UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO — UFERSA
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS VINICIUS AMARAL DE OLIVEIRA

ALGORITMOS DO TIPO SIMPLEX PARA PROBLEMAS DE FLUXO DE CUSTO MÍNIMO

Mossoró
2015

Lucas Vinicius Amaral de Oliveira

ALGORITMOS DO TIPO SIMPLEX PARA PROBLEMAS DE FLUXO DE CUSTO MÍNIMO

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal Rural do Semi-Árido — UFERSA

Orientador:
Prof. Dr. Judson S. Santiago

Mossoró
2015

O conteúdo desta obra é de inteira responsabilidade de seus autores

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central Orlando Teixeira (BCOT)
Setor de Informação e Referência

O48a Oliveira, Lucas Vinicius Amaral de

Algoritmos do tipo simplex para problemas de fluxo de custo mínimo / Lucas Vinicius Amaral de Oliveira — Mossoró, 2015.
65 f.: il.

Orientador: Prof. Dr. Judson S. Santiago

Monografia (Graduação em Ciência da Computação) — Universidade Federal Rural do Semi-Árido. Pró-Reitoria de Graduação.

1. Programação. 2. Criss-cross 3. Fluxo de custo mínimo. 4. Simplex. I. Título.

RN/UFERSA/BCOT/161-15 CDD 005.1

Bibliotecária: Vanessa de Oliveira Pessoa
CRB 15/453

Lucas Vinicius Amaral de Oliveira

ALGORITMOS DO TIPO SIMPLEX PARA PROBLEMAS DE FLUXO DE CUSTO MÍNIMO

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal Rural do Semi-Árido — UFERSA

APROVADO EM: 02/02/2015

BANCA EXAMINADORA

_____

Prof. Dr. Judson Santos Santiago – UFERSA
Orientador e Presidente

_____

Prof. Dr. Tibérius de Oliveira e Bonates – UFC
Primeiro Membro

_____

Prof. M.Sc. Paulo César Linhares da Silva – UFERSA
Segundo Membro

A minha família, que é a base sólida de minha vida.

If I have seen farther than others,
it is because I stood on the shoulders of giants.
— SIR ISAAC NEWTON

AGRADECIMENTOS

A Deus.

Aos meus amados pais, João e Clivanete, que sempre me apoiaram em todas as minhas escolhas e me incentivaram a ir mais longe. Sua dedicação e exemplo me fizeram o que sou hoje.

Aos meus queridos irmãos, Gemison, Matheus e George e irmãs, Luana e Jeane, pela paciência e incentivo que sempre me deram. A presença de cada um é sempre confortante e divertida.

Aos meus amigos de longa data, pelos momentos de descontração sempre bem vindos quando as coisas ficavam difíceis.

Aos meus colegas de faculdade, que dividiram as dificuldades e partilharam as conquistas. Alguns se tornaram grandes amigos que vou ter comigo a vida toda.

Ao meu orientador Prof. Judson Santiago, exemplo de profissional e de pessoa, que sempre esteve disposto a ajudar qualquer aluno que o procurasse e, de maneira não diferente, a me ajudar neste trabalho.

Ao professor Tibérius Bonates, pelo apoio e o incentivo que me deu durante toda a graduação e por toda a ajuda que, mesmo de longe, sempre esteve disposto a me prestar.

Ao professor Paulo César, integrante da banca, que contribuiu para a conclusão deste trabalho.

Ao professor José Figueira, um dos idealizadores deste projeto, pelo apoio e incentivo. Sua contribuição foi fundamental para o sucesso deste trabalho.

A todos os professores da graduação, que me ensinaram mais que apenas conteúdo. A lição mais importante sempre é a amizade que mesmo na relação professor aluno é possível construir.

Aos meus parentes, que sempre desejaram meu sucesso, assim como desejo o de todos eles.

A “la casa”.

"*A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original*".
(Albert Einstein)

RESUMO

Os problemas de fluxo de custo mínimo representam uma importante classe de problemas de programação linear. Os algoritmos da família simplex, principal algoritmo utilizado atualmente para resolver problemas de programação linear, apresentam uma deficiência: a dificuldade de se obter uma base inicial viável. Essa deficiência do simplex impulsionou o surgimento do algoritmo criss-cross, que pode caminhar por bases inviáveis e não necessita de uma base inicial viável. A versão finita do criss-cross para problemas de programação linear clássicos utiliza a regra de pivoteamento do menor índice. Implementamos o criss-cross com a regra de menor índice adaptada para problemas de rede, como o problema de fluxo de custo mínimo. Sugerimos algumas modificações na regra do menor índice e conseguimos uma grande melhoria de desempenho no algoritmo. Realizamos testes e comparamos o desempenho das variantes do criss-cross, de uma implementação própria do simplex de rede e da implementação do simplex disponibilizada pela ferramenta CPLEX. Apresentamos detalhes da implementação e discutimos os resultados obtidos.

Palavras-chave: programação linear. simplex. fluxo de custo mínimo. criss-cross. regra de pivoteamento.

ABSTRACT

The minimum cost flow problems represent an important class of linear programming problems. The simplex algorithms, the most used tool to solve linear programming problems, present a deficiency: the cost of finding a feasible initial basis. This deficiency of the simplex algorithm gave rise to the emergence of the criss-cross algorithm, which can move through unfeasible bases and does not require a feasible initial basis. The finite criss-cross for classic linear programming problems uses the minimum index rule. We implemented the criss-cross with an adaptation of the minimum index rule for network problems, such as minimum cost flow problems. We also suggest some modifications to the minimum index rule that substantially improved the performance of the algorithm. We performed some tests and performance comparison using variations of the criss-cross algorithm, as well as our own implementation of the simplex algorithm and the simplex implementation available on the CPLEX tool. We present implementation details and discuss the tests results.

Keywords: linear programming. simplex. minimum cost flow. criss-cross. pivoting rule.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

pred. predecessor

prof. profundidade

LISTA DE SIGLAS

FCM	Fluxo de Custo Mínimo
EAG	Estrutura de Árvore Geradora
PL	Programação Linear
GPS	Global Positioning System
PO	Pesquisa Operacional

SUMÁRIO

1 INTRODUÇÃO

"The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man."

George Bernard Shaw

A pesquisa operacional (PO) constitui uma das principais áreas da matemática aplicada. Ao longo do último século, a PO passou por um período de desenvolvimento. O principal feito foi a descoberta de um método com bom desempenho prático para resolver problemas de programação linear (PL), o método simplex.

Atualmente os softwares para resolver problemas de PL são amplamente utilizados em grandes empresas. Problemas de PL são de grande aplicabilidade prática e podem ser utilizados para resolver os mais diversos problemas da vida real. Esse problemas vão desde a otimização da utilização de recursos de produção, de forma a reduzir custos, até a descoberta de rotas de distribuição de mercadorias que gerem menor custo. Este último exemplo faz parte de uma classe muito importante de problemas de PL, os problemas de fluxo de custo mínimo (FCM).

Os problemas de FCM, apesar de representarem uma classe dos problemas de PL, são problemas de rede e apresentam características bem distintas. Fazem parte dos problemas de rede os problemas reais que envolvem otimização, por exemplo, em redes de rodovias, redes elétricas, redes de computadores, redes de relacionamento e os mais variados tipos de rede. A maioria dos problemas de rede podem ser traduzidos no problema de FCM. Os algoritmos para resolver problemas de PL comuns geralmente são menos eficientes que os algoritmos especializados ao resolver problemas de rede. Devido a isso, surgiu o simplex de rede, uma adaptação do simplex exclusivo para resolver problemas de FCM.

1.1 Motivação

Desde sua descoberta o simplex vem sendo amplamente estudado e aprimorado. No entanto, ainda apresenta pelo menos uma deficiência, do ponto de vista prático: a necessidade de se obter uma solução inicial viável para só então iniciar a busca pela solução ótima. A busca pela solução inicial viável pode consumir muito tempo e chegar a ser tão custosa quanto encontrar a solução ótima. Essa deficiência motivou pesquisas com objetivo de tornar mais eficientes as técnicas de busca pela solução inicial viável e também de encontrar métodos que pudessem iniciar sem tal solução viável.

No fim dos anos 60 foi mencionado pela primeira vez o algoritmo criss-cross. Esse algoritmo, assim como o simplex, utiliza a técnica de pivoteamento, mas é capaz de resolver problemas de PL sem a necessidade de uma solução inicial viável. É possível encontrar alguns estudos sobre o criss-cross que mostram regras de pivoteamento finitas (Fukuda e Terlaky

(1997)) e também comparações de desempenho do método (Bonates (2001), Bonates e Maculan (2003)).

O método criss-cross ainda é pouco conhecido e estudado. Apesar do tempo de existência, não existe na literatura uma versão do criss-cross para resolver problemas de FCM. O objetivo deste trabalho é desenvolver uma versão do criss-cross com a adaptação da regra de pivoteamento para resolver problemas de rede. E também realizar uma comparação de desempenho entre o criss-cross proposto neste trabalho e o simplex de rede.

1.2 Organização do texto

No Capítulo 2 fazemos uma introdução ao problema de FCM e mostramos alguns exemplos de aplicações práticas. No Capítulo 3 descrevemos a implementação do algoritmo simplex de rede e no capítulo seguinte discutimos uma técnica para se obter a solução inicial viável necessária para a inicialização do simplex. No Capítulo 4 mostramos uma possível implementação do algoritmo criss-cross com a regra de pivoteamento adaptada para problemas de rede. Mostramos ainda algumas alterações na regra de pivoteamento que levaram a um grande ganho de desempenho. O Capítulo 5 apresenta os resultados dos testes realizados bem como uma pequena análise sobre o comportamento dos algoritmos. E por fim, no Capítulo 6 fazemos algumas considerações finais com uma análise geral sobre os resultados obtidos no trabalho e algumas indicações de trabalhos futuros na área.

2 FLUXO DE CUSTO MÍNIMO (FCM)

Segundo Papadimitriou e Steiglitz (1998), os problemas de fluxo de custo mínimo representam uma classe geral e muito importante de problemas de rede, pois apresentam restrições de capacidade e custo não trivial. Isso significa que os problemas pertencentes a essa classe são mais próximo dos problemas reais, onde os recursos apresentam capacidade limitada e custo de utilização.

Vários problemas reais podem ser traduzidos em problemas de FCM. Isso o torna um problema com grande aplicabilidade prática e de grande importância comercial, e o estudo de algoritmos cada vez mais eficientes para resolvê-lo tem sido objeto de estudo de vários pesquisadores ao longo dos anos. Outra evidência de sua importância é que o resolvedor de problemas de programação linear líder do mercado oferece um módulo dedicado ao problema de FCM, ao invés de tratá-lo como um caso particular do problema de PL.

2.1 Definição do problema de FCM

Considere o grafo $G = (V, A)$ e a rede $N = (s, V, A, l, u)$ definida sobre G, sendo s, l e $u \in R^m$. Considere, também, um custo c_{ij} para cada arco $(i, j) \in A$. Queremos enviar uma quantidade de fluxo através da rede. A demanda de fluxo no nó $i \in V$ é dada por s_i. Se $s_i > 0$ significa que o nó i "produz" s_i unidades de fluxo que devem ser enviadas através da rede. Se $s_i < 0$ significa que o nó i "consome" s_i unidades de fluxo. O fluxo deve sair dos nós produtores e chegar nos nós consumidores. Se $s_i = 0$, não é produzido nem consumido fluxo no nó i, ou seja, a quantidade de fluxo que entra deve ser a mesma que sai do nó.

Podemos então definir o problema do fluxo de custo mínimo.

Definição 2.1 *O problema do FCM é encontrar o fluxo x_{ij} em cada arco $(i, j) \in A$ que minimize uma dada função linear de custos, sujeito às restrições de limites de fluxo e respeitando a conservação da demanda s em cada nó $i \in V$.*

Em termos de PL, o problema define-se por

$$minimizar \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1}$$

sujeito às restrições

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(i,j)\in A} x_{ji} = s_i, \qquad \forall i \in V, \tag{2.2}$$

$$l_{ij} \leq x_{ij} \leq u_{ij}, \qquad \forall (i, j) \in A, \tag{2.3}$$

onde:

l_{ij} e u_{ij}: são os limites inferior e superior, respectivamente, do fluxo em (i, j),

s_i: é a demanda no nó i.

