Caminhos mínimos com recursos limitados

Joel Silva Uchoa

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre

Programa: Ciência da Computação

Orientador: Prof. Dr. Carlos Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, agosto de 2012

Caminhos mínimos com recursos limitados

Esta dissertação trata-se da versão original
do aluno (Joel Silva Uchoa).

Agradecimentos

Primeiramente, agradeço a Deus pela oportunidade de desenvolver este trabalho e de encontrar pessoas nesta jornada que me fazem crescer tanto intelectualmente como moralmente.

Resumo

Caminhos mínimos com recursos limitados

Palavras-chave: Otimização combinatória, .

Abstract

Resource constrained shortest path problem

Keywords: Combinatorial Optimization, .

Sumário

1

Introdução

O problema de caminhos mínimos (– *shortest path problem*) é um dos problemas fundamentais da computação. Vem sendo estudado com profundidade e há uma grande quantidade de publicações a respeito do mesmo. Inclusive, são conhecidos várias soluções eficientes (algoritmos de tempo polinomial) para o problema. O é frequentemente colocado em prática em uma grande variedade de aplicações em diversas áreas, não somente em computação. Nessas aplicações geralmente se deseja realizar algum tipo de deslocamento ou transporte entre dois ou mais pontos específicos em uma rede. Tal ação deve ser executada de forma ótima em relação a algum critério, por exemplo o menor custo possível, ou o menor gasto de tempo ou o máximo de confiabilidade/segurança.

Conforme essas soluções para o foram sendo apresentadas, novas necessidades foram levantadas e surgiram variações do problema para modelar tais necessidades. Uma dessas variantes, advém do fato de que, na prática, muitas vezes não desejamos apenas o menor custo ou o menor tempo, mas desejamos otimizar uma combinação de diferentes critérios, por exemplo, um caminho que seja rápido e barato. Este é conhecido como o problema de caminhos mínimos multi-objetivo. Como não é possível otimizar sobre todos os critérios de uma só vez, nós escolhemos um dos critérios para representar a função custo, que será minimizada, e para os demais critérios representamos como recursos e definimos os limites que julgamos aceitáveis para o consumo de cada um desses recursos [1]. Esta

[1]Restrições deste tipo, onde temos o consumo de recursos em um orçamento que limita a quantidade disponível destes recursos são chamadas de *knapsack constraints* bordorfer:09.

variação é chamada de problema de caminhos mínimos com restrições por recursos, ou como preferimos chamar, **problema de caminhos mínimos com recursos limitados** (– *resource constrained shortest path problem* [2]), o qual será o objeto de estudo neste trabalho.

A adição de restrições por recursos no , infelizmente torna o problema $\mathcal{NP}$-difícil, mesmo em grafos acíclicos, com restrições sobre um único recurso, e com todos os consumos de recursos positivos. Temos reduções dos famosos problemas $\mathcal{NP}$-difíceis e para o nosso problema.

Em contextos diversos são encontrados problemas de cunho teórico e prático que podem ser formulados como problemas de caminhos mínimos com recursos limitados, o que nos motivou a estudá-lo a fim de desenvolver um trabalho que resumisse informações suficientes para auxiliar pesquisadores ou desenvolvedores que tenham interesse no problema. Nós apresentamos aqui, uma detalhada revisão bibliográfica do , tendo como foco o desenvolvimento de algoritmos exatos para o caso onde possuímos um único recurso e a implementação e comparação dos principais algoritmos conhecidos, observando-os em situações práticas.

1.1 Aplicações

O problema de caminhos mínimos com recursos limitados pode ser aplicado em uma imensa quantidade de problemas práticos. Esta seção vai descrever algumas destas aplicações.

1.1.1 Qualidade de serviço em redes de computadores

A qualidade de serviço (QoS – *quality of service*) é um aspecto importante para as redes de pacotes como um todo e para as redes IP em particular. Tem um valor fundamental para o desempenho de determinadas aplicações fim-a-fim, como vídeo e áudio conferência e transferência de dados.

Existem redes de computadores que estão oferecendo garantias de QoS para diversas aplicações. Estas aplicações possuem diversos requisitos que precisam ser atendidos para o seu bom funcionamento. Como exemplo de requisitos podemos citar largura mínima de banda, tempo máximo de atraso e quantidade máxima de perda de pacotes. O problema em que estamos interessados, ou seja, serviços baseados em QoS, é determinar uma rota que usa os recursos da rede de forma eficiente e satisfaz, ao mesmo tempo, requisitos como qualidade

[2] beasley:89 foi um dos primeiros a chamar o problema desta forma, antes disso era comum utilizar apenas (constrained shortest path problem). A versão com um único recurso pode ser referenciada como (*single*), ou ainda como WCSP (*weight constrained shortest path problem*) dumitrescu:03.

de conexão. Este problema é conhecido como roteamento baseado em QoS aurrecoechea:98.

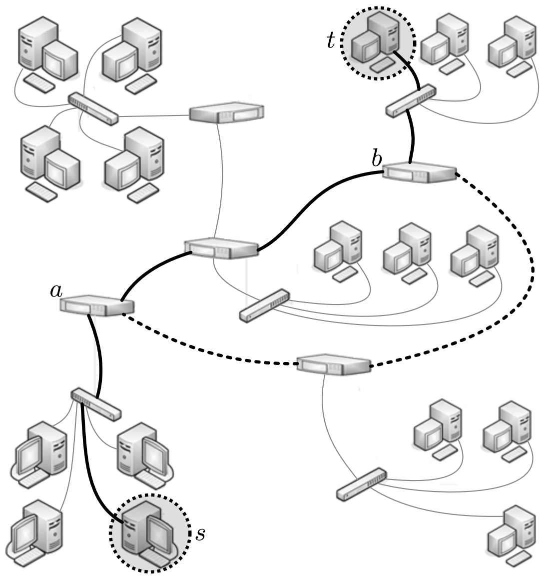


Figura 1.1: *Representação de uma rede de computadores. Os seguimentos pretos e contínuos compõem uma rota entre os computadores s e t. Substituindo-se o trecho entre os roteadores a e b pelo trecho pontilhado, temos uma outra rota que pode ser usada para a comunicação entre s e t. Dependendo das propriedades destas rotas e das necessidades dos usuários, uma ou outra pode ser mais apropriada para o uso.*

Formalizando um pouco melhor o problema, podemos representar nossa rede como um grafo direcionado $G = (V, A)$, onde V é o conjunto de vértices e A é o conjunto de arcos. Cada arco $uv \in A$ é associado com um custo c_{uv} e M pesos não negativos w_{uv}^k, $k = 1, 2, \cdots, M$ que representam a valoração dos requisitos no arco (ambos, pesos e custos são aditivos em um caminho). Dada uma aplicação que exige M requisitos com valoração máxima R^k , $k = 1, 2, \cdots, M$, o problema é encontrar um caminho P da origem s até o destino t que respeita os requisitos e que minimiza o custo total do caminho. Problema este que pode ser modelado da seguinte forma:

$$\text{minimize} \quad \text{c(P)} \quad = \quad \sum_{uv \in P} c_{uv}$$

$$\text{sujeito a} \sum_{uv \in P} w_{uv}^k \leq R^k parak = 1, \ldots, M$$

Este problema é uma aplicação direta do problema de caminhos mínimos com recursos limitados. É comum, neste tipo de aplicação, querermos uma rota passando pelo menor número de vértices possível, neste caso a função de custo possui valor unitário para todos os arcos. É comum também existirem restrições que não são aditivas pelo caminho, mesmo com este tipo de restrição as soluções para o podem ser aplicadas, podemos contorná-las geralmente com algum preprocessamento ou pequenas alterações nos algoritmos.

1.1.2 Roteamento de tráfego de veículos

Quando precisamos nos deslocar de um ponto a outro em uma rede de tráfego veicular é natural nos preocuparmos com uma série de fatores a respeito da rota escolhida para o trajeto. Geralmente desejamos percorrer o caminho no menor tempo possível dentro de determinadas restrições, como consumo de combustível, gastos com pedágio e distância máxima percorrida. Outras restrições relevantes, são por exemplo, segurança e possibilidade de congestionamentos (essas características são mais subjetivas, geralmente baseadas em dados estatísticos).

Dentro deste contexto, jahn:05 propõe uma aplicação interessante que usa o como subproblema. Nesta aplicação, não se deseja minimizar apenas o tempo de percurso de cada usuário individualmente, o objetivo é minimizar o tempo total de percurso do sistema como um todo. Com a função objetivo da forma que foi descrito acima, é possível que para atingir a eficiência global, alguns usuários precisem realizar caminhos muito piores do que realizariam caso utilizassem uma estratégia egoísta. Assim, em um sistema de apoio à decisão, poderia acontecer dos usuários não seguirem as recomendações apresentadas. Pensando nisto, jahn:05 aplica uma restrição para que o caminho de cada usuário não seja demasiadamente pior do que o melhor caminho possível. Desta forma, é possível obter uma solução aceitável para cada usuário, que objetiva aumentar a eficiência global do sistema [3].

Hoje em dia existem sistemas de informação e orientação projetados para auxiliar motoristas a tomarem decisões de rota. Vamos idealizar um sistema que pode fornecer informações como congestionamentos, bloqueios ou acidentes, e dar recomendações baseado nestes dados. O motorista cadastraria suas preferências, entraria com o seu destino e o sistema calcularia a rota baseado nos mapas digitais, preferências do usuário, especificidades do veículo (dimensões, quantidade de combustível disponível e consumo, por exemplo) e nas condições atuais do trânsito. Sistemas deste tipo poderiam ter o seguinte cenário, as condições de tráfego seriam obtidas através de sensores e transmitidas a uma central de controle de tráfego, que por sua vez, receberia (talvez do computador de bordo do carro) as preferências dos usuários, as características dos veículos, as posições atuais e seus destinos, podendo assim fazer certas distribuições de rotas aos motoristas.

Descrevemos todo um conjunto de restrições complexo e detalhista, porém, na nossa formulação levaremos em consideração apenas os níveis de congestionamento e um limitante superior para a degradação de um caminho em relação ao melhor caminho (consideramos um caminho viável apenas se este é mais demorado que o caminho mais rápido até um certo limite).

[3]Soluções onde se atribui o caminho mais rápido para cada usuário sob as condições correntes, são chamadas de solução *user optimal* ou *user equilibrium*. Soluções onde minimiza-se o tempo total do sistema, são chamadas de *system optimal*.

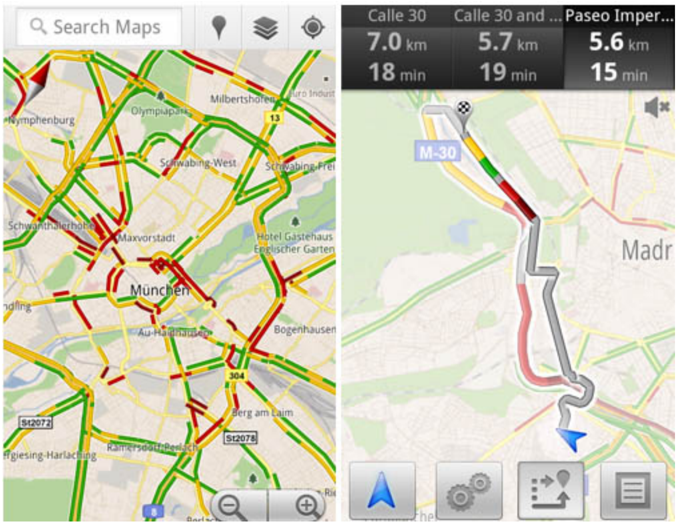


Figura 1.2: *Exemplo de um sistema de orientação para motoristas. No lado esquerdo podemos ver para uma determinada região o tráfego em cada trecho (vermelho, amarelo e verde representam respectivamente tráfego pesado, médio e leve). No lado direito temos a representação de uma rota, começando no triângulo azul (posição atual do veículo) e terminando na marcação com um circulo quadriculado xadrez.(Figura retirada de Google Mobile Blog – http://googlemobile.blogspot.com.br/2011/07/live-traffic-information-for-13.html)*

Representamos nossa rede rodoviária por um grafo direcionado $G = (V, A)$ com dois atributos em cada arco $uv \in A$: $\tau_{uv} \geq 0$ que representa uma estimativa do tempo de travessia quando não há congestionamento; e uma função $l_{uv}(x_{uv})$ (x é um fluxo na rede, x_{uv} é a parte deste fluxo correspondente ao arco uv) que computa uma estimativa do tempo de travessia do arco uv considerando o fluxo dado[4].

Nós modelamos os veículos que possuem a mesma origem e destino como um par $k = (s, t)$, definimos K como o conjunto de todos estes pares. Podemos representar cada $k \in K$ como (s_k, t_k). Definimos a demanda $d_k > 0$ para cada $k \in K$ como sendo a quantidade de fluxo a ser roteada através de k (veículos por unidade de tempo). Denotamos todos os caminhos para o par k por $\mathcal{P}_k = \{P \mid P \text{ é um caminho de } s_k \text{ até } t_k\}$, e o conjunto completo de caminhos por $\mathcal{P} = \cup_{k \in K} \mathcal{P}_k$. Para um caminho $P \in \mathcal{P}$, o tempo para percorrermos P, dado um fluxo x representando o estado atual da rede, é $l_P(x) = \sum_{uv \in P} l_{uv}(x_{uv})$, o tempo estimado de percurso sem considerar congestionamento é $\tau_P = \sum_{uv \in P} \tau_{uv}$.

Definimos um fator de tolerância $\varphi \geq 1$. Através deste fator, assumimos que para um caminho $P \in \mathcal{P}_k$ ser viável, $\tau_P \leq \varphi T_k$, onde $T_k = \min_{P \in \mathcal{P}_k} \tau_P$ é o menor tempo possível para se partir de s_k e chegar a t_k desconsiderando-se o fluxo na rede. Assim podemos denotar $\mathcal{P}_k^\varphi$ como o conjunto de todos os caminhos viáveis que partem de s_k e terminam em t_k, e $\mathcal{P}^\varphi = \cup_{k \in K} \mathcal{P}_k^\varphi$ como sendo o conjunto de todos os caminhos viáveis para os pares em K.

O sistema ótimo com restrições (CSO – *constrained system optimum*) proposto, pode ser modelado como o seguinte fluxo múltiplo de custo mínimo (*min-cost multi-commodity*

[4] TODO – Informações sobre a função l.

flow):

$$\text{minimize} \quad \mathrm{C(x)} \quad = \quad \sum_{uv \in A} l_{uv}(x_{uv}) \cdot x_{uv}$$

$$\text{sujeito a} \sum_{P \in \mathcal{P}_k^{\varphi}} x_P = d_k para k \in K$$

$$\sum_{P \in \mathcal{P}^{\varphi} | uv \in P} x_P = x_{uv} para uv \in A$$

$$x_P \geq 0 para P \in \mathcal{P}^{\varphi}$$

jahn:05 usa um algoritmo de geração de colunas para resolver o problema CSO (para uma descrição do método ver frank:56 e leblanc:85). Neste algoritmo, surge como sub-problema computar um caminho ótimo em $\mathcal{P}_k^{\varphi}$ que é precisamente o : Neste caso, cada arco $uv \in A$ tem dois parâmetros, o tempo de travessia l_{uv} e o comprimento τ_{uv}. Dado um par (s, t), o objetivo é computar um caminho mais rápido de s a t cujo tamanho não excede a um dado limite T. Ou seja, o problema é

$$\min\{l_P \mid P \text{ é uma caminho de } s \text{ até } t \text{ e } \tau_p \leq T\},$$

onde $l_P = \sum_{uv \in P} l_a$ e $\tau_P = \sum_{uv \in P} \tau_{uv}$.

