RICARDO BOVETO SHIMA

CAMINHOS MÍNIMOS SOB RESTRIÇÕES: UMA REVISÃO E APLICAÇÕES

FLORIANÓPOLIS

2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

CAMINHOS MÍNIMOS SOB RESTRIÇÕES: UMA REVISÃO E APLICAÇÕES

Dissertação submetida à Universidade Federal de Santa Catarina
como requisito parcial à obtenção do grau de

MESTRE EM ENGENHARIA ELÉTRICA

por

RICARDO BOVETO SHIMA

Florianópolis, Junho de 2006

Caminhos Mínimos sob Restrições: Uma Revisão e Aplicações

Ricardo Boveto Shima

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Engenharia** na especialidade **Engenharia Elétrica**, área de concentração **Automação e Sistemas**, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da UFSC.

Florianópolis, Junho de 2006.

Prof. Eduardo Camponogara, Ph.D. (orientador)

Prof. Nelson Sadowski, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica
da Universidade Federal de Santa Catarina.

Banca Examinadora:

Prof. Eduardo Camponogara, Ph.D. (orientador)

Prof. Jean-Marie Farines, Dr.

Prof. João Bosco Mangueira Sobral, Dr.

Prof. Rômulo Silva de Oliveira, Dr.

Dedico aos meus pais, Antonio e Sirlei.

Agradecimentos

Gostaria de agradecer a todos que contribuíram diretamente ou indiretamente para realização desta dissertação de mestrado.

Agradeço principalmente ao meu orientador Eduardo por todo o apoio, incentivo, paciência e empenho dedicado a mim e a este trabalho. Além de ter sido um ótimo orientador, foi um ótimo professor, que sempre admirei por sua dedicação e pela paixão por seu trabalho.

Agradeço ao Programa de Pós-Graduação em Engenharia Elétrica da UFSC por sua excelência em seus cursos. Aos professores do DAS que agregaram conhecimento importante na minha vida profissional e pessoal. E ao LCMI por ter me abrigado durante o período do mestrado.

Agradeço também aos colegas de trabalho da Dígitro, que me passaram importantes conhecimentos e que também valorizaram este trabalho. Agradeço aos amigos que contribuíram em muito, com apoio e amizade.

E um final agradecimento aos meus familiares, em especial aos meus pais, meu irmão, minha avó e minha bisavó que sempre me apoiaram incondicionalmente não apenas no mestrado, mas durante toda a minha vida.

Resumo

O problema clássico de caminhos mínimos pode ser resolvido com algoritmos conhecidos de tempo de execução polinomial. Mas quando restrições de tempo ou recursos são impostas ao problema, este se torna *NP*-Difícil. Este problema é conhecido como problema de caminhos mínimos sob restrições e são estudados neste trabalho a sua modelagem, algoritmos e aplicações. O algoritmo de programação dinâmica e o de aproximação-ε são os métodos de maior destaque, que nesta dissertação foram implementados e exemplificados. No presente trabalho são apresentados aplicação em diversos contextos: em agentes móveis, em tráfego urbano e em redes de computadores. No contexto de agentes móveis este tipo de algoritmo pode resolver problemas como o escalonamento de tarefas com restrição de tempo de execução, ou seja, com um *deadline*. Em redes de tráfego veicular pode auxiliar na resolução de um problema mais complexo, ou simplesmente obter o caminho mais rápido com restrições orçamentárias. E por fim, em rede de computadores pode criar rotas que satisfazem métricas de Qualidade de Serviço (QoS).

Abstract

The classical shortest path problem can be solved with polynomial time algorithms. But when constraints in time or resources are imposed, it becomes *NP*-Hard. This problem is known as constrained shorthest path problem and is studied in this work its formulation, algorithms and applications. The dynamic programming algorithm and the ε-approximation scheme are implemented and illustrated in this dissertation. This class of problems has applications in different contexts: in mobile agents, urban traffic and computer networks. In the context of mobile agents this kind of algorithm can find optimal schedules for task with deadlines. In urban traffic, it can assist the resolution of a more complex problem, or simply provide the shortest paths with budget constraints. In computer networks, it can create routes which satisfy quality-of-service (QoS) parameters.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Parte I

Introdução

O problema de caminhos mínimos é um dos problemas fundamentais da computação. Vem sendo estudado com profundidade e eficientes algoritmos de tempo polinomial são conhecidos, como o algoritmo de Dijkstra [1]. Esta classe de problemas é frequentemente colocada em prática em uma grande variedade de aplicações nas quais se deseja realizar algum tipo de transporte entre dois ou mais pontos específicos em uma rede, na qual se deseja fazer com o menor custo, ou menor tempo ou o mais viável possível.

Entretanto, na prática, muitas vezes se deseja não apenas no menor custo ou no menor tempo, mas uma combinação de diferentes critérios, por exemplo, um caminho que seja rápido e barato. Este é conhecido como problema de caminhos mínimos multi-objetivo. Como não é possível otimizar sobre todos os critérios de uma vez, é feita uma escolha do peso de cada critério na função que se deseja minimizar [2].

Uma outra variação do problema de caminhos mínimos é quando há limites de recursos ou orçamento. Desta forma é feita a escolha da função que se deseja minimizar e as limitações de recursos e orçamento são colocadas como restrições. Este é chamado de problema de caminhos mínimos sob restrições, também chamado RCSP *(resource constrained shortest path problem)* [3], o qual será estudado neste trabalho.

Este trabalho é um estudo do problema de caminhos mínimos sob restrições e algoritmos para aplicações computacionais encontradas no roteamento de fluxos em redes de computadores, em escalonamento de tarefas de agentes móveis e roteamento de veículos em redes de tráfego urbano.

Em contextos diversos são encontrados problemas de cunho teórico e prático que podem ser formulados como problemas de caminhos mínimos sob restrições, contudo este elo nem sempre é evidente, fazendo com que métodos sub-ótimos e heurísticas sejam empregados no lugar de algoritmos eficientes, capazes de produzir soluções comprovadamente ótimas.

Este trabalho foi realizado através da revisão da literatura sobre modelos e algoritmos de caminhos mínimos sob restrições. Dentre eles, salienta-se os métodos de programação dinâmica, aproximações baseadas em programação dinâmica e métodos de programação inteira.

Algoritmos foram implementados, ilustrados e experimentos numéricos foram conduzidos para avaliar o comportamento e eficiência destes. A formulação de problemas encontrados em itinerários de agentes móveis foi realizada e resultados computacionais foram obtidos. Para problemas encontrados em redes de computadores e redes de tráfego urbano veicular aplicações foram discutidas.

Objetivos

Dentre os objetivos desta dissertação podemos citar:

- apresentar de forma integrada e unificada diferentes algoritmos para o problema;

- implementar algoritmos eficientes que resolvam este problema;
- avaliar o desempenho dos algoritmos;
- modelar formalmente o problema em programação matemática e aplicá-lo em ferramentas de otimização já existentes; e
- ilustrar algumas aplicações do problema de caminhos mínimos sob restrições.

Organização

O trabalho é organizado da seguinte forma.

A Parte II desta dissertação apresenta o problema de caminhos mínimos sob restrições e faz um breve histórico das soluções. Em seguida mostra a sua modelagem matemática, demonstra e ilustra algoritmos que resolvem o problema e mostra os resultados numéricos. Posteriormente é feita uma discussão sobre os resultados obtidos. Além disso, é discutido quando são impostas restrições diferentes do convencional, como restrições probabilísticas ou não aditivas.

A Parte III é voltada para as aplicações do problema de caminhos mínimos sob restrições. Primeiramente é demonstrado como se resolve o problema de escalonamento de tarefas de agentes móveis com restrição de tempo. Em seguida, é feita uma explanação de como é utilizado este problema em redes de tráfego veicular para solucionar um problema mais complexo. Posteriormente, é mostrado o problema sob a ótica de redes de computadores, sendo aplicado para a criação de rotas que satisfazem métricas de QoS.

Finalmente, na Parte IV traz as conclusões e sugere trabalhos futuros.

Parte II

Problema de Caminhos Mínimos sob Restrições e uma Revisão

Esta Parte da dissertação traz um breve histórico sobre o problema de caminhos mínimos sob restrições. Traz um modelo formal em programação matemática do problema e demonstra que é de difícil tratamento. Posteriormente traz a implementação do algoritmo de programação dinâmica, os resultados numéricos e faz uma comparação dos resultados. E então finaliza esta Parte com discussões sobre restrições não aditivas e restrições probabilísticas.

Capítulo 1

Histórico

Os trabalhos que tratam do problema de caminhos mínimos sob restrições podem ser classificados em quatro tipos [4]:

- trabalhos que tratam um recurso e uma restrição;
- trabalhos que tratam um recurso e duas restrições, impondo um limite mínimo e máximo de consumo;
- trabalhos que tratam de problemas multi-objetivos, gerando um caminho "eficiente"; e
- trabalhos que tratam vários recursos.

O terceiro tipo de trabalho que trata de problemas multi-objetivos tem duas ou mais funções objetivo, na qual podem ser atribuídos pesos a cada uma das funções objetivo. Já no último tipo de problema, que trata de vários recursos, existem restrições associadas a cada um dos recursos existente no problema.

Este trabalho irá focar primeiramente o problema de caminhos mínimos com uma restrição. Depois trata problemas com vários recursos através de programação matemática.

Considerando o primeiro tipo de problema, Joksch [3] apresentou uma solução para um recurso baseada em programação dinâmica. Hassin [5] aplicou a técnica padrão de arredondamento e normalização (*rounding and scaling*), obtendo um algoritmo de aproximação-ε completamente polinomial.

Handler e Zang [6] apresentaram um algoritmo baseado em relaxação Lagrangeana para um recurso. Beasley e Christofides [4] apresentaram um algoritmo também baseado em relaxação Lagrangeana para múltiplos recursos.

Um trabalho mais recente é apresentado por Ziegelmann [2] e Mehlhorn [7] que trata o problema com um método de dois passos, primeiro faz uma relaxação para encontrar os limites e em seguida encontra o resultado exato.

Capítulo 2

O Problema de Caminhos Mínimos sob Restrições

O problema de caminhos mínimos sob restrições será apresentado em um modelo formal na próxima seção. Nesta definição serão utilizados conceitos de grafos, de conjuntos e de programação matemática.

2.1 Definição

Dado um grafo $G = (E, V)$, onde E é o conjunto de arestas e V é o conjunto de vértice, se deseja partir do vértice inicial s e chegar ao vértice final t, com o menor custo possível e dentro das restrições de consumo de recursos.

Assim o problema de caminhos mínimos sob restrições pode ser modelado em progra-

mação matemática conforme o seguinte modelo:

$$P: \text{ Min } \quad f = \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{2.1a}$$

S. a :

$$\sum_{\{(i,j)\in E\}} r_{ij}^{k}x_{ij} \leqslant \lambda^{k}, \quad \text{para } k = 1,...,K \tag{2.1b}$$

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = 0, \quad \forall i \in V - \{s,t\} \tag{2.1c}$$

$$\sum_{\{j:(s,j)\in E\}} x_{sj} = 1 \tag{2.1d}$$

$$\sum_{\{i:(i,t)\in E\}} x_{it} = 1 \tag{2.1e}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{2.1f}$$

onde:

- V corresponde aos vértices;
- E corresponde às arestas;
- c_{ij} é o custo de ir do vértice i para o vértice j, onde $c_{ij} \in \mathbb{Z}_{+}\backslash\{0\}$;
- x_{ij} é a variável binária de decisão, recebendo 1 se a aresta (i,j) é escolhida para o caminho e 0 caso contrário;
- r_{ij}^{k} é o consumo de recurso k para ir do vértice i diretamente ao vértice j, onde $r_{ij} \in \mathbb{Z}_{+}\backslash\{0\}$;
- λ^{k} é a quantidade disponível do recurso k;
- s é o vértice inicial;
- t é o vértice final; e
- K é o número de recursos.

Na formulação matemática do problema, primeiramente é fornecido a função objetivo (2.1a), na qual se deseja minimizar o custo de todas as arestas escolhidas no caminho. A equação (2.1b) impõe a restrição de que o consumo de recurso não deve ultrapassar o valor de λ^{k} para cada recurso k. A equação seguinte (2.1c) restringe que todos os vértices terão fluxo de entrada igual ao fluxo de saída, com exceção aos vértices iniciais e finais. E por fim,

as últimas restrições fazem com que o fluxo de saída do vértice inicial seja unitário e o fluxo de chegada ao vértice final também seja unitário.

2.2 Complexidade

O problema de caminhos mínimos sem restrições pode ser resolvido eficientemente em tempo polinomial. Com a inserção de uma restrição o problema passa a ser *NP*-Difícil [8].

Para demonstrar que o RSCP é um problema *NP*-Difícil, podemos reduzir o problema da mochila ao RCSP [8]. No problema da mochila é dado um conjunto de $n-1$ itens, cada um tendo um valor v_j e um peso w_j, para $j = 1,...,n-1$. O objetivo é colocar itens no pacote respeitando o limite λ e que o valor dos itens escolhidos seja maximizado. O problema da mochila pode ser formalmente descrito como:

$$K: \text{ Max} \quad f = \sum_{j=1}^{n-1} v_j x_j \tag{2.2a}$$

$$\text{S. a :}$$

$$\sum_{j=1}^{n-1} w_j x_j \leqslant \lambda \tag{2.2b}$$

$$x_j \in \{0,1\}; \quad j = 1,...,n-1 \tag{2.2c}$$

Para fazer a redução é construída uma rede [2] com n nós com dois arcos paralelos de cada nó j para o nó $j+1$, para $j = 1,...,n-1$. A rede é ilustrada na figura 2.1 para $n = 4$. Seja $c^1_{j,j+1} = M - v_j, r^1_{j,j+1} = w_j$ e $c^2_{j,j+1} = M$, $r^2_{j,j+1} = 0$, os parâmetros para o primeiro e segundo arcos para $j = 1,...,n-1$, respectivamente, onde $M = \text{Max}\{v_j : j = 1,...,n-1\}$. Desta forma pode ser resolvido o problema da mochila encontrando o caminho mínimo, com relação ao parâmetro c, do nó 1 até o nó n, e sujeito a uma restrição, com relação ao parâmetro r, de máximo de recurso λ. Assim se é escolhido o primeiro arco como parte do caminho mínimo, o item correspondente é colocado na mochila. Caso seja escolhido o segundo arco, o item correspondente não é colocado na mochila. Como o problema da mochila é *NP*-Difícil e pode ser reduzido ao problema RCSP, este último também é *NP*-Difícil.

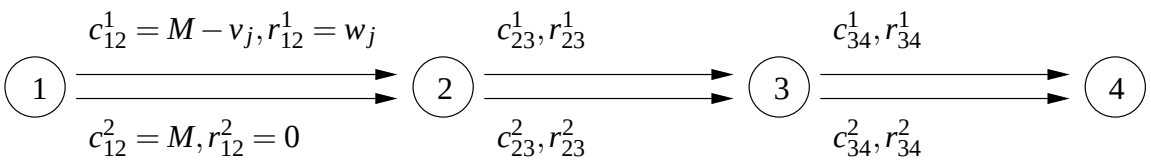

Figura 2.1: Ilustração da complexidade do RCSP

2.3 Aplicações

O problema de caminhos mínimos tem um grande número de aplicações práticas.

O primeiro problema que vem em mente é o planejamento de rotas em redes de tráfego urbano. Quando se deseja ir de um ponto a outro se deseja minimizar as possibilidades de encontrar congestionamentos ao longo do caminho. Ou então, deseja-se ir de um ponto a outro com o menor tempo possível sujeito a restrições orçamentárias de combustível e pedágios.

Em redes de comunicação, torna-se necessário Qualidade de Serviço (QoS) [9]. Neste contexto, se deseja obter um caminho de custo mínimo que atenda a restrições de atraso, limites de banda e de disponibilidade.

Um outro escopo a ser abordado é no contexto de agentes móveis onde pode ser realizado o escalonamento de tarefas com restrições de tempo ou *deadline*. Estas aplicações serão melhor discutidas na Parte III.

2.4 Sumário

Este capítulo apresentou um modelo formal para o problema de caminhos mínimos sob restrições, definindo matematicamente a função objetivo e as restrições do problema. Este modelo serve para ser utilizado com programação matemática. Foi demonstrando que o problema de caminhos mínimos se torna *NP*-Difícil com a inserção de uma restrição. E por fim, foi feito um breve esboço das aplicações do problema, que serão vistas com mais detalhes na Parte III.

Capítulo 3

Programação Dinâmica

O problema de caminhos mínimos se torna *NP*-Difícil com a introdução de apenas uma restrição, por isso existe uma dificuldade em trabalhar com este tipo de problema. Entretanto, existem formas de tratar este problema que são satisfatórias para algumas situações reais. Para isso, são utilizados mecanismos de aproximação e de programação dinâmica.

A programação dinâmica é uma técnica bastante poderosa para resolver determinados tipos de problemas computacionais. Esta técnica consiste em quebrá-los em subproblemas menores, mais fáceis de serem resolvidos [10].

O paradigma de programação dinâmica segue os seguintes passos:

- remova um elemento do problema;
- resolva o problema menor; e
- use a solução do problema menor para adicionar o elemento removido de maneira adequada, produzindo uma solução para o problema maior.

Uma das características da programação dinâmica é que traz a solução ótima para o problema. Para o problema de caminhos mínimos com uma restrição pode ser aplicada programação dinâmica.

Serão mostradas duas formas de se trabalhar com programação dinâmica para este problema, a recursão primal e a dual. Na recursão primal, pode-se encontrar todos os caminhos ótimos para o problema com um limite máximo de recursos, enquanto que na recursão dual, pode-se encontrar todos os caminhos de consumo mínimo de recursos até um limite de gastos. Esta última recursão é bastante interessante para aproximações.

3.1 Recursão Primal

Na recursão primal, têm-se um limite de consumo de recursos. Assim, partindo de uma quantidade unitária de recursos, realizam-se recursões até este limite, obtendo todos os caminhos ótimos para a restrição do problema.

Existem duas formas de tratar esta recursão, a árvore direta e a reversa. A primeira encontra todos os caminhos mínimos partindo do primeiro vértice, enquanto que na segunda árvore, se encontram todos os caminhos mínimos que chegam ao último vértice.