A restrição (2.2) é chamada restrição de conservação de fluxo e a restrição (2.3) é chamada restrição de capacidade. Em algumas ocasiões, o problema de FCM pode aparecer com algumas variações, onde o limite superior é infinito e o limite inferior é zero para todos os arcos. Nesse caso é dito que o problema é não capacitado.

Na próxima seção mostraremos algumas aplicações para o problema de fluxo de custo mínimo.

2.2 Aplicações

O problema de fluxo de custo mínimo tem várias aplicações práticas. Muitos problemas reais podem ser traduzidos em um problema de FCM e resolvidos por um algoritmo padrão. O exemplo mais usual para problema de FCM é o problema de transporte. A seguir, mostraremos esse e outros problemas de maneira sucinta.

2.2.1 Problema de transporte

Imagine uma situação em que exitem cidades fornecedoras, que produzem certa quantidade de produto, e outras cidade consumidoras que consomem certa quantidade do mesmo produto. Considere os caminhos que ligam cada um dos fornecedores a cada um dos consumidores. Considere também que existe um custo por unidade transportada associado a cada caminho entre duas cidades, ou seja, por cada unidade transportada da cidade i até a cidade j, incorreremos no custo de c_{ij} unidades monetárias.

O problema, então, é transportar o produto dos locais de produção até os locais de consumo com o menor custo total possível, respeitando os limites de produção e de consumo de cada localidade. Se tomarmos as localidade como nós e os caminhos entre as localidades como arcos, podemos então modelar o problema de transporte como um problema de FCM como mostrado a seguir.

Para m localidades de produção e n localidades de consumo, deve-se:

$$minimizar \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.4}$$

sujeito às restrições

$$\sum_{j:(i,j)\in A} x_{ij} = \alpha_i, \qquad \forall i = 1, ..., m, \tag{2.5}$$

$$\sum_{i:(i,j)\in A} x_{ij} = \beta_j, \qquad \forall j = 1, ..., n, \tag{2.6}$$

$$0 \leq x_{ij} \leq min\{\alpha_i, \beta_j\}, \qquad \forall (i,j) \in A, \tag{2.7}$$

onde α_i e β_j são escalares positivos que precisam satisfazer

$$\sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j.$$

2.2.2 Problema do caminho mínimo

Considere que cada arco (i, j) de um grafo direcionado está associado a um custo c_{ij}. Considere que o custo de um caminho direcionado é dado pela soma dos custos dos arcos pertencentes ao caminho. Para um dado par de nós s e t, o problema do caminho mínimo é encontrar o caminho direcionado que vai de s a t que tenha custo mínimo.

Se considerarmos os arcos como ruas e custo do arco como o comprimento da rua, podemos utilizar o problema do caminho mínimo para encontrar a menor rota, em termos de distância, entre dois pontos dentro de uma cidade ou em um sistema rodoviário. Esse problema é amplamente utilizado em aplicativos de mapa digital, disponíveis em *smartfones* ou em computadores de bordo de carros, que possibilitam a busca pela menor rota entre dois pontos do mapa.

O problema do caminho mínimo pode ser modelado como um problema de fluxo de custo mínimo da seguinte forma:

$$minimizar \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.8}$$

sujeito às restrições

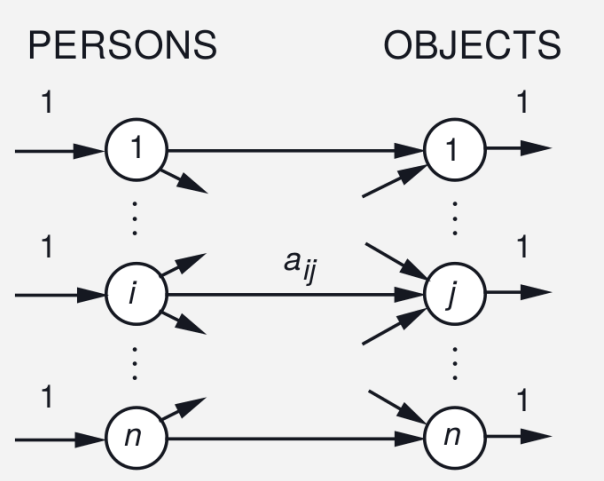
$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(i,j)\in A} x_{ji} = \begin{cases} 1, & \text{se } i = s, \\ -1, & \text{se } i = t, \\ 0 & \text{para outros casos,} \end{cases} \tag{2.9}$$

$$0 \leq x_{ij}, \qquad \forall (i,j) \in A.$$

A restrição (2.9) garante que apenas uma unidade de fluxo sairá de s e chegará a t. Nesse caso, o valor da variável x_{ij} irá sempre satisfazer a equação

$$x_{ij} = \begin{cases} 1, & \text{se } (i,j) \text{ faz parte da solução ,} \\ 0 & \text{caso contrário .} \end{cases}$$

Figura 1 – Representação gráfica do problema de atribuição



Fonte: Bertsekas (1998)

Se a solução x_P, associada ao caminho P, encontrada for mínima para a equação (2.8), significa que o caminho P é mínimo para o problema de caminho mínimo. Isso pode ser provado por refutação, pois se existir um caminho P' tal que o custo de P' seja menor que o custo de P significa que existe uma sequência de arcos direcionados que ligam s e t que é uma solução $x_{P'}$ para a equação (2.8) cujo valor é menor que x_P, o que refuta a premissa de x_P ser solução mínima do problema.

2.2.3 Problema de atribuição

Suponha que existe n pessoas e n objetos que queremos associar às pessoas em uma base de um-pra-um. Considere que há um benefício a_{ij} em associar a pessoa i ao objeto j, e queremos associar as pessoas aos objetos de forma a maximizar o benefício total. Considere, ainda, um conjunto A de pares, de modo que a pessoa i só pode ser associada ao objeto j se $(i, j) \in A$. O problema é encontrar o conjunto de pares $(1, j_1), \ldots, (n, j_n) \in A$, em que os objetos $j_1, \ldots, j_n$ sejam todos distintos, de forma a maximizar $\sum_{i=1}^{n} a_{ij_i}$.

Seja o conjunto de variáveis x_{ij}, com $(i, j) \in A$, em que $x_{ij} = 1$ se a pessoa i for associada ao objeto j e $x_{ij} = 0$ caso contrário. O problema de atribuição pode ser definido, em termo de PL, da seguinte forma:

$$maximizar \sum_{(i,j)\in A} a_{ij}x_{ij}$$

sujeito às restrições

$$\sum_{j:(i,j)\in A} x_{ij} = 1, \qquad \forall i = 1, \ldots, n, \tag{2.10}$$

$$\sum_{i:(i,j)\in A} x_{ij} = 1, \qquad \forall j = 1, \ldots, n, \tag{2.11}$$

$$0 \leq x_{ij} \leq 1, \qquad \forall (i,j) \in A.$$

O problema de atribuição pode ser facilmente visto como um problema de fluxo de custo mínimo se considerarmos as pessoas e os objetos como nós e os pares de associações possíveis (i, j) como arcos que ligam os nós. As restrições (2.10) e (2.11) representam a restrição de conservação de fluxo no problema de FCM, em que cada nó i tem demanda $s_i = 1$ (restrição (2.10)) e cada nó j tem demanda $s_j = -1$ (restrição (2.11)). Para transformar o problema de atribuição em um problema de minimização, o benefício a_{ij} deve ser transformado no custo $c_{ij} = -a_{ij}$.

Um exemplo de aplicação para esse problema é para atribuir atividades aos funcionários ou tarefas às máquinas. Também pode ser usado para associar doadores a pacientes com a maior probabilidade de aceitação do órgão, entre outras aplicações.

3 SIMPLEX DE REDE

Para Ahuja, Magnanti e Orlin (1993), o método simplex para resolver problemas de programação linear é provavelmente o mais poderoso algoritmo já desenvolvido para resolver problemas de otimização restritos.

Os problemas de FCM constituem uma classe especial de problemas de programação linear. É plausível considerar o simplex uma boa estratégia de solução para esses problemas. No entanto, problemas de fluxo em rede, dos quais os problemas de FCM fazem parte, apresentam uma estrutura especial. Essa estrutura especial torna o método simplex convencional não competitivo quando comparado com outros métodos combinatórios que exploram essa estrutura de rede. O simplex de rede é um algoritmo que interpreta as iterações do simplex convencional como operações em uma estrutura de rede. Com isso temos um algoritmo competitivo para resolver problemas de fluxo de custo mínimo.

3.1 Estrutura de base (T,L,U)

Para definirmos a estrutura de base (T, L, U) precisamos primeiro introduzir os conceitos de solução livre de ciclo e solução árvore geradora. Vamos considerar que $l_{ij} = 0$ para todo arco (i, j). Todo problema de FCM pode se preprocessado e colocado neste formato sem perda de generalidade. Essa consideração, portanto, não irá limitar o algoritmo e trará algumas facilidades de implementação.

Para uma solução x qualquer, dizemos que um arco (i, j) é livre se $0 < x_{ij} < u_{ij}$. Do contrário, se $x_{ij} = 0$ ou $x_{ij} = u_{ij}$ dizemos que o arco é restringido. Em um arco livre é possível aumentar ou diminuir o fluxo sem desrespeitar os limites. Já em um arco restringido, se ele estiver em seu limite inferior só é possível aumentar o fluxo, se ele estiver em seu limite superior só é possível diminuir o fluxo. Uma solução x é uma solução livre de ciclo se a rede não contem ciclos compostos apenas de arcos livres. Denotamos por solução árvore geradora uma solução viável x e uma árvore geradora da rede associada a ela em que todo arco não pertencente à árvore é um arco restringido.

Problemas de fluxo de custo mínimo sempre têm uma solução ótima que é árvore geradora e é livre de ciclos (AHUJA; MAGNANTI; ORLIN, 1993).O simplex de rede se aproveita desse resultado para explorar apenas soluções árvore geradora. Essa estratégia é similar à estratégia de busca apenas por soluções nos vértices no simplex convencional.

Para uma solução árvore geradora qualquer, uma estrutura de árvore geradora (T, L, U) relacionada à ela pode ser obtida pela partição do conjunto de arcos A tal que: T é o conjunto de arcos pertencentes à árvore geradora; L é o conjunto de arcos fora da árvore com valor do fluxo igual a zero; U é o conjunto de arcos fora da árvore com valor do fluxo igual ao limite superior de fluxo no arco.

Também é possível obter uma solução árvore geradora única a partir de uma estrutura de

árvore geradora. Para uma estrutura de árvore geradora (T, L, U), atribuímos $x_{ij} = 0$ para todo arco $(i, j) \in L$, $x_{ij} = u_{ij}$ para todo arco $(i, j) \in U$, e resolvemos a equação de balanceamento de massa para determinar o valor do fluxo nos arcos pertencentes a T. Uma estrutura de árvore geradora (EAG) é viável se a solução árvore geradora associada satisfaz todos os limites de fluxo dos arcos. Se todos os arcos pertencentes à árvore geradora forem arcos livres, a árvore geradora é não-degenerada; caso contrário, a árvore geradora é degenerada. Dizemos que uma EAG é ótima se a solução árvore geradora associada é uma solução ótima para o problema de fluxo de custo mínimo.

3.2 Algoritmo simplex de rede primal

O algoritmo simplex de rede primal sempre mantém uma estrutura de árvore geradora viável. A cada interação, é inserido um arco em T e retirado um arco pertencente a T de forma a se obter uma nova estrutura de árvore geradora também viável. Uma EAG corresponde a uma solução básica viável em programação linear e cada iteração de transição de uma EAG para outra corresponde a uma operação de pivoteamento no simplex convencional. O primeiro passo do algoritmo simplex de rede é encontrar uma EAG viável qualquer (um método para encontrar uma EAG inicial viável será mostrado na Seção 3.10). A Figura 2 mostra os principais passos do algoritmo simplex de rede primal.

