UNIVERSIDADE FEDERAL FLUMINENSE
CENTRO TECNOLÓGICO
MESTRADO EM ENGENHARIA DE PRODUÇÃO

YASMÍN SALAZAR MÉNDEZ

O PROBLEMA DO CAMINHO MAIS CURTO: Algoritmos de Dijkstra, D'Esopo – Pape, *Label Small First* e *Label Small First - Threshold*

Niterói
2008

YASMÍN SALAZAR MÉNDEZ

O PROBLEMA DO CAMINHO MAIS CURTO: Algoritmos de Dijkstra, D'Esopo – Pape, *Label Small First* e *Label Small First Threshold*

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Produção da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Sistemas, Apoio à Decisão e Logística.

Orientador: Prof. Dr. Luis Ernesto Torres Guardia

Niterói
2008

YASMÍN SALAZAR MÉNDEZ

O PROBLEMA DO CAMINHO MAIS CURTO: Algoritmos de Dijkstra, D'Esopo – Pape, *Label Small First e Label Small First Threshold*

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Produção da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Sistemas, Apoio à Decisão e Logística.

Aprovado em 16 de julho de 2008.

BANCA EXAMINADORA

_____

Luis Ernesto Torres Guardia, D.Sc. – Orientador
Universidade Federal Fluminense – UFF

_____

Annibal Parracho Sant'Anna, PhD
Universidade Federal Fluminense – UFF

_____

Mário Jorge Ferreira de Oliveira, PhD
Universidade Federal do Rio de Janeiro – COPPE – UFRJ

Niteroi
2008

A Deus.
Aos meus pais José Armando *(in memorian)* e Maria Olívia.
Aos meus irmãos Diana, José Esteban e David.

AGRADECIMENTOS

A Deus, meu protetor infalível, meu maior sustento e fortaleza especialmente nas horas difíceis.

A minha mãe e aos meus irmãos pelo imenso amor e seu apoio constante.

A minha família que sempre acreditou em mim, especialmente a Sixto, Arturo, Noralma, Chela, Elizabeth, Giovana e Leslie.

A Nayfé, Rosanna e Verônica e a todos os amigos que mesmo de longe estiveram sempre pendentes de mim. Sua amizade foi sem dúvida um grande apoio na realização deste trabalho.

A Luiz Cláudio pela grande ajuda no Brasil e sua amizade.

Ao orientador, Professor Luis Ernesto Torres Guardia, muito obrigada porque além de orientar-me com dedicação na realização deste trabalho, me ofereceu sua amizade, confiança, apoio e sábios conselhos.

Aos Professores Annibal Parracho Sant'Anna e Mário Jorge Ferreira de Oliveira, pela disponibilidade para revisar este trabalho e participar com suas valiosas sugestões.

A Guilherme Lepsch autor do Force 2.0, pelo apoio com a utilização do programa.

Aos professores e funcionários do programa de pós-graduação em Engenharia de Produção da UFF que de alguma forma contribuíram na finalização dos meus estudos.

Ao CNPq que me proporcionou uma bolsa para realizar meus estudos no Brasil.

RESUMO

Neste trabalho aborda-se o problema do caminho mais curto. Para isto apresenta-se uma descrição do problema do caminho mais curto, sua formulação e algumas aplicações, além de alguns conceitos da teoria de grafos, redes e algoritmos, que são considerados fundamentais para estudar este tema. O problema do caminho mais curto é tratado nesta dissertação com ênfase nos algoritmos do caminho mais curto do método de rotulação. Para estabelecer vantagens de uso em termos de rapidez de resposta avaliaram-se com redes de grande porte quatro algoritmos: Dijkstra com heap binário (rotulação permanente) e D'Esopo Pape, Small – Label - First e Small- Label – First - Threshold (correção de rotulação). Os resultados obtidos mostram que o algoritmo que oferece a resposta em menor tempo é o algoritmo de Dijkstra com heap binário, seguido por SLFT, SLF e D'Esopo Pape.

Palavras-chave: Problema do caminho mais curto, redes, grafos, algoritmos, rotulação permanente, correção de rotulação.

RESUMEN

En este trabajo se aborda el problema del camino más corto. Se presenta una descripción general del problema del camino más corto, su formulación y algunas aplicaciones, además de algunos conceptos de la teoría de grafos, redes y algoritmos, que son considerados como fundamentales para el estudio de este tema. El problema del camino más corto es tratado en esta disertación haciendo énfasis en los algoritmos del método de etiquetado. Para establecer ventajas en términos de rapidez de respuesta fueron evaluados con redes de gran tamaño cuatro algoritmos: Dijkstra con heap binario (fijación de etiquetas), D'Esopo Pape, Small – Label - First (SLF) y Small – Label – First - Threshold (SLFT) (corrección de etiquetas). Los resultados obtenidos muestran que el algoritmo que ofrece el menor tiempo de respuesta es Dijkstra con heap binario, seguido por SLFT, SLF e D'Esopo Pape.

Palabras-clave: Problema del camino más corto, redes, grafos, algoritmos, fijación y corrección de etiquetas.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo apresentam-se aspectos gerais da dissertação e do tema de pesquisa com a intenção de guiar ao leitor no acompanhamento de este trabalho, além de motivar nele o desejo de lê-lo através da exposição em forma clara e simples do que significa o problema do caminho mais curto, suas aplicações e as pesquisas realizadas. Apresentam-se também os objetivos e as razões de realizar este trabalho, além de aspectos relacionados com o referencial teórico.

1.1 CONTEXTUALIZAÇÃO DO TEMA

O problema do caminho mais curto "*Shortest path problem*"[1], consiste em encontrar o melhor caminho entre dois pontos chamados nós. Assim, resolver este problema pode significar determinar o caminho entre dois nós com custo mínimo, ou com o menor tempo de viagem ou com a máxima capacidade (Ahuja et al. 1993).

Os problemas do caminho mais curto constituem o maior grupo na área de pesquisa operacional e são considerados como os mais importantes da programação linear (Hillier e Lieberman, 1988).

Achar o caminho mais curto é uma situação que está presente em muitas atividades da vida prática portanto, citar todas as aplicações em detalhe é uma tarefa muito ampla. Em geral as aplicações estão relacionadas com atividades cujo objetivo é a otimização tais como o tráfego de estradas, linhas de transmissão elétrica, conexão de redes, problemas de programação de rota crítica PERT, planejamento de movimentos de um robô e outras mais complexas como no campo de biologia molecular, (Eppstein, 1994).

Ahuja et al. (1993) cita outras aplicações dos algoritmos do caminho mais curto relacionadas com a realização ótima de planos de substituição de equipamento,

[1] Nome com que é conhecido o problema do caminho mais curto na literatura americana.

elaboração de projetos, gestão de fluxo de caixa, transmissão de mensagens em sistemas de comunicação e fluxo de tráfego em cidades com muita congestão.

O problema do caminho mais curto é um tópico atrativo para pesquisadores e profissionais, pois sua resolução permite obter soluções eficientes a importantes e freqüentes problemas práticos. Estas soluções são eficientes no sentido de oferecer de modo confiável a forma mais rápida e econômica de realizar uma atividade determinada. Outro ponto que faz que o tema seja atraente para pesquisadores é porque tem a característica de que, a partir de modelos de redes simples, podem-se elaborar modelos mais complexos que são estudados na área de otimização combinatória. Não obstante que a solução de problemas do caminho mais curto seja relativamente fácil, o desenho e a análise de algoritmos eficientes de solução exigem muito engenho (Festa, 2006; Ahuja et al. 1993).

O interesse que existe por realizar pesquisas relacionadas com o caminho mais curto começou no ano de 1950 assim como o detalharam Dreyfus (1969) e Schrijver (2005), nas resenhas históricas que realizaram sobre este tema.

A partir da década dos 50 o problema foi tratado com a idéia de achar uma rota alternativa no caso de que uma rota estiver bloqueada.

É importante mencionar, que durante algum bom tempo os métodos heurísticos cobraram importância e foram objeto de pesquisas, pois para resolver muitos problemas como por exemplo para determinar a melhor rota para viajar entre dois cidades por estrada, utilizaram-se heurísticas, (Schrijver, 2005).

De 1946 a 1953 foram desenvolvidos métodos de matrizes para determinar o caminho mais curto, tendo como objetivo identificar em um grafo dirigido os nós mais próximos. Estes métodos têm aplicações em redes neurais e em sociologia animal.

Para achar o caminho mais curto para ir de um nó até um outro nó em um grafo dirigido, surgiram dois tipos de métodos, um deles trabalha com redes com pesos arbitrários e o outro trabalha com redes com pesos não negativos.

Dos algoritmos que trabalham com grafos com pesos não negativos, Dijkstra (1959) desenvolveu o primeiro algoritmo mais eficiente de este tipo (Dreyfus, 1969; Ahuja et al. 1990).

Em 1955 a 1957, o problema do caminho mais curto transformou-se em um problema de programação linear.

Em tempos mais atuais autores como Yen (1971), Martins (1984), Glover et al. (1985), Bertsekas (1993), Thorup (1997), Goldfarb e Jin (1999), Ahuja et al. (2000), Festa (2000), Hershberger (2008) contribuíram com novos algoritmos.

1.2 SITUAÇÃO PROBLEMA

As múltiplas atividades em que pode ser aplicado o problema do caminho mais curto com o intuito de realizá-las da melhor maneira possível tem feito com que a busca de algoritmos mais eficientes para solucionar este problema seja uma necessidade. A busca de melhores procedimentos basicamente traduz-se em aperfeiçoar os algoritmos tradicionais que precisam de um aporte grande de memória.

O problema do caminho mais curto tem sido objeto de interesse de pesquisadores e usuários devido a que aparece em muitas e variadas situações como transporte de materiais, ligações telefônicas, e outras atividades que precisem ser feitas de forma rápida e econômica (Festa, 2006).

Na atualidade existe uma grande quantidade de algoritmos de busca do caminho mais curto como fruto do trabalho de inúmeros pesquisadores que perseguiram o desenvolvimento de algoritmos cada vez mais eficientes e que basicamente consistem no aperfeiçoamento dos algoritmos tradicionais.

Para Dreyfus (1969), por causa da intensa busca de bons algoritmos para resolver o problema do caminho mais curto alguns autores podem ter sido omitidos ou alguns algoritmos ficaram sem ser conhecidos. Este mesmo autor faz uma advertência devido à grande quantidade de algoritmos existentes na literatura, nem sempre a eficiência deles foi comprovada ou eles têm erros, daí a importância de fazer uma busca séria e detalhada na hora de escolher algoritmos do caminho mais curto.

"Nos últimos anos tem se dedicado uma boa quantidade de esforço para o estudo do problema do caminho mais curto. Mais de 200 publicações, mas sabe-se pouco sobre sua eficiência relativa", (Pape, 1974, p.1).

"Escolher um algoritmo adequado entre os numerosos algoritmos existentes na literatura, é uma etapa crítica [...]", (Zhan, 1997, p.1).

Desde que no ano 1950 começaram a ser desenvolvidos algoritmos para resolver o problema do caminho mais curto, existem na atualidade mais de dois mil artigos

científicos que tem sido publicados na literatura, mas não tem sido determinado até hoje qual algoritmo supera aos outros, (Festa, 2006).

Na atualidade existem alguns trabalhos de avaliação de algoritmos, mas devido à grade quantidade de algoritmos é muito difícil que estes sejam avaliados em sua totalidade.

Na primeira etapa de este trabalho se realizará uma exaustiva revisão da numerosa literatura existente, com o objetivo de escolher os algoritmos que serão avaliados.

1.3 OBJETIVOS DA PESQUISA

Objetivo geral:

- Implementar e avaliar o desempenho de algoritmos de busca do caminho mais curto de: Dijkstra com *heap* binário, D'Esopo Pape, *Small-Label-First* (SLF) e Small- Label- First-*Threshold* (SLFT).

Objetivos específicos:

- Implementar os algoritmos para encontrar o caminho mais curto de: Dijkstra com *heap* binário, D'Esopo Pape, SLF e SLFT utilizando o compilador de FORTRAN, FORCE 2.0.

- Avaliar no pior caso cada um dos algoritmos propostos.

- Identificar as diferenças entre algoritmos de rotulação permanente e de correção de rotulação.

- Estabelecer diferenças entre os algoritmos programados.

1.4 ASPECTOS METODOLÓGICOS

Para a elaboração de este trabalho realizaram-se pesquisas do tipo bibliográfico e experimental. Na parte bibliográfica os principais autores consultados são: Festa (2006),

Guerreiro et al. (2001), Cormen et al. (1997), Ahuja et al. (1993), Bertsekas (1993), Glover et al. (1985).

A parte experimental se desenvolverá utilizando o método teórico de avaliação de algoritmos no pior caso.

O teste dos algoritmos será realizado em redes de grande porte com estrutura topológica de grade. Serão implementados no compilador de FORTRAN 77, FORCE 2.0 os algoritmos de busca do caminho mais curto de: Dijkstra com *heap* binário, D'Esopo Pape, *Small – Label - First* e *Small – Label – First - Threshold.*

1.5 QUESTÕES DA PESQUISA

As questões a serem respondidas nesta pesquisa são:

- Quais são as diferenças entre os algoritmos de rotulação permanente e de correção de rotulação.

- Quais são as diferenças de cada um dos algoritmos programados.

- Qual é o desempenho de cada um dos algoritmos implementados no pior caso.

1.6 REFERENCIAL TEÓRICO

Os principais conceitos tratados nesta dissertação são: grafos, redes, problema do caminho mais curto, algoritmos de busca do caminho mínimo.

Os principais autores do referencial teórico são: Festa (2006), Guerreiro et al. (2001), Cormen et al. (1997), Ahuja et al. (1993), Bertsekas (1993), Glover et al. (1985).

A parte experimental se desenvolverá utilizando o método teórico de avaliação no pior caso.

1.7 JUSTIFICATIVA E RELEVÂNCIA

Encontrar o caminho mínimo em uma rede é um tema que tem sido objeto de múltiplos estudos na área da pesquisa operacional e, por ser um tópico que tem uma

grande quantidade de informação motiva a estudá-lo para descobrir quais são as melhores alternativas de algoritmos de solução. Assim, e com o interesse de realizar um trabalho que tenha grande importância na comunidade científica da pesquisa operacional, escolheu-se este tema que além de possuir relevância, é muito interessante pelas grandes possibilidades de aplicações em distintos âmbitos da vida real – transporte, eletrônica, produção, projetos, finanças, biologia, robótica, sistemas de informação, turismo, etc. É um tópico muito rico no sentido de fazer estudos posteriores e maiores, como a avaliação de mais algoritmos dos já existentes ou a elaboração de um algoritmo próprio.

Com respeito ao mestrado em engenharia de produção com área de concentração em sistemas, apoio à decisão e logística da Universidade Federal Fluminense, justifica-se a escolha de este tema porque existe uma linha de pesquisa em logística e cadeias de suprimento que realiza além de outros estudos, pesquisas relacionadas com algoritmos de roteamento de veículos em áreas urbanas enfocadas a tratar problemas de roteirizarão de veículos procurando que seus percorridos sejam feitos da forma mais econômica possível.

Na ordem social, expressar o problema do caminho mais curto como a realização de uma atividade em geral com custo mínimo é, sem dúvida, um tema de grande interesse. O exigente ambiente competitivo em que as empresas se desenvolvem lhes obriga a procurar cada vez mais e em forma acelerada melhores maneiras de produzir, significando isto a utilização de menos recursos e a obtenção de maiores benefícios.

Cientes da ampla variedade de aplicações práticas que o problema do caminho mais curto tem em transporte, comunicações, eletrônica, robótica, biologia, projetos, finanças e na área produtiva em geral, constitui um aporte significativo para o ambiente produtivo, especialmente, e para a sociedade em geral realizar um estudo que como resultado final seja a resposta de quais métodos permitiriam realizar atividades utilizando uma quantidade mínima de recursos.

1.8 DELIMITAÇÃO DA PESQUISA

Nesta pesquisa se abordarão conceitos da teoria de grafos, redes, problema do caminho mais curto e os algoritmos para resolver este problema de: Dijkstra com *heap* binário, D'Esopo Pape, *Small-Label-Fist* e *Small-Label-First-Threshold.*

1.9 ESTRUTURAÇÃO DA PESQUISA

Esta dissertação encontra-se estruturada em cinco capítulos e resumidamente cada um deles contém o seguinte:

- O capítulo 1 é o introdutório e contém a contextualização do tema, a situação problema, os objetivos, metodologia, justificativa, delimitação e estrutura do trabalho.

- No capítulo 2 apresentam-se os principais conceitos básicos e terminologia da teoria de grafos, redes e algoritmos.

- No capítulo 3 apresenta-se o problema do caminho mais curto.

- No capítulo 4 apresentam-se os algoritmos a serem avaliados e os testes realizados.

- O capítulo 5 contém as conclusões da dissertação assim como recomendações de trabalhos futuros do problema do caminho mais curto.

- A seguir apresenta-se a bibliografia citada e consultada para a realização de esta pesquisa.

- Anexos.

2 GRAFOS, REDES E ALGORITMOS

Um grafo é uma estrutura matemática que permite representar visualmente um conjunto de dados (nós) e a relação existente entre eles (arcos). Uma rede é considerada um grafo com um fluxo de algum tipo em seus arcos. Este fluxo pode ser custo, capacidade, distância, tempo, nível de importância, etc.

Estruturas de diversos campos podem ser representadas através de grafos. Na matemática, eletrônica, confecção, engenharia civil, transporte, logística, informática, biologia, química, eletricidade, comunicações, psicologia, sociologia e outras áreas se usam grafos para fazer representações.

Por exemplo, um mapa rodoviário, pode ser representado em uma rede, cada cidade é um nó, as estradas que as comunicam são os arcos e a distância existente entre elas é o fluxo de cada arco.

As atividades a serem desenvolvidas em um projeto, também podem ser representadas em uma rede. Assim cada atividade é um nó e o tempo de execução é o fluxo de cada arco.

A hierarquia de uma empresa igualmente pode ser representada em uma rede. Neste caso cada funcionário é um nó e para o fluxo pode-se atribuir um peso dependendo da função que o funcionário desempenhe na organização.

Em uma estação de combustível, as estações de bombeamento são os nós, os tubos os arcos e o fluido por eles transportado é o fluxo.

Em uma fábrica, os centros de trabalho são os nós, as rotas de manuseio de materiais os arcos e os trabalhos realizados em cada um deles é o fluxo.

Os exemplos acima citados constituem uma minúscula mostra das inúmeras representações de estruturas que podem ser feitas usando grafos e redes. A grande variedade de aplicações deles e em distintas áreas do conhecimento contribuiu a que estes dois conceitos tenham sido difundidos amplamente e que na atualidade falar de

grafos e redes não seja um tema especifico somente de pessoas relacionadas com as matemáticas ou a informática. No obstante, esta difusão também fez com que na literatura se encontre uma ampla diversidade de terminologia, de ali a importância de além de definir os conceitos ligados à teoria de grafos e redes adotar uma terminologia para ser usada durante todo o trabalho.

2.1 CONCEITOS FUNDAMENTAIS DE GRAFOS

Um grafo G = (N, A) é uma estrutura constituída por dois conjuntos finitos, N que é um conjunto de nós e A que é um conjunto de arcos (ou ligações ou arestas ou ramos), com A $\subseteq$ N × N. O número de nós e arcos do grafo G, N e A respectivamente são tais que $1 \leq N < \infty$ e $0 \leq A < \infty$. O número de nós estará representado por n, e o número de arcos por m.