1.1.3 Compressão de imagens (aproximação de curvas)

Uma curva/função é linear por partes (*piecewise linear curve*) se pudermos subdividi-la em intervalos que são lineares. Este tipo de curva/função é frequentemente usado para aproximar funções complexas ou objetos geométricos.

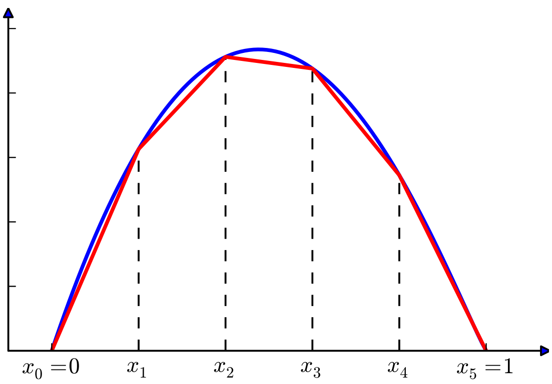


Figura 1.3: *Exemplo de uma função linear por partes. A função em azul é uma função não linear, e a função em vermelho é uma aproximação da primeira que atende a nossa definição de curva linear por partes.*

O uso dessas curvas é muito comum em áreas como computação gráfica, programação matemática, processamento de imagens e cartografia. Curvas lineares por parte são populares porque são fácil de se criar e manipular, além de fornecer, em geral, uma aproximação suficientemente boa para os problemas estudados.

Aplicações nas áreas citadas no parágrafo anterior, frequentemente incluem uma enorme quantidade de dados (as curvas em geral possuem uma grande quantidade de partes ou pontos de quebra). Isto causa dificuldades, por exemplo, com o espaço de armazenamento, taxa de transmissão ou tempo gasto para renderizar a curva em um dispositivo gráfico. Isto naturalmente nos faz pensar no problema de redução/compressão de dados, onde nós queremos determinar uma nova curva linear por partes que é tão aproximada quando possível da original, mas tem um número menor de pontos de quebra.

dahl:96, nygaard:98 estudaram este problema e mostraram como ele poderia ser modelado como um problema de caminhos mínimos com recursos limitados: Os pontos de quebra $v_1, v_2, \cdots, v_{n-1}, v_n$ são os vértices do grafo $G = (V, A)$ e para todo para $1 \leq i \leq j \leq n$ nós temos um arco $v_i v_j$. O custo de um arco c_{uv} é o erro introduzido na aproximação por tomar o "atalho" indo direto de u para v ao invés da curva original[5]. O recurso de uma aresta r_{uv} é 1 para $uv \in A$. Agora, nós podemos computar a melhor aproximação usando no máximo k pontos de quebra [6].

1.2 Objetivos

Os principais objetivos deste trabalho, de forma sucinta são:

- levantar um conjunto de referências bibliográficas relevantes, cobrindo o máximo possível de variações e aplicações do problema;
- apresentar o e suas diversas abordagens com uma notação padronizada;
- implementar um subconjunto dos principais algoritmos conhecidos;
- avaliar o desempenho prático dos algoritmos implementados.

1.3 Preliminares

Para o perfeito entendimento do conteúdo desde trabalho devemos salientar a necessidade de conhecimento prévio em alguns assuntos que enumeraremos a seguir. A maioria dos estudantes de computação deve estar familiarizado com os conceitos, mas faremos indicações de publicações que definem e usam notações próximas da que estamos utilizando.

[5]Existem diversos tipos de métricas que podem ser usadas para calcular este erro.

[6]Alternativamente, nós podemos limitar o erro de aproximação e computar o menor número de pontos de quebra.

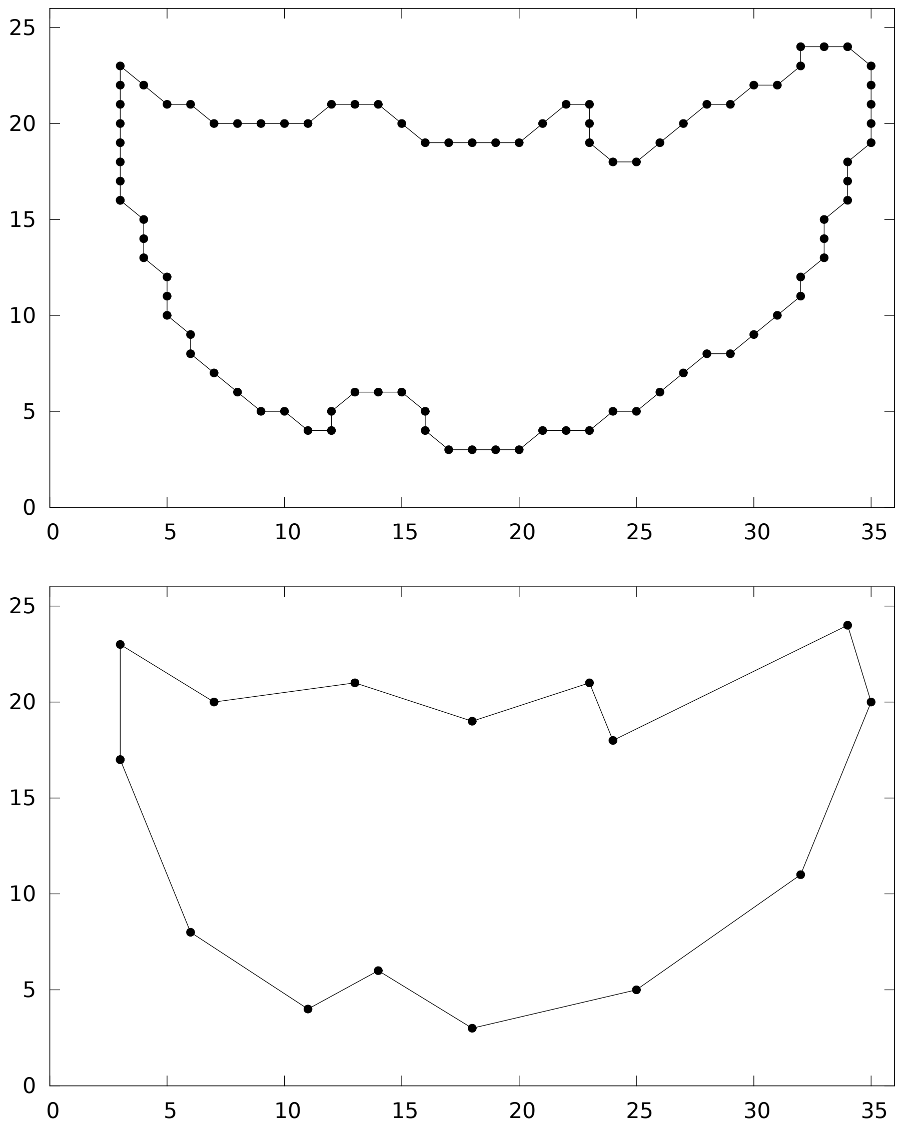


Figura 1.4: *Exemplo de uma curva e sua aproximação. A curva original possui* 90 *pontos, enquanto a sua aproximação possui* 15 *pontos.*

Um conhecimento prévio dos seguintes assuntos é recomendado:

- Teoria dos Grafos
- Algoritmos em Grafos
- Fluxo em Redes
- Programação Linear
- Programação Inteira
- Relaxação Lagrangiana
- Algoritmos de Aproximação

- Complexidade Computacional

No que se trata da parte de teoria dos grafos, fluxo em redes e algoritmos em grafos, seguimos de perto a nomenclatura e conceitos definidos em pf:fluxos. Um livro muito completo em relação aos conceitos de programação linear e relaxação que fazemos questão de indicar é o wolsey1998integer. Carvalhoetal01 fala sobre algoritmos de aproximação, e é uma ótima referencia sobre o assunto. Por fim temos o livro clrs:introalg-2001 que pode ser usado para o estudo de complexidade computacional.

1.4 ORGANIZAÇÃO

O trabalho é organizado da seguinte forma.

O Capítulo 1 – Introdu, apresenta uma visão geral do problema e da dissertação. Apresentamos textualmente uma definição do , alem de citar informações sobre complexidade e descrever alguns problemas interessantes aos quais o pode ser aplicado. Enumeramos também os tópicos relevantes para o entendimento do nosso trabalho fazendo referência a textos que podem ajudar os leitores a adquirir tais conhecimentos. Discorremos um pouco também a respeito do foco e objetivos deste trabalho.

O Capítulo 2 – Caminhos mmos (sem restris), trás uma descrição do problema de caminhos mínimos clássicos, problema este que deu origem ao . Alem da descrição apresentamos algoritmos eficientes para o problema, e também definimos alguns conceitos importantes, usados no decorrer da dissertação.

No Capítulo 3 – Caminhos mmos com recursos limitados, definimos formalmente o problema de caminhos mínimos com recursos limitados, que é o foco deste trabalho. Apresentamos um breve histórico listando as principais soluções conhecidas para o problema. Expomos também uma prova que mostra que o é um problema $\mathcal{NP}$-difícil. Por fim descrevemos alguns algoritmos relevantes que despertaram nosso interesse.

O Capítulo 4 – Experimentos, expõe estatísticas e percepções a respeito dos experimentos realizados com os algoritmos implementados.

2

Caminhos mmos (sem restris)

Como dito anteriormente, o problema de caminhos mmos com recursos limitados a generaliza do problema de caminho mmo clico (– *shortest-path problem*). Tal problema consiste em encontrar um caminho de um vice origem at vice destino com menor custo em um grafo direcionado. A importia do se deve por suas inmeras aplicas e generalizas.

Dada a sua relevia, vamos dedicar este caplo ao . Na primeira parte descrevemos os principais conceitos relacionados ao problema, em seguida definimos o problema formalmente e por fim fazemos uma exposi de alguns algoritmos que resolvem o problema. Este caplo foi desenvolvido baseado em paulo:97, fabio:09 e shigueo:02.

2.1 Definis bcas

Podemos definir uma fun custo c em um grafo $G = (V, A)$ como sendo uma fun sobre A, onde, para todo $uv \in A$, $c(uv)$ alor de c em uv (o custo do arco uv).

Seja um caminho P e uma fun custo c em um grafo $G = (V, A)$, definimos o custo do caminho P como $c(P) = \sum_{uv \in P} c(uv)$ a soma dos custos de todos os arcos em P.

Dizemos ainda que um caminho P tem custo mmo se, seja e o inicio e o fim de P respectivamente, vale que $c(P) \leq c(Q)$ para todo caminho Q que comem e termina em .

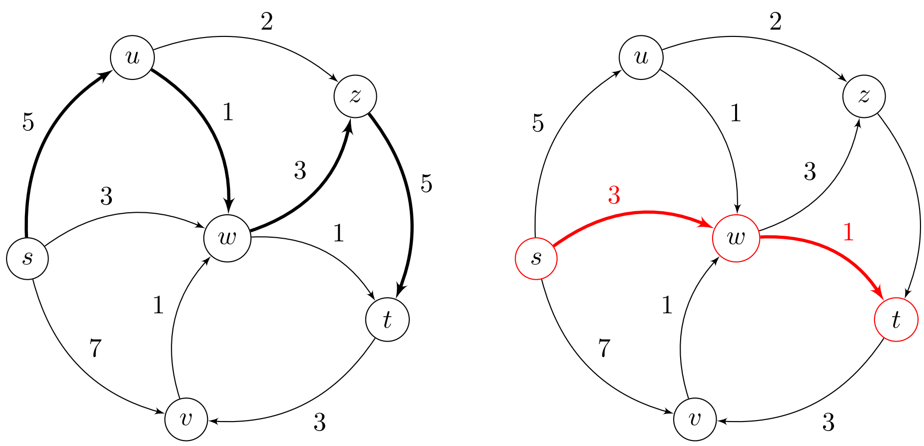


Figura 2.1: *Exemplo de um grafo com uma fun custo sobre os arcos. No lado esquerdo temos um caminho* $(,u,w,z,)$ *com custo igual a 14. direita temos o caminho* $(,w,)$ *em vermelho, que caminho de custo mmo de .*

Definimos a distia de um vice a um vice como o custo de um menor caminho de a . Representamos a distia de a por ,. A distia de a , na figura 2.1, .

Vamos denotar por $C = \max\{c(uv) : uv \in A\}$ o maior custo de um arco. No grafo representado na figura 2.1 temos que o maior custo =7.

2.2 Defini formal do problema

Com as definis que acabamos de apresentar podemos fazer uma defini formal para o problema do caminho mmo, denotado por :

Problema 1. $(G,c,,)$ *Como partros do problema sados*

- *um grafo direcionado* $G = (V, A)$*,*
- *uma fun custo* c *sobre* G*,*
- *um vice origem e*
- *um vice destino .*

O problema consiste em encontrar um caminho de custo mmo de a .

Na literatura essa vers conhecida como *single-pair shortest path problem* ou ainda como *single-source, single-sink shortest-path problem* zhu:05.

2.3 Funs potenciais

Vamos definir o seguinte programa linear, que chamamos de primal, e a relaxa do problema do caminho mmo: encontrar um vetor x indexado por A que minimize $\sum_{uv \in A} c(uv)x_{uv}$

sob as restris $\sum_{vw \in A} x_{vw} - \sum_{uv \in A} x_{uv} = \begin{cases} 1 & parav = \\ 0 & paratodov \in V \setminus \{,\} \\ -1 & parav = \end{cases}$

$x_{uv} \geq 0\ paratodouv \in A.$

O vetor caracterico de qualquer caminho de a a solu vil do problema primal. Vamos definir agora, o respectivo problema dual, que consiste em encontrar um vetor y indexado por V que maximize y()-y()

sob as restris y(v) - y(u) $\leq c(uv)$para todo $uv \in A$.

Uma **fun-potencial** a fun sobre V que associa a cada vice um valor. Se y a fun potencial e c a fun custo, ent dizemos que y c-potencial se

$$y(v) - y(u) \leq c(uv) \text{ para cada arco } uv \in A.$$

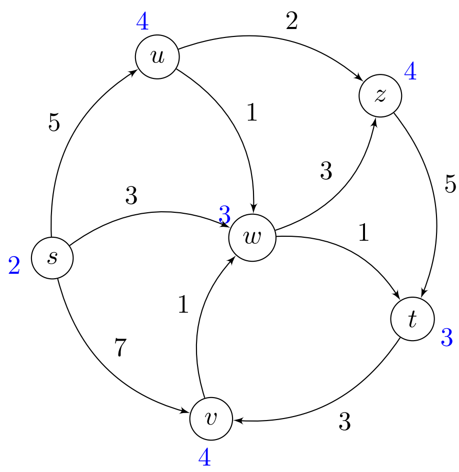


Figura 2.2: *Grafo com uma fun custo c sobre os arcos e um c-potencial associado aos vices em azul.*

nteressante que um algoritmo que resolve o , apresente, juntamente com a solu, certificados como garantia que sua solu rreta. O primeiro seria um certificado que garanta que os caminhos fornecidos smos (certificado de otimalidade), este pode ser extra a partir de uma particulariza do lema da dualidade de programa linear pf:proglin. O segundo seria o certificado de ncessibilidade, que pode ser apresentado da seguinte forma: se n possl atingir um vice a partir de , mostrar uma parte S de V tal que $\in S$, $\notin S$ e nxiste $uv \in A$ com u

em S e v em $V \setminus S$. A partir de um c-potencial, podemos extrair ambos os certificados de otimalidade dos caminhos encontrados, e o certificado de ncessibilidade de alguns vices por .