Figura 2 – Algoritmo simplex de rede

algoritmo simplex de rede;
início
    determinar uma estrutura (T, L, U) inicial viável;
    seja x o fluxo e π o potencial dos nós;
    enquanto existir algum arco inviável fora da árvore faça
        escolher um arco (k, l) inviável para entrar na base;
        adicionar o arco (k, l) na árvore e determinar um arco (p, q) para sair da árvore;
        atualizar a árvore e a solução x e π;
    fim
fim

O potencial $\pi = (\pi(1), \ldots, \pi(n))$ relacionado a cada nó está associado à solução dual do problema. Na próxima seção veremos a justificativa para seu uso e uma maneira de calculá-lo. Todos os passos do algoritmo simplex de rede serão detalhados nas próximas seções.

3.3 Potencial dos nós e custo reduzido

O potencial $\pi(i)$ associado a um nó i pode ser entendido como o custo potencial de transportar uma unidade de fluxo através da árvore T a partir da raiz até o nó i. Os potenciais dos nós representam o vetor solução para o problema dual de FCM. A partir dos potenciais dos nós, podemos definir o custo reduzido de um arco da seguinte forma:

$$c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j).$$

O custo reduzido de um arco representa a redução que será obtida na função objetivo se o arco entrar na base. Para os arcos pertencentes a base o custo reduzido é sempre zero. Ahuja, Magnanti e Orlin (1993) mostra algumas propriedades relacionadas ao custo reduzido, bem como alguns resultados obtidos com sua utilização. Utilizamos o custo reduzido para verificar o estado de viabilidade do arco. Na Seção 3.5 mostraremos as condições de otimalidade de uma solução que se baseia no valor do custo reduzido dos arcos.

3.4 Cálculo do Fluxo e do Potencial dos Nós

É possível calcular o fluxo e o potencial dos nós utilizando os índices predecessor, profundidade e *thread* da árvore geradora.

- Predecessor de um nó i é o nó imediatamente anterior a ele no caminho da raiz até o nó i. Existe um único caminho ligando qualquer nó de uma árvore até a raiz (NETTO, 2006), portanto o predecessor de um nó também é único. Por convenção, considera-se que o predecessor do nó raiz é sempre zero.

- Profundidade de um nó é o número de nós no caminho deste nó até a raiz.

- *Thread* é o passeio pela árvore iniciando na raiz, visitando os nós em pré-ordem e depois retornando para a raiz.

Os três índices da árvore citados acima podem ser obtidos realizando-se uma única busca em profundidade. Antes de visitar os nós filhos, computam-se os índices de predecessor e profundidade para cada um deles. Para cada nó filho, o predecessor será o nó pai e a profundidade será a do nó pai acrescida de 1 (a profundidade do nó raiz é 0). O índice thread será a sequencia em que os nós forem visitados.

3.4.1 Cálculo do Fluxo

Para uma EAG (T, L, U), o vetor de fluxos x pode ser obtido seguindo os seguintes passos: (1) para todo arco $(i, j) \in L$, atribuir $x_{ij} = 0$; (2) para todo arco $(i, j) \in U$, atribuir $x_{ij} = u_{ij}$, aumentar $b(i)$ e diminuir $b(j)$ em u_{ij} unidades; (3) escolher um nó folha j pertencente a T e obter o nó $i = predecessor(j)$; (4) se o arco (i, j) pertencer a T atribuir $x_{ij} = -b(j)$, se o arco (j, i) pertencer a T atribuir $x_{ij} = b(j)$ (5) aumentar $b(i)$ em $b(j)$ unidades e excluir j e o arco (i, j) de T; (6) repetir (3), (4), (5) até não restar nenhum arco em T.

A Figura 3 mostra um algoritmo eficiente para calcular o fluxo a partir da estrutura de árvore geradora utilizando os índices da árvore.

Figura 3 – Algoritmo para calcular o fluxo.

algoritmo cálculo do fluxo;
início
  $b'(i) := b(i)$, para todo $i \in$ **N**;
  para cada $(i, j) \in$ **L faça**
    $x_{ij} := 0$;
  fim
  T':=T;
  enquanto T' $\neq \{1\}$ **faça**
    escolher um nó folha j (que não seja o nó 1) na subárvore **T'**;
    $i := pred(j)$;
    se $(i, j) \in$ **T' então**
      $x_{ij} := -b'(j)$;
    senão
      $x_{ij} := b'(j)$;
    fim
    $b'(i) := b'(i) + b'(j)$;
    remover o nó j e o arco (i, j) da subárvore **T'**;
  fim
fim

3.4.2 Cálculo do Potencial dos Nós

O simplex sempre mantém a condição de que para todo arco $(i, j) \in T$ o custo reduzido c^{π}_{ij} é zero (AHUJA; MAGNANTI; ORLIN, 1993), ou seja,

$$c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j) = 0 \text{ para todo arco} (i, j) \in T. \tag{3.1}$$

É possível adicionar uma constante k ao potencial todos os nós sem alterar o custo reduzido dos arcos, $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j) = c_{ij} - [\pi(i) + k] + [\pi(j) + k]$. Com isso podemos atribuir arbitrariamente o valor do potencial de um dos nós da árvore e então calcular o valor do potencial dos nós restantes. A ideia básica para o cálculo do potencial dos nós é atribuir zero ao potencial do nó 1, $\pi(1) = 0$, e calcular o potencial dos nós adjacentes de forma recursiva utilizando a equação 3.1. Uma forma simples de calcular o potencial de todos os nós é percorrendo a árvore na ondem dada pelo índice $thread$. Dessa forma, garantimos que ao tentar calcular o potencial de um nó, o potencial de seu predecessor já terá sido calculado. A Figura 4 mostra o algoritmo para o cálculo do potencial dos nós.

Figura 4 – Algoritmo para calcular o potencial dos nós

algoritmo cálculo do potencial;
início
  $\pi(1) := 0$;
  $j := thread(1)$;
  enquanto $j \neq 1$ **faça**
    $i := pred(j)$;
    se $(i, j) \in A$ **então** $\pi(j) := \pi(i) - c_{ij}$;
    se $(j, i) \in A$ **então** $\pi(j) := \pi(i) + c_{ji}$;
    $j := thread(j)$;
  fim
fim

3.5 Condições de otimalidade

Seja (T, L, U) uma EAG viável do problema de fluxo de custo mínimo, é dito que um arco (i, j) satisfaz as condições de otimalidade se

$$(i, j) \in L \ \ \textbf{e} \ \ c_{ij}^{\pi} \geq 0, \tag{3.2}$$

ou

$$(i, j) \in U \ \ \textbf{e} \ \ c_{ij}^{\pi} \leq 0. \tag{3.3}$$

Uma estrutura de árvore geradora é ótima se todos os arcos em L e em U satisfazem as condições de otimalidade. Quando tal estrutura é encontrada o algoritmo termina com solução ótima. Se algum arco não satisfaz as condições de otimalidade ele é elegível para entrar na base.

3.6 Regra de escolha dos arcos de E e S

Qualquer arco elegível pode ser escolhido para entrar na base sem comprometer a finitude do algoritmo simplex. No entanto, diferentes regras de escolha do arco provocam diferentes comportamentos na maneira como o algoritmo encontra o ótimo e consequentemente no tempo de execução do algoritmo.

- Regra de Dantzig: A regra sugere que a cada iteração deve-se escolher o arco com a maior violação da condição de otimalidade. O arco com a maior violação também é o arco que causa a maior redução no valor da função objetivo. O resultado da regra de Dantzig é uma quantidade menor de iterações se comparado a outras regras. O motivo é que a cada iteração ela realiza a maior redução possível no valor da função objetivo. No entanto, apesar de realizar menos iterações, a cada iteração o algoritmo avalia todos os

arcos fora da árvore para identificar o arco com a maior violação, o que representa um grande consumo de tempo no resultado final.

- Regra do primeiro arco elegível: Nessa regra, o primeiro arco elegível encontrado será o arco escolhido para entrar na base. Cada iteração geralmente realiza poucos passos, já que ao encontrar um arco elegível ele já é escolhido, sem a necessidade de avaliar todos os arcos fora da árvore. Todavia, a regra tem uma deficiência: como a escolha do arco não leva em consideração a capacidade do arco de reduzir o valor da função objetivo, na maioria dos casos o algoritmo irá realizar mais iterações que a regra de Dantzig até atingir o ótimo.

- Regra da lista de candidatos: A regra da lista de candidatos tenta unir as vantagens das regras de Dantzig e do primeiro arco elegível em uma única regra. O método utiliza uma técnica de duas fases: uma de maior esforço e outra de menor esforço. Na fase de maior esforço é criada uma lista de arcos elegíveis. Na fase de menor esforço é escolhido o arco da lista com maior violação da otimalidade.

  A lista deve ter um tamanho máximo definido. Na fase de maior esforço, o algoritmo inicia avaliando os arcos que saem do nó 1, depois os do nó 2 e assim sucessivamente, adicionando os arcos elegíveis à lista até atingir o tamanho máximo da lista ou até ter avaliado todos os nós. A próxima fase de maior esforço deve iniciar do nó em que a última parou.

  Na fase de menor esforço, o algoritmo deve avaliar os arcos da lista para identificar o arco com maior violação da otimalidade. Na medida em que os arcos são avaliados, o algoritmo deve remover da lista os arcos que já não são mais elegíveis. Quando não houver mais arcos na lista ou quando o número máximo de execuções da fase de menor esforço for atingido uma nova lista deve ser preenchida executando-se uma fase de maior esforço.

A regra da lista de candidatos é a que apresenta os melhores resultados práticos dentre as três regras mostradas (AHUJA; MAGNANTI; ORLIN, 1993). Essa regra é bastante flexível e alterando-se os valores de tamanho da lista e numero máximo de execuções da fase de menor esforço pode-se atingir diferentes resultados. De fato, se definirmos o tamanho da lista e o número de execuções da fase de menor esforço para 1, podemos notar que a regra da lista de candidatos terá o mesmo resultado da regra do primeiro arco elegível. De forma similar pode-se atingir o resultado da regra de Dantzig.

3.6.1 Escolha do arco de saída

Tendo-se escolhido o arco de entrada na árvore, deve-se então escolher um arco para sair da árvore. Se o arco de entrada for inserido na árvore, será criado um único ciclo W com os

arcos da árvore. Os arcos nesse ciclo serão os candidatos a sair da árvore. O sentido do ciclo é definido pelo arco de entrada. Se o arco de entrada pertencer a L o sentido do ciclo será o mesmo do arco. Se o arco pertence a U o ciclo terá sentido contrário ao do arco.

Os arcos pertencentes a W podem ser divididos em dois subconjuntos: (1) o conjunto $\overline{W}$ de arcos com o mesmo sentido de W; (2) o conjunto $\underline{W}$ de arcos com sentido contrário ao de W. O sentido o ciclo determina o sentido em que uma quantidade Δ de fluxo será aumentada. Aumentar o fluxo em W significa aumentar o fluxo dos arcos em $\overline{W}$ e diminuir o fluxo nos arcos em $\underline{W}$. Como uma operação de pivoteamento deve manter a viabilidade da solução, cada arco (k, l) só pode aumentar ou diminuir uma certa quantidade δ_{kl} sem que seus limites inferior e superior de fluxo sejam ultrapassados. O variação máxima do fluxo, δ_{kl}, em cada arco será de:

$$\delta_{kl} = u_{kl} - x_{kl} \text{ se } (k, l) \in \overline{W}; \tag{3.4}$$

$$\delta_{kl} = x_{kl} \text{ se } (k, l) \in \underline{W}. \tag{3.5}$$

O arco escolhido para sair da base será aquele que limitar mais o aumento do fluxo, ou seja, o arco com o menor δ_{kl}. O valor a ser aumentado no ciclo será dado por $\Delta = min\{\delta_{kl} : (k, l) \in W\}$. O valor da função objetivo sofrerá uma redução de $\Delta|c^{\pi}_{ij}|$ unidades. Se mais de um arco é elegível para sair da base (mais de um arco com o valor de δ mínimo), a árvore geradora resultante é sempre degenerada e a iteração é degenerada.