Os grafos são de três tipos:

- Orientado (direcionado) é aquele em que todos os arcos estão orientados
- Não orientado (não direcionado) é aquele em que todos os arcos estão no orientados
- Misto, alguns arcos são orientados e outros são não orientados.

Cada arco está representado por um par (*i, j*), com $i \neq j$ e *i, j* $\in$ N; *i* é o nó inicial do arco *cauda,* e *j* é o nó final *cabeça*. Diz-se que o arco (*i, j*) sai do nó i e chega ao nó j.

Em um arco (i,j) $\in$ A, o nó j é sucessor de i, e i é predecessor de j. i e j dizem-se adjacentes ou vizinhos. O arco (i, j) é incidente nos nós i e j. Dois arcos dizem-se adjacentes se existe entre eles pelo menos um nó comum. Um nó é de ordem *k se tiver k arcos adjacentes a ele.*

O conjunto de arcos que saem de um nó tal que $A(i) = \{(i, j) \in A : j \in N\}$ é chamado de lista de adjacência de arcos. A lista de adjacência de nós A(i), é o conjunto de nós adjacentes ao nó, assim $A(i) = \{(i, j) \in A : j \in A\}$.

Chama-se de grau exterior de i $d^+(i)$ ao número de arcos que saem de este nó, |A(i)|.

Grau interior de j $d^-(j)$ é o número de arcos que chegam a este nó.

O grau de um nó i $\in$ N é a soma dos graus exterior e interior, $d = d^+(i) + d^-(i)$. Na Figura 1 no nó 3, $d^+(3) = 1$, $d^-(3) = 2$ e d = 3.

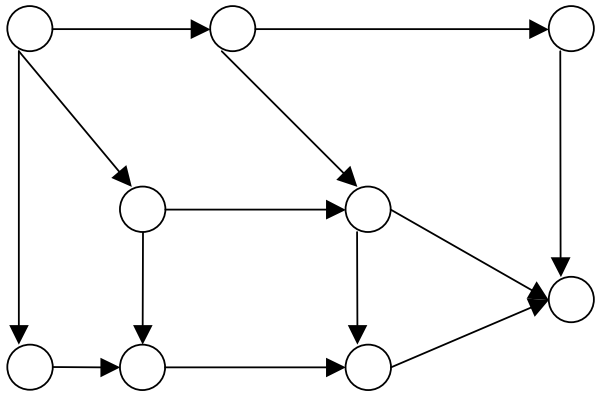

Figura 1 Grafo orientado

A soma de todos os graus interiores de todos os nós é igual à soma de todos os graus exteriores de todos os nós, e ambos os valores são o número de arcos m de uma rede.

Um nó i ∈ N tal que seu grau interior $d^{-}(j) = 0$ conhece-se como nó fonte ou origem e se o grau exterior $d^{+}(i) = 0$ conhece-se como nó isolado.

Se G = (N, A) é um grafo não dirigido com n = |N| e m = |A| então $\sum_{x \in V} d(x) = 2m$.

A lista de adjacência de arcos A(i) de um nó i é o conjunto de arcos que saem de um nó, tal que $A(i) = \{(i, j) \in A : j \in N \}$.

A lista de adjacência de nós A(i) é o conjunto de nós adjacentes ao nó, tal que $A(i) = \{(j \in N : (i, j) \in A \}$. Serão omitidos os termos nó e arcos e simplesmente se falará de lista de adjacência.

|A(i)| é igual ao grau exterior do nó i. A soma de todos os graus exteriores é igual a m, então $\sum_{i \in N} | A(i) | = m$.

A densidade de um grafo é a razão entre a quantidade de arcos do grafo e a quantidade de arcos do grafo completo com a mesma quantidade de nós.

Um arco formado por um par de nós idênticos recebe o nome de laço.

Um arco é múltiplo ou paralelo quando existe mais de um arco entre o mesmo par de nós.

Um grafo que permite a existência de arcos múltiplos diz-se de multigrafo. Ver Figura 2.

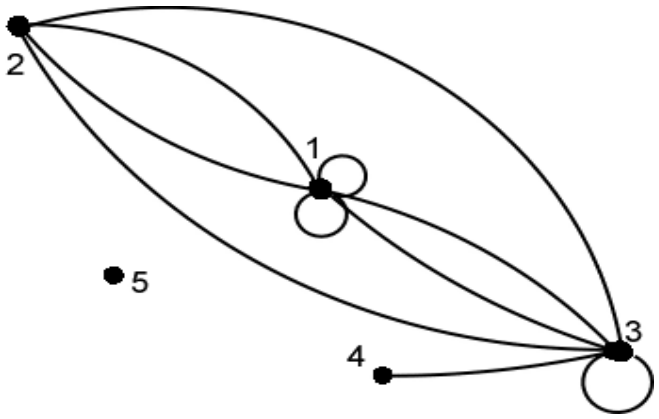

Figura 2 Multigrafo

Quando um grafo não tem arcos que se repetem diz-se que G é um grafo simples.

O grafo G' = (N', A') é um subgrafo de G = (N, A) se N' $\subseteq$ N e A' $\subseteq$ A. G' = (N', A').

Cadeia de um grafo orientado G = (N, A) é um subgrafo de G que tem a seqüência de nós e arcos $i_1 - a_1 - i_2 - a_2 - \cdots - i_{r-1} - a_{r-1} - i_r$ que satisfazem a propriedade que para todo $1 \le k \le r-1$, ou $a_k = (i_k, i_{k+1}) \in A$ ou $a_k = (i_{k+1}, i_k) \in A$.

Quando a cadeia é orientada é uma cadeia orientada. Para dos nós consecutivos que estiverem na cadeia i_k e $i_{k+1}, (i_k, i_{k+1}) \in A$.

Caminho de i $\in$ N até j em G = (N, A) é uma sucessão $C = \{a_1, a_2, \ldots, a_k)$ de arcos adjacentes tal que:

- O nó inicial de a_1 é i.
- O nó final de a_k é j.
- O nó final de a_i é igual ao nó inicial de $a_{i+1}, (i = 1,2,\ldots,k-1)$

"k" é a longitude do caminho.

Um caminho é simples se os arcos não se repetem. Um caminho é elementar se cada nó do grafo é incidente no máximo a dois arcos do caminho. Se um caminho é elementar então o caminho é simples. Um caminho elementar que contém todos os nós de G conhece-se como caminho Hamiltoneano.

Um ciclo é uma cadeia de i para i. Um ciclo simples é um caminho de i para i. Um circuito (ciclo orientado) é um ciclo formado por arcos orientados. Um grafo orientado sem ciclos diz-se acíclico. Uma rede diz-se acíclica se não contém um ciclo orientado.

Em alguns casos os pesos dos arcos são negativos. Se um grafo G = (N, A) não contém ciclos de comprimento negativo alcançáveis desde o nó origem s até todos os n nós $\in N$, o comprimento do caminho mínimo está bem definido inclusive no caso de que este seja um valor negativo. Se existir um ciclo de comprimento negativo accessível desde o nó raiz r, o comprimento do caminho mínimo não está bem definido. Nenhum caminho da raiz s até um outro nó no ciclo pode ser o caminho mais curto, pois o caminho de menor comprimento.

Se há um ciclo de comprimento negativo em um caminho do nó origem s até um outro nó, este comprimento se define como $-\infty$.

Para ilustrar melhor a idéia de ciclos com comprimentos negativos, ver a Figura 3.

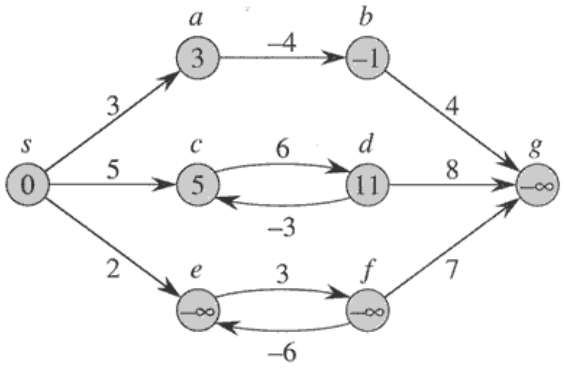

Figura 3 Grafo com ciclo negativo

Fonte: Cormen et al. (1997), p. 546

Na Figura 3, o comprimento de s até a é 3, $c_{sa} = 3$. O comprimento do caminho de s a b é $c_{sb} = c_{sa} + c_{sb} = 3 + (-4) = -1$.

O comprimento do ciclo c – d – c é 6 + (-3) = 3 > 0; o comprimento do caminho mais curto de s a c é 5, e de s a d é 11.

No ciclo e-f-e o comprimento é 3 + (-6) = -3 < 0. Para atravessar o ciclo de comprimento negativo e – f – e desde s, $c_{se} = -\infty$ e também $c_{sf} = -\infty$. Como g é accessível desde f, e para ir de s a g existem comprimentos com valor negativo, então $c_{sg} = -\infty$.

Os nós h, i, j também formam um ciclo de peso negativo. Não existe um caminho alcançável desde s até estes pontos e $c_{sh} = c_{si} = c_{sj} = \infty$.

Dois nós i e j são conectados (ligados) se existir um cadeia quaisquer entre os dois nós. Um grafo é conectado se cada par de nós são conectados, se não o grafo é desconectado.

Um grafo tem conexidade forte se contém pelo menos um caminho orientado de cada nó até os outros nós.

Corte é uma partição do conjunto de nós N em duas partes, S e $\bar{S} = N - S$. Cada corte define um conjunto de arcos que têm um nó em S e o outro nó em $\bar{S}$.

Árvore é um grafo conectado que não contém nenhum ciclo. Uma árvore com n nós contém exatamente n-1 arcos. Uma árvore tem pelo menos dois nós com grau 1. Cada par de nós da árvore estão ligados por um caminho único. Ver Figura 4.

Uma árvore geradora (abrangente ou total) de um grafo G é uma árvore formada por arcos de G e que contenha todos os vértices de G.

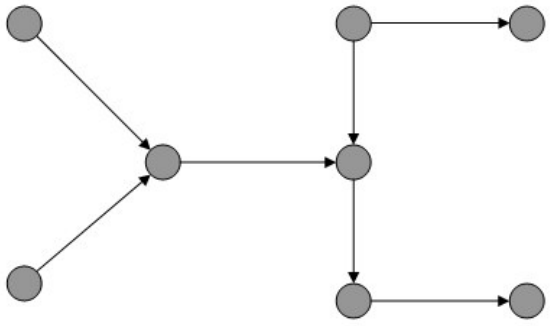

Figura 4 Exemplo de árvore

Bosque ou floresta é todo grafo de um ciclo, ou seja, todo grafo cuja componente conexa seja árvore. Diz-se que uma floresta é a coleção de árvores. Subárvore é um subgrafo conectado de uma árvore.

Dado um grafo orientado G = (N, A) diz-se que um nó r ∈ N, é uma raiz se é possível construir um caminho de r a todo nó do grafo.

Diz-se que um grafo orientado G = (N, A) é uma arborescência se:

- G contém uma raiz.
- G é uma árvore.

Uma subárvore geradora ou árvore geradora (*spanning tree*) de um grafo *G* é qualquer subárvore de *G* que contenha todos os vértices de *G*. Cada árvore geradora e um grafo conectado com n nós tem n – 1 arcos. Os arcos que pertencem à árvore geradora dizem-se arcos da árvore. Somente os grafos conexos têm árvores geradoras. Reciprocamente, todo grafo conexo tem uma árvore geradora. (Em geral, um grafo conexo tem muitas árvores geradoras diferentes.).

Cortes fundamentais, seja T uma árvore geradora do grafo G. A supressão de qualquer um arco da árvore geradora cria um grafo desconectado que contém dois subárvores T_1 e T_2. Arcos com nós que pertencem a diferentes subárvores constituem um corte.

2.2 CONCEITOS FUNDAMENTAIS DE REDES

Uma rede é um grafo cujos nós e/ou arcos têm associados valores numéricos "pesos" (custos, capacidades e/ou fornecimento e demanda).

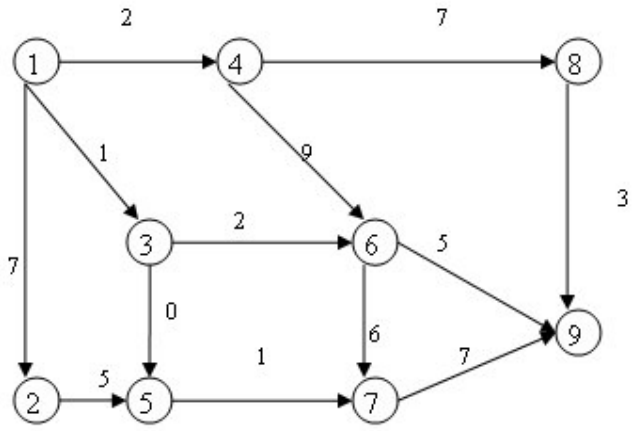

Figura 5 Exemplo de rede

Uma rede pode ser representada por G = (N, A, C), em que (N, A) é um grafo e C é o conjunto dos valores numéricos associados aos arcos (*comprimentos* dos arcos); ou seja, ao arco (i, j) está associado o valor c_{ij} (o peso do arco (i, j) é de c_{ij}). De uma maneira geral, os conceitos utilizados para grafos são extensíveis a redes.

Se um grafo orientado é uma rede, a orientação dos arcos é a direção viável do fluxo no arco. Uma rede não precisa ser orientada porque pode ser viável haver um fluxo em ambas as direções no arco.

A capacidade de fluxo de um arco em uma direção específica é o limite superior para a magnitude viável da taxa de fluxo no arco naquela direção. Esta capacidade pode ser qualquer quantidade não negativa inclusive infinito.

Um arco é orientado se a capacidade de fluxo for zero em uma direção. Um nó também pode ter uma capacidade de fluxo limitada.

Considere-se um caminho p de i para j na rede G. O peso do caminho p, c(p), é a soma dos pesos dos arcos que pertencem a aquele caminho; ou seja,

$$c(p) = \sum_{(i,j)\in p} c_{ij}$$

O conjunto de todos os caminhos de i para j em uma rede G identifica-se por P. O caminho de menor peso é conhecido como o *caminho mais curto*.

2.3 ESTRUTURAS DE DADOS

Os algoritmos relacionados com redes requerem a manipulação de dados que particularmente representam informação de arcos, nós e árvores. As formas de representação de estes elementos têm sido objeto de pesquisas que procuram armazenar e manipular os dados da melhor forma possível, pois a escolha de estas estruturas pode modificar a eficiência de um algoritmo. A seguir apresenta-se algumas de estas estruturas.

2.3.1 REPRESENTAÇÃO DE REDES

2.3.1.1 Matriz de incidência de nós - arcos

Seja G = (N, A, C) uma rede direcionada com n nós e m arcos. A matriz de representação de incidência de nós – arcos, ou simplesmente matriz de incidência, é

uma matriz $n \times m$ onde as filas correspondem aos nós e as colunas correspondem aos arcos da rede.

Na coluna correspondente ao arco (i, j) no caso das redes dirigidas pode ter dois valores não nulos +1 (que corresponde a cauda do arco) e -1 (correspondente à cabeça). Assim, na coluna do arco (i, j), deve colocar-se +1 na fila i e -1 na fila j. Os outros valores correspondentes do arco (i, j) são zero.

$$A_{is} = 1$$
$$A_{js} = -1$$
$$A_{ks} = 0 \text{ if } k \neq i, k \neq j$$

O número de +1 de uma fila corresponde ao grau exterior do nó, e o número de -1 de uma fila corresponde ao grado interior do nó.

Nas redes não orientadas, colocam-se em todas as entradas da matriz +1, pois neste caso de redes um mesmo nó é considerado cauda e cabeça de um arco.

Pela estrutura da matriz de incidência, somente 2m de seus n x m entradas são zero. Os valores não nulos de esta matriz são +1 e -1, e cada coluna tem exatamente um +1 e um -1.

Por causa de esta estrutura, a matriz de incidência torna-se ineficiente no tempo de pesquisa e no espaço de memória necessário para o armazenamento dos dados, pois considerando que da quantidade de elementos reservados para a rede n x m esta tem apenas m elementos para serem armazenados (ou 2m se for uma rede não dirigida). Em ambos os casos a matriz de incidência apenas tem 2m valores diferentes de zero (dos nm existentes): cada coluna tem exatamente um (+1) e um (−1) (rede orientada) (ou dois (+1)) rede não orientada.

Além disso, a quantidade de (+1) em uma linha é igual à quantidade de arcos que saem do nó correspondente, qualquer que seja o tipo de rede. Da mesma forma, em redes dirigidas, a quantidade de (−1) em uma linha é igual à quantidade de arcos que chegam ao nó. As Figuras 6 e 7 contêm um grafo e sua matriz de incidência de nós-arcos respectivamente.

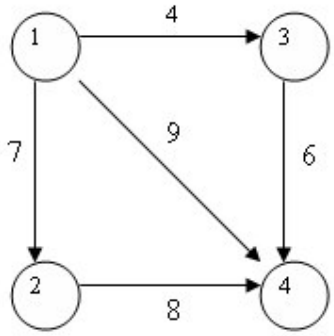

Figura 6 Exemplo de rede

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{bmatrix}$$

Figura 7 Matriz de incidência de nós - arcos

2.3.1.2 Matriz de adjacência de nós - arcos

A matriz de adjacência nó-arcos, ou simplesmente matriz de adjacência, armazena a rede em uma matriz $n \times n$, $H = \{h_{ij}\}$.

A matriz tem uma fila e uma coluna correspondentes a cada nó. Cada elemento hij da matriz assume o valor 1 se (i, j) $\in$ A ou 0 se (i, j) $\notin$ A.

Nas redes não dirigidas, pode considerar-se que cada arco (i, j) corresponde a dois arcos dirigidos: (i, j) e (j, i). Neste caso a matriz de adjacência é simétrica.

Adicionalmente para cada parâmetro associado aos arcos da rede (custo, comprimento, capacidade, etc.) terá que existir uma matriz C de ordem n e para cada parâmetro associado aos nós terá que existir um vetor U de n elementos.

A matriz de adjacência tem n^2 elementos, e de estes m valores são diferentes de zero.

Em problemas em que a densidade da rede é muito baixa (matriz com grande quantidade de elementos nulos), como acontece na maioria dos problemas de circulação, este tipo de estrutura é bastante ineficiente no tempo de pesquisa e no espaço

de memória necessário para o armazenamento dos dados. Isto porque a matriz de adjacência permite armazenar n^2 arcos e a rede apenas tem m (ou 2m caso seja uma rede dirigida).

A simplicidade de esta representação permite a sua utilização para implementar com relativa facilidade algoritmos de redes, pois permitem:

- Determinar os parâmetros associados a qualquer arco (i, j), tomando simplesmente o elemento (i, j) das respectivas matrizes.

- Obter facilmente os arcos que saem do nó i, examinando a linha i: se o j-ésimo elemento dessa linha tem o valor um (1) então (i, j) é um arco da rede.

- Obter os arcos que chegam ao nó j, examinando a coluna j: se o i-ésimo elemento dessa coluna tem o valor um (1) então (i, j) é um arco da rede.