Lema 2.1 (lema da dualidade): *Seja (V, A) um grafo e c uma fun custo sobre V. Para todo caminho P com ino em e tino em e todo c-potencial y vale que*

$$c(P) \geq y() - y().$$

Demonstração: Suponha que P aminho $(= v_0, \alpha_1, v_1, \ldots, \alpha_k, v_k =)$. Temos que

$$\begin{aligned} c(P) &= c(\alpha_1) + \ldots + c(\alpha_k) \\ &\geq (y(v_1) - y(v_0)) + (y(v_2) - y(v_1)) + \ldots + (y(v_k) - y(v_{k-1})) \\ &= y(v_k) - y(v_0) = y() - y(). \end{aligned}$$

□

Do lema 2.1 tem-se imediatamente os seguintes corolos.

Corolário 2.2 (condi de inacessabilidade): *Se (V, A) grafo, c a fun custo, y c-potencial e e sertices tais que*

$$y() - y() \geq nC + 1$$

ent n acessl a partir de .

Corolário 2.3 (condi de otimalidade): *Seja (V, A) um grafo e c a fun custo. Se P caminho de a e y c-potencial tais que $y() - y() = c(P)$, entPcaminhoquetemcustommo.*

2.4 REPRESENTA DE CAMINHOS

Uma **fun predecessor** a fun sobre V tal que, para cada v em V,

$$(v) = \quad \text{ou} \quad ((v), v) \in A.$$

Funs desse tipo sma maneira compacta e eficiente de representar caminhos de um dado vice atda um dos demais vices de um grafo.

Dado um grafo direcionado $G = (V, A)$, uma fun predecessor sobre V e um caminho

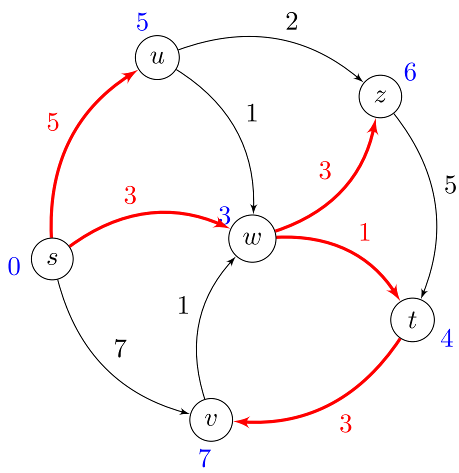


Figura 2.3: *Grafo com custos nos arcos e um potencial nos vices. O potencial exibido garante que qualquer caminho formado por vices vermelhos a partir de   caminho de custo mmo.*

$P = (v_0, v_1, \ldots, v_k)$, dizemos que P **caminho determinado por**   se

$$v_0 = (v_1), v_1 = (v_2), \ldots, v_{k-1} = (v_k).$$

Dado um grafo $G = (V, A)$ e uma fun predecessor   em V, dizemos que o grafo induzido por , da forma (V, Ψ) grafo de predecessores, onde $\Psi = \{uv \in A : u = (v)\}$.

vice	
w	
u	
	w
z	w
v	

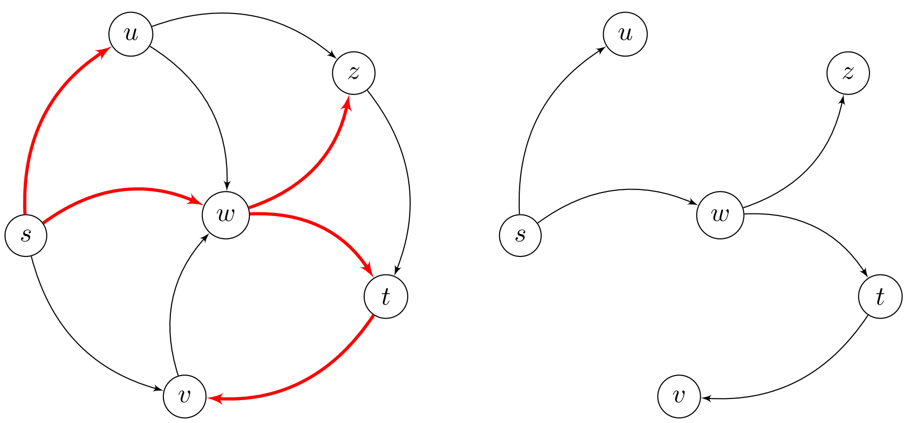


Figura 2.4: *Exemplo de uma fun predecessor   e de uma grafo de predecessores induzido por ela. Acima temos os valores de   para cada vice. O grafo da esquerda mostra as arestas induzidas por   em vermelho. A grafo da direita rafos de predecessores a partir de .*

2.5 Examinando arcos e vices

Temos que ale uma fun predecessor para representar os caminhos, um outro elemento muito til em algoritmos que resolvem o a fun potencial. O custo dos caminhos que tem como origem o vice simitados inferiormente por esta fun.

Examinar um arco ou relaxar/rotular um arco (*relaxing* [?], *labeling step* [?]) a opera bca envolvendo uma fun predecessor e uma fun potencial y, e consiste em verificar se y respeita c em uv e caso nespeite, ou seja,

$$y(v) - y(u) > c(uv) \text{ ou, equivalentemente } y(v) > y(u) + c(uv)$$

fazer

$$y(v)y(u) + c(uv) \text{ e } (v)u.$$

Podemos interpretar esta opera como a tentativa de encontrar um "atalho"para o caminho de a v no grafo de predecessores, passando pelo arco uv.

(uv) $\triangleright$ examina o arco uv

$y(v) > y(u) + c(u, v)$

  então $y(v)y(u) + c(uv)$

    $(v)u$

Podemos estender a opera de examinar um arco em outra opera bca, que seria examinar um vice. Examinar um vice u consiste em examinar todos os arcos da forma uv, $v \in V$.

(u) $\triangleright$ examina o vice u

uv em A **faça**

  (uv)

Abaixo temos uma versxpandida.

(u) $\triangleright$ examina o vice u

uv em A **faça**

  $y(v) > y(u) + c(uv)$

    então $y(v)y(u) + c(uv)$

      $(v)u$

As operas de examinar arcos ou vices sempre diminuem o valor da fun potencial em um vice visando respeitar a fun custo no elemento em que se estaminando. Ou seja, tentando tornar y em um c-potencial.

2.6 Algoritmos

Existem vos algoritmos eficientes para resolver o problema de caminho mmo, sendo os mais conhecidos os algoritmos de Dijkstra e o de Ford. Ambos aplicam-se ao problema como definimos, caminho mmo de um vice inicial para um nal ou de para todos os outros vices. Apresentamos com detalhes o algoritmo de Dijkstra.

2.6.1 Algoritmo de Dijsktra

Vamos descrever agora o famoso algoritmo de Edsger Wybe Dijkstra dijkstra:59 que resolve o problema do caminho mmo em grafos onde a fun custo possui apenas custos negativos, ou seja, $c(uv) \geq 0$ para todo $uv \in A$. Nosso texto segue de perto fabio:09 e pf:fluxos.

Descri

O algoritmo erativo. No ino de cada itera tem-se os conjuntos S e Q, que sma parti do conjunto de vices do grafo ($S \cap Q = \emptyset$ e $S \cup Q = V$). O algoritmo conhece caminhos partindo de a cada vice em S, caminhos estes que sarantidamente de custo mmo, e conhece caminhos a uma parte dos vices em Q. Cada itera consiste em retirar um determinado vice de Q, examino e adiciono a S. Eventualmente, ao examinar tal vice, descobrimos caminhos a vices em Q attlcanos, ou melhores que os jnhecidos.

O algoritmo recebe um grafo $G = (V, A)$, uma fun custo c de A em e um vice e devolve uma fun-predecessor e uma fun-potencial y que respeita c tais que, para cada vice , se essl a partir de , ent$determinaumcaminhodeaquetemcomprimento$y() - y()$, casocontro$y()-y() $=$ nC+1$, onde$C := \max\{c(uv) : uv \in A\}$ [1].

$(V, A, c,)$ $\quad \triangleright\ c \geq 0$

v em V **faça**

$y(v) nC + 1$ $\quad \triangleright\ nC + 1$ faz o papel de ∞

(v)

---

[1] Se determina um caminho de a um vice , entste caminho tem custo mmo (condi de otimalidade, corolo 2.3). Se y c-potencial com $y() - y() = nC + 1$, entxiste caminho de a (condi de inacessibilidade, corolo 2.2).

$y()0$

QV $\triangleright$ Q func. como uma fila com prioridades

$Q \neq ()$ **faça**

retire de Q um vice u com $y(u)$ mmo

(u)

e y

Abaixo temos uma versxpandida.

$(V, A, c,)$ $\triangleright$ $c \geq 0$

v em V **faça**

$y(v)nC + 1$ $\triangleright$ $nC + 1$ faz o papel de ∞

(v)

$y()0$

QV $\triangleright$ Q func. como uma fila com prioridades

$Q \neq ()$ **faça**

retire de Q um vice u com $y(u)$ mmo

uv em A **faça**

$y(v) > y(u) + c(uv)$ **então**

$y(v)y(u) + c(uv)$

$(v)u$

e y

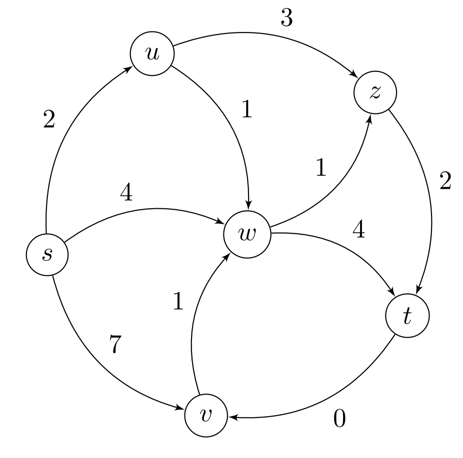
Simula

A seguir, temos uma simula para uma chamada do algoritmo . Na figura (1) temos o grafo (V, A) com custo sobre os arcos. Nas figuras os vices em Q tnterior azul claro. A funpotencial y dicada pelos nmeros em azul pros cada vice. Na figura (2) temos que todos os vices foram inseridos em Q e os potencias foram inicializados ($y() = 0$ e o potencial dos demais vices $\times C + 1 = 6 \times 7 + 1 = 43$).

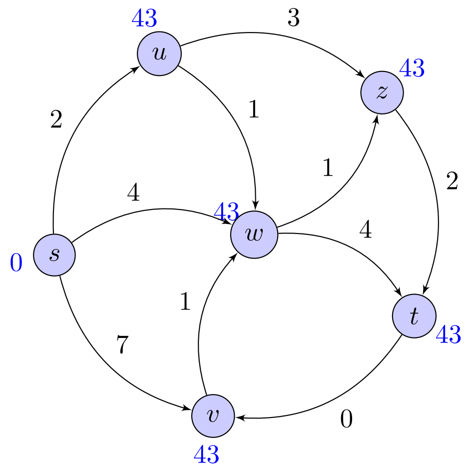
Um vice sendo examinado tem a cor rosa. O arco sendo examinado tem a cor azul (na figura (3) o vice sendo examinado $eoarcosendoexaminadov$).

Os vices em S (jaminados) t cor verde. Na figura (10) vemos que w estndo examinado, e u ss vices em S e os demais estm Q.

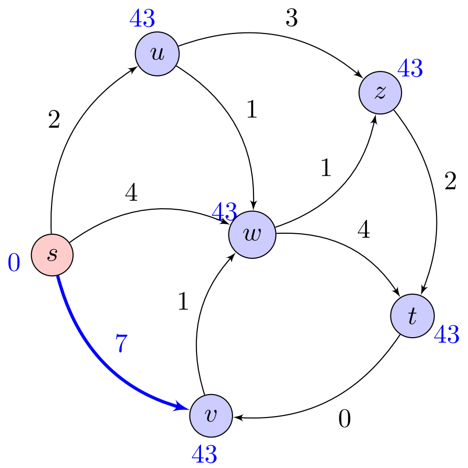
Os arcos jaminados t cor vermelha se fazem parte do grafo de predecessores, caso contro sracejados.



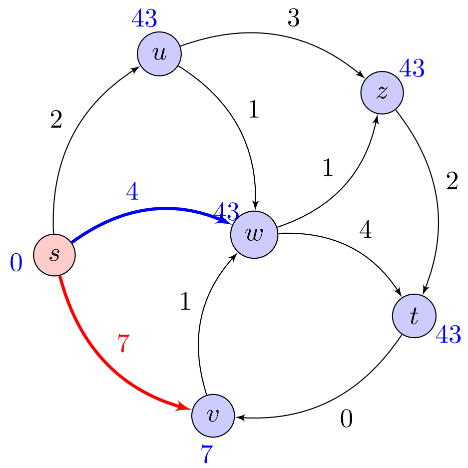
(1)



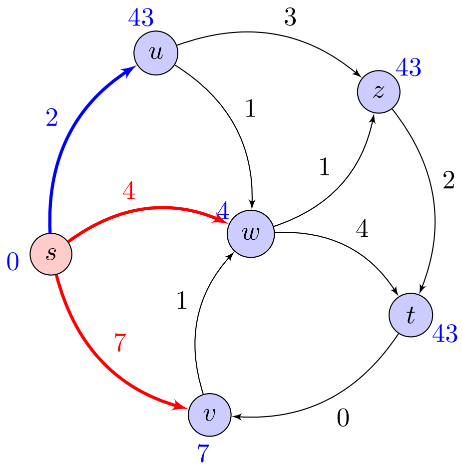
(2)



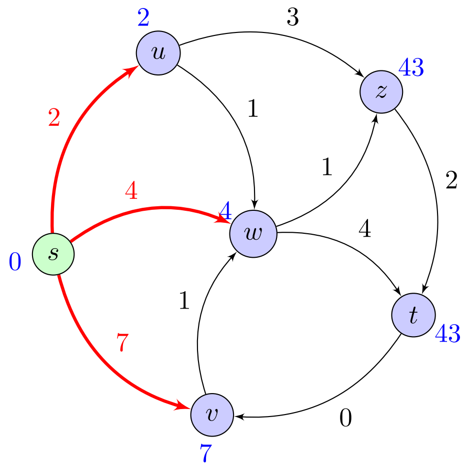
(3)