3.1.1 Árvore Direta

Um caminho do nó i ao nó j é chamado de caminho r se tiver um consumo de recursos menor ou igual a r. Deseja-se encontrar o menor caminho λ de 1 até n. Seja $c_j(r)$ o custo do menor caminho r do nó $s = 1$ até o nó j. Então têm-se a seguinte recursão:

$$c_j(r) = \min\{c_j(r-1), \min_{(i,j)\in E, r_{ij}\leqslant r}\{c_i(r-r_{ij}) + c_{ij}\}\}. \tag{3.1}$$

Fazendo $c_1(r) = 0$ para $0 \leqslant r \leqslant \lambda$ e $c_j(0) = \infty$ para todo $j = 2,...,n$, o ótimo do problema de caminhos mínimos sob uma restrição, que é dado por $c_n(\lambda)$, pode ser computado recursivamente [1].

Seja n a quantidade de vértices, q a quantidade de recursos disponíveis, c_{ij} o custo para ir do vértice i até j e r_{ij} o consumo de recursos para ir do vértice i até j, temos o algoritmo 3.1, que resolve o problema de caminhos mínimos sob restrições.

Os valores de π contêm os vértices anteriores a cada vértice e a cada quantidade de recurso disponível, dentro da instância do problema. Estes valores são utilizados para montar o caminho, propriamente dito, de menor custo dentro da restrição imposta.

O tempo de execução deste algoritmo é $O(m\lambda)$ e o consumo de espaço é $O(n\lambda)$, onde $n = |V|$ e $m = |E|$. Ou seja, este algoritmo tem tempo de execução pseudo-polinomial. Caso o tamanho de λ seja da forma k^m para uma constante k o tempo de execução do algoritmo se torna polinomial no tamanho da entrada.

3.1.2 Exemplo

Para ilustrar o funcionamento da recursão de programação dinâmica será utilizada a instância da tabela 3.1. Foi considerada a quantidade de 11 recursos disponíveis para esta ins-

[1] Lembrando que $c_{ij} > 0$

Algoritmo de Programação Dinâmica - Árvore Direta

Para $r = 0$ até λ faça
    $c_1(r) \leftarrow 0$
    $\pi_1(r) \leftarrow 0$
Para $j = 2$ até n faça
    $c_j(0) \leftarrow \infty$
    $\pi_j(0) \leftarrow 0$
Para $r = 1$ até λ faça
    Para $j = 2$ até n faça
        $c_j(r) \leftarrow c_j(r-1)$
        $\pi_j(r) \leftarrow \pi_j(r-1)$
        Para todo $i \in \{i : (i, j) \in E\}$ faça
            Se $(r_{ij} \leqslant r)$ e $(c_i(r - r_{ij}) + c_{ij} < c_j(r))$ faça
                $c_j(r) \leftarrow c_i(r - r_{ij}) + c_{ij}$
                $\pi_j(r) \leftarrow i$

Algoritmo 3.1: Programação Dinâmica - Árvore Direta

tância.

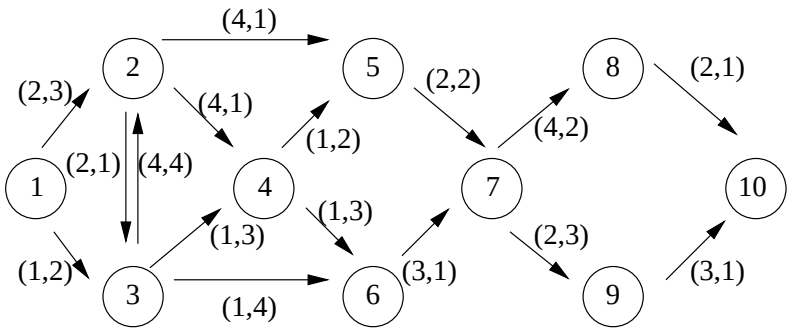

Figura 3.1: Grafo do exemplo

Partindo da recorrência (3.1) podemos obter a tabela 3.2. Para auxiliar a geração do caminho ótimo para o problema foi gerada uma tabela de vértices anteriores, tabela 3.3. Desta forma, é possível montar o caminho ótimo que começa no primeiro vértice e chega a qualquer outro vértice da instância.

Para montar o caminho ótimo a partir da tabela de vértices anteriores pode ser utilizado o algoritmo 3.2.

Ao final é obtido o caminho mínimo do primeiro vértice ao último vértice. Para encontrar o caminho mínimo do primeiro vértice a um outro qualquer, basta fazer j receber o valor do vértice-destino desejado.

Tabela 3.1: Instância

Origem	Destino	Custo	Consumo do Recurso
1	2	2	3
1	3	1	2
2	3	2	1
2	4	4	1
2	5	4	1
3	2	4	4
3	4	1	3
3	6	1	4
4	5	1	2
4	6	1	3
5	7	2	2
6	7	3	1
7	8	4	2
7	9	2	3
8	10	2	1
9	10	3	1

Tabela 3.2: Programação Dinâmica $c_j(r)$ - Recursos $\times$ Gastos

$c_j(r)\backslash r$	0	1	2	3	4	5	6	7	8	9	10	11
$c_1(r)$	0	0	0	0	0	0	0	0	0	0	0	0
$c_2(r)$	∞	∞	∞	2	2	2	2	2	2	2	2	2
$c_3(r)$	∞	∞	1	1	1	1	1	1	1	1	1	1
$c_4(r)$	∞	∞	∞	∞	6	2	2	2	2	2	2	2
$c_5(r)$	∞	∞	∞	∞	6	6	6	3	3	3	3	3
$c_6(r)$	∞	∞	∞	∞	∞	∞	2	2	2	2	2	2
$c_7(r)$	∞	∞	∞	∞	∞	∞	8	5	5	5	5	5
$c_8(r)$	∞	∞	∞	∞	∞	∞	∞	∞	12	9	9	9
$c_9(r)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	10	7	7
$c_{10}(r)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	14	11	10

Tabela 3.3: Vértices Anteriores ($\pi_j(r)$)

$\pi_j(r)\backslash r$	0	1	2	3	4	5	6	7	8	9	10	11
$\pi_1(r)$	0	0	0	0	0	0	0	0	0	0	0	0
$\pi_2(r)$	-	-	-	1	1	1	1	1	1	1	1	1
$\pi_3(r)$	-	-	1	1	1	1	1	1	1	1	1	1
$\pi_4(r)$	-	-	-	-	2	3	3	3	3	3	3	3
$\pi_5(r)$	-	-	-	-	2	2	2	4	4	4	4	4
$\pi_6(r)$	-	-	-	-	-	-	3	3	3	3	3	3
$\pi_7(r)$	-	-	-	-	-	-	5	6	6	6	6	6
$\pi_8(r)$	-	-	-	-	-	-	-	-	7	7	7	7
$\pi_9(r)$	-	-	-	-	-	-	-	-	-	7	7	7
$\pi_{10}(r)$	-	-	-	-	-	-	-	-	-	8	8	9

Algoritmo para montar o caminho ótimo

$j \leftarrow n$
$r' \leftarrow r$
$p \leftarrow \emptyset$
Enquanto $\pi_j(r') \neq 0$
    $i \leftarrow \pi_j(r')$
    Adicionar (i, j) ao caminho p
    $r' \leftarrow r' - r_{ij}$
    $j \leftarrow i$

Algoritmo 3.2: Caminho ótimo

Para obtermos o caminho mínimo partindo do vértice 1 e chegando ao vértice 10 com 11 recursos disponíveis fazemos os seguintes passos:

Execução do algoritmo para montar caminho ótimo

$r' \leftarrow 11$
$j \leftarrow 10$
Enquanto $(\pi_{10}(11) = 9) \neq 0$
    $i \leftarrow \pi_{10}(11) = 9$
    adicionar $(i, j) = (9, 10)$
    $r' \leftarrow (11 - r_{9,10} = 11 - 1 = 10)$
    $j \leftarrow 9$
Enquanto $(\pi_9(10) = 7) \neq 0$
    $i \leftarrow \pi_9(10) = 7$
    adicionar $(i, j) = (7, 9)$
    $r' \leftarrow (10 - r_{7,9} = 10 - 3 = 7)$
    $j \leftarrow 7$
Enquanto $(\pi_7(7) = 6) \neq 0$
    $i \leftarrow \pi_7(7) = 6$
    adicionar $(i, j) = (6, 7)$
    $r' \leftarrow (7 - r_{6,7} = 7 - 1 = 6)$
    $j \leftarrow 6$
Enquanto $(\pi_6(6) = 3) \neq 0$
    $i \leftarrow \pi_6(6) = 3$
    adicionar $(i, j) = (3, 6)$
    $r' \leftarrow (6 - r_{3,6} = 6 - 4 = 2)$
    $j \leftarrow 3$
Enquanto $(\pi_3(2) = 1) \neq 0$
    $i \leftarrow \pi_3(2) = 1$
    adicionar $(i, j) = (1, 3)$
    $r' \leftarrow (2 - r_{1,3} = 2 - 2 = 0)$
    $j \leftarrow 1$
Enquanto $(\pi_1(0) = 0) \neq 0$
Caminho mínimo = { (1,3), (3,6), (6,7), (7,9), (9,10) }

Como ilustração, pode-se observar pela figura 3.2 os caminhos possíveis com até 11 recursos disponíveis. Na coluna à esquerda desta figura têm-se o consumo de recursos para se chegar até aquele vértice e entre parênteses está o seu custo.

Note que à medida que a quantidade de recursos aumenta, o custo dos caminhos diminui.

Tabela 3.4: Soluções

Quantidade de recurso	Custo mínimo	Caminho
11	10	1 3 6 7 9 10
10	11	1 3 6 7 8 10
9	14	1 2 5 7 8 10

Para o caso do vértice 10, são necessários 9 unidades de recurso para alcançá-lo, tendo um custo de 14 unidades. Com 10 unidades de recurso é possível alcançá-lo com um custo de 11 unidades. E finalmente, com 11 unidades de recurso, pode-se alcançá-lo com um custo de 10 unidades. As soluções do problema podem ser observadas na tabela 3.4.

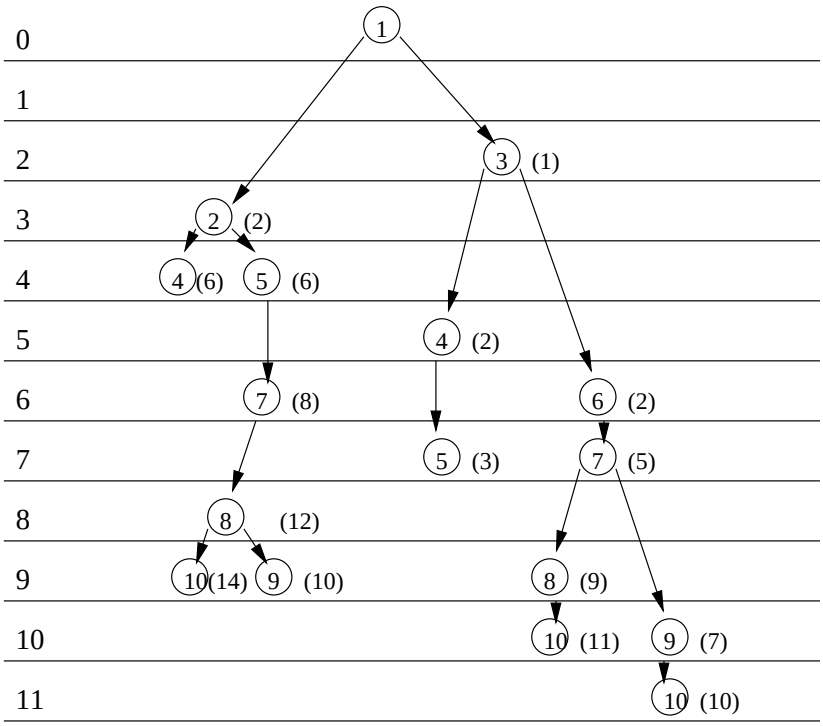

Figura 3.2: Caminhos possíveis com 11 recursos disponíveis

3.1.3 Árvore Reversa

Fazendo uma pequena modificação na recursão (3.1) é possível obter os caminhos mínimos que chegam a um determinado vértice, partindo de qualquer outro vértice.

Da mesma forma que na árvore direta, um caminho do nó i ao nó n é chamado de caminho r se tiver consumo de recursos menor ou igual a r. Deseja-se encontrar o menor caminho λ de 1 até n. Seja $c'_i(r)$ o custo do menor caminho r do nó i até o nó n. Então têm-se a recursão:

$$c'_i(r) = \min\{c'_i(r-1), \min_{(i,j)\in E, r_{ij}\leqslant r}\{c'_j(r-r_{ij}) + c_{ij}\}\}. \tag{3.2}$$

Fazendo $c'_n(r) = 0$ para $0 \leqslant r \leqslant \lambda$ e $c'_i(0) = \infty$ para todo $i = 1, \ldots, n-1$, o ótimo do problema de caminhos mínimos sob uma restrição, que é dado por $c'_1(\lambda)$, pode ser computado recursivamente.

Seja n a quantidade de vértices, λ a quantidade de recursos disponíveis, c_{ij} o custo para ir do vértice i até j e r_{ij} o consumo de recursos para ir do vértice i até j, temos o algoritmo 3.3.

Algoritmo de Programação Dinâmica - Árvore Reversa

```
Para r = 0 até λ faça
    c'_n(r) ← 0
    π_n(r) ← 0
Para i = 1 até n − 1 faça
    c'_i(0) ← ∞
    π_i(0) ← 0
Para r = 1 até λ faça
    Para i = n − 1 até 1 faça
        c'_i(r) ← c'_i(r − 1)
        π_i(r) ← π_i(r − 1)
        Para todo j ∈ {j : (i, j) ∈ E} faça
            Se (r_ij ⩽ r) e (c'_j(r − r_ij) + c_ij < c'_i(r)) faça
                c'_i(r) ← c'_j(r − r_ij) + c_ij
                π_i(r) ← j
```

Algoritmo 3.3: Programação Dinâmica - Árvore Reversa

A complexidade deste algoritmo é $O(\lambda m)$ e consumo de memória é $O(\lambda n)$, onde $n = |V|$ e $m = |E|$.

Na recursão com árvore reversa, a tabela auxiliar contém os vértices posteriores a cada vértice e a cada quantidade de recurso disponível. Assim como na árvore direta, esta tabela é usada para montar o caminho propriamente dito de menor custo dentro da restrição imposta.

Esse tipo de recursão pode parecer menos útil, mas em aplicações onde é necessário recalcular o caminho dinamicamente pode ser mais interessante. Em agentes móveis, durante o percurso, um agente pode consumir mais ou menos recurso (tempo) do que o calculado previamente. Por isso, pode ser interessante obter uma nova rota que melhore o desempenho

Tabela 3.5: Programação Dinâmica - Árvore Reversa - Recursos × Gastos

$c'_j(r)\backslash r$	0	1	2	3	4	5	6	7	8	9	10	11
$c_1(r)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	14	11	10
$c_2(r)$	∞	∞	∞	∞	∞	∞	12	11	11	11	11	11
$c_3(r)$	∞	∞	∞	∞	∞	∞	∞	∞	10	9	9	9
$c_4(r)$	∞	∞	∞	∞	∞	∞	∞	9	8	8	8	8
$c_5(r)$	∞	∞	∞	∞	∞	8	7	7	7	7	7	7
$c_6(r)$	∞	∞	∞	∞	9	8	8	8	8	8	8	8
$c_7(r)$	∞	∞	∞	6	5	5	5	5	5	5	5	5
$c_8(r)$	∞	2	2	2	2	2	2	2	2	2	2	2
$c_9(r)$	∞	3	3	3	3	3	3	3	3	3	3	3
$c_{10}(r)$	0	0	0	0	0	0	0	0	0	0	0	0

Tabela 3.6: Vértices Posteriores - Árvore Reversa

$\pi_i(r)\backslash r$	0	1	2	3	4	5	6	7	8	9	10	11
$\pi_1(r)$	-	-	-	-	-	-	-	-	-	2	3	3
$\pi_2(r)$	-	-	-	-	-	-	5	5	5	5	5	5
$\pi_3(r)$	-	-	-	-	-	-	-	-	6	6	6	6
$\pi_4(r)$	-	-	-	-	-	-	-	5	5	5	5	5
$\pi_5(r)$	-	-	-	-	-	7	7	7	7	7	7	7
$\pi_6(r)$	-	-	-	-	7	7	7	7	7	7	7	7
$\pi_7(r)$	-	-	-	8	9	9	9	9	9	9	9	9
$\pi_8(r)$	-	10	10	10	10	10	10	10	10	10	10	10
$\pi_9(r)$	-	10	10	10	10	10	10	10	10	10	10	10
$\pi_{10}(r)$	0	0	0	0	0	0	0	0	0	0	0	0

dinamicamente, que pode ser obtida diretamente na tabela.

3.1.4 Exemplo

Utilizando o mesmo grafo do exemplo da figura 3.1, a partir da recursão (3.2) podemos obter as tabelas 3.5 e 3.6.

Como ilustração da programação dinâmica com árvore reversa, pode-se observar pela figura 3.3 os caminhos possíveis com os mesmos 11 recursos disponíveis. Na coluna à esquerda desta figura têm-se o gasto de recursos para se chegar até aquele vértice e entre parênteses está o seu custo.

Para ilustrar o aspecto dinâmico da árvore reversa serão mostrados dois casos. O primeiro caso, quando o consumo de recursos é maior do que o esperado para uma aresta. O segundo,

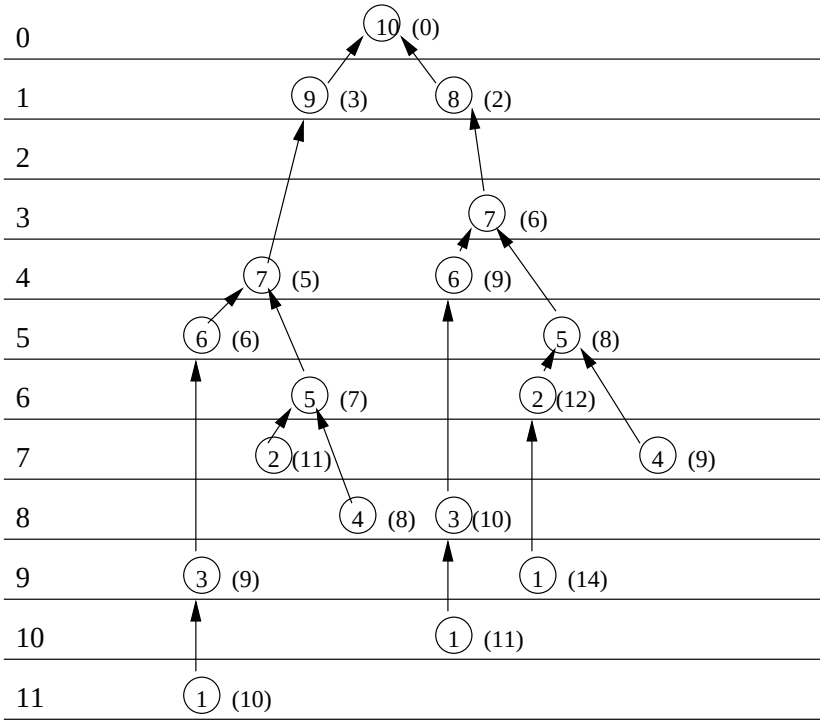

Figura 3.3: Caminhos possíveis com 11 recursos disponíveis - Árvore Reversa

quando o consumo é menor do que o esperado para uma aresta.