A Figura 8 contém a matriz de adjacência de nós – arcos da Figura 6.

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 8 Matriz de adjacência de nós - arcos

2.3.1.3 Listas de adjacência

Anteriormente definiu-se a lista de adjacência do arco i, A(i) como o conjunto de arcos que saem de este nó.

Similarmente define-se a lista de adjacência do nó i como o conjunto de nós j tales que $(i,j) \in A$.

A lista de adjacência do nó i, A(i), será uma lista ligada com |A(i)| células, em que cada célula corresponde a um arco (i, j) da rede. A célula do arco (i, j) terá uma quantidade de campos igual à quantidade de informação que é preciso guardar. Assim,

um campo guardará o nó j e um outro conjunto guardará dados referentes ao custo, capacidade, etc., do arco e um outro campo de ligação guardará um ponteiro para a próxima célula de esta lista ligada que é o correspondente a um outro arco que sai do nó i.

A última célula da lista ligada por convenção terá o valor zero.

Dado que é necessário guardar e aceder a n listas ligadas (uma para cada nó), também é necessário uma tabela de ponteiros que apontem para a primeira célula de cada lista ligada. Para tal, define-se uma tabela de dimensão n, designada por *primeiro ("first")*, cujo elemento primeiro (i) guarda um ponteiro para a primeira célula da lista de adjacência do nó i, A(i); se a lista de adjacência do nó i é vazia, então primeiro (i) = 0.

Em redes não dirigidas a informação de cada arco é guardada em duas células distintas, pois cada arco pertence a duas listas de adjacência: o arco (i, j) pertence às listas de adjacência do nó i e do nó j, (i, j), pois (i, j) = (j, i).

2.3.1.4 Vetores simulando listas múltiplas

A *"forward star form"* é uma representação muito semelhante à representação com listas de adjacência de arcos no sentido que também guarda a lista de adjacência de cada nó. Não obstante, em vez de listas ligadas são utilizados vetores para guardar a informação associada aos arcos da rede.

Nas representações que utilizam vetores para simular listas múltiplas a numeração dos arcos segue uma determinada ordem: primeiro são numerados os arcos que saem do nó 1, depois os arcos que saem do nó 2, etc.. Os arcos que saem do mesmo nó, podem ser numerados em uma ordem arbitrária.

Guarda-se cauda, cabeça, custos e capacidades em quatro vetores cauda, cabeça, custos e capacidades. Se o arco (i, j) é o arco número 20, seus elementos serão guardados nos vetores de posição cauda (20), cabeça (20), custo (20), e capacidade (20). Mantendo o ponteiro no nó i, ponteiro (i), este indica o menor número de arco que sai do nó i.

Se o nó i, não tem arcos que saem dele, ponteiro (i) = ponteiro (i+1).

Assim, determinam-se todos os arcos que saem de qualquer nó. Os vetores utilizados são os seguintes:

- Point, de dimensão n+1, que é definido assim:

Point (1) = 1; Point (i+1) = Point (i) + quantidade de arcos que saem de i ($1 \leq i \leq n$); (Point (n+1) = 1 + m).

- Suc (a) = nó cabeça do arco a. Se Point (i) = Point (i+1) então não saem arcos do nó i.

A *"reverse star form"* permite determinar de forma eficiente todos os arcos que chegam a qualquer nó. Os vetores utilizados são os seguintes:

- RPoint, de dimensão n+1, que se define da seguinte forma :

RPoint (1) = 1

RPoint (i+1) = RPoint (i) + quantidade de arcos que chegam a i ($1 \leq i \leq n$)

(RPoint (n+1) = 1 + m)

- Ant, em que Ant (a) = nó cauda do arco a. Se RPoint (i) = RPoint (i+1) então não chegam arcos ao nó i.

Em redes não dirigidas, tal como acontece para a representação com listas de adjacência de arcos a informação associada a cada arco é armazenada duas vezes.

2.3.2 REPRESENTAÇÃO E ESTRATÉGIAS DE SELEÇÃO DE NÓS A EXPLORAR

2.3.2.1 Array

Array é a mais simples estrutura de dados para armazenar um conjunto ordenado de dados. Esta representação usa um vector de tamanho n, chamado lista, a posição i-ésima do array contém o i-ésimo elemento do conjunto de dados denota-se como list[i].

Em determinados casos resulta conveniente fazer algumas operações usando este tipo de estruturas, como por exemplo, ao determinar o k-ésimo elemento de uma lista. Para isto é preciso estabelecer se uma lista contém um elemento dado α, se cria um índice i que varia de 1 até o último elemento da lista, até que list (i) = α. Esta operação

requer um tempo proporcional ao número de elementos na lista que deverão ser examinados. No caso de inserir um elemento ao fim da lista, o incremento se faz ao fim da lista e é armazenado o novo elemento.

No entanto, outras operações não podem ser realizadas com a facilidade das operações descritas no parágrafo anterior. Por exemplo, supor que se deseja eliminar o k-ésimo elemento de uma lista, com k < último elemento da lista. Eliminado o k-ésimo elemento a posição dos elementos armazenados em k + 1, k + 2,..., último muda, portanto cada elemento tem que ser transferido uma posição. No pior caso, eliminar um elemento e revisar a lista requer tantas operações como elementos contenha a lista. Inserir um elemento na lista requer também um trabalho similar. Neste caso esta representação não é conveniente.

Os arrays têm a vantagem de que os seus elementos são acessíveis de forma rápida, mas têm uma notável limitação: são de tamanho fixo, não podem ser incrementados ou diminuídos sem implicar complexos processos de cópia.

Estas estruturas de dados são ajeitadas nas situações em que o acesso aos dados seja realizado de forma aleatória e impredizível. Porém, se os elementos podem estar ordenados e vai-se empregar um acesso seqüencial, seria mais ajeitada uma lista.

A Figura 9 mostra um array do conjunto ordenado {10, 5, 8, 9, 7}. Neste exemplo último = 5, indicando que a lista não contém elementos nas posições 6, 7, ..., n.

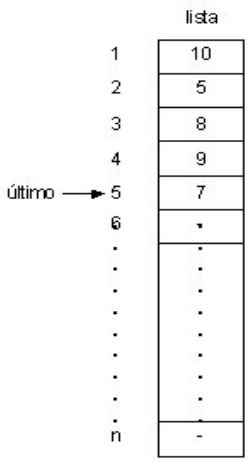

Figura 9 Array

2.3.2.2 Listas encadeadas

Uma lista encadeada é uma seqüência de informação armazenadas em algum lugar da memória, sendo que as mesmas estão ligadas entre si por um endereço

(pointer). As listas encadeadas se diferençam dos arrays porque armazenam os elementos de um conjunto ordenado em forma seqüencial.

Uma lista encadeada é um tipo de dado auto-referenciado, pois contém um ponteiro ou link para outro dado do mesmo tipo. Permitem a inserção ou remoção de qualquer nó em qualquer posição da lista a qualquer tempo, porém não permitem o acesso aleatório aos dados.

Existem muitos tipos de listas encadeadas, mas aqui serão apresentadas as listas simplesmente encadeadas e as listas duplamente encadeadas.

As listas simplesmente encadeadas (s*ingly-linked list*) são o tipo mais simples de lista encadeada, pois tem um link apenas por nó. O link aponta para o próximo nó na lista ou para um valor nulo indicando que este é o fim da lista, se o primeiro nó apontar para um valor nulo a lista encadeada está vazia.

Lista duplamente encadeada *(doubly-linked list)* cada nó tem dois links: um aponta para o nó anterior ou para um valor nulo indicando fila vazia, ou para o próximo nó ou para um valor nulo indicando o nó final da lista; isto com exceção do primeiro e do último nodo, cujos endereços posterior e antecessor são respectivamente NIL.

2.3.2.3 Filas e pilhas

Os dois tipos mais conhecidos de listas são filas e pilhas. Uma pilha é uma lista ordenada na qual as inserções e eliminações são feitas somente no final, chamado topo. Uma pilha pode ser armazenada como um array ou como uma lista encadeada.

Uma pilha é representada como um array n-dimensional, lista que armazena o elemento de um conjunto, e um escalar topo que denota o índice da última entrada na lista array. Se a pilha está vazia, topo = 0. Por exemplo, supondo que se começa com uma lista vazia e são inseridos os elementos na ordem 5, 4, 8, 7, 10; então a pilha tem a aparência da Figura 10 e topo = 5.

As operações mais freqüentes com pilhas são inserções e eliminações. Para inserir um novo elemento i na pilha, incrementa-se um elemento o topo e list(top) = i. Para eliminar o elemento topo i da pilha, o conjunto i = list(topo) é diminuído em um elemento. Cada uma de estas etapas requer O(1). Observar que uma pilha sempre remove o último elemento inserido, conseqüentemente os elementos são examinados com a regra LIFO (*last-in-first-out*).

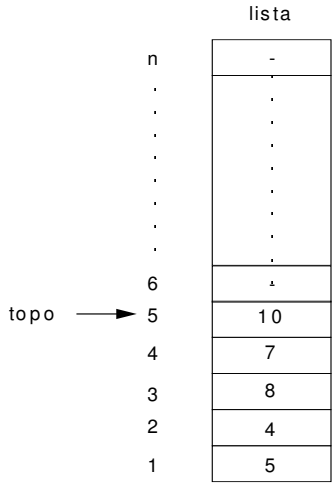

Figura 10 Exemplo de pilha

As filas são estruturas baseadas no princípio FIFO *(first-in- first- out*), em que os elementos que foram inseridos no início são os primeiros a serem removidos, ou seja, Em uma fila as inserções são na cauda e as eliminações na cabeça.

Outro tipo de listas são dqueue e 2queue. Nas primeiras dqueue, as adições podem ser feitas na cabeça ou no final, enquanto que as eliminações podem ser feitas só na cabeça. A lista 2queue se diferença da anterior em que os elementos se acrescentam ao topo da cola Q'.

A Tabela 1 contém a complexidade de várias operações para arrays e para listas encadeadas.

Operação	**Array**	**Lista simplesmente encadeada**	**Lista duplamente encadeada**
Inserir um elemento no final	O(1)	O(1)	O(1)
Inserir um elemento em uma posição arbitrária	O(n)	O(1)	O(1)
Eliminar um elemento do final	O(1)	O(1)	O(1)
Eliminar um elemento de uma posição arbitrária	O(n)	O(n)	O(1)
Determinar o k-ésimo elemento da lista	O(1)	O(k)	O(k)
Determinar os elementos de uma lista	O(n)	O(n)	O(n)

Tabela 1. Complexidade de operações de arrays e listas encadeadas

2.3.2.4 Filas de prioridade

Um tipo de filas de prioridade é o *bucket*. Se f é uma função monótona inteira que relaciona cada um dos rótulos d com os inteiros de 1 a K, define-se o bucket k como o conjunto de todos os elementos v que pertencem a uma fila tais que f(dv) = k.

Outro tipo de filas de prioridade é o *heap,* que é uma estrutura de dados que permite armazenar e manipular eficientemente um conjunto H de elementos, donde cada $i \in H$ tem um valor real associado chamado key(i). As seguintes operações são utilizadas com o heap:

- create-heap (H): Cria um heap vazio.
- find-mini (i, H):encontrar e retornar um objeto i com o mínimo key.
- insert (i, H): Insere um novo objeto i com um key predefinido.
- decrease-key (valor, i, H): Reduz o key de um objeto i de seu valor atual, e o substitui por um valor menor.
- delete-min(i, H): Remove m objeto i com o mínimo key.

O heap binário é um caso especial de d-heap com d = 2, é o de mais interesse neste trabalho porque será avaliado o algoritmo de Dijkstra com heap binário.

Em um d-heap, os nós são armazenados como uma árvore com raiz cujos arcos representam as relações de predecessor e sucessor (pais – filhos). A árvore usando índices predecessores é um conjunto de sucessores como segue:

Pred (i) : o predecessor do nó i no d-heap. O nó raiz não tem predecessor, ou seja, seu predecessor é igual a zero.

Suc (i): é o conjunto de sucessores ou filhos do nó i no d-heap.

Em um d-heap define-se a profundidade de um nó i como o número de arcos que pertencem ao único caminho do nó i até a raiz. Por exemplo, no d-heap da Figura 11, o nó 5 tem uma profundidade de 0 e os nós 9, 8 e 15 têm uma profundidade de 1.

Cada nó pertencente ao d-heap tem no máximo d sucessores, e assume-se que estes estão ordenados de esquerda a direita. Os sucessores de um nó são conhecidos como seus irmãos.

Esta estrutura permite armazenar os dados em um array, e os nós são armazenados considerando sua profundidade em ordem crescente e de esquerda a direita. No exemplo dheap = {5, 9, 8, 15, 10, 12, 9, 6, 7, 38, 67, 3, 6}, além posição (4) = 15 e posição (8) = 6. Adicionalmente o parâmetro último especifica o número de nós guardados no array d-heap. Neste exemplo último = 6.

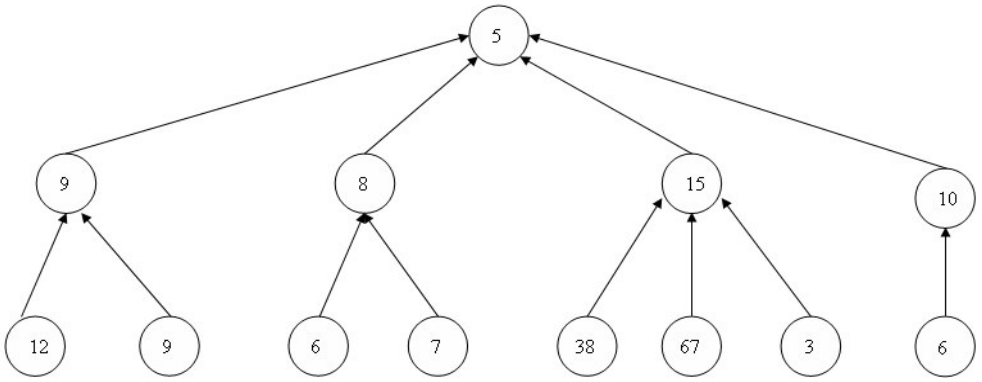

Figura 11 Exemplo de d-heap para d = 2

A operação find-min(i, H) requer um tempo O(1), pois a raiz de um nó é o nó com mínimo key e ocupa a primeira posição do array d-heap.

Insert(i, H) e decrease-key(i, value, H) requerem um tempo $O(\log_d n)$.

Delete-min(i,H) requer um tempo $O(d \log_d n)$.

O heap radix de um nível é uma das variantes que existem dos buckets. Um heap radix de um nível é uma coleção de B = [log (C + 1)] + 2 buckets numerados de a até B, onde se procura que os buckets iniciais contenham os rótulos menores e com os valores menores, enquanto que os outros sejam de um valor maior e também contenham rótulos maiores.

Um heap-Fibonacci é um conjunto de árvores com raiz, cada nó i da árvore representa um elemento i e cada arco (i, j) representa um predecessor- sucessor, assim o nó j é o predecessor do nó i. Estes heaps podem realizar as operações insert, find-min e decrease-key em um tempo O(1) e as operações delete-min, delete e decrease-key em um tempo O(log n).

2.4 ALGORITMOS

Um algoritmo para um problema é uma seqüência finita de operações matemáticas e lógicas que permitem resolvê-lo.

Para solucionar um mesmo problema podem existir vários algoritmos com a diferença de que para dar uma resposta podem desenvolver um número diferente de operações, isto determina a eficiência de um algoritmo.

A teoria da complexidade de algoritmos estuda o tempo e a memória que um algoritmo necessita dependendo do tamanho dos dados de entrada, com o objeto de comparar diferentes algoritmos que resolvem um mesmo problema.

A complexidade de um algoritmo consiste na quantidade de trabalho necessário para a sua execução. Esta quantidade de trabalho vem definida pelas operações que são executadas no desenvolvimento de um algoritmo e variam de acordo com cada um deles e em função do volume de dados.

Existem dois tipos de complexidade:

Espacial que é o espaço de memória exigido em função do tamanho (n) da entrada. Representada por S(n).

Temporal que é o tempo real necessário para a execução de um algoritmo. Também é o número de instruções necessárias à execução. T(n) representa o tempo de execução em função do tamanho (n) da entrada.

O estudo da complexidade de um algoritmo é feito desde três perspectivas:

- Caso pior se testa o algoritmo com "dados perversos", ou seja, com dados extremos com o objetivo de provar que o algoritmo funciona em todos os casos. Está representado por O( ). Assim para um algoritmo g(x) cuja complexidade no pior caso é n, a complexidade de g(x) = O(n). A maioria dos estudos utilizam este método para mostrar a complexidade de um algoritmo.

- Caso melhor se testa o algoritmo com "dados benéficos", tenta-se que estes sejam dados que produzam os melhores resultados. Representado por Ω( ).

- Caso médio usam-se "dados aleatórios" para garantir que as experiências testem o algoritmo e não apenas os dados específicos. Precisa-se de análise estatística e como tal muitos testes. É representado por $\theta(\ )$. O limite superior de um algoritmo (*upper bound*) é estabelecido pelo melhor algoritmo do momento. Assim o limite superior pode ser melhorado por um algoritmo mais veloz. Por exemplo, se para resolver um problema determinado o melhor algoritmo tem uma complexidade $O(n^2)$, este é o seu limite superior e a complexidade de outro algoritmo não poderá ser maior do que este. O limite superior de um algoritmo pode mudar ao ser desenvolvido um algoritmo melhor.

Para resolver um problema (sem considerar o algoritmo a ser usado) requere-se de realizar pelo menos um determinado número de operações. O limite inferior não depende do algoritmo, mas do problema que será resolvido, e denota-se por Ω. Se um algoritmo tem uma complexidade que é igual ao limite inferior do problema então o algoritmo é ótimo.

Seja $P(\ n\ )$ um polinômio em n. Um algoritmo diz-se limitado polinomialmente ou que seu tempo de execução é polinomial se a sua complexidade está limitada por um polinômio P (n). Um algoritmo diz-se limitado exponencialmente o que seu tempo de execução é exponencial se a sua complexidade está limitada por uma expressão do tipo $2^{P(n)}$.

A classe de todos os problemas que admitem algoritmos polinomiais é chamada de P. Diz-se que um problema P_1 pertence à classe P se existe algum algoritmo em tempo polinomial que resolva este problema. Os problemas polinomiais são considerados "fáceis" (não importa o grau do polinômio).

A classe NP contém todos os problemas que admitem algoritmos não determinísticos polinomiais. Estes algoritmos usam uma operação que faz que dada uma lista de possibilidades esta "adivinhe" qual é a correta. Os algoritmos não determinísticos polinomiais têm um tempo de execução exponencial.

Existem também os problemas pertencentes à classe NP-Completo (NPC). Um problema P_1 diz-se NP-completo se P_1 pertence a NP e se todos os problemas da classe NP podem ser transformados polinomialmente a P_1. Cada problema em NP pode ser transformado em tempo polinomial em um dado problema NPC tal que a solução do problema NPC é a solução de outro problema em NP.

Finalmente existem os problemas pertencentes à classe NP-Hard. Um problema P_1 pertence à classe NP-hard se todos os outros problemas da classe NP podem ser reduzidos a P_1 polinomialmente.