Figura 5 – Algoritmo para identificar o ciclo com o arco de entrada.

algoritmo identificação do ciclo;
início
  $i := k$ e $j := l$;
  se $q \in T_2$ **então** $y := q$ **senão** $y := p$;
  se $k \in T_1$ **então** $alteracao := -c^{\pi}_{kl}$ **senão** $alteracao := c^{\pi}_{kl}$;
  enquanto $i \neq j$ **faça**
    se $prof(i) > prof(j)$ **então** $i := pred(i)$;
    senão se $prof(j) > prof(i)$ **então** $j := pred(j)$;
    senão $i := pred(i)$ e $j := pred(j)$;
  fim
fim

No Apêndice A mostramos uma iteração do algoritmo simplex de rede que exemplipifica uma operação de pivoteamento.

3.7 Atualização da árvore

Ao realizar um pivoteamento, se o arco que sai for diferente do arco que entra, o potencial em alguns nós são alterados e é necessário atualizar o seu valor. Os potenciais nos nós podem

ser recalculados utilizando o algoritmo mostrado na Figura 4, no entanto, como apenas alguns nós tem o potencial alterado, existem maneiras mais eficientes de se obter os índices atualizados.

Seja (p, q) o arco escolhido para sair da árvore e (k, l) o arco escolhido para entrar na árvore. Se o arco (p, q) for diferente de (k, l), então o simplex deve retirar (p, q) da árvore e inserir (k, l). A remoção de (p, q) irá formar duas subárvores: T_1 contendo a raiz e T_2 não contendo a raiz. O potencial dos nós em T_1 não será alterado, já o potencial dos nós em T_2 será alterado em uma certa quantidade. É fácil mostrar esse resultado utilizando as condições $\pi(1) = 0$ e $c_{ij} - \pi(i) + \pi(j) = 0$ para calcular o potencial em todos os nós. Se $k \in T_1$ e $l \in T_2$ então os nós em T_2 sofrerão uma alteração de $-c^{\pi}_{kl}$ unidades. Se $k \in T_2$ e $l \in T_1$ então os nós em T_2 sofrerão uma alteração de c^{π}_{kl} unidades.

Figura 6 – Algoritmo para atualizar o potencial dos nós

algoritmo atualização do potencial;
início
    (seja (p, q) o arco que sai da árvore e (k, l) o arco que entra na árvore);
    se $q \in T_2$ **então** $y := q$ **senão** $y := p$;
    se $k \in T_1$ **então** $alteracao := -c^{\pi}_{kl}$ **senão** $alteracao := c^{\pi}_{kl}$;
    $\pi(y) := \pi(y) + alteracao$;
    $z := thread(y)$;
    enquanto $prof(z) > prof(y)$ **faça**
        $\pi(z) := \pi(z) + alteracao$;
        $z := thread(z)$;
    fim
fim

O Figura 6 mostra a rotina para atualizar rapidamente o potencial dos nós que sofreram alteração.

Da mesma forma, a atualização do fluxo nos arcos pode ser realizada mais rapidamente se a necessidade de recalculá-lo. Como só ocorre alteração de fluxo ao longo do circuito formado pelo arco de entrada, é suficiente atualizar apenas o fluxo dos arcos pertencentes ao circuito. A Figura 5 mostra uma rotina para a identificação do ciclo.

Após a atualização do potencial e do fluxo, é necessário atualizar os índices da árvore. Ahuja, Magnanti e Orlin (1993) diz que é possível calcular os índices em tempo inferior a $O(n)$ e dá referências de trabalhos que mostram como realizar a atualização dos índices em suas notas de referências.

3.8 Finitude do método simplex

Se realizarmos apenas iterações não degeneradas é fácil notar a finitude do simplex. O algoritmo sempre passa de uma EAG viável para outra. Existe um número finito de EAG em uma rede e cada EAG é associada a um custo. Se a cada iteração o algoritmo encontra uma EAG com o custo não superior ao anterior, o algoritmo encontrará a EAG ótima e terminará em

um número finito de passos.

No entanto, iterações degeneradas pode ocorrer com frequência. O algoritmo pode ficar preso em um ciclo de iterações degeneradas e não chegar ao fim. Mas a ciclagem pode ser evitada se tomarmos mais cuidado ao escolher um arco de saída em uma iteração degenerada. Uma forma simples de evitar ciclagem por degeneração e garantir a finitude do simplex é manter uma EAG fortemente viável.

3.9 EAG fortemente viável

Seja (T, L, U) uma estrutura de árvore geradora. Sendo T uma árvore geradora, os arcos em T podem apontar para a raiz ou apontar para as folhas.

Definição 3.1 (Árvore geradora fortemente viável) *T é árvore geradora fortemente viável se todos os arcos com fluxo em zero apontam para a raiz e todos os arcos com fluxo igual a capacidade do arco apontam para as folhas.*

O resultado dessa definição é que é sempre possível aumentar uma quantidade positiva de fluxo no caminho que liga qualquer nó à raiz sem ultrapassar os limites de fluxo dos arcos no caminho. Se T é fortemente viável, dizemos que a EAG (T, L, U) também é fortemente viável.

Um arvore geradora não degenerada (sem arcos com fluxo igual ao limite inferior ou superior) é sempre fortemente viável. Uma árvore degenerada pode ou não ser fortemente viável. Portanto, em uma iteração degenerada, se o arco de saída for escolhido arbitrariamente, pode-se ou não encontrar uma árvore geradora fortemente viável. A seguir veremos uma uma regra de escolha do arco de saída que sempre mantém uma árvore geradora fortemente viável.

Seja (i, j) o arco escolhido para entrar na base T. Se (i, j) for inserido em T um único circuito w será formado. Se $(i, j) \in L$ o circuito terá o mesmo sentido do arco, se $(i, j) \in U$ o circuito terá sentido oposto ao do arco. E seja um nó k o ápice do circuito, ou seja, o nó mais próximo à raiz dentre todos os nós do circuito.

Regra 3.1 (Escolha do arco de saída) *Se a iteração é degenerada, escolha o último arco limitante do fluxo no sentido do circuito w e iniciando no nó k.*

O simplex deve sempre manter uma EAG fortemente viável ao realizar um pivoteamento. Para tal, o algoritmo deve iniciar com uma EAG fortemente viável. Na Seção 3.10 será mostrado um método de se encontrar uma base inicial viável que é sempre EAG fortemente viável.

3.10 Primeira fase do método simplex de rede

O método simplex requer uma base inicial viável para iniciar a busca pela solução ótima do problema. No entanto, raramente se dispõe de uma base inicial viável. Várias técnicas para se chegar a uma base inicial viável para o simplex convencional foram desenvolvidas ao longo

dos anos. A maioria delas utiliza variáveis artificiais. A etapa de busca pela base inicial viável é chamada "primeira fase" do método simplex. Devido a isso, o método é conhecido também como "simplex de duas fases".

No simplex de rede, a EAG corresponde à base no simplex convencional. Algumas das técnicas do simplex convencional para encontrar uma base viável foram adaptadas para encontrar uma EAG viável no simplex de rede. Bonates (2001) mostra duas técnicas conhecidas para se obter uma base para o simplex primal. Ahuja, Magnanti e Orlin (1993) apresenta uma adaptação dessas técnicas para obter uma EAG inicial para o simplex de rede primal. Mostraremos a ideia principal dessa técnica a seguir.

Para obtermos a EAG (T, L, U) inicial viável deve-se, primeiro, inserir alguns arcos artificiais ao problema original da seguinte forma: para todo nó j, que não seja o nó 1, se $b(j) \geq 0$, será criado o arco artificial $(j, 1)$ e inserido em T, se $b(j) < 0$, o arco artificial $(1, j)$ deve ser criado e inserido em T. Os arcos artificiais são arcos que não pertencem ao problema original, e devem ser eliminados da base durante a execução da primeira fase para se obter a EAG inicial. Para tal, devem ser criados com custo e capacidade suficientemente altos para serem eliminados de T naturalmente.

Todos os arcos do problema original devem ir para L, e U permanece vazio. Desse modo tem-se uma EAG artificial viável. Para se obter a EAG inicial viável para o problema original, deve-se executar o simplex com a EAG artificial até que não reste nenhum arco artificial na EAG. Se o algoritmo encontrar uma solução ótima antes de eliminar todos os arcos artificiais, então o problema não tem solução.

A primeira fase do simplex através da inserção de arcos artificiais pode se muito custosa. Como para cada nó é inserido um arco artificial que deve ser eliminado da base, para se obter a EAG inicial são necessárias pelo menos $|N|$ iterações. Este fato foi a principal justificativa pela busca por um método que não necessite de uma base inicial viável, podendo caminhar por bases inviáveis, e ainda chegar ao ótimo. Os algoritmos da família criss-cross fazem parte desse tipo de algoritmo e são o objeto de estudo do próximo capítulo.

4 ALGORITMO CRISS-CROSS DE REDE

O algoritmo criss-cross para problemas de PL surgiu como uma tentativa de se evitar a primeira fase do método simplex. A primeira fase do método simplex pode ser muito custosa. Sua execução leva no mínimo n iterações para chegar a uma base inicial viável. A principal vantagem do criss-cross sobre os algoritmos da família simplex é sua capacidade de iniciar a busca pela solução ótima sem a necessidade de uma base viável.

O algoritmo criss-cross para resolver programas lineares apresenta desempenho similar aos algoritmos da família simplex (BONATES; MACULAN, 2003). Fukuda e Terlaky (1997) apresentam uma versão finita do algoritmo criss-cross para problemas de PL e mostram algumas regras de pivoteamento de natureza combinatória, com destaque para a regra de menor índice.

Neste capítulo introduzimos uma versão do criss-cross para rede baseado no algoritmo finito abordado por Fukuda e Terlaky (1997). A versão aqui mostrada utiliza a regra do menor índice adaptada para problemas de rede.

4.1 Criss-cross finito

Na inicialização do criss-cross é necessário definir uma estrutura (T, L, U) qualquer, viável ou inviável. Para tal, basta encontrar uma arvore geradora T qualquer da rede; os demais arcos ficam em L e U permanece vazio. Note que, apesar de o criss-cross necessitar de uma EAG, essa não precisa ser viável, e encontrar uma árvore geradora é muito menos custoso que encontrar uma EAG viável. A Figura 7 mostra o algoritmo criss-cross com regra do menor índice.

O fluxo e o potencial dos nós para o algoritmo criss-cross podem ser obtidos utilizando a mesma rotina apresentada na Seção 3.4. Da mesma forma, é possível atualizar o fluxo e o potencial de maneira mais eficiente seguindo a rotina mostrada na Seção 3.7.

A solução inicial do criss-cross pode ser primal e dual inviável. Isso significa que existem arcos inviáveis (não satisfazem as condições de otimalidade) dentro e fora da base. Enquanto existirem arcos que não satisfazem as condições de otimalidade, devem ser escolhidos dois arcos, um pertencente a T e outro fora de T, para se realizar o pivoteamento.

A escolha dos arcos para entrar e sair de T segue a regra de pivoteamento de menor índice. Se uma solução for encontrada em que todos os arcos satisfazem as condições de otimalidade, então essa solução é ótima e o algoritmo encerra.

É possível que, no decorrer da busca pela solução ótima, eventualmente se encontre uma solução que é primal ou dual viável. Bonates (2001) propõe a utilização do simplex para continuar a busca no momento em que tal cenário for alcançado e relata que obteve bons resultados. Nesse caso o criss-cross funciona como a primeira fase do método simplex para encontrar uma base viável, mas pode encontrar uma base mais próxima da solução ótima do problema. No futuro, utilizaremos o termo CC fino para referenciarmos o criss-cross funcionando como primeira fase do simplex.

Figura 7 – Algoritmo criss-cross de rede

Criss-cross – regra do menor índice
início
seja (T, L, U) uma solução inicial qualquer;
seja x o fluxo e π o potencial dos nós;
enquanto verdadeiro faça
se não existe arcos inviáveis então
pare: solução ótima encontrada;
senão
seja (i, j) o arco inviável de menor índice;
se $(i, j) \in T$ **então**
encontre o arco $(p, q) \notin T$ de menor índice que forma um ciclo que contem o arco (i, j);
se $\nexists (p, q)$ **então**
pare: o problema é primal inviável;
fim
senão
encontre o arco (p, q) de menor índice pertencente ao ciclo C formado pelo arco (i, j);
se $\nexists (p, q)$ **então**
pare: o problema é dual inviável;
fim
fim
realize o pivoteamento entre (i, j) e (p, q);
atualize a estrutura (T, L, U), a solução x e π;
fim
fim
fim

4.2 Condições de otimalidade

Seja (T, L, U) uma EAG viável do problema de fluxo de custo mínimo, é dito que um arco (i, j) satisfaz as condições de otimalidade se

$$(i, j) \in T \text{ e } l_{ij} \leq x_{ij} \leq u_{ij}, \tag{4.1}$$

ou

$$(i, j) \in L \text{ e } c^{\pi}_{ij} \geq 0, \tag{4.2}$$

ou

$$(i, j) \in U \text{ e } c^{\pi}_{ij} \leq 0. \tag{4.3}$$

Se um arco não satisfaz as condições de otimalidade, então o arco é inviável e é elegível para realizar pivoteamento. Se todos os arcos básicos (arcos em T) forem viáveis, ou seja, satisfazem a equação (4.1), significa que a solução é primal viável. Da mesma forma, se todos os arcos fora da base forem viáveis, ou seja, os arcos em L satisfazem 4.2 e os arcos em U satisfazem 4.3,

significa que a solução é dual viável. Uma solução para o problema de fluxo de custo mínimo é ótima quando ela é primal viável e dual viável (AHUJA; MAGNANTI; ORLIN, 1993).

4.3 Regra de pivoteamento do menor índice

O primeiro passo do criss-cross é escolher um arco inviável (i, j). O método criss-cross realiza dois tipos de iteração: primal e dual. Se o arco escolhido pertencer a T a iteração é primal, caso contrário, o arco pertence a L ou U e a iteração é dual. Dizemos então que o arco (i, j) escolhido no primeiro passo é o arco ativo, pois ele determina o tipo da iteração.

Em uma iteração primal, o arco (i, j) pertencente a T é escolhido para sair de T. Dessa forma, um arco (p, q) não pertencente a T deve ser escolhido para entrar em T. Em uma iteração dual, o arco (i, j) não pertencente a T é escolhido para entrar em T. Um arco (p, q) pertencente a T deve então ser escolhido para sair de T. O arco (p, q) é o arco passivo pois sua escolha é dependente do arco ativo e do tipo de iteração.

O arco escolhido para entrar na árvore T, passiva ou ativamente, forma um único ciclo C com os arcos de T. Uma quantidade positiva de fluxo deve, então, ser aumentada no sentido de C. Se o arco entrar ativamente na árvore, e seja (i, j) esse arco, o sentido de C será mesmo do arco (i, j) se $(i, j) \in L$ e será oposto ao sentido de (i, j) se $(i, j) \in U$. Caso contrário, o arco entra passivamente na árvore. Seja (i, j) o arco que sai ativamente de T, se $x_{ij} < l_{ij}$ o sentido do ciclo C será o mesmo do arco (i, j), se $x_{ij} > u_{ij}$, o sentido de C será oposto ao do arco (i, j). Todos os arcos que tenham o mesmo sentido do C são chamados arcos *forward*. Os arco com sentido oposto ao do ciclo C são chamados arcos *backward*.

Pela regra de menor índice, o arco ativo deve ser escolhido da seguinte forma:

Escolha do arco ativo. Escolha o arco (i, j) inviável pertencente a T, L ou U que tenha menor índice.

Uma vez que o arco ativo e o tipo da iteração foram determinados, será escolhido então um arco passivo.

Arco que sai passivamente. O arco escolhido para sair passivamente de T será o arco $(p, q) \in C$ que tenha menor índice.

Arco que entra passivamente. O arco escolhido para entrar passivamente em T será o arco $(p, q) \notin T$ com menor índice que forme um ciclo C, tal que $(i, j) \in C$, e (p, q) satisfaça:

1. $(p, q) \in L$ e é forward ou
2. $(p, q) \in U$ e é backward.

Uma maneira eficiente de se obter o arco ativo é percorrer os arcos em ordem crescente de índice e realizar o teste de otimalidade, o primeiro arco inviável encontrado deve ser escolhido

para o pivoteamento. Esta regra é semelhante a regra do último arco elegível mostrada na Seção 3.6, no entanto, como o criss-cross não tem base viável, o teste de otimalidade é feito tanto nos arcos pertencentes a T como nos arcos fora de T. O arco de saída passiva também pode ser facilmente obtido utilizando o algoritmo da Figura 5 para encontrar o ciclo C. Uma vez identificados os arcos pertencentes a C, basta escolher o de menor índice. O algoritmo para escolha do arco de entrada passiva é um pouco mais complexo e será abordado na próxima subseção.

O último passo após a escolha dos arcos de entrada e saída é a operação de pivoteamento. O arco de entrada deve ser inserido em T e o arco de saída deve ser retirado de T e inserido em L ou U da seguinte forma:

Se o arco de saída (i, j) for o arco ativo, então (i, j) sairá de T e será inserido em L se $x_{ij} < l_{ij}$ e será inserido em U se $x_{ij} > u_{ij}$. Caso contrário, o arco (i, j) sairá de T passivamente e será inserido em U se for *forward* ou em L se for *backward*.

É possível que o arco escolhido para entrar ativamente na base seja igual ao arco da saída. Isso ocorre quando o arco de entrada ativa tem menor índice dentre todos os arcos no ciclo C. Nesse caso, o arco de entrada ativa não entrará na árvore e esta não sofrerá alterações. Se o arco de entrada estiver em L passará para U e se estiver em U passará para L.

4.3.1 Escolha do arco que entra passivamente

A definição da regra de escolha do arco de entrada passiva é simples, mas pode apresentar desafios complexos de implementação. Para facilitar a identificação de todos os arcos fora de T que formam um ciclo com o (i, j) podemos utilizar algumas propriedades de árvores. Seja (i, j) o arco escolhido para sair da árvore, sua remoção irá formar duas subárvores. Seja T_1 a subárvore contendo a raiz e T_2 a subárvore não contendo a raiz. Qualquer arco fora de T que ligue T_1 a T_2 forma um ciclo contendo (i, j).

Se $i \in T_1$ e (i, j) for *forward* (o fluxo em (i, j) deve aumentar) ou se $i \in T_2$ e (i, j) for *backward* (o fluxo em (i, j) deve diminuir) então os arcos elegíveis para entrar na base serão os arcos que saem de T_2 e chegam em T_1 que estejam em L ou os arcos em U que saem de T_1 para T_2. Se $i \in T_1$ e (i, j) for *backward* ou $i \in T_2$ e (i, j) for *forward*, então os arcos elegíveis a sair da base serão os arcos saem de T_2 e chegam em T_1 que estejam em U ou os arcos em L que saem de T_1 para T_2. A Figura 9 mostra o algoritmo para encontrar o arco de entrada passiva. O conjunto de nós T_2 pode ser encontrado utilizando o índice *thread* e o índice profundidade dos nós. O algoritmo para encontrar T_2 é mostrado na Figura 8.

No Apêndice A mostramos exemplos de uma iteração primal e de uma iteração dual do algoritmo criss-cross com a regra do menor índice.

Figura 8 – Algoritmo para encontrar o conjunto T_2

início
(seja (i, j) o arco que sai da árvore);
$T_2 := \{\}$;
se $prof(i) > prof(j)$ **então** $y := i$ **senão** $y := j$;
adicione y a T_2;
$z := thread(y)$;
enquanto $prof(z) > prof(y)$ **faça**
adicione z a T_2;
$z := thread(y)$;
fim
fim

Figura 9 – Algoritmo para encontrar o arco que entra passivamente

início
(seja (i, j) o arco que sai da árvore);
$I = \{\}$;
se $i \notin T_2$ **e** (i, j) **é forward ou** $i \in T_2$ **e** (i, j) **é backward então**
para todo arco $(p, q) \notin T$ **faça**
se $p \in T_2$ **e** $q \notin T_2$ **e** $(p, q) \in L$ **ou** $q \in T_2$ **e** $p \notin T_2$ **e** $(p, q) \in U$ **então**
adicione (p, q) a I;
fim
fim
fim
se $i \notin T_2$ **e** (i, j) **é backward ou** $i \in T_2$ **e** (i, j) **é forward então**
para todo arco $(p, q) \notin T$ **faça**
se $p \in T_2$ **e** $q \notin T_2$ **e** $(p, q) \in U$ **ou** $q \in T_2$ **e** $p \notin T_2$ **e** $(p, q) \in L$ **então**
adicione (p, q) a I;
fim
fim
fim
escolha o arco $(p, q) \in I$ com menor índice para entrar na base;
fim

4.4 Atualização do fluxo e do potencial

Depois de realizado o pivoteamento, o fluxo nos arcos e o potencial nos nós devem ser atualizados. Na Seção 3.7 foi mostrado um método eficiente para atualizar o potencial nos nós. O fluxo também pode ser atualizado de maneira eficiente. Dado que cada operação de pivoteamento aumenta uma certa quantidade de fluxo através do ciclo C, é suficiente, portanto, atualizar o fluxo apenas nos arcos pertencentes a C.

Se o arco de saída (i, j) for o arco ativo, e seja $\underline{\delta}_{ij}$ a quantidade máxima de fluxo que é possível de se aumentar no arco $(i, j) \in C$ até atingir o primeiro limite de fluxo no sentido de C, ou seja,

1. $\underline{\delta}_{ij} = l_{ij} - x_{ij}$ se $x_{ij} < l_{ij}$,

2. $\underline{\delta}_{ij} = x_{ij} - u_{ij}$ se $x_{ij} > u_{ij}$,

então a quantidade de fluxo a ser aumentada através de C será $\Delta = \underline{\delta}_{ij}$.

Caso contrário o arco de saída (i, j) é passivo. Seja $\overline{\delta}_{ij}$ a quantidade máxima de fluxo que é possível de se aumentar no arco $(i, j) \in C$ até atingir o último limite de fluxo no sentido de C, ou seja,

1. $\overline{\delta}_{ij} = u_{ij} - x_{ij}$ se (i, j) é *forward*,

2. $\overline{\delta}_{ij} = x_{ij} - l_{ij}$ se (i, j) é *backward*,

então a quantidade de fluxo a ser aumentada através de C será $\Delta = \overline{\delta}_{ij}$.

O fluxo é aumentado sempre no sentido de C. Isso significa que os arcos *forward* sofrerão um aumento de Δ unidades de fluxo e os arcos *backward* sofrerão um aumento de $-\Delta$ unidades de fluxo.

4.5 Melhorando a regra de menor índice

A regra de pivoteamento de menor índice sempre vai escolher o arco com menor índice dentre os arcos elegíveis, tanto para o arco ativo quanto para o arco passivo. Isso significa que essa regra não se preocupa com as consequências da escolha, apenas com a finitude do método.

Por ser puramente combinatória, a regra do menor índice pode, em uma única operação de pivoteamento, inviabilizar vários arcos em detrimento da viabilidade de apenas um.

Em uma operação de pivoteamento em que o arco ativo (i, j) entra e o arco passivo (p, q) sai da base, a quantidade de fluxo a ser aumentada através do ciclo C será $\overline{\delta}_{pq}$. Se $\overline{\delta}_{pq}$ for o maior dentre todos os arcos de C, então, com exceção do arco de saída (p, q), todos os arcos em C se tornarão primal inviáveis.

Uma forma mais inteligente de escolher o arco de saída passiva seria escolher o arco pertencente a C com menor $\overline{\delta}$. Dessa forma, a viabilidade da maioria dos arcos ao longo de C seria mantida e as iterações para levá-los de volta à viabilidade seriam evitadas. Essa ideia é a mesma da regra de escolha do arco de saída do algoritmo simplex de rede mostrada na Seção 3.6, mas com a diferença de que $\overline{\delta}$, no criss-cross, pode assumir valores negativos já que alguns arcos ao longo de C poderão estar inviáveis.

Da mesma forma, uma operação de pivoteamento em que o arco ativo sai e o arco passivo entra na base também pode gerar alguns arcos inviáveis. A alteração de fluxo, nesse caso, será dada por $\underline{\delta}$ do arco de saída. Se a capacidade de aumento de fluxo do arco de entrada for inferior a $\underline{\delta}$ ele entrará na base inviável. Se alterarmos a regra de escolha do arco de entrada passiva para escolher o arco com maior capacidade de alteração de fluxo em detrimento do arco com índice mínimo, as chances do arco entrar inviável na base diminuem.

No Capítulo 5 mostraremos que essas alterações na regra de escolha do arco passivo causam uma redução significativa no número total de iterações necessárias para encontrar a solução

ótima. Nos referenciaremos a essa regra como “regra do menor índice de menor impacto”, pois o arco ativo continua sendo escolhido pelo menor índice, mas o arco passivo será escolhido de forma a reduzir o impacto no estado de viabilidade da solução. Chamaremos o algoritmo criss-cross de rede com a regra do menor índice de CCMI. O criss-cross de rede com a regra de menor índice de menor impacto chamaremos de CCMI^2.

No Apêndice A mostramos exemplos de uma iteração primal e de uma iteração dual do algoritmo criss-cross com a regra do menor índice de menor impacto.

4.6 Finitude do criss-cross

Fukuda e Terlaky (1997) demonstra a finitude da regra do menor índice do algoritmo criss-cross para problemas de PL. Uma das premissas básicas citadas pelos autores é a de que ao fim de um pivoteamento, se a iteração for primal, o arco ativo e o arco passivo devem se tornar primal viáveis; se a iteração for dual, ambos os arcos devem se tornar dual viáveis. Para o problema de fluxo de custo mínimo não-capacitado, isto é, quando o limite inferior é zero e o superior é infinito, essa premissa se faz verdadeira. No entanto, para o problema de fluxo de custo mínimo capacitado, que é o caso de estudo desse trabalho, essa premissa não é sempre satisfeita.

A prova apresentada por Fukuda e Terlaky (1997) pode, portanto, não ser condizente com a regra de pivoteamento do menor índice para o criss-cross de rede apresentada neste capítulo (algoritmo CCMI), e pode exigir um estudo mais detalhado do comportamento do algoritmo em cada iteração de pivoteamento. A prova da finitude do CCMI bem como a do CCMI^2 é o principal objetivo para trabalhos futuros.

Apesar de não demostrarmos a finitude dos algoritmos CCMI e CCMI^2, por não ser o foco deste trabalho, realizamos, em cada algoritmo 30.000 (trinta mil) testes com instancias aleatórias e ambos foram bem sucedidos em todas as execuções, chegando ao resultado ótimo em um número finito de iterações. Isso nos permite conjecturar a finitude destes algoritmos. Se a finitude do CCMI^2 se comprovar, teremos encontrado a primeira regra de pivoteamento não combinatória para criss-cross. Embora essa regra seja projetada para o cenário particular dos problemas de FCM, sua descoberta representa uma contribuição interessante para a área.

5 TESTES COMPUTACIONAIS

Neste capítulo mostramos uma comparação de desempenho entre o algoritmo simplex de rede e as duas versões do criss-cross mostradas no Capítulo 4: o CCMI com a regra de menor índice adaptada para problemas de rede e o CCMI^2 com a regra do menor índice de menor impacto, modificação da regra do menor índice proposta neste trabalho. Mostraremos ainda os resultados para o CCMI^2 Fino e para o CPLEX(netopt), que é a implementação do simplex do software CPLEX. Iniciamos mostrando alguns detalhes relevantes da implementação dos algoritmos, em seguida relatamos a origem das instâncias do problema de FCM para a realização dos testes. Por último mostramos os resultados da comparação dos algoritmos e análise sobre os resultados obtidos.

5.1 Implementação e ambiente de testes

Existem várias ferramentas que disponibilizam implementações prontas do simplex de rede, como é o caso da ferramenta CPLEX, que utilizam suas próprias estruturas de dados otimizadas. No entanto, com o objetivo de realizarmos uma comparação mais justa de desempenho, optamos por implementar o simplex de rede compartilhando as mesmas estruturas utilizadas para a implementação das versões do criss-cross.

O núcleo dos algoritmos foi implementado de maneira idêntica, alterando-se apenas as regas de pivoteamento, e, no caso do simplex, a inicialização da base. As implementações foram escritas na linguagem C++ standard. Utilizamos o compilador gcc versão 4.8.2 no sistema operacional Ubuntu 14.04 LTS 64 bits. Os programas foram executados em uma máquina dedicada com processador *Intel Core i3* de 2.3 GHz e 4 núcleos, 4 GB de memória RAM.

5.2 Instâncias NETGEN

NETGEN é um conhecido gerador de instancias aleatórias para o problema do FCM e outros problemas de otimização em rede. O NETGEN foi desenvolvido por Klingman, Napier e Stutz (1974) e tem código aberto. Uma versão do código fonte escrita em C, funcional e equivalente, pode ser encontrada em DIMACS (1990-1991).

Para a realização dos teste foram utilizadas duas classes de instâncias:

- **NETGEN-LN.** Instâncias esparsas; $m \approx n \ln(n)$.
- **NETGEN-SR.** Instâncias densas; $m \approx n\sqrt{n}$.

Para ambas as classes as instâncias são geradas da seguinte forma. O custo e a capacidade dos arcos são escolhidos aleatoriamente no intervalo [1,100]. O número de nós de produção e de consumo são ambos iguais a $\sqrt{n}$ aredondado para baixo e a quantidade de total fluxo que deve

Tabela 1 – Instâncias NETGEN-LN.

Instâncias	Nós	Arcos
NGLN_1	40	147
NGLN_2	60	245
NGLN_3	80	350
NGLN_4	100	460
NGLN_1	500	3107

Fonte: Elaborada pelo autor.

Tabela 2 – Instâncias NETGEN-SR.

Instâncias	Nós	Arcos
NGSR_1	40	252
NGSR_2	60	464
NGSR_3	80	715
NGSR_4	100	1000
NGSR_5	500	11180

Fonte: Elaborada pelo autor.

ser enviada dos nós de produção aos nós de consumo é definida em $100\sqrt{n}$ também aredondado para baixo.

Para a realização dos testes definimos cinco tamanhos diferentes de instâncias para cada classe, como é mostrado na Tabela 1 e na Tabela 2.

5.3 Resultados e análise dos testes

A seguir mostraremos algumas tabelas com os resultados dos testes realizados. A Tabela 3 mostra o tempo de execução dos algoritmos. A coluna *instâncias* referencia as instâncias mostradas nas tabelas 1 e 2. As demais colunas relacionam, respectivamente, os algoritmos CCMI, CCMI^2, CCMI^2 Fino, Simplex e CPLEX(netopt). A coluna Simplex representa o algoritmo simplex aqui implementado partilhando as estruturas básicas das implementações do criss-cross.

O tempo mostrado em cada célula foi medido em segundos e é o valor médio dentre três execuções do algoritmo para uma mesma instância. A presença do símbolo "–", em todas as tabelas, significa que o algoritmo correspondente àquela coluna não foi capaz de resolver a instância correspondente em um tempo inferior a 20 minutos (1200 segundos) e a execução foi interrompida.

A Tabela 4 mostra o número total de iterações que cada algoritmo levou para resolver cada uma das instâncias. A Tabela 5 mostra o número de iterações na primeira e na segunda fase de cada algoritmo na resolução de cada instância. São mostrados apenas os algoritmos com duas fases: CCMI^2 Fino, Simplex e CPLEX(netopt).

Tabela 3 – Tempo de execução dos algoritmos.

Instâncias	CCMI	CCMI^2	CCMI^2 Fino	Simplex	CPLEX(netopt)
NGSR_1	15,2908	0,0149	0,0032	0,0024	0,0012
NGSR_2	407,0350	0,0359	0,0075	0,0052	0,0009
NGSR_3	–	0,1101	0,0176	0,0106	0,0011
NGSR_4	–	0,3316	0,0594	0,0227	0,0015
NGSR_5	–	83,4082	3,4041	1,3396	0,0227
NGLN_1	0,5788	0,0056	0,0031	0,0019	0,0012
NGLN_2	18,1281	0,0151	0,0060	0,0040	0,0009
NGLN_3	99,4038	0,0598	0,0137	0,0085	0,0009
NGLN_4	180,1450	0,0966	0,0239	0,0157	0,0011
NGLN_5	–	11,1839	2,4481	0,5006	0,0081

Fonte: Elaborada pelo autor.

Uma célula com o valor 30 / 40 significa que foram realizadas 30 iterações na primeira fase e 40 na segunda fase do algoritmo. Pada o CCMI^2 Fino significa também que a viabilidade primal foi atingida com 30 iterações, e outras 40 iterações do simplex primal foram necessárias para se alcançar a solução ótima.

Tabela 4 – Número de iterações dos algoritmos.

Instâncias	CCMI	CCMI^2	CCMI^2 Fino	Simplex	CPLEX(netopt)
NGSR_1	469860	427	156	155	110
NGSR_2	7484807	620	254	209	209
NGSR_3	–	1351	451	313	345
NGSR_4	–	2983	1224	529	426
NGSR_5	–	68266	10340	3782	3880
NGLN_1	23075	194	148	123	108
NGLN_2	462449	357	229	182	165
NGLN_3	1810799	1044	378	288	202
NGLN_4	2561662	1316	544	427	282
NGLN_5	–	25044	9790	2298	1995

Fonte: Elaborada pelo autor.

5.3.1 CCMI

Em uma análise rápida das tabelas é possível notar que o algoritmo CCMI é extremamente ineficiente. Ele consome tempo e número de iterações extremamente elevamos, mesmo para instâncias pequenas. Uma justificativa para este comportamento está no fato de a regra ser puramente combinatória, e não há preocupação com o valor da função objetivo nem com a manutenção da viabilidade da solução atual. A Figura 10 mostra o comportamento usual do algoritmo CCMI. Podemos notar que não há um decréscimo contínuo do estado de inviabilidade

Tabela 5 – Número de iterações da primeira e segunda fase dos algoritmos.

Instâncias	CCMI^2 Fino	Simplex	CPLEX(netopt)
NGSR_1	87 / 69	79 / 76	61 / 49
NGSR_2	156 / 98	94 / 115	98 / 111
NGSR_3	276 / 175	115 / 198	136 / 209
NGSR_4	955 / 269	190 / 339	166 / 260
NGSR_5	7360 / 2980	907 / 2875	971 / 2909
NGLN_1	114 / 34	69 / 54	87 / 21
NGLN_2	146 / 83	90 / 92	97 / 68
NGLN_3	231 / 147	125 / 163	133 / 69
NGLN_4	368 / 176	210 / 217	179 / 103
NGLN_5	8528 / 1262	754 / 1544	817 / 1178

Fonte: Elaborada pelo autor.

total da solução. Uma iteração primal reduz ligeiramente a inviabilidade primal, mas aumenta a inviabilidade dual consideravelmente. O mesmo ocorre em uma iteração dual.

5.3.2 CCMI^2

O algoritmo CCMI^2 representa uma considerável melhora de desempenho comparando-se ao CCMI. Sua operação de pivoteamento se preocupa com o estado de viabilidade da base na iteração atual. No entanto, parte da regra se baseia em uma escolha combinatória, sem se preocupar com o valor da função objetivo. Ocorre uma redução relativamente constante da inviabilidade, porém essa redução ocorre lentamente, o que gera um grande número de iterações e consequentemente grande consumo de tempo.

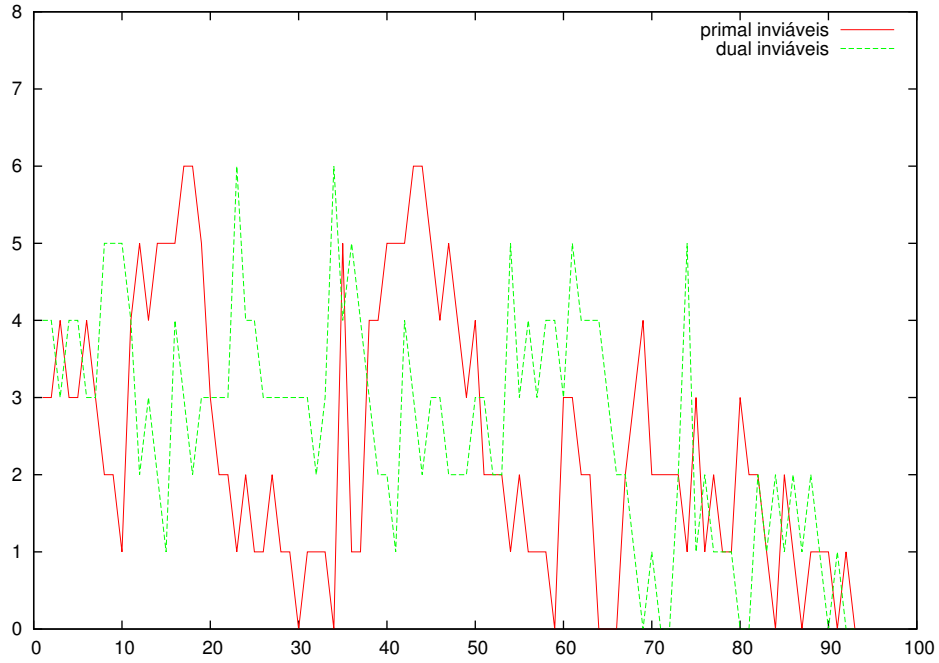
5.3.3 CCMI^2 Fino

O CCMI^2 na grande maioria das vezes alcança a viabilidade primal rapidamente. Esse fato motivou a utilização do CCMI^2 como primeira fase para o simplex (CCMI^2 Fino). As características próprias da regra fazem com que ela se comporte como o simplex primal de índice mínimo quando a viabilidade primal é alcançada. Nesse momento, o algoritmo deixa de usar a regra de menor índice de menor impacto e passa e utilizar a regra de Dantzig para se obter um melhor desempenho.

De fato o CCMI^2 Fino representa um grande ganho de desempenho sobre o CCMI^2. Foi o algoritmo criss-cross que mais se aproximou do desempenho dos algoritmos da família simplex em termos de número de iterações e de tempo, se comparado com o simplex aqui implementado. Também chegou a números próximos aos do CPLEX(netopt) em termos de número de iterações.

Na Tabela 5 podemos notar que o CCMI^2 Fino na maioria das vezes realiza um número menor de iterações na segunda fase que o algoritmo Simplex. Isso significa que a base primal

Figura 10 – Comportamento do algoritmo CCMI



Fonte: Autor

viável encontrada no CCMI^2 Fino é melhor que a encontrada na primeira fase do Simplex, que utiliza a técnica de arcos artificiais. No entanto, o número de iterações na primeira fase do CCMI^2 Fino ainda é superior ao número de iterações na primeira fase do Simplex. O Simplex converge mais rápidamente para uma base viável pois visa o valor da função objetivo. O CCMI^2 Fino ainda utiliza uma regra que se baseia no menor índice até encontrar a viabilidade primal, o que representa uma desvantagem por ser uma regra parcialmente combinatória. Um possível trabalho futuro seria encontrar uma regra não combinatória finita para o criss-cross, que possa convergir mais rapidamente para a viabilidade primal.

6 CONCLUSÃO

Os algoritmos criss-cross ainda são pouco estudados e raramente mencionados na literatura. A adaptação do algoritmo criss-cross para resolver problemas de rede, como o de FCM, é inédita na literatura. Isso nos trouxe o desafio de adaptar a regra do menor índice do criss-cross convencional para ser aplicada em problemas de rede, o que resultou no algoritmo CCMI.

Os testes realizados no Capítulo 5 mostram que o algoritmo CCMI é muito ineficiente se comparado com os algoritmos da família simplex. A ineficiência do CCMI se justifica na natureza combinatória da regra de pivoteamento do menor índice. Com pequenas melhorias na regra de menor índice obtivemos o algoritmo CCMI^2, que apresentou uma grande melhoria de desempenho em relação ao CCMI. Isso mostra que o estudo de uma regra de pivoteamento não combinatória para algoritmos criss-cross é muito promissor e pode levar a bons resultados.

Alguns testes que mostravam que o CCMI^2 encontra uma base primal viável rapidamente nos motivou a desenvolver o CCMI^2 Fino que utiliza a regra do menor índice de menor impacto para encontrar uma base viável e a partir desse ponto utiliza a regra de pivoteamento de nossa implementação do Simplex para alcançar a solução ótima. O CCMI^2 Fino apresentou uma grande melhoria de desempenho sobre o CCMI e CCMI^2 e se aproximou do CPLEX(netopt) em termos de número de iterações, apesar de apresentar tempos de execução muito superiores. Uma justificativa para a discrepância de tempo é que CPLEX é uma ferramenta bem desenvolvida que utiliza estruturas de dados otimizadas e bem definidas. Os testes mostraram que CCMI^2 Fino obteve tempo próximo ao da nossa implementação do simplex, que compartilha as mesmas estruturas de dados utilizadas na implementação das versões do criss-cross. Esse fato nos leva a acreditar que uma comparação mais justa de tempo com o CPLEX(netopt) poderia ser realizada se utilizássemos as estruturas do CPLEX na implementação do CCMI^2 Fino.

Obtivemos bons resultados com os algoritmos CCMI^2 e CCMI^2 Fino, apesar de os mesmos não terem superado os algoritmos simplex. Alcançamos o objetivo do trabalho e obtivemos um método para resolver problemas de FCM que não necessita de duas fases e se aproxima ao desempenho dos algoritmos da família simplex tão amplamente estudados na literatura. O principal objetivo em trabalhos futuros seria de encontrar uma regra de pivoteamento que acelere a convergência para a solução ótima no caso do CCMI^2, ou para uma base viável no caso do CCMI^2 Fino.

Também se faz necessária a prova da finitude dos algoritmos criss-cross propostos, que não foi realizada neste trabalho. Mas devemos resaltar que as evidências empiricas suportam a conjectura da finitude dos algoritmos CCMI, CCMI^2 e CCMI^2 Fino baseadas nos resultados dos 30 mil testes com instâncias aleatórias realizados nos mesmos. Resaltamos ainda que, se a finitude do CCMI^2 se comprovar, teremos realizado uma contribuição interessante para a área ao encontrar a primeira regra de pivoteamento não combinatória para criss-cross, apesar de ser projetada apenas para o problema de FCM.

REFERÊNCIAS

AHUJA, R. K.; MAGNANTI, T. L.; ORLIN, J. B. **Networks Flows**: theory, algorithms, and applications. Upper Saddle River: Prentice Hall, 1993. 846 p.

BERTSEKAS, D. P. **Network Optimization**: continuous and discrete models. Belmont: Athena Scientific, 1998. 585 p.

BONATES, T. de Oliveira e. **Implementação do método criss-cross para problemas de programação linear de grande porte**. 2001. 93 p. Dissertação de Mestrado(Programa de Engenharia de Sistemas e Computação) — COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2001.

BONATES, T. de Oliveira e; MACULAN, N. Performance evaluation of a family of criss–cross algorithms for linear programming. **International Transactions in Operational Research**, Rio de Janeiro, v. 10, n. 1, p. 53–64, Jan. 2003.

DIMACS. **Network flows and matching**: first DIMACS implementation challenge. 1990-1991.

FUKUDA, K.; TERLAKY, T. Criss-cross methods: a fresh view on pivot algorithms. **Mathematical Programming**, Delft, p. 369–395, 1997.

KLINGMAN, D.; NAPIER, A.; STUTZ, J. Netgen: a program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. **Management Science**, [S.l.], v. 20, n. 5, p. 814–821, 1974.

NETTO, P. O. B. **Grafos**: teoria, modelos, algoritmos. Quarta. ed. São Paulo: Edgar Blucher, 2006. 328 p.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization**: algorithms and complexity. Mineola: Dover Publications, Inc., 1998. 496 p.

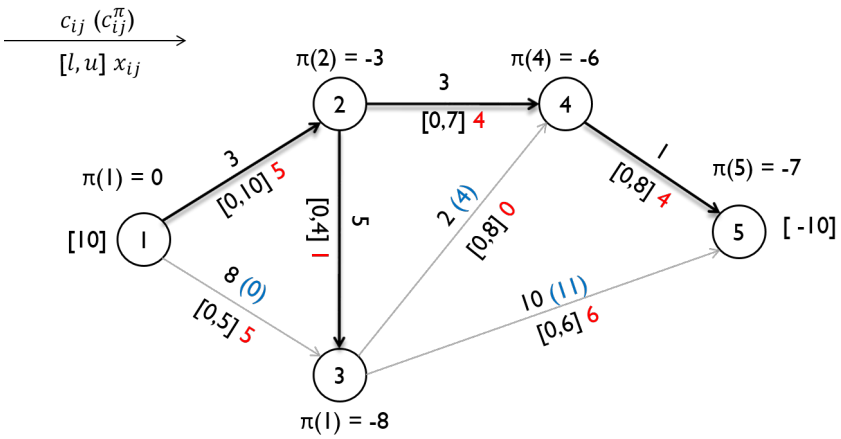
APÊNDICE A – EXEMPLOS DE PIVOTEAMENTO

A seguir mostraremos exemplos das operações de pivoteamento dos algoritmos: simplex, criss-cross com regra do menor índice e criss-cross com regra do menor índice de menor impacto.

A.1 Simplex

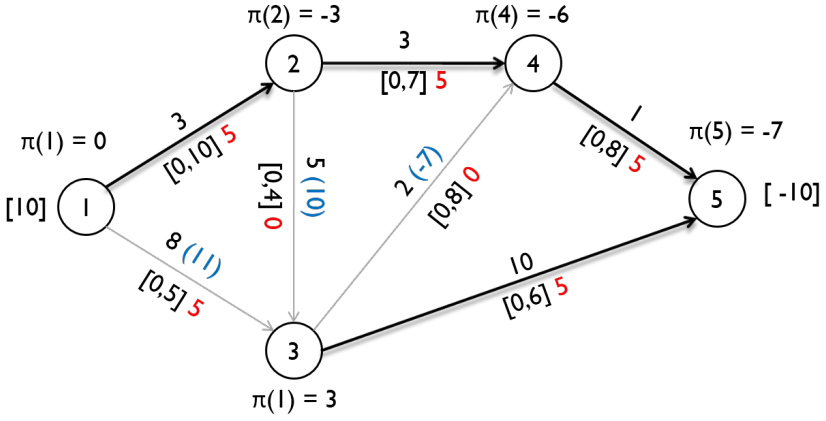
A Figura 11 mostra uma EAG viável que será usada como solução inicial do simplex para o nosso exemplo. Nesse momento temos o arco $(3, 5)$ inviável. De acordo com a algoritmo simplex com a regra de pivoteamento de Dantzig o arco $(3, 5)$ é escolhido para entrar na base e o arco $(2, 3)$ é escolhido para sair.

Figura 11 – Simplex: Estado 0



A Figura 12 mostra o resultado da operação de pivoteamento entre os arcos escolhidos para entrar e sair da base.

Figura 12 – Simplex: Estado 1



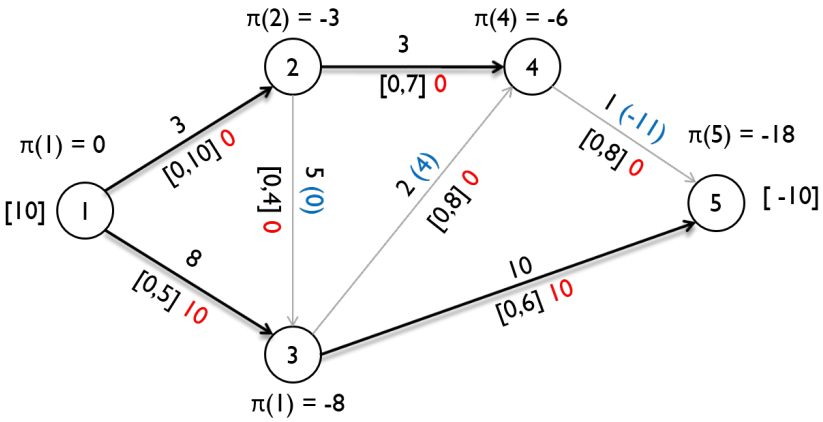
A.2 Criss-cross Menor Índice

A seguir mostraremos uma iteração primal e uma iteração dual do algoritmo criss-cross com a regra do menor índice.

A.2.1 Iteração Primal

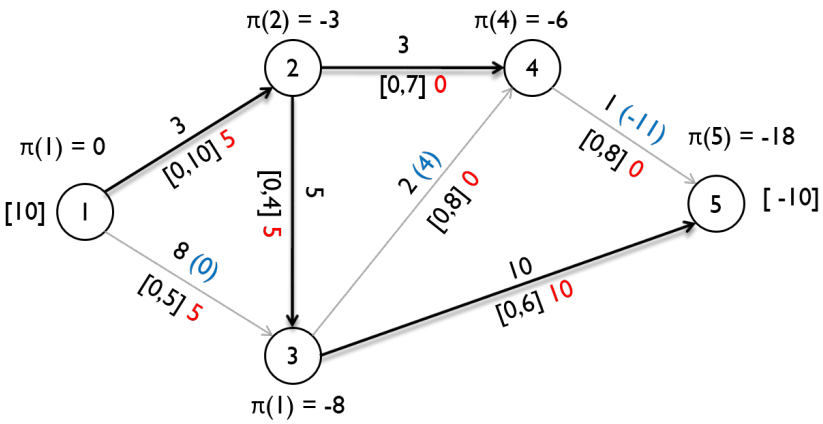
A Figura 13 mostra uma EAG inviável que será usada como solução inicial para o nosso exemplo de iteração primal do criss-cross com regra do menor índice. Os arcos $(1,3)$, $(3,5)$ são primal inviáveis e o arco $(4,5)$ é dual inviável. Pela regra do menor índice o arco $(1,3)$ é escolhido para sair da base ativamente e o arco $(2,3)$ é escolhido para entrar na base passivamente.

Figura 13 – CCMI: Estado 0



A Figura 14 mostra o resultado da operação de pivoteamento entre os arcos escolhidos para entrar e sair da base.

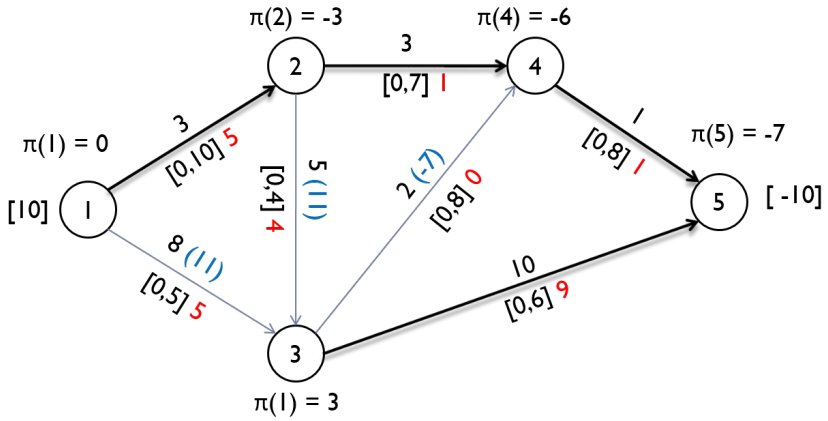
Figura 14 – CCMI: Estado 1



A.2.2 Iteração Dual

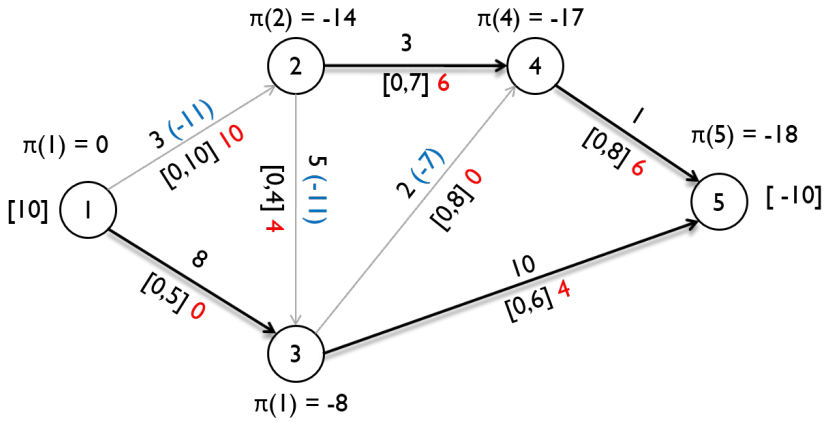
A Figura 15 mostra uma EAG inviável obtida após algumas iterações do criss-cross de menor índice. Temos os arcos $(1,3)$, $(2,3)$ e $(3,4)$ dual inviáveis e o arco $(3,5)$ primal inviável. Pela regra do menor índice o arco $(1,3)$ é escolhido para entrar na base ativamente e o arco $(1,2)$ é escolhido para sair da base passivamente.

Figura 15 – CCMI: Estado 2



A Figura 16 mostra o resultado da operação de pivoteamento entre os arcos escolhidos para entrar e sair da base.

Figura 16 – CCMI: Estado 3



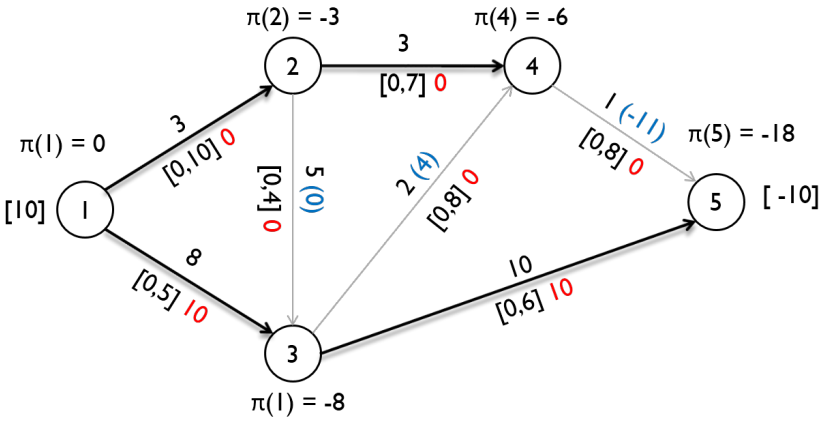
A.3 Criss-cross Menor Índice de Menor Impacto

Mostraremos agora uma iteração primal e uma iteração dual do algoritmo criss-cross com a regra do menor índice de menor impacto.

A.3.1 Iteração Primal

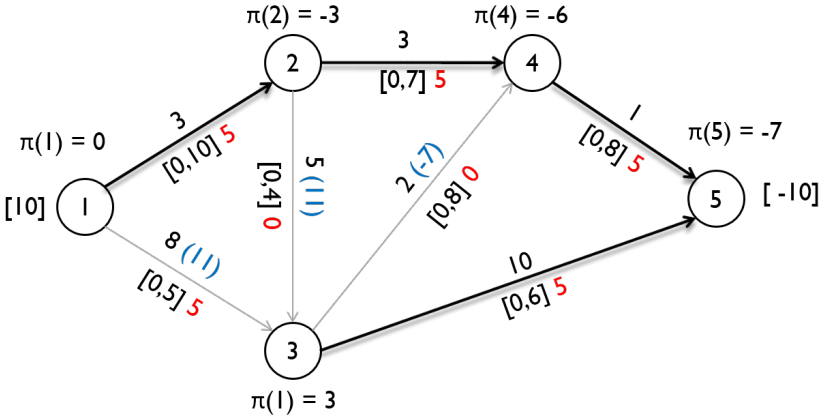
Vamos partir do mesmo estado inicial que utilizamos para exemplificar a iteração primal do CCMI. A Figura 17 mostra a EAG inicial. Temos os arcos $(1,3)$, $(3,5)$ primal inviáveis e o arco $(4,5)$ dual inviável. Pela regra de menor índice de menor impacto o arco $(1,3)$ é escolhido para sair ativamente da base e o arco $(4,5)$ é escolhido para entrar na base passivamente.

Figura 17 – CCMI^2: Estado 0



A Figura 18 mostra o resultado do pivoteamento entre os arcos escolhidos para entrar e sair da base com a regra do menor índice de menor impacto. Na Figura 14 podemos ver que o arco que entra se torna primal inviável com a aplicação da regra de menor índice. Já com a regra de menor índice de menor impacto aplicada no mesmo exemplo o arco que entra não se torna inviável.

Figura 18 – CCMI^2: Estado 1

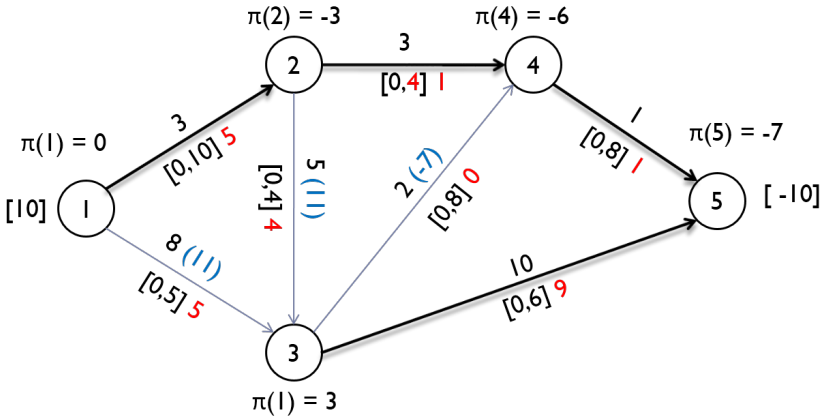


A.3.2 Iteração Dual

Vamos partir, também, do mesmo estado que utilizamos para exemplificar a iteração dual do CCMI, mas com uma pequena alteração no limite superior do arco $(2,4)$. A Figura 19 mostra

a EAG para para o exemplo. Temos os arcos $(1,3)$, $(2,3)$ e $(3,4)$ dual inviáveis e o arco $(3,5)$ primal inviável. Pela regra de menor índice de menor impacto o arco $(1,3)$ é escolhido para sair ativamente da base e o arco $(2,4)$ é escolhido para entrar na base passivamente.

Figura 19 – CCMI^2: Estado 2



A Figura 20 mostra o resultado do pivoteamento entre os arcos escolhidos para entrar e sair da base com a regra do menor índice de menor impacto. A aplicação da regra de menor índice nesse exemplo tornaria primal inviável o arco $(2,4)$. Já com a regra de menor índice de menor impacto aplicada no mesmo exemplo nenhum arco se torna inviável no ciclo formado pelo arco que entra e os arcos já pertencentes a base.

Figura 20 – CCMI^2: Estado 3