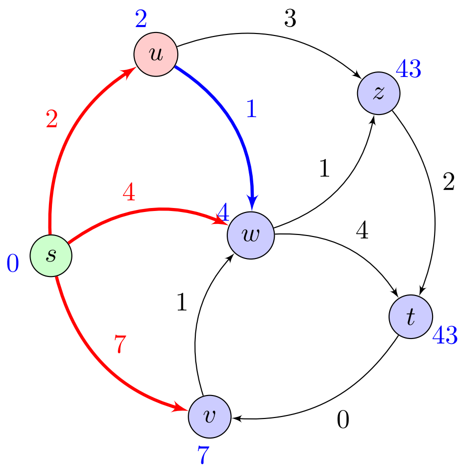
(4)



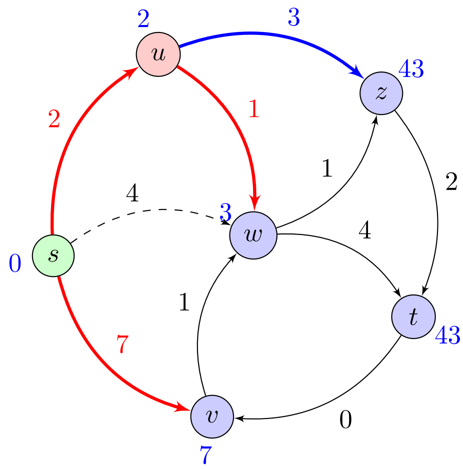
(5)



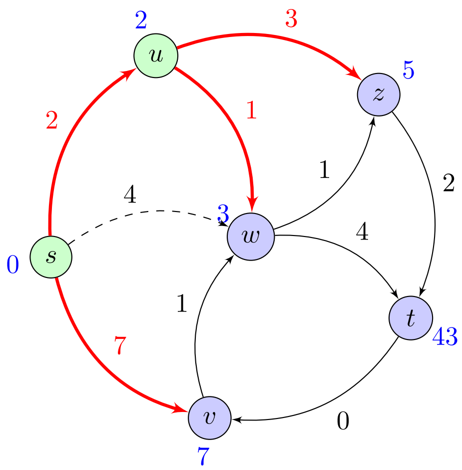
(6)



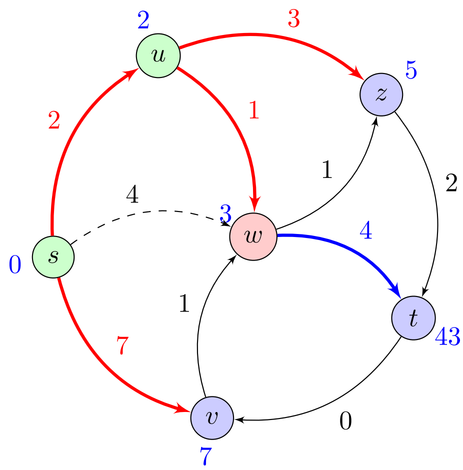
(7)



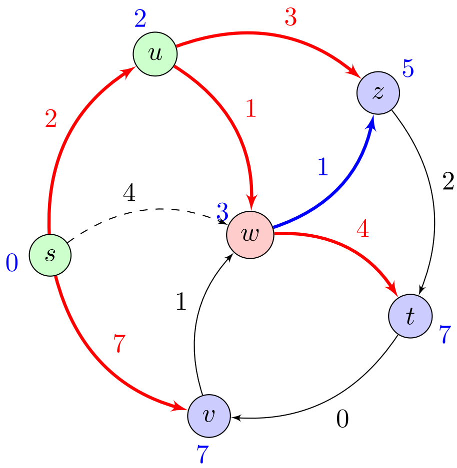
(8)



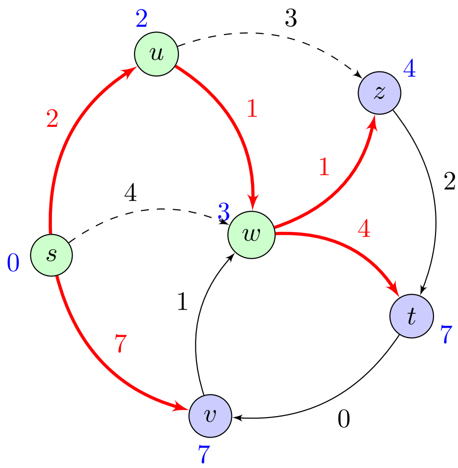
(9)



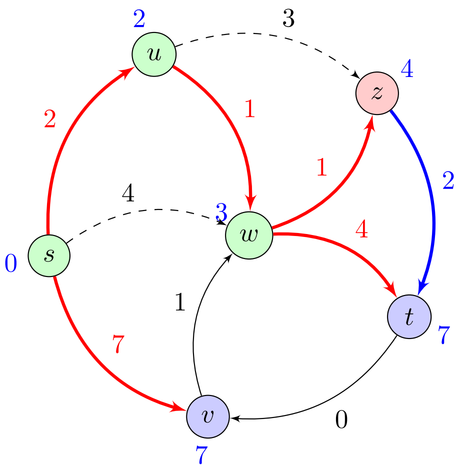
(10)



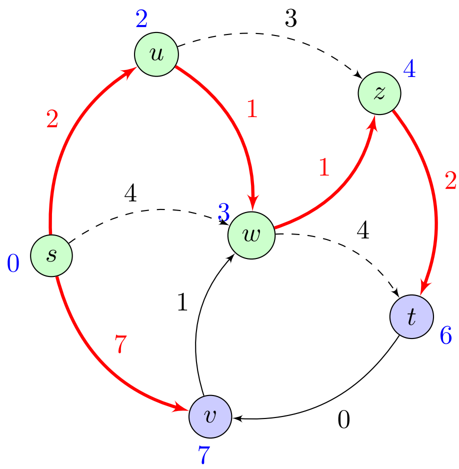
(11)



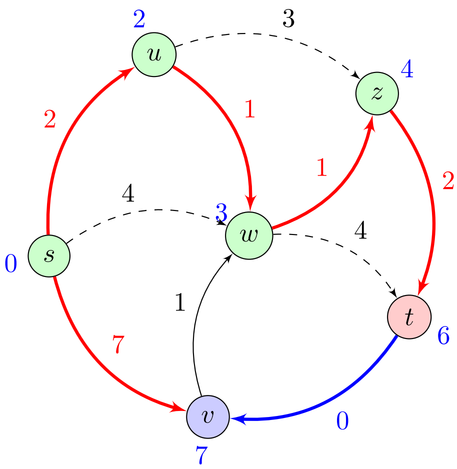
(12)



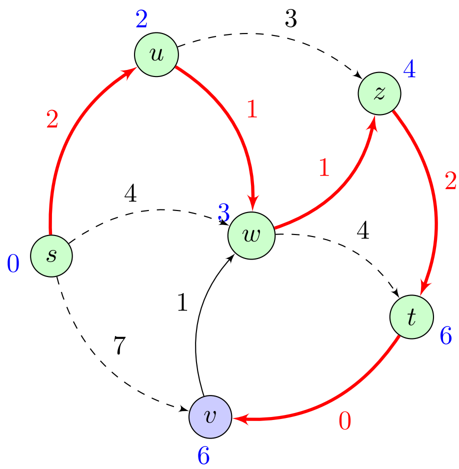
(13)



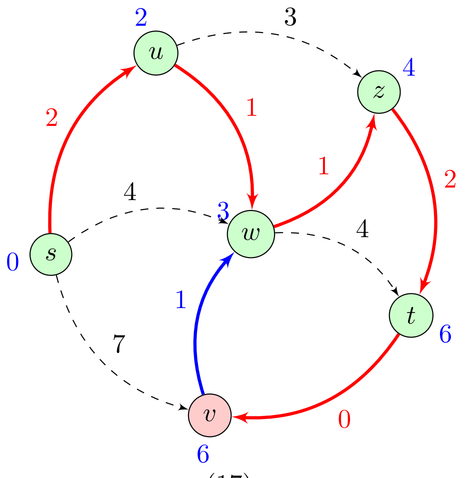
(14)



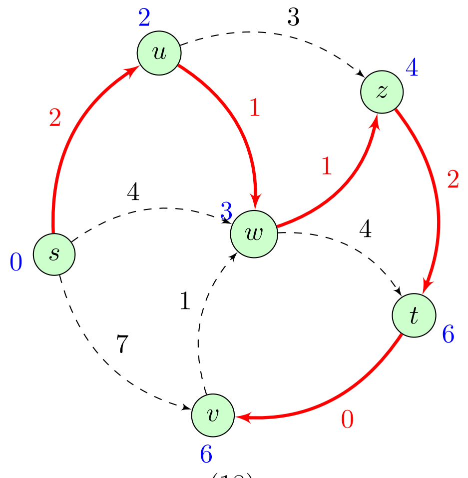
(15)



(16)



(17)



(18)

CORRETUDE

A corretude do algoritmo de Dijkstra baseia-se nas demonstras da validade de uma se de relas invariantes. Estas relas sfirmas envolvendo os dados do problema V, A, c e e os objetos $y, , S$ e Q. Para uma prova detalhada recomendamos a leitura de shigueo:02. Aqui faremos apenas uma argumenta bca para mostrar que o algoritmo rreto.

Teorema 2.4 (da corretude): *Dado um grafo (V, A), uma fun custo c e um vice o algoritmo corretamente encontra um caminho de custo mmo de a , para todo vice acessl a partir de .*

Demonstração: Como Q zio no final do processo, vale que todos os vices, e portanto todos os arcos, foram examinados, o que garante que a fun y c-potencial. Se $y() < nC + 1$ ent valor de $y()$ foi atualizado ao menos uma vez, e assim vale que $() \neq$. Logo, segue que existe um st-caminho P no grafo de predecessores e P a caminho de custo mmo pela condi de otimalidade porque

$$y(v) = y(u) + c(uv), \forall uv \in \Psi \Rightarrow c(P) = y() - y() = y().$$

Je $y() = nC + 1$, temos que $y() - y() = nC + 1$ e da condi de inexistia conclus que nxiste caminho de a no grafo.

Conclus portanto que o algoritmo faz o que promete. □

CONSUMO DE TEMPO

As duas principais operas no algoritmo ss seguintes (analisadas no pior caso):

1. escolher um vice com potencial mmo, que pode gastar tempo $O(n)$ e ecutada at$vezes(at$ ficar vazio), ou seja, consome tempo $O(n^2)$;
2. e atualizar o potencial, que pode acontecer para todas as arestas, ou seja, gasta tempo $O(m)$.

Assim, o consumo de tempo do algoritmo no pior caso $(\mathrm{n}^2 + m) = O(n^2)$. Para grafos esparsos, existem mdos sofisticados, como o heap de Johnson johnson:heap, o heap de Fibonacci FredTarjan:Fibonacci, que permitem diminuir o tempo gasto para encontrar um vice com potencial mmo, gerando assim implementas que consomem menos tempo para resolver o problema.

Dijkstra e filas de prioridades

Uma fila de prioridades [**?**, **?**] a estrutura de dados que armazena uma cole de itens, cada um com uma prioridade associada. Os itens serasicamente vices em nosso contexto.

Temos as seguintes operas para uma fila de prioridade Q:

- (v, p, Q): adiciona o vice v com prioridade p em Q.
- (v, Q): remove o vice v de Q.
- (Q): devolve o vice com a menor prioridade e o remove de Q.
- (v, p, Q): muda para p a prioridade associada ao vice v em Q (p deve ser menor que a atual prioridade associada a v).
- (V): recebe o conjunto V de vices em que cada vice v tem prioridade $y(v)$ e constra fila de prioridades Q.

A maneira mais popular para implementar o algoritmo de Dijkstra ilizando uma fila de prioridades para representar , onde a prioridade de cada vice v eu potencial $y(v)$. A descri do algoritmo de Dijkstra logo a seguir faz uso das operas , e , especificadas acima.

$(V, A, c,)$ $\quad \triangleright\ c \geq 0$

v em V **faça**

$\quad y(v) nC + 1 \quad \triangleright\ nC + 1$ faz o papel de ∞

$\quad (v)$

$y()0$

$Q\,() \quad \triangleright\ Q$ min-heap

	heap		fibonacci heap	bucket heap	radix heap
	$O(\log n)$	$O(\log_D n)$	$O(1)$	$O(1)$	$O(\log(nC))$
	$O(\log n)$	$O(\log_D n)$	$O(\log n)$	$O(C)$	$O(\log(nC))$
	$O(\log n)$	$O(\log_D n)$	$O(1)$	$O(1)$	$O(1)$
Dijkstra	$O(m \log n)$	$O(m \log_D n)$	$O(m + n \log n)$	$O(m + nC)$	$O(m + n \log(nC))$

Tabela 2.1: *Complexidade do algoritmo de Dijkstra de acordo com as filas de prioridade.*

$Q \neq ()$ **faça**

$u\,(Q)$

uv em $A(u)$ **faça**

$y(u) + c(uv)$

$y(v) >$ **então**

$(, v, Q)$

$(u)v$

e y

O consumo de tempo do algoritmo Dijkstra pode variar conforme a implementa de cada uma dessas operas da fila de prioridade: , e . Existem muitos trabalhos envolvendo implementas de filas de prioridade com o intuito de melhorar a complexidade do algoritmo de Dijkstra. Para citar alguns exemplos temos [**?**, **?**, **?**].

A tabela a seguir resume o consumo de tempo de vas implementas de filas de prioridade e o respectivo consumo de tempo resultante para o algoritmo de Dijkstra [**?**].

Fredman e Tarjan [**?**] observaram que como o algoritmo de Dijkstra examina os vices em ordem de distia a partir de , o algoritmo estmplicitamente, ordenando estes valores. Assim, qualquer implementa do algoritmo de Dijkstra realiza, no pior caso, $\Omega(n \log n)$ comparas e faz $\Omega(m + n \log n)$ operas elementares.

2.6.2 Algoritmo de Ford

Ao contro do algoritmo , este algoritmo, que foi proposto por Ford bellman:58, pode ser aplicado a grafos cujos arcos tssociado custos negativos, nodendo contudo existir circuitos de custo negativo carlos:98.

Este algoritmo consiste em examinar os vices do grafo, corrigindo os potencias as vezes que for necesso, ate todos os potencias satisfa a condi de otimalidade. Neste algoritmo, os potenciais dos vices t mesmo significado dos potencias utilizados no algoritmo , s agora os potenciais sornam definitivos aprminar a execu do algoritmo.

O algoritmo recebe um grafo $G = (V, A)$, uma fun custo c de A em e um vice e devolve uma fun-predecessor e uma fun-potencial y que respeita c tais que, para cada vice , se essl a partir de , ent$determinaumcaminhodeaquetemcomprimento$y() - y(), $casocontro$y()-y() = nC+1, $onde$C := max$\{c(uv) : uv \in A\}$. Versenca do algoritmo.

$(V, A, c,)$ $\triangleright$ (V, A, c) nossui ciclos negativos

v em V **faça**

$y(v)nC + 1$ $\triangleright$ $nC + 1$ faz o papel de ∞

(v)

$y()0$

$y(v) > y(u) + c(uv)$ **algum** $uv \in A$ **faça**

$y(v)y(u) + c(uv)$

$(v)u$

e y

O consumo de tempo desta vers $O(n^2mC)$ pf:fluxos. O consumo de tempo o elevado porque o algoritmo pode examinar cada arco muitas veze (o valor de $y(v)$ pode diminuir muitas vezes antes de atingir seu valor final). A seguir temos a melhor versonhecida, do ponto de visa do consumo assinto de tempo, para o problema do caminho mmo com custos arbitros.