Existem duas metodologias para avaliar a complexidade computacional de um algoritmo Glover et al. (1985).

Uma delas consiste em fazer uma abordagem teórica no pior caso do número de somas, multiplicações, comparações, ou de alguma outra medida do trabalho requerido para resolver um problema. Por exemplo, em problemas relacionados com redes este valor está relacionado com o número de nós e/ou arcos. Fazer uma análise no pior caso fornece informações relacionadas com a dificuldade da solução de problemas e de algoritmos individuais desenvolvidos para um tipo específico de problema com dados extremos.

O outro método para avaliar a complexidade computacional é o teste empírico. O computador em que se faz a implementação dos algoritmos é utilizado tanto para avaliar a eficiência de estes como para fornecer estatísticas sobre as operações dos principais passos algorítmicos desenvolvidos em diferentes condições de prova. Estas estatísticas permitem aos pesquisadores ter informação sobre como executar diferentes algoritmos desde distintas perspectivas e assim melhorar o desenvolvimento de algoritmos. Esta forma de avaliação é um processo que é análogo às pesquisas de outras disciplinas que são feitas em laboratórios.

Uma característica importante dos testes teóricos no pior caso, é que os resultados obtidos com este tipo de testes são válidos para a estrutura topológica das redes testadas, pois o desempenho de um algoritmo depende do tipo de estrutura dos dados. Este fato pode ser observado em alguns trabalhos de avaliação teórica no pior caso que testam o desempenho de alguns algoritmos com diferentes tipos de redes como: Russy e Palacios (2005), Goldberg (2001), Guerreiro (2000), Chen e Powell (1997), Cherkassky et al. (1996), Bertsekas (1993), Divoky e Hung (1990), Gallo e Pallotino (1988), Glover et al. (1985), Denardo e Fox (1979).

Outra característica presente nos trabalhos mencionados, é que com o método de avaliação teórica no pior caso não é possível obter informações sobre o desempenho de um algoritmo no caso médio, isto faz com que os usuários de algoritmos de busca do caminho mais curto possam ter problemas na hora de utilizar um algoritmo com dados reais, pois pode acontecer que algoritmos com bons resultados de desempenho no pior caso não necessariamente tenham um bom desempenho na prática.

3 O PROBLEMA DO CAMINHO MAIS CURTO

Uma empresa de distribuição de produtos deseja elaborar a rota que deverão seguir seus motoristas para fazer as entregas de maneira ótima (minimizando custo, tempo, etc.) entre diferentes pontos de uma cidade. As possibilidades de rotas a seguir com certeza serão muitas e a obtenção do melhor caminho não poderá ser feita com uma simples olhada em um mapa da cidade. Uma alternativa para resolver esta situação é fazer uma lista das possíveis rotas com suas distâncias e escolher dessa lista a que tenha o menor valor. Considerando uma cidade de grande porte esta tarefa pode-se converter em infinita. Uma maneira eficiente de resolver um problema de este tipo é modelá-lo de forma que possa ser resolvido como um problema do caminho mais curto.

Resolver o problema do caminho mais curto consiste em determinar o melhor caminho para ir de um ponto até um outro ponto de uma rede de tal forma que isto seja feito da maneira mais econômica, ou no menor tempo possível ou com o máximo da capacidade.

Pelo fato de ter inúmeras aplicações na prática como já foi mencionado de forma geral nos capítulos anteriores, este problema tem sido objeto de pesquisa de alguns investigadores que procuraram desenvolver algoritmos eficientes que permitam achar melhores soluções. Alguns dos pesquisadores que podem ser mencionados são: Bellman (1958), Dijkstra (1959), Dial (1969), Pape (1974), Martins (1984), Fredman e Tarjan (1987), Gallo e Pallotino (1988), Ahuja et al. (1990), Bertsekas (1993), Cherkassky et al. (1993), Festa (2000), Goldberg (2001, 2008).

Para fazer o estudo do problema do caminho mais curto é necessário descrevê-lo formalmente e conhecer alguns outros aspectos relacionados com este e sua solução. Neste capítulo apresenta-se a formulação matemática do problema, as hipóteses que são necessárias para fazer o estudo, os tipos de problemas do caminho mais curto que

existem e têm sido estudados, os tipos de algoritmos que existem para resolvê-lo e algumas aplicações.

3.1 FORMULAÇÃO

Seja uma rede dirigida G = (N, A) com um peso de arco c_{ij} associado a cada arco $(i, j) \in A$. Seja s ∈ G = (N, A) o nó origem da rede.

O peso de um caminho é a soma dos pesos de cada arco que pertence ao caminho. O problema do caminho mais curto é determinar para cada nó $i \in N, i \neq s$ (que não seja origem), o caminho de peso mínimo para ir do nó s até o nó i.

A formulação matemática do problema é a seguinte:

$$Minimizar \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.1a}$$

$$sujeito\ a: \quad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} n-1 & \text{para todo i = s} \\ -1 & \text{para todo } i \in N - \{s\} \end{cases} \tag{3.1b}$$

$$x_{ij} \geq 0 \quad \text{para todo} (i, j) \in A \tag{3.1c}$$

Para realizar o estudo do problema do caminho mais curto, consideram-se algumas hipóteses:

3.1.1 HIPÓTESE 1

Os valores dos pesos dos arcos são inteiros. Esta condição é necessária somente para alguns algoritmos, pois outros não precisam porque podem ser relaxados. Os problemas com valores de pesos de arcos pertencentes aos números racionais podem ser resolvidos com a transformação de um número racional em um número inteiro, através da multiplicação do primeiro por um número adequado (um número de grande valor).

Em problemas que contenham pesos de arcos associados ao campo dos números irracionais, estes devem ser convertidos em números racionais para que possam ser representados em um programa de computação.

Pelas razões expostas, a hipótese de que os pesos dos arcos sejam inteiros, na prática não se faz necessária estritamente.

3.1.2 HIPÓTESE 2

A rede contém um caminho orientado desde o nó s até qualquer outro nó i da rede.Se a rede não satisfizer esta condição, pode-se aderir um arco fictício (s,i) com um peso de grande valor para cada nó i que não tenha conexão com s em um caminho orientado.

3.1.3 HIPÓTESE 3

A rede não contém ciclos negativos. Alguns problemas do caminho mais curto podem conter arcos com pesos negativos, contudo, se apesar disto acontecer o grafo não tiver um ciclo negativo acessível a partir do nó origem é possível determinar o caminho mais curto, sem ser importante o fato de existir arcos com pesos negativos. No entanto, se existir um ciclo negativo do nó origem até os outros nós, não é possível encontrar uma solução porque esta não está definida, pois sempre será possível encontrar um caminho mais curto através de uma nova passada pelo ciclo negativo. Observar que nesta situação a expressão (3.1) não tem uma solução definida. Se existe um ciclo de peso negativo em algum caminho a partir do nó origem s até um nó i define-se este peso como infinito.

O problema do caminho mais curto com um ciclo negativo é um problema difícil de resolver, é um problema NP completo. Quando existe um ciclo negativo em algum caminho a partir do nó origem *s* até um nó i, o peso do arco (s, i) é representado por infinito.

3.1.4 HIPÓTESE 4

A rede é orientada. Se a rede na que se deseja encontrar o caminho mais curto é não orientada, mas todos os pesos dos arcos são não negativos, é possível transformar esta rede em orientada.

Um arco não orientado (i, j) com peso $c_{ij} \geq 0$ e capacidade u_{ij} permite que o fluxo passe do nó i até o nó j, assim como do nó j até o nó i . O fluxo total tem um valor limite máximo de u_{ij} (isto é o fluxo do nó i até o nó j mais o fluxo do nó j até o nó i).

O modelo não orientado tem na função objetivo a expressão $c_{ij}x_{ij} + c_{ij}x_{ji}$, e a restrição $x_{ij} + x_{ji} \leq u_{ij}$.

Desde que o custo seja $c_{ij} \geq 0$, em alguns casos na solução ótima x_{ij} e x_{ji} podem ser zero.

Por notação nesta parte o grafo não orientado (i,j) será representado como $\{i, j\}$. Assume-se que o fluxo mínimo em qualquer direção do arco $\{i, j\}$ é zero.

Para transformar o problema não orientado em orientado, cada arco não orientado $\{i, j\}$ será substituído por dois arcos orientados (i,j) e (j,i), ambos com custo $c_{ij} \geq 0$ e capacidade u_{ij} . Para justificar esta substituição é preciso mostrar que para cada fluxo da rede original que não se sobrepõe existe um fluxo associado na rede transformada com igual custo, e vice-versa.

Se o arco não orientado $\{i, j\}$ transporta α unidades de fluxo do nó i até o nó j, na rede transformada $x_{ij} = \alpha$ e $x_{ji} = 0$. Se o arco não orientado $\{i, j\}$ transporta α unidades de fluxo do nó j até o nó i, na rede transformada $x_{ij} = 0$ e $x_{ji} = \alpha$.

Agora, em forma inversa se x_{ij} e x_{ji} são os fluxos nos arcos (i,j) e (j,i) na rede orientada, x_{ij} - x_{ji} ou $x_{ji} - x_{ij}$ são os fluxos associados ao arco $\{i, j\}$ na rede não orientada, prevalecendo o valor positivo. Se x_{ij} - x_{ji} é positivo, o fluxo do nó i até o nó j no arco $\{i, j\}$ é igual a este valor. Se $x_{ji} - x_{ij}$ é positivo, este será o valor do fluxo do nó j até o nó i no arco $\{i, j\}$. Em ambos os casos, o fluxo na direção contrária é zero. Se $x_{ji} - x_{ij} = 0$, o fluxo no arco $\{i, j\}$ é 0.

3.2 TIPOS DE PROBLEMAS DO CAMINHO MAIS CURTO

Existem alguns tipos de problema do caminho mais curto, estes são:

- Determinar o caminho mais curto entre um par específico de nós.

- Determinar o caminho mais curto entre todos os pares de nós da rede.

- Determinar o segundo, terceiro, quarto, etc., caminho mais curto.

- Encontrar o caminho mais rápido em uma rede com tempos de viagem e dependendo de uma hora de saída.

- Encontrar o caminho mais curto entre nós finais específicos precisando-se para chegar a eles percorrer nós intermediários

A seguir apresenta-se uma descrição geral de cada um dos tipos de problema citados acima. Recomenda-se ao leitor que tenha interesse em aprofundar mais sobre os pesquisadores que desenvolveram algoritmos para resolver o problema do caminho mais curto revisar Dreyfus (1969) e Schrijver (2005).

3.2.1 O CAMINHO MAIS CURTO ENTRE UM PAR ESPECÍFICO DE NÓS

Dado um conjunto de N nós, numerados arbitrariamente de 1 até N, e uma matriz C de ordem NxN, não necessariamente simétrica, cujos elementos c_{ij} representam o peso do arco orientado (i,j), encontrar o caminho de menor peso para conectar o nó 1 com o nó N.

Inicialmente assume-se que $c_{ii} = 0$ e $c_{ij} \geq 0$. Se não existir um arco orientado entre o nó i e o nó j, então $c_{ij} = \infty$, em casos de programas computacionais c_{ij} é um valor muito alto.

O primeiro algoritmo mais eficiente para resolver este problema foi desenvolvido por Dijkstra (1959), (Dreyfus, 1969; Ahuja et al. 1993).

O algoritmo de Dijkstra é o mais famoso dos algoritmos para a determinação do caminho mínimo entre um par de nós de um grafo e, na prática, o mais empregado (Festa, 2006; Ahuja et al. 1990).

Escolhido um nó como origem da busca, este algoritmo calcula o custo mínimo para ir do nó para todos os demais nós do grafo. O algoritmo pode ser usado sobre grafos orientados ou não, e admite que todos os arcos tenham pesos não negativos ou zero. Esta restrição é perfeitamente possível no contexto de redes de transportes, onde

os arcos representam normalmente distâncias ou tempos médios de percurso; poderão existir, no entanto, aplicações onde os arcos apresentam pesos negativos, em estes casos o algoritmo não funcionará corretamente.

Para o caso de redes com pesos de arcos negativos, autores como Pape (1974), Martins (1984), Fredman e Tarjan (1987), Gallo e Pallotino (1988), Cormen et al. (1990), Ahuja et al. (1990), Bertsekas (1993), Cherkassky et al. (1993), Festa (2000), Goldberg (2001) desenvolveram algoritmos para resolver este tipo de problema.

3.2.2 DETERMINAR O CAMINHO MAIS CURTO ENTRE TODOS OS PARES DE NÓS DA REDE

Para determinar o caminho mais curto entre todos os pares de nós, pode-se aplicar o algoritmo de Dijkstra (descrito em 3.2.1) recursivamente n vezes, utilizando cada nó, sucessivamente, como nó origem.

Dreyfus (1969) menciona que Floyd em 1962 apresentou o primeiro algoritmo para determinar o caminho mais curto nesta situação, baseado no procedimento de Warshall de 1962, e que posteriormente este mesmo algoritmo depois foi melhorado por Murchland em 1965.

Outros pesquisadores que desenvolveram algoritmos para resolver este tipo de problema do caminho mais curto são Johnson (1977) e Halpein e Zwick (1997). Em Han et al. (2006) o leitor que tenha interesse em aprofundar em este tipo de problema pode encontrar ampla informação.

3.2.3 DETERMINAR O SEGUNDO, TERCEIRO, QUARTO, ETC., CAMINHO MAIS CURTO

O problema consiste em determinar os melhores caminhos (estabelecendo uma ordem entre eles) para ir de um nó específico 1 até um outro nó específico N.

Com a resolução de este problema é possível obter não somente o caminho mais curto entre dois nós de uma rede, se não também o segundo, o terceiro,..., o k-ésimo caminho de menor ou igual valor do j-ésimo caminho, com $j > k$.

A busca de mais de uma solução ótima faz-se necessária quando por alguma situação complexa é preciso considerar algumas restrições, e por causa disto a melhor solução não pode ser o caminho mais curto. Neste caso o objetivo é procurar o caminho

que além de ser curto satisfaça a restrição mencionada. Por exemplo, em problemas de roteamento de veículos, é preciso determinar caminhos alternativos a serem usados no caso que o caminho mais curto apresente algum problema como engarrafamentos, vias suspendidas por realização de trabalhos, acidentes, etc.

Dois caminhos são considerados diferentes quando apesar de visitar os mesmos nós, estas visitas são realizadas em ordens diferentes. Assume-se que os valores das distancias de todos os caminhos determinados são diferentes. As soluções que contemplam caminhos com laços também são consideradas, e efetivamente em alguns casos podem ser considerados como segundo melhor caminho em problemas com $d_{ii} > 0$.

Dreyfus (1969) menciona que Hoffman e Pavley em1959 apresentaram o primeiro algoritmo para encontrar os k melhores caminhos e que Bellman e Kalaba em 1960 apresentaram posteriormente outro algoritmo para resolver este tipo de problemas.

Outros algoritmos foram desenvolvidos Dreyfus em 1969, Yen (1971), Martins em 1984, Hershberger (2008).

O leitor que tenha interesse em aprofundar sobre este tipo de problema do caminho mais curto, pode revisar Guerreiro et al. (2001) e Eppstein (1994), que oferecem informação interessante sobre alguns pesquisadores que trabalharam para resolver este tipo de problema.

3.2.4 ENCONTRAR O CAMINHO MAIS RÁPIDO EM UMA REDE COM TEMPOS DE VIAGEM E DEPENDENDO DE UMA HORA DE SAÍDA

O problema do caminho mais curto dependente do tempo está definido em uma rede que tem características que mudam com o tempo, sendo estas mudanças previsíveis. Este tipo de problemas tem aplicações freqüentes em áreas de planejamento de transporte, transportação veicular e roteamento de comunicações.

O problema consiste em encontrar o caminho mais rápido entre cidades considerando que o tempo de viagem da cidade i até a cidade j depende da hora de saída da cidade i. Se t é a hora de saída da cidade i até a cidade j, $c_{ij}(t)$ representa o tempo de viagem e é um valor que pertence aos inteiros positivos. O procedimento que é iterativo consiste em determinar os mais rápidos caminhos de todas as cidades até a cidade N, começando na cidade i no tempo 0.

O caminho mais curto determinado como solução em um momento dado no tempo atual pode não ser a melhor solução em outro tempo, considerando mudanças previsíveis no futuro, como por exemplo, o comportamento do tráfego em horas pico.

Neste tipo de problemas, assume-se que o tempo de viagem de cada arco (i,j) é uma função que depende das horas de saída e é tal que todas suas funções são conhecidas com antecedência durante todo o tempo.

Os primeiros em abordar este problema foram Cooke e Holsey em 1966, (Dreyfus 1969). Outros pesquisadores que trabalharam com este tipo de problema são: Bakó (1973), Frigioni et al. (2000), Narváez et al. (2000), Ahuja et al. (2002).

3.2.5 ENCONTRAR O CAMINHO MAIS CURTO ENTRE NÓS FINAIS ESPECÍFICOS PRECISANDO-SE PARA CHEGAR A ELES PERCORRER NÓS INTERMEDIÁRIOS

Dado um conjunto N de nós e distâncias $c_{ij} \geq 0$, neste tipo de problema o objetivo é encontrar o caminho mais curto entre os nós 1 e N visitando $k-1$ nós $2,3,\ldots,k \leq N-1$, chamados nós específicos.

Dreyfus (1969) p.41, faz referência a Saksena e Kumar em 1966, como os primeiros autores que desenvolveram um algoritmo para resolver este problema, mas qualifica-o de "completamente errado" e sem sucesso.

3.3 ÁRVORE DE CAMINHO MAIS CURTO

Seja um grafo G=(N,A), com |N| = n e |A| = m.

O objetivo é determinar o caminho mais curto entre um nó raiz r e os outros (n-1) nós. Para armazenar todos os caminhos gerados precisa-se de (n-1) espaços de memória. Este resultado é conseqüência de que é possível determinar uma árvore com a característica de que o único caminho da raiz r até um outro nó é o caminho mais curto. A existência da árvore de caminho mais curto possui a seguinte propriedade:

Propriedade 1: Se o caminho $r = i_1 - i_2 - \cdots - i_h = k$ é o menor caminho do nó r ao nó k, então para cada $q = 2,3,\ldots,h-1$, o subcaminho $r = i_1 - i_2 - \cdots - i_q$ é o menor caminho do nó raiz r até um nó i_q.

A propriedade anterior implica que se P é o menor caminho do nó raiz até algum nó k, então $d(j) = d(i) + c_{ij}$ para cada arco $(i, j) \in P$. O recíproco também é verdadeiro, ou seja se $d(j) = d(i) + c_{ij}$ para algum arco que pertença ao caminho orientado P entre o nó raiz e algum nó k, então P é o caminho mínimo entre estes nós.

3.4 ALGORITMOS DE BUSCA DO CAMINHO MAIS CURTO DO MÉTODO DE ROTULAÇÃO

Os algoritmos do método de rotulação (*labeling methods algorithms*) contêm os algoritmos de caminho mais curto que começam com uma árvore T que depois é ampliada ou modificada em cada iteração, até que esta seja a árvore de caminho mínimo. Os dois tipos de algoritmos de rotulação utilizam este critério, ou seja, constroem uma árvore de caminho mínimo desde um nó raiz r até um nó i. O resultado final dos algoritmos está representado por um único caminho de r até i. Enquanto se faz a construção e melhora da árvore, três dados importantes são mantidos para cada nó i:

- A distância rotulada d(i)
- O nó predecessor de i p(i)
- O status do nó i $S(i) \in$ {não visto, rotulado, examinado}

A distância rotulada d(i) é usada para registrar o limite superior da distância do nó raiz r até um outro nó i na árvore. No final de todas as iterações a distância rotulada representa a menor distância do r até o no i.