O primeiro caso, a instância tem 11 unidades de recurso disponíveis e a solução era $1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow 10$, com um custo de 10 unidades. Durante o percurso na aresta $(1,3)$ foram consumidos 3 unidades do recurso, quando era esperado apenas 2 unidades. Assim se continuasse pelo mesmo caminho os recursos iriam se esgotar, não sendo possível alcançar o vértice 10. Utilizando a tabela dos vértices posteriores (Tabela 3.6) e partindo do vértice 3 com 8 unidades de recursos, podemos chegar ao caminho $3 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 10$. Ao final teremos o custo de 11 unidades, que é maior que o estimado inicialmente, e com o mesmo consumo de 11 unidades de recurso. Mas desta forma os recursos não terão se esgotado durante o percurso.

No segundo caso, a instância tem 9 recursos disponíveis e a solução era $1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 10$, com um custo de 14 unidades. Durante o percurso na aresta $(1,2)$ foi consumido apenas 2 unidades de recurso, quando eram esperados 3 unidades. Seria possível continuar pelo mesmo caminho, mas não seria a solução ótima. Assim, utilizando a tabela dos vértices posteriores (Tabela 3.6) e partindo do vértice 2 com 7 unidades de recursos, podemos chegar ao caminho $2 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 10$. Ao final teremos o custo total de 13 unidades, que é menor que o estimado inicialmente, e com o mesmo consumo de 9 unidades de recurso.

3.2 Recursão Dual

Uma outra forma de utilizar programação dinâmica para o problema de caminhos mínimos sob restrições é a recursão dual [5]. Desta forma, em vez de utilizar a variação na restrição para fazer as recursões, é utilizada a variação dos custos para a elaboração da tabela de programação dinâmica.

Em sua forma básica, a recursão dual não traz grandes vantagens. Mas se torna bastante interessante quando trabalhamos com aproximações.

Um caminho do nó i ao nó j é chamado de caminho c se tiver custo menor ou igual a c. Deseja-se encontrar o menor caminho c^* de 1 até n. Seja $r_j(c)$ o consumo de recursos do menor caminho c do nó 1 até o nó j. Então têm-se a recursão:

$$r_j(c) = \min\{r_j(c-1), \min_{(i,j)\in E, c_{ij}\leqslant c}\{r_i(c-c_{ij}) + r_{ij}\}\}. \tag{3.3}$$

Fazendo $r_1(c) = 0$ para $0 \leqslant c \leqslant c^*$ e $r_j(0) = \infty$ para todo $j = 2,...,n$, o ótimo do problema de caminhos mínimos sob uma restrição, que é dado por $r_n(c^*)$, pode ser computado recursivamente.

Note que o valor de c^* não é conhecido a princípio, mas pode ser conhecido por aproximações ou por condições de parada que serão discutidas mais à frente. A complexidade deste algoritmo é $O(|E|c^*)$.

Dados a quantidade n de vértices, o custo máximo c^* a ser verificado, o custo c_{ij} para ir do vértice i até j e o consumo de recursos r_{ij} para ir do vértice i até j, temos o algoritmo 3.4.

3.2.1 Exemplo

Utilizando o mesmo exemplo da figura 3.1 e utilizando a recursão (3.3), podemos obter as tabelas 3.7 e 3.8. Estas tabelas foram geradas fazendo c^* igual a 14, que foi o maior custo gerado para se chegar ao último vértice, ou seja, qualquer valor maior ou igual a 14 é suficiente para encontrar todas as soluções ótimas do problema.

3.2.2 Condições de Parada (c^*)

Como condição de parada para a recursão dual pode-se utilizar o custo no momento em que for encontrado um valor $r_j(c) \leq \lambda$, ou seja, quando for encontrado um resultado menor ou igual a quantidade de recursos disponível. Isso garante que a solução ótima seja encontrada.

Tabela 3.7: Programação Dinâmica $c_j(r)$ - Recursão Dual - Gastos $\times$ Recursos

$r_j(c)\backslash c$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$r_1(c)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$r_2(c)$	∞	∞	3	3	3	3	3	3	3	3	3	3	3	3	3
$r_3(c)$	∞	2	2	2	2	2	2	2	2	2	2	2	2	2	2
$r_4(c)$	∞	∞	5	5	5	5	4	4	4	4	4	4	4	4	4
$r_5(c)$	∞	∞	∞	7	7	7	4	4	4	4	4	4	4	4	4
$r_6(c)$	∞	∞	6	6	6	6	6	6	6	6	6	6	6	6	6
$r_7(c)$	∞	∞	∞	∞	∞	7	7	7	6	6	6	6	6	6	6
$r_8(c)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	9	9	9	8	8	8
$r_9(c)$	∞	∞	∞	∞	∞	∞	∞	10	10	10	9	9	9	9	9
$r_{10}(c)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	11	10	10	10	9

Tabela 3.8: Vértices Anteriores - Recursão Dual

$\pi_j(c)\backslash c$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\pi_1(c)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\pi_2(c)$	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1
$\pi_3(c)$	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\pi_4(c)$	-	-	3	3	3	3	2	2	2	2	2	2	2	2	2
$\pi_5(c)$	-	-	-	4	4	4	2	2	2	2	2	2	2	2	2
$\pi_6(c)$	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3
$\pi_7(c)$	-	-	-	-	-	6	6	6	5	5	5	5	5	5	5
$\pi_8(c)$	-	-	-	-	-	-	-	-	-	7	7	7	7	7	7
$\pi_9(c)$	-	-	-	-	-	-	-	7	7	7	7	7	7	7	7
$\pi_{10}(c)$	-	-	-	-	-	-	-	-	-	-	9	8	8	8	8

Algoritmo de Programação Dinâmica Dual

Para $c = 0$ até c^* faça
    $r_1(c) \leftarrow 0$
    $\pi_1(c) \leftarrow 0$
Para $j = 2$ até n faça
    $c_j(0) \leftarrow \infty$
    $\pi_j(0) \leftarrow 0$
Para $c = 1$ até c^* faça
    Para $j = 2$ até n faça
        $r_j(c) \leftarrow r_j(c-1)$
        $\pi_j(c) \leftarrow \pi_j(c-1)$
        Para todo $i \in \{i : (i, j) \in E\}$ faça
            Se $(c_{ij} \leqslant c)$ e $(r_i(c - c_{ij}) + r_{ij} < r_j(c))$ faça
                $r_j(c) \leftarrow r_i(c - c_{ij}) + r_{ij}$
                $\pi_j(c) \leftarrow i$

Algoritmo 3.4: Programação Dinâmica Dual

Caso não haja solução para o problema, a recursão poderia continuar buscando uma solução indefinidamente. Assim, para garantir a parada da recursão, caso não exista solução para a instância, é necessário encontrar um limite superior c^*. Este limite garantiria que a solução seja encontra, caso exista, dentro daquele limite. Uma forma simples de calcular c^* pode ser vista na equação:

$$c_{max} = \max\{c_{ij} : (i, j) \in E\} \tag{3.4a}$$

$$c^* = c_{max}(n-1) \tag{3.4b}$$

Nesta equação, é multiplicado o maior custo dentre todas as arestas pelo maior número possível de arestas em um caminho mínimo que é $n-1$, ou seja, passa por todos os vértices.

Uma forma de encontrar um limite superior c^* mais acurado é utilizar o algoritmo de Dijkstra usando o recurso como função das arestas. Assim, c^* seria o custo do caminho de recurso mínimo, obtido na árvore de caminhos mínimos.

3.3 Sumário

Neste capítulo foi discutido a definição de programação dinâmica e como aplicá-la ao problema de caminhos mínimos sob restrição. Os métodos para tratar o problema com programação dinâmica foram divididos em duas formas: recursão primal e dual. Na primal é utilizada a quantidade de recursos disponíveis para ser realizada a iteração do algoritmo. Já na recursão dual é utilizado o custo das arestas como base da iteração.

Estas duas recursões podem ser divididas novamente em duas partes: árvore direta e árvore reversa. A primeira descobre todos os possíveis destinos para um determinado nó origem. Já a árvore reversa encontra todas as possíveis origens para um determinado nó destino.

Por fim, é apresentado um exemplo que serve para ilustrar cada tipo de recursão. E também é feita a discussão das condições de paradas da recursão dual.

Capítulo 4

Estudo Comparativo (Análise Computacional)

As instâncias utilizadas para a experimentação numérica foram algumas das utilizadas por Beasley e Christofides [4].

Para fazer a modelagem em programação matemática foi utilizada a linguagem AMPL [11], que é uma linguagem flexível de modelagem algébrica para problemas de programação linear, não-linear e inteira, geralmente encontrados em otimização.

Para resolver as instâncias foi utilizado o software de otimização CPLEX [12], que resolve uma grande variedade de problemas de programação matemática.

Neste experimento foram utilizados 8 instâncias, descritas na tabela 4.1.

Tabela 4.1: Instâncias

	Vértices	Arestas	Restrições	Recursos
Instância 1	100	955	1	73
Instância 2	100	955	1	65
Instância 3	100	959	1	17
Instância 4	100	959	1	15
Instância 5	200	2040	1	13
Instância 6	200	2040	1	12
Instância 7	500	4858	1	198
Instância 8	500	4858	1	176

Tabela 4.2: Resultados - Instâncias

	Tempo PD (mm:ss)	Tempo PM (mm:ss)	Resultado	Recursos
Instância 1	$<$ 0:00.1	0:11.9	131	44
Instância 2	$<$ 0:00.1	0:11.5	131	44
Instância 3	$<$ 0:00.1	0:07.7	2	15
Instância 4	$<$ 0:00.1	0:08.1	2	15
Instância 5	$<$ 0:00.1	0:24.9	420	12
Instância 6	0:00.2	0:24.1	420	12
Instância 7	0:00.9	2:24.1	652	143
Instância 8	0:00.8	2:16.0	652	143

4.1 Resultados Numéricos

Neste experimento, foi especificado que se desejava descobrir todos os caminhos mínimos partindo do primeiro vértice. Assim, foi necessário executar o algoritmo de programação dinâmica apenas uma vez, pois este traz todos estes caminhos. Já com o modelo em programação matemática, foi necessário executar $n-1$ vezes para retornar todos os caminhos desejados. Os resultados podem ser observados na tabela 4.2. Na primeira coluna temos o tempo de execução do algoritmo de programação dinâmica. Na segunda coluna temos o tempo de execução do modelo em AMPL, utilizando o solver CPLEX. Na terceira o resultado ótimo da instância e por fim o consumo de recursos.

4.2 Discussão dos Resultados

Como já era esperado, as soluções encontradas nos experimentos utilizando programação dinâmica e programação matemática foram os mesmos. Com relação ao tempo computacional gasto para encontrar todos os caminhos mínimos sob uma restrição partindo do primeiro vértice, o experimento utilizando programação dinâmica foi mais rápido. Porém se fosse apenas necessário calcular um caminho mínimo de um vértice outro, o tempo de execução da programação matemática levaria vantagem, pois o tempo de execução seria quase n vezes menor, enquanto que o tempo da programação dinâmica continuaria o mesmo.

Um outro aspecto relevante nos resultados é que o algoritmo de programação dinâmica se restringe a apenas uma restrição, enquanto que não existe um número máximo de restrições quando se utiliza programação matemática.

Além disso, deve-se levar em conta a disponibilidade de uma ferramenta de programação matemática. Neste trabalho, foi utilizado o CPLEX. Já o algoritmo de programação dinâmica pode ser implementado na maioria das linguagens de programação.

Capítulo 5

Aproximação-ε

A técnica de aproximação ε para o problema de caminhos mínimos sob restrição foi apresentada por Hassin [5]. Nela foi aplicada a técnica de *rounding and scaling* para obter um algoritmo completamente polinomial por aproximação ε.

No Capítulo 3 foi apresentada a recursão dual (3.3) de programação dinâmica para o RCSP. Nesta recursão é apresentada uma forma de calcular um caminho mínimo em função dos recursos com um limite máximo de gastos, quando no problema original queremos calcular o gasto mínimo a partir de um limite de recursos. Para se realizar a recursão dual é necessário saber um valor máximo de c, chamado de c^*, que encontra a solução ótima do problema conforme já explicado.

Para se realizar a aproximação é necessário um procedimento de teste. Seja V um certo valor, e suponha que queremos testar se $V \geqslant c^*$. Um procedimento polinomial que responde a pergunta pode ser estendido a um algoritmo polinomial que determina o valor exato valor de c^*. O valor de c^* pode ser encontrado através de pesquisa binária em $0, ..., B$, onde B é o limite superior para c^*. O valor exato de c^* é encontrado a partir da aplicação de $O(\log B)$ procedimentos polinomiais. Como o problema é *NP*-Difícil ele terá que ser satisfeito com um teste fraco.

Seja ε fixado em $0 < \varepsilon < 1$. Suponha que queremos construir uma aproximação-ε. Este teste tem as seguintes propriedades: se $c^* \geqslant V$, então resulta em positivo, caso contrário negativo. Se o resultado for negativo, então sabemos que $c^* < V(1+\varepsilon)$.

Para realizar o teste os pesos de c_{ij} são arredondados para:

$$\left\lfloor \frac{c_{ij}}{V\varepsilon/(n-1)} \right\rfloor \frac{V\varepsilon}{(n-1)} \tag{5.1}$$

Este arredondamento resulta num decréscimo em cada aresta de no máximo $V\varepsilon/(n-1)$, e em cada caminho de no máximo $V\varepsilon$. Desta forma, é possível resolver este problema

utilizando a recursão dual (3.3) para uma rede com arestas de custos $\lfloor c_{ij}/(V\varepsilon/(n-1))\rfloor$. Os valores $r_j(c)$, $j = 2,...,n$, são primeiramente calculados para $c = 1$, então para $c = 2,3,...$ até que $r_n(c) \leqslant R$ para algum $c = \hat{c} < (n-1)/\varepsilon$, ou $c \geqslant (n-1)/\varepsilon$, onde R é a quantidade de recursos disponível.

Caso $r_n(c) \leqslant R$ seja verdadeiro então um caminho de no máximo

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

terá sido encontrado. O custo é no máximo $\frac{\hat{c}V\varepsilon}{(n-1)} + V\varepsilon < \frac{(n-1)}{\varepsilon}\frac{V\varepsilon}{(n-1)} + V\varepsilon = V + V\varepsilon = V(1+\varepsilon)$. Caso contrário, cada caminho tem $c' \geqslant (n-1)/\varepsilon$ ou $c \geqslant V$, assim $c^* \geqslant V$. Este teste pode ser resumido pelo algoritmo 5.1.

Algoritmo de Teste para Aproximação ε
$Testa(V,\varepsilon)$

$c \leftarrow 0$
1. Para todo $(i,j) \in E$ faça
    Se $c_{ij} > V$ então
        $E \leftarrow E \setminus \{(i,j)\}$
    senão
        $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\varepsilon \rfloor$
2. Enquanto (verdadeiro)
        Se $c \geqslant (n-1)/\varepsilon$ então
            retorna VERDADEIRO
        senão
            Use o algoritmo dual 3.4 para computar $r_j(c)$ para $j = 2,...,n$
            Se $r_j(c) \leqslant R$
                retorna FALSO
            senão
                $c \leftarrow c+1$

Algoritmo 5.1: Teste ε

Obter a parte inteira de um número A que se sabe que tem limite superior U pode ser feito em $O(\log U)$ passos, utilizando pesquisa binária. O passo 1 do algoritmo 5.1 requer $O(|E|)$ operações. Como serão arredondados apenas os custos c_{ij} que são menores que V, então todos os números que serão aplicados operações de chão serão menores que $(n-1)/\varepsilon$. Então o passo 1 requer tempo $|E|\log(n/\varepsilon)$. Como cada iteração do algoritmo dual 3.4 requer tempo $O(|E|)$, e como o passo 2 requer $O(n/\varepsilon)$ iterações, então o tempo computacional deste passo é $O(|E|n/\varepsilon)$.

Para realizar a aproximação precisamos primeiramente computar um limite mínimo e um limite máximo. Um limite superior, chamado de UB, pode ser calculado pelo somatório dos $n-1$ maiores custos das arestas, ou pelo custo do caminho que tem o menor consumo de recursos. Caso este caminho não seja um caminho factível ao problema, então o problema não tem solução. O limite inferior, chamado de LB, pode ser calculado pelo caminho mínimo sem restrição, ou simplesmente fazendo $LB = 1$, já que arestas de peso nulo não estão presentes.

Para o grafo exemplo da figura 3.1, um limite superior UB frouxo seria o número de vértices vezes o maior custo das arestas, ou seja, $10 * 4 = 40$. Um outro limite superior um pouco melhor, é obtido através dos $n-1$ maiores custos, $4+4+4+4+3+3+2+2+2+2=$ 26. Um limite mais elaborado seria calcular o caminho com o menor consumo de recurso possível, desta forma já seria possível obter uma solução factível, caso exista. Para obter este caminho pode ser utilizado o algoritmo de Dijkstra [1] utilizando como pesos das arestas o seu respectivo consumo de recursos. O consumo mínimo de recursos para este caso é 9 e o custo do caminho gerado é 14, desta forma o valor de UB seria 14.

Para o limite inferior LB pode ser atribuído o valor 1, pois será considerado que todas as arestas têm custo de no mínimo 1. Um limite LB mais elaborado pode ser calculado pelo caminho mínimo sem restrições utilizando o algoritmo de Dijkstra. O valor do caminho mínimo para o exemplo é de 10, que pode ser utilizado como limite inferior na aproximação.

