritmo dinâmico para manter a árvore com raiz em um vértice especial origem e formada por caminhos mínimos para cada vértice do grafo. O grafo apresenta custo nas arestas e não apresenta restrições adicionais quanto ao domínio do custo, desde que os valores não sejam negativos. Aborda-se o problema com múltiplas operações simultâneas sobre o grafo: aumento e diminuição de custo das arestas. Sua complexidade de pior caso é $O(m + n \log n)$ para um grafo com n vértices e m arestas. O tempo de execução é menor ou igual à computação de uma nova árvore de caminhos mínimos. As operações sobre grafo são tratadas por um único modelo, similar ao algoritmo de Dijkstra.

Sumário

1 Introdução **2**
1.1 Objetivos . . . 3
1.2 Aplicações . . . 3
1.3 Trabalhos relacionados . . . 4

2 Conceitos Preliminares **6**
2.1 Considerações Gerais sobre o Grafo . . . 6
2.2 Descrição do Grafo . . . 7
2.3 Operações sobre o Grafo . . . 8
2.4 Descrição da Solução . . . 8

3 Algoritmo Dinâmico Simplificado **9**
3.1 Algoritmo de Dijkstra . . . 10
3.2 Estados dos Vértices . . . 11
3.3 Descrição do Algoritmo . . . 13
3.4 Pseudo Código para o Algoritmo Básico . . . 15
3.5 Formalização dos Estados de Vértice . . . 19
3.6 Condições para Relaxação de Arestas . . . 20

4 Algoritmo dinâmico **21**
4.1 Estados dos Vértices . . . 21
4.2 Condições para Relaxação de Arestas . . . 23
4.3 Arestas agendadas para relaxação . . . 23
4.4 Descrição do Algoritmo . . . 25
4.5 Heurísticas para Seleção de Vértice . . . 26
4.6 Pseudo Código para o Algoritmo Básico . . . 27
4.7 Lógica de atribuição de estados . . . 31
4.7.1 Lógica do procedimento AtualizaSucessores . . . 33
4.7.2 Lógica do procedimento RelaxaVizSucessora . . . 34
4.7.3 Observações sobre as tabelas . . . 36

5 Complexidade do Algoritmo **36**
5.1 Complexidade de espaço . . . 36
5.2 Complexidade de tempo . . . 37
5.3 Componentes afetados no grafo . . . 38
5.4 Comparação com algoritmos da bibliografia . . . 38

6 Conclusão **39**

1 Introdução

Na década de 1990, vários trabalhos dedicaram atenção ao problema de se manter uma estrutura dinâmica para a árvore de caminhos mínimos. Apesar do esforço dedicado à pesquisa, ainda permaneceram em aberto várias questões relativas ao problema dinâmico completo.

Uma pesquisa bibliográfica não revelou algoritmos que lidam simultaneamente com custos nas arestas e nos vértices, nem com alterações simultâneas sobre múltiplos componentes do grafo. Tampouco os algoritmos tratam todas as operações de remoção e adição de arestas ou vértices. Muitas

vezes, os casos de aumento e diminuição de custo são tratados como problemas isolados, e até incompatíveis para tratamento simultâneo. Alguns trabalhos impõem restrições quanto à direção das arestas ou restringem o domínio do custo de componentes do grafo. Outros algoritmos limitam-se ao problema de uma única modificação de custo. A questão de modificações múltiplas e simultâneas, quando mencionada, é tratada como caso especial e não é objeto de estudo mais aprofundado.

Este trabalho visa preencher algumas destas lacunas, permitindo modificações de custo (tanto aumento como diminuição) simultaneamente em múltiplas arestas.

1.1 Objetivos

Consideraremos um problema de caminho mínimo em grafos, formado pela união de duas generalizações de problemas de caminho estudados na literatura. O problema da *árvore de caminhos mínimos* (também conhecida como *SPT - shortest path tree*) busca os caminhos de menor custo de um vértice especial (raiz) para todos os demais vértices do grafo. O problema do *caminho mínimo dinâmico* atualiza uma solução já computada após modificações de custo, ou após operações de inserção e remoção de elementos no grafo.

Supõe-se a existência prévia de uma árvore de caminhos mínimos, calculada por algum algoritmo tradicional, como, por exemplo, o algoritmo de Dijkstra [6]. Esta árvore descreve o caminho de menor custo da raiz para cada vértice. Dada esta árvore, aplica-se ao grafo um conjunto de perturbações, como alterações no custo ou inserção ou remoção de componentes. Em seguida, é necessário atualizar a árvore de caminhos mínimos. O objetivo é restabelecer o caminho de menor custo da raiz até cada um dos demais vértices.

Propõe-se um algoritmo e uma estrutura de dados para manter e recalcular estes caminhos mínimos. A solução proposta aborda diversas restrições e limitações encontradas na literatura para o problema. Em particular, ela lida com as seguintes características:

- Custo não negativo em arestas e em vértices, sem restrições de integralidade;
- Modificações de custo: aumento e redução;
- Grafo sem restrições específicas, desde que conexo.

O principal destaque do algoritmo está na sua eficiência. Ele é capaz de atualizar toda a árvore de caminhos mínimos com uma única visita a cada vértice, mesmo para alterações simultâneas de topologia ou de custo sobre um número arbitrário de arestas ou vértices. A versatilidade do algoritmo não introduz estruturas de dados complexas, nem exige procedimentos elaborados. Um modelo simples e unificado é capaz de tratar de forma elegante todas as modificações no grafo. O algoritmo mantém-se relativamente simples para implementação.

1.2 Aplicações

Na Internet, o envio de pacotes de dados entre dois dispositivos segue um caminho determinado por tabelas de roteamento presentes nos dispositivos da rede. Estes dispositivos utilizam protocolos de comunicação específicos para calcular o custo de cada conexão, baseado em parâmetros de desempenho. Cada dispositivo calcula um caminho mínimo para todos os outros dispositivos importantes da rede. O conjunto de caminhos pode ser representado como uma árvore, através da qual se calcula a tabela de roteamento. Quando ocorre uma alteração na topologia da rede, como uma falha ou variação na carga sobre uma conexão, todos os dispositivos roteadores são informados sobre esse evento. Em implementações tradicionais, o dispositivo recalcula todos caminhos mínimos e não aproveita o conhecimento presente na rede antes da falha. Esta estratégia consome capacidade de processamento desnecessariamente e, assim, restringe a amplitude de cobertura dos dispositivos alcançados pelos roteadores. Se existirem múltiplas opções de rotas mínimas entre dois dispositivos, o cálculo de uma nova árvore de caminhos mínimos, sem conhecimento de rotas anteriores, pode escolher caminhos

mínimos diferentes a cada execução. Neste caso, seriam necessárias várias atualizações de tabelas de rotas.

Outra aplicação interessante para caminhos mínimos dinâmicos são sistemas interativos de auxílio à navegação, disponíveis atualmente em carros e computadores portáteis. Estes sistemas possuem um banco de dados das vias públicas e são capazes de identificar a posição do usuário (tipicamente por GPS). Quando o usuário seleciona a origem e o destino, o sistema de navegação sugere uma rota baseada em parâmetros como tempo de viagem, distância ou custo de pedágio e combustível. Estes sistemas recebem informações atualizadas periodicamente sobre o estado das vias, como por exemplo, congestionamentos ou trechos interditados. Uma vez verificada alguma dificuldade na rota prevista, o sistema deve determinar rapidamente um caminho alternativo até o destino. Também, em qualquer momento, o usuário tem a liberdade de se desviar da rota sugerida pelo sistema. Nestes casos a reação do sistema precisa ser imediata, sempre sugerindo novos caminhos a partir do ponto atual. Sem um algoritmo dinâmico eficiente, um sistema de navegação não apresentará a agilidade exigida para um sistema interativo dessa natureza.

Diversos outros problemas de computação recorrem ao algoritmo de caminhos mínimos, o qual é utilizado como subrotina para outros algoritmos mais complexos. Se o método de solução consulta freqüentemente a árvore de caminhos mínimos para realizar decisões, e também realiza alterações na topologia ou em custos de componentes da árvore, uma estrutura de dados junto com um algoritmo dinâmico poderá reduzir o tempo de processamento em algumas ordens de grandeza.

Um caso típico de algoritmos desta natureza são heurísticas GRASP (Greedy Randomized Adaptive Search Procedure). Em cada passo, na fase de construção da solução, o algoritmo realiza uma decisão local e aleatória. As possibilidades de decisão são determinadas através de uma heurística que estima as melhores decisões locais. O GRASP é adaptativo pois as possibilidades para o próximo passo são reavaliadas de acordo com a decisão atual. Uma ilustração é a construção de uma rede de serviço de comunicação, interligando todos os pontos a partir de um ponto de distribuição. A instalação de uma conexão entre dois pontos isolados gera um custo inicial elevado. No entanto, uma vez realizada uma conexão cara, outros pontos próximos podem ser atendidos pela conexão por um custo significativamente mais baixo. Na abordagem deste problema através de um algoritmo de busca GRASP, cada passo seleciona uma conexão entre dois vértices. As possibilidades são dadas pelas conexões de menor custo, estimadas por uma heurística que considera o caminho mínimo até o ponto de distribuição. No próximo passo, o algoritmo será obrigado a re-considerar o custo dos pontos próximos. Ao invés de recalcular a árvore de caminhos mínimos a cada passo, a heurística pode adotar a variante dinâmica para árvores de caminhos mínimos.

1.3 Trabalhos relacionados

A árvore de caminhos mínimos em um grafo de arestas com custos é um dos problemas melhor estudados em computação. A melhor implementação para o caso estático (quando os custos são invariantes) é dada pelo algoritmo de Dijkstra [6] com complexidade $O(m + n \log n)$ para um grafo direcionado com n vértices e m arestas, utilizando uma fila de prioridades de Fibonacci, conforme descrito em [4].

O estudo do problema dinâmico é relativamente recente. A maioria das publicações ocorreu na década de 90 e coincidem com a necessidade de algoritmos mais eficientes para roteamento de dados na Internet. A grande maioria da literatura recente, como por exemplo [14, 12, 13, 8, 11, 10, 22], recai sobre uma modificação do algoritmo de Dijkstra, diferenciando-se somente na estratégia adotada para propagar uma alteração no caminho mínimo para os vértices adjacentes.

Um tratamento teórico sobre a complexidade dos algoritmos dinâmicos em grafos é relatado em [22]. Um resultado importante é encontrado em [24, 14] para o problema de árvores de caminhos mínimos de uma única raiz e de arestas com custos positivos: nenhum algoritmo incremental que

realiza apenas inserção de arestas e que reaproveita somente a árvore de caminhos mínimos anterior, pode apresentar complexidade de tempo de execução melhor que o algoritmo estático.

Outra questão interessante, discutida em [22], é o número de vértices que são afetados pelas modificações resultantes do problema dinâmico. Para todas as instâncias possíveis, e todas as modificações dinâmicas, a complexidade de tempo de execução depende somente do número de elementos afetados. No entanto, não existe uma forma de prever quais elementos são afetados em conseqüência de uma determinada modificação. Para o problema de caminhos mínimos, a complexidade de pior caso é $O(m_a + n_a \log n_a)$, onde n_a é o número de vértices afetados (cujos caminhos mudaram de custo ou percurso) e m_a é o número de arestas incidentes em vértices afetados.

Limitantes melhores para o tempo médio de execução podem ser enunciados somente quando existem características específicas que evitam a ocorrência freqüente de certos casos patológicos. Podem ser características do problema, da estratégias para atualização de aresta, ou dos tipos de alterações permitidas. Um exemplo muito comum são trabalhos que assumem um grafo planar, ou que aceitam apenas operações de inserção ou de remoção de arestas.

Um grande número dos primeiros algoritmos propostos apresenta algum tipo de restrição quanto à dinâmica do problema, limitando-se a um subconjunto das operações de modificação do grafo. Por exemplo, [2] aceita apenas inserção de arestas de custo inteiro e de domínio limitado por um intervalo fixo. Outras soluções, como em [16], propõem algoritmos eficientes, com complexidade de tempo linear para grafos planares, às custas de um algoritmo complexo e impraticável, baseado numa partição topológica do grafo obtida através da aplicação recursiva de separadores descritos em [17].

Já os algoritmos apresentados em [12, 13, 14] não impõem restrições sobre o grafo (direção, ciclos ou laços) e os custos são números reais. Eles permitem tratar facilmente acréscimo e decréscimo de custo nas arestas. A remoção e inserção de arestas são casos especiais de acréscimo e decréscimo de custos. A solução é uma variante do algoritmo de Dijkstra, baseado em uma heurística que seleciona, para cada vértice, apenas um subconjunto dos vizinhos que potencialmente são afetados por novos caminhos. Para suporte à heurística, em cada vértice, duas filas de prioridades mantém as diferenças entre o custo do caminho para o vértice e para cada vizinho, e vice-versa. O algoritmo é ligeiramente mais eficiente que a versão original de Dijkstra, que visita todos os vizinhos, às custas da manutenção das filas de prioridade. No entanto, deixa-se em aberto, como estudos futuros, um tratamento eficiente de alterações simultâneas sobre um conjunto de várias arestas. A complexidade de tempo para o algoritmo proposto é $O(m \log n)$ para um grafo com n vértices e m arestas. A análise da complexidade assume a existência de uma função que associa uma orientação para as arestas do grafo, que só é conhecida para determinados tipos de grafos, entre eles os planares. Esta função é detalhada em [14], onde também são apresentados limitantes mais precisos para a complexidade de tempo, além de tratamento para ciclos com custos negativos. Apesar de resultados bastante completos, trata-se individualmente uma alteração de custo, inserção ou remoção de arestas, a cada execução do algoritmo. Para uma seqüência de alterações em componentes do grafo é necessário executar o algoritmo para cada uma delas.

O trabalho descrito em [22] apresenta uma outra alternativa para o algoritmo incremental e outra ainda para o decremental. Alteração de custo em uma aresta, tal como em [14], é tratada como uma remoção da aresta seguida por inserção da mesma com novo custo. A inicialização determina quais vértices são afetados. Uma segunda etapa propaga a alteração de custos decorrentes da remoção ou inserção da aresta, e, se necessário, determina novos caminhos para estes vértices. A complexidade de cada um destes algoritmos é $O(m_a + n_a \log n_a)$, onde n_a é o número de vértices afetados por mudanças no caminho e m_a o número de arestas ligadas a um vértice afetado.

Em [19, 20] propõe-se uma arquitetura de algoritmos para atualizar árvores de caminhos mínimos. Ela utiliza uma fila, na qual cada elemento descreve uma atualização nos caminhos de um vértice. Nesta arquitetura, diferentes algoritmos dinâmicos podem ser implementados para as respectivas

soluções estáticas de Bellman-Ford, Dijkstra ou D'Esopo-Pape, variando somente as estratégicas de retirar um elemento desta fila, de visitar vizinhos afetados e de adicionar novos elementos à fila. A operação de inserção de uma aresta é implementada como redução de custo infinito para o novo valor. A remoção aumenta o custo da aresta para infinito. Este é o único conjunto de algoritmos encontrado na literatura capaz de lidar simultaneamente com mais de uma modificação sobre os custos das arestas. Primeiro, uma execução do algoritmo atualiza a árvore de caminhos mínimos para todas as arestas que têm seu custo aumentado. Em seguida, uma segunda execução considera as arestas que diminuíram seu custo. O melhor resultado obtido para a complexidade de tempo de pior caso foi $O(n_a \log n_a + kn_a)$, onde k é o grau máximo dos vértices no grafo não orientado, n_a é o número de vértices afetados por mudanças no caminho e m_a o número de arestas ligadas a um vértice afetado. Em [18], os mesmos autores discutem uma solução através de uma interpretação física, simulada por um algoritmo semelhante ao discutido para uma das variantes em [19].

2 Conceitos Preliminares

Esta seção apresenta conceitos e notações utilizados neste trabalho. Descreve-se algumas características do grafo base e como elas influenciam a idéia do algoritmo proposto. Também são especificadas as estruturas de dados que descrevem o grafo e as perturbações sobre o grafo.

2.1 Considerações Gerais sobre o Grafo

Antes de iniciar a descrição detalhada do algoritmo, é importante apresentar algumas características do grafo base.

Assume-se a existência de um vértice diferenciado, t, que será denominado de *raiz* ou *origem*.

Custos nos vértices e nas arestas

O custo do caminho desde a raiz até um vértice é dado pela soma dos custos individuais das arestas e vértices que formam o caminho. O custo da aresta é computado ao utiliza-la para ligar dois vértices de um caminho. O custo do vértice é computado quando ele está situado entre duas arestas adjacentes de um caminho. Não será contabilizado o custo do vértice quando este ocorre no início ou no final do caminho. Se caminhos até vértices diferentes compartilham arestas ou vértices, então eles são contabilizados para cada um destes caminhos. Indiretamente, isto influencia o algoritmo a buscar caminhos com poucos vértices, ou a concentrar os caminhos em elementos de baixo custo. Em problemas reais, custos em arestas são freqüentemente associados com o custo operacional de conexões ou de transporte entre dois pontos. Os custos em vértices também podem representar o esforço de instalação de pontos de distribuição, ou o custo de estoque e roteamento. O custo em vértices não será tratado neste texto. Seria possível distribuir o custo dos vértices para as arestas incidentes e utilizar uma modificação do algoritmo de caminhos mínimos com custos nas arestas para também tratar o custo dos vértices.

Custos negativos

Se vértices ou arestas de um ciclo têm custos negativos, tal que a soma dos valores sobre o ciclo seja negativo, cada percurso sobre o ciclo melhora o custo do caminho. A solução para caminhos mínimos poderá percorrer o ciclo inúmeras vezes, obtendo um custo arbitrariamente pequeno. Para evitar esses casos, nenhuma aresta ou vértice pode apresentar custo negativo.

Ciclos e laços

Como não há custos negativos, o caminho mínimo não deverá repetir vértices nem arestas. Arestas que representam laços podem ser ignoradas pelo algoritmo, sem perda de generalidade.

Arestas múltiplas

Elas representam mais de uma possibilidade de percorrer o caminho entre dois vértices. Como o algoritmo visa minimizar os custos, quando existirem múltiplas arestas entre dois vértices, sem perda de generalidade o algoritmo considera somente uma das arestas de menor custo e ignora as demais.

Arestas direcionadas

Arestas não direcionadas devem ser tratadas como um par de arestas direcionadas, com sentidos opostos e mesmo custo. Esta abordagem exige um cuidado especial, pois atualizar o custo de uma aresta não direcionada implica em alterar as duas respectivas arestas direcionadas. Deste ponto em diante, considera-se apenas grafos direcionados.

Existência de caminhos

Para que o algoritmo funcione corretamente, é necessário que mesmo após uma remoção de vértices e arestas sempre haja um caminho orientado da raiz até cada vértice do grafo. Supõe-se ainda a existência prévia de uma árvore de caminhos mínimos. Ela descreve, para cada vértice do grafo, o caminho de menor custo partindo do vértice raiz. Dada a árvore de caminhos mínimos, aplica-se ao grafo um conjunto de perturbações, podendo conter inserção ou remoção de elementos, ou alterações de custos associados a elementos. Em seguida, o algoritmo descrito neste trabalho é executado para recalcular a árvore de caminhos mínimos, restabelecendo, para cada vértice, o caminho de menor custo do vértice raiz até este vértice.

2.2 Descrição do Grafo

Assume-se as definições comuns da teoria de grafos. Além disso, apresentaremos algumas notações específicas adotadas neste trabalho. Seja $G = \langle V, A \rangle$ um grafo direcionado, formado pelos pelos conjuntos de vértices V e de arestas A. Seja $t \in V$ um vértice diferenciado, denominado *raiz* ou *origem*. De ora em diante, t representará sempre o vértice raiz. Os caminhos mínimos desejados partem de t e terminam nos demais vértices.

Para a cada aresta $a \in A$, denota-se por $c(a)$ o custo incorrido ao percorrer a aresta a. Escrevemos que a aresta a sai do vértice $\text{start}(a)$ e entra no vértice $\text{end}(a)$. Para dois vértices v e w, a aresta a, tal que $v = \text{start}(a)$ e $w = \text{end}(a)$, pode ser apresentada de forma sucinta por a_{vw}. Para cada vértice $v \in V$, $c(v)$ denota o custo associado quando um caminho passa pelo vértice v. Sem perda de generalidade, $c(t) = 0$, onde t é o vértice raiz.

Seja $N \subseteq V$ um conjunto de vértices. Definimos $\text{out}(N) = \{\, e \in N \mid \text{start}(e) \in N \text{ e } \text{end}(e) \notin N \,\}$ como sendo o conjunto de todas as arestas saindo de um vértice em N e entrando em vértices de $V - N$. Analogamente, $\text{in}(N) = \{\, e \in N \mid \text{start}(e) \notin N \text{ e } \text{end}(e) \in N \,\}$ é o conjunto de todas as arestas entrando em um vértice de N e saindo de um vértice de $V - N$. Escreve-se também $\text{out}(v)$ e $\text{in}(v)$, respectivamente, para $\text{out}(\{v\})$ e $\text{in}(\{v\})$ (figura 1). Observe que os conjuntos $\text{in}(\cdot)$ e $\text{out}(\cdot)$ dependem somente da topologia do grafo e são independentes da árvore de caminhos mínimos. Eles permanecem inalterados após operações de alteração de custo em componentes do grafo ou após atualizações na árvore de caminhos mínimos.

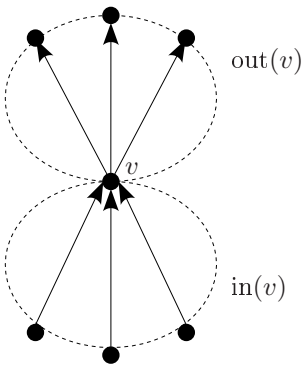

Figura 1: Conjunto de arestas entrando e saindo em v

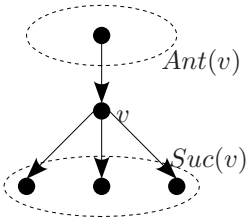

Figura 2: Vértice antecessor e vértice sucessor

2.3 Operações sobre o Grafo

Inicialmente, trataremos apenas o caso de alterações sobre o custo de arestas.

Seja G_o o grafo original. Ele sofre um conjunto $\sigma = \{\mu_1, \mu_2, ...\mu_h\}$ de h perturbações simultâneas sobre o custo das arestas, onde cada perturbação é representada por uma operação μ_i sobre o grafo. Sejam $c(a)$ e $c'(a)$, respectivamente, os custos da aresta a antes e depois das operações descritas por σ.

Seja $\delta(a) = c'(a) - c(a)$ a diferença de custo aresta a. Se $\delta(a) > 0$, então denominaremos a de *aresta aumentada*. Neste caso, usaremos a notação $\mu_+(a, \delta(a))$. Se $\delta(a) < 0$, denominaremos a de *aresta diminuída*. Neste caso, usaremos a notação $\mu_-(a, \delta(a))$. Caso $\delta(a) = 0$, então simplesmente chamaremos a de *aresta inalterada*, sem necessidade de notação adicional.

2.4 Descrição da Solução

Seja T_o a árvore de caminhos mínimos associada ao grafo original G_o, onde os caminhos são direcionados da raiz t para os demais vértices do grafo. Denotamos por G_f o novo grafo resultante após a inserção de um conjunto de modificações. Associa-se a G_f uma nova árvore de caminhos mínimos, T_f. O algoritmo recebe como entrada G_o e T_o, além da lista de modificações sobre o grafo. Seu objetivo é produzir T_f. A cada iteração o algoritmo trabalha sobre uma solução intermediária, T_a. No início, T_a é idêntica a T_o. Terminado o algoritmo, T_a representará T_f e, a cada iteração, o algoritmo pode introduzir até uma modificação em T_a, como veremos.

Para um vértice w, o *conjunto de antecessores* de w na árvore atual T_a, simbolizado por $Ant(w)$,

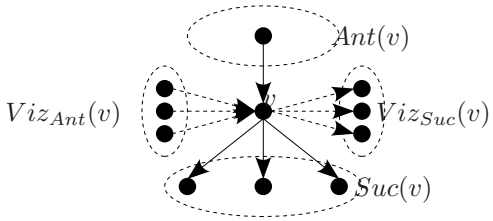

Figura 3: Vizinho antecessor e sucessor

é o conjunto contendo apenas um único vértice, aquele que antecede w no caminho da raiz t até w. Veja a figura 2. Ou seja, para $w \in V - \{t\}$, $Ant(w) = \{v\}$ sse existe $a_{vw} \in T_a$ [1]. E $Ant(t) = \varnothing$. É claro que $Ant(\cdot)$ depende da árvore atual, T_a. Para não sobrecarregar a notação, não estamos indicando a árvore T_a na notação $Ant(\cdot)$; isso deve ficar claro do contexto. Ainda, por conveniência, abusaremos da notação escrevendo $Ant(w)$ para indicar o único elemento deste conjunto. A *vizinhança de antecessores* de um vértice v, simbolizada por $Viz_{Ant}(v)$, é o conjunto dos vizinhos alcançados por uma aresta entrando em v e que não é aresta de T_a. Então, para $v \in V$, $w \in Viz_{Ant}(v)$ sse existe $a_{wv} \in A - T_a$ [2] Veja a figura 3, onde as linhas sólidas indicam arestas de T_a, e as linhas tracejadas indicam arestas fora de T_a. Note que $Ant(v) \cap Viz_{Ant}(v) = \varnothing$, para todo vértice v. Os elementos de $Viz_{Ant}(v)$ são importantes para o algoritmo, pois após uma modificação de custos, podem servir como antecessores alternativos de menor custo para v.

O *conjunto de sucessores* de um vértice v, simbolizado por $Suc(v)$, é o conjunto de todos os vértices que têm $\{v\}$ como antecessor em T_a. Então, $w \in Suc(v)$ sse $Ant(w) = \{v\}$ (figura 2). Se $Suc(v) = \varnothing$, então o vértice v é uma folha na árvore de caminhos mínimos. A *vizinhança de sucessores* do vértice v, simbolizado por $Viz_{Suc}(v)$, é o conjunto dos vizinhos alcançados por uma aresta saindo de v e que não é aresta de T_a. Assim, para $v \in V$, $w \in Viz_{Suc}(v)$ sse existe $a_{vw} \notin T_a$ (figura 3). Note que $Suc(v) \cap Viz_{Suc}(v) = \varnothing$. Durante a execução do algoritmo, uma redução do custo de v pode se propagar para a vizinhança $Viz_{Suc}(v)$.

A árvore de caminhos T_a é caracterizada totalmente pela estrutura $Ant(\cdot)$. Com o progresso do algoritmo, $Ant(\cdot)$ é ajustado em cada iteração, até refletir a solução T_f, que representa a nova árvore de caminhos mínimos. Os conjuntos $Ant(\cdot)$, $Suc(\cdot)$, $Viz_{Suc}(\cdot)$ e $Viz_{Ant}(\cdot)$ dependem exclusivamente da árvore de caminhos atual e, portanto, variam durante a execução do algoritmo.

Para um vértice v, $C(v)$ representa o custo acumulado no caminho sobre a árvore T_a, da raiz t até o vértice v, utilizando o caminho caracterizado por $Ant(\cdot)$. Este valor é considerado, durante a execução do algoritmo, como um limitante superior para o custo de um caminho mínimo. É claro que $C(\cdot)$ também depende da árvore atual, T_a.

3 Algoritmo Dinâmico Simplificado

Uma implementação menos elaborada do algoritmo dinâmico poderia calcular a nova árvore de caminhos mínimos em duas etapas. Na primeira, registraria as operações de custo das arestas e recalcularia os novos custos dos caminhos mantendo a árvore da solução original, T_o.

Como resultado, todos os vértices do grafo teriam em $C(\cdot)$ os custos iniciais corretos dos caminho indicado pela estrutura $Ant(\cdot)$ que descreve T_o, já levando em consideração as modificações de custos.

[1] A notação $a_{vw} \in T_a$ indica que a_{vw} é aresta de T_a.
[2] A notação $A - T_a$ indica o conjunto de arestas do grafo que não são arestas de T_a.

Numa segunda etapa, o algoritmo relaxaria todas as arestas, de forma semelhante ao algoritmo de Dijkstra. Obter-se-ia uma nova atribuição para $Ant(\cdot)$ representando uma nova árvore de caminhos mínimos, T_f, que refletiria as modificações indicadas pelo conjunto de operações σ. Esta estratégia potencialmente percorrerá as arestas sem necessidade, considerando que o número de arestas alteradas é pequeno em relação ao número total de arestas.

Existe uma estratégia mais eficiente capaz de realizar estas mesmas operações em uma única etapa, interagindo uma só vez sobre cada vértice do grafo e procurando evitar percorrer as arestas que não contribuirão para minimizar caminhos afetados. Esta estratégia aplica duas heurísticas. A primeira procura omitir a visita de arestas para evitar propagação desnecessária de alterações de custo. A segunda é uma forma mais eficiente de gerenciar a fila de vértices que serão selecionados pelo algoritmo para iniciar relaxações. Inicialmente, apresentaremos uma versão básica do algoritmo, onde estas heurísticas são omitidas. Um algoritmo dinâmico mais eficiente, e que usa essas heurísticas, será discutido mais adiante.

3.1 Algoritmo de Dijkstra

O algoritmo de caminhos mínimos dinâmico proposto neste trabalho está baseado em uma aplicação mais ampla de vários conceitos apresentados por Dijkstra [6]. O algoritmo de Dijkstra determina uma árvore de caminhos mínimos do vértice raiz t até os demais vértices em um grafo direcionado e com custos não negativos associados às arestas. Ele mantém para cada vértice v o custo $C(v)$ do menor caminho conhecido até o momento da raiz t até v. Inicialmente, este custo é zero para a raiz t e "infinito" para os demais vértices. O valor infinito para $C(v)$ sugere que ainda não se conhece um caminho até o vértice v. Após o término do algoritmo, o valor de $C(v)$ será o custo de um melhor caminho da raiz t até v, ou infinito se este caminho não existir.

A operação básica do algoritmo de Dijkstra é a *relaxação* de uma aresta. A cada iteração, o algoritmo escolhe um vértice v para o qual $C(v)$ já indica o custo de um caminho mínimo da raiz t até um vértice v. Se existe uma aresta a_{vw}, saindo do vértice v e entrando em w, então podemos formar um caminho de v até w tomando o caminho mínimo da raiz t até v e terminando com a aresta a_{vw}. O custo deste novo caminho até w será $C(v) + c(a_{vw})$. Se este custo for menor que o custo do melhor caminho atual de t até w, $C(w)$, então $C(w)$ pode ser substituído pelo novo valor $C(v) + c(a_{vw})$ e v deve ser atribuído a $Ant(w)$, passando v a ser o antecessor de w na nova árvore de caminhos. Se este não for o caso, ou seja, se $C(w) \leq C(v) + c(a_{vw})$, então $C(w)$ e $Ant(w)$ permanecem inalterados. Essa decisão caracteriza a relaxação da aresta a_{vw}.

A relaxação de arestas é aplicada até que todos os valores de $C(\cdot)$ apresentem o custo de um caminho mínimo para cada vértice, e $Ant(\cdot)$ reflita a estrutura de uma árvore de caminhos mínimos no grafo. Em particular, o algoritmo de Dijkstra é construído de tal forma que uma aresta a_{vw} é relaxada apenas uma vez e somente quando $C(v)$ apresenta o valor mínimo. Isto garante que uma alteração de custo em um vértice será propagada somente uma vez para os vizinhos de v, de forma que o número de iterações do algoritmo é proporcional ao número total de arestas no grafo.

O algoritmo de Dijkstra também pode ser entendido classificando-se os vértices em três conjuntos: S_C, S_V e S_N. O conjunto S_C contém o vértices *consolidados*. Para $v \in S_C$, o custo $C(v)$ é mínimo para os todos os caminhos da raiz t até o vértice v e que passem somente por vértices do próprio conjunto S_C. O conjunto S_V contém os vértices v para os quais é conhecido um custo $C(v)$ de um caminho, ainda não necessariamente mínimo, de t até v. Os demais vértices, para os quais $C(v)$ é "infinito", encontram-se no conjunto S_N.

Inicialmente, S_C contém somente a raiz t com custo zero, e S_N contém os demais vértices. O próximo passo é relaxar todas as arestas a_{tv} saindo de t para vértices v em S_N, movendo esses vértices de S_N para S_V. Em seguida, a cada iteração do algoritmo, um vértice v é movido de S_V para S_C. Esse vértice v é escolhido entre aqueles em S_V que apresentam o menor valor para $C(v)$. Neste momento, o algoritmo de Dijkstra relaxa todas as arestas a_{vw} saindo de v. Se a relaxação

da aresta a_{vw} atualiza o custo de w, então ou w está em S_V e pode ter o valor de $C(w)$ reduzido conforme explicado anteriormente, ou w encontra-se em S_N e é movido para S_V sendo $C(v)+c(a_{vw})$ o novo valor de $C(w)$. Se a relaxação de a_{vw} não atualiza o custo de w, então w está em S_C, ou w está em S_V com $C(w) \leq C(v) + c(a_{vw})$.

O algoritmo de Dijkstra explora a ordenação dos vértices segundo os custos em $C(v)$. A ordenação permite prever, em cada iteração, o próximo vértice de S_V que já apresenta em $C(v)$ o custo de um caminho mínimo. Este é o vértice selecionado para ser movido de S_V para S_C.

É interessante notar que, para um vértice w, na primeira vez que uma aresta entrando em w é relaxada, seja ela a_{vw}, ocorre a primeira atribuição para $C(w)$. Esta relaxação atribui $C(v)+c(a_{vw})$ para $C(w)$ e move w de S_N para S_V. A relaxação de uma outra aresta a_{zw} reduzirá ainda mais o valor de $C(w)$ somente se $C(z) + c(a_{zw}) < C(w)$, mantendo w em S_V.

3.2 Estados dos Vértices

A versão básica do novo algoritmo já é uma extensão do algoritmo de Dijkstra, e faz uso da noção de *estados*, um rótulo atribuído aos vértices. Nesta versão simplificada do algoritmo dinâmico, os estados poderão assumir um dentre quatro valores: `nV`, `Cr`, `C` e `Ce`.

Em particular, os estados vão separar os vértices de S_N em duas categorias distintas. Um exemplo simples ilustrará a necessidade do uso de estados atribuídos aos vértices. A figura 4 representa um grafo inicial G_o, junto com a árvore original, T_o.

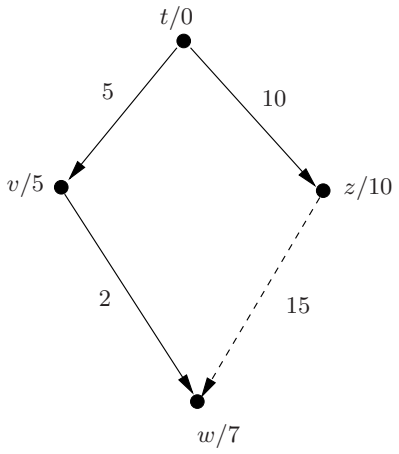

Figura 4: Um grafo e árvores iniciais

Arestas sólidas estão em T_o, e os custos das arestas e caminhos minimos são indicados por valores próximos às arestas e aos vértices correspondentes. Suponha que o custo da aresta a_{tz} foi reduzido para 1 e o custo da aresta a_{vw} subiu para 20. A figura 5 ilustra a situação.

É óbvio que agora os custos dos caminhos para z e w estão incorretos, não refletindo custos mínimos até esses vértices. Tampouco a árvore inicial indica caminhos mínimos. Se o algoritmo de Dijkstra fosse usado sem alterações, e levando-se em conta os novos valores de custos nas arestas à medida que o algoritmo progredisse, teríamos a situação da figura 6, após relaxar as arestas a_{tv} e a_{tz}.

A próxima iteração do algoritmo de Dijkstra moveria o vértice z para S_C, uma vez que $C(z) < C(v)$. É aqui que reside o problema. Ao mover z para S_C, a aresta a_{zw} seria relaxada. Porém, $C(z) + c(a_{zw}) + \delta(a_{zw}) = 1 + 15 + (15 - 15) = 16$. Como $C(w) = 7$ nesse instante, e $16 > 7$, o custo

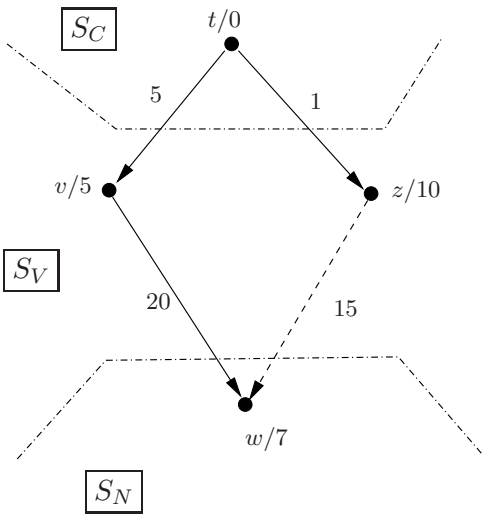

Figura 5: Após consolidar a raiz

do vértice w *não seria* alterado e o vértice z *não seria* considerado o novo antecessor de w na árvore de caminhos mínimos que está sendo recalculada. No entanto, essa seria a atitude correta, uma vez que o custo da aresta a_{vw} foi bastante aumentado, tornando o caminho para w através de v bem mais custoso. Aliás, no instante em que a aresta a_{zw} é relaxada, o custo armazenado em $C(w)$ não é sequer confiável.

A alteração proposta detectaria, durante o relaxamento da aresta a_{zw}, que w ainda não foi visitado, e designaria z como um "antecessor candidato" para w, associando a esse antecessor candidato o custo do novo caminho até w passando por z, ou seja 16. Mais ainda, w seria rotulado com um estado especial para indicar que já dispõe de um antecessor candidato. Quando, por sua vez, a aresta a_{vw} for relaxada, o algoritmo compararia o custo do antecessor candidato, 16, com o novo custo do caminho passando por v, ou seja $C(v) + c(a_{vw}) + \delta(a_{vw}) = 5 + 2 + (20 - 2) = 25$. Nesse ponto, o algoritmo perceberia que o antecessor candidato ofereceria um caminho melhor até w e alteraria a estrutura da árvore atual, T_a, de forma a colocar z como o antecessor de w em $Ant(w)$.

Vértices consolidados

Define-se o conjunto S_C, dos vértices cujo estado é `C`. Afirma-se que, para $v \in S_C$, o custo $C(v)$ é mínimo em relação a todos caminhos da raiz t até v, e que sejam formados apenas por vértices do conjunto S_C.

Vértices certificados

Define-se o conjunto S_V, dos vértices cujo estado é `Ce`. O custo associado a um vértice v em S_V já será confiável, ou seja, corresponderá ao custo do caminho atual da raiz t até v, já considerando as alterações de custos aplicadas a G_o.

Vértices não certificados

Define-se o conjunto S_N, dos vértices cujo estado é `nV` ou `Cr`. Vértices nesse conjunto, se rotulados como `nV`, ainda não foram visitados. Se rotulados como `Cr` já foram visitados, mas o custo do

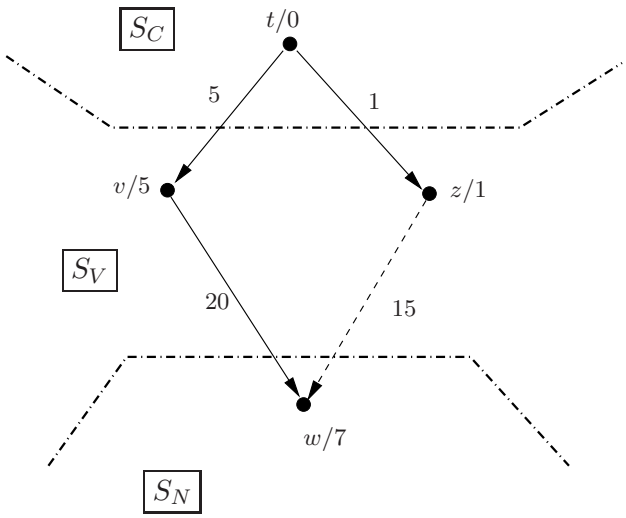

Figura 6: Após consolidar a raiz, e os vértices v e z

caminho não é confiável.

3.3 Descrição do Algoritmo

O algoritmo dinâmico é construído sobre a premissa de que, após as modificações nos custos das arestas, a nova solução T_f manterá uma forte semelhança com a solução original T_o e, portanto, permitirá o reaproveitamento significativo de subgrafos de T_o para obter T_f.

Inicialização

A inicialização dos conjuntos S_C, S_V e S_N é semelhante àquela prescrita pelo algoritmo de Dijkstra. O conjunto S_C contém somente a raiz t, com estado `C`. O conjunto S_V começa vazio. Os demais vértices encontram-se no conjunto S_N, todos com estado `nV`. À estrutura $Ant(\cdot)$ recebe inicialmente a mesma árvore de caminhos da solução original T_o. Diferentemente do algoritmo de Dijkstra, que atribui "infinito" para $C(\cdot)$, o algoritmo dinâmico inicia o custo dos caminhos de acordo os valores presentes na árvore original T_o.

O algoritmo dinâmico tenta amortizar o custo de manipulação da fila ordenada de vértices ao reaproveitar os custos da solução original. Pelo fato de assumir a árvore original na inicialização o algoritmo assume uma situação otimista na qual espera relaxar somente as arestas T_o.

Relaxação

A operação básica do algoritmo dinâmico continua sendo a relaxação de arestas. A cada iteração do algoritmo, o vértice v em S_V que apresenta o menor valor para $C(v)$ tem o seu estado trocado de `Ce` para `C`. Esta operação efetivamente move v de S_V para S_C e é chamada de *consolidação* do vértice v. Após consolidar um vértice v, o algoritmo dinâmico relaxa as arestas a_{vw} saindo de v.

Suponha que existam as arestas a_{vw} e a_{zw}, onde a primeira é a aresta incidente em w na solução original T_o, e a segunda é uma outra aresta incidente em w. Suponha que a aresta a_{zw} é relaxada antes que a aresta a_{vw}. No algoritmo de Dijkstra, a relaxação de a_{zw} moveria o vértice w para S_V, e ajustaria os valores de $Ant(w)$ e de $C(w)$ de modo a refletir a melhor opção para w nesse momento.

No algoritmo dinâmico isso não ocorre. Nesse instante, se tivermos $C(w) > C(z) + c(a_{zw}) + \delta(a_{zw})$, então o vértice z passa a ser o ***antecessor candidato*** do vértice w, o custo $C(z) + c(a_{zw}) + \delta(a_{zw})$ passa a ser o ***custo candidato*** do vértice w, e o estado do vértice w passa de `nV` para `Cr`. Note que w ainda permanece no conjunto S_N. Os valores para o antecessor candidato e seu custo são armazenados em estruturas auxiliares: $C_C(\cdot)$ e em $Ant_C(\cdot)$, respectivamente. Essas estruturas são distintas de $C(\cdot)$ e de $Ant(\cdot)$, de forma que atribuições a $C_C(\cdot)$, ou a $Ant_C(\cdot)$, não alteram a árvore atual, T_a. É importante notar que, na árvore atual T_a, o caminho da raiz t até o vértice w ainda termina na aresta a_{vw}. Se esta aresta ainda não foi relaxada, então o custo armazenado em $C(w)$ pode não refletir o custo real do caminho de t até w na árvore atual T_a, em particular por conta de alguma alteração no custo da própria aresta a_{vw}. Nessa situação, dizemos que o custo $C(w)$ é *não confiável*. Note, porém, que todo custo armazenado em $C_C(w)$ sempre reflete corretamente o custo de um caminho da raiz t até w. Entretanto, enquanto a aresta a_{vw} não for relaxada esse caminho não coincide com o caminho de t até w na árvore T_a. Em particular, a última aresta do caminho alternativo, a_{zw}, não está em T_a.

À medida em que outras arestas incidentes em w, distintas de a_{vw}, forem sendo relaxadas, o antecessor candidato e seu custo podem ser atualizados. Eventualmente, a aresta a_{vw} é relaxada. Nesse instante, comparam-se os valores de $C_C(w)$, se houver, com $C(v) + c(a_{vw}) + \delta(a_{vw})$. O menor valor é selecionado e só então são atualizados os valores de $C(w)$ e de $Ant(w)$, se for o caso. Isso feito, o vértice w é rotulado com o estado `Ce` e é movido de S_N para S_V, sendo essa operação também chamada de *certificação* do vértice w. As potenciais atualizações de $C(w)$ e de $Ant(w)$ podem alterar a árvore atual. De qualquer forma, alterando ou não esses valores, após a operação de certificação do vértice w o custo em $C(w)$ garantidamente passa a refletir o custo do caminho da raiz t até o vértice w na árvore atual, T_a, já levando em conta todas as alterações de custos nas arestas desse caminho. Nessa situação, dizemos que o custo $C(w)$ é *confiável*. Se existirem evidências para um vértice em S_N e com estado `Cr`, de que o caminho através do antecessor candidato apresentará custo menor que o caminho através do antecessor atual, então a certificação de v pode ser ignorada, simplesmente substituindo $Ant(v)$ e $C(v)$ por $Ant_C(v)$ e $C_C(v)$.

Uma vez movido para o conjunto S_V, o vértice w terá um comportamento semelhante ao que teria no algoritmo de Dijkstra, eventualmente entrando para o conjunto S_C. Nesse momento, consolida-se o vértice w, e o valor do seu custo, armazenado em $C(w)$, passa a refletir o custo de um caminho mínimo da raiz t até este vértice, no grafo G_f.

Propriedades

Observa-se as seguintes propriedades, enunciadas aqui para reforçar a intuição do leitor acerca do funcionamento do algoritmo:

- Para um vértice v em S_C, o valor de $C(v)$ é mínimo e confiável. Uma vez no conjunto S_C, o custo $C(v)$ e o caminho da raiz t até o vértice v não serão mais modificados durante todo o resto da execução do algoritmo.
- Para um vértice v em S_V, o valor $C(v)$ é confiável e serve como *limitante superior* para o custo do novo caminho mínimo da raiz t até v na árvore final T_f.
- O caminho em T_a, da raiz t até um vértice v em S_V, será sempre formado por uma seqüência de vértices em S_C e terminando com o vértice v.
- Sempre teremos $S_C \cup S_V \cup S_N = V$, com S_C, S_V e S_N dois a dois disjuntos.

Figura 7: Algoritmo Dinâmico de Caminhos Mínimos

Procedure `AlgoritmoDinâmico`

Data: $Ant(\cdot)$ e $C(\cdot)$ da solução T_o anterior
Result: $Ant(\cdot)$ e $C(\cdot)$ atualizados para a nova solução T_f

```
1  Inicializacao(T_o)
   /* Consolida a raiz                                                    */
2  AtualizaSucessores(t)
3  RelaxaVizSucessora(t)
4  E(t) ← C
   /* Iteração de consolidação e certificação de vérices:                */
5  Q ← V − {t}, (ordem crescente por min{C(·), C_C(·)})
6  enquanto Q ≠ ∅ faça
7      v← Extrai(Q) /* Retira de Q o vértice com menor valor para C(·)   */
8      se E(v) ∉ {nV, C} então
9          se E(v)= Cr então
10             Ant(v) ← Ant_C(v)
11             C(v) ←C_C(v)
12         AtualizaSucessores(v)
13         RelaxaVizSucessora(v)
14         E(v) ← C
```

Fila de Ordenada de Vértices

De forma semelhante ao algoritmo de Dijkstra, a simplificação do algoritmo dinâmico utiliza uma fila para determinar o próximo vértice, ordenada de acordo com as estimativas de custo dos caminhos. A cada iteração, o vértice de menor estimativa de custo é extraído da fila.

Como cada vértice está associado a dois valores, o custo (originário do custo do caminho original ou recalculado ao certificar o vértice) e o custo candidato (atribuído por arestas relaxadas antes de certificar o vértice), a fila de vértices deve considerar ambos para determinar a ordenação.

Se existe um antecessor candidato que resulta em um caminho de menor estimativa de custo, então o vértice v deve ser ordenado com o custo candidato $C_C(v)$, caso contrário, com a estimativa $C(v)$. A fila de vértices deve ser ordenada de acordo com $\min\{C(\cdot), C_C(\cdot)\}$.

Enquanto o custo do caminho de um vértice apresentar um valor não confiável (seu estado é `nV`), ele poderá apresentar um valor de $C(\cdot)$ abaixo do valor de seu verdadeiro caminho mínimo. Talvez, ele seja extraído precipitadamente da fila com estado `nV`. Neste caso, o algoritmo ignora o vértice, pois ele será re-inserido novamente na fila quando for relaxado pela primeira vez com um custo de valor confiável ou com um custo candidato (ou seja, estado `Cr`).

Possívelmente, o vértice é extraído da fila utilizando o valor do custo candidato e com estado `Cr`. Sabemos que a certificação do vértice será com um valor mais alto que o custo candidato. Neste caso, podemos consolidar adiantadamente o vértice, a necessidade de certifica-lo primeiro.

3.4 Pseudo Código para o Algoritmo Básico

Veja a figura 7. O algoritmo inicia chamando o procedimento `Inicializacao` (linha 1) para atribuir valores às estruturas de dados do algoritmo, baseando-se na solução inicial T_o. Em seguida, os

Figura 8: Algoritmo Dinâmico de Caminhos Mínimos

Procedure `Inicializacao`(T_o)

1 $Ant(\cdot)\leftarrow$ antecessores de T_o
2 $C(\cdot)\leftarrow$ custos dos caminhos de T_o
3 **para cada** $v \in V$ **faça** $Ant_C(v) \leftarrow \varnothing$; $C_C(v) \leftarrow C(v)$
4 **para cada** $v \in V - \{t\}$ **faça** $E(v) \leftarrow$ `nV`
5 $E(t) \leftarrow$ `Ce`

vértices sucessores de t, na árvore T_a, são visitados (linha 2). O próximo passo visita os vértices na vizinhança de sucessores de t (linha 3) e, no passo seguinte, a raiz é consolidada (linha 4). Desta forma, o algoritmo dinâmico consolida a raiz t em primeiro lugar. Se a raiz t fosse consolidada nas demais iterações, o algoritmo poderia extrair de S_V, erroneamente, um outro vértice z com $C(z) = 0$ antes de extrair a raiz t.

O algoritmo dinâmico prossegue iterativamente, consolidando um vértice de cada vez (linhas 6 – 14). O algoritmo opera com auxílio da estrutura `Q`, que ordena os vértices de forma crescente segundo valores de $C(\cdot)$ e $C(\cdot)$. A cada passo, o algoritmo retira de `Q` o vértice de menor valor da estimativa de custo (linha 7). Vértices marcados com estado `nV` ou `C` são simplesmente ignorados. O estado `C` sinaliza que o vértice já está consolidado, apresentando valor correto para um caminho de custo mínimo e, portanto, não faz sentido consolida-lo novamente. Um vértice com estado `nV` apresenta custo de valor ainda não confiável, que reflete o custo do caminho mínimo antes das perturbações sobre custos de arestas terem sido aplicadas. Ele será ignorado neste momento e adicionado novamente à estrutura `Q` pelo procedimento `RelaxaVizSucessora`.

Vértices marcados com estado `Cr` (linha 9) já apresentam um custo confiável, pois têm um antecessor candidato. Nesses casos, o vértice é certificado e a estrutura da árvore é modificada (linhas 9 – 11).

A relaxação das arestas saindo de v é delegada para os dois procedimentos chamados nas linhas 12 e 13. O primeiro, `AtualizaSucessores`(v), certifica vértices sucessores de v em T_a, relaxando as arestas para sucessores de v. O procedimento `RelaxaVizSucessora`(v) relaxa as demais arestas, incidentes em elementos da vizinhança de sucessores de v.

Na linha 14, o vértice v é consolidado. Neste momento, garante-se que o custo do caminho, em T_a, da raiz até v é mínimo, e não sofrerá alterações nas próximas iterações do algoritmo.

Inicialização

O pseudocódigo do procedimento `Inicializacao` é mostrado na figura 8. Ele recebe como parâmetro a solução T_o e inicializa as variáveis utilizadas pelo algoritmo dinâmico.

A solução T_a, caracterizada por $Ant(\cdot)$ e $C(\cdot)$ recebe uma cópia de T_o (linhas 1 – 2). Em seguida, o armazenamento para antecessor e custo candidato é criado (linha 3). Finalmente, marca-se todos os vértices, exceto a raiz t, com o estado `nV` (linha 5), indicando que eles não foram certificados, nem possuem antecessor candidato, e portanto pertencem ao conjunto S_N.

Certificação

O pseudocódigo do procedimento `AtualizaSucessores` é mostrado na figura 9. Ele recebe como parâmetro o vértice v. Seu objetivo é certificar o custo de cada um dos vértices que na árvore atual T_a são sucessores de v.

Figura 9: Atualização de vértice sucessor

Procedure `AtualizaSucessores(`v`)`

Data: Vértice v
Result: Percorre os sucessores de v na árvore T_a, certificando esses vértices

```
1  para cada w ∈ Suc(v) faça
2      se E(w) = C então reinicia no passo 1
       /* Custo confiável através da aresta a_vw                    */
3      c ← C(v) + c(a_vw) + δ(a_vw)
4      se E(w) = Cr e C_C(w) < c então
           /* Já existe antecessor candidato de menor custo para o sucessor w.  */
5          Ant(w) ← Ant_C(w)
6          C(w) ←C_C(w)
7      senão
8          C(w) ←c
9      E(w) ← Ce
10     Corrige a posição de w em Q, caso w esteja em Q, ordenado por min{C(·), C_C(·)}
```

O algoritmo ignora os vértices w já consolidados (linha 2), pois seu custo mínimo já é conhecido através de outro caminho. Em seguida, o procedimento calcula o custo confiável do vértice sucessor w (variável `c`, linha 3), considerando o caminho através de v. Se já existe um antecessor candidato para w através do qual é possível obter um caminho de custo menor que através de v, o procedimento redefine o antecessor de w como sendo este antecessor candidato (linhas $4-6$). Caso contrário, apenas atualiza o valor do caminho (linha 8). Finalmente, w é certificado, tendo seu estado marcado como `Ce`.

Se w pertence à fila `Q`, então sua posição precisa ser corrigida, para manter a ordenação de `Q` em relação à $\min\{C(\cdot), C_C(\cdot)\}$. Caso contrário, w é inserido na devida posição (linha 10).

Relaxação

O procedimento `RelaxaVizSucessora`, cujo pseudocódigo é mostrado na figura 10, recebe como parâmetro um vértice v. Os elementos da vizinhança sucessora $Viz_{Suc}(v)$ são visitados para determinar se v serve como vértice antecessor que resulta em um caminho de custo menor. Primeiro, o algoritmo determina o conjunto N, que contém a vizinhança sucessora de v (linha 1).

Para cada vértice w em N, o algoritmo calcula o custo para o caminho alternativo através do vértice v (linha 3). Se o vértice w apresenta estado `nV`, então seu custo, potencialmente, ainda não reflete o custo real do caminho da raiz até w, após as perturbações de custos sobre as arestas do grafo. Portanto, ainda não é possível avaliar se o caminho através de v apresenta uma estimativa vantajosa. O vértice v é armazenado como antecessor candidato, o custo `c` como custo candidato de w e o estado de w passa a ser `Cr` (linhas $4-7$).

Se w já foi visitado (possui um antecessor candidato e seu estado é `Cr`), então v substitui o atual antecessor candidato somente se `c` apresenta valor menor que o custo candidato atual (linhas $8-11$).

Vértices consolidados são ignorados (linha 13).

Resta a possibilidade de w já ser certificado. Nesses casos, isto é, quando seu estado é `Ce`, o procedimento substitui o antecessor de w por v (linhas $16-17$) somente se isso resulta em um

Figura 10: Relaxação de vizinho sucessor

Procedure `RelaxaVizSucessora`(v)

Data: Vértice v
Result: Visita a vizinhança sucessora de v, i.e., sucessores que não estão na árvore T_a.

/* Determinar os vizinhos que devem ser visitados */
$N \leftarrow Viz_{Suc}(v)$
/* Verifica se existe a possibilidade do vizinho reduzir o custo através de v */
para cada $w \in N$ **faça**
  $\mathtt{c} \leftarrow C(v) + c(a_{vw}) + \delta(a_{vw})$
  se $E(w) =$ `nV` **então**
    /* O vértice v torna-se antecessor candidato. */
    $E(w) \leftarrow$ `Cr`
    $C_C(w) \leftarrow \mathtt{c}$
    $Ant_C(w) \leftarrow v$
  senão, se $E(w) =$ `Cr` **então**
    se $\mathtt{c} < C_C(w)$ **então**
      /* O vértice v substitui o antecessor candidato. */
      $C_C(w) \leftarrow \mathtt{c}$
      $Ant_C(w) \leftarrow v$
  senão, se $E(w) =$ `C` **então**
    reinicia com o próximo vértice de N
  senão
    /* Estado de w é `Ce` */
    se $\mathtt{c} < C(w)$ **então**
      /* Relaxa normalmente a aresta a_{vw}. */
      $C(w) \leftarrow \mathtt{c}$
      $Ant(w) \leftarrow v$
  Corrige a posição de w em `Q`, caso w esteja em `Q`, ordenado por $\min\{C(\cdot), C_C(\cdot)\}$

caminho de custo menor da raiz t até o vértice v (linha 15).

Se w pertence à fila `Q`, então sua posição precisa ser corrigida, para manter a ordenação de `Q` em relação à $\min\{C(\cdot), C_C(\cdot)\}$. Caso contrário, w é inserido na devida posição (linha 18).

3.5 Formalização dos Estados de Vértice

O algoritmo utiliza o estado do vértice para determinar a estratégia da relaxação de arestas. Nesta seção, discute-se em detalhes os estados possíveis para um vértice v. A *condição* descreve quando o vértice deve ser atribuído ao estado e a *conseqüência* descreve como o vértice será tratado uma vez que ele assume o estado.

Os estados e suas propriedades estão descritos a seguir:

- **Não Visitado (`nV`):** O vértice v está no conjunto S_N

  Condição: O vértice v nunca foi visitado através de uma relaxação (implementada pelas funções `RelaxaVizSucessora` e `AtualizaSucessores`). Para o vértice v, conhece-se somente o caminho e o respectivo custo para a solução original T_o antes, desconsiderando as operações σ que afetam o custo das arestas.

  Conseqüência: O valor de $C(v)$ não é confiável e não poderá ser utilizado pelo algoritmo para relaxar uma aresta a_{vw} para um vizinho w.

- **Candidato à Relaxação (`Cr`):** O vértice v está no conjunto S_N

  Condição: O vértice v está no conjunto S_N, mas já foi visitado por uma relaxação (mais precisamente, pela função `RelaxaVizSucessora`). Além do caminho da solução original T_o, conhece-se também um caminho alternativo até v, através do antecessor candidato $Ant_C(v)$.

  Conseqüência: O valor de $C(v)$ não é confiável e não poderá ser utilizado pelo algoritmo para relaxar uma aresta a_{vw} para um vizinho w. O antecessor candidato, que oferece o caminho alternativo, está armazenado em $Ant_C(\cdot)$. O custo do caminho alternativo é dado por $C_C(v)$. O algoritmo ainda não tem informação suficiente para decidir qual destas duas opções representa o caminho de menor estimativa de custo. A decisão sobre adotar ou não o vizinho como antecessor fica adiada até o momento quando o algoritmo obtiver a estimativa de custo atualizado do vértice através de $Ant(v)$.

- **Certificado (`Ce`):** É o estado dos vértices compondo o conjunto S_V.

  Condição: O vértice v está no conjunto S_V e foi visitado pela função `AtualizaSucessores`. No entanto, ainda não é possível afirmar se o custo do caminho até v é mínimo.

  Conseqüência: O valor de $C(v)$ é confiável e poderá ser utilizado pelo algoritmo para relaxar uma aresta a_{vw} para um vizinho w. Veremos que os vértices antecessores situados no caminho da raiz t até v também apresentam estado `C`.

- **Consolidado (`C`):** É o estado dos vértices compondo o conjunto S_C.

  Condição: O vértice já foi certificado e é possível comprovar que o valor de $C(v)$ é igual ao valor de um caminho mínimo.

  Conseqüência: Como o custo do caminho é mínimo em v, nenhum outro vizinho será capaz de atribuir uma estimativa de custo menor. O Antecessor $Ant(v)$ e o estado $E(v)$ não sofrem mais alterações. De fato, o vértice com estado `C` nunca mais será analisado pelo algoritmo. É o estado dos vértices, após a execução do algoritmo. Veremos que o estado do antecessor em T_a também é consolidado.

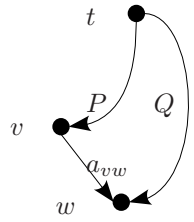

Figura 11: Caminhos P e Q de t para v e w

3.6 Condições para Relaxação de Arestas

Sejam v e w dois vértices adjacentes no grafo através da aresta a_{vw}. Em outras palavras, ou v é um vértice antecessor de w, ou é vizinho antecessor. Conforme a figura 11, sejam, respectivamente, P e Q os caminhos de v e w até a raiz t, com custo $C(v)$ e $C(w)$.

Deseja-se estabelecer condições que ajudem a prever se o vértice w reduziria sua estimativa de custo atual $C(w)$ caso assumisse v como antecessor.

Será discutido na seção 5 que a complexidade do tempo de execução depende do número de arestas e vértices visitados. Com auxílio de *condições necessárias* e *condições suficientes* é possível melhorar eficiência do algoritmo com auxílio de uma heurística que procura evitar operações de relaxar arestas.

Uma *condição necessária*, quando satisfeita para uma aresta a_{vw}, revela a possibilidade do caminho da raiz t até o vértice v, estendido pela aresta a_{vw}, reduzir o custo do caminho atual do vértice w. É um indício que o algoritmo deve relaxar a aresta, mas não uma garantia que isto reduzirá o custo do caminho até w. Quando nenhuma condição necessária é satisfeita para uma aresta a_{vw}, este fato sugere (mas não garante) que não existe um caminho mínimo contendo a aresta a_{vw}. O algoritmo dinâmico da seção 4 utiliza este fato para economizar passos do processamento. É importante ressaltar que se trata de uma heurística construída sobre decisões otimistas, que podem estar equivocadas. Este algoritmo dinâmico será capaz de reconhecer casos quando erra para então corrigir a decisão.

Uma *condição suficiente* é um fato ainda mais restrito. Se satisfeita para uma aresta a_{vw}, garante sem verificar a condição da proposição 3.1, que o caminho da raiz t até v, estendido pela aresta a_{vw}, reduz o custo do caminho atual de w.

Proposição 3.1 (Condição trivial, necessária e suficiente) *Se* $C(v)+c(a_{vw})+\delta(a_{vw}) < C(w)$, *então w reduz o valor da estimativa de custo $C(w)$ ao adotar v como antecessor.*

É importante que, ao realizar a comparação, os vértices v e w estejam certificados, ou seja, apresentem as estimativas de custo de caminho $C(v)$ e $C(w)$ com valores confiáveis de acordo com as perturbações sofridas pelo grafo.

A condição trivial da proposição 3.1 permite estabelecer as seguintes condições suficientes:

Proposição 3.2 (Condição suficiente) *Seja um vértice w para o qual vale: $E(w) = \mathtt{Cr}$, w é recém extraído da fila* Q *e $Ant_C(w) = w_{ac}$. Então os vértices w e w_{ac} satisfazem a condição trivial da proposição 3.1.*

Seja $w_a = Ant(w)$ e a_{wac} a aresta entre w e w_{ac}. Então $C(w_a) > C(w_{ac}) + c(a_{wac}) + \delta(a_{wac})$. Se não fosse, então w_a seria extraído da fila Q antes de w_{ac}.

Extrair o vértice com estado `Cr`, garante-se a relaxação da aresta w_a reduz a estimativa de custo do caminho até w. Por acaso, este caminho também é mínimo, motivo pelo qual o algoritmo também consolida o vértice.

Proposição 3.3 (Condição suficiente) *Seja w um vértice para o qual vale: $E(w) =$ `Ce`, $Ant_C(w) = w_{ac}$ e $C(w_{ac}) + c(a_{wac}) + \delta(a_{wac}) < C(w)$. Então w e $Ant_C(w)$ satisfazem a condição trivial da proposição 3.1.*

Esta é a decisão de manter o antecessor ao certificar um vértice que já possui um antecessor candidato.

4 Algoritmo dinâmico

Esta seção apresenta como obter o algoritmo dinâmico, baseando-se na descrição do algoritmo simplificado da seção anterior.

Os conceitos como *consolidar* vértices, *certificar* sucessores, *antecessor candidato* e *custo candidato* são mantidos do algoritmo simplificado. O conceito de "estado" de um vértice precisa ser redefinido, pois ele servirá também como memória de decisão para heurística que relaxa as arestas.

O algoritmo dinâmico parte do pressuposto que uma quantidade relativamente pequena de vértices será afetada, e dessa forma busca relaxar primeiro as arestas da própria solução original, relaxando as demais somente se necessário. Em especial, o algoritmo dinâmico tentará evitar relaxar arestas que resultarão na atribuição de antecessores candidatos. São definidas novas *condições necessárias* e *condições suficientes* para suportar as decisões do algoritmo dinâmico.

Os custos da solução original T_o servirão para acelerar a manipulação da fila de vértices. Espera-se que a atribuição dos novos antecessores afete poucos vértices e é razoável que um conjunto pequeno de vértices altere os custos dos caminhos. Desta forma, é importante reaproveitar a ordenação de vértices obtida pelo cálculo da árvore de caminhos original T_o.

4.1 Estados dos Vértices

Os novos estados para o algoritmo dinâmico estão descritos a seguir:

- **Não Visitado (`nV`):** Igual ao algoritmo simplificado.
- **Candidato à Relaxação (`Cr`):** Igual ao algoritmo simplificado.
- **Consolidado (`C`):** Igual ao algoritmo simplificado.
- **Certificado (`Ce`):** O algoritmo dinâmico não possui mais o estado `Ce`. Ele é substituído por estados específicos (`O`, `A`, `D`, `AD` e `R`), descritos a seguir.
- **Original (`O`):** O vértice v está no conjunto S_V, manteve seu caminho original de T_o e a estimativa de custo $C(v)$ não se alterou em relação a T_o.

  Condições:

  - O caminho da raiz t até v está inalterado, permanece igual ao da solução original T_o.
  - Não existe aresta com custo aumentado no caminho.
  - Não existe aresta com custo diminuído no caminho.
  - A estimativa de custo $C(v)$ está inalterada, permanece igual a da solução original T_o.
  - Não existe vizinho antecessor consolidado conhecido capaz de melhorar a estimativa de custo do vértice v.

Por indução, é fácil concluir se um vértice v apresenta estado `O`, então todos os vértices antecessores situados no caminho da raiz t até v também apresentam estado `O`.

Conseqüência: O custo do caminho pode ser reduzido somente através de um caminho pelo vizinho antecessor que tenha diminuído sua estimativa de custo em relação a solução original T_o.

- **Aumentado (`A`)**: O vértice v está no conjunto S_V, manteve seu caminho original de T_o, mas a estimativa de custo $C(v)$ aumentou em algum trecho do caminho.

  Condições:

  - O caminho da raiz t até v está inalterado, permanece igual ao da solução original T_o.
  - Existe pelo menos uma aresta com custo aumentado no caminho.
  - Não existe aresta com custo diminuído no caminho.
  - A estimativa de custo $C(v)$ é maior que a da solução original T_o.
  - Não existe vizinho antecessor consolidado conhecido capaz de melhorar a estimativa de custo do vértice v.

  Conseqüência: O custo do caminho pode ser (potencialmente) reduzido por qualquer vizinho antecessor.

- **Diminuído (`D`)**: O vértice v está no conjunto S_V, manteve seu caminho original de T_o, mas a estimativa de custo $C(v)$ diminuiu em algum trecho do caminho.

  Condições:

  - O caminho da raiz t até v está inalterado, permanece igual ao da solução original T_o.
  - Não existe aresta com custo aumentado no caminho.
  - Existe pelo menos uma aresta com custo diminuído no caminho.
  - A estimativa de custo $C(v)$ é menor que a da solução original T_o.
  - Não existe vizinho antecessor consolidado conhecido capaz de melhorar a estimativa de custo do vértice v.

  Conseqüência: O vértice pode (potencialmente) reduzir o custo de caminho de vizinhos sucessores.

- **Aumentado e Diminuído (`AD`)**: É uma combinação dos dois estados anteriores. O caminho permaneceu igual ao original, mas a estimativa de custo $C(v)$ é diferente (não se sabe se maior ou se menor que da solução original T_o).

  Condições:

  - O caminho da raiz t até v está inalterado, permanece igual ao da solução original T_o.
  - Existe pelo menos uma aresta com custo aumentado no caminho.
  - Existe pelo menos uma aresta com custo diminuído no caminho.
  - A estimativa de custo $C(v)$ é diferente que a da solução original T_o.
  - Não existe vizinho antecessor consolidado conhecido capaz de melhorar a estimativa de custo do vértice v.

  Conseqüência: Como não é possível afirmar se a estimativa de custo efetivamente aumentou ou diminuiu, é necessário combinar as propriedades dos estados `A` e `D`. O custo do caminho pode (potencialmente) ser reduzido por qualquer vizinho antecessor. O vértice pode (potencialmente) reduzir o custo de caminhos de vizinhos sucessores.

- **Relaxado (`R`)**: O vértice v está no conjunto S_V, o caminho até a raiz foi modificado, e dessa forma conseguiu reduzir o valor de sua estimativa de custo. O desvio do caminho original ocorre sempre através de uma aresta relaxada.

  Condições:

- Algum vértice no caminho da raiz t até v trocou de antecessor, de forma a reduzir sua estimativa de custo.
- Não existem restrições quanto: à existência de arestas aumentadas ou reduzidas neste caminho, ao novo valor da estimativa de custo $C(v)$ e à existência ou não de vizinhos antecessores consolidados conhecidos capazes de melhorar o custo do vértice v.

Conseqüência: O vértice (potencialmente) reduziu sua estimativa de custo, mas não é possível afirmar se ela aumentou ou diminuiu. O custo do caminho pode ser (potencialmente) reduzido por qualquer vizinho antecessor. O vértice pode (potencialmente) reduzir o custo de caminho de vizinhos sucessores.

4.2 Condições para Relaxação de Arestas

Serão apresentadas novas proposições interessantes para o algoritmo, decorrentes da classificação dos estados em `A`, `D`, `AD` e `R`. Sejam v e w dois vértices adjacentes no grafo através da aresta a_{vw}. Em outras palavras, ou v é um vértice antecessor de w, ou v é vizinho antecessor de w.

Proposição 4.1 (Condição necessária) *Se $\delta(a_{vw}) < 0$, independente dos estados de w e v, então a_{vw} pode ser relaxada.*

Uma forma para satisfazer a condição da proposição 3.1 é atribuir um valor suficientemente negativo para $\delta(a_{vw})$, que é a variação de custo da aresta a_{vw}. Portanto, $\delta(a_{vw}) < 0$ é uma condição necessária, porém não suficiente.

Proposição 4.2 (Condição necessária) *Se $E(w) \in \{$ `A`, `AD`, `R` $\}$, independente dos estados de w, então a_{vw} pode ser relaxada.*

Outra forma para satisfazer a condição da proposição 3.1 é aumentar suficientemente o valor de $C(w)$. Isto ocorre, por exemplo, se o caminho da raiz t até w, diferente do caminho de v, contém uma aresta que aumentou consideravelmente seu custo.

Os estados `A` e `AD` em w garantem a existência de pelo menos uma aresta de custo aumentado. Se o caminho de w sofreu alguma alteração, então o estado será `R`, e mesmo assim, o novo caminho pode apresentar custo maior que o caminho original, que também caracteriza uma possibilidade de satisfazer a condição apresentada na proposição.

Proposição 4.3 (Condição necessária) *Se $E(v) \in \{$ `D`, `AD`, `R` $\}$, independente do estado de w, então a_{vw} pode ser relaxada.*

A terceira possibilidade de atender a condição da proposição 3.1 é diminuir suficientemente o valor de $C(v)$. Tipicamente, o caminho de v reduziu seu custo devido a uma aresta cujo custo diminuiu suficientemente ou a um vértice que trocou de antecessor, reduzindo assim a estimativa de custo de todo o caminho. Estes casos correspondem aos estados `D` ou `AD` e `R`

As condições suficientes são muito parecidas com as descritas para o algoritmo simplificado, na seção 3.6. A proposição 3.2 continua válida. Já a proposição 3.3 exigirá uma pequena adaptação, simplesmente trocando `Ce` por `O`, `A`, `D`, `AD` ou `R`.

4.3 Arestas agendadas para relaxação

O algoritmo dinâmico utiliza uma estrutura de dados denominada *arestas agendadas* para relaxação, simbolizada por $Ag(\cdot)$. Para cada aresta a_{vw}, $Ag(a_{vw}) \in \{\mathsf{sim}, \mathsf{não}\}$. O algoritmo dinâmico atribui $Ag(a_{vw}) = \mathsf{sim}$ quando há necessidade de relaxação da aresta em futuras iterações. Ele utiliza uma heurística para optar por omitir a relaxação de arestas, exceto quando marcadas em $Ag(\cdot)$.

Denomina-se *vizinhança aumentada*, simbolizada por $Suc_{Aum}(v)$, o conjunto de vizinhos sucessores de v que apresentam estado `A`, `AD` ou `R`. A vizinhança aumentada contém os vértices que, potencialmente, aumentaram de custo e assim satisfazem a condição da proposição 4.2.

Denomina-se *vizinhança diminuída*, simbolizada por $Suc_{Dim}(v)$, o conjunto de vizinhos sucessores de v que são alcançados através de uma aresta diminuída (dada pela operação $\mu_-(a_{vw}, \delta(a_{vw})) \in \sigma$). A vizinhança diminuída está relacionada à proposição 4.1.

Para avaliar as condições das proposições 4.1 e 4.2, será importante determinar rapidamente os vizinhos sucessores de um vértice v cujos caminhos podem ter seu custo aumentado ou diminuído. Não é permitido verificar todos os vizinhos sucessores através de visitas, uma vez que tal procedimento seria tão caro quanto relaxar todas as arestas sucessoras, justamente a operação que se deseja evitar para melhorar a eficiência do algoritmo dinâmico.

As estruturas $Suc_{Aum}(v)$ e $Suc_{Dim}(v)$ representam características dinâmicas do grafo, que não depende apenas a topologia do grafo, mas também a iteração atual do algoritmo dinâmico. É necessário estabelecer um procedimento simples e eficiente para determinar estas características.

O conceito de arestas agendadas será utilizado com este propósito. Primeiro, antes da execução do algoritmo dinâmico, todas as arestas diminuídas são marcadas em $Ag(\cdot)$. Isto garante que elas serão relaxadas confirme a proposição 4.1.

Proposição 4.4 *Para uma aresta a_{vw}, $Ag(a_{vw}) = \mathsf{sim}$ se $\mu_-(a_{vw}, \delta(a_{vw})) \in \sigma$.*

Desta forma, os vértices do conjunto $Suc_{Dim}(v)$ estão entre os vértices alcançados através de arestas marcadas em $Ag(\cdot)$

São marcadas em $Ag(\cdot)$ as arestas que entram em um vértice certificado cujo caminho potencialmente aumentou sua estimativa de custo. Isto corresponde aos vértices de estado `A`, `AD` e `R`.

Proposição 4.5 *Se o vértice v é certificado com estado `A`, `AD` ou `R`, então para $a_{wv} \in in(v)$, $Ag(a_{wv}) = \mathsf{sim}$.*

Note que são marcadas as arestas que saem de w e entram em v. Quando o vértice w for consolidado, ele relaxará as arestas sucessoras, ou seja, que saem de w. O vértice $v \in Suc_{Aum}(w)$ está contido no conjunto de vértices alcançados através de arestas marcadas em $Ag(\cdot)$.

Utilizando a mesma figura 12 para ilustrar uma outra situação, seja um vértice v com estado `A`, `AD` ou `R`, e a sub-árvore S_v, com raiz em v, da árvore de caminhos atual T_a. Quando é consolidado um vizinho antecessor z de um vértice $w \in S_N$, e se o caminho original da raiz t até w contém uma aresta aumentada, então a aresta a_{zw} deve ser relaxada devido à proposição 4.2. Os vértices do conjunto S_N não nos permitem determinar se a estimativa de custo de seu caminho pode ter aumentado, uma vez que os estados `Cr` e `nV` informam que os valores de custo não são confiáveis. Todavia, o algoritmo dinâmico deverá garantir que a aresta a_{zw} será relaxada. Se a heurística omitisse a aresta, então o algoritmo falharia caso ela pertença a um caminho mínimo.

Todos os vértices de S_v são considerados vértices que potencialmente aumentaram a estimativa de custo de seu caminho. Para cada vértice $w \in S_v$, é necessário adicionar w ao conjunto $Suc_{Aum}(z)$ para todos os vértices z que são vizinhos antecessores de w.

Proposição 4.6 *Se o vértice v é certificado com estado `A`, `AD` ou `R`, então para cada $w \in S_v$, $a_{zw} \in in(w)$, $Ag(a_{zw}) = \mathsf{sim}$.*

Não é correto simplesmente estender a proposição 4.5 para também tratar igualmente os vértices de S_v. Considerando mais uma vez a figura 12, ao consolidar um vértice z, pode ocorrer que um vizinho sucessor w pertença a S_v e onde v seja um vértice que será certificado com estado `A`, `AD`. Supondo que v ainda não foi certificado, pois a estimativa de custo de seu caminho aumentou e

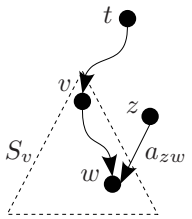

Figura 12: Sub-árvore S_v com raiz em v

supera o valor da estimativa de z, então todos os vértices de S_v apresentarão estado `nV` ou `Cr`. A condição da proposição 4.5 (na suposta versão estendida) ainda não foi satisfeita e, portanto, a aresta a_{zw} ainda não estará marcada com $Ag(a_{zw}) = \mathsf{sim}$. O algoritmo dinâmico ignoraria a relaxação de a_{zw}, potencial candidata para obter um antecessor de um caminho de menor estimativa de custo para o vértice w.

A solução consiste em determinar, junto com a inicialização, todas as sub-árvores S_v para vértices v cujo caminho contém uma aresta aumentada. Durante a execução do algoritmo dinâmico, tais vértices serão marcados com estado `A` ou `AD`. Note que, caso exista mais de uma aresta aumentada sobre o mesmo caminho, basta obter a sub-árvore correspondente à aresta aumentada mais próxima da raiz, pois ela engloba as sub-árvores relativas às outras arestas aumentadas do mesmo caminho. Para cada vértice w que pertence a uma destas sub-árvores, é necessário adicionar w ao conjunto $Suc_{Aum}(z)$ para todos os vértices z que são vizinhos antecessores de w.

A proposição 4.7 é uma generalização da proposição 4.6:

Proposição 4.7 *Se o caminho do vértice v contém uma aresta aumentada, então para cada $w \in S_v$, $a_{zw} \in in(w)$, $Ag(a_{zw}) = \mathsf{sim}$.*

Para cada vértice $w \in S_v$, temos que $w \in Suc_{Aum}(z)$ para todos os vizinhos antecessores z. Além disso, w está contido no conjunto de vértices alcançados através de arestas marcadas em $Ag(\cdot)$.

A inicialização de $Ag(\cdot)$ poderá ser realizada em $O(n + m)$ e não prejudica a complexidade assintótica do algoritmo dinâmico.

O leitor certamente notará que esta solução derivada da proposição 4.7 está incompleta, pois não considera os vértices que serão marcados com estado `R` durante a execução do algoritmo. De fato, é impossível prever quais vértices receberão estado `R`. Será mostrado no pseudo-código que o algoritmo dinâmico não será prejudicado. Um vértice com estado `R` sempre relaxa todas as arestas sucessoras, sejam elas marcadas em $Ag(\cdot)$ ou não.

4.4 Descrição do Algoritmo

O algoritmo dinâmico foi elaborado a partir do algoritmo simplificado descrito na seção 3.

Inicialização

A inicialização dos conjuntos S_C, S_V e S_N segue a descrição do algoritmo dinâmico simplificado, exceto pela raiz t. O conjunto S_C começa vazio. O conjunto S_V contém a raiz com estado `O`. Os demais vértices encontram-se no conjunto S_N. A $Ant(\cdot)$ é atribuído a mesma árvore de caminhos da solução original T_o. Os valores de $C(\cdot)$ são inicializados de acordo os custos dos caminhos de T_o e $C(t) = 0$. Arestas serão marcadas como agendadas se ela diminuiu seu custo (proposição 4.4) ou

se ela é uma aresta antecessora para um vértice que pertence à sub-árvore com raiz cujo caminho contém uma aresta que aumentou de custo (proposição 4.7).

Relaxação

A operação básica do algoritmo dinâmico continua sendo a relaxação de arestas. A cada iteração do algoritmo, o vértice v em S_V que apresenta o menor valor para $C(v)$ ou $C_C(v)$ tem o seu estado atribuído para `C` e é movido de S_V para S_C, operação chamada de *consolidação* do vértice v. Neste momento, o algoritmo relaxará somente algumas arestas específicas.

Sejam duas arestas a_{vw} e a_{zw} incidentes em w, tal que a primeira pertence a solução original T_o. Se a aresta a_{zw} é relaxada antes que a aresta a_{vw}, então a atribuição de antecessor candidato ocorre nos mesmos moldes que para o algoritmo dinâmico simplificado. Obviamente, o algoritmo dinâmico procura evitar relaxar arestas antes de a_{vw}.

Em algum momento, é relaxada a aresta a_{vw} que está em T_o. Somente neste instante, o vértice w é certificado, atribuindo $C(w)$, e movendo w de S_N para S_V. Se existir antecessor candidato para o vértice w, o custo do respectivo caminho é comparado com o custo através do antecessor v, prevalecendo como antecessor opção de menor valor. O vértice w é atribuído para um dos estados: `O`, `A`, `D`, `AD` ou `R`, decisão que depende do estado anterior de v e da variação de custo da aresta a_{vw}, ou da escolha pelo antecessor candidato.

Uma vez certificado o vértice w, as relaxação das demais arestas entrando em w ocorre normalmente, sempre observando a atribuição correta de estado `O`, `A`, `D`, `AD` ou `R` para w. O algoritmo também procura evitar relaxar arestas entrando em um vértice já certificado.

Uma implementação tradicional baseada no algoritmo de Dijkstra tenta relaxar todas as arestas saindo de cada vértice. Desta forma, o algoritmo acaba por percorrer todas as arestas do grafo. Com a premissa de que as operações do conjunto σ sobre o grafo afetam poucas arestas, é possível supor que parte significativa dos caminhos mínimos é preservada. A maioria das arestas mantém inalterada a diferença de custo dos vértices adjacentes e portanto sua relaxação não altera a árvore de caminhos.

Para selecionar o subconjunto de arestas que serão relaxadas, a heurística procede da seguinte forma, de acordo com as proposições da seção 4.2. Uma aresta diminuída a_{vw} é relaxada pois oferece uma nova possibilidade para o caminho mínimo do vértice w, de acordo com a proposição 4.1.

O mesmo é verdadeiro para a aresta a_{vw} se $C(v)$ diminuiu devido a uma aresta diminuída no caminho da raiz t até v ou devido a um vértice neste caminho que trocou seu antecessor por outro que reduz a estimativa de custo do caminho. Esta caso corresponde à proposição 4.3. Também é verdade para a aresta a_{vw} se $C(w)$ aumentou e a_{vw} está em um caminho alternativo da raiz t até w. Trata-se da proposição 4.2.

Para identificar estes casos, a heurística utiliza o conhecimento do estado dos vértices em S_V e do conjunto das arestas agendadas $Ag(\cdot)$. Com esta informação, determina-se um subconjunto das arestas cuja relaxação seja pertinente para restabelecer a árvore de caminhos mínimos. As demais arestas são ignoradas com propósito de melhorar a eficiência do algoritmo dinâmico.

4.5 Heurísticas para Seleção de Vértice

A escolha do vértice de menor custo de S_V exige uma estrutura de dados `Q`, capaz de determinar rapidamente o vértice $v \in S_V$ de menor estimativa de custo $C(v)$ ou $C_C(v)$. Implementações eficientes utilizam uma fila de prioridades, tipicamente *heap de Fibonacci*, conforme descrito em [9, 4]. Supondo que, apesar das operações do conjunto σ, a solução atualizada T_f mantém forte semelhança com a árvore de caminhos original T_o, é razoável esperar que em relação ao custo dos caminhos mínimos, a maioria dos vértices preserva a sua ordem relativa.

É interessante observar que, para construção de uma solução original T_o, por exemplo utilizando o algoritmo de Dijkstra, os vértices são adicionados a S_C em ordem crescente de custo dos caminhos. Com uma pequena modificação no algoritmo estático, ele passa a armazenar a ordenação dos vértices em uma lista linear. Se a estrutura `Q` for justamente esta lista linear, então o próximo vértice poderá ser obtido em tempo constante, bastando consultar o próximo elemento desta lista. Tampouco será necessário realizar uma ordenação inicial dos vértices de S_V. Com esta abordagem será possível retirar elementos um a um da estrutura `Q` em tempo linear em relação ao número de vértices inalterados.

No entanto, os custos dos vértices podem ser alterados pela relaxação das arestas afetadas pelas operações do conjunto σ. Estes vértices serão tratados em uma estrutura paralela, implementada como uma fila de prioridades. Quando o custo de um caminho que termina num vértice sofre alteração, então o vértice é retirado da lista linear e movido para a fila de prioridades como, por exemplo, um heap de Fibonacci. Para determinar o próximo vértice de menor custo em S_V, será necessário consultar tanto o vértice de menor custo da lista linear como o da fila de prioridades. Escolhe-se o menor entre os dois. Estas operações exigem tempo constante. Se o vértice está na lista linear, então o algoritmo ganha eficiência, pois a seleção levou tempo constante. Caso contrário, então será necessário atualizar a fila de prioridades, e somente neste caso não haverá ganho de eficiência. Outra vantagem desta heurística é gerenciar uma fila de prioridades relativamente pequena em relação ao número total de vértices no grafo G.

Ao selecionar um vértice $v \in S_V$, a heurística precisa garantir que, considerando sua estimativa de custo $C(v)$ atualizada, ele nunca será selecionado antes de vértices atualizados com estimativa de custo menor, nem depois de vértices atualizados de estimativa maior. O tratamento destas condições é bem simples: se $v \in S_V$, então o vértice é aceito. Se $v \in S_N$, então v foi selecionado considerando um valor não atualizado de $C(v)$ (e portanto não confiável). Neste caso, é certo que seu custo será necessariamente maior. O vértice é ignorado e a heurística seleciona um novo vértice. A construção do algoritmo garante que durante as próximas iterações, a atualização ou relaxação de v adicionará novamente este vértice ignorado à fila de prioridades. Note que se o vértice é extraído com estado `Cr`, então ele é antes promovido para `Ce` (no algoritmo simplificado) ou para `R` (no algoritmo dinâmico) e portanto ele é aceito como pertencente a S_V e o vértice não é ignorado.

4.6 Pseudo Código para o Algoritmo Básico

O pseudo-código da figura 13 ilustra o procedimento principal do algoritmo dinâmico. São tratados somente os casos de aumento e diminuição de custo de arestas. O algoritmo é muito semelhante ao apresentado para a versão simplificada na figura 7.

Inicialização

O pseudocódigo do procedimento `Inicializacao` é mostrado na figura 14. Ele é praticamente idêntico a inicialização do algoritmo simplificado, exceto pela atribuição do estado para a raiz, que ao invés de receber `Ce`, agora recebe `O` (linhas 1–6).

Para o algoritmo dinâmico, a inicialização inclui a marcação das arestas agendadas. Todas as arestas são marcadas como `não` (linha 7). Em seguida, as arestas que diminuíram de custo são marcadas com `sim` (linha 8). Também são marcadas como `sim` aquelas arestas que entram na sub-árvore cuja raiz está situada no final de uma aresta aumentada, de acordo com o algoritmo sugerido nas linhas 9–21.

O conjunto V_M armazena os vértices utilizados para marcar arestas. Isto evita que eles sejam marcados mais de uma vez. Para cada aresta a_{vw} que aumentou de custo, são visitados os vértices da sub-árvore com raiz na extremidade w (linhas 12–20). O pseudo-código sugere uma visita em

Figura 13: Algoritmo Dinâmico de Caminhos Mínimos

```
Procedure AlgoritmoDinâmico
   Data: Ant(·) e C(·) da solução T_o anterior
   Result: Ant(·) e C(·) atualizados para a nova solução T_f
1  Inicializacao(T_o)
   /* Consolida a raiz                                                    */
2  AtualizaSucessores(t)
3  RelaxaVizSucessora(t)
4  E(t) ← C
   /* Iteração de consolidação e certificação de vértices:               */
5  Q ← V − {t}, (ordem crescente por min{C(·), C_C(·)})
6  enquanto Q ≠ ∅ faça
7      v← Extrai(Q) /* Retira de Q o vértice com menor estimativa de custo */
8      se E(v) ∉ {nV, C} então
9          se E(v)= Cr então
10             E(v) ← R
11             Ant(v) ← Ant_C(v)
12         AtualizaSucessores(v)
13         RelaxaVizSucessora(v)
14         E(v) ← C
```

profundidade (pré-ordem), no entanto, qualquer outra ordem é válida. Os vértices a serem visitados são armazenados no conjunto F_M. Se o vértice z (extraído de F_M) já foi utilizado anteriormente para marcar as arestas, então ele e sua sub-árvore correspondente são ignorados (linha 15). Caso contrário, o vértice z é adicionado à lista V_M de vértices utilizados. As arestas entrando em z são marcadas (linhas 17–18) e os sucessores são adicionados ao conjunto F_M de vértices a serem visitados (linhas 19–20).

Certificação

O novo pseudocódigo do procedimento `AtualizaSucessores` é mostrado na figura 15. Ele recebe como parâmetro o vértice v. Seu objetivo é atualizar o custo dos vértices sucessores $w \in Suc(v)$, tendo em vista o novo valor $C(v) + c(a_{vw}) + \delta(a_{vw})$ na árvore atual T_a.

O algoritmo ignora os vértices consolidados (linha 2), pois seu custo mínimo já é conhecido através de outro caminho. Em seguida, o procedimento calcula o custo confiável do vértice sucessor w (variável c, linha 3), considerando o caminho através de v.

O procedimento verifica se já existe um outro vértice antecessor candidato para w, através do qual é possível obter um caminho com custo menor que através de v. Havendo esta possibilidade, o procedimento redefine o antecessor de w como sendo este o antecessor candidato (linhas 6-7). Caso contrário, o estado do vértice sucessor é determinado nas linhas 9-19, variável e, segundo descrito na seção 4.7.1. O vértice w recebe seu custo confiável cujo valor está armazenado na variável c (linhas 21-22).

Se w pertence à fila Q, então sua posição precisa ser corrigida (linha 23), para manter a ordenação de Q em relação à $\min\{C(\cdot), C_C(\cdot)\}$. Caso contrário, w é inserido na devida posição.

Figura 14: Algoritmo Dinâmico de Caminhos Mínimos

Procedure `Inicializacao`(T_o)

1 $Ant(\cdot) \leftarrow$ antecessores de T_o
2 $C(\cdot) \leftarrow$ custos dos caminhos de T_o

3 **para cada** $v \in V$ **faça** $Ant_C(v) \leftarrow \varnothing$
4 **para cada** $v \in V$ **faça** $C_C(v) \leftarrow \infty$

5 **para cada** $v \in V - \{t\}$ **faça** $E(v) \leftarrow$ `nV`
6 $E(t) \leftarrow$`0`

/* Marca as arestas agendadas */
7 **para cada** $a_{vw} \in A$ **faça** $Ag(a_{vw}) \leftarrow$ `não`
8 **para cada** $\mu_-(a_{vw}, \delta(a_{vw})) \in \sigma$ **faça** $Ag(a_{vw}) \leftarrow$ `sim`
9 $V_M \leftarrow \varnothing$
10 **para cada** $\mu_+(a_{vw}, \delta(a_{vw})) \in \sigma$ **faça**
11 $w \leftarrow \text{end}(a_{vw})$
12 $F_M \leftarrow \{w\}$
13 **enquanto** $F_M \neq \varnothing$ **faça**
14 $z \leftarrow$ primeiro elemento de F_M
15 **se** $z \notin V_M$ **então**
16 $V_M \leftarrow V_M \cup \{z\}$
17 $A_M \leftarrow \text{in}(w)$
18 **para cada** $a_{vw} \in A_M$ **faça** $Ag(a_{vw}) \leftarrow$ `sim`
19 $A_S \leftarrow \text{out}(w)$
20 **para cada** $a_{zv} \in A_S$ **faça** $F_M \leftarrow \{\text{end}(a_{zv})\}$
21 $V_M \leftarrow V_M \cup \{w\}$

Figura 15: Atualização de vértice sucessor

Procedure `AtualizaSucessores`(v)

Data: Vértice v
Result: O custo, o antecessor e o estado dos vértice sucessores de v são atualizados

```
1  para cada w ∈ Suc(v) faça
2      se E(w) = C então reinicia com o próximo vértice de Suc(v)
       /* Custo atualizado através da aresta a_vw                    */
3      c ← C(v) + c(a_vw) + δ(a_vw)
       /* Atualização do vértice w                                   */
4      se E(w) = Cr
5      e C_C(w) < c então
           /* Prevalece o antecessor candidato                       */
6          E(w) ← R
7          Ant(w) ← Ant_C(w)
8      senão
           /* Decisão para o estado do vértice sucessor w            */
9          se δ(a_vw) < 0 então
10             se E(v) = A então
11                 e ← AD
12             senão
13                 e ← D
14         senão, se δ(a_vw) > 0 então
15             se E(v) = D então
16                 e ← AD
17             senão
18                 e ← A
19         senão
20             e ←E(v)
           /* Prevalece o antecessor v                               */
21         E(w) ← e
22         C(w) ← c
23         Corrige a posição de w na fila Q
```

Relaxação

O novo procedimento `RelaxaVizSucessora`, cujo pseudocódigo é mostrado na figura 16, recebe como parâmetro um vértice v. Um subconjunto dos vizinhos sucessores $Viz_{Suc}(v)$, são verificados para determinar se v serve como vértice antecessor que reduz a estimativa de custo do caminho para o vértice w. Tenta-se evitar visitar vizinhos, pois esta operação é um das principais parcelas na complexidade do algoritmo.

Primeiro, o algoritmo determina o conjunto N, que contém os sucessores que potencialmente poderão ter suas estimativas de custo reduzidas quando se atribui v seu como antecessor (linhas 1-5). Estas condições são discutidas mais adiante, de acordo com a lógica que será descrita na seção 4.7.2.

Em seguida, o procedimento calcula o custo alternativo do vértice w quando o caminho passa através de v (variável `c`, linha 7). A relaxação é realizada nas linhas 8-22.

Se o custo do vértice w não é certificado (ele apresenta estado `nV` ou `Cr`), então o procedimento ainda não é capaz de avaliar se o caminho alternativo através de v apresenta estimativa de custo vantajosa. Isto é indicado colocando v como antecessor candidato de w, em $Ant_C(w)$, postergando a decisão. Se w já possui um antecessor candidato (estado `Cr`), então o caminho alternativo é substituído por v, somente se ele apresenta estimativa de custo menor que o antecessor candidato atual em $Ant_C(w)$. Vértices consolidados são ignorados (linha 17).

Para os demais estados (`O`, `A`, `D`, `AD` e `R`), o procedimento utiliza v como novo antecessor somente se ele resulta em um caminho de uma estimativa de custo menor que o do antecessor atual em $Ant(w)$.

Se w pertence à fila `Q`, então sua posição precisa ser corrigida (linha 23), para manter a ordenação de `Q` em relação à $\min\{C(\cdot), C_C(\cdot)\}$. Caso contrário, w é inserido na devida posição.

4.7 Lógica de atribuição de estados

Por uma questão de conveniência, serão definidas operações que resumem os procedimentos necessários para decidir a atribuição de um estado a um vértice. Estas operações serão utilizadas na descrição das funções `AtualizaSucessores` e `RelaxaVizSucessora`. Cada uma das operações recebe uma aresta a_{vw} como parâmetro, tal que $v = \text{start}(a_{vw})$ e $w = \text{end}(a_{vw})$.

- `O`(a_{vw}): Atribui estado `O` para o vértice w e torna v seu antecessor através da aresta a_{vw}. O custo de w é calculado pela soma do custo do caminho da raiz t até o vértice v somado ao novo custo da aresta a_{vw}.

  1 $E(w) \leftarrow$ `O`
  2 $C(w) \leftarrow C(v) + c(a_{vw}) + \delta(a_{vw})$
  3 $Ant(w) \leftarrow v$

- `Oc`(a_{vw}): Semelhante à `O`(a_{vw}), mas assume-se que $E(w) =$ `Cr`. A operação será realizada somente se o caminho do vértice w através da aresta a_{vw} apresentar custo menor que o custo candidato. Caso contrário, o vértice w mantém seu estado, seu custo candidato e seu antecessor candidato.

  1 **se** $C(v) + c(a_{vw}) + \delta(a_{vw}) < C_C(w)$ **então**
  2 $E(w) \leftarrow$ `O`
  3 $C(w) \leftarrow C(v) + c(a_{vw}) + \delta(a_{vw})$
  4 $Ant(w) \leftarrow v$
  5 **senão**
  6 $E(w)$, $C_C(w)$ e $Ant_C(w)$ permanecem inalterados.

Figura 16: Relaxação de vizinho sucessor

Procedure `RelaxaVizSucessora`(v)

Data: Vértice v
Result: São modificados os caminhos de vizinhos para os quais v é um antecessor de caminho de menor custo.

```
   /* Determinar os vizinhos que devem ser visitados                          */
1  se E(v) ∈ {R, AD, D} então
2  |  N ← Viz_Suc(v)
3  senão
4  |  N ← Suc_Aum(v) ∪ Suc_Dim(v)
   |  /* Ou, usando arestas agendadas:                                       */
5  |  N ← { w | w = end(a_vw), a_vw ∈ out(v), Ag(a_vw) = sim }

   /* Verifica se existe a possibilidade do vizinho reduzir o custo através de v */
6  para cada w ∈ N faça
7  |  c ← C(v) + c(a_vw) + δ(a_vw)
8  |  se E(w) = nV então
   |  |  /* O vértice v torna-se antecessor candidato                        */
9  |  |  E(w) ← Cr
10 |  |  C_C(w) ← c
11 |  |  Ant_C(w) ←v
12 |  senão, se E(w) = Cr então
13 |  |  se c < C_C(w) então
   |  |  |  /* Substitui o antecessor candidato por outro de menor custo    */
14 |  |  |  C_C(w) ← c
15 |  |  |  Ant_C(w) ←v
16 |  senão, se E(w) = C então
17 |  |  reinicia com o próximo vértice de N
18 |  senão
19 |  |  se c < C(w) então
   |  |  |  /* Relaxa vértice w                                            */
20 |  |  |  C(w) ← c
21 |  |  |  Ant(w) ←v
22 |  |  |  E(w) ← R
23 |  |  Corrige a posição de w na fila Q
```

- $\mathsf{A}(a_{vw})$, $\mathsf{D}(a_{vw})$, $\mathsf{AD}(a_{vw})$ e $\mathsf{R}(a_{vw})$: As operações são idênticas à $\mathsf{O}(a_{vw})$, trocando o estado `O`, respectivamente, por `A`, `D`, `AD` e `R`.

- $\mathsf{A_C}(a_{vw})$, $\mathsf{D_C}(a_{vw})$, $\mathsf{AD_C}(a_{vw})$ e $\mathsf{R_C}(a_{vw})$: As operações são idênticas à $\mathsf{O_C}(a_{vw})$, trocando o estado `O`, respectivamente, por `A`, `D`, `AD` e `R`.

- $\mathsf{R_N}(a_{vw})$: Semelhante com $\mathsf{R_C}(a_{vw})$. A operação será realizada somente o caminho do vértice w através da aresta a_{vw} apresentar custo menor que o caminho através do antecessor atual. Caso contrário, o vértice w mantém seu estado, seu o custo do caminho e seu antecessor. *A diferença em relação à $\mathsf{R_C}(a_{vw})$ está na comparação com o caminho atual, e não o caminho através do antecessor candidato.*

  se $C(v) + c(a_{vw}) + \delta(a_{vw}) < \mathbf{C(w)}$ **então**
  - $E(w) \leftarrow$ `R`
  - $C(w) \leftarrow C(v) + c(a_{vw}) + \delta(a_{vw})$
  - $Ant(w) \leftarrow v$

  senão
  - $E(w)$, $C(w)$ e $Ant(w)$ permanecem inalterados.

- $\mathsf{Cr}(a_{vw})$: Assume-se que $E(w) =$ `nV`. Atribui o estado `Cr` ao vértice w e torna v seu antecessor candidato através da aresta a_{vw}. O custo candidato de w é calculado pela soma do custo do caminho da raiz t até o vértice v somado ao novo custo da aresta a_{vw}.

  $E(w) \leftarrow$ `Cr`
  $C_C(w) \leftarrow C(v) + c(a_{vw}) + \delta(a_{vw})$
  $Ant_C(w) \leftarrow v$

- $\mathsf{Cr_N}(a_{vw})$: Semelhante à $\mathsf{Cr}(a_{vw})$, mas assume-se que $E(w) =$ `Cr`. A operação será realizada somente o caminho do vértice w através da aresta a_{vw} apresentar custo menor que o custo candidato. Neste caso, v é um antecessor candidato que reduz o custo candidato. Caso contrário, o vértice w mantém seu estado, seu custo candidato e seu antecessor candidato.

  se $C(v) + c(a_{vw}) + \delta(a_{vw}) < C_C(w)$ **então**
  - $E(w)$ continua `Cr`
  - $C_C(w) \leftarrow C(v) + c(a_{vw}) + \delta(a_{vw})$
  - $Ant_C(w) \leftarrow v$

  senão
  - $E(w)$, $C_C(w)$ e $Ant_C(w)$ permanecem inalterados.

4.7.1 Lógica do procedimento AtualizaSucessores

A tabela 1 resume o comportamento do procedimento `AtualizaSucessores` para visitar os vértices sucessores. A linha representa um possível estado do vértice v que foi extraído da fila Q. Seja w um sucessor de v na solução T_a, a_{vw} a aresta de v a w e $\delta(a_{vw})$ a variação de custo da aresta a_{vw}. As colunas correspondem aos possíveis estados dos vértices sucessores w, alcançados através da aresta a_{vw}. Cada elemento da tabela informa uma ou mais condições para operações elementares descritas anteriormente. Considera-se ainda, para a coluna `Cr`, o vértice w_{ac} como antecessor candidato de w através da aresta a_{wac}.

Os casos das linhas `nV` e `C` não ocorrem na heurística. Vértices extraídos da fila Q são ignorados pelo algoritmo dinâmico quando apresentam estado `nV` ou `C`.

Os casos da linha Cr não existem. Quando o algoritmo dinâmico extrai da fila Q um vértice com estado Cr, ele é promovido para o estado R.

A coluna nV corresponde a vértices sucessores w que serão certificados sem antes possuirem um antecessor candidato definido. Note que um vértice sucessor w recebe estado A quando passa a existir em seu caminho uma aresta de custo aumentado. Recebe estado D quando passa a existir uma aresta de custo diminuído. No entanto, quando o caminho passa a possuir tanto uma aresta de custo aumentado, como uma de custo diminuído, o estado do sucessor w será AD. Caso não haja nem aresta de custo aumentado, nem de custo diminuído, o estado do sucessor w continua em O.

A exceção ocorre quando o vértice v apresenta estado R, ou seja, em seu caminho existe um vértice que mudou de antecessor. Neste caso, não interessa mais para o algoritmo se existem arestas de custo aumentado ou diminuído, pois não é possível ponderar a variação de custo em relação a mudanças de valores das arestas do caminho original.

A coluna Cr apresenta a mesma lógica que a coluna nV. Entretanto, o vértice sucessor w já possui um antecessor candidato. Neste caso, é necessário verificar qual opção resulta no caminho de menor custo para w: o antecessor candidato ou o vértice v. Caso a melhor opção seja o vértice v, então w é certificado com um dos O, A, D, AD ou R, tal como na coluna nV. Senão, o antecessor candidato torna-se o antecessor e o vértice é marcado com estado R.

As colunas O, A, D, AD e R não foram especificadas, uma vez que pela construção do algoritmo, somente os já certificados podem apresentar estes estados.

4.7.2 Lógica do procedimento RelaxaVizSucessora

A tabela 2 resume o comportamento do procedimento RelaxaVizSucessora para visitar os vizinhos sucessores. A linha representa um possível estado do vértice v que foi extraído da fila Q. Seja w um vizinho sucessor de v na solução T_a, a_{vw} a aresta de v a w e $\delta(a_{vw})$ a variação de custo da aresta a_{vw}. As colunas correspondem aos possíveis estados dos vizinhos sucessores w. Cada elemento da tabela informa uma ou mais condições para operações elementares descritas anteriormente.

Os casos das linhas nV e C não ocorrem na heurística. Vértices extraídos da fila Q são ignorados pelo algoritmo dinâmico quando apresentam estado nV ou C.

Os casos da linha Cr não existem. Quando o algoritmo dinâmico extrai da fila Q um vértice com estado Cr, ele é promovido para o estado R.

As colunas O, A, D, AD e R representam casos quando a heurística decide sobre uma aresta para um vizinho sucessor que já foi certificado e ele apresenta caminho com custo confiável. Para ocorrer a relaxação, é necessário que pelo menos uma das três condições seja satisfeita: a aresta a_{vw} diminuiu de custo, o caminho do vizinho sucessor w aumentou seu custo ou diminuiu o custo do caminho do vértice v.

As primeiras duas condições são as arestas agendadas quando v apresenta estado O ou A. A terceira condição é o vértice v com estado D, AD ou R. Se pelo menos uma for satisfeita, então verifica-se qual antecessor resulta no caminho de menor custo: o antecessor atual de w ou o vértice v. O custo do caminho de w é atribuído de acordo.

A coluna nV descreve os casos quando a heurística decide sobre uma aresta para um vizinho sucessor que ainda não foi visitado. Para ocorrer a relaxação, é necessário que pelo menos uma das três condições seja satisfeita: a aresta a_{vw} diminuiu de custo, o caminho do vizinho sucessor w aumentará seu custo ou o caminho do vértice v diminuiu de custo.

As primeiras duas condições são as arestas agendadas quando v apresenta estado O ou A. A terceira condição é o vértice v com estado D, AD ou R. Se uma delas for satisfeita, então vértice v é definido como antecessor candidato de w e o custo candidato do caminho de w é atribuído de acordo.

v	w							
x	nV	C	O	A	D	AD	R	Cr
nV	ignora	ignora	–	–	–	–	–	ignora
C	ignora	ignora	–	–	–	–	–	ignora
O	se $\delta(a_{vw}) < 0$: $\mathsf{D}(a_{vw})$ se $\delta(a_{vw}) = 0$: $\mathsf{O}(a_{vw})$ se $\delta(a_{vw}) > 0$: $\mathsf{A}(a_{vw})$	ignora	–	–	–	–	–	se $\delta(a_{vw}) < 0$: $\mathsf{D_C}(a_{vw})$ se $\delta(a_{vw}) = 0$: $\mathsf{O_C}(a_{vw})$ se $\delta(a_{vw}) > 0$: $\mathsf{A_C}(a_{vw})$ Se inalterado: $\mathsf{R}(a_{wac})$
A	se $\delta(a_{vw}) < 0$: $\mathsf{AD}(a_{vw})$ se $\delta(a_{vw}) = 0$: $\mathsf{A}(a_{vw})$ se $\delta(a_{vw}) > 0$: $\mathsf{A}(a_{vw})$	ignora	–	–	–	–	–	se $\delta(a_{vw}) < 0$: $\mathsf{AD_C}(a_{vw})$ se $\delta(a_{vw}) = 0$: $\mathsf{A_C}(a_{vw})$ se $\delta(a_{vw}) > 0$: $\mathsf{A_C}(a_{vw})$ Se inalterado: $\mathsf{R}(a_{wac})$
D	se $\delta(a_{vw}) < 0$: $\mathsf{D}(a_{vw})$ se $\delta(a_{vw}) = 0$: $\mathsf{D}(a_{vw})$ se $\delta(a_{vw}) > 0$: $\mathsf{AD}(a_{vw})$	ignora	–	–	–	–	–	se $\delta(a_{vw}) < 0$: $\mathsf{D_C}(a_{vw})$ se $\delta(a_{vw}) = 0$: $\mathsf{D_C}(a_{vw})$ se $\delta(a_{vw}) > 0$: $\mathsf{AD_C}(a_{vw})$ Se inalterado: $\mathsf{R}(a_{wac})$
AD	$\mathsf{AD}(a_{vw})$	ignora	–	–	–	–	–	$\mathsf{AD_C}(a_{vw})$ Se inalterado: $\mathsf{R}(a_{wac})$
R	$\mathsf{R}(a_{vw})$	ignora	–	–	–	–	–	$\mathsf{R_C}(a_{vw})$ Se inalterado: $\mathsf{R}(a_{wac})$
Cr	ignora	ignora	–	–	–	–	–	ignora

Tabela 1: Tabela de operações para atualizar sucessores na árvore atual

v	w							
x	nV	C	O	A	D	AD	R	Cr
nV	ignora	ignora	ignora	ignora	ignora	ignora	ignora	ignora
C	ignora	ignora	ignora	ignora	ignora	ignora	ignora	ignora
O	$seAg(a_{vw})$: $\mathsf{Cr}(a_{vw})$	ignora	$seAg(a_{vw})$: $\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$seAg(a_{vw})$: $\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$seAg(a_{vw})$: $\mathsf{R_N}(a_{vw})$	$seAg(a_{vw})$: $\mathsf{Cr_N}(a_{vw})$
A	$seAg(a_{vw})$: $\mathsf{Cr}(a_{vw})$	ignora	$seAg(a_{vw})$: $\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$seAg(a_{vw})$: $\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$seAg(a_{vw})$: $\mathsf{R_N}(a_{vw})$	$seAg(a_{vw})$: $\mathsf{Cr_N}(a_{vw})$
D	$\mathsf{Cr}(a_{vw})$	ignora	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{Cr_N}(a_{vw})$
AD	$\mathsf{Cr}(a_{vw})$	ignora	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{Cr_N}(a_{vw})$
R	$\mathsf{Cr}(a_{vw})$	ignora	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{R_N}(a_{vw})$	$\mathsf{Cr_N}(a_{vw})$
Cr	ignora	ignora	ignora	ignora	ignora	ignora	ignora	ignora

Tabela 2: Tabela de visitas em vizinhos sucessores

A coluna Cr corresponde ao caso quando a heurística decide sobre uma aresta para um vizinho sucessor que já foi visitado, mas ainda não foi certificado e o custo de seu caminho ainda não é confiável. As condições são as mesmas três que para a coluna nV. Se uma delas for satisfeita, então verifica-se qual antecessor candidato resulta no caminho de menor custo candidato: o antecessor candidato atual ou vértice v. O custo candidato do caminho de w é atribuído de acordo.

4.7.3 Observações sobre as tabelas

Analisando as tabelas que descrevem a lógica do procedimento RelaxaVizSucessora e do procedimento AtualizaSucessores, observa-se que ambos podem atribuir um vértice para o estado R. Esta atribuição para o estado R é imutável até o algoritmo dinâmico consolidar o vértice. Enquanto encontra-se no estado R o vértice pode receber novas atribuições antecessores, conforme vizinhos são descobertos com caminhos alternativos de menor custo.

O procedimento RelaxaVizSucessora apresenta somente operações que alteram o antecessor (ou antecessor candidato) de um vértice, caso ela resulte em redução do custo (ou custo candidato) do caminho.

Nota-se também que um vértice poderá ser marcado com os estados O, A, D e AD exclusivamente por seu antecessor da árvore atual, nunca por um outro vizinho antecessor.

5 Complexidade do Algoritmo

5.1 Complexidade de espaço

As arestas e os vértices armazenam as operações de modificações que afetam os mesmos. Esta informação ocupa espaço linear em relação ao número de componentes do grafo. Em cada vértice, o vértice antecessor, o antecessor candidato e a estimativa de custo do caminho requerem espaço constante. O conjunto de sucessores, a lista de sucessores aumentados, e as listas de vizinhos antecessores e vizinhos sucessores exigem, para cada vértice, espaço máximo igual ao grau do grafo. Em cada aresta, a marca de aresta agenda requer espaço constante. A fila Q que determina a ordem em que os vértices são consolidados apresenta tamanho linear em relação ao número de vértices.

Proposição 5.1 *A quantidade de memória necessária para o algoritmo dinâmico é $O(m+kn)$ para um grafo com grau k.*

5.2 Complexidade de tempo

Na literatura, são utilizadas duas formas para estudar a complexidade dos algoritmos dinâmicos para árvores de caminhos mínimos. A abordagem tradicional mede em relação ao tamanho da entrada. Conta-se tipicamente o número de comparações entre custos de caminhos (ou seja, número de arestas relaxadas) e número de operações (inserção, atualização e extração) na fila de prioridades que controla a ordenação do vértices. Os algoritmos dinâmicos apresentam, no pior caso, a mesma complexidade que o algoritmo estático.

Conforme discutido em [22], medida tradicional dificulta a comparação entre algoritmos dinâmicos, pois ela fornece poucas informações de como os componentes afetados pelas modificações influenciam o número de operações do algoritmo. A diferença entre estes algoritmos está no tratamento do melhor caso e do caso médio.

Ao outra abordagem mede a complexidade em relação ao número de componentes do grafo que são afetados pelas modificações das arestas.

Considera-se que obter informações sobre um elementos do grafo envolve um número aproximadamente igual de acessos à memória e de operações aritméticas. Assume-se complexidade $O(1)$ para estas consultas. Extrair o próximo vértice da fila `Q` será $O(1)$ caso ele se encontre na lista ordenada e $O(\log n)$ se está na fila de prioridades (de Fibonacci). A atualização de custo ou a inserção de um elemento é $O(1)$ em tempo amortizado, conforme descrito em [9].

Proposição 5.2 *A complexidade assintótica do tempo execução do algoritmo dinâmico para caminhos mínimos é* $O(m + n \log n)$.

A inicialização de $Ant(\cdot)$, $Ant_C(\cdot)$, $E(\cdot)$ e $C(\cdot)$ é realizada uma única vez para cada vértice. A atribuição de $Ag(\cdot)$ ocorre uma única vez para cada aresta. A complexidade resultante será $O(n+m)$.

Em relação a inicialização das arestas agendadas de acordo com as operações de σ, realiza-se uma atribuição para cada aresta diminuída (até m arestas) e uma atribuição de sub-árvore para cada aresta aumentada (até n sub-árvores). Uma sub-árvore percorre até n vértices, cada qual resulta em até k atribuições de arestas agendadas, onde k é o grau do grafo. Isto resultaria em complexidade $O(mnk)$, se não fosse a implementação cuidadosa da inicialização. Quando ocorre sobreposição de sub-árvores, os vértices e as arestas da sobreposição são atribuídas apenas uma única vez. No total, serão percorridos no máximo n vértices e atribuídas m arestas, ou seja, complexidade $O(n + m)$.

Durante toda a execução, chama-se o procedimento `AtualizaSucessores` e o procedemento `RelaxaVizSucessora` somente uma única vez para cada vértice. A iteração do algoritmo no pior caso, extrai duas vezes cada vértice: da lista ordenada (erroneamente devido estimativa de custo não atualizada) e da fila de prioridades. Para filas de prioridade de Fibonacci, a extração tem complexidade $O(\log n)$. Predomina para todos os vértices no pior caso, $O(n \log n)$.

O procedimento `AtualizaSucessores` visita todos os sucessores na árvore atual T_a, atualiza as estimativas de custos e estado. É interessante notar que um vértice é certificado no máximo uma vez, independente do número de chamadas de `AtualizaSucessores`, e independente das variações de custo das componentes do grafo. A árvore atual T_a apresenta sempre n vértices e $n - 1$ arestas. No pior caso, todos os vértices precisam ser adicionados ou re-posicionados fila de prioridades. Na fila de prioridades de Fibonacci, estas duas operações têm complexidade $O(1)$. A complexidade da certificação de todos os vértices, é $O(n)$.

O procedimento `RelaxaVizSucessora`, no pior caso, relaxa todas as arestas para vizinhos sucessores e re-posiciona ou adiciona todos os vértices vizinhos na fila de prioridades. A complexidade, para todos os vértices, é $O(n + m)$, assumindo m como limitante superior para o número de arestas relaxadas.

Desta forma, a execução completa do algoritmo requer tempo proporcional a n, m e $n \log n$, ou seja $O(m + n \log n)$.

Proposição 5.3 *O algoritmo dinâmico para caminhos mínimos é assintoticamente melhor ou igual ao algoritmo estático.*

Em [4], a complexidade do algoritmo estático de Dijkstra é mostrada como $O(m + n \log n)$. O pior caso dos dois algoritmos é igual.

No entanto, o algoritmo de Dijkstra sempre realiza $O(m + n \log n)$ operações, sendo que o algoritmo dinâmico apresentado neste trabalho, no melhor caso, realiza $O(m + n)$ operações.

5.3 Componentes afetados no grafo

Seja n_a o número de vértices que devido alguma operação em σ mudaram o custo ou o percurso do caminho até a raiz t.

Proposição 5.4 *A complexidade assintótica do algoritmo dinâmico para caminhos mínimos é* $O(m + n + n_a \log n_a)$.

A inicialização e a iteração do algoritmo sempre realizam $O(n + m)$ iterações. Somente os vértices afetados são adicionados à fila de prioridades, ou seja n_a operações extração. As demais extrações ocorrem sobre a lista ligada em tempo constante para cada vértice. Todas as chamadas do procedimento `AtualizaSucessores` juntas, sempre executam $O(n)$ operações, independente do número de vértices afetados. A iteração do procedimento `RelaxaVizSucessora` é executada para todas as arestas que reduziram de custo, e que apresentam uma das extremidades com vértice afetado, com complexidade assintótica $O(n + n_a k)$.

Considerando que $n_a k = O(m)$ e somando todas as operações, obtém-se complexidade $O(m + n + n_a \log n_a)$.

5.4 Comparação com algoritmos da bibliografia

O algoritmo proposto será comparado com os trabalhos [20] e [14]. Assume-se um grafo G, de grau k, com b arestas alteradas. Note que $b = O(m)$.

Primeiro, será analisado o algoritmo de algoritmo de [20] em relação ao número de operações realizadas. Ele é composto por duas fases: inicialização e execução. Para uma aresta modificada a e uma árvore de caminhos T, define-se $T(a)$ como sendo uma subárvore de T com raiz em start(a). A inicialização atualiza as estimativas de custos dos vértices de $T(a)$ (conforme a variação de custo de a). Isto resulta em $|T(a)|$ vértices visitados.

Seja M o conjunto de arestas de fronteira entre vértices de $T(a)$ e os demais vértices. O objetivo da inicialização é determinar se o vértice de uma das extremidades de uma aresta de M permite melhorar a estimativa de custo da outra extremidade. Caso afirmativo, o vértice é adicionado a uma fila de prioridades, a ser manipulada na execução. Percorre-se as arestas incidentes a um vértice de $T(a)$ para determinar quais delas pertencem a M, acarretando um total de $|T(a)|k$ arestas e $|T(a)|$vértices. Note que $O(|T(a)|) = n$. A complexidade total da inicialização para a aresta a será $O(nk)$, já contando os de vértices adicionados à fila de prioridade. Note que m é uma aproximação mais justa para o número de arestas visitadas que nk. É necessário realizar a inicialização para cada aresta modificada. O número de arestas percorridas e vértices visitados é $O(bm)$ para a inicialização com b modificações.

O algoritmo precisa ser executado duas vezes: uma apenas para arestas de custo aumentado, outra para as arestas de custo diminuído. A complexidade da fase de execução é $O(m + n \log n)$.

Consolidando os resultados da inicialização e da execução, obtém-se para [20]: $O(bm + m + n \log n)$. Como $b = O(m)$, a complexidade das duas etapas pode ser reescrita como $O(m^2 + n \log n)$. Fica evidente o ganho do algoritmo proposto neste trabalho, que não necessita a fase de inicialização para cada aresta modificada.

No algoritmo apresentado em [14] não existe uma discussão clara sobre a complexidade de modificações em múltiplas arestas. A complexidade para ele é $O(m \log n)$ para uma operação de aumento ou uma de diminuição de custo, podendo ser melhorado para $O(m + n \log n)$ com heap de Fibonacci. Para modificações de b arestas, é necessário executar o algoritmo b vezes, resultando em complexidade $O(b(m + n \log n)) = O(m^2 + mn \log n)$.

6 Conclusão

Este trabalho apresenta um algoritmo dinâmico para restabelecer árvores de caminhos mínimos após o gráfico sofrer várias perturbações simultâneas que afetam o custo das arestas. A complexidade assintótica é $O(m + n \log n)$ para o pior caso, e $O(m + n)$ para o melhor. No pior caso, o algoritmo dinâmico possui eficiência igual ao do algoritmo estático. Para um grafo no qual n_a vértices são afetados pelas modificações de custo nas arestas, a complexidade será $O(n + m + n_a \log n_a)$.

O algoritmo possui a característica inovadora de utilizar um único modelo para lidar simultaneamente com alterações de aumento e de diminuição de custo nas arestas. Ele distingue-se dos algoritmos estudados na literatura por restabelecer a árvores de caminhos mínimos do vértices afetados em um único passo. A ausência de um procedimento elaborado de inicialização ou pré-processamento permitiu reduzir a complexidade assintótica em um fator de m.

Nos algoritmos de caminhos mínimos, a complexidade assintótica é influenciada principalmente pelo número de operações da fila ordenada de vértices. Ao contrário do algoritmo dinâmicos estudados na literatura, que procuram visitar somente os vértices afetados e assim evitar acessos à fila, o algoritmo dinâmico apresentado neste trabalho visita todos os vértices. Isto é possível pois o modelo de fila utilizado permite extrair eficientemente aqueles que não foram afetados pelas modificações de custo das arestas. Para isso, o algoritmo dinâmico requer como entrada os custos da árvore original de caminhos.

O uso do algoritmo dinâmico apresentado neste trabalho é vantajoso quando uma fração das arestas sofre perturbações de custo, de preferência, se as perturbações ocorrem em arestas próximas. Ele também é interessante para casos onde existem, simultaneamente, arestas que aumentam e outras que diminuem de custo. Para um grafo onde ocorre uma modificação para apenas uma única aresta, já existem algoritmo eficientes, como descritos em [14] ou [20].

O problema dinâmico de caminhos mínimos poderia ser estendido para aceitar outros tipos de operações que afetam o grafo, tais como: custos em vértices, aumento e diminuição de custo nos vértices, inserção e remoção de arestas e inserção e remoção de vértices.

Afirma-se que é possível repassar o custo dos vértices para as arestas adjacentes, de forma a utilizar uma adaptação do mesmo algoritmo dinâmico apresentado para tratar o aumento e a diminuição de custo em vértices. Acredita-se também que a adição de novas arestas possa ser tratada de forma semelhante à operação de redução de custo de aresta. E que a remoção de uma aresta seja similar à operação de aumentar o custo de aresta. Já a remoção de um vértice seria equivalente à remoção das arestas adjacentes. A inserção de vértice, por sua vez, equivalente à adição das arestas adjacentes.

A heurística que decide a relaxação de uma aresta ainda é bastante conservadora. Na forma atual, ela prefere relaxar mais arestas que o necessário a adotar um modelo mais preciso para descrever o estado dos vértices. Por exemplo, o estado `AD` indica que o caminho do vértice contém tanto uma aresta aumentada, como uma diminuída. Com apenas esta informação, a heurística desconhece se

o custo resultante do caminho é de fato maior ou menor que o custo original. A heurística relaxa arestas de acordo com as duas possibilidades, sendo que apenas uma delas seria realmente necessária.

O estado `R` também é utilizado de forma ambígua. Ele indica que o caminho do vértice é diferente do caminho original, mas não a causa por que algum vértice do caminho trocou de antecessor. Existem duas causas:

- Um caminho que reduziu seu custo tende a propagar esta redução para os vértices próximos, que poderão mudar de antecessor. Neste caso, estes vértices apresentarão um novo caminho cujo custo é *menor* que seu custo original.
- Vértices sobre um caminho que aumentou de custo tendem a mudar de antecessor para caminhos próximos. Neste caso, o novo custo é reduzido, mas continua *maior* que seu custo original.

Para um vértice de estado `R`, a heurística relaxa todas as arestas, sendo que poderia utilizar o conhecimento do custo do novo caminho em relação ao custo original para relaxar apenas um subconjunto das arestas.

A marcação de arestas agendadas é outra característica que permite melhorias através de um estudo mais aprofundado. A inicialização do algoritmo dinâmico marca muito mais arestas que realmente necessário para restabelecer a árvore de caminhos mínimos.

Ocorre que, se uma aresta aumenta de custo, o algoritmo dinâmico identifica a sub-árvore com raiz no final desta aresta, e marca todas as arestas que entram em vértices desta sub-árvore. No entanto, para todos os vértices desta sub-árvore, custo do caminho aumenta em igual quantidade. Seria necessário marcar somente arestas situadas na fronteira desta sub-árvore, mais especificamente, as arestas que procedem de vértices cujo caminho aumentou seu custo em menor quantidade (isto inclui também vértices cujo caminho manteve o custo inalterado ou que o diminuiu). As demais arestas contidas na própria sub-árvore poderiam ser ignoradas, de forma que menos arestas seriam relaxadas.

O mesmo argumento é válido para propor um tratamento mais inteligente para o estado `D`. Quando uma aresta diminui de custo, o algoritmo poderia identificar a sub-árvore com raiz no final desta aresta. Seria interessante relaxar somente as arestas que saem desta sub-árvore para vértices cujo caminho diminuiu o custo em menor quantidade (isto inclui também vértices cujo caminho manteve o custo inalterado ou que o aumentou). As demais arestas da sub-árvore seriam ignoradas.

Isto exigirá uma reflexão sobre casos quando arestas sobre um mesmo caminho sofrem alterações. Por exemplo, é possível as duas arestas aumentem de custo, definindo duas sub-árvores de vértices cujo caminho aumentou de custo. Uma sub-árvore está contida na outra e a sub-árvore interna inclui o aumento da sub-árvore externa. É necessário marcar corretamente as arestas que estão na fronteira saindo da sub-árvore externa e entrando na sub-árvore interna. O caso de duas arestas diminuídas requer um cuidado semelhante. A configuração torna-se ainda mais complicada quando uma aresta aumenta de custo, e outra diminui.

Referências

[1] Ausiello, Italiano, Spaccamela, and Nanni. Incremental algorithms for minimal length paths. *ALGORITHMS: Journal of Algorithms*, 12, 1991.

[2] Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *J. Algorithms*, 12(4):615–638, 1991.

[3] Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. In *SODA '90: Proceedings of the first annual*

ACM-SIAM symposium on Discrete algorithms, pages 12–21, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[4] Michael Barbehenn. A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. *IEEE Trans. Computers*, 47(2):263, 1998.

[5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[6] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1999.

[8] Paolo Giulio Franciosa, Daniele Frigioni, and Roberto Giaccio. Semi-dynamic shortest paths and breadth-first search in digraphs. In *14th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *lncs*, pages 33–46, Lübeck, Germany, 27 February–March 1 1997. Springer.

[9] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[10] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for the single-source shortest path problem. In *FSTTCS*, pages 113–124, 1994.

[11] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic output bounded single source shortest path problem (extended abstract). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 212–221, Atlanta, Georgia, 28–30 January 1996.

[12] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Semidynamic algorithms for maintaining single-source shortest path trees. *Algorithmica*, 22(3):250–274, 1998.

[13] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *J. Algorithms*, 34(2):251–281, 2000.

[14] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights. *J. Algorithms*, 49(1):86–113, 2003.

[15] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 81, Washington, DC, USA, 1999. IEEE Computer Society.

[16] Philip Klein, Satish Rao, Monika Rauch, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, STOC'94 (Montréal, Québec, Canada, May 23-25, 1994)*, pages 27–37, New York, 1994. ACM Press.

[17] Lipton, R. J. and Tarjan, R. E. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–199, 1979.

[18] Paolo Narvaez, Kai-Yeung Siu, and Hong-Yi Tzeng. New dynamic SPT algorithm based on a ball-and-string model. In *INFOCOM (2)*, pages 973–981, 1999.

[19] Paolo Narváez, Kai-Yeung Siu, and Hong-Yi Tzeng. New dynamic algorithms for shortest path tree computation. Technical report, Bell Labs, Lucent Technologies, 5 1998.

[20] Paolo Narváez, Kai-Yeung Siu, and Hong-Yi Tzeng. New dynamic algorithms for shortest path tree computation. *IEEE/ACM Trans. Netw.*, 8(6):734–746, 2000.

[21] G. Ramalingam and Thomas Reps. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms*, 21(2):267–305, 1996.

[22] G. Ramalingam and Thomas Reps. On the computational complexity of dynamic graph problems. *Theor. Comput. Sci.*, 158(1-2):233–277, 1996.

[23] G. Ramalingam and Thomas W. Reps. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms*, 21(2):267–305, 1996.

[24] P. M. Spira and A. Pan. On finding and updating spanning trees and shortest paths. *SIAM Journal on Computing*, 4(3):375–380, sep 1975.