O predecessor do nó i, p(i) armazena o nó que precede imediatamente ao nó i na árvore.

Um nó tem o status de não visto quando sua distância rotulada tem o valor de infinito (computacionalmente este valor está representado por uma quantidade muito grande).

Um nó tem o status de rotulado quando sua distância foi atualizada pelo menos uma vez, ou seja, a sua distância é diferente do infinito.

Um nó é considerado examinado se nele foi realizada a operação de examinado que consiste em diminuir a distância rotulada para algum nó j tal que $(i, j) \in A$, e conseqüentemente a sua distância rotulada não pode ser mais atualizada.

O método de rotulação inicia com os seguintes dados:

- d(i) = ∞
- p(i) = Ø
- S(i) = não examinado para cada nó i
- d(r) = 0
- S(r) = rotulado

O método avança através do examinado aos nós rotulados começando pelo nó raiz r até que o nó destino i é examinado e rotulando permanentemente (rotulação permanente) ou até que os nós sejam examinados e rotulados permanentemente (correção de rotulação). O método finaliza quando é determinada a árvore de caminho mínimo em um subconjunto de nós (rotulação permanente) ou no conjunto inteiro de nós (correção de rotulação).

Se a distância rotulada ao nó j pode ser diminuída a árvore é modificada pela substituição dos predecessores do nó j pelo conjunto p(j) = i e o nó j é considerado rotulado.

Cada nó cuja distância rotulada foi melhorada é colocado em uma lista de nós examinados elegíveis *(scan eligible node list)*, representada por S. Esta lista no inicio está vazia.

Em resumo a operação de examinado de um nó i para os dois métodos de rotulação é:

```
procedimento examinar (i)
para todo arco (i,j) ∈ A fazer
se d(i) + c_ij < d(j) então
d(j) = d(i) + c_ij
S(j) = rotulado
P(j) = i
S(i) = examinado
fim
```

Figura 12 Procedimento para examinar nós

Os algoritmos do método de rotulação dividem-se em dois tipos: algoritmos de rotulação permanente (*labeling-setting algorithms*) e algoritmos de correção de rótulos (*labeling-correcting algorithms*). Na maior parte da literatura relacionada com algoritmos de caminho mais curto, este tema é abordado com esta terminologia.

A maioria dos algoritmos do caminho mais curto baseiam-se no método de rotulação (Festa 2006, Cherkassky et al. 1996; Ahuja et al. 1993).

Depois de fazer uma descrição geral do método de rotulação agora se fará uma descrição específica de cada um dos tipos de algoritmos de este método.

3.4.1 ALGORITMOS DE ROTULAÇÃO PERMANENTE

A aplicação dos algoritmos de rotulação permanente pode ser feita a redes com arcos de pesos não negativos unicamente. Começam com uma árvore T que contém um nó raiz r e um conjunto de distâncias rotuladas tal que $d(r) = 0$ e $d(i) = \infty$.

O nó raiz e seus arcos associados são examinados com o objetivo de determinar se alguma distância pode ser melhorada. Esta etapa é chamada de examinado.

Cada nó cuja distância rotulada foi melhorada é colocado na lista S, e seu arco associado é registrado. Em cada seguinte iteração o nó que pertence a S com a mínima distância rotulada é selecionado, removido da lista e agregado à arvore T junto com seu arco associado. Este nó é examinado em forma análoga com o nó raiz, e depois a lista S é atualizada. O algoritmo termina quando a lista S fica vazia.

A propriedade fundamental dos algoritmos de rotulação permanente é que T é sempre a árvore de menor caminho desde r até todos os nós que pertencem a T em cada iteração. Portanto o método de rotulação permanente desenvolve-se em exatamente |n| iterações e examina cada arco uma vez só. Pode-se parar o algoritmo depois de realizar

as |n| iterações. É conveniente fazer isto especialmente quando se for trabalhar com o limite no pior caso. Observa-se que estes algoritmos só examinam os nós com distâncias rotuladas.

Uma característica dos algoritmos de rotulação permanente é que desenvolvem um procedimento que encontra o nó de mínima distância rotulada na lista de nós SE.

O primeiro algoritmo de rotulação permanente é atribuído a Dijkstra (1959) com uma complexidade no pior caso de $O(|n|^2)$, outros algoritmos foram desenvolvidos por Dantzing (1960), Hillier e Whiting (1960). Têm sido desenvolvidas algumas novas versões do algoritmo de Dijkstra, estas versões deram origem ao nascimento da família dos algoritmos mais eficientes (Festa, 2006). Algumas de estas versões são apresentadas no Capítulo 4.

3.4.2 ALGORITMOS DE CORREÇÃO DE RÓTULOS

Os algoritmos de correção de rótulos são mais gerais do que os algoritmos de rotulação permanente, pois podem ser aplicados a redes com arcos de pesos negativos além de que podem começar com qualquer árvore. Não obstante existem alguns algoritmos de correção de rótulos que são para ser usados especificamente com redes de pesos não negativos.

Em cada iteração, os algoritmos de correção de rótulos tentam melhorar a árvore T através da redução da distância rotulada de pelo menos um nó que lhe pertença com a agregação de um novo arco e um novo nó a T, esta agregação é feita substituindo um arco existente. O objetivo de fazer esta substituição é diminuir o caminho até o nó em na árvore.

```
Começar
d(s): = 0 e pred (s): = 0;
d(j): = ∞ para cada j ∈ N - {s};
enquanto algum arco (i,j) satisfaça d(j) > d(i) + c_ij fazer
começar
d(j): = d(j) + c_ij;
pred(j):= i;
fim;
fim;
```

Algoritmo 1 Genérico correção de rotulação

Com esta descrição pode-se já apreciar uma diferença entre os dois tipos de métodos de algoritmos rotulados. O critério de seleção de nós dos algoritmos de rotulação permanente é escolher o nó com a menor distância rotulada da lista de nós SE, enquanto os algoritmos de correção de rotulação podem selecionar qualquer nó de esta lista.

Lembrando a propriedade fundamental dos algoritmos de rotulação permanente que é a existência de uma árvore T que é sempre a árvore de menor caminho desde r até todos os nós que lhe pertencem em cada iteração, esta propriedade não tem mais sentido com a forma de seleção de nós nos algoritmos de correção permanente. Também isto implica que a complexidade de estes algoritmos não seja polinomial.

O primeiro algoritmo de correção rotulada é atribuído a Bellman – Ford - Moore e tem uma complexidade no pior dos casos de O(|n||a|), (Glover et al.1985). Neste algoritmo os nós são selecionados de lista de nós S com o critério da regra primeiro em entrar primeiro em sair (FIFO, first-in-first-out). Yen (1971) fez modificações a este algoritmo.

Outros autores desenvolveram algoritmos de correção de rótulos utilizando o critério último em entrar primeiro em sair (LIFO *last-in-first-out*), nestes algoritmos os primeiros nós em ser agregados à árvore são aqueles que têm uma posição na frente da lista de nós SE e de igual forma estes são os primeiros em ser removidos.

D'Esopo e Pape (1974) desenvolveram um algoritmo utilizando um critério bidirecional que consiste em agregar um nó que ocupe um lugar na frente da lista de nós SE só se este aparecesse na lista antes de chegar ao fim, caso contrário os nós da frente da lista são removidos. Grande parte dos esforços estão concentrados no desenvolvimento de algoritmos do método de correção de rotulação, isto devido a que este tipo de algoritmos são mas gerais na aplicação, pois determinam caminhos mais curtos em redes com pesos negativos e não negativos. Alguns autores são: Martins (1984), Fredman e Tarjan (1987), Gallo e Pallotino (1988), Bertsekas (1993), Cherkassky et al. (1993), Goldberg and Radzik (1993), Festa (2000), Goldberg (2001).

3.5 APLICAÇÕES

O problema do caminho mais curto é um tema de muita importância na área cientifica pela grande quantidade de problemas que na prática podem ser resolvidos utilizando o critério de busca do caminho mínimo.

Além das aplicações que estão associadas diretamente à busca do caminho mais curto, alguns outros problemas de programação inteira podem ser resolvidos através da formulação de este problema em forma direta ou aplicando alguma estratégia de relaxamento.

Os seguintes problemas de programação inteira que foram resolvidos com o critério do caminho mais curto: localização, problema generalizado de atribuição, problema de emparelhamento máximo, problema do caixeiro viajante, problema da mochila e problemas de equilíbrio de tráfego (Festa, 2006; Ahuja et al. 1993; Divoky e Hung, 1990).

Como foi mencionado no capítulo introdutório da presente dissertação, é de interesse neste trabalho além de avaliar os algoritmos propostos, mostrar ao leitor como o problema do caminho mais curto pode ser usado na prática com exemplos simples, portanto foram realizadas duas aplicações que são apresentadas a seguir:

3.5.1 EXEMPLO 1

Na Figura 13 vai ser determinado o caminho mais curto para unir o nó 1 e o nó 6.

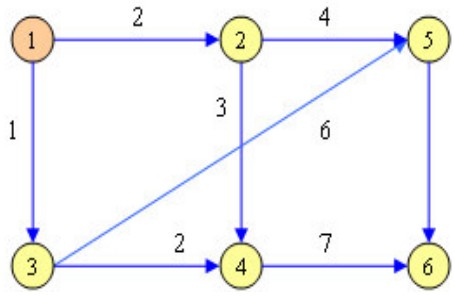

Figura 13 Rede orientada

As possibilidades para ir do nó 1 até o nó 6 são:

Trajeto	Distância
1 – 2 – 5 – 6	14
1 – 3 – 4 – 6	10
1 – 2 – 4 – 6	12
1 – 3 – 5 – 6	15

Tabela 2 Trajetos para unir o nó 1 e o nó 6 da Figura 13

Na Tabela 2 o caminho mais curto é o caminho 2.

Para a resolução de este e outros exemplos maiores realizou-se a implementação do algoritmo de Dijkstra com o compilador e editor de FORTRAN, FORCE 2.0. Os resultados obtidos no programa realizado para a rede da Figura 13 são os seguintes:

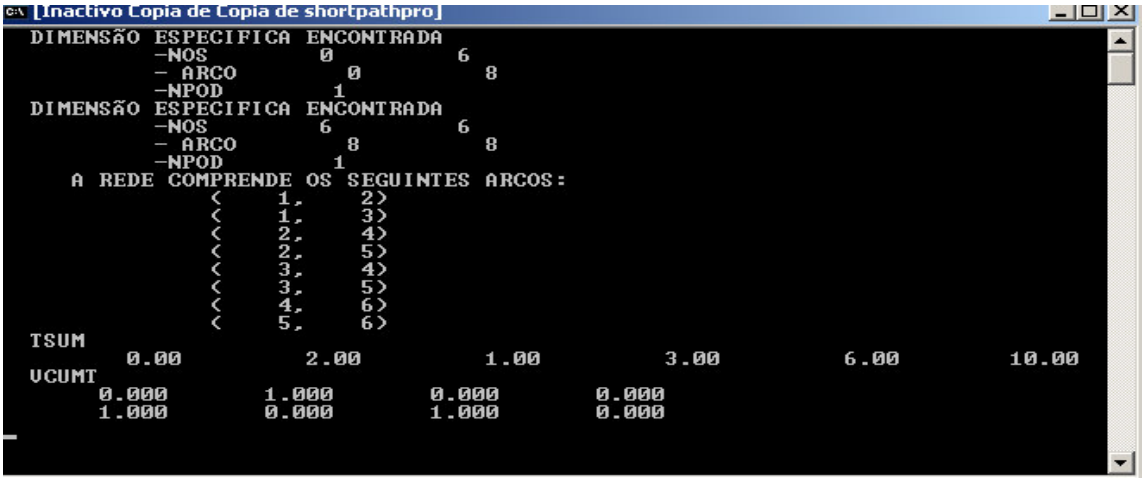

Figura 14 Resultados de implementação do algoritmo de Dijkstra

Dimensão específica contém o número de nós e arcos, 6 e 8 respectivamente. Depois aparecem cada um dos arcos da rede, arco 1 compreendido entre os nós (1, 2), arco 2 compreendido entre os nós (1, 3) e assim por diante até o arco 8 compreendido entre os nós (5, 6). TSUM contém as distâncias rotuladas mínimas obtidas a cada iteração (comparar com a Tabela 2). Finalmente VCUMT indica os arcos que pertencem ao caminho mais curto, tal que se o valor que aparece é 1, então o arco pertence ao caminho mais curto e 0 no caso contrário. Assim para a rede dada a solução contém os arcos 2, 5 e 7 (linha vermelha) assim:

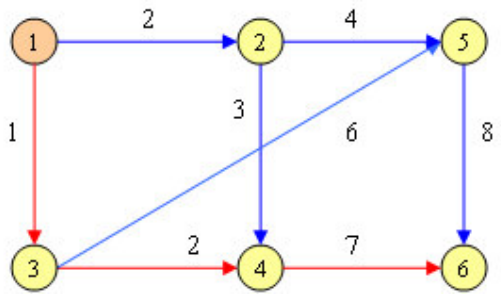

Figura 15 Caminho mínimo da Figura 13 determinado a partir do algoritmo de Dijkstra

3.5.2 EXEMPLO 2

Para a rede da Figura 16 que contém 20 nós e 30 arcos, o objetivo é determinar o caminho mais curto para unir o nó origem 1 com o nó destino 20.

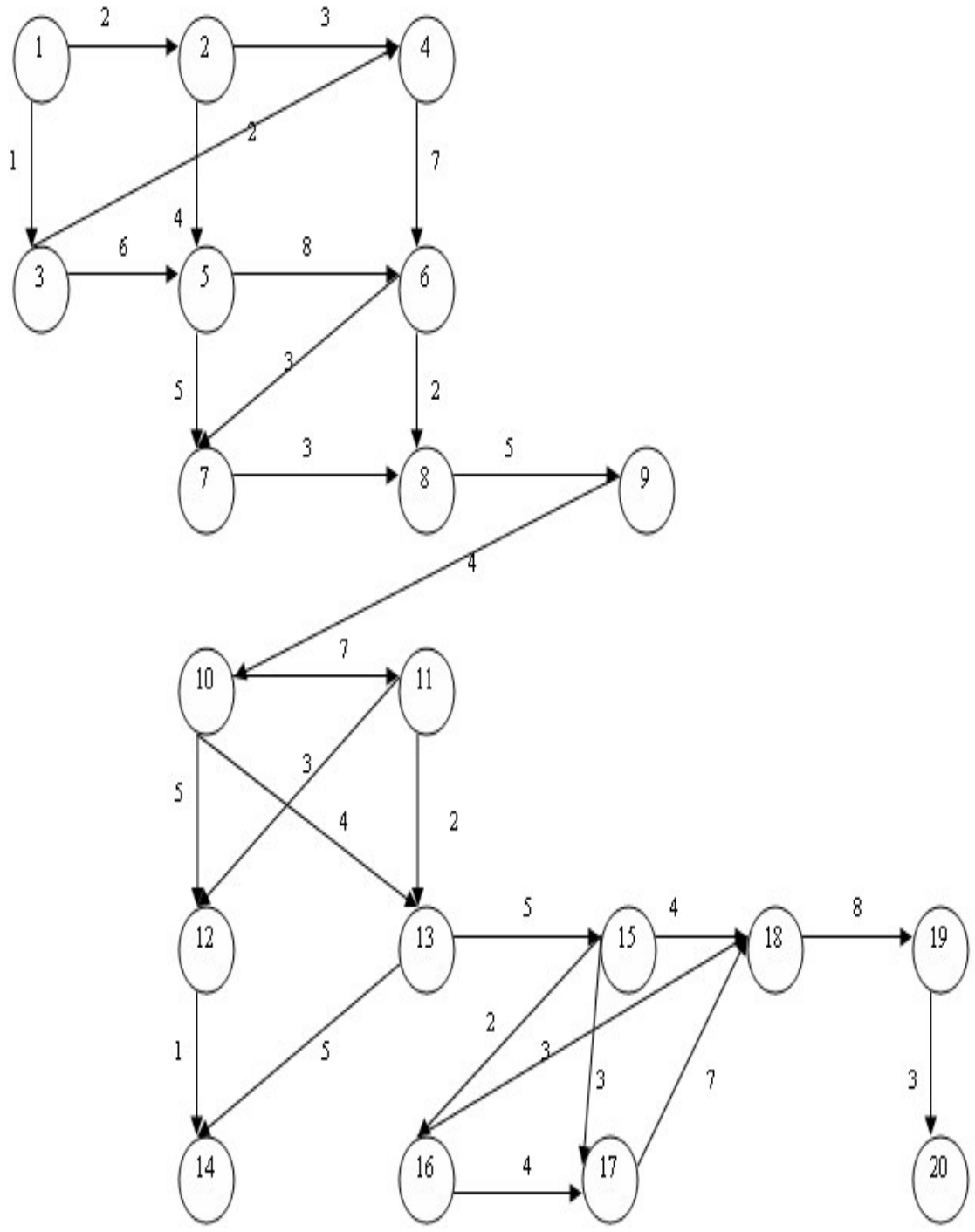

Figura 16 Rede do exemplo 3.5.2

A solução obtida na implementação do algoritmo apresenta-se a seguir:

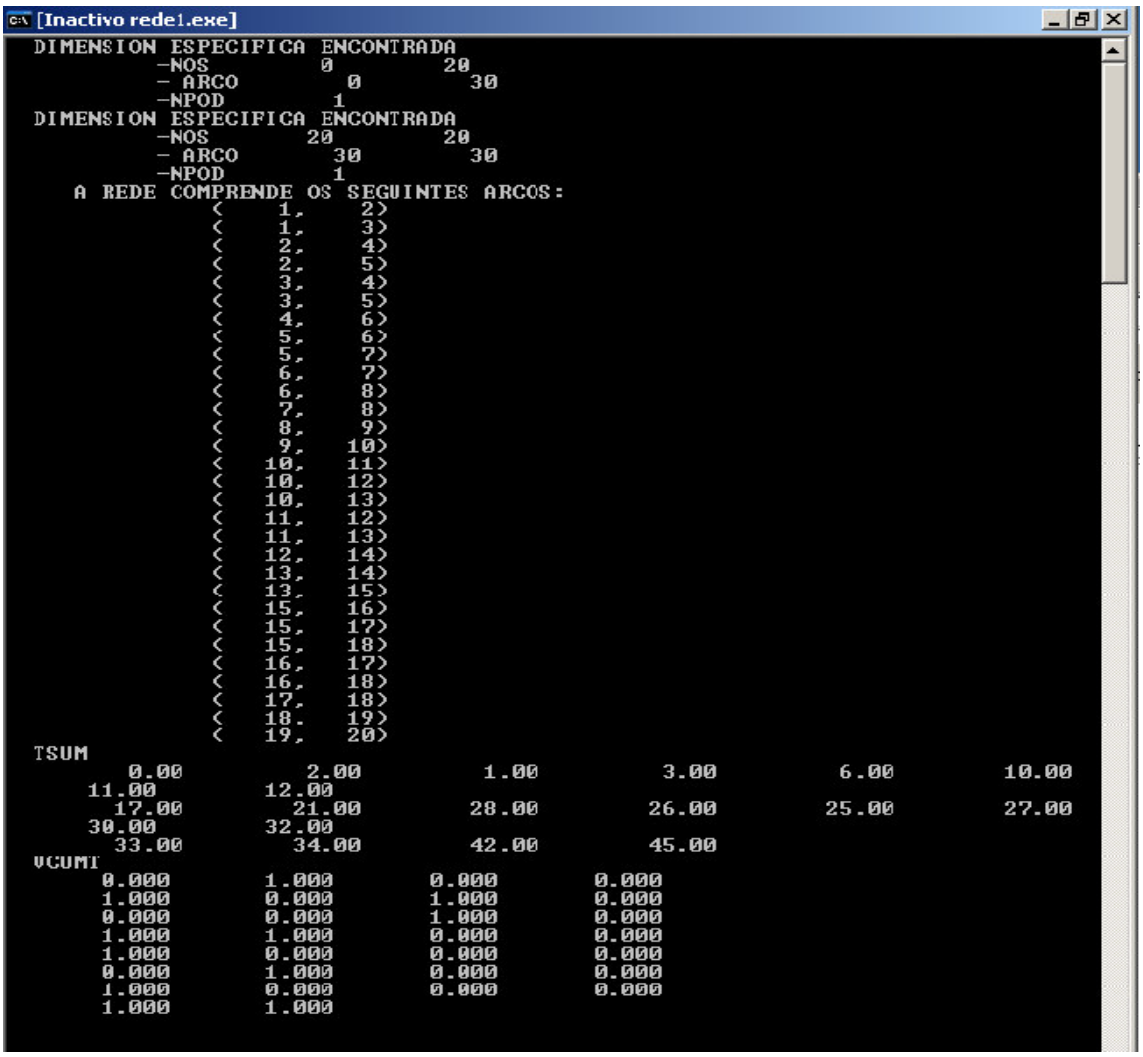

Figura 17 Caminho mínimo para a rede da Figura 16 determinado a partir da implementação do algoritmo de Dijkstra

Dimensão específica contém o número de nós e arcos, 20 e 30 respectivamente. Depois aparecem cada um dos arcos da rede, arco 1 compreendido entre os nós (1, 2), arco 2 compreendido entre os nós (1, 3) e assim por diante até o arco 20 compreendido entre os nós (19, 20). TSUM contém as distâncias rotuladas mínimas obtidas a cada iteração. Finalmente VCUMT indica os arcos que pertencem ao caminho mais curto, tal que se o valor que aparece é 1, então o arco pertence ao caminho mais curto e 0 no caso contrário. Assim na rede dada para ir do nó um até o nó 20, a solução contém os arcos

2, 5, 7, 11, 13, 14, 17, 22, 25, 29, 30. Ver Figura N 18. As linhas vermelhas indicam a rota do caminho mínimo. O valor do caminho mínimo é 45.

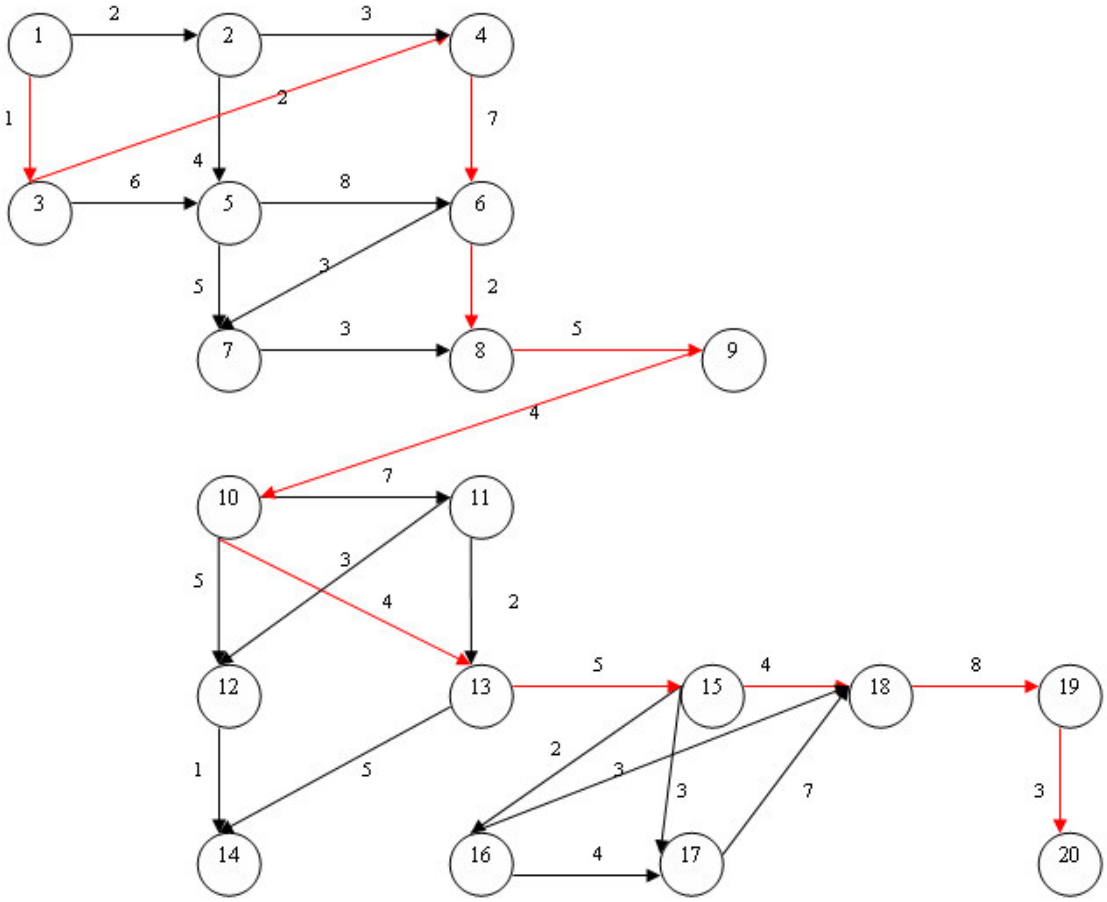

Figura 18 Caminho mínimo para a rede da Figura 16

3.6 AVALIAÇÕES DE ALGORITMOS DE BUSCA DO PROBLEMA DO CAMINHO MAIS CURTO

Até a atualidade alguns trabalhos de avaliação de algoritmos de busca do caminho mais curto foram realizados. Revisando estes trabalhos obtiveram-se algumas observações que são apresentadas a seguir :

- Avaliações teóricas no pior caso com diferentes tipos de estruturas topológicas de dados, por exemplo, redes aleatórias, redes acíclicas, redes de grades quadradas, longas, compridas, de trânsito. Trabalhos de este tipo foram feitos por: Russy e Palacios (2005), Goldberg (2001),

Guerreiro et al. (2001), Chen e Powell (1997), Cherkassky et al. (1996), Bertsekas (1993), Divoky e Hung (1990), Gallo e Pallotino (1988), Glover et al. (1985), Denardo e Fox (1979).

- Realizando melhoras nos algoritmos tradicionais. Neste tipo de trabalhos a comparação do desempenho dos algoritmos é feita entre o algoritmo original e a nova versão. Por exemplo, em Ahuja et. al (1990), apresenta-se uma implementação ao algoritmo de Dijsktra original. Outros pesquisadores que realizaram trabalhos de este tipo são: Goldberg (2008), Bardossy e Shier (2006), Duin (2005), Festa et al. (2000), Cherkassky et al. (1996), Goldberg e Silverstein (1997), Bertsekas (1993), Ahuja et al. (1990), Gallo e Pallotino (1988). Em estes trabalhos podem ser encontradas avaliações com distintos tipos de estruturas de dados.

- Avaliações com redes reais: Klunder e Post (2006), Lux e Furtado (2001), Zhan e Noon (2000), Zhan e Noon (1998), Zhan (1997).

- O maior foco de atenção dos trabalhos de avaliação concentra-se nos algoritmos do método de correção de rotulação devido a que estes algoritmos são mais gerais na sua aplicação, pois podem resolver problemas do caminho mais curto em redes que contenham pesos de arcos negativos ou não negativos. Apesar de este fato acontecer, pode-se apreciar um grande interesse em avaliar e/ou fazer melhoras ao algoritmo de Dijkstra que pertence aos algoritmos do método de rotulação permanente. Este interesse é devido a que o algoritmo de Dijkstra é considerado um dos mais eficientes para resolver problemas do caminho mais curto em redes com pesos não negativos.

- Nos trabalhos consultados pode-se apreciar que os trabalhos mais antigos são uma base de referência para os trabalhos mais novos, no sentido de que a maioria de autores antes de escolher os algoritmos a serem avaliados nos seus trabalhos, considera os desempenhos obtidos pelos

algoritmos em trabalhos anteriores, novas estratégias de busca, estruturas mais eficientes de implementação.

- Nos trabalhos consultados de avaliação de algoritmos de busca do caminho mais curto, os algoritmos foram implementados em C, FORTRAN, CRAY J90 em ambiente UNIX e WINDOWS.

4 AVALIAÇÃO DOS ALGORITMOS

Neste capítulo se faz uma exposição de cada um dos algoritmos mencionados, descreve-se as características do problema teste e da realização dos testes, e se apresentam os resultados obtidos em cada um dos testes. Antes de começar a revisão dos algoritmos, a seguir são apresentadas algumas informações relacionadas com este trabalho e com a escolha dos algoritmos avaliados.

- A presente dissertação foi realizada com a abordagem de avaliação teórica no pior caso com redes da família de grades. Contém a implementação e avaliação dos algoritmos de Dijkstra com heaps, D'Esopo – Pape, *Small Label First* e *Small Label First Threshold*, utilizando compilador FORCE 2.0, que é um sistema de desenvolvimento de programas de FORTRAN para ambiente WINDOWS.

- Os algoritmos escolhidos determinam o caminho mais curto de um nó origem até um nó destino. O algoritmo de Dijkstra trabalha com pesos de arcos não negativos e os algoritmos de D'Esopo Pape, Small Label First e Small Label First Threshold, trabalham com pesos de arcos negativos e não negativos.

- Os algoritmos mencionados pertencem aos dos tipos de algoritmos do método de rotulação, ou seja, rotulação permanente e correção de rotulação.

- O algoritmo de Dijkstra pode ser considerado um algoritmo clássico, porque formalmente é considerado o primeiro algoritmo do método de rotulação permanente, além está presente em boa parte dos trabalhos

revisados. Apesar de que o algoritmo de Dijkstra é do ano 1959, é um dos mais utilizados por sua eficiência. De ai que até agora os pesquisadores realizam melhoras em este algoritmo.

- O algoritmo de D'Esopo Pape pode ser considerado de um algoritmo tradicional, pois também tem sido objeto de algumas avaliações e melhoras por parte de alguns pesquisadores.

- Os algoritmos SLF e SLFT são algoritmos com estratégias relativamente mais novas, e há uma presença menos forte de estes algoritmos nos trabalhos existentes. O motivo de ter incluído este algoritmos em este trabalho é porque pessoalmente existe o interesse em realizar pesquisas futuras de possíveis combinações entre os algoritmos do tipo SLF e os outros algoritmos da família Threshold.

- Nos trabalhos consultados não foi encontrada uma avaliação que contenha a combinação dos algoritmos: Dijkstra com heaps, D'Esopo – Pape, SLF e SLFT.

- Existe informação sobre a eficiência relativa e desempenho dos algoritmos avaliados. Isto com o intuito de ter uma referência e comparar os resultados obtidos neste trabalho.

- As avaliações feitas neste trabalho baseiam-se em três pesquisas previas realizadas sobre o tema, Guerreiro et al. (2001), Bertsekas (1993) e Glover et al. (1985). Isto com objetivo de facilitar as comparações dos resultados obtidos. Estes trabalhos foram escolhidos como referência porque contêm avaliações com os algoritmos da estratégia SLF e Threshold.

- O trabalho de Guerreiro et al. (2001) avalia alguns algoritmos do método de correção de rotulação entre eles SLF e SLFT. Um dos conjuntos de dados pertence à família de redes com grade. Bertsekas (1993) avaliou

além de outros algoritmos DEP SLF e SLFT. Os testes também foram realizados com redes de grade. O conjunto de provas foi gerado pelo gerador GRIDGEN Bertsekas (1992), com modificações próprias de cada autor.

- O trabalho de Glover et al. (1985) contém a avaliação dos algoritmos da família threshold. Em este trabalho avaliam-se estes algoritmos com estruturas de rede com grade, fazendo para o mesmo número de nós algumas combinações.

- O conjunto de redes foi gerado por GRIDGEN com algumas modificações que são descritas em 4.4.1.

- A implementação feita em FORCE 2.0, contém a programação dos algoritmos descritos e de um gerador de redes. Os programas contêm listas de arcos devido a que uma estrutura muito eficiente para resolver este tipo de problemas.

- Nos estudos realizados nenhum deles, utilizou o compilador FORCE 2.0. Este sistema foi escolhido porque é um programa de domínio público (pode ser conseguido na internet grátis) e tem uma utilização muito amigável.

4.1 ALGORITMO DE DIJKSTRA

O algoritmo de Dijkstra permite determinar o menor caminho de um nó origem **s** até os outros nós de uma rede orientada para o caso em que todos os pesos dos arcos sejam não negativos.

O algoritmo mantém a distância rotulada d(i) para cada nó i, e é o limite superior do caminho mais curto até o nó i.

Durante o desenvolvimento do algoritmo os nós são divididos em dois grupos: rotulados permanentemente (ou permanentes) e rotulados temporariamente (temporários). A distância rotulada permanente de algum nó i representa o caminho

mais curto deste nó até o nó origem s. No caso dos nós temporários, esta distância representa o limite superior da distância do caminho mais curto até o nó.

O algoritmo examina todos os nós sem incluir o nó origem s e rotula permanentemente os nós em ordem das distâncias ao nó origem.

Inicia-se o algoritmo com d(s) = 0 e $d(i) = \infty$, para todo nó i que pertence à rede.

Em cada iteração o rótulo do nó i representa a menor distância desde este nó i até o nó origem num caminho que contém outros nós intermediários. O algoritmo escolhe o nó **i** com a mínima distância rotulada temporária converte-a em permanente e visita os outros nós, ou seja, examina os arcos de A(i) e atualiza as distâncias rotuladas dos nós adjacentes. O algoritmo de Dijkstra finaliza quando todos os nós são designados como permanentes.

O algoritmo de Dijkstra mantém uma árvore orientada T com raiz o nó origem e que abrange os nós com distâncias finitas rotuladas. O algoritmo mantém a árvore com os índices predecessores, ou seja, para $(i, j) \in T$, armazena pred(j) = i.

Este algoritmo conserva a propriedade que para cada arco $(i, j) \in T$, $d(j) = d(i) + c_{ij}$. No fim do algoritmo quando as distâncias rotuladas representam os caminhos mais curtos, T é a árvore de caminho mínimo.

No algoritmo de Dijkstra a operação de seleção da mínima distância rotulada temporária conhece-se como "seleção de nós". Outra operação deste algoritmo é a "atualização de distâncias" que corresponde à verificação dos rótulos atuais dos nós i e j que satisfaçam a condição $d(j) > d(i) + c_{ij}$ e por tanto $d(j) = d(i) + c_{ij}$. Assim, o número de operações que o algoritmo realiza vem determinado a partir das duas operações descritas.

A seleção de nós é realizada em n nós e a partir de cada um destes são analisados (n-1) nós associados aos (n-1) arcos que saem de cada nó. Esta operação não é feita nos nós que têm o rótulo permanente por tanto a partir do k-ésimo nó selecionado são analisados n-k nós. Assim o número de operações da seleção de nós é:

$$n + (n-1) + (n-2) + \ldots + 1 = O(n^2)$$

A atualização de distâncias é realizada para cada nó durante |A(i)| vezes, e $\sum_{i \in N} | A(i) | = m$, então o algoritmo requer de executar esta operação m vezes, por tanto

a complexidade é de O(m). Com a análise das duas componentes, então a complexidade do algoritmo é:

$$O(n^2) + O(m) = O(n^2), \text{pois } m < n^2$$

A seguir apresenta-se o algoritmo de Dijsktra:

```
começar
        S: = Ø;    S̄ : = N;
        para cada nó i ∈ N;
        d(s): =0; pred(s):=s;
        d(i): = ∞; pred(s): = s;
        enquanto |S| < n fazer
        começar
                selecionar o nó  i ∈ S̄ tal que  d( i ) = min {d(j): j ∈ S̄ };
                S: = S ⋃   {i};
                S̄ : = S̄  - {i};
                atualizar a distância para cada (i, j) ∈ A(i)
                 fazer
                se d(j) > d(i) + cij então d(j) : = d(i) + cij e pred (j): = i;
        fim;
fim.
```

Algoritmo 2: Determinar o caminho mais curto Dijkstra

4.1.1 ALGORITMO DE DIJKSTRA INVERSO

No algoritmo de Dijkstra descrito anteriormente determina-se o caminho mais curto de um nó s até qualquer outro nó que pertence a N – {s}, mas se o objetivo é determinar o caminho mais curto de qualquer nó $j \in N - \{t\}$ até um nó final t, resolve-se o problema com uma modificação ao algoritmo de Dijkstra original e este algoritmo é conhecido com algoritmo de Dijkstra inverso.

O algoritmo de Dijkstra inverso mantém a distancia d'(j) para cada nó j, como um limite superior do comprimento do caminho mais curto do nó j até o nó t.

Ao igual que no algoritmo original, no algoritmo inverso de Dijkstra os nós são divididos em dois conjuntos de nós, S' que contém os nós rotulados permanentemente e $\overline{S}$' que contém os nós rotulados temporariamente. A cada iteração o algoritmo designa o

nó com a mínima distância rotulada temporária, d'(j) como permanente, e posteriormente examina cada entrada do arco (i, j) e modifica a distância rotulada do nó i a min $\{d'(i), c_{ij} + d'(j)\}$. O algoritmo finaliza igual ao algoritmo de Dijkstra original quando todos os nós tenham sido rotulados permanentemente.

4.1.2 OUTRAS VERSÕES DO ALGORITMO DE DIJKSTRA

Em cada iteração do algoritmo de Dijkstra é removido o nó com a mínima distância rotulada da lista de nós S e são atualizadas as distâncias temporárias dos nós adjacentes a i. No cálculo da complexidade do algoritmo pode-se observar que o número de operações que realiza o algoritmo não estão balanceadas, e que a operação de seleção de nós é a mais crítica pelo número de cálculos que realiza. Este fato foi considerado para melhorar o desempenho do algoritmo de Dijkstra, e foram desenvolvidos alguns algoritmos que se baseiam na versão original, e que se diferenciam na forma de representar os dados da lista de nós candidatos, na remoção e inserção de nós, e na forma de encontrar o rótulo mínimo.

Das várias versões que surgiram, as mais destacadas são: algoritmo de Dial, *heap*-Dijkstra,, d-*heap*, Fibonacci *heap*, radix *heap* e a implementação de Johnson. Na Tabela 3 apresenta-se um resumo da complexidade de cada uma das implementações.