Caso $UB \leqslant (1+\varepsilon)LB$ então UB é uma aproximação-ε para c^*. Supondo que $UB > (1+\varepsilon)LB$. Seja V um valor dado e $LB < V < UB(1+\varepsilon)^{-1}$. O teste em cima de V pode ser aplicado para melhorar os limites de c^*. Ou seja, podemos aumentar LB para V ou então diminuir UB para $V(1+\varepsilon)$. Realizando uma seqüência de testes a proporção de UB/LB poderá ser reduzida. Tão logo esta constante esteja abaixo de um valor pré-determinado, a aproximação-ε pode ser obtida aplicando a recursão dual para os valores arredondados de $\lfloor c_{ij}/(LB\varepsilon/(n-1))\rfloor$. O erro introduzido é de no máximo εLB, onde $\varepsilon LB < \varepsilon c^*$ e o tempo computacional para calcular o caminho é $O(|E|n/\varepsilon)$, assim como uma simples aplicação do teste de intervalo.

A redução da proporção entre UB/LB é mais bem atingida aplicando uma pesquisa binária no intervalo de (LB, UB) em escala logarítmica. A cada teste os limites são modificados. Para garantir uma redução rápida o teste será executado em um ponto x, onde $UB/x = x/LB$, que é $x = (UB \cdot LB)^{1/2}$. O número de testes necessários para se ter uma proporção menor que 2 é $O(\log\log(UB/LB))$.

Algoritmo de Aproximação

Iniciar valores de LB e UB com seus valores iniciais.
Enquanto $(2LB < UB)$
    $V \leftarrow (LB \cdot UB)^{1/2}$
    Se Testa(V, ε) = VERDADEIRO
        $LB \leftarrow V$
    senão
        $UB \leftarrow V(1+\varepsilon)$
Para todo $(i, j) \in E$ faça
    $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\varepsilon LB \rfloor$
Aplique a recursão dual para computar o caminho.

Algoritmo 5.2: Aproximação por ε

5.1 Exemplo

As instâncias utilizadas nesta seção são as mesmas da tabela 4.1. Neste exemplo, para execução foi utilizada a instância 1 com $\varepsilon = 0,4$.

Na figura 5.1 é mostrado em porcentagem do resultado ótimo os valores dos limites inferior e superior em função das iterações. Nesta figura pode ser observada a convergência dos limites a cada iteração, fazendo com que o intervalo entre os limites fique menor e o resultado mais preciso. Neste gráfico foram eliminadas as duas primeiras iterações para deixar o gráfico mais claro. Assim foram retiradas a iteração que tinha limite inferior 1 e a iteração com o limite superior de $(n-1) * \textit{maior custo}$. Esses valores iniciais para os limites foram atribuídos para validar o algoritmo de aproximação e para ficar mais ilustrativo, já que poderiam ser utilizados valores inicias mais elaborados, como foi citado anteriormente.

A tabela 5.1 mostra os dados desta mesma execução. Nesta tabela, são mostrados os valores numéricos do limite inferior (LB) e superior (UB) em cada iteração; além disso, são mostrados os valores de $(LB \cdot UB)^{1/2}$ e a resposta do teste-ε. Pela tabela, pode-se observar que as iterações são executadas até que o resultado seja satisfatório (neste caso iteração 9), que neste algoritmo está definido quando $(UB/LB) < 2$.

Ao final deste experimento o caminho obtido para a instância 1 foi o mesmo gerado pelos experimentos anteriores, ou seja, foi obtido o resultado ótimo. Como este algoritmo é uma aproximação, nada garante que este resultado seja ótimo; seria apenas garantido que o resultado estaria dentro dos limites obtidos e que o ótimo também estaria neste intervalo.

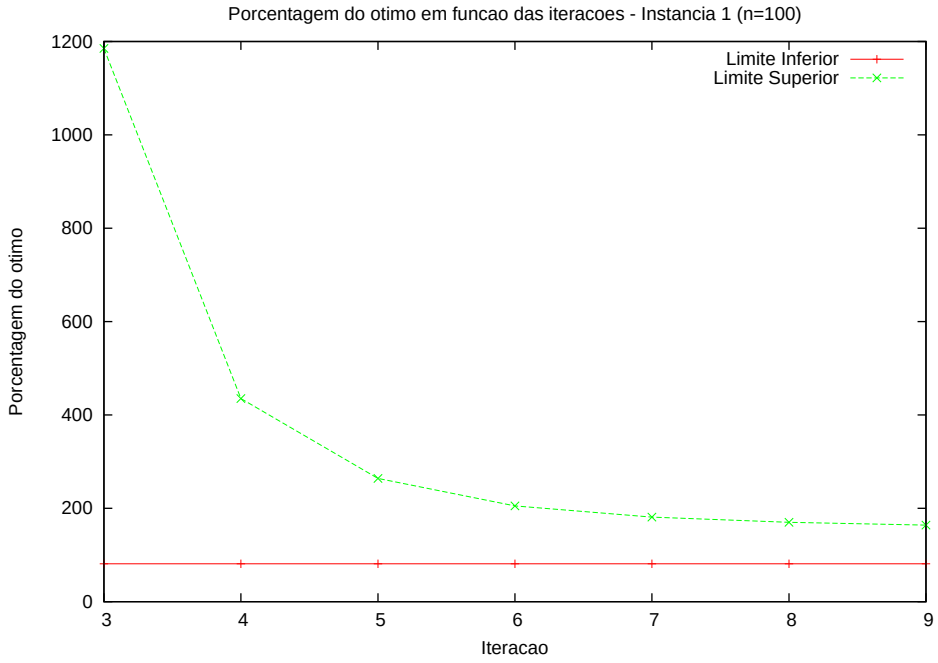

Figura 5.1: Gráfico dos limites inferior e superior em função das iterações com $\varepsilon = 0,4$

5.2 Resultados Numéricos

As respostas obtidas nestes experimentos foram as mesmas respostas obtidas com os algoritmos determinísticos de programação dinâmica, ou seja, os resultados obtidos foram ótimos.

Na figura 5.2, é mostrado o limite inferior em porcentagem da solução ótima que é obtido ao final da execução de cada experimento variando o valor de ε. Nestes experimentos, o valor de ε variou de 0,1 até 0,9, com acréscimo de 0,1 a cada execução. Nesta figura, pode

Tabela 5.1: Tabela de Execução - Instância 1, $\varepsilon = 0,4$

Iteração	Limite inferior (LB)	Limite superior (UB)	$(LB \cdot UB)^{1/2}$	Teste-ε
1	1,000000	11484,000000	107,163429	Sim
2	107,163429	11484,000000	1109,353394	Não
3	107,163429	1553,094727	407,964417	Não
4	107,163429	571,150208	247,399307	Não
5	107,163429	346,359039	192,657791	Não
6	107,163429	269,720917	170,012405	Não
7	107,163429	238,017365	159,708344	Não
8	107,163429	223,591675	154,792923	Não
9	107,163429	216,710098	152,392242	Não

ser observado que o limite inferior se manteve estável, mesmo havendo variação no valor de ε. O limite mais próximo do valor ótimo foi na execução da instância 1 quando foi obtido um limite inferior de 81,8% do resultado ótimo. O limite inferior mais longe foi na instância 3 que obteve 50% do ótimo.

Os resultados dos limites superiores dos experimentos estão ilustrados na figura 5.3. Esta figura reflete os mesmos experimentos ilustrados na figura anterior, mas nesta figura percebe-se que o limite superior sofreu influência da variação do valor de ε. O limite superior mais próximo ocorreu na instância 5 com $\varepsilon = 0,1$, onde o limite ficou 123,63% do valor ótimo. O mais longe ficou a 310,44% do valor ótimo na instância 9 com $\varepsilon = 0,9$.

Comparando os dois gráficos anteriores, observa-se que em alguns casos a razão entre o UB e o LB é maior que 2, sendo que a condição de parada era quando esta razão fosse menor. Neste caso, os limites pararam de convergir pois o valor de $(UB/LB)^{1/2} * (1+\varepsilon)$ era maior que o limite superior da iteração anterior. Nestes casos, há duas opções: ficar com o resultado já obtido ou ajustar o valor de ε para que voltasse a convergir. Como foram realizados experimentos com diversos valores de ε para efeito de comparação, simplesmente ficamos com o resultado quando a convergência foi interrompida.

Analisando os gráficos 5.2 e 5.3, pode-se observar uma tendência que quanto menor o valor de ε maior a precisão do intervalo entre os limites.

A figura 5.4 ilustra o tempo de execução das instâncias 1, 3 e 5 da aproximação para diferentes valores de ε. Neste gráfico, pode ser observado que o tempo de execução teve tendência de queda para valores de ε mais próximos de 1. Já a figura 5.5 ilustra o tempo de execução da instância 9 para os mesmos valores de ε. Nesta última instância, para valores de ε maiores que 0,5 o algoritmo não convergiu para que satisfizesse a condição de parada. Por isso, foi necessário esperar que o algoritmo convergisse acarretando um tempo de execução maior do que o esperado.

Algumas vezes com poucas iterações o resultado ótimo já pode ser obtido utilizando o caminho gerado naquela iteração, mas como o algoritmo de aproximação não pode afirmar que aquele resultado é ótimo, o algoritmo continuou até que os valores dos limites inferior e superior se tornassem satisfatórios ou senão os limites parassem de convergir.

Foi realizado um outro experimento com o intuito de mostrar o tempo de execução com a variação dos recursos disponíveis em uma instância. Para este teste foi escolhida a instância 1 e a quantidade recursos foi alterada para serem medidos os tempos de execução do algoritmo de programação dinâmica primal e da aproximação-ε. Originalmente, a instância 1 tinha 73 recursos disponíveis e este valor foi alterado nos testes para 100, 1000, 10000 e 100000. O valor de ε nas aproximações foi ajustado da mesma forma que no experimento anterior, ou seja, variou de 0,1 a 0,9. O caminho gerado neste experimento foi o mesmo em todas as execuções, já que atingia o ótimo com 81 recursos disponíveis e por isso não foi citado nesta

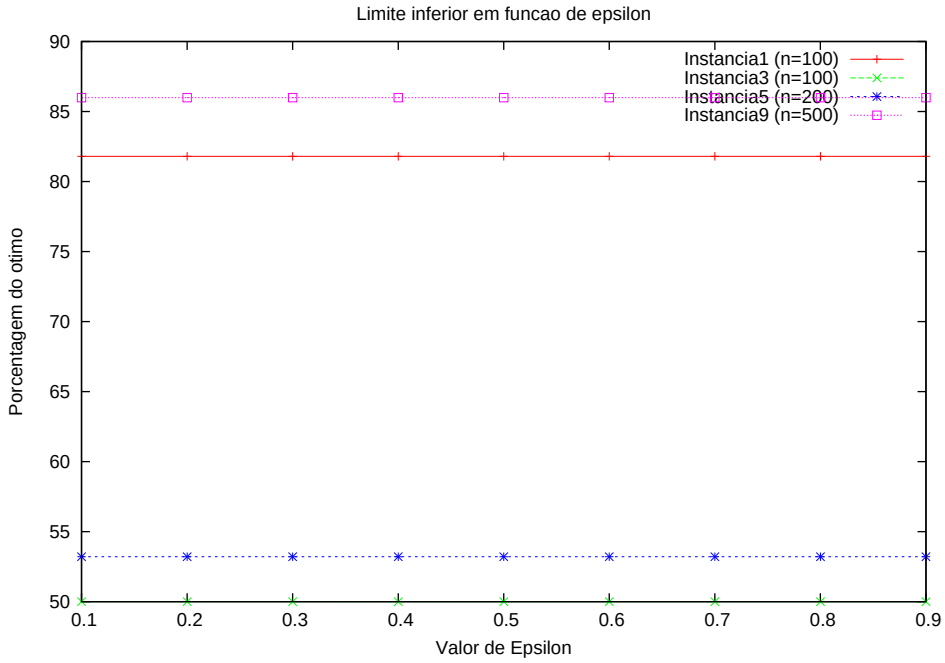

Figura 5.2: Gráfico do limite inferior em função do ε

seção.

A tabela 5.2 mostra na primeira linha os tempos de execução para a programação dinâmica e em seguida vêm as aproximações. Na tabela pode ser observado que o tempo de execução da programação dinâmica primal é dependente da quantidade de recursos disponíveis, o tempo cresceu linearmente com o aumento de recursos. Já a aproximação foi menos dependente deste valor. Um outro aspecto que se pode observar é que inicialmente o algoritmo primal estava um pouco mais rápido que as aproximações e logo que os recursos disponíveis aumentaram o tempo de execução se inverteu, fazendo com que as aproximações fossem consideravelmente mais rápidas.

5.3 Discussão dos Resultados

Os resultados obtidos para as instâncias testadas pela aproximação-ε foram ótimos. Em geral, o tempo de execução das instâncias foi próximo à programação dinâmica primal, com valores de ε bem ajustados. Mas quando a quantidade de recursos disponíveis é elevada, a aproximação tem um desempenho melhor com relação ao tempo de execução.

A escolha do valor de ε se mostrou bastante importante, tanto para o tempo de execução como para que se garanta que a solução esteja dentro de uma margem de erro. Além disso, o ajuste dos valores de ε é crucial para que a convergência ocorra da forma esperada e que o teste de parada tenha sucesso. Pode-se mostrar que quanto mais próximo de zero melhor será

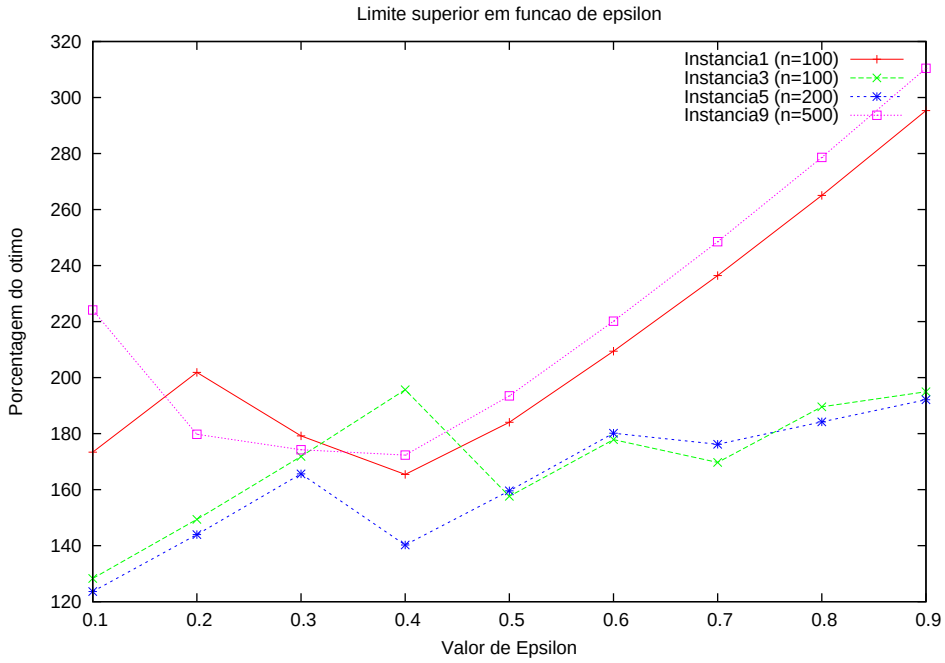

Figura 5.3: Gráfico do limite superior em função do ε

a qualidade da solução e mais tempo será necessário para executar e quanto mais próximo de 1 menos tempo será necessário para a execução e o intervalo entre os limites superiores e inferiores fica maior.

A aproximação-ε se mostrou uma boa técnica para a solução de problemas onde o valor dos recursos disponíveis é grande. Neste caso, a programação dinâmica primal tem um desempenho proporcionalmente ruim. Além disso, os resultados foram bastante satisfatórios, pois nestes experimentos o caminho ótimo acabou sendo gerado. Lembrando que não havia garantias que os valores ótimos seriam encontrados. Em aplicações onde o resultado ótimo é necessário ainda é melhor utilizar a programação dinâmica original, pois desta forma, sempre o melhor resultado será obtido se o problema for factível.

5.4 Sumário

Neste capítulo é apresentado a aproximação-ε, que utiliza programação dinâmica para resolver o problema de caminhos mínimos sob restrição. Foram apresentados algoritmos que resolvem o problema por aproximação em tempo polinomial, apresentados exemplos e resultados numéricos deste método.

A aproximação-ε se mostrou uma boa técnica para solução de problemas onde a quantidade de recursos disponíveis é grande. Mas quando isso não ocorre a programação dinâmica simples pode ser mais útil.

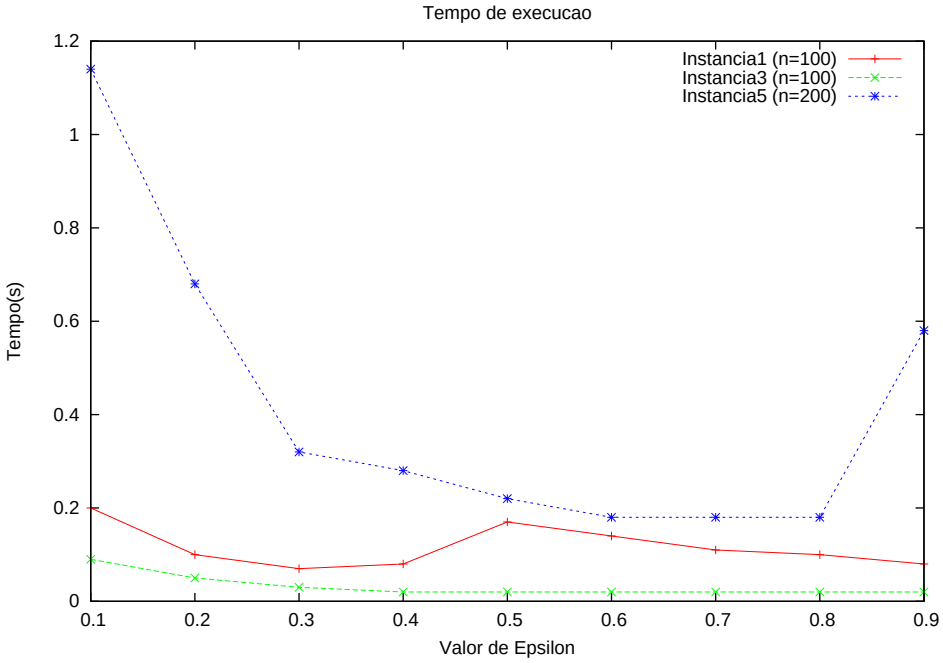

Figura 5.4: Gráfico do tempo de execução em função do ε

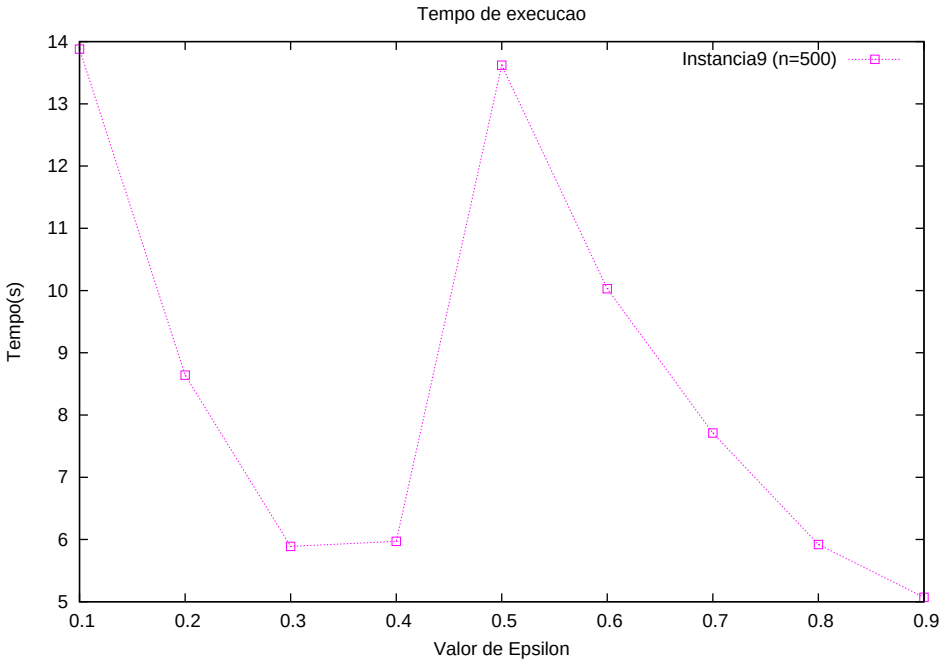

Figura 5.5: Gráfico do tempo de execução em função do ε

Tabela 5.2: Comparação - Tempo de execução (s) - Instância 1

Algoritmo	R=100	R=1000	R=10000	R=100000
PD Primal	$<$ 0,1	0,2	2,3	23,8
Aprox. $\varepsilon = 0,1$	0,1	0,1	0,2	0,8
Aprox. $\varepsilon = 0,2$	$<$ 0,1	0,1	0,1	0,7
Aprox. $\varepsilon = 0,3$	$<$ 0,1	$<$ 0,1	0,1	1,0
Aprox. $\varepsilon = 0,4$	$<$ 0,1	$<$ 0,1	0,1	0,9
Aprox. $\varepsilon = 0,5$	$<$ 0,1	$<$ 0,1	0,1	1,3
Aprox. $\varepsilon = 0,6$	$<$ 0,1	$<$ 0,1	0,1	0,8
Aprox. $\varepsilon = 0,7$	0,1	0,1	0,1	1,6
Aprox. $\varepsilon = 0,8$	0,1	0,1	0,1	0,7
Aprox. $\varepsilon = 0,9$	$<$ 0,1	0,1	0,1	1,2

Capítulo 6

Tratamento de Objetivos Não Aditivos

Uma variação do problema de caminhos mínimos sob restrições é quando a função objetivo é não aditiva, ou seja, tratando-se de minimização, deseja-se encontrar o caminho que minimize o maior valor das arestas que fazem parte deste caminho. Um exemplo desta variação é quando se paga apenas a maior taxa ou pedágio de uma aresta. Um outro exemplo de objetivo não aditivo é quando se deseja maximizar a banda de uma conexão.

6.1 Minimização

Originalmente uma função objetivo aditiva de minimização pode ser formulada como:

$$\text{Min} \quad f = \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{6.1}$$

Quando a função objetivo passa a ser não aditiva, esta pode ser formulada como:

$$\text{Min} \quad f = \max\{c_{ij} x_{ij} : (i,j) \in E\} \tag{6.2}$$

lembrando que c_{ij} é o custo do arco (i,j) e x_{ij} é uma variável binária que assume o valor 1 quando o arco (i,j) pertence ao caminho, caso contrário x_{ij} assume o valor 0. Esta equação ilustra uma minimização de uma função objetivo não aditiva, ou seja, deseja-se obter o mínimo do maior valor dentro do caminho escolhido.

Para tratar um problema de caminhos mínimos sob restrições com função objetivo não aditiva em programação matemática é necessário fazer algumas alterações em sua formulação. Uma forma é colocar uma restrição na qual a função objetivo é maior ou igual a qualquer

um dos valores das arestas que fazem parte do caminho escolhido:

$$P: \text{ Min } \quad f = u \tag{6.3a}$$

$$\text{S. a :}$$

$$u \geqslant c_{ij}x_{ij}, \quad \forall (i,j) \in E \tag{6.3b}$$

$$\sum_{\{(i,j)\in E\}} r_{ij}^k x_{ij} \leqslant \lambda^k, \quad \text{para } k = 1,...,K \tag{6.3c}$$

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = 0, \quad \forall i \in V - \{s,t\} \tag{6.3d}$$

$$\sum_{\{j:(s,j)\in E\}} x_{sj} = 1 \tag{6.3e}$$

$$\sum_{\{i:(i,t)\in E\}} x_{it} = 1 \tag{6.3f}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{6.3g}$$

onde:

- $c_{ij} \in \mathbb{Z}_+ \backslash \{0\}$;
- $r_{ij} \in \mathbb{Z}_+ \backslash \{0\}$.

Esta variação do problema de caminhos mínimos sob restrições também pode ser tratada com programação dinâmica. A recursão de programação dinâmica é dada por:

$$c_j(r) = \min\{c_j(r-1), \min_{(i,j)\in E, r_{ij}\leqslant r} \max\{c_i(r-r_{ij}), c_{ij}\}\} \tag{6.4}$$

onde $c_1(r) = 0$ para $r = 0,...,\lambda$ e $c_j(0) = \infty$ para $j = 2,...,n$.

O algoritmo 6.1 é resultante da recursão com objetivo não aditivo. Este algoritmo é baseado no algoritmo 3.1, formulado para função aditiva.

6.1.1 Exemplo

Utilizando o exemplo da figura 3.1 podemos aplicar a recursão de programação dinâmica para função objetivo não aditiva. A tabela 6.1 mostra para cada quantidade de recurso disponível o valor da função objetivo não aditiva para se chegar naquele vértice.

A tabela de vértices anteriores (Tabela 6.2) mostra o vértice anterior em cada vértice para cada quantidade de recurso disponível. O algoritmo 3.2 pode ser utilizado para encontrar

Algoritmo de Programação Dinâmica - Objetivo não aditivo

Para $r = 0$ até λ faça
    $c_1(r) \leftarrow 0$
    $\pi_1(r) \leftarrow 0$
Para $j = 2$ até n faça
    $c_j(0) \leftarrow \infty$
    $\pi_j(0) \leftarrow 0$
Para $r = 1$ até λ faça
    Para $j = 2$ até n faça
        $c_j(r) \leftarrow c_j(r-1)$
        $\pi_j(r) \leftarrow \pi_j(r-1)$
        Para todo $i \in \{i : (i, j) \in E\}$ faça
            Se $(r_{ij} \leqslant r)$ e $(c_j(r) > \max\{c_i(r - r_{ij}), c_{ij}\})$ faça
                $c_j(r) \leftarrow \max\{c_i(r - r_{ij}), c_{ij}\}$
                $\pi_j(r) \leftarrow i$

Algoritmo 6.1: Programação Dinâmica - Objetivo não aditivo

o caminho mínimo deste caso e é o mesmo da programação dinâmica com função objetivo aditiva.

O caminho gerado para se sair do nó 1 e chegar ao nó 10 com função não aditiva e com 11 recursos disponíveis é o mesmo gerado quanto temos função aditiva, ou seja o caminho continuou sendo: $1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow 10$. Mas agora o resultado da função é 3, já que os arcos de maior valor/custo neste caminho são $(6,7)$ e $(9,10)$, onde $c_{6,7} = 3$ e $c_{9,10} = 3$.

Uma mudança que pode ser observada seria na geração do caminho do nó 1 ao nó 9 com 12 recursos disponíveis. Anteriormente com o objetivo aditivo o caminho seria $1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9$, agora com o objetivo não aditivo o caminho fica $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 9$ e tem resposta 2, que são os pesos dos arcos $(5,7)$ e $(7,9)$.

6.2 Maximização

O problema de maximização de caminhos mínimos sob restrição com função objetivo não aditiva pode ser exemplificado quando se deseja obter a maior banda possível em uma rede de computadores.

A função objetivo não aditiva de maximização do problema de caminhos mínimos é dada por:

Tabela 6.1: Programação Dinâmica - $c_j(r)$ - Recursos $\times$ Função objetivo não aditiva

$c_j(r)\backslash r$	0	1	2	3	4	5	6	7	8	9	10	11	12
$c_1(r)$	0	0	0	0	0	0	0	0	0	0	0	0	0
$c_2(r)$	∞	∞	∞	2	2	2	2	2	2	2	2	2	2
$c_3(r)$	∞	∞	1	1	1	1	1	1	1	1	1	1	1
$c_4(r)$	∞	∞	∞	∞	4	1	1	1	1	1	1	1	1
$c_5(r)$	∞	∞	∞	∞	4	4	4	1	1	1	1	1	1
$c_6(r)$	∞	∞	∞	∞	∞	∞	1	1	1	1	1	1	1
$c_7(r)$	∞	∞	∞	∞	∞	∞	4	3	3	3	3	3	3
$c_8(r)$	∞	∞	∞	∞	∞	∞	∞	∞	4	4	4	4	4
$c_9(r)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	4	3	3	2
$c_{10}(r)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	4	4	3	3

Tabela 6.2: Vértices Anteriores ($\pi_j(r)$) - Função objetivo não aditiva

$\pi_j(r)\backslash r$	0	1	2	3	4	5	6	7	8	9	10	11	12
$\pi_1(r)$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\pi_2(r)$	-	-	-	1	1	1	1	1	1	1	1	1	1
$\pi_3(r)$	-	-	1	1	1	1	1	1	1	1	1	1	1
$\pi_4(r)$	-	-	-	-	2	3	3	3	3	3	3	3	3
$\pi_5(r)$	-	-	-	-	2	2	2	4	4	4	4	4	4
$\pi_6(r)$	-	-	-	-	-	-	3	3	3	3	3	3	3
$\pi_7(r)$	-	-	-	-	-	-	5	6	6	5	5	5	5
$\pi_8(r)$	-	-	-	-	-	-	-	-	7	7	7	7	7
$\pi_9(r)$	-	-	-	-	-	-	-	-	-	7	7	7	7
$\pi_{10}(r)$	-	-	-	-	-	-	-	-	-	8	8	9	9

$$\text{Max} \quad f = \min\{c_{ij}x_{ij} : (i,j) \in E\} \tag{6.5}$$

Lembrando que c_{ij} é o consumo na aresta (i,j) e x_{ij} é a variável binária de decisão se a aresta (i,j) faz parte da solução.

Partindo da formulação da função objetivo, obtém-se o modelo em programação matemática do problema de caminhos mínimos sob restrições com função objetivo não aditivo:

$$P: \text{ Max} \quad f = l \tag{6.6a}$$

S. a :

$$l \leqslant c_{ij}x_{ij}, \quad \forall (i,j) \in E \tag{6.6b}$$

$$\sum_{\{(i,j)\in E\}} r_{ij}^k x_{ij} \leqslant \lambda^k, \quad \text{para } k = 1,...,K \tag{6.6c}$$

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = 0, \quad \forall i \in V - \{s,t\} \tag{6.6d}$$

$$\sum_{\{j:(s,j)\in E\}} x_{sj} = 1 \tag{6.6e}$$

$$\sum_{\{i:(i,t)\in E\}} x_{it} = 1 \tag{6.6f}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{6.6g}$$

onde: $c_{ij} \in \mathbb{Z}_+ \backslash \{0\}, \quad r_{ij} \in \mathbb{Z}_+ \backslash \{0\}$ para $\forall (i,j) \in E$

Em programação dinâmica o problema fica:

$$c_j(r) = \max\{c_j(r-1), \max_{(i,j)\in E, r_{ij}\leqslant r} \min\{c_i(r-r_{ij}), c_{ij}\}\} \tag{6.7}$$

onde $c_1(r) = 0$ para $r = 0,...,\lambda$ enquanto $c_j(0) = -\infty$ para $j = 2,...,n$

Para exemplificar uma aplicação desta variação do problema de caminhos mínimos sob restrições, considere o problema onde temos em cada arco (i,j):

- a capacidade de transmissão c_{ij} (ou vazão); e
- o custo de transmissão r_{ij} através do arco.

O problema é encontrar um caminho de s para t com maior capacidade possível de transmissão (ou vazão) que não ultrapasse o orçamento. A vazão do caminho é dada pelo arco de menor vazão.

Capítulo 7

Tratamento de Restrições Não Aditivas

Uma outra variação neste tipo de problema é quando temos restrições não aditivas. Suponha que o problema P tenha este tipo de restrição.

No caso I, as restrições aditivas não devem ultrapassar o valor de λ e tem a seguinte formulação:

$$\sum_{(i,j)\in E} r_{ij}^k x_{ij} \leqslant \lambda^k \tag{7.1}$$

Quando passamos as restrições para que sejam não aditivas estas ficam com o seguinte formato:

$$\text{Max}\{r_{ij}^k x_{ij} : (i,j) \in E\} \leqslant \lambda^k \tag{7.2}$$

No caso II, quando as restrições aditivas devem ultrapassar o valor de λ, estas ficam com o seguinte formato:

$$\sum_{(i,j)\in E} r_{ij}^k x_{ij} \geqslant \lambda^k \tag{7.3}$$

Passando para restrições não aditivas, temos:

$$\text{Min}\{r_{ij}^k x_{ij} : (i,j) \in E\} \geqslant \lambda^k \tag{7.4}$$

Para exemplificar uma aplicação desta variação de problema seja:

- c_{ij} é o custo de transmissão através do arco (i, j); e
- r_{ij}^k é a vazão do arco (i, j).

Assim o problema seria o de encontrar o caminho de menor custo de s para t tal que a vazão mínima seja λ^k.

Para tratar restrições não aditivas deve-se fazer:

- No caso (I), basta remover do grafo os arcos (i, j) tais que $r_{ij}^k > \lambda^k$;
- No caso (II), basta remover do grafo os arcos (i, j) tais que $r_{ij}^k < \lambda^k$.

Após a remoção dos arcos, a restrição não aditiva k será satisfeita por qualquer caminho no grafo. Assim, podemos remover a restrição do problema, pois esta será satisfeita automaticamente.

O problema de caminhos mínimos com restrições apenas não aditivas se torna um problema de caminhos mínimos convencional e pode ser resolvido com algoritmos convencionais. Dentre eles podemos citar os algoritmos de Dijkstra e Bellman-Ford [13].

A complexidade do algoritmo de Dijkstra é $O(V^2 + VE) = O(V^3)$ com fila de prioridades com listas ligadas, $O((V + E)\lg V)$ com *heap* binário e $O(V \lg V + E)$ com *heap* Fibonacci. A complexidade do algoritmo de Bellman-Ford é $O(VE)$, sendo $O(V^2)$ em grafo esparso e $O(V^3)$ em um grafo denso. V é o número de nós do grafo e E é o número de arestas.

O algoritmo de Dijkstra é aplicável apenas quando os custos dos arcos são não negativos. Bellman-Ford aceita custos negativos, podendo detectar a existência de ciclos de custo estritamente negativo.

Capítulo 8

Tratando Restrições Probabilísticas

Considere um recurso k e a restrição onde as parcelas são variáveis aleatórias:

$$\sum_{(i,j)\in E} \mathbf{r}_{ij}^k x_{ij} \leqslant \lambda^k \tag{8.1}$$

ou seja, $\mathbf{r}_{ij}^k$ é uma variável aleatória para todo $(i,j) \in E$. Suponha que $\mathbf{r}_{ij}^k$ é uma variável aleatória com distribuição normal tendo esta média μ_{ij}^k e desvio padrão σ_{ij}^k. Ou seja, $E[\mathbf{r}_{ij}^k] = \mu_{ij}^k$ e $E[(\mathbf{r}_{ij}^k - E[\mathbf{r}_{ij}^k])^2] = (\sigma_{ij}^k)^2$, onde $(\sigma_{ij}^k)^2$ é a variância.

Nesta seção será feita uma breve revisão de conceitos básicos sobre variáveis e distribuições. Em seguida, será ilustrada a conversão de uma restrição probabilística simples para uma restrição determinística equivalente. E por fim, serão aplicadas restrições probabilísticas a caminhos mínimos.

8.1 Conceitos Básicos Sobre Variáveis Aleatórias

Seja $\mathbf{r}^k = \sum_{(i,j)\in E} \mathbf{r}^k_{ij} x_{ij}$ uma variável aleatória dada pela soma das variáveis aleatórias $\mathbf{r}^k_{ij} x_{ij}$, $(i,j) \in E$. Sabemos da Teoria da Probabilidade [14] que:

$$E[\mathbf{r}^k] = E\left[\sum_{(i,j)\in E} \mathbf{r}^k_{ij} x_{ij}\right] \tag{8.2a}$$

$$= \sum_{(i,j)\in E} E[\mathbf{r}^k_{ij} x_{ij}] \tag{8.2b}$$

$$= \sum_{(i,j)\in E} E[\mathbf{r}^k_{ij}] x_{ij} \tag{8.2c}$$

$$= \sum_{(i,j)\in E} \mu^k_{ij} x_{ij} \tag{8.2d}$$

$\mathrm{Var}(\mathbf{r}^k)$ é a variância de $\mathbf{r}^k$ sendo definida por:

$$\mathrm{Var}(\mathbf{r}^k) = E\left[\left(\mathbf{r}^k - E[\mathbf{r}^k]\right)^2\right] \tag{8.3a}$$

$$= \sum_{(i,j)\in E} \mathrm{Var}\left(\mathbf{r}^k_{ij} x_{ij}\right) + \sum_{(i,j)\in E} \sum_{(l,t)\in E, (i,j)\neq(l,t)} \mathrm{Cov}\left(\mathbf{r}^k_{ij} x_{ij}, \mathbf{r}^k_{lt} x_{lt}\right) \tag{8.3b}$$

$\mathrm{Cov}(\mathbf{x}, y)$ é a covariância entre as variáveis aleatórias $\mathbf{x}$ e $\mathbf{y}$, sendo definida por:

$$\mathrm{Cov}(\mathbf{x},\mathbf{y}) = E\left[(\mathbf{x} - E[\mathbf{x}])(\mathbf{y} - E[\mathbf{y}])\right] \tag{8.4a}$$

$$= E[\mathbf{xy}] - E[\mathbf{x}]E[\mathbf{y}] \tag{8.4b}$$

Se $\mathbf{x}$ e $\mathbf{y}$ são independentes, então $\mathrm{Cov}(\mathbf{x},\mathbf{y}) = 0$, pois $E[\mathbf{xy}] = E[\mathbf{x}]E[\mathbf{y}]$. Podemos assumir que $\mathbf{r}^k_{ij}$ e $\mathbf{r}^k_{lt}$ são independentes, o que nos leva a deduzir que:

$$\mathrm{Var}(\mathbf{r}^k) = \sum_{(i,j)\in E} \mathrm{Var}(\mathbf{r}^k_{ij} x_{ij}) \tag{8.5}$$

Podemos ter o seguinte tipo de restrição probabilística:

$$P\left(\sum_{(i,j)\in E} \mathbf{r}_{ij}^{k} x_{ij} \leqslant \lambda^{k}\right) \geqslant \alpha \tag{8.6}$$

em palavras, queremos satisfazer a restrição $\sum_{(i,j)\in E} \mathbf{r}_{ij}^{k} x_{ij} \leqslant \lambda^{k}$ com probabilidade igual ou superior a α:

$$P\left(\sum_{(i,j)\in E} \mathbf{r}_{ij}^{k} x_{ij} \leqslant \lambda^{k}\right) \geqslant \alpha \Leftrightarrow P\left(\mathbf{r}^{k} \leqslant \lambda^{k}\right) \geqslant \alpha \tag{8.7}$$

onde:

$$\mathbf{r}^{k} = \sum_{(i,j)\in E} \mathbf{r}_{ij}^{k} x_{ij} \tag{8.8a}$$

$$E[\mathbf{r}^{k}] = \sum_{(i,j)\in E} E[\mathbf{r}_{ij}^{k}] x_{ij} \tag{8.8b}$$

$$\mathrm{Var}(\mathbf{r}^{k}) = \sum_{(i,j)\in E} \mathrm{Var}(\mathbf{r}_{ij}^{k} x_{ij}) \tag{8.8c}$$

8.2 Restrição Probabilística Simples

Vamos agora considerar uma restrição probabilística simples, onde $\mathbf{x}$ é uma variável aleatória com distribuição normal, com média $\mu = 0$ e variância $\sigma^2 = 1$, conforme ilustra a figura 8.1.

$$P(\mathbf{x} \leqslant \xi) \geqslant \alpha \Leftrightarrow P(\xi \geqslant \mathbf{x}) \geqslant \alpha \tag{8.9}$$

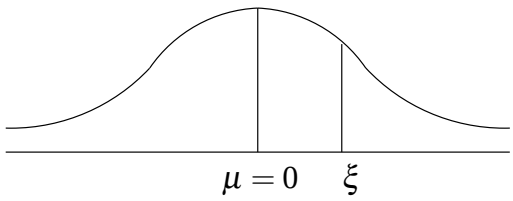

Figura 8.1: Função densidade de probabilidade $f_{\mathbf{x}}(y)$

Seja $F_{\mathbf{x}}(y) = P(\mathbf{x} \leqslant y)$ a distribuição de probabilidade cumulativa de $\mathbf{x}$, $F_{\mathbf{x}}(y) = \int_{x=-\infty}^{x=y} f_{\mathbf{x}}(x)dx$, onde $f_{\mathbf{x}}(x)$ é a função densidade de probabilidade. Então $F_{\mathbf{x}}^{-1}(\alpha) = \eta$ é

o percentil α, onde $\eta = \min\{y : F_{\mathbf{x}}(y) \geqslant \alpha\}$, conforme ilustra a figura 8.2. Então podemos substituir a restrição probabilística por uma determinística equivalente:

$$P(\xi \geqslant \mathbf{x}) \geqslant \alpha \Leftrightarrow \xi \geqslant F_{\mathbf{x}}^{-1}(\alpha) \Leftrightarrow F_{\mathbf{x}}^{-1}(\alpha) \leqslant \xi \tag{8.10}$$

Observe que através do equivalente determinístico eliminou-se a variável aleatória **x**. Assim temos uma restrição determinística em função de uma variável ξ e uma constante.

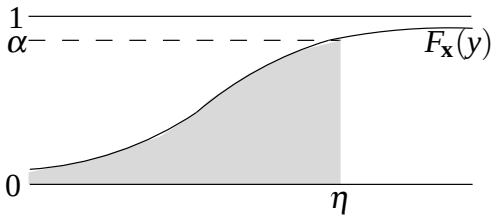

Figura 8.2: Distribuição de probabilidade cumulativa $F_{\mathbf{x}}(y)$

Seja $\hat{\mathbf{x}}$ uma variável com distribuição normal, tendo média μ não necessariamente igual a 0 e variância σ^2 não necessariamente igual a 1. Seja então uma variável aleatória definida em função de $\hat{\mathbf{x}}$: $\mathbf{x} = (\hat{\mathbf{x}} - \mu)/\sigma$. Note que

$$E[\mathbf{x}] = E\left[\frac{\hat{\mathbf{x}} - \mu}{\sigma}\right] = \frac{E[\hat{\mathbf{x}}]}{\sigma} - \frac{\mu}{\sigma} = \frac{\mu}{\sigma} - \frac{\mu}{\sigma} = 0 \tag{8.11}$$

portanto **x** tem média nula. Observe ainda que

$$\text{Var}[\mathbf{x}] = E\left[(\mathbf{x} - E[\mathbf{x}])^2\right] = E\left[\mathbf{x}^2\right] = E\left[(\hat{\mathbf{x}} - \mu)^2/\sigma^2\right] = E\left[(\hat{\mathbf{x}} - \mu)^2\right]/\sigma^2 = \sigma^2/\sigma^2 = 1 \tag{8.12}$$

portanto **x** tem variância igual a 1.

Considere a restrição probabilística:

$$P(\mathbf{x} \leqslant \xi) \geqslant \alpha \Leftrightarrow P(\xi \geqslant \mathbf{x}) \geqslant \alpha \tag{8.13a}$$

$$\Leftrightarrow P(\xi \geqslant \hat{\mathbf{x}}\sigma + \mu) \geqslant \alpha \text{ pois } \mathbf{x} = \frac{\hat{\mathbf{x}} - \mu}{\sigma} \Rightarrow \hat{\mathbf{x}} = \sigma\mathbf{x} + \mu \tag{8.13b}$$

$$\Leftrightarrow P\left(\frac{\xi - \mu}{\sigma} \geqslant \mathbf{x}\right) \geqslant \alpha \tag{8.13c}$$

Mas (8.13c) tem uma variável aleatória **x** com distribuição normal, média $\mu = 0$ e variância $\sigma = 1$. Podemos assim obter o equivalente determinístico de (8.13c), da mesma forma

que foi obtido em (8.10):

$$\frac{\xi - \mu}{\sigma} \geqslant F_{\mathbf{x}}^{-1}(\alpha) \Leftrightarrow \xi \geqslant \sigma F_{\mathbf{x}}^{-1}(\alpha) + \mu \tag{8.14}$$

Assim eliminamos a variável aleatória **x** e colocamos a restrição em uma forma determinística equivalente.

8.3 Restrições Probabilísticas em Caminhos Mínimos

Vamos tratar um caso mais geral de restrição probabilística:

$$P\left(\sum_{(i,j)\in E} \mathbf{r}_{ij}^k x_{ij} \leqslant \lambda^k\right) \geqslant \alpha \Leftrightarrow P\left(\lambda^k \geqslant \sum_{(ij)\in E} \mathbf{r}_{ij}^k x_{ij}\right) \geqslant \alpha \tag{8.15a}$$

$$\Leftrightarrow P\left(\lambda^k \geqslant \mathbf{r}^k\right) \geqslant \alpha \tag{8.15b}$$

onde $\mathbf{r}^k = \sum_{(i,j)\in E} \mathbf{r}_{ij}^k x_{ij}$ é uma variável aleatória com distribuição normal, tendo média $E[\mathbf{r}^k] = E[\sum_{(i,j)\in E} \mathbf{r}_{ij}^k x_{ij}] = \sum_{(i,j)\in E} E[\mathbf{r}_{ij}^k] x_{ij}$.

Estamos assumindo que cada $\mathbf{r}_{ij}^k$ tem uma distribuição normal. Observe ainda que

$$\text{Var}\left(\mathbf{r}^k\right) = \sum_{(i,j)\in E} \text{Var}\left(\mathbf{r}_{ij}^k x_{ij}\right) + \sum_{(i,j)\in E} \sum_{(l,t)\in E, (i,j)\neq(l,t)} \text{Cov}\left(\mathbf{r}_{ij}^k x_{ij}, \mathbf{r}_{lt}^k x_{lt}\right) \tag{8.16a}$$

$$= \sum_{(i,j)\in E} \text{Var}\left(\mathbf{r}_{ij}^k x_{ij}\right) \tag{8.16b}$$

se assumirmos que $\mathbf{r}_{ij}^k$ e $\mathbf{r}_{lt}^k$ são independentes.

Então, temos a restrição determinística equivalente:

$$\lambda^k \geqslant \sqrt{\mathrm{Var}(\mathbf{r}^k)}F_{\mathbf{x}}^{-1}(\alpha) + E[\mathbf{r}^k] \tag{8.17a}$$

$$\lambda^k \geqslant F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}\mathrm{Var}\left(\mathbf{r}_{ij}^k x_{ij}\right)} + \sum_{(i,j)\in E} E[\mathbf{r}_{ij}^k]x_{ij} \tag{8.17b}$$

$$= F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E} E\left[\left(\mathbf{r}_{ij}^k x_{ij} - E[\mathbf{r}_{ij}^k]x_{ij}\right)^2\right]} + \sum_{(i,j)\in E} E[\mathbf{r}_{ij}^k]x_{ij} \tag{8.17c}$$

$$= F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E} E\left[\left(\mathbf{r}_{ij}^k - E[\mathbf{r}_{ij}^k]\right)^2 x_{ij}^2\right]} + \sum_{(i,j)\in E} E[\mathbf{r}_{ij}^k]x_{ij} \tag{8.17d}$$

$$= F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}\mathrm{Var}\left(\mathbf{r}_{ij}^k\right)x_{ij}^2} + \sum_{(i,j)\in E} E[\mathbf{r}_{ij}^k]x_{ij} \tag{8.17e}$$

$F_{\mathbf{x}}$ é a distribuição cumulativa da variável aleatória $\mathbf{x}$ com distribuição normal, média nula e variância unitária. Note que (8.17e) é uma restrição determinística equivalente à restrição probabilística (8.15a). A dificuldade em se trabalhar com esta restrição equivalente determinística está no fato de ser não-linear nas variáveis de decisão x_{ij}. Um alternativa para esta dificuldade está em se utilizar limites superiores para o termo $F_{\mathbf{x}}^{-1}(\alpha)\sqrt{\sum_{(i,j)\in E}\mathrm{Var}(\mathbf{r}_{ij}^k)x_{ij}^2}$ da restrição, que passaria a ser linear e poderia ser facilmente inserida nos modelos já desenvolvidos, como programação matemática, programação dinâmica e aproximação-ε.

8.4 Sumário

Neste capítulo são revisados conceitos básicos sobre variáveis e distribuições estatísticas. São aplicados estes conceitos no problema de caminhos mínimos, gerando uma restrição probabilística. E por fim, é substituída a restrição probabilística por uma restrição determinística equivalente. Porém quando é realizada esta substituição, o problema passa a ser um problema não-linear. Uma forma de tratar este problema seria encontrar limites para o termo não-linear.

Parte III

Aplicações

Capítulo 9

Agentes Móveis

A tecnologia de agentes móveis é uma importante área de pesquisa no escopo de código móvel (*code mobility*) [15]. Um agente móvel é um elemento formado por um componente de software e tem a capacidade de migrar de forma autônoma pela rede. Além disso, é responsável pela execução de alguma tarefa, e ser capaz interromper esta tarefa em um lugar, mover-se para outro lugar e retomar aquela tarefa. Eles podem se adaptar dinamicamente e executar tarefas de forma assíncrona e autônoma.

Em se tratando de agentes móveis, existem aplicações que são baseadas em restrições de tempo. Desta forma, um agente teria a capacidade de negociar a qualidade do resultado em troca de atender o prazo estabelecido. Este tipo de agente teria a vantagem de diminuir o tráfego na rede, atuar de forma autônoma e assíncrona.

Uma aplicação que ilustra o uso de agentes móveis com restrições de tempo é um sistema de geração e transmissão de energia elétrica, onde os dados se propagam em grande escala, seguindo uma estrutura hierárquica. No controle e supervisão de um sistema energético, a grande escala geográfica e a resposta em tempo real são requisitos básicos, além de também possuir uma diversidade de equipamentos. Neste tipo de sistema podem existir milhares de nós que podem ser monitorados [16].

Segundo Rech *et al.* [17], um agente móvel poderia substituir um técnico (ou engenheiro) que com seu laptop visitaria seção por seção do sistema. Assim, em vez do técnico percorrer cada seção, seria o agente que visitaria um nó, coletaria os dados e decidiria o próximo nó a visitar para coletar os dados. O diagnóstico da rede seria atingido visitando alguns nós e coletando dados para se tomar a decisão.

Nesse tipo de aplicação, é possível utilizar caminhos mínimos sob restrições para encontrar uma rota satisfatória para o problema. Além disso, existe a possibilidade de alterar a rota durante o percurso para que se obtenha um melhor desempenho caso algo ocorra com um gasto de tempo maior ou menor do que o esperado.

9.1 Formulação do Problema

Um sistema distribuído é caracterizado por um conjunto N de nós e um conjunto R de tipos de recursos. Cada nó n_i possui um sub-conjunto r_i de recursos, isto é, $R_i \subseteq R$. Cada tipo de recurso pode ser encontrado em um ou mais nós, podem existir recursos duplicados desde que estejam em diferentes nós. A figura 9.1 ilustra um exemplo onde o recurso r_1 está presente em dois nós da rede.

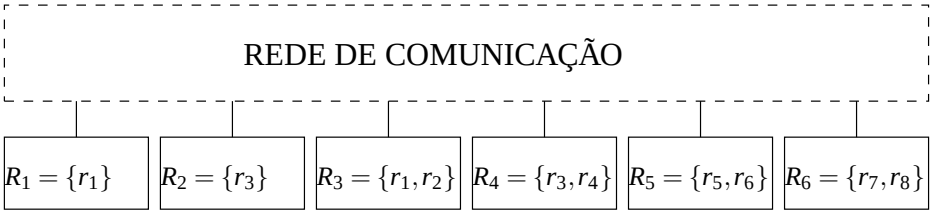

Figura 9.1: Recursos em um sistema distribuído

Um itinerário é definido como uma rota em um sistema distribuído. Ele descreve uma seqüência de nós $n_i \in N$. Um itinerário pode passar pelo mesmo nó mais de uma vez e não precisa incluir todos os nós.

Cada agente móvel Z tem uma missão M que define uma função B para cada tipo de recurso. Esta função determina quanto de benefício B_i cada tipo de recurso r_i contribui para a missão do agente. Em outras palavras $B \rightarrow \mathbb{R}^+$ é uma função dos recursos para os reais não negativos.

O agente tem o objetivo de maximizar o total de benefício obtido durante o itinerário no sistema. O agente deve começar no nó N_0 e deve retornar neste mesmo nó ao fim da missão. Cada missão M tem um *deadline* D associado para ser executado. O agente Z deve concluir a missão, retornando ao nó inicial antes do prazo D expirar, ou será perdido todo o benefício coletado.

O diagrama de recursos pode ser visto na figura 9.2, que corresponde a um diagrama de atividade UML que descreve a existência de relações de precedência sobre o uso dos recursos. O recurso do tipo r_0 é um pseudo-recurso, indicando que o agente deve iniciar e retornar ao nó N_0.

No diagrama da figura 9.2, a legenda $\langle\langle variável \rangle\rangle$ indica que a quantidade de tempo utilizado para obter o recurso r_1 é variável conforme o benefício, ou seja, quanto mais tempo o agente permanecer no nó maior será o benefício. Depois de obter o recurso r_1, o agente deve se obter o recurso r_2. Da forma que estão posicionados os recursos r_3, r_4 e r_5, mostram que o recurso r_3 deve ser obtido antes de r_4 e que o recurso r_5 pode ser obtido paralelamente aos outros dois. Ou seja, podem ocorrer as seguintes ordens de precedência para a obtenção dos recursos: $\langle r_3, r_4, r_5 \rangle$, $\langle r_3, r_5, r_4 \rangle$ e $\langle r_5, r_3, r_4 \rangle$. Em seguida, é o recurso r_6 que deve ser

obtido. Por seqüência, os recursos r_7 e r_8 podem ser obtidos ou não em qualquer ordem de precedência. For fim, deve se obter o pseudo-recurso r_0, que é retornar ao nó inicial.

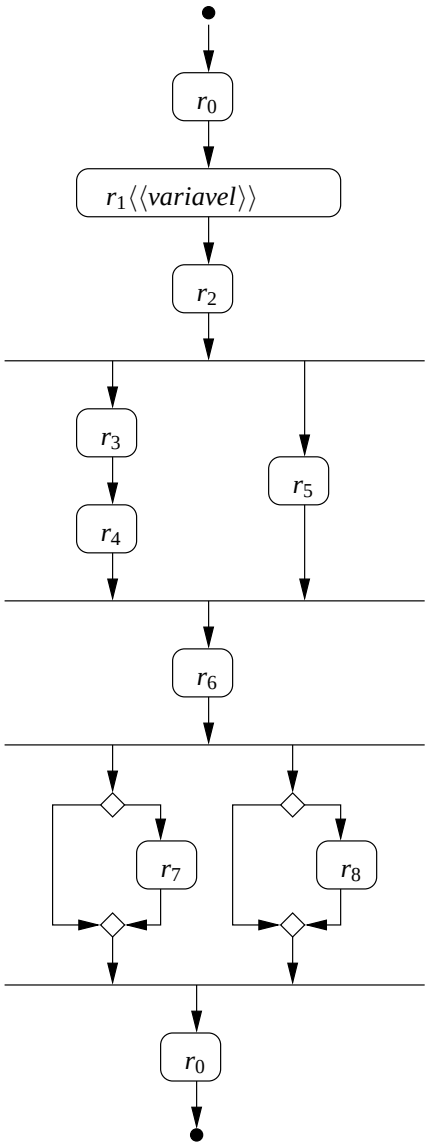

Figura 9.2: Missão em função dos recursos

O diagrama de nó para uma missão M corresponde a um diagrama de atividade UML construído a partir do diagrama de recursos para esta missão, onde cada recurso é substituído pelo nó ou nós que contém este recurso. A figura 9.3 corresponde ao diagrama resultante para a missão referente a figura 9.2.

Conforme a figura 9.3, o agente deve partir do nó inicial N_0 e seguir para o nó N_1 ou para o nó N_2, e no nó que escolheu obter o recurso r_0, permanecendo uma quantidade variável de tempo, gerando um benefício conforme a sua permanência. Em seguida, o agente deve seguir para o nó N_1 ou então permanecer lá caso já esteja neste nó, para obter o recurso r_2. Para obter os recursos r_3, r_4 e r_5, conforme a figura 9.2 as seqüências possíveis de visitas aos

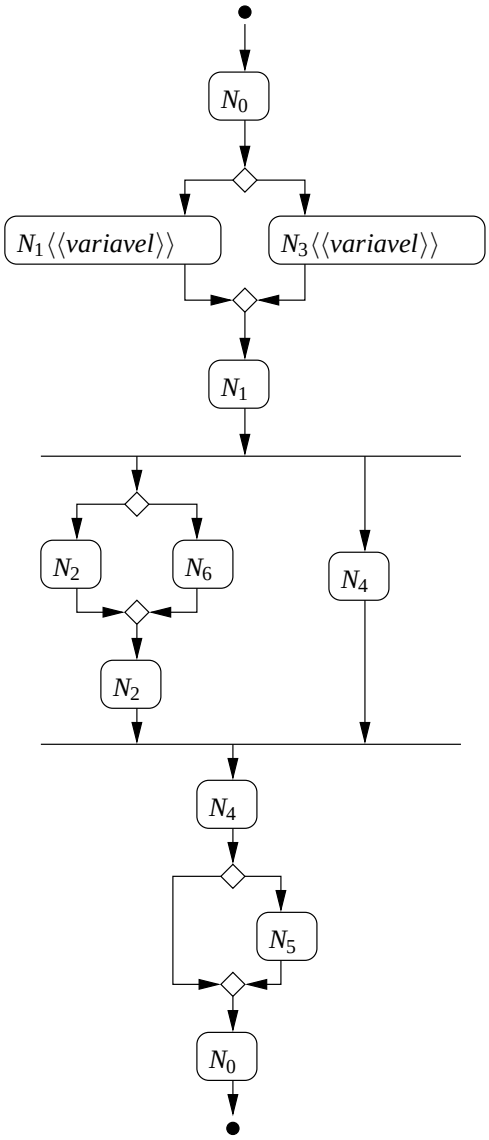

Figura 9.3: Missão em função dos nós

nós N_2, N_6 e N_4 são: $\langle N_2, N_2, N_4\rangle$, $\langle N_2, N_4, N_2\rangle$, $\langle N_4, N_2, N_2\rangle$, $\langle N_6, N_2, N_4\rangle$, $\langle N_6, N_4, N_2\rangle$ e $\langle N_4, N_6, N_2\rangle$. O agente então deve seguir para N_4, ou permanecer lá caso já esteja neste nó, para então obter o recurso r_6. Por fim, o agente tem a opção de seguir ou não para o nó N_5 para obter r_7 ou r_8 ou ambos. Desta forma, a missão estará completa e o agente deverá retornar ao nó inicial N_0.

Os tempos T_i para obtenção e os benefícios B_i de cada recurso estão na tabela 9.1. Para se deslocar de um nó para outro qualquer, T_i é igual a uma unidade de tempo. E o tempo para se manter no mesmo nó e começar a obter outro recurso é zero.

Para utilizar o problema de caminhos mínimos sob restrições foi necessário utilizar a formulação de grafo, que pode ser observada na figura 9.4. Para fazer esta formulação em grafo é necessário criar todas as possibilidades de nós que o agente pode tomar dentro da missão. É necessário relacionar a figura 9.2 com o tempo necessário para obter o recurso, o

Tabela 9.1: Benefício e tempo de obtenção dos recursos

Recurso	Benefício B_i	Tempo T_i
r_1	10	10
r_2	3	2
r_3	3	7
r_4	7	2
r_5	5	8
r_6	3	5
r_7	6	7
r_8	8	3

benefício gerado pelo recurso e o tempo de deslocamento entre dois locais.

Na figura 9.4, pode ser observado nas arestas entre parênteses, primeiramente o benefício B_i gerado pelo recurso e em seguida o tempo T_i necessário para obtê-lo. Quando uma aresta tiver colchetes estará indicando o recurso que está obtido. Lembrando que em algumas arestas não há deslocamento de local e nem obtenção de recurso, podendo ser apenas a mudança de recurso a ser obtido.

A partir do grafo da figura 9.4, pode ser aplicado programação dinâmica para o problema de caminhos mínimos sob restrições, sendo que neste caso a restrição é o tempo e os custos são os benefícios. Como se deseja maximizar o benefício, deve-se considerar que ele seja negativo no momento que seja aplicado programação dinâmica.

9.2 Algoritmos para Definição de Itinerários

Nesta seção são apresentadas brevemente as heurísticas utilizadas por Rech [17]. Estas heurísticas se baseiam apenas nos dados dos possíveis nós posteriores.

9.2.1 Algoritmo *Lazy*

O algoritmo *lazy* se baseia em tentar obter todos os recursos necessários o mais rapidamente possível. Ele ignora o benefício de cada recurso. Recursos opcionais não são obtidos, e recursos com benefício variável são executados com o menor tempo possível. Quando existem mais de uma rota, será escolhida a rota que tiver o menor tempo de execução, isto é, quando não existe ordem de precedência entre dois recursos, será escolhido o que apresentar o menor tempo de obtenção. Desta forma, o benefício pode ser mínimo, até mesmo zero. Em caso de existir recursos duplicados, primeiramente o agente verifica se existe o recurso

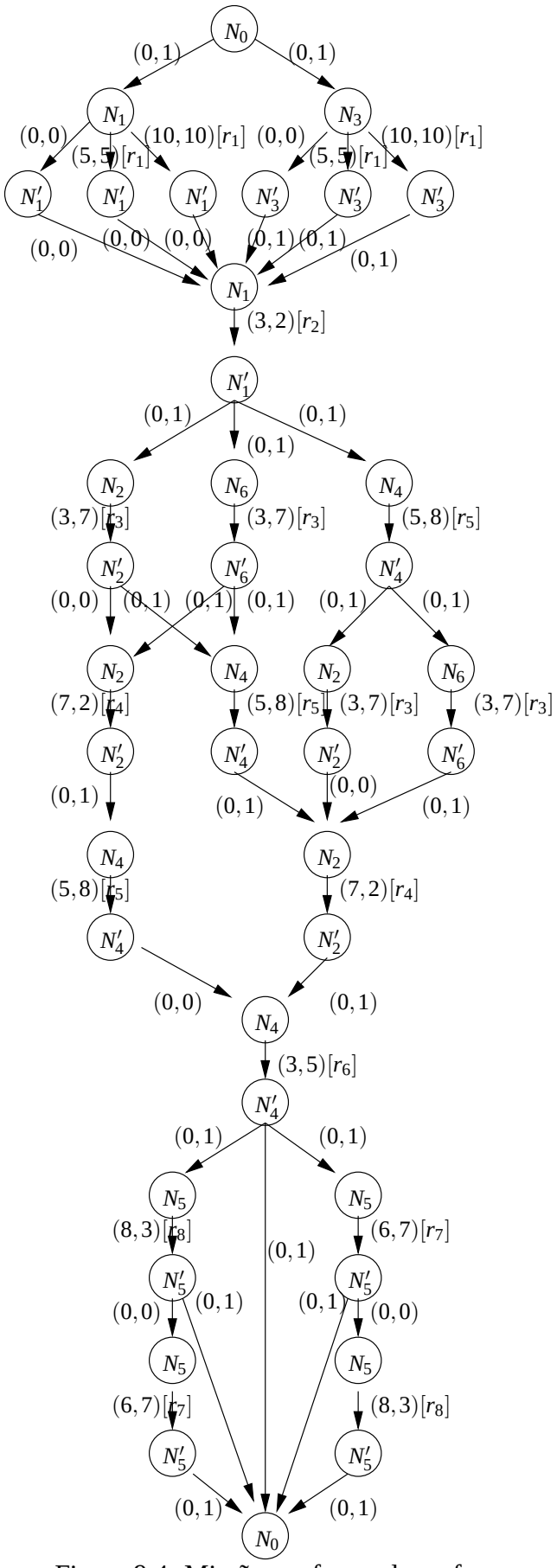

Figura 9.4: Missão em forma de grafo

no nó onde está e caso contrário escolhe o nó mais próximo que contém aquele recurso.

9.2.2 Algoritmo Guloso

O algoritmo guloso escolhe o nó que gera o maior benefício para a missão, ignorando o tempo gasto. Este algoritmo pode facilmente estourar o limite de tempo da missão. Em caso de benefício variável, o agente fica o tempo necessário para obter o máximo benefício daquele recurso. Quando existem recursos duplicados em diferentes nós, a preferência é para o nó mais próximo. Quando não existe ordem de precedência entre recursos, este algoritmo vai escolher aquele que oferece o maior benefício.

9.2.3 Algoritmo Aleatório

Este algoritmo escolhe aleatoriamente o próximo nó para ser visitado, não levando em conta nem o tempo utilizado, nem o benefício obtido para a missão.

9.2.4 Algoritmo de Maior Densidade

O algoritmo de maior densidade escolhe o nó que tem o recurso com a melhor relação benefício/tempo. Esta relação só é possível entre recursos sem relação de precedência, sendo definida por:

$db_i = B_i/(C_i + T_{ij})$

onde db_i é a densidade do benefício do recurso i, B_i é o tempo necessário para obter o recurso i, $T_{i,j}$ é o tempo necessário para o agente se mover do nó atual até o nó mais próximo que contenha o recurso i. Neste exemplo, como já foi dito o tempo de deslocamento é 1 para mudar de nó ou zero para se manter no mesmo nó.

O agente mantém um valor médio db até aquele momento. Quando um recurso é opcional, este será obtido apenas se sua densidade for maior que a densidade média da missão até aquele momento.

9.2.5 Versões com Tempo

Uma possível variação dos algoritmos citados é estimar o tempo necessário para executar o restante da missão antes de decidir o próximo recurso para ser obtido. Uma folga é calculada pela diferença entre o prazo restante da missão e o tempo de obtenção dos recursos obrigatórios, mais o tempo necessário para se deslocar entre os nós. Desta forma, os recursos opcionais serão obtidos apenas se a folga for suficientemente grande. Esta estimativa pode

Tabela 9.2: Comparação de itinerários

Algoritmo	Itinerário	Tempo	Benefício
Lazy	$\langle N_1, N_6, N_2, N_4, N_4 \rangle$	29	21
Guloso	$\langle N_1, N_1, N_4, N_6, N_2, N_4, N_5, N_5 \rangle$	51	25
Aleatório	$\langle N_3, N_1, N_4, N_2, N_2, N_4, N_5 \rangle$	48	37
Densidade	$\langle N_1, N_1, N_4, N_2, N_4, N_5 \rangle$	43	39

Tabela 9.3: Comparação de itinerários

	D=20	**D=30**	**D=40**	**D=50**	**D=60**
Guloso*	T=30,B=0(21)	T=30,B=21	T=40,B=31	T=48,B=37	T=51,B=45
Aleatório*	T=29,B=0(21)	T=29,B=21	T=37,B=28	T=48,B=42	T=48,B=42
Densidade*	T=29,B=0(21)	T=30,B=21	T=39,B=31	T=43,B=39	T=43,B=39
PD	T=28,B=0(21)	T=28,B=21	T=39,B=35	T=49,B=45	T=49,B=45

ser aplicada em todos os algoritmos descritos, com exceção do algoritmo *lazy*, que não é afetado por esta estimativa.

9.3 Comparação de Resultados

Primeiramente são apresentados na tabela 9.2 os resultados dos algoritmos, onde o *deadline* da missão não surte efeito no resultado. Em seguida, são apresentados na tabela 9.3 os resultados onde o *deadline* pode ser aplicado, sendo que o algoritmo $guloso^*$ é a versão com relógio do algoritmo *guloso*, e assim por diante.

Como o algoritmo de programação dinâmica para o problema de caminhos mínimos sob restrições calcula a rota ótima, já era esperado que os resultados fossem melhores do que heurísticas. Como pode ser observado na tabela 9.3, onde D é o *deadline*, o resultado da programação dinâmica é melhor ou igual aos resultados das heurísticas.

9.4 Sumário

Neste capítulo foi aplicado o problema de caminhos mínimos sob restrições para resolver o problema de escalonamento de agentes móveis em sistemas distribuídos. A missão de um agente foi modelada em forma de grafo para que o maior benefício pudesse ser obtido.

Diversos experimentos numéricos foram realizados, comprovando a eficiência do problema de caminhos mínimos sob restrições aplicado no contexto de agentes móveis.

Capítulo 10

Aplicação em Redes de Tráfego Veicular

Em redes de tráfego veicular, o RCSP pode ser aplicado quando se deseja ir de um ponto a outro com o menor tempo possível dentro de determinadas restrições, como gastos com pedágio, combustível e distância máxima. Além disso, pode-se colocar outras restrições estatísticas como segurança e possibilidade de congestionamentos.

Uma aplicação do RCSP em tráfego veicular é proposta por Jahn *et al.* [18]. Nesta proposta, o objetivo é minimizar o tempo total de percurso do sistema como um todo, e não apenas minimizar o tempo de percurso de cada usuário individualmente. Para atingir a eficiência global pode ser necessário que alguns usuários realizem caminhos piores do que realizariam normalmente, fazendo com que estes não seguissem as recomendações. Assim, Jahn *et al.* aplicam 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 maximiza a eficiência global do sistema.

10.1 Roteamento com Restrições do Usuário

Sistemas de informação e orientação de rota são projetados para auxiliar motoristas a tomarem decisões de rota. Alguns dispositivos podem fornecer informações, como congestionamentos, bloqueios ou acidentes, e outros podem dar recomendações, como “vire à esquerda” ou “deixe a rodovia na próxima saída”. Esta seção será concentrada nos dispositivos onde podem ser dadas recomendações de rota. O motorista entraria com o seu destino e o sistema calcularia a rota baseado nos mapas digitais e nas condições atuais de trânsito. A posição atual do veículo pode ser obtida através de GPS para ser fornecida ao sistema [19]. Tais dispositivos podem oferecer uma forma visual de orientação para indicar a melhor rota.

A maioria dos sistemas de navegação veicular funciona baseado em dados estáticos [20]. Estes sistemas com dados estáticos são pouco atualizados, e quando são, apenas a cartografia

é atualizada. Neste caso, o objetivo principal é fornecer informações para motoristas que não conhecem bem a área. Geralmente, levam em conta apenas as distâncias e o tempo de viagem aproximado.

Alguns sistemas de auxílio de rota mais sofisticados levam em conta informações da rede de trânsito atual. Para implementar sistemas deste tipo é necessário uma comunicação unidirecional ou bidirecional. Em comunicação de uma via, as condições de tráfego são obtidas através de sensores e transmitidas em *broadcast* para os dispositivos, os quais poderiam calcular a melhor rota baseado nas condições atuais. Com equipamentos com comunicação bidirecional, uma central de controle de tráfego receberia a posição atual dos usuários e seus destinos, podendo assim fazer certas distribuições de rotas aos motoristas. Estas rotas poderiam ser atribuídas aleatoriamente aos motoristas e transmitidas aos seus dispositivos de orientação.

A resposta às das condições de tráfego pode ser tratada de duas formas. A primeira é reativa [21], onde são consideradas apenas as condições naquele exato momento. Os sistemas baseados no modo reativo têm a vantagem de responderem mais rapidamente às mudanças e são conceitualmente mais simples.

A segunda forma é baseada em previsões de como as condições de tráfego estarão no futuro [22]. Neste modelo, quando a utilização da orientação de rota for baixa, os efeitos do próprio sistema podem ser ignorados. Quando esta utilização for alta, seus efeitos terão que ser considerados. Além disso, existem variáveis como quanto os motoristas respeitarão as indicações, deixando as decisões ainda mais complexas.

Segundo Bottom [20], alguns sistemas calculam o menor caminho, outros os k menores caminhos para um parâmetro k dado. Outros sistemas fazem o cálculo *online*, outros ainda fazem o pré-processamento dos dados. Uma possibilidade alternativa é atribuir o caminho mais rápido para cada usuário sob as condições correntes, esta solução é chamada de solução *user optimal* ou *user equilibrium*. Uma alternativa é minimizar o tempo total do sistema, uma solução chamada de *system optimal*.

Existem diversas propostas para orientação de rota tanto buscando o ótimo para o usuário ou buscando o ótimo para o sistema. Entretanto os sistemas têm se polarizado para utilizar o ótimo do usuário [18].

Além disso, é bem entendido que para atingir o ótimo global do sistema, alguns usuários poderão fazer uma jornada maior que o usual [23]. Desta forma, é bem provável que muitos usuários não aceitariam sugestões que poderiam ser tão ou mais ineficientes do que as suas próprias escolhas.

Neste contexto, a proposta de Jahn *et al.* [18] se encaixa para tentar resolver este tipo de problema. Nesta proposta são submetidas restrições de caminhos aceitáveis ao usuário, ou seja, que não tenham uma performance muito degradada para a melhoria do sistema como

um todo.

Aqui está inserido o desenvolvimento desta dissertação, ou seja, no cômputo de caminho mínimo que tenham como restrições degradações máximas definida pelo usuário, buscando ao final a máxima eficiência do sistema.

O problema que está sendo proposto é conhecido como *constrained system optimum* (CSO) e pode ser formulado da seguinte forma:

$$\text{Min} \quad C(x) = \sum_{a \in A} l_a(x_a) x_a \tag{10.1a}$$

S. a :

$$\sum_{P \in \mathscr{P}_k^{\varphi}} x_P = d_k, \qquad \forall k \in K \tag{10.1b}$$

$$\sum_{P \in \mathscr{P}^{\varphi} : a \in P} x_P = x_a, \qquad \forall a \in A \tag{10.1c}$$

$$x_P \geqslant 0, \quad P \in \mathscr{P}^{\varphi} \tag{10.1d}$$

onde:

- x_a é a taxa de tráfego através do arco a;
- $l_a(x_a)$ é uma função que mapeia a taxa de tráfego ao tempo de percurso/trajeto, segundo o *U.S. Bureau of Public Roads* (1964):

$$l_a(x_a) = l_a^0 \left(1 + \alpha \left(\frac{x_a}{c_a} \right)^{\beta} \right) \tag{10.2}$$

  onde $l_a^0 > 0$ é o tempo de trajeto em rede não congestionada (*free-flow travel time*), e $\alpha \geqslant 0$, $\beta \geqslant 0$ e c_a são parâmetros;
- K é conjunto de *commodities*, onde $k \in K$ representa o fluxo de veículos do nó s_k para o nó t_k, ou seja, o par origem-destino $(s_k, t_k) \in V \times V$.
- d_k representa a demanda de fluxo associada ao par origem-destino (s_k, t_k), isto é, ao *commodity k* (veículos por unidade de tempo);
- $\mathscr{P}_k = \{P : P \text{ é um caminho direto de } s_k \text{ para } t_k\}$ é o conjunto de caminhos origem-destino para o par k;
- $\mathscr{P} = \bigcup_{k \in K} \mathscr{P}_k$ é o conjunto de todos os caminhos.

Para um fluxo $x = [x_a : a \in A]$ e um caminho P, o tempo de trajeto neste caminho é dado por $l_P(x) = \sum_{a \in P} l_a(x_a)$, enquanto $\tau_P = \sum_{a \in P} \tau_a$ é o comprimento de percurso normal.

Considera-se soluções para as quais o tempo de percurso normal de qualquer caminho origem-destino do par k não é muito maior que o caminho mais curto s_k para t_k (com respeito aos tempos normais). Mais especificamente, para uma tolerância $\varphi \geqslant 1$, um caminho $P \in \mathscr{P}_k$ é factível se $\tau_P \leqslant \varphi T_k$ onde $T_k := \min_{P \in \mathscr{P}_k} \tau_P$ é o comprimento do trajeto mais curto de s_k para t_k.

Assim $\mathscr{P}_k^{\varphi}$ é o conjunto de caminhos origem-destino para o par k que são factíveis (quanto à tolerância φ) e $\mathscr{P}^{\varphi} = \bigcup_{k \in K} \mathscr{P}_k^{\varphi}$ é o conjunto de todos os caminhos factíveis.

Cada arco $a \in A$ possui dois parâmetros: l_a (tempo de percurso) e τ_a (comprimento de arco). Dado um par origem-destino (s_k, t_k), o objetivo é computar o caminho $s_k - t_k$ mais rápido cujo comprimento não exceda um limite superior φT_k. Ou seja, queremos resolver o problema:

$$\text{Min}\{l(P) : P \text{ é um caminho de } s_k \text{ para } t_k \text{ tal que } \tau_P \leqslant \varphi T_k\} \tag{10.3}$$

Este problema nada mais é que o problema de caminhos mínimos sob restrições e pode ser resolvido pelos métodos apresentados neste trabalho. Neste problema Jahn *et al.* utilizam o algoritmo de Dijkstra estendido para o caso de duas funções objetivo que foi apresentado por Aneja *et al.* [24].

10.2 Sumário

O problema de caminhos mínimos sob restrições já vem sendo estudado para aplicações em tráfego urbano, seja para resolver um sub-problema de rotemeamento completo de uma malha viária ou então para resolver o roteamento de um único veículo com algum tipo de retrição. Neste breve capítulo foram apresentadas algumas aplicações encontradas na literatura onde este problema pode ser aplicado em tráfego urbando.

Capítulo 11

Redes de Computadores

A Qualidade de Serviço (QoS) em redes é um aspecto de implantação e operação importante para as redes de pacotes como um todo e para as redes IP em particular. Tem um aspecto operacional fundamental para o desempenho fim-a-fim de determinadas aplicações, como vídeo, áudio e dados.

Existem redes de computadores integradas que estão oferecendo garantias de QoS fim-a-fim para diversas aplicações. Algumas aplicações possuem diversos requisitos em termos de banda, atraso e perdas. Um dos mais importantes problemas em serviços baseado em QoS é determinar uma rota que satisfaz requisitos de uma conexão enquanto atinge uma utilização eficiente dos recursos da rede. Este problema é conhecido como roteamento baseado em QoS [25].

Segundo Chen [26], o processo de roteamento consiste de duas etapas básicas. Primeiro coletar informações de estado da rede e em seguida com estas informações encontrar um caminho factível. O que será discutido nesta seção será a segunda tarefa, onde já seriam conhecidas informações sobre os nós e se concentraria em determinar um caminho que satisfaz os requisitos.

Considerando uma rede que é representada por um grafo direto $G = (N,A)$, onde N é o conjunto de nós e A é o conjunto de arestas (arcos). Cada aresta $(i, j) \in A$ é associada com M pesos não negativos (valores QoS aditivos, como tempo) $w_k(i, j)$ para $k = 1,2,...,M$. Com M restrições recebidas como entrada $C_k, k = 1,2,...,M$, o problema de encontrar um caminho p que possui uma origem s e um destino t tal que o número de nós em p seja mínimo, pode ser

modelado da seguinte forma:

$$\text{Min} \quad \sum_{(i,j)\in p} 1 \tag{11.1a}$$

S. a :

$$\sum_{(i,j)\in p} w_k(i,j) \leqslant C_k \qquad \textit{para } k = 1,2,...,M \tag{11.1b}$$

Este problema foi tratado por Korkmaz e Krunz [9] utilizando algoritmos randomizados. Nesta formulação não foram tratadas restrições não-aditivas, como largura de banda, já que este tipo de restrição poderia ser facilmente acomodada. Para resolver isso seria necessário excluir os arcos que não satisfazem a restrição.

Este problema é bastante parecido com o problema de caminhos mínimos sob restrições. Uma diferença é que se deseja minimizar o número de *hops* em uma transmissão fazendo com que o valor a ser minimizado em cada arco seja 1, diferentemente no problema de caminhos mínimos sob restrições, onde este valor é variável.

Para resolver este problema de QoS com caminhos mínimos sob restrições não é necessário fazer nenhuma alteração no algoritmo que poderia resolver este problema com certa eficiência, garantindo que a solução seria ótima.

Um outro aspecto quando se trata de QoS, é que existem restrições que não são aditivas, ou seja, a restrição não se faz pela soma dos valores de cada arco e sim pelo menor valor encontrado em um arco pertencente a um trajeto. Um bom exemplo deste tipo de restrição é a largura de banda, na qual a restrição se impõe no arco com menor capacidade de transmissão.

Quando se tratando de restrições não aditivas, o algoritmo de programação dinâmica primal 3.1 não poderia ser utilizado, já que a sua recursão tem como variável a quantidade de recursos disponíveis na restrição. Já o algoritmo dual 3.7 poderia ser muito bem utilizado com algumas adaptações para este tipo de restrição, assim como a sua aproximação-ε 5.2.

A adaptação necessária é trocar a somatória dos recursos utilizados em todas as arestas que fazem parte do caminho pelo menor valor de recursos encontrado entre as arestas que fazem parte do caminho. Desta forma, o algoritmo de programação dual poderia resolver o problema de roteamento com QoS.

Para resolver o problema de roteamento para restrições QoS, Korkmaz e Krunz alteram a formulação do problema retirando a função objetivo por uma restrição de número máximo

de *hops* em um caminho. Desta forma, o problema fica da seguinte forma:

$$\sum_{(i,j)\in p} 1 \leqslant H \tag{11.2a}$$

$$\sum_{(i,j)\in p} w_k(i,j) \leqslant C_k \qquad \textit{para } k = 1,2,...,M \tag{11.2b}$$

onde H é o limite de *hops* que um caminho p pode ter. Sendo que seu valor pode ser entre 0 e $n-1$, onde n é o número de nós no grafo.

Este problema continua sendo *NP*-Díficil e os autores resolvem este problema com heurísticas e algoritmos randomizados, com resultados bastante satisfatórios.

11.1 Sumário

O problema de caminhos mínimos sob restrições pode ser utilizado principalmente para criar roteamentos ótimos que cumpram as métricas QoS. Neste capítulo foram introduzidos alguns tópicos onde o problema teria aplicação prática na área de redes de compoutadores.

Parte IV

Conclusões e Trabalhos Futuros

Nesta dissertação, o problema de caminhos mínimos sob restrição foi modelado formalmente. Podendo este modelo ser utilizado em programação matemática para resolver instâncias através de ferramentas de otimização. Além disso, foi provado que o problema de caminhos mínimos se torna *NP*-Difícil com a adição de apenas uma restrição.

Neste trabalho foi implementado o algoritmo de programação dinâmica que resolve o problema de forma ótima, mas no qual o tempo de execução cresce não só em função do tamanho da instância, mas também em função da quantidade de recursos disponíveis. Para sobrepor este obstáculo foi implementado a aproximação-ε que garante que o tempo de execução seja polinomial em função da quantidade de vértices, sendo pouco dependente da quantidade de recursos disponíveis. Esta aproximação se mostrou bastante dependente do valor de ε, sendo que quando se deseja um resultado que tenha maior garantia de estar próximo do ótimo o valor de ε deve ser ajustado para próximo de 0 e quando se deseja encontrar um resultado mais rapidamente o valor de ε deve ser ajustado para um valor próximo de 1.

O estudo comparativo entre programação matemática e programação dinâmica mostra que para simplesmente encontrar o caminho entre dois pontos, a programação matemática pode ser mais eficiente dependendo da ferramenta de otimização utilizada. Nos experimentos utilizando CPLEX, a programação matemática foi mais eficiente. Já a programação dinâmica tem a vantagem de encontrar o caminho ótimo, se existir, para todos os destinos disponíveis. Ou então, a partir de um destino, encontrar o caminho ótimo para todas as possíveis origens. Um outro aspecto importante, é verificar a disponibilidade de algumas ferramentas de otimização, que geralmente são pouco acessíveis.

Uma outra discussão se torna necessária quando a função objetivo é não aditiva. Neste caso, é feita uma pequena alteração na recursão de programação dinâmica original para que este funcione também neste caso. Já quando existem restrições não aditivas, o problema pode ser resolvido facilmente eliminando os arcos que não satisfazem as restrições e mantendo o restante do problema igual, resolvendo através de caminho mínimo simples.

Um outro tópico abordado é a associação do problema de caminhos mínimos sob restrições com problemas estocásticos, no qual o consumo de recursos é uma variável aleatória. Neste caso, foi possível transformar uma variável probabilística em uma determinística, mas com isso foi gerada uma não linearidade na restrição, fazendo com que não fosse possível aplicar os métodos apresentados neste trabalho. Mas seria possível, caso fosse obtido um limite superior da variável não linear.

O problema de caminhos mínimos sob restrições pode ser aplicado com eficiência em agentes móveis, garantindo um escalonamento ótimo para problemas onde se sabe o tempo de execução previamente de cada tarefa. Caso o tempo de execução possa mudar durante a tarefa, ainda é possível utilizar a tabela gerada pelo algoritmo de programação dinâmica para gerar um novo itinerário. Isso só é possível se o agente tiver a capacidade de armazenar esta

tabela. Dessa forma, o itinerário completo pode deixar de ser ótimo e também pode deixar de ser factível caso despenda mais tempo para executar uma tarefa do que o previsto.

Uma outra aplicação é em tráfego urbano, onde pode ser utilizado para resolver um subproblema de uma solução para roteamento completo de tráfego urbano. Ou, até mesmo, ser utilizado diretamente no problema, onde existe um *deadline* para a chegada e se deseja minimizar o custo de um trajeto. Ou ainda, quando se tem limitação orçamentária com gasto de combustível e pedágios e se deseja minimizar o tempo despendido no trajeto.

Por fim, existem aplicações em redes de computadores, principalmente se tratando de QoS. Neste o problema de roteamento com QoS, pode ser resolvido com o problema de caminhos mínimos. Tratando-se apenas com restrições aditivas o roteamento pode ser resolvido sem alterações nos algoritmos implementados neste trabalho. Com restrições não aditivas, como largura de banda, pode ser tratado apenas excluindo os arcos que não satisfazem a restrição.

Desta forma, fica demonstrada a importância do estudo deste tipo de problema, o qual pode auxiliar na melhoria da qualidade das soluções em aplicações em contextos diferentes da pesquisa operacional, muitas vezes substituindo soluções onde técnicas menos adequadas são empregadas.

Como sugestão de trabalhos futuros é apontado o estudo do problema de caminhos mínimos associado a problemas estocásticos, onde o consumo de recursos é probabilístico. Em tais problemas, a solução encontrada deverá ser viável com uma certa garantia probabilística. Foi demonstrando neste trabalho que se pode transformar uma restrição probabilística em uma restrição determinística, mas esta restrição se torna não linear. Assim um possível trabalho futuro seria encontrar limites para esta restrição ou então uma outra abordagem diferente da programação inteira. Uma outra linha sugerida é a classe dos algoritmos randomizados, que também podem ser empregados na solução dos problemas discutidos nesta dissertação. Também fica como sugestão a continuidade do estudo do problema com diversas restrições.

Referências Bibliográficas

[1] DIJKSTRA, E. W. A Discipline of Programming. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997. ISBN 013215871X.

[2] ZIEGELMANN, M. Constrained Shortest Paths and Related Problems. Tese (Doutorado) — Universität des Saarlandes, Julho 2001.

[3] JOKSCH, H. C. The shortest route problem with constraints. Journal of Mathematical Analysis and Applications, v. 14, p. 191–197, 1966.

[4] BEASLEY, J. E.; CHRISTOFIDES, N. An algorithm for the resource constrained shortest path problem. Networks, v. 19, p. 379–394, 1989.

[5] HASSIN, R. Approximation schemes for the restricted shortest path problem. Math. Oper. Res., INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 17, n. 1, p. 36–42, 1992. ISSN 0364-765X.

[6] HANDLER, G. Y.; ZANG, I. A dual algorithm for the constrained shortest path problem. Networks, v. 10, p. 293–310, 1980.

[7] MEHLHORN, K.; ZIEGELMANN, M. Resource constrained shortest paths. In: ESA '00: Proceedings of the 8th Annual European Symposium on Algorithms. London, UK: Springer-Verlag, 2000. p. 326–337. ISBN 3-540-41004-X.

[8] GAREY, M. R.; JOHNSON, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN 0716710455.

[9] KORKMAZ, T.; KRUNZ, M. A randomized algorithm for finding a path subject to multiple QoS requirements. Computer Networks (Amsterdam, Netherlands: 1999), v. 36, n. 2–3, p. 251–268, 2001.

[10] CAMPONOGARA, E. Métodos de Otimização: Teoria e Prática. Florianópolis, BR: [s.n.], 2004.

[11] FOURER, R.; GAY, D.; KERNIGHAN, B. AMPL: A Mathematical Programming Language. 1989. Disponível em: <citeseer.ist.psu.edu/fourer89ampl.html>.

[12] CPLEX A DIVISION OF ILOG, INC. CPLEX: Using the CPLEX Callable Library. [S.l.], 1997.

[13] CORMEN, T. H. et al. Introduction to Algorithms. [S.l.]: McGraw-Hill Higher Education, 2001. ISBN 0070131511.

[14] GARCIA, A. L. Probability and Random Processes for Electrical Engineering. Reading, MA: Addison-Wesley, 1994. ISBN 020150037X.

[15] BAEK, J.-W.; KIM, G.-T.; YEOM, H.-Y. Timed mobile agent planning for distributed information retrieval. In: AGENTS '01: Proceedings of the fifth international conference on Autonomous agents. New York, NY, USA: [s.n.], 2001. p. 120–121.

[16] TOLBERT, L. M.; QI, H.; PENG, F. Z. Scalable multi-agent system for real-time electric power management. In: IEEE Power Engineering Society Summer Meeting. Vancouver, Canada: [s.n.], 2001. p. 1676–1679.

[17] RECH, L.; OLIVEIRA, R. S.; MONTEZ, C. Dynamic determination of the itinerary of mobile agents with timing constrains. In: The 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05). Compiègne, France: [s.n.], 2005.

[18] JAHN, O. et al. System-optimal routing of traffic flows with user constraints in networks with congestion. Oper. Res., INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 53, n. 4, p. 600–616, 2005. ISSN 0030-364X.

[19] HENRY, J. J.; CHARBONNIER, C.; FARGES, J. L. Route guidance. In: PAPAGEORGIOU, M. (Ed.). Concise Encyclopedia of Traffic and Transportation Systems. Oxford, UK: Pergamon Press, 1991. p. 417–422.

[20] BOTTOM, J. A. Consistent anticipatory route guidance. Tese (Doutorado) — Massachusetts Institute of Technology, Cambridge, MA, 2001.

[21] FRIESZ, T. L. et al. A variational inequality formulation of the dynamic network user equilibrium problem. Oper. Res., INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 41, n. 1, p. 179–191, 1993. ISSN 0030-364X.

[22] KAUFMAN, D. E.; SMITH, R. L.; WUNDERLICH, K. E. An iterative routing/assignment method for anticipatory real-time route guidance. In: SOCIETY OF AUTOMOTIVE ENGINEERS. IEEE Vehicle Navigation and Information Systems Conf. Warrendale, PA, 1991. v. 2, p. 693–700.

[23] MAHMASSANI, H. S.; PEETA, S. Network performance under system optimal and user equilibrium assignments: Implications for advanced traveler information systems. Transportation Research Record, n. 1408, p. 83–93, 1993.

[24] ANEJA, Y. P.; AGGARWAL, V.; NAIR, K. P. K. Shortest chain subject to side constraints. Networks, v. 25, p. 295–302, 1983.

[25] AURRECOECHEA, C.; CAMPBELL, A. T.; HAUW, L. A survey of QoS architectures. Multimedia Systems, v. 6, n. 3, p. 138–151, 1998.

[26] CHEN, S.; NAHRSTEDT, K. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. 1998. Disponível em: <citeseer.ist.psu.edu/chen98overview.html>.