$(V, A, c,)$ $\triangleright$ (V, A, c) nossui ciclos negativos

v em V **faça**

$y(v)nC + 1$ $\triangleright$ $nC + 1$ faz o papel de ∞

(v)

$y()0$

$n - 1$ **vezes**

uv em A **faça**

$y(v) > y(u) + c(uv)$ **então**

$y(v)y(u) + c(uv)$

$(v)u$

e y

O algoritmo consome $O(nm)$ unidades de tempo. Como $m = O(n^2)$ no pior caso, pode-se dizer que o algoritmo (n^3).

3

Caminhos mmos com recursos limitados

3.1 Defini do problema

Problema 2. *$(G, , , k, r, \lambda, c)$ Como partros do problema sados*

- *um grafo dirigido $G = (V, A)$,*
- *um vice origem $\in V$ e um vice destino $\in V$, $\neq$,*
- *um nmero $k \in \mathbb{N}$ de recursos disponis $\{1, \ldots, k\}$,*
- *o consumo de recursos $r^i_{uv} \in \mathbb{N}_0$ de cada arco de G sobre os k recursos disponis, $i = 1, \ldots, k$, $uv \in A$,*
- *o limite $\lambda^i \in \mathbb{N}_0$ que dispomos de cada recurso, $i = 1, \ldots, k$,*
- *o custo $c_{uv} \in \mathbb{N}_0$, para cada arco, $uv \in A$.*

O consumo de um recurso i, $i = 1, \ldots, k$ em um st-caminho P $^i(P) = \sum_{uv \in P} r^i_{uv}$. Um st-caminho P mitado pelos recursos $1, \ldots, k$ se este consome nais que o limite disponl de cada recurso, ou seja, se $r^i(P) \leq \lambda^i$, $i = 1, \ldots, k$. O custo de um st-caminho P $(P) =$

$\sum_{uv \in P} c_{uv}$. *O problema consiste em encontrar o caminho limitado pelos recursos de menor custo.*

Usaremos no decorrer deste trabalho $n = |V|$ e $m = |A|$. Quando estivermos tratando de um contexto onde existe apenas um recurso (), ou seja, $k = 1$, usaremos apenas λ para representar λ^1 e apenas r_{uv} para representar r^1_{uv}.

3.2 REVISIBLIOGRCA

Pelo fato do estar "embutido" em um grande nmero de problemas prcos, ele tem sido extensivamente estudado como strado na tabela 3.1 que mostra uma lista com algumas caractericas dos principais algoritmos disponis.

Basicamente, duas famas de algoritmos tem sido propostas: uma envolve resolver o problema relaxado usando relaxa linear ou relaxa lagrangiana e a outra usa programa dinca. Mdos baseados em relaxas geralmente consistem em trassos: (1) computar limites inferior e superior para a solu a resolvendo o problema relaxado, (2) usar os resultados do primeiro passo para reduzir o grafo, e (3) eliminar a folga da dualidade.

Seguindo esta linha, zang:80 resolveram um dual lagrangiano (para a versom um nico recurso) no passo (1), para eliminar a folga da dualidade, eles usaram o algoritmo de yen:71 (que algoritmo que calcula todos os caminhos entre um par de vices em ordem crescente de custo, desenvolvido para o problema do k-mo menor caminho, ou KSP – *k shortest path problem*). A ideia era que, embora estivessem usando o algoritmo de Yen que ponencial, o nmero de caminhos computados seria bastante reduzido. beasley:89 apresentaram uma abordagem baseada em relaxa lagrangiana que usa o mdo do sub-gradiente para resolver o dual lagrangiano como primeiro passo e *branch and bound* para eliminar a folga da dualidade. mehlhorn:00 propuseram o mdo do envolt que resolve uma relaxa linear para o caso com mltiplos recursos, para eliminar a folga eles usam um mdo de enumera de caminhos como em hassin:92.

Um nmero considerl de publicas abordam solus baseadas em programa dinca para o . joksch:66 props uma programa dinca, bem como hassin:92, que apresentou dois algoritmos exatos para uso com grafos acicos. aneja:83 adaptou o esquema de rotulamento permanente (*label-setting*) de dijkstra:59. A abordagem de rotulamento permanente a generaliza do algoritmo , que rotula e processa os vices em ordem, baseado nos consumos de recursos. Alem do mdo de rotulamento permanente, temos a abordagem de rotulamento corretivo (*label-correcting*) que a generaliza do algoritmo ; este trata cada vice vas vezes, atualizando os ros quando necesso. [?] investigaram varias do algoritmo de rotulamento corretivo, eles apresentaram uma verselhorada do algoritmo de aneja:83, alem de apresentar um algoritmo

Referencia	Verso	Mdo	Custo	Grafo
zang:80	nico recurso	RL + YEN	$c_{uv} \geq 0$	gerais
beasley:89	mltiplos recursos	RL + BB	irrestrito	gerais
mehlhorn:00	mltiplos recursos	PL	$c_{uv} \geq 0$	gerais
joksch:66	nico recurso	PD	$c_{uv} > 0$	gerais
aneja:83	mltiplos recursos	LS	$c_{uv} \geq 0$	gerais
hassin:92	nico recurso	PD	$c_{uv} > 0$	acicos
dumitrescu:03	mltiplos recursos	LS	$c_{uv} \geq 0$	gerais

Tabela 3.1: *Principais algoritmos disponis para o .*
RL – relaxa lagrangiana; KSP – k-mo menor caminho; BB – *branch and bound*; PL – relaxa linear; PD – programa dinca; LS – rotulamento permanente.

baseado em um mdo de escalar pesos.

3.3 COMPLEXIDADE

zang:80, jaffe:84 mostraram que o NP$-difl, mesmoemgrafosacicos, comrestrissobreumnicorecurso, ecomtodososconsumosderecursosposi$

hassin:92 mostrou que o tem solu polinomial se os custos dos arcos e consumos de recursos simitados. Temos por beasley:89 que a solu do rantidamente elementar (nassa por um mesmo vice duas vezes) se os custos dos arcos snegativos e temos apenas restris que limitam a quantidade mma de recursos consumidos ou o grafo ico. A seguir, mostraremos uma redu do **problema da mochila** (*knapsack*), definido a seguir, ao problema (baseada em zang:80).

Problema 3. MOCHILA*(N,w,v,D) Como partros do problema sados:*

- *um conjunto de itens* $N = \{1, \ldots, n\}$*,*
- *pesos* $w_i \in \mathbb{N}$*,* $i = 1, \ldots, n$*, para esses itens,*
- *valores* $v_i \in \mathbb{N}$*,* $i = 1, \ldots, n$*, para esse itens,*
- *um peso limite* $D \in \mathbb{N}_0$*.*

O peso de um subconjunto $I \subseteq N$ *(I)* $= \sum_{i \in I} w_i$*, e seu valor (I)* $= \sum_{i \in I} v_i$*. O problema* MOCHILA *consiste em encontrar um subconjunto de itens com valor mmo, cujo peso nxcede o limite* D*.*

Teorema 3.1: *O* $NP-difl$*.*

Demonstração: A prova se dla redu do problema MOCHILA ao . Vamos tomar uma instia I do problema MOCHILA. Ndemos construir uma instia I' para o como se segue:

- $V = N \cup \{0\}$.
- Entre cada par de vice $i-1$ e i, onde $i = 1, 2, \cdots, n$, teremos duas arestas paralelas $(i-1, i)$ que estareparadas, uma em cada um dos dois subconjuntos A_1 e A_2 que compem A.
  - $A = A_1 \cup A_2$,
  - $A_1 = \{(i-1, i) : i = 1, \ldots, n\}$,
  - $A_2 = \{(i-1, i) : i = 1, \ldots, n\}$.
- $r_{uv} = \begin{cases} w_i, & \text{se } uv \in A_1, \\ 0, & \text{caso contro} \end{cases}$ para todo $uv \in A$.
- $c_{uv} = \begin{cases} M - v_i, & \text{se } uv \in A_1, \\ M, & \text{caso contro} \end{cases}$ para todo $uv \in A$.
- $= 0$.
- $= n$.
- $k = 1$.
- $\lambda = D$.

A constante M pode ser definida como um grande inteiro de tal forma que $M - v_i$, para qualquer i, seja negativo. Por queste praticidade, vamos convencionar que representaremos um arco $(i-1, i) \in A_1$ como a_i^1 e um arco $(i-1, i) \in A_2$ como a_i^2.

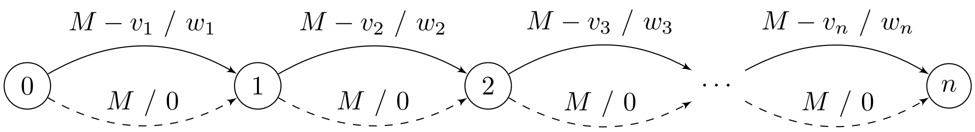


Figura 3.1: *Os arcos contos representam os arcos no conjunto A_1 e os arcos tracejados representam os arcos no conjunto A_2. O ro de cada arco uv representa o seu custo e o seu consumo de recurso, respectivamente (c_{uv} / r_{uv}).*

Como $= 0$, $= n$; podemos ver que qualquer -caminho P em $G = (V, A)$ contu a_i^1 ou a_i^2, $i = 1, \ldots, n$. Vamos dividir os arcos de P em dois conjuntos X e Y, onde X conts arcos em P que estm A_1, e Y conts demais arcos. A partir de X, vamos definir um subconjunto $S \subseteq N$, tal que $i \in S$ se e somente se $a_i^1 \in X$. Com isso,

$$\begin{aligned} c(P) &= \sum_{a_i^1 \in X} (M - v_i) + \sum_{a_i^2 \in Y} M \\ &= n \cdot M - \sum_{a_i^1 \in X} v_i \\ &= n \cdot M - v(S) \end{aligned} \tag{3.1}$$

$$
\begin{aligned}
r(P) &= \sum_{a_i^1 \in X} w_i + \sum_{a_i^2 \in Y} 0 \\
&= \sum_{a_i^1 \in X} w_i \\
&= w(S)
\end{aligned}
\tag{3.2}
$$

Daonclus que todo subconjunto $S \subseteq N$ contm st-caminho P associado, e vice-versa, por meio da equivalia $i \in S \Leftrightarrow a_i^1 \in P$. Pelas equas 3.1 e 3.2, um conjunto S e um caminho P associados, possuem $r(P) = w(S)$ e $c(P) = n \cdot M - v(S)$. E damos dois resultados:

$$
\begin{aligned}
r(P) \leq \lambda &\iff w(S) \leq D \\
\text{minimizar } c(P) &\iff \text{maximizar } v(S)
\end{aligned}
$$

□

3.4 Preprocessamento

Nesta se nvisamos algumas possibilidades de redu do tamanho do problema pela elimina de vices e/ou arcos que nodem fazer parte de uma solu a.

3.4.1 Redu baseada nos recursos

Vamos denotar R_{uv} como sendo a menor quantidade de recurso que podemos usar partindo do vice u atice v. Qualquer vice v para o qual vale que

$$R_v + R_v > \lambda$$

pode ser removido do grafo, tendo em vista que este vice node pertencer a qualquer caminho vil. De forma similar, qualquer arco uv para o qual vale que

$$R_u + r_{uv} + R_v > \lambda$$

pode ser removido do grafo pelo mesmo motivo. aneja:83 foi o primeiro a apresentar redus baseadas nos recurso como descrito acima. Podemos executar esta redu com duas computas do algoritmo (para calcular R_v e R_v para qualquer $v \in A$) seguido pela itera sobre todos os vices e arcos verificando as condis acima descritas ($O(n^2 + m + n)$).

3.4.2 Redu baseada nos custos

Caso tenhamos um limite superior UB conhecido para a solu a do problema, ndemos extender a ideia da redu baseada em recursos para uma redu baseada em custos. Vamos denotar C_{uv} como sendo o menor custo possl para um caminho partindo de u at.$Qualquervicev paraoqualvaleque C_v + C_v > UB podeserremovido, damesmaformaquequalquerarcouvparaoqualvaleque C_u + c_{uv} + C_v > UB podeserremovido. Nestecaso, aefetividadedaredudependediretamentedaqualidadedolimitesuperior$

dumitrescu:03 props que esta redu fosse executada repetidas vezes, atualizando-se o limite superior UB caso possl, ate nosse feita nenhuma remo no grafo. O mdo funciona bem quando o instia possui restris bem "apertadas".

3.5 Programa dinca

Programa dinca a tica bastante poderosa para resolver determinados tipos de problemas computacionais. Muitos algoritmos eficientes fazem uso desse mdo. Basicamente, esta estrata de projeto de algoritmos, traduz uma recursara uma forma iterativa que utiliza uma tabela como apoio para as computas.

Da mesma forma que acontece em um algoritmo recursivo, em um algoritmo baseado em programa dinca, cada instia do problema solvida a partir da solu de subinstias menores da instia original. O que distingue a programa dinca de uma recursomum so da tabela que "memoriza" as solus das subinstias evitando assim o recalculo caso esses valores sejam necessos mais de uma vez.

Para que o paradigma da programa dinca possa ser aplicado, eciso que o problema tenha as seguintes caractericas: subestrutura a e superposi de subproblemas. Um problema apresenta uma subestrutura a quando uma solu a para o problema contou pode ser calculada a partir de) solus as para subproblemas. A superposi de subproblemas acontece quando um algoritmo recursivo recalcula o mesmo problema muitas vezes.

Temos alguns algoritmos baseados em programa dinca conhecidos para o , todos eles tem complexidade pseudo-polinomial, pois suas complexidades de tempo dependem do tamanho dos custos e consumo de recursos dos arcos. Neste caplo iremos falar um pouco sobre trestas solus. A primeira itera sobre os possis valores de consumo, minimizando o custo; aqui iremos referencia como programa dinca primal. A segunda, muito similar a primeira, itera sobre os possis valores de custo, verificando se o consumo de recursos atende ao limite imposto; iremos referencia como programa dinca dual. Por ltimo temos uma modelagem baseada em uma generaliza do algoritmo , geralmente relacionada a solus do problema de caminho mmo multi-objetivo, que aqui chamaremos de programa dinca por ros. Embora todos os algoritmos possam ser usados na verso problema com mltiplos recursos, por queste praticidade vamos descrevos na versom um nico recurso.

3.5.1 Programa dinca primal

Um dos primeiros a descrever um algoritmo baseado em programa dinca para o foi joksch:66 (ver tambgoldman:65, lawler:76). Nesta se iremos descrever esse algoritmo.

Na programa dinca primal, iteramos sobre o consumo de recursos, partindo de uma quantidade unita atimite imposto, obtendo os caminhos mmos limitados pelos recursos com custo mmo partindo de . Vamos definir a recorria sobre a qual o algoritmo plementado da seguinte forma. Definimos $f_j(r)$ como sendo o menor custo possl para um caminho de a j, que consome no mmo r unidades de recurso, e assim temos:

$$
f_v(r) = \begin{cases} 0, & se v = \\ & e r = 0, \ldots, \lambda \\ \\ \infty, & se v \neq \\ & e r = 0 \\ \\ \min\left\{ f_v(r-1), \min\limits_{u \mid r_{uv} \leq r} \{ f(r - r_{uv}) + c_{uv} \} \right\}, & se v \neq \\ & e r = 1, \ldots, \lambda \end{cases}
$$

A partir da recorria podemos extrair de forma direta, uma recurse complexidade exponencial. Temos como base da recursf$_(r) = 0$ para $1 \leq r \leq \lambda$ e $f_j(0) = \infty$ para $j \neq$. Abaixo temos o algoritmo recursivo denominado de [1] [2].

[1]Para que os partros no algoritmo normem uma lista muito extensa, vamos considerar que temos acesso de forma global ao grafo $G = (V, A)$, as funs c e r sobre os arcos, o vice de origem e a fun predecessor .

[2]Na descri do algoritmo denotaremos por r (sem ice) o consumo mmo de recursos permitido para a chamada corrente. Sempre que r representar a fun de consumo sobre os arcos ele estarompanhado pelo ice que representa o arco, r_{uv} por exemplo.

(v, r)

```
(v, r)
 v = então
     0
 r = 0 então
     ∞
custo (v, r − 1)
(v, r)(v, r − 1)
  uv em A faça
     r_uv ≤ r então
          valor (u, r − r_uv) + c_uv
          custo > valor então
               (v, r)u
               custovalor
  custo
```

mportante salientar que se no grafo, existir ciclos com consumo nulo de recursos, o algoritmo recursivo node ser aplicado diretamente pois entraria em “loop infinito”. O que garante que o algoritmo termina ato de que o partro r sempre diminui nas chamadas recursivas e a base da recursesponde de forma direta aos casos com r nulo. Nestes casos, precisaros realizar um preprocessamento no grafo baseados no seguinte fato: Se o arco $uv \in A$ possui $r_{uv} = 0$, temos que todo arco $wu \in A$ pode ser “estendido” como um arco wv onde $c_{wv} = c_{wu} + c_{uv}$ e $r_{wv} = r_{wv} + 0$. O preprocessamento entubstituiria todas os arcos com consumo nulo pelos arcos “estendidos” ate nxistisse arcos com consumo nulo no grafo. Podemos adaptar o algoritmo de floyd:62 para realizar tal processamento em $O(n^3)$.

O valor do caminho o OPT pode ser encontrado pela chamada $(, \lambda)$ do algoritmo que corresponde ao valor $f_(\lambda)$ da recorria, e o caminho o propriamente dito pode ser constru usando a fun predecessor montada no decorrer da recursDefinimos fun predecessor na se 2.4 do caplo 2. Lla era indexada pelos vices, aqui, temos um versstendida que dexada por um par vice e consumo de recursos, por uso ual em ambos os casos..

A partir do algoritmo recursivo, podemos implementar um algoritmo iterativo que computa o valor de um caminho o em tempo $O(m\lambda)$ e consumo de mem $O(n\lambda)$. joksch:66 apresentou melhoras prcas para este algoritmo, contudo a complexidade de pior caso o melhor que a obtida com a ideia bca.

$(,\lambda)$

[ ] $\triangleright$ tabela de programa dinca

r em $\{0, 1, \cdots, \lambda\}$ **faça**

$(, r)$

$[, r]0$

v em $V \setminus \{\}$ **faça**

$(v, 0)$

$[v, 0]\infty$

r de 1 at**faça**

v em $V \setminus \{\}$ **faça**

$(v, r)(v, r - 1)$

$[v, r][v, r - 1]$

uv em A **faça**

$r_{uv} \leq r$ **então**

$[v, r] > [v, r - r_{uv}] + c_{uv}$ **então**

$(v, r)u$

$[v, r][v, r - r_{uv}] + c_{uv}$

$[, \lambda]$

A programa dinca iterativa nda mais sensl a arcos com consumo nulo de recurso que a recurs neste caso, nrecisamos ter ciclos com consumo nulo, um simples arco $uv \in A$ com $r_{uv} = 0$, pode nos fazer acessar uma posi ainda nalculada da tabela e assim computar uma solu incorreta. Desta forma o preprocessamento descrito para remover arestas sem consumo de recursos deve ser executado antes da chamada de .

3.5.2 Programa dinca dual

Na sessnterior vimos um procedimento simples baseado em programa dinca que objetiva minimizar os custos iterando sobre os recursos. hassin:92 descreveu uma versiferente do algoritmo acima mais til para seu propo, que era desenvolver um algoritmo de aproxima para o , que serscutido mais a frente. Nesta se descreveremos a verse Hassin.

Na programa dinca dual, iteramos sobre os custos, partindo de uma quantidade unita atcontrarmos um caminho vil, sempre minimizando o consumo de recursos dos caminhos computados. Digamos que $g_v(c)$ denota o menor consumo de recursos possl para um caminho de a v que custa no mmo c. Ent seguinte recursode ser definida.

$$g_v(c) = \begin{cases} 0, & sev = \\ & ec = 0, \ldots, OPT \\ \\ \infty, & sev \neq \\ & ec = 0 \\ \\ \min\left\{ g_v(c-1), \min\limits_{u|c_{uv} \leq c} \{g(c - c_{uv}) + r_{uv}\} \right\}, & sev \neq \\ & ec = 1, \ldots, OPT \end{cases}$$

Observe que OPT n um valor conhecido no inicio da execu, mas ele pode ser expresso como $OPT = \min\{c \mid g_(c) \leq \lambda\}$. Devemos computar a fun g iterativamente, primeiro para $c = 1$ e $v = 2, \ldots, n$, entara $c = 2$ e $v = 2, \ldots, n$, e assim sucessivamente, atrimeiro valor c' tal que $g_(c') \leq \lambda$. Seremos o conhecimento do valor $OPT = c'$. A seguir temos o algoritmo desenvolvido a partir da recorria apresentada.

(v, c)

(v, c)

$v =$ **então**

0

$c = 0$ **então**

∞

$recurso\ (v, c-1)$

$(v, c)(v, c-1)$

uv em A **faça**

$c_{uv} \leq c$ **então**

$valor\ (u, c - c_{uv}) + r_{uv}$

$custo > valor$ **então**

$(v, c)u$

$recursovalor$

$recurso$

A partir da recurscima podemos implementar o seguinte algoritmo iterativo:

$(, \lambda)$

[ ] $\triangleright$ tabela de programa dinca

$(, 0)$

$[, 0]0$

v em $V \setminus \{\}$ **faça**

$(v, 0)$

$[v, 0]\infty$

$c0$

$[, c] > \lambda$ **faça**

$cc + 1$

$(, c)$

```
11        [, c]0
12         v em V \ {} faça
13           (v, c)(v, c − 1)
14           [v, c][v, c − 1]
15             uv em A faça
16               c_uv ≤ c então
17                 [v, c] > [v, c − c_uv] + r_uv então
18                   (v, c)u
19                   [v, c][v, c − c_uv] + r_uv
20    OPTc
21     OPT, [, OPT]
```

Um cuidado a se tomar com o algoritmo iterativo que acabamos de apresentar e ele pressupe que a instia l, ou seja, que possui solu. O "enquanto" da linha nmero oito sterrompido quando encontramos a solu. Para se verificar a viabilidade da instia, pode-se rodar um , usando a fun de consumo de recurso como fun custo, se o caminho encontrado possui consumo menor que o nosso limite, este caminho j candidato a solu.

Todas a recomendas e observas feitas a splicis a trocando-se os pap entre custo e consumo de recursos. A complexidade de tempo do algoritmo sugerido acima (mOPT)$eocomplexidadedeespa$O(nOPT)$.Observeaindaquena, devolvemosalocustoovalormmodeconsum$

Da mesma forma que acontece com o problema MOCHILA, o algoritmo pseudo-polinomial baseado em programa dinca pode ser adaptado para fornecer um para o com o uso da tica de escalar e arredondar. Discutiremos isso com mais detalhes se 3.6.

3.5.3 Programa dinca por ros

A abordagem de programa dinca por ros (*labeling*) pode ser vista como uma extensos mdos por programa dinca clicos. Um vw-caminho p por um vw-caminho q se $c_p \geq c_q$ e $r_p \geq r_q$, ou seja, q is eficiente que p tanto a respeito de custo quanto ao consumo de recursos [3].

Temos vas verses de programa dinca por ros conhecidas, entre elas ests descritas por aneja:83, desrochers:88, desrosiers:95, stroetmann:97. Elas usam um conjunto de ros para

[3] Embora tenhamos descrito para o caso com um nico recurso, a defini pode ser estendida para o caso com mltiplos recursos.

cada vice. Cada ro representa um caminho q partindo de atice que tal ro estsociado, e presentado pelo par (c_q, r_q), o custo e consumo de recursos do caminho. Somente caminhos nominados podem ter seus correspondentes ros armazenados na lista de cada vice em ordem crescente de custo, o que implica que estm ordem decrescente de consumo de recursos (no caso com um nico recurso). Na figura 3.2 podemos ver esta ordem exemplificada. joksch:66 observou que a lista de ro nominados sices de uma fun escada (*step-function*) e apenas esses devem ser considerados para procurar uma solu a.

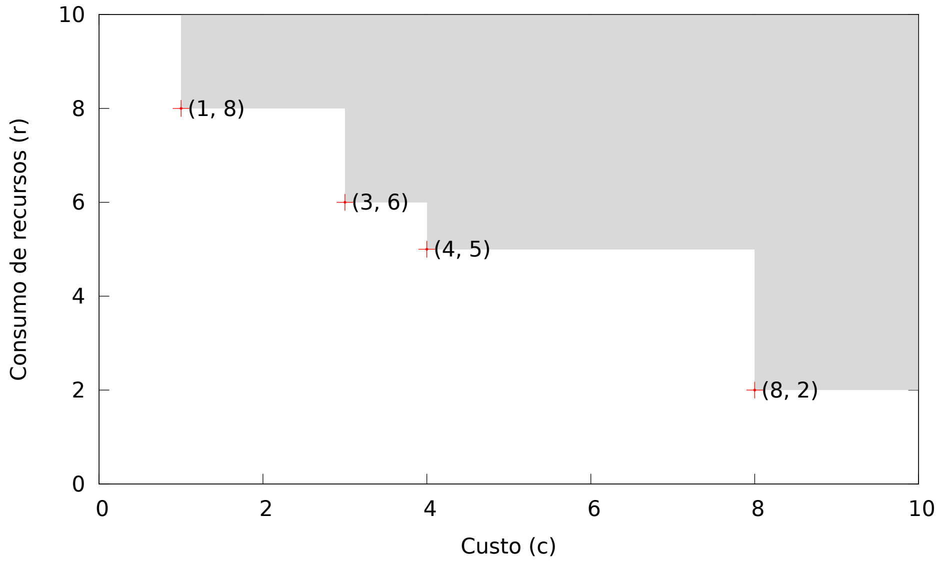


Figura 3.2: *Nesta figura temos o exemplo de uma lista de ros, exibidos em um plano para exemplificar a fun escada. A a em cinza representa a a que minada pelos ros da lista.*

Pode-se dizer que um algoritmo baseado em programa dinca por ros, procura todos os caminhos nominados, para cada vice. Comedo com o ro $(0, 0)$ na lista de ros do vice e as listas dos demais vices vazias. O algoritmo estende a lista de ros conhecidos adicionando um arco ao final do caminho associado a cada ro, esses novos ros srmazenados caso nejam dominados e sejam solus viis.

$(, \lambda)$

[ ] ▷ tabela de programa dinca

$(, (0, 0))$

$()() \cup \{(0, 0)\}$

existe novos ros n‘expandidos” **faça**

u em V **faça**

ro (c, r) em $[u]$ **faça**

uv em A **faça**

```
$(c', r')(c + c_{uv}, r + r_{uv})$
$(c', r')$ n dominado por nenhum ro em $[v]$ **então**
    **remova** os ros em $[v]$ que sominados por $(c', r')$
    $[v][v] \cup \{(c', r')\}$
    $(v, (c', r'))u$
o ro de menor custo em [] que consome nais que $\lambda$ recursos
```

O algoritmo acima descreve o ncleo da abordagem de forma genca. Temos ainda as variantes de ro permanente (*labeling setting*) e ro corretivo (*labeling correcting*) do algoritmo, que basicamente fazem a expansos ros em uma ordem especca. Independente da estrata usada, o pior caso permanece o mesmo, temos uma complexidade de tempo de $O(m\lambda)$.

3.6 ϵ-Aproxima

hassin:92 aplicou a tica de escalar e arredondar para obter um esquema de aproxima totalmente polinomial ($-$ *fully polynomial ϵ-approximation scheme*) para o . Vamos rever, resumidamente, este mdo agora.

Na sess3.5 nmos alguns procedimentos baseados em programa dinca. Para nossos propos aqui, a programa dinca dual (3.5.2) stante til. Nela iteramos sobre o valor de custo c atrimeiro valor c' tal que $g_t(c') \leq R$. Assim, temos o conhecimento do valor $OPT = c'$.

Agora, digamos que V seja um certo valor, e suponha que queremos testar se $OPT \geq V$ ou n Um procedimento polinomial que responde essa questode ser estendido em um algoritmo polinomial parar encontrar OPT simplesmente usando uma uma busca bina. Como nosso problema NP$-difl, temosquenossatisfazercomumtestemaisfraco$.

Tomemos um ϵ fixo, $0 < \epsilon < 1$. Agora, nemos descrever um teste polinomial ϵ-aproximado com as seguintes propriedades: se tal teste devolve uma sa positiva, entefinitivamente $OPT \geq V$. Se ele revolver uma sa negativa, entbemos que $OPT < (V + \epsilon)$.

O teste arrendonda o custo c_{ij} dos arcos, substituindo seu valor por:

$$\left\lfloor \frac{c_{ij}}{\epsilon V/(n-1)} \right\rfloor \cdot \frac{\epsilon V}{(n-1)}$$

.

Isto diminui todos os custos de arcos em no mmo $\epsilon V/(n-1)$, e todos os custos de caminhos em no mmo ϵV. Agora o problema pode ser resolvido aplicando o algoritmo anterior ao grafo com os custo dos arcos escalados para $\lfloor c_{ij}/(\epsilon V/(n-1))\rfloor$. Os valores de $g_j(c)$ para $j = 2, ..., n$, srimeiro computados para $c = 1$, depois para $c = 1, 2, \cdots$ ate $g_n(c) \leq R$ para algum $c = \hat{c} < (n-1)/\epsilon$, ou $c \geq (n-1)/\epsilon$.

No primeiro caso, um caminho de custo de no mmo

$$\frac{V\epsilon}{n-1}\hat{c} + V\epsilon < V(1+\epsilon)$$

foi encontrado, e segue que $OPT < V(1+\epsilon)$. No segundo caso, cada caminho tem $c' \geq (n-1)/\epsilon$ ou $c \geq V$, entOPT $\geq V$. Assim, o teste funciona como queros.

A complexidade de tempo linomial cara fixado o ϵ plicada em seguida: Tomar a parte inteira de um nmero negativo no intervalo $\{0, \cdots, U\}$ pode ser feito em tempo $O(\lg(U))$ usando busca bina. Arrendondar o custo dos arcos toma tempo $O(mlg(n/\epsilon))$ desde que ncalamos somente os arcos com custo menor que V (o resultado mmo $(n-1)/\epsilon$). Depois executamos $O(n\epsilon)$ iteras do algoritmo acima que novamente toma tempo $O(|E|\lg(n/\epsilon))$. E essa mb complexidade do procedimento de teste inteiro.

Agora namos este teste para chegar a um baseado em escalar e arrendondar: Para aproximar OPT nimeiramente determinamos um limite superior (UB) e um limite inferior (LB). O limite superior UB pode ser setado como a soma das $n-1$ arcos com maior custo, ou o custo da caminho que consome menos recursos. O limite inferior LB pode ser setado como 0 ou o caminho de menor custo.

Se $UB \leq (1+\epsilon)LB$, entUB$a\epsilon$-aproxima de OPT. Suponha que $UB > (1+\epsilon)LB$. Seja V um dado valor $LB < V < UB/(1+\epsilon)$. O procedimento TESTE pode agora ser aplicado para melhorar os limites para OPT. Especificamente, ou LB mentado ou UB minu para $V(1+\epsilon)$. Executando uma sequia de testes, a razUB/LB$podeserreduzida.Umavezquearaztingaovalordeumaconstantepredefinida,digamos2,entm$ aproxima pode ser obtida aplicando-se o algoritmo por programa dinca para o grafo com os custo dos arcos escalados para $\lfloor c_{ij}/(LB/(n-1))\rfloor$. O erro final mmo $\epsilon LB < \epsilon OPT$.

A complexidade de tempo para o ltimo passo (|E|n/ϵ). A redu da razUB/LB$lhoralcanaporbuscabinanointervalo(LB, UB)emescalalogarica.Depoisdecadatestenualizamososlimites.Paragarantirumardareduderazecutam$ $(LB \cdot UB)^{1/2}$. O nmero de testes necessos para reduzir a razara abaixo de 2 ($\lg\lg(UB/LB)$) e cada teste toma tempo $O(|E|n/\epsilon)$. hassin:92 mostrou como computar um valor inteiro pro a $O(UB \cdot LB)^{1/2}$ em tempo $O(\lg\lg(UB/LB))$. Isto da complexidade de tempo, total de $O(\lg\lg(UB/LB)(|E|(n\epsilon) + \lg\lg(UB/LB)))$ para este algoritmo ϵ-aproximado como um para o CSP.

hassin:92 tambostrou uma cuja a complexidade depende somente do nmero de variis e $1/\epsilon$ e possui complexidade de tempo de $O(|E|(n^2/\epsilon)\lg(n/\epsilon))$. O melhor foi obtido por phillips:93 que atingiu a complexidade de tempo de $O(|E|(n/\epsilon) + (n^2/\epsilon)\lg(n/\epsilon))$.

3.7 Ranqueamento de caminhos

O problema de ranqueamento de caminhos, mais conhecido como o problema dos k menores caminhos (KSP – k *shortest path*), consiste me determinar o k-mo menor caminho em um grafo. Este problema foi estudado primeiramente por hoffman:59. Desde ent o problema tem sido massivamente estudado e vas solus foram propostas.

Muitos desses mdos tem complexidade de tempo polinomial para um k fixo. eppstein:94 descreveu um algoritmo com complexidade de tempo $O(m+n\lg n+k)$ que resolve o problema quando ciclos sermitidos. Podemos usar o KSP para resolver o . Neste caso, ignoramos o k e enumeramos os caminhos em ordem necrescente de custo atcontrar um caminho vil, tal abordagem resulta em um algoritmo de complexidade exponencial para o .

O KSP n recomendado para ser usado diretamente para resolver o . Porem, ele tem sido usado com sucesso como sub-problema para mdos mais sofisticados como a relaxa lagrangiana proposta por zang:80. Uma extensa bibliografia para o problema dos k menores caminhos pode ser encontrada em eppstein:12.

3.8 Relaxa Lagrangiana

A seguir vamos apresentar o algoritmo proposto por zang:80, que se utiliza de uma relaxa de um problema de programa linear que modela o com um nico recurso.

Inicialmente, vamos apresentar uma formula para o problema usando programa linear. Nela teremos uma varil x_{uv} para cada $uv \in A$, $x_{uv} = 1$ significa dizer que o arco uv est solu e $x_{uv} = 0$ o contro. Vamos nos referir ao problema abaixo como (P).

minimize c(x) $= \sum_{uv\in A} c_{uv}x_{uv}$

sob as restris $\sum_{vw\in A} x_{vw} - \sum_{uv\in A} x_{uv} = \begin{cases} 1, & parav = \\ 0, & paratodov \in V \setminus \{,\} \\ -1, & parav = \end{cases}$ (1)

$\sum_{uv\in A} r_{uv}x_{uv} \leq \lambda(2)$

$x_{uv} \in \{0,1\},\ uv \in A(3)$

Na formula acima, a restri (3) sponsl por delimitar os possis valores que um componente do vetor x pode assumir. A restri (1), por sua vez, sponsl por garantir que para um vetor x ser solu vil do problema, ele deve "conter" um caminho do vice ao vice . Por fim, a restri

(2) nos garante que o conjunto de arcos induzido por um vetor x vil, nxcede os recursos disponis.

Por conveniia de nota, nemos definir os seguintes termos. Vamos definir $\mathcal{X}$ denotando o conjunto de vetores x que satisfazem as equas (1) e (3), ou seja, vetores x que contm caminho de a . Vamos definir tamb seguinte fun.

$$g(x) = \sum_{uv \in A} r_{uv} x_{uv} - \lambda$$

Com as definis acima, resolver (P) uivalente a resolver o seguinte.

$$c^* = c(x^*) = min\{\ c(x) \mid x \in \mathcal{X} \text{ e } g(x) \leq 0\ \}$$

Agora iremos aplicar a teoria da dualidade lagrangiana (como apresentada, por exemplo, em [?], [?]) como primeiro passo para resolver o . Tendo em vista que o problema lativamente mais simples de resolver quando a restri $g(x) \leq 0$ laxada (sem essa restri, o problema se reduz a caminho mmo simples), nossa estrata serstamente retirar essa "restri complicada" do conjunto de restris e a usarmos como penalidade na fun objetivo (tica essa que ssia da relaxa lagrangiana).

Para qualquer $u \in \mathbb{R}$, definimos a fun lagrangiana.

$$L(u) = \min_{x \in \mathcal{X}} L(u, x), onde L(u, x) = c(x) + ug(x)$$

Perceba que encontrar a solu de $L(u)$ roblema de caminho mmo no grafo original, porom os custos dos arcos alterados para $c_{uv} + ur_{uv}$, $uv \in A$. Temos que $L(u) \leq c^*$ para qualquer $u \geq 0$ (teorema fraco da dualidade), pois

$$g(x^*) \leq 0 \Rightarrow L(u) \leq c(x^*) + ug(x^*) \leq c(x^*) = c^*,$$

o que nos permite usar $L(u)$ como um limite inferior para o problema original. Para encontrarmos o limite inferior mais justo possl, resolvemos o problema dual a seguir. Vamos nos referir ao problema abaixo como (D).

$\mathrm{L}^* = L(u^*) = \max_{u \geq 0} L(u)$

Pode ser que exista uma folga na dualidade (*duality gap*), ou seja, pode ser que L^* seja estritamente menor que c^*. Nos casos que existir essa folga, teremos que trabalhar um pouco mais para elimina.

Vamos, agora, descrever um mdo para resolver o programa (P), que usa como passo, resolver o problema (D). Por praticidade vamos denotar $x(u)$ como um caminho que possui valor o associado n $L(u)$.

O mais natural e, como primeiro passo, verifiquemos se o menor caminho (nimitado, $\min_{x \in \mathcal{X}} c(x)$) respeita nossas restris. Vamos chamar esse caminho de $x(0)$, pois $L(0) = c^*$.

- Se $g(x(0)) \leq 0$, entx(0)*aramenteumasoluade*(P).*Sen*x(0)*nosserve, pelomenos, comolimiteinferiorparaasolu.*

Como segundo passo, devemos verificar se o caminho que consome menor quantidade de recursos ($\min_{x \in \mathcal{X}} g(x)$) respeita nossas restris. Vamos chamar esse caminho de $x(\infty)$, pois para valores muito grandes de u, o partro $c(x)$ na fun $L(u)$ dominado" por $ug(x)$.

- Se $g(x(\infty)) > 0$, o problema nem solu, pois o caminho que consume a menor quantidade de recursos consome uma quantidade maior do que o limite.
- Sen $x(\infty)$ a solu vil para a instia e nos serve de limite superior para problema.

Agora com os resultados dos passos anteriores, se nemos ainda a solu ou a prova de que a instia vil, temos a seguinte situa: Dois caminhos, $x(0)$, que **n solu** e **limite inferior** e $x(\infty)$, que **lu vil** e **limite superior**, $g(x(0)) > 0$ e $g(x(\infty)) \leq 0$.

Da forma como desenvolvemos a solu att podemos interpretar cada caminho no grafo como uma reta no espa(u,L)*daforma*L = c(x) + u g(x), *ondeussavaril*,c(x)*ssotermoindependente*(*pontoondearetacortaoeixo*L)eg(x)*ssocoeficienteangular.I*

Com a interpreta geomica dos caminhos, temos a informa que retas **crescentes** sssociadas a caminhos **niis** para o nosso problema, enquanto as retas **nrescentes** ss viis. Como estamos procurando o valor de L^* (o ponto "mais alto" da fun $L(u)$) vamos analisar o ponto (u', L') que ntercessas retas associadas a $x(0)$ e $x(\infty)$.

$$u' = (c(x(\infty)) - c(x(0)))/(g(x(0)) - g(x(\infty)))$$

$$L' = c(x(0)) + u' \cdot g(x(0))$$

ato que $u' \geq 0$, pois $c(x(0))$ nimo, $g(x(\infty)) \leq 0$ e $g(x(0)) > 0$. Claramente, se existem apenas dois caminhos o ponto (u', L') ue maximiza $L(u)$. O mesmo acontece quando existem vos caminhos e $L(u') = L'$, ou seja, $L(u', x) \geq L'$ para qualquer $x \in \mathcal{X}$. Um ltimo caso "especial" ando existe um caminho $x_h \in \mathcal{X}$ tal que $g(x_h) = 0$ e $L(u') = L(u', x_h) < L'$. Como a reta associada a x_h rizontal, ela limita superiormente $L(u)$, e como temos o ponto $(u', L(u'))$ sobre ela, $c^* = c(x_h) = L^* = L(u')$ (neste caso nxiste folga na dualidade).

Falamos, especificamente, sobre os caminhos $x(0)$ e $x(\infty)$ no parafo anterior, mas o que foi dito vale no caso geral, onde temos dois caminhos disponis $x^+, x^- \in \mathcal{X}$, tal que $g^+ \equiv g(x^+) >$

0, $g^- \equiv g(x^-) \leq 0$ e $c^- \equiv c(x^-) \geq c^+ \equiv c(x^+)$. Ent temos que $u' = (c^- - c^+)/(g^+ - g^-)$ e $L' = c^+ + u'g^+$ definem o ponto de intercess no espa(u,L), $dasretasassociadasaoscaminhos$x$^+$ e x^-. Se $L(u') = L'$ ou se $g(x(u')) = 0$, entL(u*) $= L(u')$ olu do nosso problema dual (D). Caso contro, se $g(x(u')) < 0$, entx(u')$ossonovocaminho$x$^-$, e se $g(x(u')) > 0$, entx(u')$ossonovocaminho$x$^+$. O procedimento se repete atterminarmos a solu do problema (D). Com a realiza do procedimento temos disponis um limite inferior LB (*lower bound*) e um limite superior UB (*upper bound*) para o valor de c^*. Nmos que $LB = L(u^*) \leq c(x^*)$ (pelo teorema fraco da dualidade); e por defini segue que qualquer x^- usado durante o procedimento a solu vil, assim UB alor do ltimo c^- ou o valor de $c(x(u'))$ associado com o ltimo caminho $x(u')$ **se** $g(x(u')) \leq 0$.

Tendo resolvido o problema (D), temos limites $LB \leq c^* \leq UB$ e uma solu vil associada a UB para o . Quando $LB = UB$, esta solu ima. Por quando $LB < UB$ temos um folga na dualidade. Para eliminarmos essa folga poderos considerar usar um algoritmo de k-mo menor caminho (k-*shortest path*) a partir do primeiro caminho x tal que $c(x) \geq LB$ atrimeiro x_k tal que $g(x_k) \leq 0$. Como esse algoritmo precisa do conhecimento de todos os caminhos anteriores para gerar o pro, essa abordagem nomaria nenhum proveito da resolu do dual. Em contraste, determinar o k-mo menor caminho em rela a fun lagrangiana $L(u^*, x)$ (o que uivalente a usar a fun c' como custo, $c'_{uv} = c_{uv} + u^* \cdot r_{uv}$, $uv \in A$) rfeitamente aplicl a partir da solu dual.

Vamos denotar $L_k(u^*)$, para $k = 1, 2, \ldots$, como sendo o valor do k-mo menor caminho $x_k \in \mathcal{X}$ em rela a fun de custo $L(u^*, x)$. Os caminhos x_1 e x_2 jo conhecidos, eles sx$^+$ e x^- respectivamente, pois se interceptam no ponto $(u^*, L(u^*))$, o que significa que possuem valor mmo em rela a fun $L(u^*, x)$. Iterando sobre o k-mo caminho, $k \geq 3$, nualizamos UB quando $g(x_k) \leq 0$ e $c(x_k) < UB$; e atualizamos $LB = L_k(u^*)$, pois essa a sequia necrescente $(L_{k-1}(u^*) \leq L_k(u^*))$. O procedimento continua ate $LB \geq UB$, e entemos a solu do problema (P), associada a $UB = c^*$, solu do .

-LAGRANGIANA(G, , , $k = 1$, r, λ, c)

▷ Inicializa

$x_0, c_0, g_0 \leftarrow L(0)$

$g_0 \leq 0$

então $x^*, c^* \leftarrow x_0, c_0$

$x^+, c^+, g^+ \leftarrow x_0, c_0, g_0$

$x_\infty, c_\infty, g_\infty \leftarrow L(\infty)$

$g_\infty > 0$

então $x^*, c^* \leftarrow NULL, NULL$ ▷ Nem solu!

$x^-, c^-, g^- \leftarrow x_\infty, c_\infty, g_\infty$

▷ Resolvendo o Dual

$x^+ \neq NIL$ e $x^- \neq NIL$ ▷ Se entrou nos dois "ent' acima

$LB \leftarrow 0$; $UB \leftarrow c^-$

$LB < UB$ **faça**

$u' \leftarrow (c^- - c^+)/(g^+ - g^-)$; $L' \leftarrow c^+ + u'g^+$; $x', c', g' \leftarrow L(u')$

$g' = 0$

então $x^*, c^* \leftarrow x', c'$; $LB \leftarrow UB \leftarrow c'$

PRA o enquanto

$L(u') = L'$ e $g' < 0$

então $LB \leftarrow L'$; $UB \leftarrow \min\{UB, c'\}$; $x^- \leftarrow x'$; $u^* \leftarrow u'$

PRA o enquanto

$L(u') = L'$ e $g' > 0$

então $LB \leftarrow L'$; $u^* \leftarrow u'$

PRA o enquanto

$L(u') < L'$ e $g' > 0$

então $x^+, c^+, g^+ \leftarrow x', c', g'$

$L(u') < L'$ e $g' < 0$

então $x^-, c^-, g^- \leftarrow x', c', g'$; $UB \leftarrow \min\{UB, c'\}$

▷ Eliminando a folga da dualidade

$x_1, x_2 \leftarrow x^+, x^-$; $k \leftarrow 2$

$LB < UB$ **faça**

$k \leftarrow k + 1$; $x_k, c_k, g_k \leftarrow L_k(u^*)$; $LB \leftarrow L_k(u^*)$

$g_k \leq 0$ e $c_k < UB$

então $x^-, UB \leftarrow x_k, c_k$

$LB \geq UB$

então $x^*, c^* \leftarrow x^-, UB$

x^*, c^*

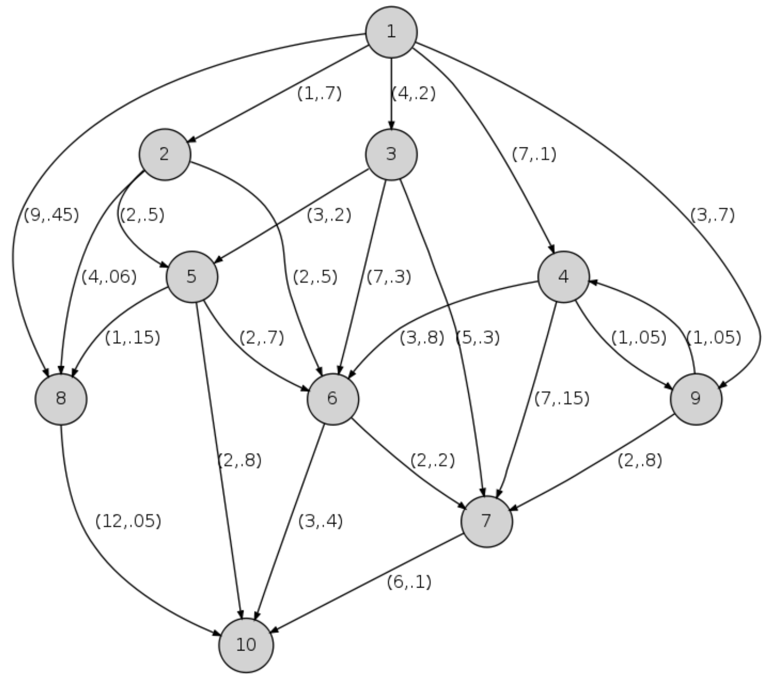


Figura 3.3: *Grafo exemplo; os ros dos arcos representam* (c_{uv}, r_{uv}).

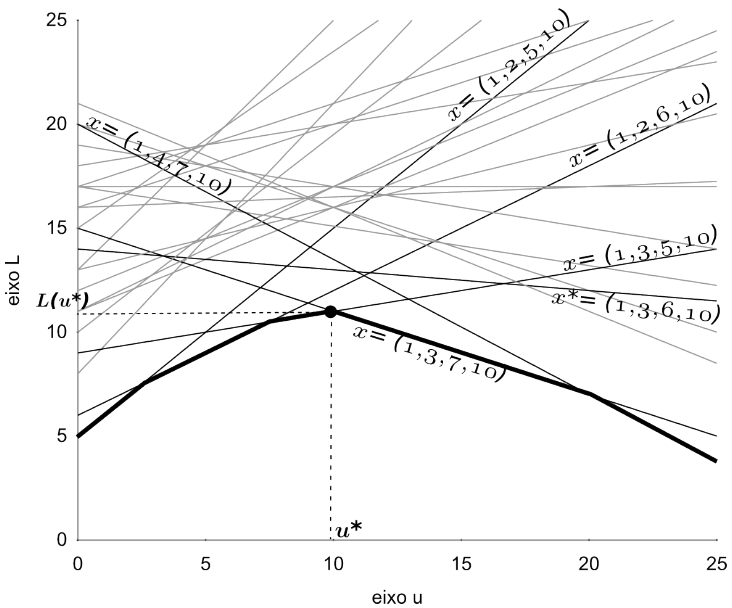


Figura 3.4: *Representa geomica do grafo da Figura 3.3. As retas pretas representam os caminhos que selevantes ao algoritmo. A "curva" de segmentos mais espessos representa o fun* $L(u)$.

4

Experimentos

No presente caplo, iremos fazer experimentos com alguns dos diferentes mdos propostos para o problema de caminhos mmos com recursos limitados. Por uma queste praticidade, todas as implementas sara a verso problema com um nico recurso. Como a performance dos algoritmos ril para diferentes tipos de grafos, nemos experimentos com dados reais e randmicos usando os seguintes tipos de grafos: grafos em grade, grafos de ruas, grafos de curva de aproxima, e grafos aleats.

4.1 Ambiente computacional

Todas as nossas experiias foram executadas em um *laptop Sony Vaio*, com processador *Intel Core i3 CPU M 330 @ 2.13GHz* e 2GB de mem RAM. O sistema operacional utilizado buntu, vers12.04 LTS 64 bit, kernel 3.2.0-27-generic. Os cos foram implementados usando a linguagem de programa C++ e compilados usando o compilador g++ da GNU, vers4.6.3.

4.2 Dados de Teste

Namos os seguintes quatro tipos de grafos:

DEM: Modelos digitais de eleva (*digital elevation models*) srafos em forma de grade onde cada vice tem um valor de altura associado. Usamos exemplos de modelos de elevas da Europa cedidos por Mark Ziegelmann ([**?**]).

Em nossos exemplos DEM, temos que arcos si-direcionados, ou seja, m roximadamente $4n$. Utilizamos o valor absoluto das diferen de altura dos vices como fun custo, esses valores esto intervalo $[0, 600]$. Namos inteiros aleats dentro do intervalo $[10, 20]$ como consumo de recursos. Por fim, estamos interessados em minimizar a diferene altura acumulada no caminho com limita do comprimento do caminho.

	Nmero de Vertices	Nmero de Arcos
Austria grande	41600	165584
Austria pequeno	11648	46160
Esc grande	63360	252432
Esc pequeno	16384	65024

Tabela 4.1: *Casos de teste do tipo* ***DEM***

ROAD: Temos exemplo de grafos de ruas dos Estados Unidos fornecidos por Mark Ziegelmann. Os arcos modelando ruas sovamente bi-direcionados, e a estrutura nos d$aproximadamente$2.5n.$Namosumindicequeavaliaocongestionamentocomofundecusto.Definimososc$

	Nmero de Vertices	Nmero de Arcos
Road 1	77059	171536
Road 2	24086	50826

Tabela 4.2: *Casos de teste do tipo ROAD*

CURVE: Nos problema de curvas de aproxima neremos aproximar uma fun linear por partes (definida no caplo de introdu) por uma nova curva com menos pontos de quebra. Isto ito importante para problemas de compresse dados em as como cartografia, computa grca, e processamento de sinais.

Assumindo que os pontos de quebra na curva dada ocorrem na ordem $v_1, v_2, \cdots, v_n$, namos os pontos de quebra como vices e adicionamos arcos $v_i v_j$ para cada $i < j$. O custo dos arcos ribu como um erro de aproxima que troduzido por tomar o atalho ao inva curva original.

BC: beasley:89 disponibilizaram 24 casos de teste para o problema. Os dados foram gerados de forma randmica e contt00$vicese$4800$arcos.Paramaisinformasarespeitodosdados,recomendamosaleituradoartigoorigin$

	Nmero de Vertices	Nmero de Arcos
Curva 1	10000	99945
Curva 2	10000	199790
Curva 3	1000	9945
Curva 4	1000	19790
Curva 5	5000	49945
Curva 6	5000	99790

Tabela 4.3: *Casos de teste do tipo* ***CURVE***

	Nmero de Vices	Nmero de Arcos
1	100	955
2	100	955
3	100	959
4	100	959
5	100	990
6	100	990
7	100	999
8	100	999
9	200	2040
10	200	2040
11	200	1971
12	200	1971
13	200	2080
14	200	2080
15	200	1960
16	200	1960
17	500	4858
18	500	4858
19	500	4978
20	500	4978
21	500	4847
22	500	4847
23	500	4868
24	500	4868

Tabela 4.4: *Casos de teste do tipo beasley:89,*

4.3 Resultados

O que primeiro se percebe, e os algoritmo nada triviais de serem implementados. Existe uma grande quantidade de fatores relevantes a implementa. Implementamos apenas a programa dinca primal, o algoritmo de Yen para ranqueamento de caminhos e o algoritmo proposto por Hander e Zang.

Dentre todos os casos de teste que utilizamos, apenas os 24 casos de beasley:89 ofereceram boas comparas entre os algoritmos. Isto aconteceu porque os demais casos de teste eram muito grandes, e o fato de as nossas implementas nerem terformcas, nos impossibilitou de usar tais casos de teste de uma melhor forma. Devido a pouca mem da mina usada para os testes, aconteciam travamentos por exemplo.

Logo abaixo vemos os resultados obtidos com a execu das nossas implementas usando as entradas de beasley:89. O algoritmo proposto por Handler e Zang nos surpreendeu bastante com uma a performance de tempo e com uso moderado de mem.

	PDP t(s)	PDP m(KB)	Yen t(s)	Yen m(KB)	HZ t(s)	HZ m(KB)
1	0.06	23248	0.01	6336	0.00	7088
2	0.07	23248	0.02	6336	0.00	7072
3	0.06	23152	0.01	6288	0.00	7056
4	0.07	23152	0.02	6320	0.00	6656
5	0.07	23376	0.01	6352	0.01	7104
6	0.08	23344	0.01	6352	0.01	7104
7	0.06	23168	0.01	6320	0.00	7088
8	0.07	23168	0.01	6336	0.00	7088
9	0.06	23264	0.08	6544	0.01	8496
10	0.07	23264	0.08	6544	0.00	8512
11	0.07	23312	0.01	6496	0.00	7456
12	0.07	23296	0.02	6496	0.00	7456
13	0.07	23376	0.02	7456	0.01	8304
14	0.07	23360	0.02	7456	0.01	8320
15	0.06	23312	0.02	6528	0.00	7456
16	0.07	23296	0.02	6512	0.00	7456
17	0.16	25120	0.03	9536	0.02	13296
18	0.14	24928	0.04	9536	0.02	13296
19	0.07	23728	0.04	9584	0.02	13408
20	0.07	23696	0.05	9584	0.01	11296
21	0.11	24272	0.04	9520	0.02	13280
22	0.10	24208	0.04	9520	0.02	13280
23	0.07	23680	0.04	9520	0.01	13360
24	0.07	23696	0.04	9520	0.01	13344

Tabela 4.5: *Tempo de execu para os testes BC*
Usamos PDP para denotar programa dinca primal. Usamos HZ para denotar o algoritmo de Handler e Zang. As colunas que possuem $t(s)$ representam o consumo de tempo em segundos. As colunas que possuem $m(KB)$ contem o consumo de mria em kilobytes.

Dado estes resultados, nossos pros passos em continuidade a este trabalho servisar todos os cos dando uma maior aten a detalhes de implementa que diminuem constantes e consumo de mem. Outra coisa essencial a se implementar ns redus das instias sempre que possl, nodemos esquecer o problema de caminhos mmos com recursos limitados problema nolinomial, e qualquer corte que seja, pode trazer grandes benefos a eficiia das implementas.

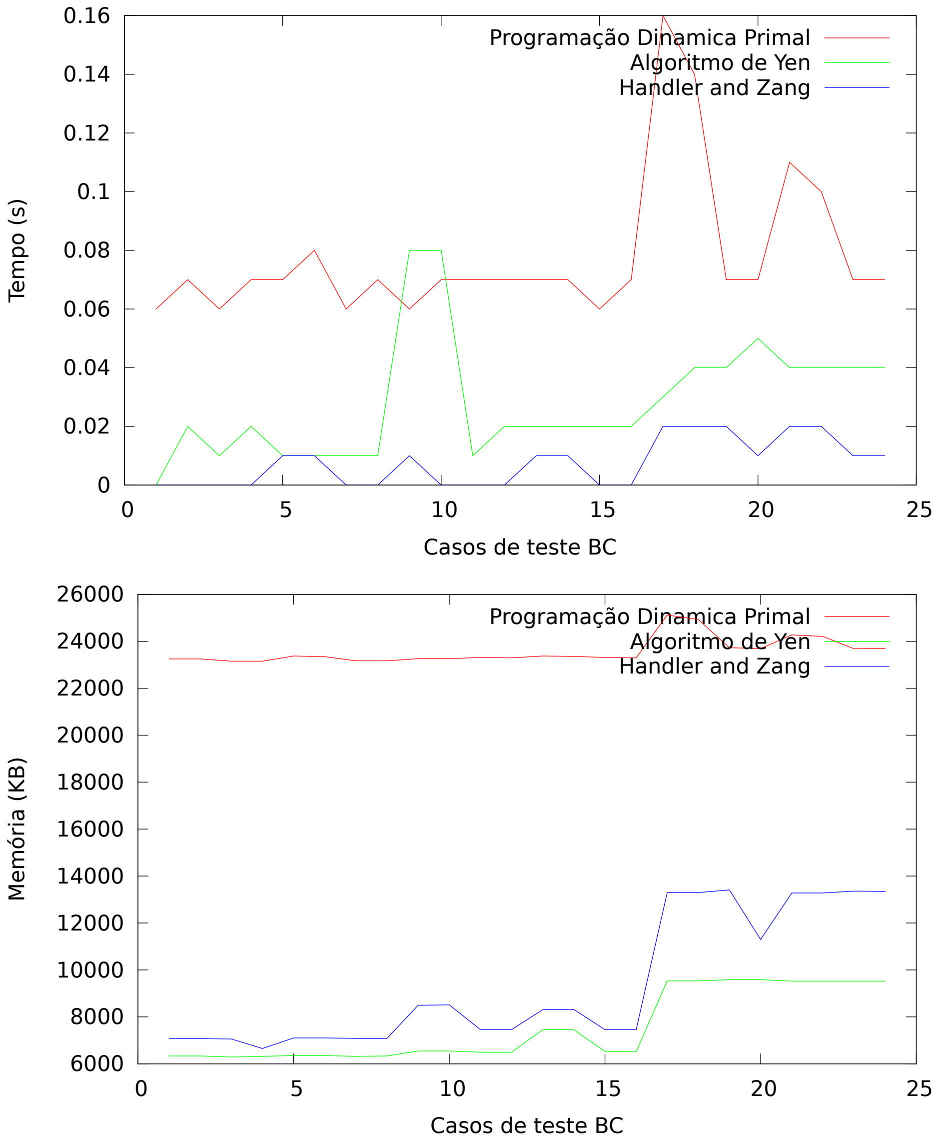


Figura 4.1: *Grcos comparando consumo de mem e tempo dos algoritmo de programa dinca primal, algoritmo de Yen e algoritmo de Handler e Zang.*

5

Conclusão