Algoritmo	**Tempo de execução**
Dijkstra original	$O(n^2)$
Dial	$O(m + nC)$
d-*heap*	$O(m\log_d n)$, com $d = m/n$
Fibonacci *heap*	$O(m + n \log n)$
Radix *heap*	$O(m + n \log(nC))$
Jonson	$O(m \log \log C)$

Tabela 3 Complexidade algoritmos diferentes versões Dijkstra

A Figura 19 contém um gráfico que representa a complexidade de cada uma das versões do algoritmo de Dijkstra mencionadas anteriormente:

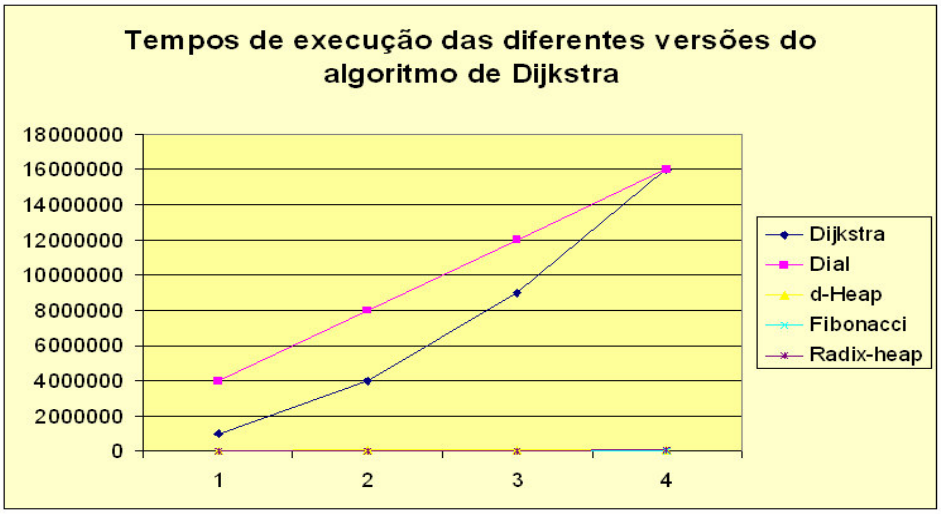

Figura 19 Complexidade algoritmos diferentes versões Dijkstra

Descreve-se em seguida além da versão do algoritmo de Dijkstra com heaps binários que será avaliada neste trabalho, as outras versões referidas acima.

4.1.2.1 Implementação de Dial

Dial em 1969 melhorou o desempenho do algoritmo de Dijkstra baseado nas seguintes propriedades:

- As distâncias rotuladas do algoritmo de Dijkstra designadas como permanentes são não decrescentes.

Esta propriedade mostra que quando é rotulado permanentemente o nó i que tem o menor rótulo temporário d(i) e enquanto os arcos de A(i) são examinados e se realiza a atualização de distâncias, a distância de algum nó rotulado temporalmente d(i) nunca decresce, isto porque os pesos dos arcos são não negativos.

O algoritmo de Dijkstra armazena os nós rotulados temporalmente com valores finitos em uma estrutura ordenada.

O algoritmo de Dial mantém nC + 1 conjuntos chamados buckets numerados de 0,1,2,...,nC, o bucket k armazena todos os nós com distâncias rotuladas iguais a k. C representa o máximo valor dos pesos da rede, e nC é um limite superior das distâncias rotuladas de algum nó rotulado com um valor finito. Não é preciso armazenar os nós com distâncias temporárias infinitas em algum dos buckets, o que se faz é inseri-los quando sua distância rotulada é finita.

O conteúdo de um bucket k está representado por content (k). Na operação de seleção de nós são examinados os buckets numerados de 0, 1, 2, ..., até identificar o primeiro bucket não vazio. Assumindo que o bucket k é o primeiro bucket não vazio, então cada nó que pertence a content (k) tem a mínima distância rotulada. Um a um, são eliminados estes nós do bucket e são designados como rótulos permanentes para depois ser examinada sua lista de adjacência de nós A (i) e atualizadas as distâncias dos nós adjacentes. Quando é atualizada a distância rotulada de um nó i de d_1 a d_2, o nó i é transferido de content (d_1) a content (d_2). Na seguinte operação de seleção de nós, resume-se o examinado dos buckets k +1, k + 2, ... com a seleção do seguinte bucket não vazio. A propriedade de que se a distância rotulada que o algoritmo designa como permanente ao inicio de uma iteração, então ao finalizar a iteração, $d(j) \leq d(i) + C$ para cada nó rotulado finito $j \in \overline{S}$, mostra que os buckets numerados 0, 1, 2, ..., k estão sempre não vazios nas iterações subseqüentes e não precisam ser examinados novamente.

A estrutura de dados que armazena os buckets guarda cada conjunto content (k) como uma lista duplamente encadeada, que é um tipo de estrutura de dados que consiste numa seqüência de nós, na que cada nó tem dois links: um aponta para o nó anterior ou para um valor nulo indicando fila vazia, ou para o próximo nó ou para um valor nulo indicando o nó final da lista. Esta estrutura de dados permite o desempenho das operações nos tempos seguintes:

O(1): Verificar os buckets vazios e não vazios, remover um elemento de um bucket e aderir um elemento a um bucket. Assim, com este tipo de estrutura o algoritmo requer um tempo de O (1) para atualizar cada distância, e um tempo O(m) para atualizar todas as distâncias.

A operação crítica nesta implementação é examinar os nC + 1 buckets durante a seleção de nós. Assim o tempo de execução do algoritmo é O(m + nC).

O algoritmo de Dial utiliza nC + 1 buckets.

O algoritmo de Dial examina os buckets seqüencialmente, para identificar os bukets não vazios. Na seguinte iteração, reexamina os buckets começando no lugar donde ficou anteriormente. Uma desvantagem do algoritmo de Dial comparando com a complexidade do algoritmo de Dijkstra O (n^2) é que no caso de que C seja um valor muito alto, será requerido um grande espaço de memória. Além o tempo de execução do algoritmo O (m + nC) não é polinomial. Por exemplo se C = n^4, o tempo de execução

do algoritmo é O (n^5), e si C = 2^n, o algoritmo requer um tempo exponencial no pior caso. Em aplicações nas que C é um valor não muito grande, o algoritmo é uma boa opção, mas para aplicações no pior caso não é muito recomendável.

4.1.2.2 Implementação *heap*-Dijkstra

Na implementação do algoritmo de Dijkstra usando um *heap*, H é o conjunto de nós com distâncias rotuladas temporárias finitas e o key de um nó é a sua distância rotulada. No algoritmo de Dijkstra as operações find-min, delete-min, e insert são executadas num tempo no máximo de n e a operação decrease-key é executada num tempo no máximo de m.

```
algoritmo heap-Dijkstra
começar
S: = Ø;    S̄ : = N;
para cada nó i ∈ N;
d(s): =0; pred(s):=s;
d(i): = ∞; pred(s): = s;
        create-heap ( H );
        insert (s,0, H);
        enquanto H ≠ Ø fazer
        começar
                find-min(i, H);
                delete-min(i, H);
                para cada arco (i, j) ∈ A(i) fazer
                        valor: = d(i) + c_ij;
                        se d(j) > valor então
                            d(j):=valor;
                            p(j):=i;
                            se j não pertence a H então
                            insert (j, valor, H)
                        se não
                        decrease-key(valor, j, H);
                fim
        fim
fim
```

Algoritmo 3 Determinar o caminho mais curto com Dijkstra usando um *heap* binário

A implementação com um *heap* binário requer um tempo O(log n) para realizar as operações insert, decrease-key e delete-min e requer O(1) para as outras operações. A versão de Dijkstra tem um tempo O(m log n). Para um parâmetro $d \geq 2$, as operações insert e decrease-key requerem $O(\log_d n)$, a operação delete-min requer $O(d \log_d n)$ e um tempo O(1) para as outras operações com *heaps*. O algoritmo de Dijkstra com nesta versão tem uma complexidade de $O(m \log_d n + nd \log_d n)$.

A estrutura de dados *heap* Fibonacci requer um tempo O(1) para desenvolver suas operações, menos a operação delete-min que requer um tempo O(log n). A versão de Dijsktra com *heap* Fibonacci requer um tempo O(m + n log n).

A estrutura de Johnson é aplicável a redes com pesos de arcos inteiros. Nesta estrutura requerem-se O(log log C) para realizar cada operação, e a implementação do algoritmo de Dijkstra com esta estrutura requer um tempo de O(m log log C).

A implementação do algoritmo de Dijkstra com radix *heap* requer um tempo de O(m + n log (nC)).

4.2 ALGORITMO DE D'ESOPO PAPE

A implementação do algoritmo que será avaliado neste trabalho é corresponde a versão polinomial desenvolvida por Gallo e Pallotino (1988).

No algoritmo de D'Esopo Pape original o primeiro nó que entra à lista queue S é colocado no fundo da lista S. Para os seguintes nós, estes são inseridos sempre no topo de S se o nó não tiverem estado antes em S, caso contrário são é inseridos ao fundo da lista. A estratégia de seleção de nós neste algoritmo baseia-se em remover um nó e atualizar as distâncias de seus nós adjacentes.

A diferença com a versão de Gallo e Pallotino é que mantêm o conjunto de nós rotulados em dois conjuntos: S_1 e S_2, o primeiro contém os nós rotulados que têm sido examinados pelo menos uma vez e S_2 contém os nós que não foram examinados $S_1 \subseteq S$ e $S_2 \subseteq N - \{S\}$.

O seguinte nó para ser examinado é escolhido de S_1 mas se este conjunto estiver vazio o nó é escolhido de S_2 e este nó é aderido a S.

O conjunto S_1 é chamado o conjunto de alta prioridade e o conjunto S_2 é conhecido como conjunto de baixa prioridade.

O algoritmo de D'Esopo Pape mantém S_1 como uma pilha LIFO e S_2 como uma fila FIFO.

Inicialmente a pilha está vazia e a fila contém o nó origem s. O seguinte nó a ser examinado é removido do topo da pilha se esta não estiver vazia, se não se faz isto iniciando na parte final da fila.

Se existir um nó rotulado que foi examinado com anterioridade, este é colocado no topo da pilha ou da cauda da fila segundo o caso. O algoritmo termina quando a pilha e a fila estão vazios.

O algoritmo de D'Esopo Pape tem uma complexidade exponencial de $O(n2^n)$.

O algoritmo de Gallo e Pallotino mantém S_1 e S_2 usando filas FIFO, Q_1 e Q_2. O seguinte nó a ser examinado é removido da cabeça de Q_1 se a fila estiver não vazia ou da cabeça de Q_2 no caso contrário. Um nó que foi rotulado anteriormente é aderido à parte final de Q_1 ou à cauda de Q_2. O algoritmo termina quando as duas filas estão vazias.

```
começar
d(s): = 0; pred (s): = s;
d(j): = ∞;  pred(j):= nil; para cada j ∈ N-{s};
S: = {s};
enquanto LISTA ≠ Ø fazer
        começar
        remover um elemento i do topo de S;
        atualizar (i)
        para cada j tal que d(j) > d(i) + cij
        inserir j em S e
        d(j):= d(i) + cij; p(j):=i;
        se j não pertence a S então
        inserir ao fundo de S
fim;
fim.
```

Algoritmo 4 Determinar o caminho mais curto D'Esopo Pape

4.3 ALGORITMOS *SMALL LABEL FIRST* (SLF)

4.3.1 ALGORITMO SLF

Bertsekas (1993) propus uma nova estratégia de inserção queue, denominada *Small Label First* (SLF).

Quando um nó j entra na queue S, a distância rotulada d_j é comparada com a distância rotulada d_i do topo de S. Se $d_j \leq d_i$, o nó j é colocado no topo de S, se não j é colocado ao fundo de S.

```
começar
d(s): = 0; pred (s): = s;
d(j): = ∞;  pred(j):= nil; para cada j ∈ N-{s};
S: = {s};
enquanto LISTA ≠ Ø fazer
        começar
        para cada nó j que ingressa à lista S;
        se  d(j) > d(i) + cij
        inserir j no fundo de S
        se não no topo de S
        d(j):= d(i) + cij; p(j):=i;
        fim;
fim.
```

Algoritmo 5 Determinar o caminho mais curto Small Label First

Chen e Powell (1996) mostraram que o algoritmo da estratégia SLF tem uma complexidade exponencial no pior caso.

4.3.2 ALGORITMO SLF-*THRESHOLD*

Glover, Glover e Klingman (1986), propuseram um método denominado *Threshold* .

Bertsekas (1993) propus um algoritmo que combina os dois algoritmos: SLF e *Threshold.* O algoritmo SLF já foi descrito. Apresenta-se a seguir a descrição do algoritmo *Threshold.*

A lista de candidatos S é dividida em dois conjuntos S_1 e S_2 usando um parâmetro umbral s, tal que os nós que pertencem a S_1 não podem conter rótulos maiores do que s, enquanto S_2 contém os nós restantes. A cada iteração, o nó é removido de S_1 e algum nó j será aderido à cauda de S_1 ou S_2, dependendo se $d(j) \leq s$, ou se $d(j) > s$, respectivamente. Quando S_1 começa vazio o umbral s é ajustado e as filas S_1 e S_2, são recalculadas, mas deve-se considerar que S_1 pode conter nós com rótulos cujo valor não é maior do que o novo umbral.

Com a descrição do algoritmo feita até agora, pode-se apreciar uma similaridade com o algoritmo de Dijkstra, considerando que s é a mínima distância rotulada, isto no caso de trabalhar com pesos de arcos negativos, e assim a complexidade é de O(mn). Para o outro caso ainda não se tem um valor definido da complexidade do algoritmo e esta depende de s, mas as pesquisas realizadas até o momento sugerem uma complexidade polinomial.

A versão combinada de Bertsekas (1993) designa rótulos permanentes aos nós de S_1 (threshold) e em S_1 aplica a estratégia SLF já descrita, ou seja, quando um nó j entra na fila Q, a distância rotulada d_j é comparada com a distância rotulada d_i do topo da cabeça da fila Q. Se $d_j \leq d_i$, o nó j é colocado na cabeça de Q, se não j é colocado na cauda de Q. um nó j que entra em S_1 é aderido ao topo ou ao fundo de S_1 dependendo se $d(j) \leq d(i)$ ou $d(j) > d(i)$, i é o nó que está no topo de S_1. A transferência de nós de S_1 a S_2 realiza-se com os nós que não excedem o umbral, e quando S_1 começa vazio, os nós de S_2 são examinados seqüencialmente do primeiro até o último, e se j satisfaz a condição desejada para entrar em S_1, este é inserido no topo ou no fundo de S_1 dependendo se $d(j) \leq d(i)$ ou $d(j) > d(i)$, com i o nó que está no topo de S_1. Quando um nó j entra em S_2 este é aderido com o mesmo critério utilizado em S_1

.

```
começar
d(s): = 0; pred (s): = s;
d(j): = ∞;  pred(j):= nil; para cada j ∈ N-{s};
S1: = Ø, i elemento de S1, d(i) ≠ ∞
S2:= {s}; i elemento de S2, d(i) = ∞
Escolher i da cabeça de S1, se S1 ≠ Ø
Se não i da cabeça de S2
Para cada j se d(j) > d(i) + cij
Se d(j) < s então j é transferido a S2
Se d(j) > d(i), j é colocado no fundo
        se não no topo de S
        d(j):= d(i) + cij; p(j):=i;
        fim;
fim.
```

Algoritmo 6 Determinar o caminho mais curto Small Label First Threshold

4.4 IMPLEMENTAÇÃO E TESTES COMPUTACIONAIS

4.4.1 PROBLEMAS DE TESTE

As redes de teste pertencem à família topológica das redes com grades. Uma rede retangular X *x* Y é uma coleção de nós arranjados em X filas paralelas que contém Y nós cada uma. Os nós de este tipo de rede correspondem a pontos no plano bidimensional. Ver Figura 20.

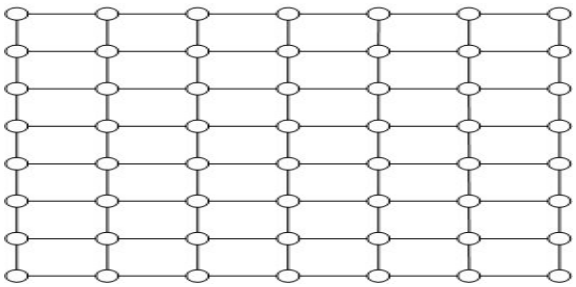

Figura 20 Rede com grades

Uma rede de grade de X *x* Y tem X *x* Y nós e os valores dos pesos dos arcos são gerados aleatoriamente com uma distribuição probabilística uniforme, com valores inteiros.

Algumas das redes da família grade são: de grade quadrada onde X = Y GridSSquare, de grade largo GridSWide com X = 16 e Y variável, de grade longo GridSLong com Y = 16 e X variável, e grade de trânsito GridTransit .

Os problemas foram gerados com uma versão modificada do gerador GRIDGEN de Bertsekas (1992).

Os nós são arranjados em grades planares com origem o nó 1. Cada para de nós adjacentes de grade é conectado em ambas direções.

O número de nós, N é: N = X x Y = 8000.

Número de nós: A = 6N

Para dar maior complexidade às redes foram acrescentados 1000 arcos adicionais a cada rede.

Os pesos dos arcos foram designados com valores nas faixas 1 – 10, 1 – 100, 1 – 1000, seguindo uma distribuição uniforme.

A máxima capacidade dos arcos é de 1000 unidades.

As redes geradas para a avaliação dos algoritmos deste trabalho têm as seguintes características:

X x Y = 8000, com os diferentes produtos 10 x 800, 20 x 400, 32 x 250, 50 x 160, 80 x 100.

Arcos = 48000

Arcos adicionais = 1000

Arcos totais: 49000

Mínimo peso, Máximo peso: 1 – 10, 1 – 100, 1 – 1000.

O nó origem em todos os casos é o nó 34 e o nó de destino o nó 8000.

4.4.2 IMPLEMENTAÇÃO

A implementação dos algoritmos foi realizada num computador pessoal com processador Mobile AMD Sempron (tm) 795 MHz, 512 MB de RAM e trabalhou-se

com o compilador e editor de FORTRAN 77, FORCE 2.0. Os algoritmos avaliados neste trabalho são:

- Dijkstra com *heap*: DIJK-H
- D'Esopo Pape: DEP
- *Small Label First*: SLF
- *Small Label First – Threshold*: SLFT

4.4.3 RESULTADOS DOS TESTES COMPUTACIONAIS

Para obter os dados que se apresentam na Tabela 4, realizaram-se cinco execuções de cada algoritmo utilizando os mesmos parâmetros para cada rede e se obteve a média.

Rede	X x Y	Faixa dos pesos dos arcos	Tempo milisegundos algoritmo DIJK-H	Tempo milisegundos algoritmo DEP	Tempo milisegundos algoritmo SLF	Tempo milisegundos algoritmo SLFT
1	10 X 800	1 - 10	1390	1516	1250	1391
2	10 X 800	1 - 100	906	1641	953	1047
3	10 X 800	1 - 1000	1031	1046	1250	953
4	20 X 400	1 - 10	890	1578	1468	1093
5	20 X 400	1 - 100	1250	1546	1343	781
6	20 X 400	1 - 1000	1016	1187	1046	1437
7	32 X 250	1 - 10	1110	1891	1188	1922
8	32 X 250	1 - 100	922	1015	1140	875
9	32 X 250	1 - 1000	813	1062	1078	906
10	50 X 160	1 - 10	1235	1062	875	1016
11	50 X 160	1 - 100	907	985	953	781
12	50 X 160	1 - 1000	984	1110	1000	1234
13	80 X 100	1 - 10	984	985	1125	937
14	80 X 100	1 - 100	969	829	984	781
15	80 X 100	1 - 1000	828	1141	1875	1078
TOTAL			**15235**	**18594**	**17528**	**16232**

Tabela 4 Tempo em milisegundos para determinar o caminho mais curto nos algoritmos de DIJK-H, DEP, SLF e SLFT

Na Tabela 4, as caixas verdes e amarelas indicam o mínimo e o máximo valor obtidos em os testes respectivamente, para cada uma das diferentes combinações realizadas.

A Figura 21 contém um gráfico que representa os tempos em milisegundos obtidos em cada uma das 15 redes testadas.

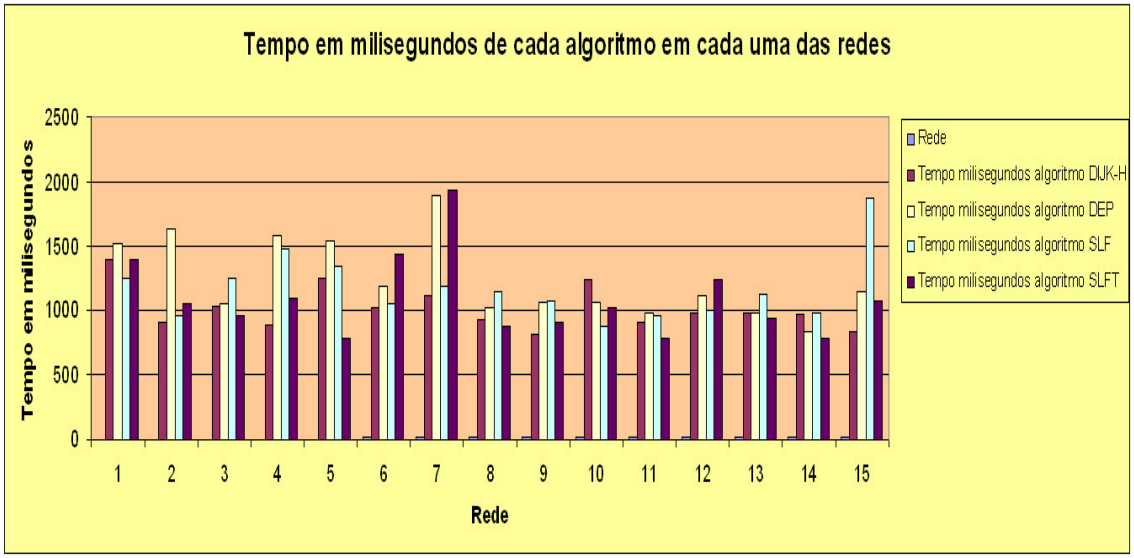

Figura 21 Tempo em milisegundos de cada algoritmo em cada um dos testes

Para as redes testadas em este trabalho, o algoritmo de DIJK-H apresenta o menor tempo de execução total 15235 milisegundos, seguido por SLFT com 16232 milisegundos, SLF com 17528 milisegundos e DEP com 18594 milisegundos.

Na análise realizada a variante do algoritmo de Dijkstra apresentou melhor desempenho com respeito aos outros algoritmos testados. Analisando todas as provas realizadas, este algoritmo apresentou o melhor desempenho em 47% das provas e o pior desempenho em 7% de todos os testes. No algoritmo de DEP, obtiveram-se os piores tempos em 33% dos testes e 0% de melhor tempo. O algoritmo SLF apresentou os melhores tempos em 13% dos testes e pior tempo em 40%. Finalmente o algoritmo de SLFT apresentou no 40% dos testes o melhor tempo e o pior em 20%.

Na Tabela 5 apresentam-se os valores mínimo, máximo, a média e o desvio padrão estándar dos dados obtidos. O mínimo e o máximo tempo registraram-se no algoritmo SLFT. No algoritmo de DIJK-H, pode-se observar que seu máximo tempo de 1390 milisegundos, é o menor tempo comparando com os tempos máximos dos outros

algoritmos nos que obtiveram-se 1891, 1875 e 1922 milisegundos para DEP, SLF e SLFT respectivamente. Além no algoritmo DIJK-H o valor do desvio padrão é menor, o que garante maior uniformidade nos dados.

Algoritmo	Mínimo tempo (milisegundos)	Máximo tempo (milisegundos)	Média	Desvio padrão
DIJK-H	813	1390	1016	165
DEP	829	1891	1240	310
SLF	875	1875	1169	254
SLFT	781	1922	1082	309

Tabela 5 Resumo dos tempos obtidos nos testes de cada algoritmo

Na Tabela 6 se da uma medida geral do desempenho de cada um dos programas. Os pontos que cada um dos programas obteve estão na escala de 0 – 10 e é relativo ao programa mais rápido desta análise, o qual tem a máxima qualificação que é 10 e para os outros o valor obtido de 10*min/t, com min o menor tempo obtido e t o tempo total obtido pelo programa respectivo.

ALGORIMTO	PONTOS
DIJK-H	10
DEP	8.19
SLF	8.69
SLFT	9.39

Tabela 6 Pontos obtidos por cada algoritmo avaliado

Dos testes realizados neste trabalho conclui-se que o algoritmo de DIJK-H tem o melhor desempenho, seguido por SLFT, SLF e DEP.

Centrando a análise aos algoritmos do método correção de rotulação, em os algoritmos avaliados os algoritmos da estratégia SLF apresentaram os melhores desempenhos, SLFT e SLFT em essa ordem. O algoritmo de DEP apresentou o pior desempenho

Uma observação importante deste estudo é que todos os algoritmos foram testados no mesmo computador e com o mesmo compilador. Devido às características do computador, não foi possível realizar mais provas com redes de menor porte devido a que os tempos são muito pequenos e por tanto difíceis de ser determinados num computador como o utilizado. Em testes prévios obtiveram-se tempos em todos os algoritmos de quase zero, dados com os quais fazer esta análise é difícil.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 CONCLUSÕES

Neste trabalho apresentou-se o problema do caminho mais curto, principais conceitos, formulação matemática, aplicações, e algoritmos do caminho mais curto do método de rotulação os quais se subdividem em algoritmos do método de rotulação permanente e de correção de rótulos.

O problema do caminho mais curto consiste em determinar o melhor caminho para unir dois pontos de uma rede, com o objetivo de realizar uma atividade determinada da melhor forma, com o menor custo possível, no menor tempo, etc. Pela freqüência com que na prática se apresentam problemas que podem ser resolvidos com uma formulação de um problema do caminho mais curto e pela eficiência e rapidez com que se podem obter respostas a situações complexas, este tema é de grande interesse para pesquisadores e profissionais, que procuram algoritmos cada vez mais rápidos e eficientes.

As aplicações do problema do caminho mais curto são variadas, em transporte, comunicações, conexão de redes, programação de rota crítica PERT, elaboração de projetos, substituição de equipamentos, gestão de fluxo de caixa, entre outras atividades que precisem ser realizadas da forma mais econômica e rápida possível.

Para resolver o problema do caminho mais curto, existe uma ampla variedade de algoritmos, o que dificulta definir com certeza qual é o melhor deles, além que o desempenho de cada um dos algoritmos depende da estrutura topológica do conjunto de dados. Esta situação faz com que a tarefa de escolher um algoritmo seja mais complexa ainda, pois também deverão ser consideradas as características dos dados, e os resultados que se obtenham servirão para o conjunto de dados analisados e nas condições nas que a análise foi realizada.

Dentro dos algoritmos para achar o caminho mais curto, existem os algoritmos do método de rotulação que designam distâncias rotuladas aos nós em cada etapa, estas distâncias estimam o caminho mais curto e são atualizadas em cada etapa até determinar o caminho mais curto. Os algoritmos do método de rotulação são de dois tipos: rotulação permanente e correção de rotulação.

Os algoritmos do método de rotulação permanente designam uma distância como permanente em cada iteração. Em contraste os algoritmos do método de correção de rotulação consideram todas as distâncias como temporárias, e é na etapa final quando as designa como permanentes.

Os algoritmos de correção de rotulação são mais gerais na sua aplicação porque servem para resolver todas as classes de problemas, incluindo com valores de pesos negativos.

Analisando a complexidade de diferentes algoritmos dos dois tipos de métodos, pode-se concluir que os algoritmos de correção permanente são mais eficientes, porque sua complexidade é polinomial.

Existem alguns trabalhos de avaliação de algoritmos de busca do caminho mais curto. Estes podem conter avaliações teóricas no pior caso de algoritmos já existentes, ou fazer melhoras às versões originais e posteriormente comparar o algoritmo original e a nova versão, e os trabalhos de avaliação feitos com dados reais.

Para estabelecer vantagens de uso em termos de rapidez de resposta avaliaram-se quatro algoritmos, divididos assim: Dijkstra com *heap* (rotulação permanente) e D'Esopo Pape, *Small-Label-First* e *Small- Label- First-Thereshold* (correção de rotulação).

Os algoritmos mencionados foram implementados com o compilador de FORTRAN 77, FORCE 2.0. Realizou-se o teste com redes da família topológica com grades, usando o gerador GRIDGEN. Testaram-se 15 redes retangulares de 10 X 800, 20 X 400, 32 X 250, 50 X 160, 80 X 100. As redes obtidas têm 9000 nós e 48000 arcos. A faixa dos pesos dos arcos é de 1 – 10, 1 – 100 e de 1 – 1000 cada rede e sua designação é feita com uma distribuição de probabilidade uniforme. Pelas características do gerador, também é necessário acrescentar arcos adicionais, por tanto foram acrescentados 1000 arcos adicionais.

Os resultados obtidos mostram que o algoritmo que oferece a resposta em menor tempo é o algoritmo de DIJK-H, seguido por SLFT, SLF e DEP.

Os tempos obtidos com o algoritmo de DIJK-H são 6,54% mas rápidos com respeito aos tempos obtidos com o algoritmo SLFT, 15% mais rápido com respeito ao algoritmo SLF e 22% mais rápido que o algoritmo de DEP.

Deve considerar-se que os resultados obtidos neste e em outros trabalhos de avaliação são específicos para os dados analisados, para o computador em que foi realizado o teste e para o compilador utilizado.

5.2 TRABALHOS FUTUROS

Apesar de existir uma grande variedade de algoritmos para determinar o caminho mais curto, realizar melhoras nos algoritmos já existentes ou desenvolver novas estratégias para a solução do problema é de interesse na comunidade cientifica e especialmente de grande utilidade para os usuários.

Para acrescentar os resultados obtidos neste trabalho recomenda-se realizar testes adicionais considerando outras famílias topológicas como: redes aleatórias, redes acíclicas e de grade cada uma com suas variantes. Pois resultados obtidos em estudos realizados previamente mostram que o comportamento dos algoritmos é diferente dependendo do tipo de rede.

A grande quantidade de algoritmos existentes também faz com que os trabalhos deste tipo sejam limitados no sentido ao número de algoritmos que podem ser testados, pois não é possível testar a totalidade deles. Uma recomendação é testar algoritmos que ainda não foram testados em nenhum dos trabalhos feitos até a atualidade para ter mais possibilidade de escolha.

De outros estudos se sabe que a implementação de Dijkstra com *heaps* é muito eficiente em tempos de execução, de forma que realizar mais pesquisas deste algoritmo pode oferecer resultados interessantes.

Outra recomendação consiste em realizar avaliações de algoritmos do problema do caminho mais curto com redes reais, pois os testes realizados aqui foram feitos com redes geradas. Por exemplo, poderia construir um sistema de otimização de rotas para a cidade do Rio de Janeiro.

Para melhorar a qualidade dos resultados obtidos deveria ser usado um computador adequado que permita determinar tempos muito pequenos e que permita trabalhar com redes de maior porte.

Deveriam ser consideradas otimizações nos algoritmos para melhorar o desempenho dos programas, pois os resultados também dependem das melhoras feitas por compiladores e programadores em suas implementações.

BIBLIOGRAFÍA

AHUJA, R., ORLIN, J., PALLOTINO, S., SCUTELLÀ, M., Dynamic Shortest Paths Minimizing Travel Times and Costs, Department of Industrial & Systems Engineering, University of Florida, 2002.

AHUJA, R., MAGNANTI, T., ORLIN, J., Network Flows Theory: algorithms and applications, Prentice – Hall, 1993.

AHUJA, R., MEHLHORN, K., ORLIN, J., TARJAN, R., Faster Algorithms for the shortest path, 1990.

BAKO, A., On the determination of the shortest path in a network having gains, Computing Centre of the Hungarian Academy of Sciences, Budapest, Hugary, Vol 4, 1: 63 – 68, 1973.

BARDOSSY, M., SHIER, D., Label-Correcting Shortest Path Algorithms Revisited. Perspectives in Operations Research Papers in Honor of Saul Gass' 80th Birthday. December 2006

BERTSEKAS, D. A Simple and Fast Label Correcting Algorithm for Shortest Paths. Networks, 23: 703-709, 1993.

_____. Linear Network Optimization: Algorithms and Codes, M.I.T.Press, Cambridge, MA, 1992.

CHEN, L., POWELL, W., 1997. A note on Bertsekas' Small-Label-First Strategy, Networks, 29: 111 – 116, 1997.

CHERKASSKY, B. V., GOLDBERG A. V., RADZIK, T. Shortest Paths Algorithms: Theory and Experimental Evaluation, Mathematical Programming, 73: 129 – 174, 1996.

CHERKASSKY, B. V., GOLDBERG A. V., Negative – cycle detection algorithms. Proceedings of the 4th European Symposium on Algorithms, Barcelona, Espanha, 1993.

COHEN, E., Shortest-Path Algorithms, Handbook of Applied Optimization, Editado por Pardalos Panos e Resende Mauricio, Oxford University Press, 2002.

CORMEN, T., LEISERSON, C., RIVEST, R., Introduction to Algorithms. MIT Press, 1997.

DENARDO, E., FOX, B., Shortest –Route Methods. 1 Reaching, Punting and Buckets, Oper. Res, 27: 161 – 186, 1979.

DIAL, R., Algorithm 360 Shortest Path Forest with Topological Ordering, Comm. ACM, 12: 632 – 633, 1969.

DIJKSTRA, W., A note on two problems in connection with graphs. Numerische Mathematik., 1: 269 – 271, 1959.

DIVOKY,J., HUNG, M., Performance of shortest path algorithms in network flow problems, Management Science, 36: 661- 674, 1990.

DREYFUS, S, An appraisal of some shortest path algorithms. Operations Research, 17: 395 - 411, 1969.

EPPSTEIN, D., Finding the k Shortest Paths. Tech. Report 94-26, Univ. of California, Irvine, Dept. of Information and Computer Science, 1994, p. 23.

FESTA, P., Shortest Path Algorithms, Handbook of Optimization in telecommunications, editado por Mauricio Resende e Panos Pardalos, Springer Science+Business Media, Inc, p. 185 – 210, 2006.

FESTA, P. New Auction Algorithms for Shortest Path Problems. Tesis PhD, Universidade de Naples, FEDERICO II, Naples, Italy, February, 2000.

FREDMAN, M., TARJAN, R., Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of ACM* **34:**, 596-615 1987.

FRIGIONI, D., MARCHETTI-SPACCAMELA, A., NANNI, U., Fully dynamic algorithms for maintaining shortest paths trees, J. Algorithms, 34: 251 –281, 2000.

GALLO, G., PALLOTINO, S., Shortest paths algorithms, Annals of Operations Research 13: 3-79, 1988.

GLOVER, F., KLINGMAN, D., PHILLIPS N., SCHNEIDER, R., New Polynomial Shortest path algorithms and their computational attributes, Management Science, 31: 1106 – 1128, 1985.

GLOVER, F., GLOVER, R., KLINGMAN, D., The Threshold Shortest Path Algorithm, Networks, 14: 256 – 282, 1986.

GOLDFARB, D., JIN, Z., An O(nm)- time network simplex algorithm for the shortest path problem, Operations Research, 47: 445 – 448, 1999.

GOLDBERG, A., A Practical Shortest Path Algorithm with Linear Expected Time, SIAM J. Computational Vol 37, 5: 1637 – 1655, 2008.

GOLDBERG, A., A simple shortest path algorithm with linear average time. Technical Report STAR – TR- 01- 03, 2001.

GUERREIRO, F., MUSMANNO, R., LACAGNINA, V., PECARELLA, A., A Class of label – correcting Methods for the k Shortest Path Problem, Operations Research, Vol 49, 3: 423 – 429, 2001.

HAN, S., FRANCHETTI F., PUSCHEL, M., Program Generation for the All – Pairs Shortest Path Problem, ACM 1 – 59593 – 264 – X/06/0009, 16 – 20, 2006.

HERSHBERGER, J., MAXEL, M., SURI, S., Finding the k shortest simple paths: A nex algorithm and its implementation. ACM, Transactions on Algorithms 3: 2007.

HILLIER, F. S., LIEBERMAN G. J.. Introdução à pesquisa operacional. 3. Ed. Rio de Janeiro: Campus / São Paulo: Universidade de São Paulo, 1988.

JUNGNICKEL, D., Graphs, Networks and Algorithms. Springer-Verlag Berlin Heidelberg, 1999.

KLUNDER, G., POST, H., The shortest path problem on large-scale real-road networks. Networks Vol. 48, 4: 182 – 194, 2006.

LUX, B., FURTADO, J., Sistema de Otimização de rotas com suporte de software gerenciador de informações geográficas, Universidade de Santa Cruz do Sul, 2001.

MARTINS, E., An algorithm for ranking paths that may contain cycles. European Journal of Operations Research, 18: 123 – 130, 1984.

NARVAEZ, K., SIU, Y., TZENG, H., New dynamic algorithms for shortest path tree computation, ACM, Transactions Networks, 8: 734 – 746, 2000.

PAPE, U., Implementation and efficiency of Moore – Algorithms for the shortest route problem, Mathematical Programming, 7: 212-222, 1974.

ROSEN, K., MICHAELS, J., GROSS, J., GROSSMAN, J., SHIER, D. Handbook of Discrete and Combinatorial Mathematics, CRC Press LLC, 1999.

RUSSY, H., PALACIOS, F., Evaluación computacional de algoritmos de distancia mínima, dissertação de Mestrado, Universidad del Valle, Colombia, 2005.

SERRANO, M., SILVA, M., VENTURA, I., Grafos no secundário, 2007. Disponível em:

SCHRIJVER, A., On the history of combinatorial optimization (till 1960), Handbook of Discrete Optimization, p. 1- 68, 2005.

YEN, J. Finding the k shortest loopless paths in a network. Management Science, 17: 712 – 716, 1971.

ZHAN, B., Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures. Journal of Geographic Information and Decision Analysis, 1: 69-82, 1997.

_____, NOON, C. E. Shortest Path Algorithms: An Evaluation Using Real Road Networks, Transportation Science, 32: 65-73, 1998.

_____, A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths, *Journal of Geographic Information and Decision Analysis*; 4(2): 1-11, 2000.

ANEXOS

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo