UNESP
Faculdade de Engenharia de Guaratinguetá

Guaratinguetá
1996

UNIVERSIDADE ESTADUAL PAULISTA

FACULDADE DE ENGENHARIA

CAMPUS DE GUARATINGUETÁ

DEPARTAMENTO DE PRODUÇÃO

ALGORITMOS PARA REDES LINEARES POR PARTES: UM ESTUDO COMPARATIVO ESTATÍSTICAMENTE PLANEJADO

Trabalho apresentado, como parte dos requisitos exigidos, para o concurso de Livre-Docência, na disciplina de Pesquisa Operacional, do Departamento de Produção da Faculdade de Engenharia do Campus de Guaratinguetá, da Universidade Estadual Paulista - UNESP.

AGOSTO DE 1996

unesp

FERNANDO AUGUSTO SILVA MARINS

ENGENHEIRO MECÂNICO, FACULDADE DE ENGENHARIA DO CAMPUS DE GUARATINGUETÁ - UNESP, 1976

MESTRE EM CIÊNCIAS, INSTITUTO TECNOLÓGICO DE AERONÁUTICA, 1981

DOUTOR EM ENGENHARIA ELÉTRICA, FACULDADE DE ENGENHARIA ELÉTRICA - UNICAMP, 1987

PÓS-DOUTORAMENTO EM PROGRAMAÇÃO EM REDES LINEARES POR PARTES, BRUNEL UNIVERSITY, 1994

ALGORITMOS PARA REDES LINEARES POR PARTES: UM ESTUDO COMPARATIVO ESTATÍSTICAMENTE PLANEJADO

Trabalho apresentado, como parte dos requisitos exigidos, para o concurso de Livre-Docência, na disciplina de Pesquisa Operacional, do Departamento de Produção da Faculdade de Engenharia do Campus de Guaratinguetá, da Universidade Estadual Paulista - UNESP.

T.65.012.122(043)
M339a

312

FACULDADE DE ENGENHARIA
DE GUARATINGUETÁ
BIBLIOTECA

000312

AGOSTO DE 1996

A minha mãe Odila, in memoriam;

ao meu pai José;

a minha esposa Maria Luisa,

e aos meus filhos Livia e Fabio.

AGRADECIMENTOS

Desejo expressar meus sinceros agradecimentos:

ao amigo ***Clovis Perin Filho*** do IMECC - UNICAMP, por toda a colaboração e apoio constantes durante o desenvolvimento da pesquisa;

ao amigo ***Edson Luiz França Senne*** do DMA - FEG - UNESP, pelo auxílio importantíssimo quando da realização dos testes computacionais, bem como na leitura crítica do texto inicial;

ao amigo ***Kenneth Darby-Dowman*** do Departamento de Matemática Aplicada e Estatística - Brunel University, pelas sugestões durante a análise estatística executada neste trabalho;

ao amigo ***Antonio Fernando Branco Costa*** do DPD - FEG - UNESP, pela disposição, mesmo à distância, em discutir os aspectos relacionados com o planejamento dos experimentos realizados;

enfim, a todos que tenham colaborado, direta ou indiretamente, para a elaboração deste trabalho.

RESUMO

A Programação Linear por Partes - PLP é uma importante área da Programação Matemática, e trata da minimização de uma função objetivo convexa separável e linear por partes, sujeita a restrições lineares. Neste trabalho uma sub-área da PLP, denominada Programação em Redes Lineares por Partes - PRLP, é abordada. Apresenta-se a descrição de quatro algoritmos especializados para a PRLP, e se estuda a eficiência relativa deles: Primal Simplex, Dual Simplex, Out-of-Kilter, e Cost Scaling. Um experimento estatísticamente planejado é utilizado para realizar uma comparação computacional dos algoritmos. Neste experimento, são gerados problemas em redes lineares por partes de várias categorias, que dependem de quatro fatores considerados: a classe e a dimensão do problema, a porcentagem de arcos capacitados na rede, e o número de limitantes de intervalos lineares por partes ("breakpoints") correspondentes a cada arco da rede. Observa-se o tempo de CPU gasto pelos algoritmos na resolução destas redes geradas, para várias combinações de valores possíveis para os fatores admitidos, e são efetuados testes estatísticos com os resultados obtidos, visando obter respostas consistentes para questões colocadas acerca dos desempenhos dos algorimos nas categorias de problemas analisadas.

unesp

ABSTRACT

Piecewise-Linear Programming - PLP is an important area of Mathematical Programming, and concerns the minimisation of a convex separable piecewise-linear objective function, subject to linear constraints. In this work a subarea of PLP called Network Piecewise-Linear Programming - NPLP is explored. The work presents four specialised algorithms for NPLP, Primal Simplex, Dual Simplex, Out-of-Kilter, and Cost Scaling, and their relative efficiency is studied. A statistically designed experiment is used to perform a computational comparison of the algorithms. The response variable observed in the experiment is the CPU time to solve randomly generated network piecewise-linear problems classified according to problem class, problem size, extent of arc capacitation, and number of breakpoints per arc. Results and conclusions on performance of the algorthms are reported.

unesp

SUMÁRIO

	Página
CAPÍTULO 1. PRELIMINARES	
1.1. INTRODUÇÃO	1.1
1.1.1. A PROGRAMAÇÃO LINEAR POR PARTES	1.1
1.1.2. A PROGRAMAÇÃO EM REDES LINEARES POR PARTES	1.7
1.2. OBJETIVOS	1.13
1.3. SUMÁRIO	1.14
CAPÍTULO 2. ALGORITMOS PARA PROGRAMAS EM REDES LINEARES POR PARTES	
2.1. INTRODUÇÃO	2.1
2.2. PROGRAMAS EM REDES LINEARES POR PARTES	2.1
2.3. ALGORITMOS ESPECIALIZADOS	2.7
2.3.1. ALGORITMO PRIMAL SIMPLEX	2.7
2.3.2. ALGORITMO DUAL SIMPLEX	2.12
2.3.3. ALGORITMO OUT-OF-KILTER	2.18
2.3.4. ALGORITMO COST SCALING	2.22
CAPÍTULO 3. TESTES COMPUTACIONAIS	
3.1. INTRODUÇÃO	3.1
3.2. O PROJETO DOS EXPERIMENTOS COMPUTACIONAIS	3.2
3.2.1. HISTÓRICO	3.2
3.2.2. O EXPERIMENTO COMPUTACIONAL	3.5



SUMÁRIO

Página

CAPÍTULO 3. TESTES COMPUTACIONAIS (CONTINUAÇÃO)

3.2.3. O PROJETO 3.7

3.2.4. A IMPLEMENTAÇÃO 3.12

3.3. RESULTADOS DOS EXPERIMENTOS 3.15

CAPÍTULO 4. CONCLUSÕES E COMENTÁRIOS FINAIS

4.1. INTRODUÇÃO 4.1

4.2. ANÁLISE DOS RESULTADOS 4.1

4.3. VERIFICAÇÃO DOS OBJETIVOS 4.6

4.4. DIREÇÕES DE PESQUISAS 4.9

4.5. COMENTÁRIOS FINAIS 4.10

REFERÊNCIAS BIBLIOGRÁFICAS

APÊNDICE 1. MANUAL DE USUÁRIO DO NplSys

ÍNDICE DE FIGURAS E TABELAS

Página

Figura 1. Função convexa linear por partes 2.2

Figura 2. Possíveis situações no ciclo W 2.14

Figura 3. Diagrama de Kilter do arco j 2.19

Figura 4. Condições de otimalidade exata para um arco com $x_j \in \left[d_j^{f(j)}, d_j^{f(j)+1}\right]$ 2.25

Figura 5. Condições de α-otimalidade para um arco com $x_j \in \left[d_j^{f(j)}, d_j^{f(j)+1}\right]$ 2.26

Figura 6. Arquitetura do NplSys 3.13

Tabela 1. Níveis do fator dimensão do problema 3.6

Tabela 2. Parâmetros da rede 3.14

Tabela 3. Distribuição dos parâmetros dos problemas 3.14

Tabela 4. Experimento 1: tempos de CPU para problemas de transporte clássico 3.17

Tabela 5. Experimento 1: tempos de CPU para problemas de transporte com baldeação 3.17

Tabela 6. Experimento 1: tempos de CPU para problemas de circulação 3.18

Tabela 7. Experimento 2: tempos de CPU para problemas de transporte clássico 3.18

Tabela 8. Experimento 2: tempos de CPU para problemas de transportecom baldeação 3.19

Tabela 9. Experimento 2: tempos de CPU para problemas de circulação 3.19

unesp

ÍNDICE DE FIGURAS E TABELAS

Página

Tabela 10. Experimento 1: tempos médios de CPU para cada nível de fator 3.20

Tabela 11. Experimento 1:Análise de Variância 3.20

Tabela 12. Experimento 2: tempos médios de CPU para cada nível de fator 3.21

Tabela 13. Experimento 2: Análise de Variância 3.21

Tabela 14. Experimento 1: Teste de Tukey 3.22

Tabela 15. Experimento 2.: Teste de Tukey 3.23

Tabela 16. Experimento 1: Análise de Variância Exploratória 4.2

CAPÍTULO 1

PRELIMINARES

1.1. INTRODUÇÃO

O objetivo deste trabalho é apresentar algoritmos especializados para programas em redes lineares por partes, bem como o resultado de um estudo computacional comparando os desempenhos relativos destes algoritmos, quando aplicados a vários tipos característicos de redes lineares por partes.

Nesta seção, inicialmente, procura-se situar a **Programação Linear por Partes** em geral, explicitando sua importância tanto do ponto de vista teórico como prático; e a seguir apresenta-se uma sub-área denominada **Programação em Redes Lineares por Partes**, que é o objeto do estudo aqui descrito, suas aplicações e uma revisão da literatura pesquisada.

1.1.1. A PROGRAMAÇÃO LINEAR POR PARTES

Uma importante área da Programação Matemática é a Programação Linear Por Partes - PLP. Os modelos de PLP, que fornecem uma formulação natural para muitos problemas de otimização, tratam da minimização de uma função objetivo separável e linear por partes, respeitando um conjunto de restrições representadas por um sistema de equações lineares. Portanto, nestes modelos de PLP, a função objetivo é uma soma de funções lineares por partes, cada uma das quais expressa com respeito a apenas uma das variáveis do problema.

Existem inúmeras aplicações conhecidas para a PLP das quais passa-se a comentar brevemente as que se seguem:

unesp

(a) **Aplicação da PLP na determinação de solução viável inicial em Programação Linear - PL**: isto é feito através da resolução de um programa linear por partes, cuja solução ótima é uma solução viável para o programa linear original. Esta é a aplicação mais comum e conhecida da PLP, ver, por exemplo, Wolfe [122] e Orchard-Hays [87], sendo empregada tradicionalmente em códigos comerciais, com implementações do método simplex, apropriados para problemas de grande porte;

(b) **Aplicação da PLP como um método de penalidades**: é um procedimento bem conhecido, que acrescenta uma penalidade linear por partes à função objetivo do programa linear original. Isto é feito de tal forma que toda solução viável e ótima do programa linear original é também uma solução ótima para o problema com a penalidade. Conn [23] e Bartels [16] propuseram algoritmos especiais tipo simplex para este tipo de formulação de programas;

(c) **Aplicação da PLP em atividades reversíveis ("reversible activities")**: certas situações envolvem atividades (ou variáveis) que fazem sentido tanto quando colocadas em níveis positivos como negativos, como é o caso de sistemas de estoques, com possibilidade de haver pedidos pendentes ("backorder"). Os custos associados a este tipo de variáveis podem ser representados por funções lineares por partes, apresentando termos lineares com inclinações (ou coeficientes de custo) negativas onde as atividades forem negativas, e inclinações positivas onde as atividades forem positivas.

(d) **Aplicação da PLP em problemas de estimação L_1**: em determinados problemas de estimação (ajuste de curvas) busca-se determinar, em termos da norma L_1, a solução aproximada mais acurada para um sistema de equações lineares $Ax = b$. A partir da introdução de variáveis de folga estes problemas podem ser expressos como um caso particular da PLP, onde, no modelo linear por partes, as variáveis originais x terão coeficientes de custo nulos na função objetivo linear por partes, e as variáveis de folga terão coeficientes de custo -1 (ou +1) quando elas forem negativas (ou positivas). Várias generalizações deste problema de estimação L_1 são também passíveis de serem representadas por programas lineares por partes, conforme comentado por Fourer [42], e existem inúmeros algoritmos especializados tipo simplex desenvolvidos para estas situações, tais como os descritos por Davies [32], Barrodale e Roberts [15], e Spyropoulos et al. [108] que foram os pioneiros neste enfoque;

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

(e) **Aplicação da PLP na Programação por Metas ("Goal Programming")**: para o caso da minimização da soma ponderada dos desvios, que podem ser positivos ou negativos, do produto Ax com respeito a sua igualdade com o vetor de constantes b, sujeito a restrições lineares adicionais sobre x, este modelo de programação por metas pode ser estabelecido como sendo um modelo linear por partes. Neste caso, todas as variáveis do modelo linear por partes terão dois intervalos de variação $(-\infty, 0]$ e $[0, +\infty)$ para seus valores admissíveis. As variáveis originais x terão coeficientes de custo $-\infty$ e coeficientes de custo 0, respectivamente no primeiro e segundo intervalos. Já as variáveis de folga (desvios) terão coeficientes de custo dados, no intervalo $(-\infty, 0]$ pelo peso usado na ponderação dos desvios positivos , e no intervalo $[0, +\infty)$ pelo peso adotado para os desvios negativos;

(f) **Aplicação da PLP em Programação Linear com Variáveis e Restrições Canalizadas ("Interval Programming")**: esta é a situação de problemas lineares onde existem limitantes inferiores e superiores tanto para as variáveis e como também para as restrições envolvidas. Ocorre que, conforme comentado por Fourer [42], pela teoria da dualidade da PLP existe um programa linear por partes que é o dual deste problema com variáveis e restrições canalizadas, o que permite enfoques alternativos para o caso. Existem trabalhos que exploram estas características particulares destes modelos primal e dual e apresentam algoritmos especializados, como os de Robers e Ben-Israel [94] e o de Gunn [55];

(g) **Aplicação da PLP a Programação Não-Linear** : um procedimento simplificador, muito aplicado, no estudo de problemas em Programação Não-Linear Separável, tem sido o de substituir os termos não lineares por aproximações lineares por partes, como proposto por Charnes e Lemke [19], Charnes e Cooper [20], Dantzig [29], e Dantzig et al. [31]. Em certos casos, este método pode não produzir um problema convexo separável linear por partes equivalente, pois a função objetivo resultante pode não ser convexa, ou, ainda, podem surgir restrições lineares por partes. Contudo, algumas situações relevantes neste contexto podem ser tratadas pela PLP, como descrito por Fourer [42].

Os interessados na obtenção de maiores detalhes sobre estas, e outras, aplicações da PLP podem consultar as referências Marins [70] e Fourer [42].

unesp

Com respeito a literatura consultada, deve-se destacar uma seqüência de quatro excelentes artigos, Fourer ([40], [41], [42]) e Fourer e Marsten [43], que resgataram a relevância da PLP no cenário científico. O primeiro artigo introduz, de forma rigorosa, os conceitos de funções convexas separáveis e lineares por partes e de dualidade, estabelecendo a seguir as propriedades de um programa linear por partes, a definição de solução básica, e finalmente descrevendo um algoritmo especializado para a PLP que toma vantagem das características deste tipo de modelo para se tornar mais vantajoso, do ponto de vista computacional, do que outros enfoques alternativos.

Nos segundo e terceiro trabalhos são apresentados, e discutidos, resultados teóricos relativos a convergência ("finiteness") e a validade ("correcteness") do algoritmo, questões sobre viabilidade e o seu comportamento frente a ocorrência de degenerescência.Também é feita uma análise teórica das vantagens computacionais de se aplicar o algoritmo desenvolvido para a PLP. Já no último artigo, os autores descrevem os resultados de um interessante estudo computacional realizado envolvendo o algoritmo de Fourer e um algoritmo tipo simplex bem conhecido. Este estudo comprova as vantagens do uso de um algoritmo especializado para PLP.

Adotando os critérios considerados por Fourer, nos trabalhos acima citados, há três enfoques para a PLP que vêm a classificar as pesquisas desenvolvidas nesta área em três categorias.

Primeiramente, a **PLP como caso especial da Programação Linear**. De fato, este tem sido o tratamento tradicional dado a modelos de PLP, que são então considerados como apenas mais uma aplicação da PL. Assim o procedimento usualmente aplicado é o de converter o modelo de PLP em um modelo linear equivalente, de maior porte, através de transformações conhecidas, como as propostas por Charnes e Lemke [19], Dantzig [29], Dantzig et al. [31], e Ho [56]. Este modelo linear equivalente é então resolvido por um algoritmo tipo simplex, ou ainda, por meio de algum algoritmo simplex adaptado para explorar certos aspectos particulares do modelo linear equivalente obtido. Uma descrição sucinta dos algoritmos publicados por Snyder ([104], [105]), e Premoli [91], que abordam este último tipo de enfoque, pode ser encontrada em Marins [70].

Como apresentado por Ho [56], há pelo menos quatro métodos equivalentes para linearizar problemas convexos separáveis lineares por partes. Cada uma destas transformações

unesp

leva a diferentes formulações em PL, evidentemente todas com soluções ótimas equivalentes. Os procedimentos mais conhecidos são o δ-método e o λ-método que trabalham com as variáveis do problema, além do γ-método e do σ-método que utilizam diferentes representações para a função objetivo.

É importante observar que os modelos lineares equivalentes obtidos pelos procedimentos acima possuem um número muito maior de variáveis e restrições que o modelo linear por partes original. De fato, estes métodos definem ao menos uma variável e/ou restrição adicional para cada termo linear de todas as parcelas que formam a função objetivo convexa, separável e linear por partes original.

Um segundo enfoque dos trabalhos na área consiste em considerar a **PLP como caso especial da Programação Não-Linear (PNL)**. Vários pesquisadores têm estudado importantes problemas da PNL para os quais a PLP é extremamente útil. Particularmente na área de Programação Separável, como já comentado, podem ser encontrados relevantes resultados teóricos e práticos obtidos usando a linearização por partes. Alguns destes resultados estão descritos nos trabalhos de Ahlfeld et al. [1], Geoffrion [46], Güder e Morris [54], Hochbaum e Shanthikumar [57], Kamesam e Meyer [63], Kao e Meyer [64], Meyer [80], Meyer [81], Rockafellar [97], e de Thakur ([116], [117]), comentados a seguir.

Geoffrion estabeleceu um critério natural para selecionar a "melhor" aproximação para uma função objetivo não-linear, e a referência [117] apresenta um estudo sobre o erro de truncamento que ocorre no valor ótimo da função objetivo devido ao uso de aproximações lineares por partes. Particularmente, a referência [54] apresenta um método para obter a aproximação linear por partes ótima de uma função objetivo convexa, separável e quadrática, e recomenda o uso de algoritmos especializados para resolver o modelo de PLP resultante.

O artigo de Ahlfeld et al. [1] descreve uma especialização do algoritmo truncado de Newton para problemas não-lineares em redes generalizadas, e o uso de uma linearização por partes na obtenção de uma solução viável inicial. Os demais trabalhos sugerem ou a idéia de converter o programa linear por partes em um linear equivalente, e então aplicar um algoritmo de otimização linear, ou propõem o uso de algoritmos não-lineares especializados para a solução direta do modelo linear por partes.

unesp

Pode-se, ainda, citar, como relacionados com a PLP na área de PNL, importantes trabalhos em Otimização Monotrópica ("Monotropic Optimisation"), como o de Rockafellar [96], o de Snyder [105] em métodos do gradiente reduzido para Otimização sob Restrições , os de Bartels [16] e de Conn [23], na solução de problemas de estimação L_1, e o de Perold [90] para problemas de otimização de "portfolios" com critério quadrático.

Uma terceira categoria de trabalhos envolvendo modelos de PLP, trata a **PLP como generalização da PL**. Os artigos de Fourer ([40], [41], [42]) e Fourer e Marsten [43], e o capítulo 7 do livro de Golstein e Youdine [51] descrevem em detalhes vários aspectos teóricos ligados a esta visão da PLP.

Deve-se observar que alguns artigos publicados há algum tempo, como os de Golstein [50] e Orden e Nalbandian [88], já propunham algoritmos especializados para modelos de PLP. Além destes autores, vários pesquisadores, tais como Beale [17], Dantzig [30], Ferguson e Dantzig [37], Williams [121] em Programação Linear sob Incerteza, Charret [21] e Rosvany ([99], [100]) em Projeto Estrutural, publicaram descrições de muitas e variadas situações onde modelos de PLP vinham desempenham importante papel.

Uma pesquisa bibliográfica mais ampla acerca deste enfoque para a PLP pode ser encontrada em Marins [70], onde se incluem os temas de várias de teses de mestrado e doutoramento (Fernandes [38], Ferreira [39], Garcia [45], Ribeiro [93], e Souza [107]) desenvolvidas na Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, no período de 1977 a 1979, que abordam a área em questão .

A idéia principal deste último tratamento da PLP é o desenvolvimento de teoria e algoritmos próprios para a PLP, através da extensão natural dos conceitos e definições válidas para a PL. É neste contexto que se enquadra o presente texto, que aborda especificamente uma sub-área da PLP, de muita importância, que será discutida na seção seguinte.

Com respeito a PLP em geral, deve-se, ainda, referenciar o artigo de De Wolf et al. [34] no qual os autores considerando problemas separáveis, onde funções não-lineares também podem aparecer nas restrições, propõem o tratamento destes problemas pela PLP. Eles citam como exemplos de aplicação problemas de transmissão de gás com perdas na tubulação (De Wolf et al. [34]), e outras situações similares, como problemas de distribuição de água com perdas ("head

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

losses"), e problemas de simulação de transmissão de eletricidade usando uma aproximação das equações de fluxo de carga em corrente contínua.

1.1.2. A PROGRAMAÇÃO EM REDES LINEARES POR PARTES

O uso de modelos em redes tem crescido substancialmente nos últimos anos, dado o seu reconhecimento como uma importante ferramenta para problemas associados com a Teoria da Decisão. Como comentado por Glover et al. [47], este fenômeno é devido a três propriedades inerentes aos modelos de redes:

(a) *Fácil Visualização*, de fato as relações relevantes existentes no problema podem ser convenientemente representadas por meio de diagramas (grafos) que permitem uma identificação rápida por parte do usuário do modelo;

(b) *Flexibilidade de Modelo*, como pode ser constado na abundante literatura existente sobre modelos em redes (ver, por exemplo, Glover et al ([47], [48] e Ahuja et al. [4], Mulvey [85], e Veldhorst [119]), estes modelos são extremamente úteis e cobrem uma enorme diversidade de aplicações nas variadas áreas;

(c) *Facilidade de Solução*, devido a suas características particulares, a proposição de novas estruturas de armazenamento de dados extremamente eficientes, e ao surgimento de enfoques alternativos para o desenvolvimento de algoritmos, atualmente existem disponíveis inúmeros métodos de solução especializados para modelos em redes, inclusive para as de grande porte, possibilitando a solução direta de muitos problemas que, até recentemente, não podiam ser resolvidos em tempo razoável.

Esta importância de modelos em redes tem atraído a atenção de pesquisadores que vêm desenvolvendo novos métodos, para a abordagem das muitas classes de modelos em redes conhecidas. Pesquisadores interessados em estudar o comportamento destes tipos de algoritmos, têm realizado testes computacionais para descobrir a eficiência relativa de algoritmos gerais e

unesp

especializados quando aplicados à solução de modelos em redes. Um texto bem atualizado, e interessante, que oferece uma descrição completa de algoritmos e classes de problemas em redes, é o livro publicado por Ahuja et al. [4].

Particularmente na pesquisa aqui relatada, procurou-se dar seqüência ao esforço do autor, na análise de uma situação particular da PLP, iniciado a partir da sua tese de doutoramento, desenvolvida sob a orientação do Prof. Clovis Perin Filho do Instituto de Matemática, Estatística e Ciências da Computação da Universidade Estadual de Campinas, e submetida, em 1987, à aprovação do Programa de Pós-Graduação da Faculdade de Engenharia Elétrica da mesma universidade .

Trata-se do estudo de programas em redes lineares por partes, que têm como meta a minimização de uma função objetivo convexa, separável e linear por partes, sujeita a um sistema de equações lineares que expressam a conservação de fluxo sobre uma matriz de incidência de uma rede conectada, que representa as relações existentes entre as variáveis do problema.

A motivação maior para a continuidade dos estudos em Programação em Redes Lineares por Partes - PRLP, iniciados em 1985, foi a verificação da relevância do tema e que a área ainda demandava maiores desenvolvimentos.

Esta constatação se deu duplamente, tanto por meio do interesse demonstrado por outros pesquisadores, e também profissionais de várias áreas, quando da divulgação dos resultados parciais da pesquisa em eventos científicos, ver Marins e Perin ([71], [72], [73], [74]) e Marins, Perin e Machado ([75], [76]), como por intermédio de pesquisa bibliográfica realizada pelo autor nos últimos dez anos na literatura disponível.

Esse levantamento bibliográfico incluiu a consulta as mais importantes revistas de divulgação científica internacionais, várias teses de autores nacionais e estrangeiros, relatórios técnicos e também inúmeros contatos (pessoais e por correspondências) realizados com pesquisadores de destaque na área.

Ao contrário do que ocorre para modelos de redes lineares, cujo tratamento teórico e estado da arte dos algoritmos disponíveis têm sido extensamente divulgados, podendo-se citar aqui a referência Ahuja et al. [4], que além de ser um excelente livro texto apresenta, ainda, uma

unesp

vasta literatura comentada sobre o assunto, muito pouco tem sido publicado especificamente para a PRLP.

Passa-se a comentar alguns trabalhos de relevância que tentam suprir a lacuna acima apontada.

Monma e Segal [83] descrevem com detalhes um algoritmo simplex para encontrar fluxo de custo mínimo em redes capacitadas que se aplica, também, para o caso de função objetivo linear por partes. O algoritmo é de grande valia para redes de grande porte pelo seu excelente desempenho computacional, bem como possui características importantes que evitam a ocorrência dos indesejáveis fenômenos de ciclagem ("cycling") e empacamento ("stalling"), decorrentes de situações de degenerescência; adicionalmente apresenta-se a estrutura de dados adotada na implementação de tal algoritmo.

Barras et al. [14] apresentam uma formulação e as definições básicas para a PRLP, bem como um algoritmo tipo simplex para redes lineares por partes, comentam, ainda, que detalhes sobre o algoritmo e sua implementação estariam numa outra referência, a qual, infelizmente, por falta de dados blibliográficos, não foi obtida na pesquisa realizada, apesar de, inclusive, ter sido enviada correspondência aos autores com a solicitação do artigo.

Já o trabalho de Sun e Tsai [113] propõe um algoritmo simplex especializado para a PRLP, que é especialmente adequado para problemas com um número grande de termos lineares nas parcelas que compõem a função objetivo. O algoritmo é do tipo Big-M, com um mecanismo auto-adaptativo para escolha do número M, o que é realizado através do uso de uma fórmula empírica que depende fundamentalmente das inclinações mínimas e máximas associadas às funções de custo lineares por partes em cada arco da rede.

Estes últimos autores estabelecem o seu método, analisam sua convergência, descrevem a estrutura de dados usada, e relatam uma comparação computacional do seu algoritmo com o bem conhecido algoritmo NETFLO, descrito em Kennington e Helgason [65]. A aplicação do NETFLO se dá após a reformulação dos modelos de PRLP como modelos em redes lineares equivalentes. Estes modelos dos testes foram gerados pelo programa gerador de redes NETGEN de Klingman et al. [67], que tem características interessantes, e tem sido adotado em muitas pesquisas.

unesp

Na sua tese de doutoramento, Marins [70], o autor apresenta a formulação de programas em redes lineares por partes, a notação adotada, e a fundamentação teórica do algoritmo implementado, que é uma especialização para a PRLP do método simplex para redes lineares proposto por Fourer [40].

Este algoritmo possui características especiais que permitem evitar a ciclagem e o empacamento, e tirar vantagem da estrutura linear por partes do problema para efetuar, numa dada iteração, uma exploração continuada de variável básica (não-básica) que permite a realização da maior mudança possível no valor do fluxo do arco básico (não-básico) a ser substituído (considerado para entrar na base), caso se constate no teste da razão que este arco básico (não-básico) pode permanecer básico (não-básico) na próxima aplicação do método, sem haver mudança da árvore geradora disponível no momento.

Nessa tese de doutoramento, ainda são apresentados resultados computacionais de testes do desempenho do algoritmo, quando combinado com várias regras de entrada de variável na base, incorporando ou não a abordagem pela técnica de mudança de escala, ou escalonamento dos dados ("scaling"), proposta por Edmonds e Karp [36].

Como comentado anteriormente, existem inúmeras aplicações da PRLP, passa-se a seguir a descrever brevemente algumas delas, que justificam a relevância da área.

Técnicas de Programação Matemática têm sido usadas com sucesso em variadas situações no campo de Sistemas de Potência, sendo que, particularmente, com freqüência o uso de modelos de PRLP no auxílio à solução de tais casos tem sido referenciado na literatura.

Nesta linha, Sasson et al. [102] elencam importantes problemas de planejamento e operação de sistemas de potência que podem ser tratados por meio de modelos de PRLP; já Barras et al. [14] apresentam um algoritmo simplex para redes lineares por partes que se aplica ao problema do cálculo de fluxo de carga em corrente alternada e corrente contínua, este problema é de particular interesse, pois se constitui em parte importante de problemas mais gerais de segurança, e na análise de contingência, em sistemas de potência.

Uma outra área fértil na utilização de modelos de PRLP é no planejamento da operação de sistemas hidroelétricos de potência, nesta linha pode-se referenciar os artigos de Dembo et al. [33], Rosenthal [98], e Soares Filho et al. [106], que descrevem para problemas reais resolvidos

unesp

na Espanha, Estados Unidos da América, e Brasil, respectivamente, a etapa de modelagem e a metodologia de solução adotada.

Soares Filho et al. [106] descrevem uma aplicação ao planejamento multiobjetivo de médio prazo da operação de um sistema de multireservatórios. O sistema analisado é o da bacia do rio Piracicaba, no estado de São Paulo. Os objetivos a serem maximizados estão associados a dois usos conflitantes da água da região: suprimento e qualidade. Outros aspectos como produção de energia elétrica e controle de cheias são considerados como limitantes sobre o nível de água armazenada e a descarga necessária de água nos reservatórios. O método aplicado representa este problema, com respeito a três cenários hidrológicos - sêco, úmido e em condições médias, como um problema em rede linear por partes, considerando a estrutura da rede hidraulica associada ao sistema de multireservatórios estendida no tempo (horizonte de planejamento de 01 ano dividido em 12 meses).

A referência seguinte, Rosenthal [98], descreve alguns problemas típicos na administração de sistemas de reservatórios, onde se busca a maximização dos benefícios proprocionados pelo sistema com respeito a várias características, tais como a existência de (a) um ou vários reservatórios; (b) período único ou multiperíodos; (c) fluxo determinístico ou estocástico; e (d) função objetivo linear, não-linear separável e não separável. O artigo também relaciona vários métodos para a solução destes problemas, onde se inclui a aproximação por funções lineares por partes para o caso de função objetivo separável.

Os problemas surgidos na administração da Hidroelétrica Espanhola, uma das duas maiores daquele país, são descritos em Dembo et al. [33]. Na solução destes problemas os autores utilizaram modelos (para problemas com horizonte de planejamento de curto prazo) que possibilitam o uso de técnicas de PRLP.

Também em sistemas de telecomunicações tem naturalmente surgido aplicações da PRLP. Três casos típicos são descritos em Monma e Segal [83]: um problema de transferência de facilidades entre áreas , um problema de programação da coleta de registros de chamadas numa rede privada de comunicação, e um problema de planejamento da expansão de capacidade de sistemas de telecomunicações. Estes autores também propõem um algoritmo eficiente para solucionar modelos em redes lineares por partes. Também França et al. [44] e Nakagawa [86]

unesp

tratam de casos similares, relativos à expansão de redes telefônicas, e alocação de centros de fios, respectivamente.

Outra situação, onde o uso de modelos de PRLP é muita valia, se refere ao problema clássico da análise de redes de tubulações, que busca encontrar o conjunto de fluxos e pressões numa rede de distribuição de água, conhecidos os níveis de suprimento e de demanda de água. A solução deste problema fornece informações básicas para todo o projeto de engenharia do sistema de distribuição de água, Collins et al. [22] relatam um caso deste tipo para a cidade de Dallas nos Estados Unidos da América.

Na área de administração de projetos pode-se citar dois artigos interessantes, Ahuja et al. [3] e Levner e Nemirovsky [69], que abordam, respectivamente, o problema de negociação entre duração do projeto e seu custo ("time-cost tradeoff problem") numa rede CPM, ver Ravindran et al. [92] para uma descrição do caso, e o problema da programação de projetos com objetivo e restrições orientadas por uma política tipo "just-in-time", proposto por Anthonisse et al. [12].

Quanto a sistemas de transportes, a dificuldade está em selecionar exemplos de aplicação da PRLP, devido a enorme quantidade e variedade destas que têm sido reportadas na literatura científica. Passa-se a comentar, sucintamente, quatro tipos de problemas de transporte que se enquadram claramente numa abordagem pela PRLP.

O primeiro deles, tratado em Klein [66], é o do modelo de transporte que admite o custo de planejamento da produção como sendo convexo, respeitando restrições de estoque e com possibilidade de haver ordens pendentes de pedidos.

Os outros dois casos estão associados a existência de incerteza nas demandas, tanto num problema de transporte clássico, como em Cooper e LeBlanc [24], e em Ferguson e Dantzig [37], como num problema que admita nós de transbordo ("transshipment nodes") tal como relatado em Sun et al. [114].

Finalmente, com respeito a outro importante caso dentro do estudo de sistemas de transportes, a publicação de LeBlanc [68] apresenta a utilidade da PRLP para a solução de modelos de equilíbrio em redes de tráfego, admitindo demandas fixas ou variáveis.

Assim, pelo que já foi anteriormente exposto, acredita-se que tenha ficado plenamente justificada a motivação para se continuar o desenvolvimento de pesquisa na área de Programas em

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Redes Lineares por Partes, bem como a importância intrínseca deste esforço para avanços significativos desta área.

Nas duas seções subseqüentes são apresentados os objetivos da pesquisa desenvolvida, e um sumário do conteúdo geral deste trabalho.

1.2. OBJETIVOS

O objetivo principal desta pesquisa foi o de propiciar aos interessados em situações em que modelos de PRLP possam ser úteis, informações acerca de quais algoritmos estão disponíveis para serem aplicados na solução de seus modelos, onde encontrá-los, sua descrição, características, e naturalmente qual o desempenho relativo de cada um destes métodos frente a diferentes categorias de problemas e de perfis de dados.

Assim, inicialmente, buscou-se a implementação de três novos algoritmos especializados para a PRLP, com características diferentes das do método primal simplex fortemente viável já apresentado em Marins [70], e que, eventualmente, pudessem ser opções viáveis e interessantes a este último em determinados casos.

Estabeleceu-se como meta, após consulta a bibliografia disponível, a implementação de um algoritmo do tipo dual simplex baseado no método devido a Ali et al. [6], outro do tipo out-of-kilter que fosse uma extensão do algoritmo apresentado por Kennington e Helgason [65], e também um terceiro que utilizasse a técnica de escalonamento de custos ("cost scaling"), proposta por Goldberg e Tarjan [49].

Neste trabalho objetiva-se, ainda, utilizar um experimento computacional, com rigorosa base estatística, para comparar com respeito a quatro fatores, a seguir descritos, quando considerados individualmente ou em combinações, os quatro algoritmos especializados para modelos de PRLP resultantes da etapa anterior da pesquisa:

(a) Fator **Classe** do problema, sendo admitidos Problemas de Transporte Clássico, Problemas de Transporte com Nós de Transbordo ("Transshipment Problems"), ou Problemas de Circulação;

(b) Fator **Dimensão** do problema, sendo medida pelo número de arcos e nós da rede associada;

(c1) Fator **Proporção de arcos capacitados**, sendo considerados os níveis de capacitação de 100%, 80%, 50%, 20%, e 0%;

(c2) Fator **Número de limitantes de intervalos de fluxo** ("breakpoints") para cada arco, sendo consideradas as situações com 03, 06 e 09 limitantes por arco;

(d) Fator **Método** aplicado, estando incluídos os quatro algoritmos especializados implementados.

Escolheu-se como sendo a variável a ser observada no experimento o tempo de CPU gasto pelos algoritmos na solução de amostras, geradas aleatoriamente, de várias categorias de modelos de PRLP.

Procurava-se, deste modo, finalmente, a obtenção de respostas para as seguintes questões:

(1) Há um algoritmo que é sempre melhor que os demais testados, em qualquer situação? Ou alternativamente, qual algoritmo tem o melhor desempenho em cada categoria de modelos de PRLP considerados?;

(2) Quais são os efeitos dos fatores considerados no desempenho de cada algoritmo?;

(3) Quais são os efeitos de mudanças conjuntas nos fatores estudados no desempenho de cada algoritmo?.

1.3. SUMÁRIO

No capítulo 1 apresenta-se a Programação Linear Partes - PLP em geral, sendo feita uma classificação das pesquisas realizadas segundo o tipo de enfoque adotado, a PLP como caso particular da Programação Linear, a PLP como caso particular da Programação Não-Linear, ou a PLP como uma generalização da Programação Linear. Também apresenta-se uma breve revisão da literatura, bem como algumas de suas áreas de atuação são realçadas. Nesse mesmo capítulo, é introduzida uma sub-área da PLP, denominada Programação em Redes Lineares por Partes, que

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

e´ o assunto estudado neste trabalho. Finalmente, são estabelecidos os objetivos a serem atingidos pela pesquisa.

A formulação de problemas em redes lineares por partes, a notação adotada, os principais resultados gerais, e as descrições dos algoritmos implementados, com suas características e propriedades fundamentais, constam do capítulo 2.

No capítulo 3, são apresentados os detalhes do projeto de experimentos utilizado, com uma justificativa da metodologia aplicada, a descrição do processo de geração de problemas aleatórios usados no teste de desempenho dos algoritmos implementados, bem como o modelo estatístico adotado para a análise de variância realizada. Os resultados computacionais, os resultados dos testes estatísticos efetuados, e uma discussão sobre o comportamento relativo dos algoritmos, também constam do capítulo 3.

No capítulo 4 apresenta-se uma discussão do comportamento relativo dos algoritmos, um sumário das conclusões obtidas, comentários gerais, e direções para prosseguimento da pesquisa. O Anexo 1 se refere ao manual de usuário do sistema NplSys, que descreve com detalhes o sistema implementado para a realização dos testes. Finalmente é apresentada a literatura relevante utilizada.

unesp

CAPÍTULO 2

ALGORITMOS PARA PROGRAMAS EM REDES LINEARES POR PARTES

2.1. INTRODUÇÃO

Neste capítulo são, inicialmente, estabelecidas a terminologia e a notação para funções lineares por partes, as quais serão utilizadas na formulação dos problemas primal e dual em redes lineares por partes. A seguir, são apresentados os resultados válidos para o cálculo das soluções básicas destes dois problemas, e as condições para otimalidade.

Na seqüência do texto são descritos os quatro algoritmos implementados, primal simplex, método dual, out-of-kilter, e cost scaling, que são especializados para programas em redes lineares por partes, e algumas de suas propriedades são discutidas.

2.2. PROGRAMAS EM REDES LINEARES POR PARTES

Uma função de várias variáveis $C(x_1, x_2, ..., x_n)$ é dita ser *separável* se ela pode ser representada como sendo uma soma de funções dependentes de uma única variável, isto é, $C(x_1, x_2, ..., x_n) = C_1(x_1) + C_2(x_2) + ... + C_n(x_n)$.

Uma função de uma variável $C_j(x_j)$ é do tipo *convexa e linear por partes* se ela é linear sobre cada um de uma série de intervalos consecutivos $[d_j^k, d_j^{k+1}]$, com coeficiente de custo c_j^k associado a cada intervalo, e considerando, ainda, que estes coeficientes de custo c_j^k e limitantes de intervalos ("breakpoints") d_j^k formam *seqüências monotonicamente crescentes* para cada

variável x_j:

$$c_j^1 < c_j^2 < \ldots < c_j^k < c_j^{k+1} < \ldots < c_j^{m(j)} \quad \text{e} \quad d_j^1 < d_j^2 < \ldots < d_j^k < d_j^{k+1} < \ldots < d_j^{m(j)}.$$

A Figura 1, abaixo, ilustra uma função convexa linear por partes típica, sendo que maiores detalhes sobre propriedades fundamentais de funções convexas, separáveis e lineares por partes, podem ser encontrados em Fourer [39].

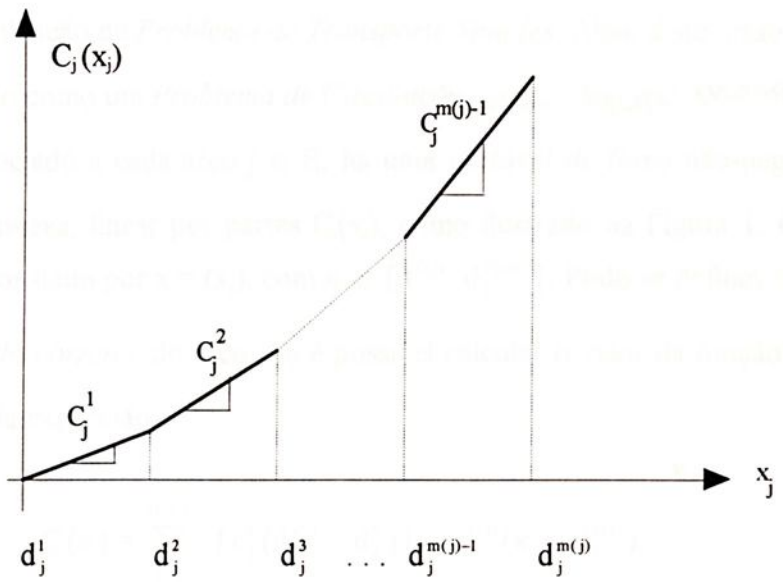


Figura 1. Função convexa linear por partes.

A idéia central deste trabalho é efetuar uma abordagem para a Programação em Redes Lineares por Partes - PRLP através da generalização da teoria existente (ver Ahuja et al. [4]) para o *problema de fluxo com custo mínimo* da Programação em Redes Lineares.

Seja G = (N, E) uma *rede orientada conectada*, onde N é um conjunto com | N | = n nós, e E é um conjunto de | E | = m arcos. Cada arco $j \in E$ é um par ordenado (t(j), h(j)) de nós distintos com t(j), h(j) $\in$ N, sendo denominados, respectivamente, o nó *cauda* e o nó *cabeça* do arco j.

Denote por $A = \{a_{ij}\}$ a *matriz de incidência nó-arco*, de ordem (n x m), desta rede G, tal que, para todos os arcos $j \in E$, têm-se: $a_{t(j),j} = -1$, $a_{h(j),j} = +1$, e demais elementos da coluna j da matriz A nulos.

Considere que o valor da *demanda* existente no nó $i \in N$ seja dado por b_i. Neste caso, o nó i é classificado como sendo um nó *origem*, um nó *destino*, ou um nó de *transbordo* ou de

baldeação ("transshipment") se $b_i < 0$, $b_i > 0$, ou $b_i = 0$, respectivamente. Suponha, ainda, que a soma das demandas para todos os nós $i \in N$ seja zero.

Esta estrutura de rede admite as três classes de problemas considerados neste trabalho. Sejam S_1 , S_2 , S_3 , os subconjuntos de nós origens, nós destinos e nós de transbordo, respectivamente, satisfazendo $S_1 \cup S_2 \cup S_3 = N$. Em geral, chama-se tal tipo de problema em rede como sendo um *Problema de Transporte com Baldeação* ("Transshipment problem"). Porém, se $S_3 = \varnothing$ ele recebe a denominação de *Problema de Transporte Simples*. Além disto, caso $S_1 = S_2 = \varnothing$, o problema é conhecido como um *Problema de Circulação*. ← FALTOU (VERIFIC. SOBRAS)

Admita que, associado a cada arco $j \in E$, há uma *variável de fluxo* não-negativa x_j, e uma função de custos convexa, linear por partes $C_j(x_j)$, como ilustrado na Figura 1. Considere, também, um vetor de fluxos dado por $x = (x_j)$, com $x_j \in [d_j^{f(j)}, d_j^{f(j)+1}]$. Pode-se definir, neste caso, f(j) como sendo o *intervalo corrente* do arco j, e é possível calcular o valor da função de custos associada, $C_j(x_j)$, através da expressão:

$$C_j(x_j) = \sum_{k=1}^{f(j)-1} [c_j^k (d_j^{k+1} - d_j^k)] + c_j^{f(j)} (x_j - d_j^{f(j)}).$$

Um problema *primal* em PRLP pode ser formulado como se segue:

min C(x)

sujeito a: Ax = b (***Problema P***)

$x \geq 0$,

onde $C(x) = \sum_{j \in E} C_j(x_j)$, o vetor de fluxos x, o vetor de demandas b, e a matriz de incidência A são como definidos anteriormente.

Como citado no capítulo anterior, este problema pode ser transformado num problema linear equivalente de maior porte. Basta criar, para cada arco $j \in E$, identificado pelo par de nós distintos (t(j), h(j)), no multigrafo resultante da transformação, tantos arcos paralelos entre os nós t(j) e h(j) quantos sejam em $C_j(x_j)$ o número de segmentos lineares, com limitantes inferior e superior de fluxo adequados. Como os coeficientes de custo c_j^k são admitidos como sendo monotonicamente crescentes, o problema linear equivalente, resultante da transformação usada,

unesp

torna-se um problema de fluxo com custo linear padrão, podendo ser resolvido por métodos convencionais. Contudo, Marins [70] mostrou que a eficiência de tal procedimento pode ser melhorada pela resolução direta do problema de PRLP.

Como apontado e discutido por Fourer [39], de um modo geral intercambiando-se os papéis dos 'breakpoints" d_j^k, e correspondentes coeficientes de custos c_j^k, na função $C_j(x_j)$, pode-se obter outra função convexa e linear por partes, $D_j(z_j)$ que é a *função dual* (ou conjugada) de $C_j(x_j)$. Esta função dual $D_j(z_j)$ terá inclinação d_j^{k+1} no intervalo $[c_j^k, c_j^{k+1}]$ e pode ser calculada pela expressão abaixo:

$$D_j(z_j) = \sum_{k=1}^{f(j)-1} [\, d_j^{k+1}(c_j^{k+1} - c_j^k)\,] + d_j^{f(j)+1}(z_j - c_j^{f(j)}).$$

Assim, pode-se formular o problema *dual* de um programa em redes lineares por partes como sendo:

max b'y - D(z)

sujeito a: A'y - z ≤ 0 ***(Problema D)***

$z_j \geq c_j^1$ para $j \in E$,

onde $D(z) = \sum_{j \in E} D_j(z_j)$.

Neste trabalho, os vetores de variáveis duais y e z, da formulação acima, são denominados como sendo o vetor de *preços duais* nos nós e vetor de *custos correntes* nos arcos, respectivamente.

Fourer [39] demonstrou, ainda, que devido à relação dual que existe entre as funções $C_j(x_j)$ e $D_j(z_j)$, tem-se que se, no primal, o fluxo no arco j é tal que $x_j \in [d_j^{f(j)}, d_j^{f(j)+1}]$ então, no dual, tem-se que $z_j \in [c_j^{f(j)}, c_j^{f(j)+1}]$.

A seguir apresentam-se algumas definições importantes para a PRLP, as quais são extensões daquelas válidas para a Programação em Redes Lineares (ver Ahuja et al. [4]).

Uma *base de trabalho* para um problema em PRLP é definida por uma *árvore geradora* ("spanning tree"), T, com (n-1) *arcos básicos*, um nó especificado como sendo a *raiz* de T, e um vetor de intervalos correntes f = (f(j)).

Para cada arco básico $j \in T$, defina R(T, j) como sendo o subconjunto de nós na subárvore que contém a raiz de T, obtida quando o arco (t(j), h(j)) é removido. Deste modo, um

unesp

arco básico $j \in T$ pode ser classificado como se segue:

- um arco básico j é *direcionado a partir da raiz* ("down") se o nó cabeça $t(j) \in R(T, j)$,
- um arco básico j é *direcionado para a raiz* ("up") se o nó cauda $h(j) \in R(T, j)$.

Defina, ainda, para todo arco básico $j \in T$ dois subconjuntos de *arcos não-básicos* incidentes aos nós de R(T, j):

- $TAIL(T, j) = \{k \notin T: t(k) \in R(T, j) \text{ e } h(k) \notin R(T, j)\}$;
- $HEAD(T, j) = \{k \notin T: t(k) \notin R(T, j) \text{ e } h(k) \in R(T, j)\}$.

Finalmente, defina para cada arco $j \in E$ um valor p_j tal que:

- $p_j = 0$ se $j \notin T$,
- $p_j = +1$ se $t(j) \in R(T, j)$,
- $p_j = -1$ se $h(j) \in R(T, j)$.

Denote por P_i o único *caminho* ("path") existente em T da raiz até o nó i. Assim, dada uma base e um vetor de custos correntes f(j) para o *Problema P* acima, a única *solução básica* {x, y, z} associada pode ser calculada por:

- $y_i = 0$, se i é a raiz de T,
- $y_i = \sum_{j \in T} [c_j^{f(j)} \mid j \text{ é arco "up" em } P_i] - \sum_{j \in T} [c_j^{f(j)} \mid j \text{ é arco "down" em } P_i]$, se $i \neq$ raiz,
- $z_j = c_j^{f(j)}$, se $j \in T$,
- $z_j = \max \{ c_j^1 , y_{h(j)} - y_{t(j)} \}$, se $j \notin T$,
- $x_j = d_j^{f(j)}$ se $j \notin T$,
- $x_j = p_j \left(\sum_{i \notin R} b_i + \sum_{k \in TAIL} x_k - \sum_{k \in HEAD} x_k \right)$ se $j \in T$

onde T é a árvore geradora correspondente a base, e, respectivamente, R, TAIL, e HEAD denotam os conjuntos R(T, j), TAIL(T, j), e HEAD(T, j). Esta solução básica é *primal viável* se $x_j \in [d_j^{f(j)}, d_j^{f(j)+1}]$, para todo j.

Observe-se que esta solução básica, assim computada, satisfaz as Condições de Folgas Complementares válidas para a PRLP (ver Marins [70]), as quais, supondo para todo arco $j \in E$ coeficientes de custo dados por $c_j^{f(j)-1} = -\infty$, caso não estejam definidos, podem ser expressas por:

$$d_j^{f(j)} < x_j < d_j^{f(j)+1} \quad \Rightarrow \quad y_{h(j)} - y_{t(j)} = c_j^{f(j)} \tag{1}$$

$$c_j^{f(j)-1} < y_{h(j)} - y_{t(j)} < c_j^{f(j)} \quad \Rightarrow \quad x_j = d_j^{f(j)} \tag{2}$$

$$z_j = \max \{ c_j^1, y_{h(j)} - y_{t(j)} \} \tag{3}$$

Note-se, ainda, que se $x_j \in [d_j^{f(j)}, d_j^{f(j)+1}]$ para $j \in T$, o vetor de fluxos nos arcos $x = (x_j)$ é viável para o *Problema P*, e os vetores de variáveis duais $y = (y_i)$, e $z = (z_j)$ são sempre viáveis para o *Problema D* associado. Com o auxílio do vetor de preços duais y, pode-se definir, para todo arco $j \in E$, dois tipos de coeficientes de custo reduzido, que avaliam a influência, no valor da função objetivo do *Problema P*, de modificações no atual valor da variável de fluxo x_j :

- c_j^-, que é denominado *coeficiente de custo reduzido à esquerda*, está relacionado com reduções no atual valor da variável de fluxo x_j,
- c_j^+, que é denominado *coeficiente de custo reduzido à direita*, relacionado com aumentos no atual valor da variável de fluxo x_j.

Para uma dada solução básica {x, y, z}, estes coeficientes de custo reduzido podem ser calculados através de:

- $c_j^- = c_j^+ = 0$, para $j \in T$ (4)
- $c_j^- = c_j^{f(j)-1} - y_{h(j)} + y_{t(j)}$, e $c_j^+ = c_j^{f(j)} - y_{h(j)} + y_{t(j)}$, para $j \notin T$ (5)

Analogamente ao que ocorre nos programas em redes lineares canalizadas, o valor da função objetivo C(x) decrescerá, como é de interesse, nos seguintes casos:

- $c_j^+ < 0$ e for aumentado o valor do fluxo x_j no arco j,
- $c_j^- > 0$ e for diminuído o valor do fluxo x_j no arco j.

Pode ser demonstrado (ver Fourer [39]) que uma dada solução básica viável {x, y, z}, associada a uma árvore geradora T e um vetor de custos correntes f, é solução ótima se para todo arco não-básico $s \notin T$, tem-se $c_s^- \leq 0$ ou $c_s^+ \geq 0$. Observe-se, que não é possível ocorrer simultaneamente as condições $c_s^- \leq 0$ e $c_s^+ \geq 0$ pois, sempre, tem-se $c_s^{f(s)-1} < c_s^{f(s)}$, devido a hipótese de convexidade admitida para a função objetivo separável e linear por partes C(x).

unesp

2.3. ALGORITMOS ESPECIALIZADOS

Descreve-se abaixo os principais passos para a aplicação, e características dos seguintes algoritmos especializados para a PRLP: Primal Simplex, Método Dual, Out-of-Kilter e Cost-Scaling. Adicionalmente, são apresentadas provas de convergências de cada algoritmo.

2.3.1 ALGORITMO PRIMAL SIMPLEX

O método descrito nesta seção é uma especialização para redes do algoritmo proposto por Fourer [39] para a PLP em geral, um maior detalhamento sobre várias de suas características e propriedades podem ser encontradas em Marins ([70]).

Além dos conceitos vistos na seção anterior, faz-se necessário para a apresentação dos passos do algoritmo o conceito de *ciclo de arcos* na rede. Da teoria de Programação em Redes Lineares sabe-se que, o acréscimo de um arco não-básico s à árvore geradora T, associada a uma base, gera um ciclo $Q_s = Q_s^+ \cup Q_s^-$ em $T \cup \{s\}$, definido por $Q_s^+ = \{j \in Q_s |$ o arco j, e o arco s têm a mesma orientação em $Q_s\}$ e $Q_s^- = Q_s \setminus Q_s^+$.

Os passos básicos do algoritmo podem ser estabelecidos como se segue. Por uma questão de notação, considere $\hat{c}_s = c_s^+$ se $c_s^+ < 0$; $\hat{c}_s = c_s^-$ se $c_s^- > 0$, e $\hat{c}_s = 0$ caso contrário. Observe-se que valores de $\hat{c}_s$ não nulos estão associados a fluxos em arcos não-básicos candidatos a melhorar a solução corrente. Sejam, respectivamente, T e f = (f(j)), a árvore geradora e um vetor de intervalos correntes nos arcos, associados com uma solução básica inicial {x, y, z}.

Algoritmo

Seja {x, y, z} uma solução básica viável associada à uma árvore geradora T e um vetor de intervalos correntes f.

1. Selecione um arco não-básico $s \notin T$, tal que $\hat{c}_s \neq 0$, para entrar na base.

Se não existe tal arco então Pare: Solução Ótima.

unesp

Caso contrário faça $s_0 \leftarrow s$ e obtenha o ciclo $Q = Q^+ \cup Q^-$ formado em $T \cup \{s_0\}$.

2. Efetue o teste da razão no ciclo Q; isto é,

se $\hat{c}_s < 0$

então faça

$$\delta_0 \leftarrow d_s^{f(s)+1} - d_s^{f(s)},$$

$$\delta_1 \leftarrow \min\{ d_j^{f(j)+1} - x_j : j \in Q^+ \setminus \{s_0\}\},$$

$$\delta_2 \leftarrow \min\{x_j - d_j^{f(j)} : j \in Q^- \setminus \{s_0\}\}.$$

caso contrário faça

$$\delta_0 \leftarrow d_s^{f(s)} - d_s^{f(s)-1},$$

$$\delta_1 \leftarrow \min\{ x_j - d_j^{f(j)} : j \in Q^+ \setminus \{s_0\}\},$$

$$\delta_2 \leftarrow \min\{ d_j^{f(j)+1} - x_j : j \in Q^- \setminus \{s_0\}\}.$$

Obtenha $\delta \leftarrow \min \{ \delta_0, \delta_1, \delta_2 \}$.

Selecione um arco $r \in Q$, que minimiza δ, para deixar a base.

Se não houver tal arco então Pare: Solução Ilimitada.

Se $\hat{c}_s < 0$ então faça $f(s) \leftarrow f(s) - 1$.

Atualize o vetor de fluxos x (os arcos no ciclo Q), e atualize o vetor de preços duais y.

Se $\hat{c}_s \neq 0$ então faça $s \leftarrow r$ e Vá ao Passo 2.

Se $r \neq s_0$ então atualize a árvore geradora T para $T \cup \{r\} \setminus \{s_0\}$.

Vá ao Passo 1.

Com respeito a inicialização deste algoritmo, pode-se evitar o uso de fase 1 para a obtenção de uma solução básica viável inicial. Basta considerar uma árvore geradora qualquer, aplicar as expressões para cálculo dos valores das variáveis primal e dual, vistas na seção anterior, acrescentar intervalos adicionais para cada arco básico que esteja com fluxo negativo, com coeficiente de custo $c = -\infty$ e limitante de intervalo $d = -\infty$, e aplicar o próprio método imediatamente.

Tendo em vista a implementação utilizada, que contempla a *exploração continuada de variável básica e não-básica* (conforme informado no capítulo 1), no contexto da PRLP define-se como sendo uma operação de *pivoteamento* cada atualização do vetor de fluxos x, e define-se

como sendo uma *iteração* cada vez que um arco não-básico é escolhido para entrar na base.

Adicionalmente, a versão implementada do algoritmo utiliza o conceito de *bases fortemente viáveis* para a PRLP, conforme definição abaixo, que é uma extensão do mesmo conceito introduzido por Cunningham ([27], [28]) e Barr et al. [13] para redes lineares. Uma interpretação de tal tipo de solução seria a possibilidade de enviar uma quantidade positiva de fluxo da raiz a qualquer nó da árvore geradora sem violação de qualquer limitante de fluxo nos arcos. Neste caso o algoritmo é denominado *algoritmo simplex fortemente viável* ("strongly feasible simplex algorithm") para a PRLP (Marins e Perin [71]).

Um arco básico j é denominado *degenerado* se o valor da variável de fluxo x_j correspondente iguala o valor de um dos "breakpoints", isto é, $x_j = d_j^k$ para algum intervalo k.

Dado um vetor de fluxos básico x, e uma árvore geradora associada T, x é denominado *fortemente viável* ("strongly feasible") se, para todo arco básico degenerado $j \in T$ tem-se:

- se $x_j = d_j^{f(j)+1}$ então o arco j é direcionado para a raiz em T,
- se $x_j = d_j^{f(j)}$ então o arco j é direcionado a partir da raiz em T.

Um vetor de fluxos básico fortemente viável inicial x, e um vetor de custos correntes f associado, podem ser obtidos a partir de uma árvore geradora viável qualquer aumentando, ou decrescendo, do valor 1, os atuais valores de intervalo corrente f(j) de cada arco básico degenerado j que não satisfaça a propriedade de viabilidade forte formulada acima. Observe-se, além disto, que este artifício altera os vetores duais y e z.

Em resumo, o algoritmo implementado tem a propriedade de evitar os fenômenos indesejáveis de *ciclagem* ("cycling") e *empacamento* ("stalling"), e converge sempre como demonstrado a seguir. Ciclagem pode ser definida como sendo uma *seqüência cíclica de pivoteamentos degenerados*, e empacamento pode ser definido como sendo uma *seqüência finita de pivoteamentos degenerados com comprimento exponencial* com relação a dimensão do problema. Um tratamento completo destas situações de ciclagem e empacamento para redes lineares pode ser encontrada em Cunningham ([27], [28]) e Maros [79].

A ciclagem é evitada mantendo na aplicação do algoritmo soluções básicas fortemente viáveis, através da escolha adequada da variável que deve ser substituída na base, como se segue: Seja s o arco considerado para entrar na base, e seja w o último nó comum (mais perto da raiz)

unesp

aos dois caminhos, na árvore geradora T, da raiz aos nós cauda t(s) e cabeça h(s) do arco s. Considere, agora, o ciclo Q formado em $T \cup \{s\}$. Percorra este ciclo Q, partindo do nó w com a mesma orientação de s se $\hat{c}_s < 0$, ou com a orientação contrária de s se $\hat{c}_s > 0$. Escolha como sendo o arco r que deve deixar a base, o primeiro arco do ciclo Q que satisfaça o teste da razão.

Já o empacamento é evitado utilizando-se a viabilidade forte em conjunto com uma regra adequada de escolha da variável considerada para entrar na base. Existem várias regras de entrada com as características desejadas, sendo que um estudo computacional, no contexto da PRLP, envolvendo algumas delas pode ser visto em Marins [70].

O algoritmo implementado adota a regra *Cíclica* ("LRC - Least Recently Considered rule"), que pode ser descrita como se segue: Seja 1, 2,..., m uma ordenação arbitrária, pré-fixada, dos arcos da rede. Suponha que o arco j foi o arco que entrou na base na última iteração. Na próxima iteração, selecione como o arco s a ser considerado para entrar na base, aquele que for o primeiro arco candidato ($\hat{c}_s \neq 0$) da seqüência de arcos j+1, j+2,..., m, 1,..., j.

Passa-se, agora, a apresentar as provas de convergência do algoritmo. Considere um problema em redes lineares por partes não-degenerado, P^*, obtido do *Problema P* através da introdução de uma *perturbação* nas demandas existentes nos nós da rede, como mostrado a seguir. Considere

$$\hat{\varepsilon} = \min \left\{ \left| \sum_{i \notin R} b_i + \sum_{k \in TAIL} d_k^{f(k)} - \sum_{k \in HEAD} d_k^{f(k)} - d_j^{f(j)} \right| \right\}$$

onde a minimização é efetuada sobre todas as possíveis árvores geradoras T, todos arcos básicos $j \in T$, e todos as possíveis combinações de valores de f(j) e f(k).

Dado ε, suficientemente pequeno, satisfazendo $0 < \varepsilon < \hat{\varepsilon}/n$, considere um *vetor de demandas perturbadas* nos nós definido por $b^* = \left(b_i^*\right)$, com $b_i^* = b_i + \varepsilon$ se o nó i não é a raiz, e $b_i^* = b_i - (n-1).\varepsilon$ se o nó i é a raiz. Defina o *problema não-degenerado perturbado* P^* como sendo:

$$\begin{aligned} &\min C(x) \\ &\text{sujeito a: } Ax = b^* \qquad \textbf{(Problema } P^*\textbf{)} \\ &x \geq 0. \\ &\text{onde } C(x) = \sum_{j \in E} C_j(x_j). \end{aligned}$$

Teorema 1. Seja x um vetor de fluxo básico associado a árvore geradora T. Então, o vetor x é viável para o *Problema* P^* se e somente se o vetor x é fortemente viável para o *Problema P*.

Prova: Sejam x e x′ os vetores de fluxo associados a mesma árvore geradora T em P e P^*, respectivamente. Portanto,

(i) para os arcos não-básicos $j \notin T$:

$x_j = d_j^{f(j)}$ and $x'_j = d_j^{f(j)} \Rightarrow x'_j - x_j = 0.$

(ii) para os arcos básicos $j \in T$:

$$x_j = p_j.\left(\sum_{i \notin R} b_i + \sum_{k \in TAIL} d_k^{f(k)} - \sum_{k \in HEAD} d_k^{f(k)}\right) \text{ e } x'_j = p_j.\left(\sum_{i \notin R} b_i^* + \sum_{k \in TAIL} d_k^{f(k)} - \sum_{k \in HEAD} d_k^{f(k)}\right)$$

$\Rightarrow x'_j - x_j = \varepsilon\, p_j\,.\,(n - |R|)$ or $0 < p_j\,.\left(x'_j - x_j\right) < \hat{\varepsilon}$, onde $|R|$ é a *cardinalidade* do conjunto R(T, j).

Em resumo, para um arco j qualquer, tem-se:

$x_j < 0 \Rightarrow x'_j < 0$,

$d_j^{f(j)} < x_j < d_j^{f(j)+1} \Rightarrow d_j^{f(j)} < x'_j < d_j^{f(j)+1}$,

$x_j = d_j^{f(j)}$ com $p_j = +1 \Rightarrow d_j^{f(j)} < x'_j$,

$x_j = d_j^{f(j)+1}$ com $p_j = -1 \Rightarrow x'_j < d_j^{f(j)+1}$.

(c.q.d.)

Teorema 2. O algoritmo primal simplex fortemente viável para a PRLP é finito.

Prova: Frente a pivoteamentos não-degenerados, o valor da função objetivo decresce estritamente, como no caso do método simplex para a programação em redes lineares. Aqui será demonstrado que, para pivoteamento degenerados, embora o valor da função objetivo permaneça o mesmo, o valor de $\sum_{i \in N} y_i$ decresce estritamente.

No Passo 2 do algoritmo suponha que $\hat{c}_s < 0$. Nesta situação o arco s considerado para entrar deve ter seu fluxo x_s aumentado para ocorrer a melhoria do valor da função objetivo. Como se trata de um pivoteamento degenerado, o arco s permanece com o valor de seu fluxo igual ao valor do limitante inferior do intervalo associado a f(s), e se torna um arco direcionado da raiz na próxima árvore geradora.

Porém, todo nó i que tenha no caminho dele à raiz este arco s, terá o valor do preço dual

associado diminuído de $|\ \hat{c}_s\ |$; nos demais casos os valores dos preços duais permanecem os mesmos anteriores. Portanto, $\sum_{i \in N} y_i$ decresce estritamente. Mais ainda, o arco r que deixa a base é diferente daquele arco s considerado para entrar, e é o primeiro arco no caminho de h(s) para o nó w que satisfaz o teste da razão. A análise para o caso $\hat{c}_s > 0$ é análoga.

Observe que uma iteração degenerada tem no máximo $|Q_s|$ pivoteamentos, onde $|Q_s|$ é o número de arcos no ciclo Q_s criado pelo acréscimo do arco s à árvore geradora T. Como o número de soluções viáveis do sistema Ax = b é finito, e todo pivoteamento termina com valores lexicograficamente decrescentes para os pares $\left(C(x), \sum_{i \in N} y_i\right)$, o algoritmo é finito.

(c.q.d.)

2.3.2. ALGORITMO DUAL SIMPLEX

O algoritmo descrito nesta seção é baseado no método dual sobre um grafo, proposto por Ali et al. [6]. Antes de se descrever o algoritmo, são discutidas características especiais do enfoque aplicado, que justificam a implementação adotada, bem como é feita a formalização de alguns conceitos gerais para a PRLP, com respeito a métodos duais.

Em geral, uma solução básica mantida por um método dual satisfaz todas as *restrições de balanço nos nós*, mas os fluxos x_j podem violar as *restrições de canalização* para algum arco j, caracterizando o que se denomina *inviabilidade primal*. Como, por construção, os arcos não-básicos têm valores de fluxo colocados iguais a algum valor de seus respectivos limitantes de intervalos, a referida inviabilidade primal só ocorre com respeito aos fluxos em arcos básicos, isto é, para algum arco básico, com intervalo corrente f(j), tem-se $x_j \notin [d_j^{f(j)}, d_j^{f(j)+1}]$.

A metodologia dual, na resolução do *Problema P*, tenta tornar viáveis os fluxos que não respeitam as restrições de canalização, identificando ciclos, contendo algum destes arcos com problemas, que permitem o envio de fluxo extra possibilitando a diminuição da inviabilidade primal existente; o término do algoritmo se dá quando não há mais fluxos inviáveis com respeito as canalizações nos arcos.

Neste processo, é realizado o chamado *pivoteamento dual*, que, em contraste com o *pivoteamento primal* usado pelo método primal simplex já apresentado, primeiro seleciona uma

unesp

variável (básica) para deixar a base e depois identifica a variável (não-básica) que irá substituí-la.

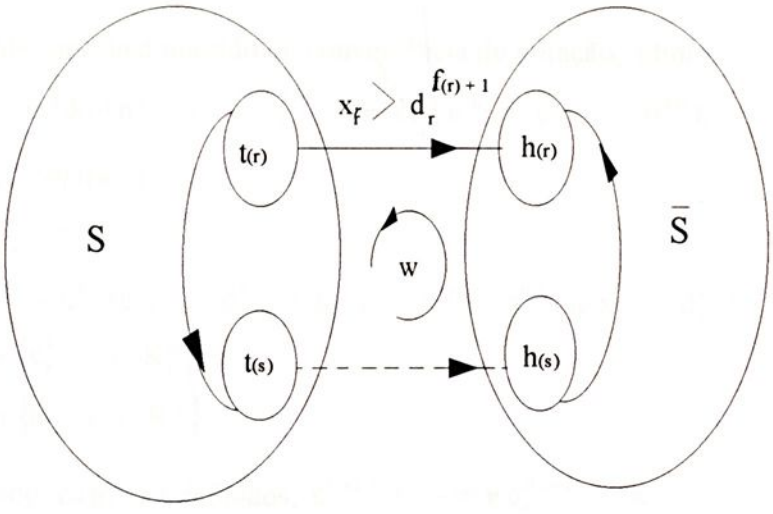
O conjunto de arcos básicos com inviabilidade primal, é denotado por $I = \{r|\ r$ é um arco básico com $x_r \notin [d_r^{f(r)}, d_r^{f(r)+1}]\}$, f(r) sendo o intervalo corrente do arco r. Na aplicação do algoritmo, todo arco $r \in I$ é um candidato a deixar a base; portanto quando $I = \varnothing$ uma solução ótima foi obtida.

Com a finalidade de proporcionar uma melhor compreensão do algoritmo descrito adiante, passa-se a analisar mais de perto a metodologia dual utilizada.

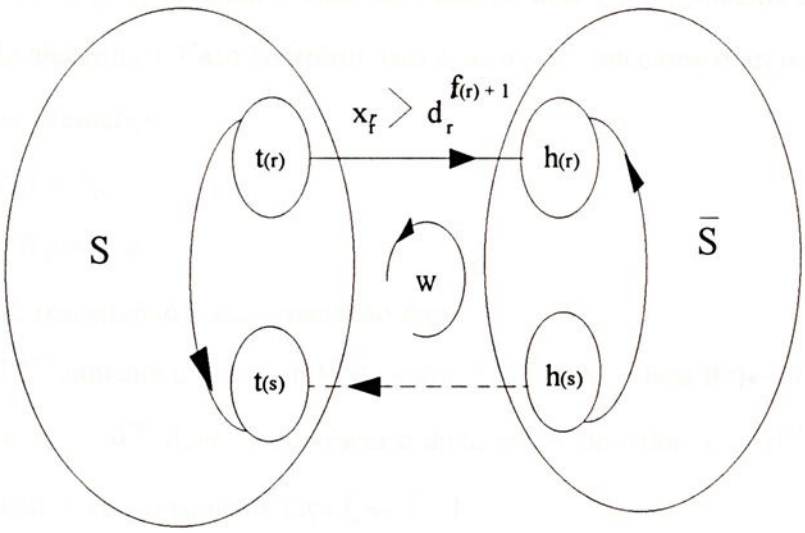
Como no método primal simplex, em cada iteração do método dual simplex mantém-se uma solução básica, com uma árvore geradora T e um vetor de intervalos correntes f associados. A remoção de um arco básico de T cria duas subárvores e uma *partição dos nós* associada, denotada por $(\overline{S}_r, S_r)$, onde $S_r = \{i \in N: r \in P_i\}$, com P_i sendo o caminho em T que conecta a raiz ao nó i, e $\overline{S}_r = N / S_r$.

Pode-se, assim, neste contexto, considerar um *corte* K_r associado ao arco básico r , denotado por $K_r = K^+_r \cup K^-_r$, sendo que $K^+_r = \{k \in E|\ t(k) \in S_r, h(k) \notin S_r\}$ é definido como o conjunto de arcos "*forward*" no corte, e $K^-_r = \{k \in E|\ t(k) \notin S_r, h(k) \in S_r\}$ é definido como o conjunto de arcos "*backward*" no corte. Observe-se que cada arco k em K_r, exceto o arco r, tem o seu valor de fluxo dado por $d_k^{f(k)}$ ou $d_k^{f(k)+1}$, onde f(k) é o intervalo corrente do arco k.

Suponha que o arco $r \in I$ escolhido para deixar a base tem fluxo $x_r > d_r^{f(r)+1}$ (o caso $x_r < d_r^{f(r)}$ é similar); nesta situação, para tornar o fluxo em r viável deve-se diminuir o seu valor x_r. Mostra-se, a seguir, como proceder a escolha do arco s que substituirá o arco r na base, de maneira a diminuir a inviabilidade primal da solução básica corrente.



(a) O arco $s \in K_r^+$



(b) O arco $s \in K_r^-$

Figura 2. Possíveis situações no ciclo W.

Adicionando-se à árvore geradora T um arco $s \notin T$ do corte K_r cria-se um único ciclo, denotado aqui por W, que contém o arco r candidato a sair da base. Para cada arco $s \in K_r$, sendo um arco "forward" ou um arco "backward", tem-se $x_s = d_s^{f(s)}$. Defina a orientação do ciclo W como sendo a orientação oposta daquela do arco r se $x_r > d_r^{f(r)+1}$ (note-se que para a situação em que $x_r < d_r^{f(r)}$ a orientação adotada para W é a mesma do arco r). A seguir é mostrado como o algoritmo trabalha nesta situação, buscando o uso do ciclo W para tornar viável o fluxo no arco r, nas duas situações ilustradas na Figura 2, que é análoga à Figura 11.21 apresentada por Ahuja et al. [4].

unesp

Inicialmente, por uma questão de conveniência de notação, admita:

$\alpha = +1$ se r é "down" e $x_r > d_r^{f(r)+1}$ (ou se r é "up" e $x_r < d_r^{f(r)}$),

$\alpha = -1$ caso contrário.

Além disto, calcule,

$\delta_0 = c_r^{f(r)+1} - c_r^{f(r)}$ se $x_r > d_r^{f(r)+1}$ (ou $\delta_0 = c_r^{f(r)} - c_r^{f(r)-1}$ se $x_r < d_r^{f(r)}$)

$\delta_1 = \min \left\{ c_s^+ : s \in K_r^\alpha \right\}$

$\delta_2 = \min \left\{ \left| c_s^- \right| : s \in K_r^\alpha \right\}$

considerando, caso não definidos, $c_s^{f(s)-1} = -\infty$ e $c_s^{f(s)+1} = \infty$.

Determine $\delta = \text{Min} \{\delta_0, \delta_1, \delta_2\}$.

Se $\delta = \infty$ não se pode diminuir o valor do fluxo no arco r e o problema é primal inviável (fim da execução do algoritmo). Caso contrário, isto é, se $\delta < \infty$, selecione o arco que minimiza δ, e efetue as seguintes alterações:

$y_i \leftarrow y_i$ para $i \notin S_r$,

$y_i \leftarrow y_i + \alpha.\delta$ para $i \in S_r$.

No ciclo W, respeitando a sua orientação faça:

se $x_r > d_r^{f(r)+1}$ aumente o fluxo em W do valor $d_r^{f(r)+1} - x_r$ e faça $f(r) \leftarrow f(r) + 1$,

(no caso de $x_r < d_r^{f(r)}$ deve-se decrescer o fluxo em W do valor $x_r - d_r^{f(r)}$).

se $s \in K_r^-$ então, adicionalmente faça $f_s \leftarrow f_s - 1$.

Com estas operações se dá a diminuição da inviabilidade primal da solução pois, agora, x_r corresponde ao valor de um limitante de intervalo de fluxo do arco r associado ao intervalo corrente f(r).

Descreve-se, a seguir, o algoritmo completo (ver Marins e Perin ([73], [74])) que incorpora a metodologia descrita acima para as demais situações possíveis. Sejam T, e f, respectivamente, uma árvore geradora e um vetor de intervalos correntes associados com uma solução básica dual viável {x, y, z}. Assuma que $c_j^{f(j)-1} = d_j^{f(j)-1} = -\infty$, e $c_j^{f(j)+1} = d_j^{f(j)+1} = +\infty$, caso não sejam definidos.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Algoritmo

1. Selecione um arco básico $r \in I$ para deixar a base, com $x_r < d_r^{f(r)}$ ou $x_r > d_r^{f(r)+1}$.

Se não houver tal arco então Pare: Solução Ótima foi obtida.

Caso contrário, obtenha o corte K_r associado ao arco r.

2. Realize o teste da razão no corte K_r.

Se $x_r > d_r^{f(r)+1}$ então faça $\delta_0 \leftarrow c_r^{f(r)+1} - y_{h(r)} + y_{t(r)}$ e $f(r) \leftarrow f(r) + 1$.

caso contrário faça $\delta_0 \leftarrow c_r^{f(r)-1} - y_{h(r)} + y_{t(r)}$.

Se ($x_r > d_r^{f(r)+1}$ e o arco r é "up") ou ($x_r < d_r^{f(r)}$ e o arco r é "down")

[caso 1] então faça

$\delta_1 \leftarrow \min \{+(c_j^{f(j)} - y_{h(j)} + y_{t(j)}) : j \in K_r^+\}$,

$\delta_2 \leftarrow \min \{-(c_j^{f(j)-1} - y_{h(j)} + y_{t(j)}) : j \in K_r^-\}$.

[caso 2] caso contrário faça

$\delta_1 \leftarrow \min \{-(c_j^{f(j)-1} - y_{h(j)} + y_{t(j)}) : j \in K_r^+\}$,

$\delta_2 \leftarrow \min \{+(c_j^{f(j)} - y_{h(j)} + y_{t(j)}) : j \in K_r^-\}$.

Obtenha $\delta \leftarrow \min \{ \delta_0, \delta_1, \delta_2 \}$.

Selecione, para entrar na base, um arco $s \in K_r$ que minimize δ.

Se não houver tal arco então Pare: Problema Inviável.

Se ($s \in K_r^-$ no [caso 1]) ou ($s \in K_r^+$ no [caso 2]) então faça $f(s) \leftarrow f(s) - 1$.

3. Atualize os fluxos x_j nos arcos j do ciclo $T \cup \{s\}$.

Atualize o vetor de preços duais $y = \{y_i\}$.

Se $r \neq s$ então atualize a árvore geradora T para $T \cup \{s\} \setminus \{r\}$.

Se ($x_s < d_s^{f(s)}$ ou $x_s > d_s^{f(s)+1}$) então faça $r \leftarrow s$ e Vá ao Passo 2.

Vá ao Passo 1.

Deve-se comentar que utilizando-se os coeficientes de custo reduzido à esquerda e à direita, definidos nas expressões (4) - (5), é possível mostrar que uma solução básica {x, y, z} é *dual viável* se $c_j^- \leq 0$ e $c_j^+ \geq 0$ para todo arco $j \in E$, ou seja, as restrições $yA - z \leq 0$ e $z \geq c_j^l$ são satisfeitas.

Observe-se que uma solução básica dual viável inicial pode ser obtida introduzindo-se

um nó artificial 0, que será a raiz de T, com demanda $b_0 = 0$ e também n arcos artificiais, ligando a raiz a todos os demais nós da rede, tendo cada um destes arcos um único intervalo viável para valores de fluxo dado por [0, 0], com correspondente coeficiente de custo nulo.

A solução inicial para este problema ampliado pode, então, ser calculada fazendo-se o vetor de preços duais y = 0, os fluxos em cada arco original $j \in E$ serem $x_j = d_j^{f(j)}$, de tal forma que $c_j^{f(j)-1} \leq 0 \leq c_j^{f(j)}$, e, após a substituição destes valores de x_j no sistema de equações associado às restrições do problema ampliado, os fluxos nos arcos artificiais são obtidos a partir da resolução do sistema resultante.

De forma a reduzir o esforço computacional deste algoritmo dual, pode-se forçar o arco r, que deve deixar a base, a ser exatamente o próprio arco s candidato a entrar na base, desde que se detecte, antes de realizar a próxima iteração, que $s \in I$. Isto pode ser feito através da comparação entre os valores de δ_0 e δ. Mais ainda, quando o arco s, que substitui o arco r na base, ao final da iteração fica com um fluxo inviável, é possível selecioná-lo, imediatamente, como sendo o próximo arco a ser substituído, na iteração seguinte, utilizando-se o mesmo corte corrente.

Desta maneira, o corte e a árvore geradora não precisam ser atualizados para alguns pivoteamentos. Assim, a exemplo do que ocorre com o método primal simplex da seção anterior, pode-se definir que um método dual, no contexto da PRLP, realiza um pivoteamento toda vez que o teste da razão dual é aplicado, e realiza uma iteração toda vez que um corte novo é determinado.

Num algoritmo dual, as operações dominantes são: o *procedimento para escolha da variável que sai da base*, o *procedimento de escolha da variável de entra na base*, e o *procedimento de atualização da base*. Cada um destes processos pode ser implementado de forma eficiente uma vez que se estudem suas características. De fato, uma estrutura de dados adequada foi adotada na implementação destes procedimentos e consistiu do uso de predecessores, "thread", e "reverse thread".

Para casos em que cada pivoteamento dual é não-degenerado ($\delta \neq 0$), Ahuja et al. [4] apresentam uma prova da convergência do algoritmo dual simplex para redes lineares. Uma demonstração análoga e válida para o caso da PRLP é fornecida a seguir, assumindo que o

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

problema é dual viável e cada pivoteamento dual é não degenerado.

Por construção, o método dual simplex apresentado mantém a viabilidade dual das soluções básicas consideradas, na ausência de degenerescência, cada pivoteamento dual ocasionará um aumento estrito ao valor da função objetivo dual, e, além disto, nenhuma solução básica será reanalisada pelo algoritmo. Deste modo, como o número de soluções básicas é finito o algoritmo também o será. (c.q.d.)

Na ocorrência de degenerescência, sabe-se que pivoteamento duais degenerados não alteram os preços duais, e assim o valor da função objetivo dual também não se modifica. Ahuja et al [4] descrevem uma técnica de perturbação dual que evita os pivoteamentos duais e proporciona um algoritmo dual simplex finito também para estas situações.

2.3.3. ALGORITMO OUT-OF-KILTER

O algoritmo, aqui proposto, para problemas em PRLP é uma extensão do método out-of-kilter apresentado por Kennington e Helgason [65]. Alguns conceitos específicos para a PRLP relacionados com a abordagem out-of-kilter são introduzidos antes de se descrever o algoritmo.

As soluções admitidas pelo algoritmo out-of-kilter, para um problema em PRLP, satisfazem somente as restrições de balanço nos nós da rede, ou seja, soluções intermediárias podem estar violando tanto as restrições de canalizações no arcos, como as condições de otimalidade, expressas pelas condições de folgas complementares (1) - (3) dadas na seção 2.2.

A idéia do método out-of-kilter é, iterativamente, modificar fluxos e preços de forma a decrescer a inviabilidade da solução, e, simultaneamente situar-se mais próximo da solução ótima.

A denominação out-of-kilter reflete o fato dos arcos da solução, em análise, satisfazer ou não as condições (1) - (3). Assim, um arco $j \in E$, é dito estar "*in kilter*" (IK), se as condições de folgas complementares são verificadas; caso contrário o arco j é dito estar "*out-of- kilter*" (OK).

O *diagrama de kilter*, construído para cada arco $j \in E$ (ver Figura 3), é uma maneira conveniente de ilustrar estas situações. Constam deste diagrama a coleção de todos os pontos (x_j, z_j) que satisfazem as condições de folgas complementares. Caso, para uma dada solução, os valores de x_j, e z_j levem a um ponto no respectivo diagrama que pertence a uma célula hachurada

unesp

então o arco em questão está IK, caso contrário o arco está OK.

Define-se o *número de kilter*, não-negativo, NK_j, para cada arco $j \in E$, como o valor da mudança no fluxo x_j que deve ser efetuada de forma que o arco j passe à condição IK. Para cada solução é possível obter-se um *número de kilter total*, NK, calculado por $NK = \sum_{j \in E} NK_j$, que fornece uma medida da "distância" entre a solução corrente e a ótima, isto é, quanto menor o valor de NK mais perto da solução ótima a solução atual se encontra. Deste modo, a estratégia do algoritmo é procurar uma solução que satisfaz NK = 0, isto é, todos os arcos da solução estão IK, e portanto esta solução é ótima.

		Valor do Fluxo x_j			
		...	$x_j = d_j^{f(j)}$	$x_j \in (d_j^{f(j)}, d_j^{f(j)+1})$	$x_j = d_j^{f(j)+1}$	...
Valor do custo corrente z_j	...	...	...	...	...	...
	$z_j \in [c_j^{f(j)-1}, c_j^{f(j)}]$	OK	IK	OK (*)	OK (*)	OK
	$z_j = c_j^{f(j)}$	OK	IK	IK	IK	OK
	$z_j \in [c_j^{f(j)}, c_j^{f(j)+1}]$	OK	OK (**)	OK (**)	IK	OK
	...	...	...	...	...	...

com: (*) $NK_j = x_j - d_j^{f(j)}$ e (**) $NK_j = d_j^{f(j)+1} - x_j$.

Figura 3. Diagrama de Kilter do arco j.

O algoritmo out-of-kilter consiste de duas fases, que são aplicadas visando reduzir o número de kilter total associado a uma dada solução, denominadas *fase primal*, que altera os fluxos nos arcos, e *fase dual*, que muda os preços duais nos nós.

Durante a fase primal, os preços nos nós são mantidos fixos e são alterados os fluxos associados com arcos de um ciclo, denotado por W, com características tais que permita um incremento de fluxo sem aumentar nenhum número de kilter, e diminua o valor de ao menos um destes números de kilter. Desta forma, a fase primal se preocupa em identificar este tipo de ciclo que deve incluir um arco s que está OK.

Um procedimento para verificar quando tal ciclo existe envolve o crescimento de uma

unesp

árvore especial H, com raiz no nó cauda, t_s, do arco s escolhido. Esta árvore é construída de modo que, para um nó $i \in H$, há possibilidade de um aumento de fluxo no caminho existente em H entre a raiz e o nó i. Se $h_s \in H$, o ciclo procurado W é formado pelo caminho, em H, de t_s a h_s acrescentado do arco s.

Eventualmente, a fase primal pode se encerrar concluindo que tal ciclo não existe no momento. Neste caso, mantendo-se, agora, os fluxos fixados, busca-se reduzir o número de kilter total ajustando adequadamente os valores dos preços duais, dando início a fase dual do algoritmo.

O propósito desta fase dual é determinar o decréscimo permitido nos preços duais dos nós da árvore H, sem causar aumento em nenhum número de kilter, além de decrescer o valor de ao menos um deles.

De um modo geral, o algoritmo trabalha numa seqüência sucessiva de utilização de fases primal e dual, até que se obtenha NK = 0.

Um método para cálculo dos valores de NK_j, para cada arco $j \in E$, pode ser o abaixo descrito. Encontre um vetor de intervalos correntes f = (f(j)) tal que a relação $c_j^{f(j)} \le z_j \le c_j^{f(j)+1}$ é verificada para todo arco $j \in E$. Cada NK_j pode, então, ser determinado pelas expressões:

se $c_j^{f(j)} < z_j < c_j^{f(j)+1}$ então faça $NK_j = | x_j - d_j^{f(j)} |$.

se $z_j = c_j^{f(j)} = y_{h(j)} - y_{t(j)}$ então faça

$NK_j = d_j^{f(j)} - x_j$, se $x_j < d_j^{f(j)}$,

$NK_j = 0$, se $d_j^{f(j)} \le x_j \le d_j^{f(j)+1}$,

$NK_j = x_j - d_j^{f(j)+1}$, se $x_j > d_j^{f(j)+1}$.

se $z_j = c_j^1 > y_{h(j)} - y_{t(j)}$ então faça $NK_j = x_j - d_j^1$.

se $d_j^{f(j)} < x_j < d_j^{f(j)+1}$ então considere o custo auxiliar $\dot{c}_j = c_j^{f(j)} - y_{h(j)} + y_{t(j)}$.

se $x_j = d_j^k$ então faça

$\dot{c}_j = c_j^k - z_j$ se $z_j < c_j^k$,

$\dot{c}_j = 0$ se $c_j^k \le z_j \le c_j^{k+1}$,

$\dot{c}_j = c_j^{k+1} - z_j$ se $c_j^{k+1} < z_j$.

Agora, os passos do algoritmo (Marins e Perin [72]) podem ser apresentados. Seja {x,

y, z} uma solução para o *Problema P* satisfazendo Ax = b, x ≥ 0, e, para cada arco $j \in E$, $z_j = \max \{c^1_j , y_{h(j)} - y_{t(j)}\}$ com $c_j^{f(j)} \le z_j \le c_j^{f(j)+1}$.

Algoritmo

1. **Teste de Otimalidade**

Calcule NK associado com a solução atual.

Se NK = 0 então Pare: Solução Ótima foi obtida.

Caso contrário, escolha um arco $s \in E$ com $NK_s > 0$.

Faça: $H \leftarrow \phi$, $\Delta_i \leftarrow 0$ para todo $i \in N$, e calcule $\dot{c}_s$.

Se $\dot{c}_s > 0$ então faça $V \leftarrow \{h(s)\}$, e $\Delta_{h(s)} \leftarrow NK_s$.

Caso contrário, faça $V \leftarrow \{t(s)\}$, e $\Delta_{t(s)} \leftarrow NK_s$.

2. **Fase Primal**

Determine os conjuntos:

$\varphi_1 \leftarrow \{j: j \neq s,\ \dot{c}_j < 0,\ t(j) \notin V,\ h(j) \in V \text{ e } x_j < d_j^{f(j)}\}$,

$\varphi_2 \leftarrow \{j: j \neq s,\ \dot{c}_j = 0,\ t(j) \notin V,\ h(j) \in V \text{ e } x_j < d_j^{f(j)}\}$,

$\varphi_3 \leftarrow \{j: j \neq s,\ \dot{c}_j > 0,\ t(j) \in V,\ h(j) \notin V \text{ e } x_j > d_j^{f(j)}\}$,

$\varphi_4 \leftarrow \{j: j \neq s,\ \dot{c}_j = 0,\ t(j) \in V,\ h(j) \notin V \text{ e } x_j > d_j^{f(j)-1}\}$.

Se $\varphi_1 \cup \varphi_2 \cup \varphi_3 \cup \varphi_4 = \phi$ então Vá ao Passo 3.

Caso contrário, escolha um arco $r \in \varphi_1 \cup \varphi_2 \cup \varphi_3 \cup \varphi_4$.

Se $r \in \varphi_1$ então faça $\Delta_{t(r)} \leftarrow \min \{\Delta_{h(r)} , d_r^{f(r)} - x_r\}$, $V \leftarrow V \cup \{t(r)\}$, e $H \leftarrow H \cup \{r\}$.

Se $r \in \varphi_2$ então faça $\Delta_{t(r)} \leftarrow \min \{\Delta_{h(r)} , d_r^{f(r)} - x_r\}$, $V \leftarrow V \cup \{t(r)\}$, e $H \leftarrow H \cup \{r\}$.

Se $r \in \varphi_3$ então faça $\Delta_{h(r)} \leftarrow \min \{\Delta_{t(r)} , x_r - d_r^{f(r)}\}$, $V \leftarrow V \cup \{h(r)\}$, e $H \leftarrow H \cup \{r\}$.

Se $r \in \varphi_4$ então faça $\Delta_{h(r)} \leftarrow \min \{\Delta_{t(r)} , x_r - d_r^{f(r)-1}\}$ $V \leftarrow V \cup \{h(r)\}$, e $H \leftarrow H \cup \{r\}$.

Se $\{ t(s), h(s) \} \not\subset V$ então Vá ao Passo 2.

Para os arcos do ciclo W, formado no subgrafo (V, H), aumente os respectivos valores de fluxos da quantidade dada pelo $\min\{\Delta_{h(s)}, \Delta_{t(s)}\}$

Vá ao Passo 1.

3. **Fase Dual**

Faça

$\varphi_1 \leftarrow \{j: h(j) \in V,\ t(j) \notin V \text{ e } \dot{c}_j > 0\}$,

$\varphi_2 \leftarrow \{j: h(j) \notin V, t(j) \in V \text{ e } \dot{c}_j < 0\}$.

Determine $\theta \leftarrow \min \{ | \dot{c}_j | : j \in \varphi_1 \cup \varphi_2 \}$.

Se $\theta = \infty$ então Pare: Solução Ilimitada foi obtida.

Atualize $y_i \leftarrow y_i + \theta$, $i \in N$.

Atualize $z_j \leftarrow \max \{ c^1_j, y_{h(j)} - y_{t(j)} \}$ para todo arco $j \in E$.

Vá ao Passo 2.

Este algoritmo pode ser inicializado com qualquer solução {x, y, z} que satisfaça Ax = b, $x \geq 0$, e $z_j = \max \{ c^1_j, y_{h(j)} - y_{t(j)} \}$ para $j \in E$. A principal característica da versão implementada deste algoritmo é a possibilidade do crescimento, simultâneo, de várias árvores enraizadas num *nó artificial* utilizando, inicialmente, os n *arcos artificiais* incorporados à rede original, que ligam a raiz (nó artificial) aos n nós originais da rede. A solução inicial terá todos os arcos originais IK e somente os arcos artificiais estarão OK.

Para provar que este algoritmo é finito, pode-se usar a mesma argumentação adotada em Kennington e Helgason [65]. Suponha que, para todo nó $i \in N$, e arco $j \in E$, os vetores b, c, d, x, y tem componentes inteiros, sendo o vetor d definido formado pelos limitantes de intervalos de cada arco.

Portanto, cada alteração nos fluxos feita na fase primal reduz o número de kilter do arco s de, no mínimo, o valor 1; e cada execução da fase dual ou reduz o número de kilter total de, no mínimo, o valor 1, ou acrescenta, no mínimo, um nó adicional à árvore corrente H atualmente em crescimento. Mais ainda, o número de sucessivas aplicações das fases primal e dual, para um particular arco s que está OK, é finito, pois a árvore H pode somente crescer até incorporar os n nós da rede em estudo. Assim sendo, o algoritmo out-of-kilter apresentado é finito. (c.q.d.).

2.3.4. ALGORITMO COST SCALING

Uma extensiva discussão sobre o *Problema de Circulação de Custo Mínimo* - PCCM, e suas aplicações, pode ser encontrada em Ahuja et al. [4], onde são apresentados algoritmos clássicos existentes, do tipo daqueles descritos nas seções anteriores para redes lineares, que

unesp

numa *análise de pior caso* ("worst-case analysis") tem uma *complexidade exponencial* com relação à dimensão dos problemas.

Ahuja et al. [4] descrevem, ainda, vários algoritmos com complexidade polinomial que se baseiam no conceito de *escalonamento* ("scaling") introduzido por Edmonds e Karp [35], que foram os primeiros a propor uma algoritmo com estas características para o PCCM. A idéia de tal técnica é resolver uma seqüência de subproblemas, cujos parâmetros numéricos vão se aproximando gradativamente dos parâmetros do problema original, e a solução de um subproblema auxilia na solução do próximo problema da seqüência.

No caso do PCCM, a sucessão de subproblemas é obtida pela consideração, a cada novo subproblema, de um "bit" a mais na precisão adotada para:

(a) as capacitações ("capacity scaling");

(b) os coeficientes de custo ("cost scaling");

(c) as capacitações e coeficientes de custo, simultaneamente ("double scaling").

O algoritmo desenvolvido nesta seção é uma extensão para PRLP do *algoritmo de circulação de custo mínimo* proposto por Goldberg e Tarjan [49] para programas em redes lineares com coeficientes inteiros. O procedimento de resolução utiliza aproximações sucessivas baseadas no *escalonamento dos custos* e combina as idéias de vários pesquisadores como Bland e Jensen [18], Röck [95], Tardos [115], dentre outros.

O método encontra uma solução aproximada e procura, iterativamente, ir refinando a solução corrente com respeito a um *parâmetro de erro* α adotado. Este parâmetro α é diminuído pela metade a cada iteração (isto é, a cada obtenção de novo subproblema) até que α fique suficientemente pequeno e a hipótese de *integralidade* ("integrality") dos valores dos coeficientes de custo garanta a otimalidade da solução corrente com relação ao problema original. Para medir a acuracidade de uma solução, Goldberg e Tarjan [49] usam o conceito de *α-otimalidade* ("α-optimality"), que é uma relaxação das condições de folgas complementares, dadas para PRLP pelas expressões (1) - (3), e está relacionado com a técnica clássica de perturbação para evitar degenerescência na Programação Linear.

Generalizações para a PRLP da conceituação introduzida por Goldberg e Tarjan [49] são apresentadas antes de se estabelecer os passos do algoritmo, que na resolução dos subproblemas

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

gerados pelo escalonamento dos custos usa uma *rotina de fluxo máximo.*

Seja x um vetor de fluxos primal viável com $x_j \in [\, d_j^{f(j)},\ d_j^{f(j)+1}\,]$. Defina um custo auxiliar dado por $\tilde{c}_j = c_j^{f(j)} - y_{h(j)} + y_{t(j)}$. Para um dado valor de $\alpha > 0$, fixado, x é chamado *α-ótimo* se:

$$\tilde{c}_j < -\alpha \qquad x_j = d_j^{f(j)}, \tag{6}$$

$$d_j^{f(j)} < x_j < d_j^{f(j)+1} \qquad -\alpha < \tilde{c}_j < +\alpha, \tag{7}$$

$$\tilde{c}_j > +\alpha \qquad x_j = d_j^{f(j)+1}. \tag{8}$$

Observe-se que no caso de $\alpha = 0$, as expressões acima coincidem com as condições de otimalidade (1) - (3). Uma ilustração das condições exatas de otimalidade é fornecida na Figura 4, onde qualquer combinação de valores $(x_j,\ \tilde{c}_j)$ que pertença à linha em destaque corresponde a uma solução ótima. Analogamente, as condições de α-otimalidade (6) - (8) implicam que qualquer combinação de valores $(x_j,\ \tilde{c}_j)$ pertencente à região hachurada na Figura 5 é α-ótima. Estas figuras são análogas a Figura 10.2 apresentada por Ahuja et al. [4].

Um vetor de fluxo x é denominado um *pseudofluxo* ("pseudoflow") se satisfaz apenas as restrições de canalização e de não-negatividade; não necessitando satisfazer as restrições com relação as demandas nos nós, Ax = b. Para um dado pseudofluxo $x = (x_j)$ defina um *vetor de excesso de fluxo nos nós* por $e = (e_i)$, com $e_i = b_i - \sum_j [x_j: t(j) = i] + \sum_j [x_j: h(j) = i]$.

Observe-se que caso $e = (e_i) = 0$ então Ax = b, e ,deste modo, o vetor de pseudofluxos x é um vetor de fluxos primal viável.

Como destacado por Ahuja et al. [4], no projeto, desenvolvimento e implementação de algoritmos para fluxos em redes, freqüentemente é conveniente medir os fluxos não em termos absolutos, mas em termos de *fluxo incremental* ("incremental flow") com respeito a alguma dada solução viável - tipicamente uma solução intermediária. Este procedimento leva a obtenção de uma nova rede, conhecida como *rede residual*, que funciona como uma rede remanescente de fluxos ("remaining flow network") para transportar o fluxo incremental.

unesp

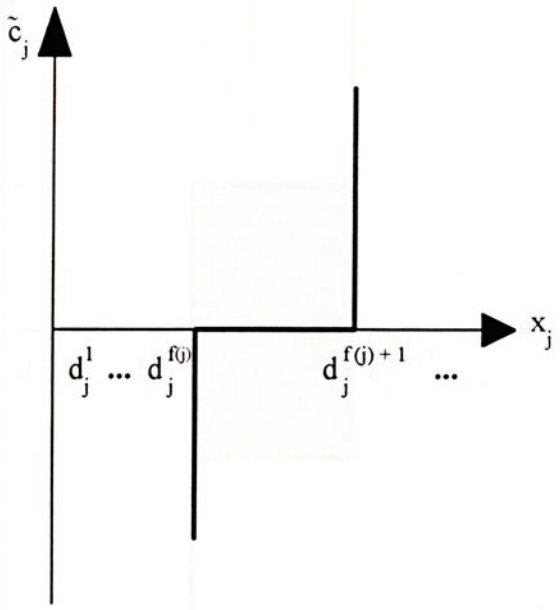


Figura 4. Condições de otimalidade exata para um arco j com $x_j \in [\, d_j^{f(j)},\ d_j^{f(j)+1}\,]$.

Ahuja et al. [4] mostram que formulações do problema na rede original e na rede residual são equivalentes, no sentido que elas apresentam uma correspondência um a um entre soluções viáveis, preservando o mesmo valor do custo total para ambas as soluções.

O conceito de rede residual é baseado na idéia a seguir exposta. Suponha que o arco j transporta x_j^0 unidades de fluxo, e $x_j^0 \in [\, d_j^{f(j)},\ d_j^{f(j)+1}\,]$ onde f(j) é o intervalo corrente do arco j. Então pode-se enviar $(d_j^{f(j)+1} - x_j^0)$ unidades de fluxo adicionais do nó t(j) ao nó h(j) pelo arco j, sem haver alterações no valor de seu intervalo corrente f(j).

Observe-se, também, que podem ser enviadas até x_j^0 unidades de fluxo do nó h(j) ao nó t(j) pelo arco j, fazendo com que o fluxo existente no arco j decresça até o valor do limitante inferior do seu fluxo corrente $d_j^{f(j)}$. Deste modo, enquanto o envio de uma unidade de fluxo de t(j) a h(j) pelo arco j aumenta o custo da solução corrente de $c_j^{f(j)}$ unidades, o envio de fluxo do nó h(j) a t(j) pelo mesmo arco decresce o custo da solução corrente de $c_j^{f(j)}$ unidades.

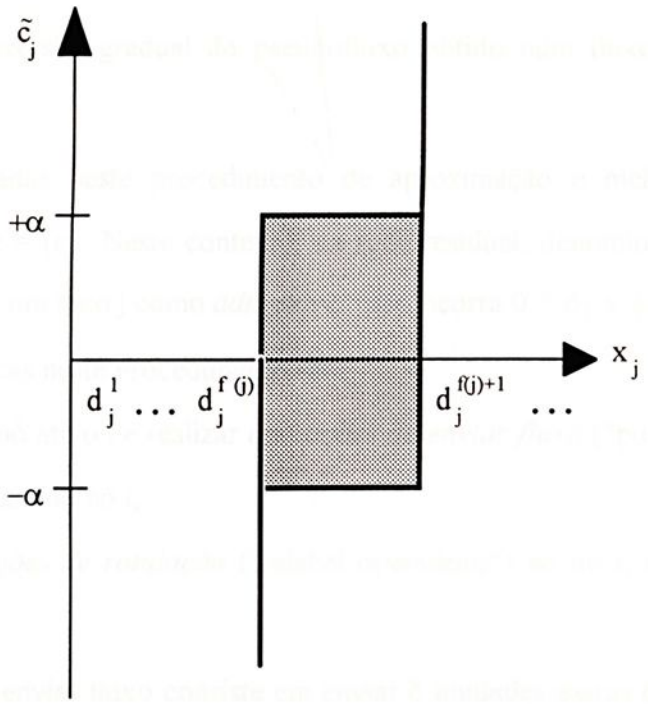


Figura 5. Condições de α-otimalidade para um arco j com $x_j \in [\, d_j^{f(j)},\ d_j^{f(j)+1}\,]$.

A partir destas idéias, pode-se definir a rede residual $G(x^0)$ com respeito a um pseudofluxo, $x_j^0 \in [\, d_j^{f(j)},\ d_j^{f(j)+1}\,]$, com vetor de intervalo correntes f, como sendo a rede formada pelos arcos com *capacidade residual* positiva, como definida abaixo. Substitua, na rede original, cada arco j por dois novos arcos:

arco j^1, de t(j) a h(j), com custo $c_j^{f(j)}$, e capacidade residual dada por $r_{j^1} = d_j^{f(j)+1} - x_j^0$,

arco j^2, de h(j) a t(j), com custo $- c_j^{f(j)}$, e capacidade residual dada por $r_{j^2} = x_j^0 - d_j^{f(j)}$.

Considerando uma rede residual G(x), as expressões (6) - (8) podem ser simplificadas. Para algum $\alpha > 0$, um vetor de fluxo (ou pseudofluxo) x é α-ótimo, em conjunto com algum vetor de preços nos nós y, se satisfaz a condição de α-otimalidade expressa por:

$\tilde{c}_j \geq +\alpha$, para todo arco j na rede residual G(x). (9)

No método implementado, α é tratado como um parâmetro, sendo, iterativamente, obtidos fluxos α-ótimos para valores sucessivamente menores de α. O algoritmo é iniciado com um fluxo α-ótimo, para algum α apropriado (ver abaixo), e realiza *fases de escalonamento nos custos*, que consistem em, repetidamente, aplicar um *procedimento de aproximação e melhoria* ("improve-approximation procedure") que transforma um fluxo α-ótimo em um fluxo $\frac{1}{2}\alpha$-ótimo.

Isto é feito, inicialmente, através da conversão do fluxo α-ótimo em um pseudofluxo $\frac{1}{2}\alpha$

-ótimo, e depois pela conversão gradual do pseudofluxo obtido num fluxo, mantendo a $\frac{1}{2}\alpha$-otimalidade.

As operações usadas neste procedimento de aproximação e melhoria criam algum excesso de fluxo nos nós e = (e_i). Neste contexto, na rede residual, denomina-se um nó i como *ativo* caso se tenha $e_i > 0$, e um arco j como *admissível* caso ocorra $0 < \tilde{c}_j \leq \frac{1}{2}\alpha$.

As operações básicas neste procedimento são:

(1) selecionar um nó ativo i e realizar *operações de enviar fluxo* ("push operations") nos arcos admissíveis j que partam do nó i,

(2) realizar *operações de rotulação* ("relabel operations") no nó i, se a rede não tiver arcos admissíveis.

Uma operação de enviar fluxo consiste em enviar δ unidades extras de fluxo a partir do nó i, pelos arcos admissíveis, de forma a:

decrescer e_i e o fluxo no arco reverso de h(j) para i (= t(j)), e

aumentar $e_{h(j)}$ e o fluxo no arco j do valor δ, com $\delta = \text{Min}\{e_i, r_j\}$.

Caso $\delta = r_j$, define-se a operação de enviar fluxo como *saturada;* caso contrário ela é chamada *não-saturada*.

Uma operação de rotulação no nó i atualiza o valor do preço dual y_i para $y_i + \alpha$, e o seu objetivo é criar novos arcos admissíveis que tenham este nó i como nó cauda.

Ao se aplicar este procedimento está se resolvendo problemas de fluxo máximo, gerados pelas fases de escalonamento nos custos, usando idéias similares dos chamados "*preflow algorithms*", que são apresentados em Ahuja et al. [4]. O processo se repete até que $\alpha < 1/n$, quando o algoritmo termina.

Os passos do algoritmo são descritos a seguir. Defina $\tilde{c}_j^- = c_j^{f(j)-1} - y_{h(j)} + y_{t(j)}$, e $\tilde{c}_j^+ = c_j^{f(j)+1} - y_{h(j)} + y_{t(j)}$. Sejam x, e y, respectivamente, uma solução viável inicial e o vetor de preços duais. Considere $\alpha = \max_{j,k}\{|c_j^k|\} + 1$, para todo $j \in E$ e $k \in \{1, ..., m(j)\}$, sendo m(j) o número de limitantes de intervalos de fluxo para o arco j.

Algoritmo

1. Enquanto $\alpha > 1/n$ faça

Para $j \in E$ faça

Se $\tilde{c}_j > +\alpha$ então enquanto $\tilde{c}_j^- \geq +\alpha$ faça $f(j) \leftarrow f(j) - 1$, e $x_j \leftarrow d_j^{f(j)+1}$.

Se $\tilde{c}_j < -\alpha$ então enquanto $\tilde{c}_j^+ \leq -\alpha$ faça $f(j) \leftarrow f(j) + 1$, e $x_j \leftarrow d_j^{f(j)}$.

Para ($i \in N$ e $e_i > 0$) faça

Repita { operações de enviar fluxo e rotular }

Se há um arco j tal que ($t(j) = i$ e $x_j < d_j^{f(j)+1}$) então $x_j \leftarrow x_j + \min\{e_i, d_j^{f(j)+1} - x_j\}$.

Se há um arco j tal que ($h(j) = i$ e $x_j > d_j^{f(j)}$) então $x_j \leftarrow x_j - \min\{e_i, x_j - d_j^{f(j)}\}$.

Se há um arco j tal que ($h(j) = i$, $x_j = d_j^{f(j)}$ e $\tilde{c}_j^- > -\alpha$) então $f(j) \leftarrow f(j) - 1$.

Se há um arco j tal que ($t(j) = i$, $x_j = d_j^{f(j)+1}$ e $\tilde{c}_j^+ < +\alpha$) então $f(j) \leftarrow f(j) + 1$.

Se não ocorrer nenhum dos casos acima então faça $y_i \leftarrow y_i + \alpha$.

Até $e_i = 0$.

Faça $\alpha \leftarrow \alpha/2$.

Caso não haja uma solução inicial disponível, é possível começar o algoritmo com $x = 0$ e $y = 0$; mas, devem ser introduzidos testes para detectar problemas inviáveis, como sugerido a seguir.

Assuma que $d_j^1 = 0$ para todo $j \in E$. Inicialmente crie uma *superorigem* e um *superdestino* para a rede original. Adicionalmente, crie arcos, com canalização superior $|b_i|$, da superorigem para todos os nós origens i, e de cada nó destino i para o superdestino. Finalmente, resolva um problema de fluxo máximo da superorigem para o superdestino, considerando que os arcos da rede original, $j \in E$, têm *limitante superior de canalização* infinito se $c_j^{m(j)} < \infty$, ou têm *limitante superior de canalização* dado por $(d_j^{m(j)} - d_j^1)$ se $c_j^{m(j)} = \infty$. Quando, $d_j^1 \neq 0$ para algum arco j, antes de se resolver o problema de fluxo máximo, se faz necessário realizar uma transformação na variável de fluxo.

Uma melhoria neste algoritmo pode ser conseguida iniciando-se com $\alpha = 2^{\lceil \log_2(nC) \rceil}$ e com critério de parada dado por $\alpha < 1$. Desta forma, todo α é uma potência de 2, e isto implica que os

unesp

valores finais dos preços duais também são inteiros. A *prova da validade* ("correcteness proof") deste algoritmo para problemas com dados inteiros é análoga aquela apresentada por Goldberg e Tarjan [49], sendo descrita abaixo.

Teorema 3.. Um fluxo viável é α-ótimo para $\alpha = C = \max_{j,k}\{|c_j^k|\} + 1$. Se $\alpha \leq 1/n$ então um fluxo viável α-ótimo é um fluxo ótimo.

Prova. Seja um vetor de fluxos viável x, com vetor de intervalos correntes f e vetor de preços duais y = 0. Então $\left|\tilde{c}_j\right| = \left|c_j^{f(j)}\right| < C$ para todo arco j, o que implica que x é C-ótimo.

Considere, agora, um vetor de fluxos x que seja 1/n-ótimo, com vetor de intervalos correntes f e vetor de preços duais y. Se x não é um fluxo ótimo, então há um ciclo orientado Q que permite um decréscimo no custo total da solução $C(x) = \sum_{j\in E} C_j(x_j)$. Este ciclo $Q = Q^+ \cup Q^-$ com $\tilde{c}_Q = \sum_{j\in Q}\tilde{c}_j < 0$, e onde $Q^+(Q^-)$ é o subconjunto de arcos de Q com a mesma (oposta) orientação de Q. Isto significa que $x_j < d_j^{f(j)+1}$ para $j \in Q^+$ e $x_j > d_j^{f(j)}$ para $j \in Q^-$.

Pela definição de fluxo α-ótimo tem-se: $\left|\tilde{c}_Q\right| = \sum_{j\in Q}\left|\tilde{c}_j\right| \leq n\alpha < 1$ para $\alpha \leq 1/n$. Portanto, $\tilde{c}_Q = 0 = \tilde{c}_j$, $j \in Q$ pois os coeficientes de custos, $c_j^{f(j)}$, são inteiros. Assim, x é um fluxo ótimo. (c.q.d.).

Teorema 4. As operações de enviar fluxo nos arcos e de rotulação de nós sempre mantém a α-otimalidade dos pseudofluxos, e ao término de sua aplicação fornece um fluxo α-ótimo.

Prova. O algoritmo sempre se inicia com um pseudofluxo α-ótimo. A operação de enviar fluxo nos arcos não altera a α-otimalidade da solução, pois não é feita nenhuma mudança nos valores dos preços duais e no intervalo corrente do arco. Uma atualização no intervalo corrente, $(f(j) \leftarrow f(j) \pm 1)$, é realizada se a α-otimalidade for mantida. Além disto, uma operação de rotulação no nó i (correspondente a se fazer uma atualização em preço dual), com excesso de fluxo $e_i > 0$, é executada quando $\tilde{c}_j^+ > \alpha$ para t(j) = i, e quando $\tilde{c}_j^- < -\alpha$ para h(j) = i. Isto significa que estes $\tilde{c}_j^-$, $\tilde{c}_j^+$ decrescem em valor absoluto e é mantida a α-otimalidade da solução.

(c.q.d.).

Finalmente deve-se destacar que este algoritmo especializado para a PRLP, da forma

como foi implementado, é do tipo *fortemente polinomial* ("strongly polynomial"), ver Marins et al.([75], [76]), e sua complexidade pode ser expressa por $O(nm' \log(n)\min\{\log(nC), m' \log(n)\})$, onde n é o número de nós, m' é o número total de intervalos lineares por partes, e C é o maior coeficiente de custo inteiro do problema.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

CAPÍTULO 3

TESTES COMPUTACIONAIS

3.1. INTRODUÇÃO

Em qualquer investigação científica deve haver a preocupação em se aplicar uma metodologia que utilize métodos estatísticos para o planejamento dos experimentos necessários, bem como para uma análise completa e rigorosa dos resultados resultantes do experimento. Contudo, como tem sido comentado na literatura, para a área de Programação Matemática poucos trabalhos, envolvendo a realização de testes computacionais comparativos entre algoritmos, vêm adotando uma metodologia adequada, o que tem levado a obtenção de conclusões imprecisas ou inconsistentes.

Neste trabalho, foi aplicado um procedimento estatístico rigoroso com a finalidade de se estudar a eficiência relativa dos quatro algoritmos apresentados no capítulo precedente.

Apresenta-se aqui, inicialmente, um levantamento bibliográfico de trabalhos relacionados com testes comparativos entre algoritmos. Na seqüência, apresenta-se o estabelecimento dos objetivos da pesquisa, o projeto dos testes computacionais comparativos realizados, incluindo a formulação do modelo estatístico adotado, e a descrição do sistema que foi implementado para se controlar e administrar os testes computacionais, além dos resultados obtidos pelos algoritmos na

unesp

solução de redes lineares por partes consideradas, e os resultados das análises estatísticas efetuadas.

3.2. O PROJETO DOS EXPERIMENTOS COMPUTACIONAIS

3.2.1. HISTÓRICO

Nos últimos 40 anos, muitas técnicas de Programação Matemática, incluindo as de Programação em Redes, têm sido desenvolvidas, analisadas e comparadas. Deste modo, uma quantidade crescente de estudos computacionais têm sido objeto de publicação em revistas, teses, e livros da área, buscando alcançar conclusões sobre a eficiência relativa de algoritmos concorrentes quando aplicados a problemas particulares. Infelizmente, como comentado por Jackson e Mulvey [62], a metodologia aplicada em muitos destes testes tem se mostrado, várias vezes, inadequada.

Visando proporcionar informações de como se realizar experimentos computacionais de forma rigorosa, inclusive no que se refere a sua documentação, vários artigos têm sido publicados, como os de Crowder et al. [26], Greenberg et al. [53], Hoffman e Jackson [58], Ignizio ([59], e [60]), e Jackson et al. [61]. A seguir passa-se a comentar vários aspectos levantados por estes e outros autores com relação a condução de experimentos computacionais comparativos entre algoritmos.

Inicialmente, em qualquer experimento científico para testar "software" matemático, as seguintes questões básicas devem ser consideradas antes da realização dos testes:

(1). Quais são os objetivos do experimento? Qual tipo de conclusões se espera obter? O que será observado (ou medido) no experimento?,

(2). Como é possível realizar o experimento? Qual é a metodologia disponível? Como é possível obter as medidas de interesse, e qual é a precisão desejada?.

unesp

Além destas perguntas, os autores devem se preocupar em como documentar adequadamente os resultados da pesquisa, de forma a, por exemplo, permitir a reprodução dos resultados dentro de alguma tolerância devido ao ambiente computacional utilizado, ou ainda permitir a extensão do que foi conseguido por outros pesquisadores. Uma excelente discussão sobre este tema, bem como recomendações de como documentar testes computacionais, pode ser vista em Crowder et al. [26], Greenberg [52], e Jackson et al. [61].

O desempenho de algoritmos, tradicionalmente, tem sido estudado sob dois enfoques, cada qual tendo seus méritos e deficiências relativas:

(a) *Análise Teórica* ou *Análise de Pior Caso* ("Worst-Case Analysis"),

(b) *Análise Empírica*.

O primeiro enfoque, que nos últimos 20 anos tem sido bem adotado para avaliar algoritmos (métodos matemáticos com convergência finita), realiza uma análise teórica baseada no número de vezes que cada etapa do algoritmo é realizada, dado um conjunto de hipóteses concernentes as características do problema em questão. Esta análise assume que cada operação do algoritmo (aritmética ou lógica) gasta uma unidade de tempo, e se ocupa, então, em determinar um limitante superior para o tempo requerido pelo método para resolver uma dada instância do problema.

Este limitante é conhecido como a *complexidade* do algoritmo, e pode ser expresso como uma função dos parâmetros do problema, usando, por exemplo, a notação conhecida como *"Big O notation"*. Comumente, para modelos em redes, estes parâmetros são, o número de nós (n), número de arcos (m), o maior coeficiente de custo nos arcos (C), e o maior valor de limitante de canalização nos arcos (U). Normalmente, obtém-se um limitante para o número de iterações, outro limitante para o número de etapas por iteração, e a complexidade do algoritmo é expressa como o produto dos valores de tais limitantes.

unesp

Embora de interesse, tal enfoque freqüentemente é de pouco valor na avaliação do comportamento típico de um algoritmo. Como colocado por Ahuja et al. [4], muitas vezes, na prática, o algoritmo tem um desempenho superior ao da sua complexidade, pois esta análise se concentra nos casos patológicos para determinar o comportamento do método, mesmo que estes casos sejam raros em situações reais. Numa tentativa de superar esta deficiência, alguns estudos têm utilizado uma *análise de caso médio* ("expected case analysis"), onde a complexidade é estudada, sob considerações probabilisticas, a partir de um conjunto de hipóteses relacionadas com a população de instâncias do problema. Estas hipóteses podem fazer a análise mais útil, mas podem, ainda, não refletir importantes características de problemas reais.

Como uma alternativa à análise teórica, principalmente se o objetivo do estudo é a obtenção de informações sobre o comportamento do algoritmo na prática, o enfoque usado tem sido o da análise empírica. Nesta análise, o desempenho real do algoritmo é observado quando ele é aplicado a um conjunto selecionado de problemas; o dilema, agora, é optar entre escolher para a realização dos testes o uso de problemas reais ou a geração de problemas aleatórios com determinadas características desejadas.

Crowder et al. [26] observam que problemas gerados aleatoriamente podem não ser representativos de problemas que comumente ocorram na prática, contudo, como a população de onde os problemas são gerados é conhecida e controlável, as generalizações inferidas podem ser consideradas válidas. O mesmo não ocorre quando se utiliza problemas reais, pois neste caso a população é desconhecida e as generalizações são questionáveis.

Estes autores sugerem, ainda, algumas medidas de desempenho relevantes, no contexto de testes computacionais com algoritmos, tais como: tempo de CPU gasto pelo algoritmo, acuracidade numérica da solução obtida, número de iterações necessárias, robustez do algoritmo frente a situações problemáticas, facilidade de aplicação do método pelo usuário, e requisitos de armazenamento de dados e para a execução do método. A escolha de qual medida de desempenho é apropriada para um determinado estudo depende dos objetivos propostos do experimento.

A importância de se considerar problemas reais tem sido reconhecida nas últimas duas décadas, e existem disponíveis algumas bibliotecas de problemas de teste associados com a

unesp

realidade, de maneira que diferentes pesquisadores podem usar os mesmos problemas e obter comparações válidas. Há bibliotecas de problemas disponíveis para uma variedade de situações, pode-se, por exemplo, citar a MIPLIB [82].

Greenberg [52] comenta que uma desvantagem em se adotar o uso de uma biblioteca de problemas reais é a impossibilidade de se efetuar, de forma rigorosa, inferências sobre o comportamento do algoritmo, ele sugere o uso de uma aleatoriedade controlada ("controlled randomisation") na geração dos problemas de teste. Jackson et al. [61], concluem que o tipo de problemas considerados no teste depende da área em que o estudo está sendo realizado, e ilustram esta opinião observando que no caso da Programação Não-Linear é mais comum o uso de conjuntos de problemas particulares selecionados ("hand-picked test sets"), ao passo que na área de Programação em Redes costuma-se utilizar geradores de problemas aleatórios.

Deve-se comentar, além disto, que para muitas situações, os dados referentes a casos reais não estão à disposição em número suficiente para possibilitar uma comparação significativa. Nestes casos, uma geração de problemas aleatórios, cuidadosamente controlada, se apresenta como uma alternativa interessante, e proporciona a oportunidade de se levar a efeito uma análise rigorosa do ponto de vista estatistico. Esta abordagem é a que foi adotada neste trabalho, os seus detalhes são explicitados e discutidos nas seções que se seguem.

3.2.2. O EXPERIMENTO COMPUTACIONAL

O principal objetivo deste trabalho é, por meio de um experimento computacional, com base estatística, buscar informações sobre o desempenho relativo dos quatro algoritmos, descritos no capítulo anterior, comparados com respeito a quatro fatores, associados aos parâmetros típicos de problemas em redes lineares por partes, quando considerados individualmente ou em combinações:

(a) Fator *Classe* do problema: foram admitidos os *Problemas de Transporte Clássico*, os *Problemas de Transporte com Baldeação* ("Transshipment Problems"), e os *Problemas de Circulação;*

unesp

(b) Fator *Dimensão* do problema: aqui considerada como sendo medida pelo número de arcos (m) e nós (n) da rede associada ao problema, conforme os valores fornecidos na Tabela 1 abaixo;

(c1) Fator *Proporção de arcos capacitados*: isto é, a proporção de arcos da rede tendo o coeficiente de custo do último intervalo de fluxo, linear por partes, igual a infinito, foram considerados *os níveis de capacitação* de 100%, 80%, 50%, 20%, e 0%;

(c2) Fator *Número de limitantes de intervalos* de fluxo ("breakpoints") para cada arco: foram consideradas as situações com 03, 06 e 09 "breakpoints" por arco;

(d) Fator *Método* aplicado: foram incluídos os quatro algoritmos especializados para redes lineares por partes implementados.

Escolheu-se como sendo a variável a ser observada (para outras opções, ver Crowder e Saunders [24]) nos experimentos o tempo de CPU gasto pelos algoritmos na solução de amostras, geradas aleatoriamente, de várias categorias de modelos de PRLP. Este tempo observado exclui o tempo gasto com o processamento da entrada de dados e saída de resultados, considerando apenas o tempo de execução de cada algoritmo.

Dimensão do Problema	n	m
Pequeno	**250**	**2400**
Médio	**1000**	**12000**
Grande	**2500**	**25000**

Tabela 1. Níveis do fator dimensão do problema

Estes experimentos buscam a obtenção de respostas para as questões já colocadas no capítulo 1:

(1) Há um algoritmo que é sempre melhor que os demais testados, em qualquer situação? Ou alternativamente, qual algoritmo tem o melhor desempenho em cada categoria de modelos de PRLP considerados?;

(2) Quais são os efeitos dos fatores considerados no desempenho de cada algoritmo?;

FACULDADE DE ENGENHARIA
DE GUARATINGUETÁ
BIBLIOTECA

000312

unesp

(3) Quais são os efeitos de mudanças conjuntas nos fatores estudados no desempenho de cada algoritmo?.

Estes, são, pois, os objetivos dos experimentos cujo projeto, modelo estatístico, e resultados serão descritos a seguir.

3.2.3. O PROJETO

Esta seção descreve o projeto adotado para o experimento computacional envolvendo a comparação dos quatro algoritmos especializados para a PRLP, apresentados anteriormente: primal simplex, dual simplex, out-of-kilter e cost scaling. Maiores detalhes sobre a teoria existente para o planejamento de experimentos podem ser encontrados em Anderson e McLean [11], ou Montgomery [84], e para uma breve revisão sobre trabalhos relacionados com o tema e Programação Matemática recomenda-se a leitura de Amini e Racer [7], e de Amini e Barr ([8], e [9]).

O procedimento escolhido foi o de gerar redes com cada uma das três classes de problemas consideradas (transporte clássico, transporte com baldeação e circulação), e cada uma das três dimensões admitidas (Pequeno, Médio, e grande), desta maneira nove estruturas de redes foram criadas.

Num *primeiro experimento*, para cada uma destas estruturas, foram introduzidos os cinco niveis de capacitação nos arcos: 100% (ou seja, todos os arcos possuem restrição de capacidade), 80%, 50%, 20% ,e 0% (ou seja, não há restrição de capacidade nos arcos). Todas as quarenta e cinco redes lineares por partes geradas foram, então, resolvidas por cada um dos dos quatro algoritmos, produzindo cento e oitenta observações de tempos de CPU. Admitiu-se, para todos os problemas estudados, o número de "breakpoints" para cada arco como sendo igual a três.

unesp

Nas Tabelas 04, 05, e 06, da seção 3.3, constam os tempos de CPU (em segundos), obtidos na resolução de tais redes lineares por partes pelos quatro algoritmos estudados; além disto na Tabela 10, da mesma seção, apresenta-se os tempos médios de CPU (em segundos) com relação a cada valor de nível admitido dos fatores considerados.

Para um *segundo experimento* realizado, para cada uma das 9 estruturas de redes criadas, introduziu-se a consideração das três possibilidades para o número de "breakpoints" para cada arco da rede: 03, 06, e 09. Foram, assim, geradas vinte e sete redes lineares por partes, considerando um nível de capacitação fixo nos arcos de 50%, e resolvidas pelos quatro algoritmos, resultando num total de cento e oito observações de tempos de CPU.

As Tabelas 07, 08, 09, da seção 3.3, apresentam os tempos de CPU (em segundos), gastos por cada método na solução das redes lineares por partes geradas. A Tabela 12, da da mesma seção, apresenta para este experimento os tempos médios de CPU (em segundos) para cada valor de nível admitido dos fatores considerados.

A natureza de tal processo de geração é tal que estas 180 e 108 observações de tempo de CPU relativas, respectivamente, ao primeiro e ao segundo experimentos, não são todas independentes e, como identificado por Amini e Barr [10], se faz apropriado o uso de um projeto de experimentos do tipo "*split plot*" para uma análise de variância adequada dos resultados, ver conceituação teórica em Anderson e McLean [11], Montgomery [84], e SAS [110].

O princípio básico deste tipo projeto de experimentos, onde dentro de cada combinação de tratamentos (ou níveis fixados para fatores) há muitas subcombinações de outros tratamentos (ou níveis possíveis de outros fatores) é o seguinte: há "plots" principais - para os quais niveis de um ou mais fatores são aplicados - que são divididos em "sub-plots" ou "split plots" - para os quais niveis de um ou mais fatores adicionais são aplicados. Este esquema de análise faz com que o número de observações necessárias para o estudo seja reduzido.

unesp

No estudo em questão, para ambos os experimentos, os "plots" principais são os fatores *classe* e a *dimensão* do problema; sendo os fatores do tipo "sub-plot", respectivamente, para o primeiro e segundo experimentos, o nível de capacitação nos arcos e o número de "breakpoints", e, novamente para os dois experimentos, o tipo de método empregado na resolução do problema sendo admitido como um fator do tipo "sub-sub-plot".

Para responder as perguntas colocadas na seção 3.2.2, deve-se comparar os algoritmos sob as diferentes combinações de tratamentos, definidos pelo experimento. Para tanto, deve-se fazer uma análise completa dos efeitos dos fatores sobre o desempenho de cada algoritmo, quando estes fatores são considerados individual ou conjuntamente, o que será medido pelo tempo de CPU gasto na solução de cada problema.

O modelo estatístico completo utilizado, pelo primeiro experimento, para representar o tempo de CPU em função dos fatores considerados e de fontes de erros nos testes, sob um projeto tipo "split-plot" é dado a seguir:

$$X_{ijkh} = \mu + K_i + S_j + (KS)_{ij} + C_k + (KC)_{ik} + (SC)_{jk} + (KSC)_{ijk} + M_h + (KM)_{ih} + (SM)_{jh} + (KSM)_{ijh} + (KCM)_{ikh} + (SCM)_{jkh} + (KSCM)_{ijkh},$$

onde,

μ é o tempo médio de CPU;

K_i é o efeito do fator "plot" principal *classe*, com $i \in$ {Transporte, Transporte com Baldeação, Circulação};

S_j é o efeito do fator "plot" principal *dimensão*, com $j \in$ {Pequeno, Médio, Grande};

C_k é o efeito do fator "sub-plot" *capacitação*, com $k \in$ {1.00, 0.80, 0.50, 0.20, 0.00};

M_h é o efeito do fator "sub-sub-plot" *método aplicado*, com $h \in$ {Primal, Dual, Out-of-Kilter, Cost Scaling};

X_{ijkh} é o tempo de CPU.

unesp

Observe-se que, os demais termos, neste modelo, se referem as interações envolvendo os fatores dois a dois, três a três, e quatro a quatro que podem afetar a variável tempo de CPU considerada.

Observe-se, ainda, que, para o segundo experimento, o modelo estatístico é idêntico ao acima, apenas substituindo-se o fator *capacitação* C_k, pelo fator *número de "breakpoints"* por arco B_l com $l \in \{03, 06, 09\}$, que será um "sub-plot" no experimento.

Pelos argumentos explicitados abaixo, a metodologia estatística requerida para o desenvolvimento da análise destes experimentos envolve a técnica de Análise de Variância (ANOVA) e, também se faz necessária, uma técnica de Comparação de Várias Médias.

A Análise de Variância fornece informações para testar a significância da diferença entre os tempos médios de CPU gastos pelos algoritmos, tanto para cada tratamento individualmente, como para as combinações de tratamentos.

Analogamente ao trabalho de Amini e Barr [10], os objetivos deste trabalho podem ser formulados em termos de hipóteses estatísticas, que foram divididas em dois grupos: hipóteses para detectar diferenças significativas (com α = nível de significância = 5%) entre os tempos médios de CPU associados a cada fator individualmente, e hipóteses para identificar diferenças significantes para as situações com interações entre fatores.

As hipóteses nulas podem ser estabelecidas como sendo: grupo (dois a dois, três a três, e quatro a quatro) de tempos médios de CPU que são iguais, considerando os efeitos de cada fator individualmente ou em combinações. As hipóteses alternativas são: ao menos dois tempos médios de CPU do grupo, sob a mesma combinação de tratamentos, não são iguais.

As Tabelas 11 e 13, da seção 3.3, apresentam, respectivamente, para o primeiro e segundo experimentos realizados, os resultados fornecidos pelo procedimento ANOVA, disponível no "software" estatístico SAS utilizado, para ambos os experimentos. A interpretação estatística destas informações é desenvolvida no Capítulo 4.

unesp

Infelizmente, quando há a comparação de várias médias, como é o caso deste trabalho, o procedimento ANOVA indica, somente, se há diferença significante entre elas, mas não indica quais médias realmente diferem, ou seja, este método não permite se fazer um ranqueamento dos algoritmos, baseado nos respectivos desempenhos, com relação a diferentes combinações de tratamentos.

Percebendo-se a ocorrência de diferenças significativas entre várias médias, é conveniente se determinar quais grupos de médias são de fato significantemente diferentes. Faz-se necessário, então, o uso de métodos de comparação de várias médias tais como: *Teste de Tukey* ("Tukey's Significant Difference Test"), *Teste de Fisher* ("Fisher's Least Significant Difference Procedure"), ou ainda o *Teste de Duncan* ("Duncan' Multiple Range Test") que são descritos em Montgomery [84] e SAS [110].

Neste trabalho, optou-se pelo Teste de Tukey, também disponível no software estatístico SAS [110] usado, pois este método controla o chamado "*experimentwise error rate* "- EER para comparações múltiplas, que é definido como a probabilidade de serem rejeitadas uma ou mais hipóteses nulas, quando se aplica o teste estatístico com duas ou mais hipóteses nulas. Neste trabalho, há uma comparação de múltiplas médias o que exige um controle adequado sobre o EER, o que vem a justificar a opção feita para os experimentos em questão.

A aplicação do Teste de Tukey, para cada combinação de valores de fatores, possibilita o ranqueamento dos algoritmos relativamente aos demais, o que é feito através de um código de letras A, B, C, e D; que são associadas a cada algoritmo, com as letras A e D indicando, respectivamente, o pior desempenho e o melhor desempenho. Quando não for detectada, pelo teste, uma diferença significativa entre os algoritmos, associa-se a mesma letra para os algoritmos, neste caso diz-se que os algoritmos são estatísticamente indistingüíveis.

Os resultados da aplicação desta metodologia, considerando o nível de significância do teste α = 5%, respectivamente, para o primeiro e segundo experimentos, estão fornecidos nas

unesp

Tabelas 14 e 15, da seção 3.3, e a interpretação estatística destas informações é feita no capítulo subseqüente.

3.2.4. A IMPLEMENTAÇÃO

Visando a simplificação e a estruturação da geração e análise dos resultados obtidos no teste, os experimentos computacionais foram inteiramente feitos utilizando-se o *NplSys - Network Piecewise-Linear Testing System*, especialmente desenvolvido (ver Senne et al. [103]) para o estudo em questão. No Apêndice 1 há a descrição completa deste sistema usado, a seguir descreve-se brevemente alguns detalhes deste sistema.

O *NplSys* é constituído de três componentes principais: um subsistema de gerenciamento (*SysMa*), um subsistema gerador de redes lineares por partes (*NplGen*), e um conjunto de quatro subsistemas com as implementações dos quatro algoritmos (*Solvers*). O sistema também produz dados no formato adequado para o seu tratamento por um subsistema desenvolvido para efetuar análises estatísticas (*StaTest*). A arquitetura do sistema está na Figura 6.

O primeiro componente SysMa, é um módulo com características amigáveis para o usuário, e sua tarefa é administrar os testes. Inicialmente, são lidos os dados de entrada de três arquivos, denominados *net.par*, *net.sys*, e *net.see*.

O arquivo net.par contém a identificação das amostras de problemas a serem geradas, e o conjunto de parâmetros associados a cada uma delas. O arquivo net.sys tem as informações referentes ao tamanho de cada amostra, o valor a ser adotado para infinito pelos algoritmos, a configuração (podendo ser interativa ou não com o usuário) do SysMa para realizar os testes, o limite de tempo máximo para execução de cada algoritmo, e qual ou quais algoritmos se deseja executar para cada amostra. Caso se opte pelo modo interativo do sistema, então é necessário que o usuário coloque no arquivo net.see o número apropriado de valores de sementes ("seeds") escolhidas para a geração de cada amostra de problemas.

unesp

Após ler os dados de entrada, o SysMa aciona o gerador de redes lineares por partes, NplGen, para gerar as redes aleatórias para a amostra de problemas correspondente ao primeiro conjunto de parâmetros de net.par, e chama do componente Solver os algoritmos selecionados para resolvê-lo.

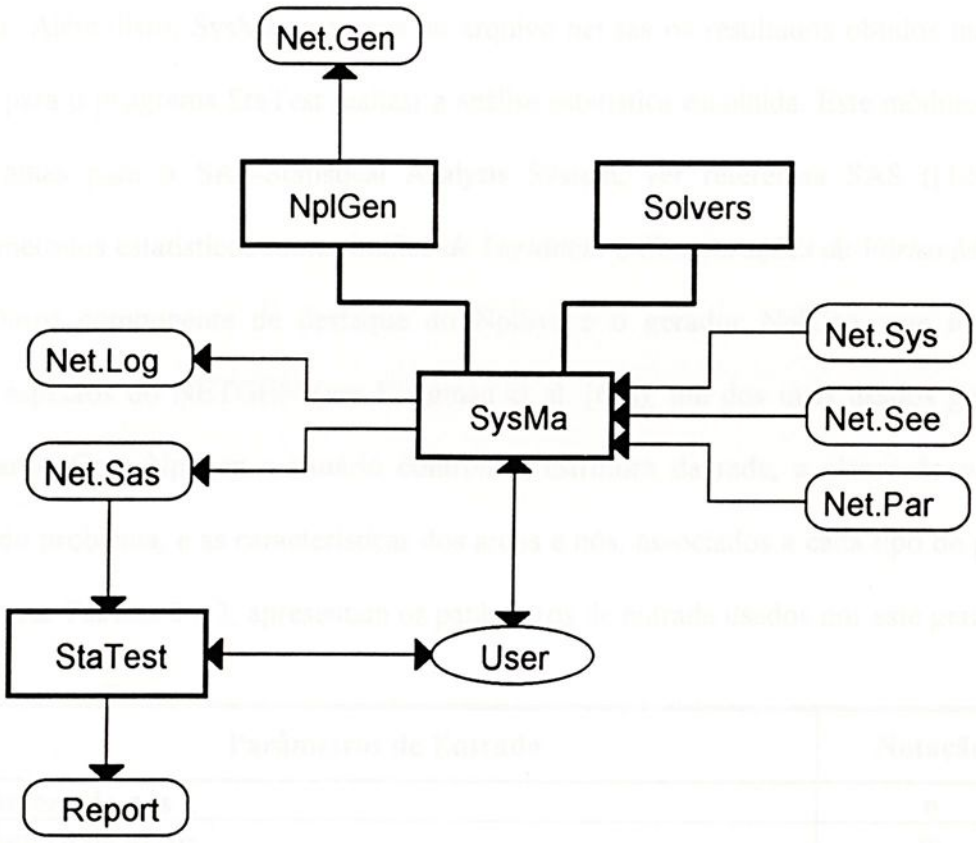


Figura 6. Arquitetura do NplSys

Para cada rede gerada e algoritmo executado, são computados os seguintes resultados: tempo de CPU, número de iterações, número de pivoteamentos/fases (ver Capítulo 2 para uma conceituação destes termos para a PRLP), os valores das funções objetivo do problema primal e do problema dual (observe-se que, para problemas com soluções ótimas estes valores devem coincidir), e um código com a informação sobre o status do problema resolvido (podendo ser inviável, apresentar solução ilimitada, ter solução ótima, ou ter o limite de tempo para execução do método ultrapassado).

Este processo continua até todas as redes de uma amostra terem sido resolvidas, estes resultados são transferidos para o arquivo net*num*.log, onde *num* é o número de identificação da amostra especificado em net.par.

Finalmente, repete-se o processo, acima descrito, para todos os conjuntos de parâmetros em net.par. Além disto, SysMa armazena no arquivo net.sas os resultados obtidos num formato adequado para o programa StaTest realizar a análise estatística escolhida. Este módulo é formado por programas para o SAS-Statistical Analysis System, ver referência SAS ([109], [110]), incluindo métodos estatísticos *como Análise de Variância* e *Comparações de Várias Médias.*

Outro componente de destaque do NplSys é o gerador NplGen, que incorpora os principais aspectos do NETGEN (ver Klingman et al. [67]), um dos mais usados geradores de redes lineares. Com NplGen o usuário controla a estrutura da rede, a classe do problema, a dimensão do problema, e as características dos arcos e nós, associados a cada tipo de problema a ser gerado. As Tabelas 2 e 3, apresentam os parâmetros de entrada usados por este gerador.

Parâmetros de Entrada	Notação
Número de nós	**n**
Número de arcos	**m**
Número de limitantes de intervalos/arco	**nb**
Proporção de nós de transbordo	p_{TRANS}
Proporção de arcos capacitados	p_{CAP}

Tabela 2. Parâmetros da rede

Atributo	Distribuição: Uniforme em (a,b)
Demanda nos nós	**U (-100, 100)**
Coeficientes de custo	**U (-5, 20)**
Limitantes de intervalos	**U (0, 250)**

Tabela 3. Distribuição dos parâmetros dos problemas.

Observe-se que p_{TRANS} é a proporção de nós de transbordo, e que para $p_{TRANS} = 0$, $0 < p_{TRANS} < 1$, e $p_{TRANS} = 1$, tem-se a geração de, respectivamente, um problema de *Transporte*

unesp

Clássico, um problema de *Transporte com Nós de Transbordo*, e um problema de *Circulação*. Para uma descrição completa de todos os parâmetros de entrada necessários para o uso do NplSys, consultar o Apêndice 1.

3.3 RESULTADOS DO EXPERIMENTO

A versão utilizada do sistema de gerenciamento, controle e análise para a realização dos testes foi a versão denominada *NplSys/UNIX*, que é adequada para uso em máquinas do tipo SUN SPARCstations, e suportam redes lineares por partes de grande dimensão. Foi utilizada, no desenvolvimento do NplSys/UNIX, a linguagem de programação Sun Pascal Release 2.1/SC 1.0 com a opção -O2 para compilação, para detalhes veja Sun Microsystem ([111], [112]).

Os testes utilizando este sistema, que envolveram a obtenção de 180 e 108 observações, respectivamente, para o primeiro e segundo experimentos, foram realizados numa máquina SUN SPARCstation 10, num ambiente de multiusuários, do Departamento de Matemática e Estatística da Brunel University, na Inglaterra (Marins et al. [77]).

Para o primeiro experimento, os tempos de CPU, de cada algoritmo para cada categoria de problema resolvida, estão dispostos nas Tabelas 04, 05, 06, e os valores médios de CPU, para cada nível de fator estudado estão na Tabela 10; com respeito ao segundo experimento, os valores correspondentes, tempos de CPU e tempo médio de CPU, estão, respectivamente, nas Tabelas 07, 08, 09, e 12.

Deve-se comentar que vários autores, como Crowder et al. [26] e Maros [79], têm observado que a execução de programas, num ambiente de multiusuários, pode levar a valores levemente diferentes de tempo de CPU para diferentes corridas dos programas, dependendo do volume de trabalho existente no ambiente computacional utilizado, no momento de cada execução dos programas.

Para minimizar os efeitos de tal problema, neste trabalho procurou-se desenvolver os testes computacionais considerando uma ordem aleatória de resolução das redes lineares por partes geradas, e também rodar o NplSys/UNIX , no modo "batch", em horários de baixa

unesp

utilização do Centro de Computação da universidade, tais como no período das 22:00 às 7:00 horas nos dias de semana, e aos finais de semana.

Uma dificuldade que surgiu, detectada durante a execução dos algoritmos, foi o fato de que o algoritmo dual simplex freqüentemente não conseguia resolver os problemas gerados, dentro do limite máximo estabelecido de duas horas de execução para cada método.

Inicialmente, pensou-se em tratar tais casos considerando os tempos de CPU superiores a 02 horas como *a ocorrência de valores não disponíveis* ("missing values") nos testes, o que é válido e tem sido adotado em algumas situações relatadas na literatura (ver Sampford e Taylor [101], e SAS [110]). Um outro procedimento possível, relatado por Taylor [116], é a obtenção de estimativas de máxima verossimilhança destas observações não disponíveis, ou, como é conhecido em estatística, o uso de "type 1 censored data".

Porém, considerando-se que em vários dos problemas resolvidos, apesar do tempo exato de solução gasto pelo algoritmo dual simplex não ter sido encontrado, conseguiu-se informações relevantes com relação a um limitante inferior para estes tempos reais de resolução. Além disto, como não ficava claro o uso no contexto dos testes em questão dos procedimentos alternativos citados na literatura, decidiu-se considerar os tempos de CPU como iguais a 2 horas, para os casos que excederam o limite de tempo de execução fixado.

De fato, este tipo de atitude é tecnicamente inválido do ponto de vista estatístico, mas é uma ação conservadora que, em vista da constatação da óbvia diferença de desempenhos dos algoritmos estudados, não interfere nas conclusões do trabalho. Uma alternativa poderia ser omitir todos os resultados do algoritmo dual simplex, excluindo, portanto, tal método da análise feita, mas preferiu-se manter o projeto original completo do experimento com o dual simplex incorporado.A seguir são apresentadas as tabelas com os resultados obtidos na resolução das redes geradas e resolvidas pelos quatro algoritmos, as análises de variância e comparações de várias médias realizadas, para ambos os experimentos realizados. Todos os tempos de CPU que constam destas tabelas são dados em segundos.

unesp

Classe (K): problemas de transporte lineares por partes					
Dimensão (S)	p_{CAP} (C)	Métodos (M)			
		Primal	Dual	Out of Kilter	Cost Scaling
Pequeno	1,00	2,31	35,76	13,81	25,55
	0,80	2,30	45,60	13,65	28,75
	0,50	2,48	52,93	13,30	28,76
	0,20	2,48	34,25	14,51	29,85
	0,00	2,40	43,15	13,23	28,05
Médio	1,00	99,23	> 7200,00	517,00	618,48
	0,80	103,33	> 7200,00	493,64	797,94
	0,50	98,61	> 7200,00	487,13	794,98
	0,20	94,66	> 7200,00	484,23	794,05
	0,00	98,75	> 7200,00	517,88	798,29
Grande	1,00	841,00	> 7200,00	2707,11	3356,58
	0,80	834,44	> 7200,00	2613,28	3832,38
	0,50	825,35	> 7200,00	2616,71	3962,52
	0,20	852,93	> 7200,00	2642,68	4351,92
	0,00	849,61	> 7200,00	2643,13	3825,21

Tabela 04. Experimento 1: tempos de CPU para problemas de transporte clássico.

Classe (K): problemas de transporte lineares por partes com baldeação					
Dimensão (S)	p_{CAP} (C)	Métodos (M)			
		Primal	Dual	Out of Kilter	Cost Scaling
Pequeno	1,00	3,05	64,85	13,63	30,38
	0,80	3,21	55,91	13,30	31,93
	0,50	3,20	50,35	12,73	32,61
	0,20	3,28	61,60	12,95	32,10
	0,00	3,18	40,43	13,03	30,83
Médio	1,00	133,10	> 7200,00	502,65	1009,00
	0,80	136,55	> 7200,00	502,65	1115,44
	0,50	138,76	> 7200,00	489,24	1119,88
	0,20	131,76	> 7200,00	510,08	1132,19
	0,00	126,54	> 7200,00	478,21	1072,35
Grande	1,00	1115,25	> 7200,00	2741,29	5778,10
	0,80	1101,86	> 7200,00	2719,84	6522,37
	0,50	1123,42	> 7200,00	2702,09	6446,12
	0,20	1129,42	> 7200,00	2668,00	6502,73
	0,00	1132,81	> 7200,00	2717,50	6340,27

Tabela 05. Experimento 1:tempos de CPU para problemas de transporte com baldeação.

unesp

Classe (K): problemas de circulação lineares por partes					
Dimensão (S)	p_{CAP} (C)	Métodos (M)			
		Primal	Dual	Out of Kilter	Cost Scaling
Pequeno	1,00	7,28	142,38	8,16	12,75
	0,80	7,91	507,50	9,21	16,06
	0,50	7,65	470,30	8,15	14,35
	0,20	7,76	149,40	8,00	14,65
	0,00	7,28	1980,76	7,13	12,66
Médio	1,00	358,18	> 7200,00	294,79	378,43
	0,80	357,15	> 7200,00	308,85	491,80
	0,50	355,68	> 7200,00	278,69	510,11
	0,20	362,40	> 7200,00	298,81	481,79
	0,00	356,31	> 7200,00	293,75	475,29
Grande	1,00	2726,57	> 7200,00	1845,16	2591,66
	0,80	2692,78	> 7200,00	1776,68	2876,76
	0,50	2731,43	> 7200,00	1769,86	2965,40
	0,20	2740,78	> 7200,00	1768,15	2860,96
	0,00	2748,23	> 7200,00	1731,45	2953,29

Tabela 06. Experimento 1 :tempos de CPU para problemas de circulação.

Classe (K): problema de transporte linear por partes					
Dimensão (S)	No. de Breakpoints (B)	Métodos (M)			
		Primal	Dual	Out of Kilter	Cost Scaling
Pequeno	03	2,48	52,93	13,30	28,76
	06	3,90	392,35	23,08	37,03
	09	4,56	>7200,00	34,61	40,71
Médio	03	98,61	>7200,00	487,13	794,98
	06	143,65	>720000	868,36	1020,46
	09	181,67	>7200,00	1235,13	1110,53
Grande	03	825,35	>7200,00	2616,71	3962,52
	06	1150,37	>7200,00	4767,01	5053,28
	09	1441,24	>7200,00	6487,09	5320,43

Tabela 07. Experimento 2: tempos de CPU para problemas de transporte clássico.



Classe (K): problema de transporte com baldeação linear por partes					
Dimensão (S)	No. de Breakpoints (B)	Métodos (M)			
		Primal	Dual	Out of Kilter	Cost Scaling
Pequeno	03	3,20	50,35	12,73	32,61
	06	5,28	670,39	21,75	42,53
	09	6,50	350,19	28,91	42,13
Médio	03	138,76	>7200,00	489,24	1119,88
	06	190,05	>7200,00	835,90	1347,34
	09	248,26	>7200,00	1125,17	1325,47
Grande	03	1123,42	>7200,00	2702,09	6446,12
	06	1554,04	>7200,00	4447,86	>7200,00
	09	2019,62	>7200,00	5935,65	>7200,00

Tabela 08. Experimento 2: tempos de CPU para o problema de transporte com baldeação.

Classe (K): problema de circulação					
Dimensão (S)	No. de Breakpoints (B)	Métodos (M)			
		Primal	Dual	Out of Kilter	Cost Scaling
Pequeno	03	7,65	470,30	8,15	14,35
	06	11,86	>7200,00	8,60	14,86
	09	17,18	>7200,00	10,33	20,15
Médio	03	355,68	>7200,00	278,69	510,11
	06	534,74	>7200,00	291,64	397,30
	09	711,49	>7200,00	386,45	563,79
Grande	03	2731,43	>7200,00	1769,86	2965,40
	06	3965,83	>7200,00	1984,46	3048,34
	09	5297,70	>7200,00	2561,38	3892,49

Tabela 09. Experimento 2: tempos de CPU para problemas de circulação.

unesp

Fator	Nível	Tempo Médio de CPU
Classe	Transporte	1933,11
	Transporte baldeação	2197,54
	Circulação	1963,01
Dimensão	Pequeno	72,42
	Médio	2149,98
	Grande	3871,26
Nível de Capacitação	100%	1977,04
	80%	2033,70
	50%	2036,47
	20%	2038,29
	0%	2070,59
Método	Primal	588,06
	Dual	4883,02
	Out-of-Kilter	939,90
	Cost Scaling	1713,90

Tabela 10. Experimento 1: tempos médios de CPU para cada nível de fator.

Análise de Variância
Projeto Split-Split-Plot

Variável Dependente: Tempo de CPU

Testes de Hipóteses usando ANOVA MS para K*S como um termo de erro

Fonte	GL	Anova SS	Quadrado Médio	Valor F	Pr > F
K	2	2516392,15	1258196,08	1,12	0,4117
S	2	434204838,81	217102419,40	192,73	0,0001

Testes de Hipóteses usando ANOVA MS para K*S*C como um termo de erro

Fonte	GL	Anova SS	Quadrado Médio	Valor F	Pr > F
C	4	164463,00	41115,75	2,27	0,1069
K*C	8	160539,48	20067,43	1,11	0,4078
S*C	8	190414,05	23801,76	1,31	0,3047

Testes de Hipóteses usando ANOVA Residual MS como um termo de erro

Fonte	GL	Anova SS	Quadrado Médio	Valor F	Pr > F
M	3	517821092,10	172607030,70	7931,80	0,0001
K*M	6	18408638,90	3068106,48	140,99	0,0001
S*M	6	278333490,70	46388915,12	2131,71	0,0001
C*M	12	488538,11	40711,51	1,87	0,0476
K*S*M	12	21950427,22	1829202,27	84,06	0,0001

Observação: GL = Graus de Liberdade, SS = Soma de Quadrados, F = Valor calculado para testar as hipóteses que o grupo de médias para o efeito são iguais, Pr > F = valor da probabilidade de significância associada com o Valor de F.

Tabela 11. Experimento 1: Análise de Variância.

unesp

Fator	Nível	Tempo Médio de CPU
Classe	Transporte	2461,07
	Transporte baldeação	2497,66
	Circulação	2511,95
Dimensão	Pequeno	668,99
	Médio	2266,41
	Grande	4535,28
Número de limitantes de intervalos por arcos ("breakpoints")	03	2036,47
	06	2612,01
	09	2922,20
Método	Primal	843,50
	Dual	5673,59
	Out-of-Kilter	1460,42
	Cost-Scaling	1983,40

Tabela 12. Experimento 2: tempos médios de CPU para cada nível de fator.

Análise de Variância
Projeto Split-Split-Plot

Variável Dependente: Tempo de CPU

*Testes de Hipóteses usando ANOVA MS para K*S como um termo de erro*

Fonte	GL	Anova SS	Quadrado Médio	Valor F	Pr > F
K	2	495956293,63	247978146,81	0,01	0,9923
S	2	2717721440384,13	1358860720192,06	42,36	0,0020

*Testes de Hipóteses usando ANOVA MS para K*S*B como um termo de erro*

Fonte	GL	Anova SS	Quadrado Médio	Valor F	Pr > F
B	2	141468323000,91	70734161500,45	10,25	0,0062
K*B	4	21502662784,15	5375665696,04	0,78	0,5690
S*B	4	40357131665,48	10089282916,37	1,46	0,2994

Testes de Hipóteses usando ANOVA Residual Ms como um termo de erro

Fonte	GL	Anova SS	Quadrado Médio	Valor F	Pr > F
M	3	3823979298877,43	1274659766292,47	176,40	0,0001
K*M	6	323867122927,48	53977853821,25	7,47	0,0001
S*M	6	770725694182,09	128454282363,68	17,78	0,0001
B*M	6	39337883016,42	6556313836,07	0,91	0,5006
K*S*M	12	341988746794,25	28499062232,85	3,94	0,0007
S*B*M	12	266817726951,63	22234810579,30	3,08	0,0045

Observação: GL = Graus de Liberdade, SS = Soma dos Quadrados, F = Valor calculado para testar as hipóteses que o grupo de médias para o efeito são iguais, Pr > F = valor da probabilidade de significância associada com o Valor de F.

Tabela 13. Experimento 2: Análise de Variância.

unesp

Alfa = 0,05 GL= 48 MSE = 1,87E4 Valor Crítico = 3,420 Diferença Mínima Significante = 60,38			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível do Fator**
A	**2197,54**	**60**	**Transporte Baldeação**
B	**1963,01**	**60**	**Circulação**
B	**1933,11**	**60**	**Transporte**

(a) Fator: Classe.

Alfa = 0,05 GL = 48 MSE = 1,87E4 Valor Crítico = 3,420 Diferença Mínima Significante = 60,38			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível do Fator**
A	**3871,26**	**60**	**Grande**
B	**2149,98**	**60**	**Médio**
C	**72,42**	**60**	**Pequeno**

(b) Fator: Dimensão.

Alfa = 0,05 GL= 48 MSE = 1,87E4 Valor Crítico = 4,008 Diferença Mínima Significante = 91,35			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível de Fator**
A	**2070,59**	**36**	**0%**
B A	**2038,29**	**36**	**20%**
B A	**2036,47**	**36**	**50%**
B A	**2033,70**	**36**	**80%**
B	**1977,04**	**36**	**100%**

(c) Fator: Capacitação nos Arcos.

Alfa = 0,05 GL= 48 MSE = 1,87E4 Valor Crítico = 3,764 Diferença Mínima Significante = 76,73			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível do Fator**
A	**4883,02**	**45**	**Dual**
B	**1713,90**	**45**	**Cost-Scaling**
C	**939,90**	**45**	**Out-of-Kilter**
D	**588,06**	**45**	**Primal**

(d) Fator: Método Aplicado.

Observação: N = número de problemas, GL = Graus de Liberdade, MSE = Média dos Quadrados dos Erros.

Tabela 14. Experimento 1: Teste de Tukey.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Alfa = 0,05 GL = 24 MSE = 6,8187E5 Valor Crítico = 3,532 Diferença Mínima Significante = 486,05			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível de Fator**
A	**2511,95**	**36**	**Circulação**
A	**2497,66**	**36**	**Transporte Baldeação**
A	**2461,07**	**36**	**Transporte**

(a) Fator: Classe.

Alfa = 0,05 GL = 24 MSE = 6,8187E5 Valor Crítico = 3,532 Diferença Mínima Significante = 486,05			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível de Fator**
A	**4535,28**	**36**	**Grande**
B	**2266,41**	**36**	**Médio**
C	**668,99**	**36**	**Pequeno**

(b) Fator: Dimensão.

Alfa = 0,05 GL= 24 MSE = 6,8187E5 Valor Crítico = 3,532 Diferença Mínima Significante = 486,05			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível de Fator**
A	**2922,20**	**36**	**09**
B A	**2512,01**	**36**	**06**
B	**2036,47**	**36**	**03**

(c) Fator: Número de Limitantes de Intervalos de Fluxo por Arco.

Alfa = 0,05 GL= 24 MSE = 6,8187E5 Valor Crítico = 3,901 Diferença Mínima Significante = 619,97			
Agrupamento de Tukey	**Tempo Médio de CPU**	**N**	**Nível de Fator**
A	**5673,59**	**27**	**Dual**
B	**1983,40**	**27**	**Cost-Scaling**
C B	**1460,42**	**27**	**Out-of-Kilter**
C	**843,50**	**27**	**Primal**

(d) Fator: Método Aplicado.

Observação: N = Número de Problemas, GL = Graus de Liberdade, MSE = Média dos Quadrados dos Erros.

Tabela 15. Experimento 2: Teste de Tukey.

unesp

CAPÍTULO 4

CONCLUSÕES E COMENTÁRIOS FINAIS

4.1. INTRODUÇÃO

Neste capítulo, inicialmente, apresenta-se uma análise dos resultados obtidos nos experimentos efetuados, particularmente no que se refere as aplicações dos testes estatísticos de Análise de Variância e Teste de Tukey para a Comparação de Várias Médias. Na seqüência, são apresentadas as conclusões acerca dos desempenhos dos algoritmos, frente as alterações possíveis nos valores dos conjuntos de quatro fatores admitidos para cada um dos experimentos realizados, e é feita uma avaliação se os objetivos propostos foram satisfatoriamente atingidos.

A seguir, comenta-se sobre algumas possibilidades promissoras para a continuidade da pesquisa em Programação em Redes Lineares por Partes, e, finalmente, são feitas considerações gerais sobre o trabalho realizado.

4.2. ANÁLISE DOS RESULTADOS

Na realização dos Experimentos 1 e 2, descritos no Capítulo 3, durante a fase de aplicação das análises estatísticas, foram feitas algumas investigações preliminares com o objetivo

unesp

de se obter informações adicionais sobre quais das interações entre os quatro fatores envolvidos, que foram o objeto do estudo para cada experimento, seriam significativas.

Neste sentido, antes do uso da metodologia "split plot", foi efetuada uma Análise de Variância exploratória, considerando um planejamento de experimentos do tipo "full factorial" para ambos os experimentos de interesse; para detalhamento teórico desta modalidade de planejamento de experimentos consultar Montgomery [84], e SAS [110].

Como uma ilustração dos resultados obtidos por tal procedimento, os resultados da Análise de Variância, realizada pelo módulo StaTest do sistema de gerenciamento NplSys e relacionada com o Experimento 1, estão dispostos na Tabela 16 abaixo.

Análise de Variância
Projeto Full Factorial

Variável Dependente: Tempo de CPU

Fonte	GL	ANOVA SS	Quadrado Médio	Valor F	Pr > F
Modelo	131	1280226077,44	9772718,15	522,61	0,0001
Erro	48	897593,67	18699,87		
Total Corrigido	179	1281123671,11			

Fonte	GL	ANOVA SS	Quadrado Médio	Valor F	Pr > F
K	2	2516392,15	1258196,08	67,28	0,0001
S	2	434204838,81	217102419,40	11609,84	0,0001
C	4	164463,00	41115,75	2,20	0,0832
M	3	517821092,10	172607030,70	9230,39	0,0001
K*S	4	4505899,07	1126474,77	60,24	0,0001
K*C	8	160539,48	20067,43	1,07	0,3975
S*C	8	190414,05	23801,76	1,27	0,2799
K*S*C	16	289843,74	18115,23	0,97	0,5034
K*M	6	18408638,90	3068106,48	164,07	0,0001
S*M	6	278333490,70	46388915,12	2480,71	0,0001
K*S*M	12	21950427,22	1829202,27	97,82	0,0001
C*M	12	488538,11	40711,51	2,18	0,0285
K*C*M	24	437006,51	18208,60	0,97	0,5140
S*C*M	24	754493,61	31437,23	1,68	0,0626

Tabela 16. Experimento 1: Análise de Variância Exploratória

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

O que pôde ser constatado nestas análises de variância preliminares, que consideraram como o resíduo as interações dos quatro fatores K*S*C*M e K*S*B*M, respectivamente, para o primeiro e segundo experimentos, é que algumas interações entre três fatores, dadas por K*C*M, S*C*M para o primeiro experimento, e por K*B*M para o segundo experimento, não foram estatisticamente significantes.

A decisão tomada foi a de, então, considerar estes termos correspondentes às interações de três fatores, acima citados, como incorporados ao resíduo de um modelo estatístico revisado, semelhante ao descrito na seção 3.2.3, para, em seguida, aplicar a abordagem pelo planejamento de experimentos do tipo "split plot" conforme já apresentado.

Com respeito ao Experimento 1, a análise pelo planejamento "split plot" envolveu o teste dos fatores plots principais, K e S, contra K*S; o fator sub-plot C e as interações entre dois fatores K*C e S*C contra K*S*C, e finalmente, o fator sub-sub-plot M, bem como as interações entre fatores K*M, S*M, C*M e K*S*M, contra o resíduo do modelo. Os resultados de tal análise, produzidos pelo StaTest, podem ser vistos na Tabela 11 do capítulo anterior.

Na mesma linha, para o Experimento 2, cujos resultados estão colocados na Tabela 13, a análise pelo procedimento "split plot", considerou os fatores plots principais K e S contra a interação K*S, o fator sub-plot B, e as interações K*B e S*B, contra a interação K*S*B, e por último, o fator sub-sub-plot M, e as interações K*M, S*M, B*M, K*S*M, e S*B*M, contra o resíduo do modelo.

Observando-se as Tabelas 11 e 13, para o nível de significância adotado de $\alpha = 5\%$, conclui-se, sem nenhuma surpresa para ambos os experimentos, que os fatores dimensão do problema (S = Pequeno, Médio, e Grande), e método utilizado (M = Primal Simplex, Dual Simplex, Out-of-Kilter, e Cost Scaling) são fatores altamente significativos na determinação dos tempos de CPU gastos na execução de cada algoritmo, pois apresentam o valor da probabilidade

unesp

de significância associada à estatística F, calculada na análise de variância, e denotadas por Pr > F nas Tabelas 11 e13, da ordem de 0,0001, sendo inferior, pois, ao valor de α adotado.

Com respeito ao Experimento 1, constata-se que nem o fator classe do problema (K = Transporte, Transporte com Baldeação, e Circulação), nem o fator associado ao nível de capacitação apresentado pelos arcos (C = 100%, 80%, 50%, 20%, e 0%), podem ser admitidos como significantes na influência sobre os tempos de execução dos algoritmos. De fato, a partir da Tabela 11, os valores de Pr > F, respectivamente, para os fatores K e C, são 0,4117 e 0,1069, bem acima do valor de α considerado.

Já para o Experimento 2, percebe-se que o fator classe também não é significante, pois Pr > F = 0,9923, mas o fator sub-plot, correspondente ao número de "breakpoints" por arco (B = 03, 06, e 09), tem influência comprovada nos tempos gastos pelos algoritmos, na solução de problemas em redes lineares por partes, pois neste caso a Tabela 13 revela que Pr > F = 0,0062.

Voltando à Tabela 11, uma constatação importante é que as interações envolvendo K*M e S*M são altamente significantes, ambos com Pr > F = 0,0001, ou seja, embora o fator classe do problema não seja um fator significativo individualmente, pois neste caso tem-se Pr > F = 0,4117, a classe quando em interação com o fator método passa a sê-lo.

Isto é comprovado por uma inspeção nas Tabelas 04, 05, e 06, que evidencia que o comportamento relativo dos algoritmos Primal Simplex e Out-of-Kilter, quando aplicados a problemas de Circulação, é diferente daquele observado, para os mesmos dois algoritmos, quando aplicados às outras duas classes de problemas estudados, problema de Transporte, e problema de Transporte com Baldeação.

O mesmo fato se repete para o Experimento 2, pois têm-se para o fator K, e para K*M e S*M , respectivamente, os valores de Pr > F dados por 0,9923, 0,0001, e 0,0001.

Objetivando a obtenção de mais informações, aplicou-se o Teste de Tukey para os dados dos dois experimentos, o que permitiu uma análise complementar dos dados, em termos das

unesp

comparações múltiplas entre os níveis de cada fator, considerando, novamente, o nível de significância dos testes como sendo $\alpha = 5\%$.

Nas Tabelas 14 e 15, estão dispostos os relatórios de saída oriundos da aplicação do Teste de Tukey, para os Experimentos 1 e 2, respectivamente. Observe-se que, nestas tabelas, cada valor de nível de cada fator é ranqueado relativamente aos demais por meio do uso de um código de letras, A, B, C, ..., associadas com tempos de CPU decrescentes. No caso de níveis de fatores que apresentem a mesma letra de código, a interpretação é que estes valores de níveis não são significantemente diferentes uns dos outros, para o valor de α considerado.

Assim, percebe-se que, para o Experimento 1, cada um dos valores de níveis do fator dimensão do problema são significativamente diferentes entre si, como indicado na Tabela 14 (b), o mesmo ocorrendo com os valores dos níveis do fator método utilizado, conforme Tabela 14 (d).

No que concerne aos dois outros fatores, classe (conforme exposto na Tabela 14 (a)) e a capacitação nos arcos (conforme exposto na Tabela 14 (c)), que não foram considerados de significância pela análise de variância anteriormente feita, constata-se que, o comportamento das redes totalmente capacitadas (C = 100%, e letra de código B associada) foi significantemente diferente daquele mostrado pelas redes totalmente sem capacitação (C = 0%, e letra de código A associada).

Além disto, o comportamento dos problemas de Transporte com Baldeação, que apresentou letra de código A, foi significantemente diverso do comportamento das outras duas classes, ou seja, dos problemas de Transporte e de Circulação, com letra de código B correspondentes a ambos. Uma consulta às Tabelas 04, 05, e 06 aponta que esta última diferenciação de comportamentos entre classes de problemas é devido ao desempenho particularmente fraco do método Cost Scaling nos problemas de Transporte com Baldeação.

Já para o Experimento 2, consultando-se a Tabela 15 (b), percebe-se que, todos os valores dos níveis do fator dimensão apresentam diferenças significativas, pois estão associados a

unesp

um ranqueamento com três letras de códigos distintas (A, B, C) para os três níveis admitidos. Já o fator classe tem os valores de seus três níveis totalmente indistingüiveis, com a Tabela 15 (a) apresentando a mesma letra de código A para todos eles,

Adicionalmente, conforme exposto na Tabela 15 (c); pode-se perceber que há uma diferença significativa entre os casos em que o número de "breakpoints" é 09 (com letra de código A associada) e 03 (com letra de código B associada).

Finalmente, também pode-se efetuar um ranqueamento entre os valores dos níveis do fator método usado, que classifica, por ordem decrescente de tempo de CPU, o método Dual Simplex (com letra de código A), o método Cost Scaling (com letra de código B), seguidos pelos métodos Cost Scaling e Primal Simplex (ambos com a letra de código C).

Deste modo, ainda permanecem indistingüiveis as diferenças que possam existir entre os algoritmos Primal Simplex e o Out-of-Kilter, bem como entre os casos em que o número de "breakpoints" é 03 (com letra de código B) ou 06 (com letras de código B e A), e também entre os casos em que o número de "breakpopints é 06 ou 09 (com letra de código A).

Feitas estas análises, passa-se, na próxima seção, a tentar elencar as diversas conclusões que se pode obter a partir do que foi constatado aqui.

4.3. VERIFICAÇÃO DOS OBJETIVOS

Este trabalho busca a obtenção de respostas para questões relativas aos desempenhos dos quatro algoritmos implementados, especializados para redes lineares por partes, frente a uma série de categorias de problemas, definidas pelas combinações de valores possíveis para os níveis dos fatores estudados, que foram, respectivamente, o conjunto de fatores = {classe, dimensão, capacitação nos arcos, e método usado} e o conjunto de fatores = {classe, dimensão, número de "breakpoints" nos arcos, e método usado}, para os Experimentos 1 e 2.

Considerando os resultados da seção 3.3, e as observações feitas na seção 4.2, pode-se estabelecer as seguintes conclusões, as quais vem ao encontro da satisfação dos objetivos propostos:

Conclusão A: Não há um algoritmo que seja dominante sobre os demais algoritmos em todas as circunstâncias que foram estudadas.

Conclusão B: O método Dual Simplex é o que apresenta, sistematicamente, o pior desempenho em todas as circunstâncias analisadas, sendo seguido pelo Cost Scaling neste aspecto.

Conclusão C: Considerando os fatores estudados no Experimento 1 (em que o número de "breakpoints" nos arcos é fixo), o desempenho do algoritmo Primal Simplex domina os demais, sendo seguido, em ordem decrescente do tempo de CPU, exigido na solução das redes lineares por partes, pelos algoritmos Out-of-Kilter, Cost Scaling, e, por último, o Dual Simplex.

Conclusão D: Considerando os fatores estudados no Experimento 2 (em que o nível de capacitação nos arcos é fixo), há uma predominância clara de todos os métodos sobre o Dual Simplex, o mesmo ocorrendo com Algoritmo Primal Simplex sobre o Cost Scaling, não havendo, porém evidências estatísticas de vantagens, em geral, do Primal Simplex com relação ao Out-of-Kilter .

Conclusão E: O método Primal Simplex apresenta o melhor desempenho dentre todos os algoritmos, nos casos de problemas de Transporte e problemas de Transporte com Baldeação, para quaisquer combinações de valores dos outros fatores, dimensão, capacitação (ou número de "breakpoints").

Conclusão F: O método Out-of-Kilter tem o melhor desempenho dentre todos os algoritmos no caso de problemas de Circulação, para quaisquer combinações de valores dos

unesp

outros fatores, dimensão, capacitação, e número de "breakpoints", a menos da situação de problemas de dimensão pequena .

Conclusão G: O método Primal Simplex tem o seu melhor desempenho, pela ordem decrescente de tempo de CPU, para os problemas de Transporte, de Transporte com Baldeação, e de Circulação.

Conclusão H: O método Out-of-Kilter tem o seu melhor desempenho, pela ordem decrescente de tempo de CPU, para os problemas de Circulação, de Transporte com Baldeação, e de Transporte.

Conclusão I: O método Cost Scaling tem o seu melhor desempenho, pela ordem decrescente de tempo de CPU, para os problemas de Circulação, de Transporte, e de Transporte com Baldeação.

Conclusão J: Todos os algoritmos tem mais dificuldade para resolver, pela ordem decrescente de tempo de CPU, problemas com dimensão Grande, Média, e Pequena.

Conclusão K: Não há evidências sobre a influência do nível de capacitação nos arcos no desempenho dos métodos, constata-se, apenas, que os problemas sem capacitação (C = 0%) apresentam mais dificuldade para serem resolvidos pelos algoritmos que aqueles problemas com capacitação completa em todos os arcos (C = 100%).

Conclusão L: Todos os algoritmos são sensíveis ao aumento do número de "breakpoints" por arco, com os problemas que tem este número igual a 09 apresentando maior dificuldade de resolução por parte de todos os métodos.

Com estas conclusões estabelecidas, espera-se ter respondido a contento as questões colocadas na seção 3.2.2, bem como ter fornecido subsídios relevantes para pesquisadores que desejem utilizar modelos em redes lineares por partes nos seus trabalhos.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

4.4. DIREÇÕES DE PESQUISAS

Existem algumas linhas de pesquisa que podem dar continuidade a este trabalho, e que merecem ser destacadas pelo seu interesse científico intrínseco, e pelo avanço que podem proporcionar para a área de Programação em Redes Lineares por Partes - PRLP. Passa-se a comentar algumas delas, que inclusive já vêm sendo objeto de estudo por parte do grupo de pesquisa ao qual pertence o autor deste trabalho.

Um desdobramento natural deste trabalho, é a consideração de novos algoritmos na comparação com o Primal Simplex e o Out-of-Kilter, que se destacaram nos testes executados pelo seu bom desempenho nos testes realizados. Neste sentido, pode-se imaginar a implementação de algoritmos do tipo Primal-Dual, como os apresentados por Papadimitriou e Steiglitz [89], ou outros algoritmos com complexidade polinomial, como os descritos em Ahuja et al. [4], ou ainda Métodos de Pontos Interiores, ver Wang e Melhotra [120] .

Outra área interessante de pesquisa é a da programação de testes computacionais envolvendo a comparação de algoritmos especializados para a PRLP com algoritmos padrões para resolução de redes lineares (como o NETFLO, que foi proposto por Kennington e Helgason [65]).

Nesta situação, conforme executado por Sun et al. [114] em relação a problemas de Transporte com Baldeação Estocásticos, os algoritmos padrões seriam aplicados às redes lineares equivalentes, obtidas a partir de alguma transformação feita nas redes lineares por partes, as quais seriam, por sua vez, resolvidas pelos algoritmos especializados envolvidos na comparação.

O desenvolvimento de procedimentos de reotimização para modelos em redes lineares por partes, também é de interesse da área de PRLP, além do que, conforme comentado por Ali et al. [5], este tópico não vem tendo a devida atenção, mesmo para a Programação em Redes Lineares.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Finalmente, pode-se citar como de relevância, o desenvolvimento de estudos considerando o conceito de "*representative operation counts*" (ver Ahuja e Orlin [2], e Ahuja et al. [4]), que podem complementar a análise dos algoritmos baseada nos tempos de CPU, como a que foi descrita neste trabalho.

A idéia central deste enfoque é identificar as "bottleneck operations", ou *operações de gargalo*, isto é, aquelas operações fundamentais que se repetem muitas vezes na aplicação do algoritmo, e cuja melhoria significará ganhos na eficiência computacional do algoritmo. Algumas vantagens de se utilizar este procedimento, são:

(a) Possibilidade de comparação de algoritmos executados em máquinas diferentes, até mesmo quando formulados em diferentes linguagens de programação;

(b) Possibilidade de se calcular o valor de limitantes, inferior e superior, para a taxa de crescimento assintótico do tempo de execução do algoritmo, como uma função da dimensão do problema;

(c) Possibilidade de se usar técnicas estatísticas para estimar o tempo de CPU gasto pelo algoritmo, numa dada máquina, como uma função linear das "representative operation counts", o que é conhecido na literatura como o *tempo de CPU virtual*. Este tempo virtual permite que situações em que os experimentos são feitos em computadores diferentes, tenham suas estimativas dos tempos de execução comparáveis, como se tivessem sido obtidos na mesma máquina. Ahuja et al. [4] apresentam um exemplo completo analisando, através deste enfoque de "representative operation counts", o método simplex para redes lineares.

4.5. COMENTÁRIOS FINAIS

Pretende-se nesta seção fazer o destaque de certos aspectos relevantes do trabalho realizado, e apresentar alguns comentários de caráter geral.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Inicialmente, deve-se ressaltar que um objetivo implícito da pesquisa foi o de chamar a atenção para as alternativas existentes para o enfoque de modelos de Programação Linear por Partes - PLP, conforme exposto no Capítulo 1, e particularmente sobre as vantagens do tratamento direto destes modelos por técnicas especializadas para a PLP, contra o uso de transformações que resultem em modelos equivalentes de Programação Linear de maior porte.

Evidentemente, por via de conseqüência do desenvolvimento do trabalho outras metas foram atingidas, além daquelas já comentadas. Nesta linha, espera-se que os resultados da extensa pesquisa bibliográfica realizada venha a proporcionar aos interessados, tanto na PLP em geral, como na Programação em Redes Lineares por Partes - PRLP, uma visão privilegiada do estado da arte destas importantes áreas da Programação Matemática, colaborando, assim, para futuros desenvolvimentos científicos correlatos.

Também a preocupação em se fundamentar na Estatística, e particularmente nos conceitos de Planejamento de Experimentos, tanto o projeto do estudo computacional, como a análise dos desempenhos relativos observados dos algoritmos, procurou evidenciar a necessidade do uso da metodologia científica na condução de testes com softwares de Programação Matemática. De fato, a ausência de um procedimento sistemático, estatisticamente válido, pode resultar em conclusões sem base científica sobre os desempenhos dos métodos testados.

Outro subproduto que resultou deste trabalho é o oferecimento de um novo programa gerador de redes (NplGen), com grande potencialidade de aplicação em estudos correlatos; bem como o próprio sistema de gerenciamento dos testes (NplSys), que foi implementado para se efetuar o controle, a execução dos testes computacionais, e a coordenação da correspondente análise estatística. No Apêndice 1, podem ser encontrados detalhes valiosos para a utilização deste sistema, sua versatilidade, e potencial. Este sistema pode ser cedido, sem ônus, a pesquisadores que tenham interesse no seu aproveitamento, bastando contatar o autor deste trabalho.

unesp

Também merece ser evidenciado que esta pesquisa só pode ser concretizada, graças ao suporte oferecido pelo CNPq - Conselho Nacional de Pesquisa e Desenvolvimento Tecnológico, através da concessão de bolsa de pesquisa nos períodos de 1988 a 1994, e 1995 a 1997, e pela FAPESP - Fundação de Apoio à Pesquisa do Estado de São Paulo, por meio da concessão de bolsa para o desenvolvimento de pós-doutoramento na Inglaterra em 1994. Naturalmente, sem a concordância, o incentivo, e o apoio dos colegas do Departamento de Produção do Campus de Guaratinguetá - UNESP, no qual o autor está lotado, este trabalho não se efetivaria.

Outros pesquisadores também colaboraram, no decorrer da pesquisa, e que devem ser mencionados, como o Prof. Edson Luiz França Senne, do Departamento de Matemática do Campus de Guaratinguetá - UNESP, particularmente na implementação do NplSys, o Prof. Kenneth Darby-Dowman, do Departamento de Matemática e Estatística - Brunel University - Inglaterra, no apoio à análise estatística realizada, e a Profa. Arlene Fortunato Machado, do IMECC - UNICAMP, na discussão e na proposição do algoritmo Cost Scaling.

Finalmente, deve-se esclarecer que este trabalho teve o suporte, e a participação decisiva, do Prof. Clovis Perin do IMECC - UNICAMP, na discussão contínua sobre os rumos da pesquisa e na proposição dos algoritmos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Ahlfeld, D. P., R. S. Dembo, J. M. Mulvey, and S. A. Zenios. 1987. Nonlinear Programming on Generalized Networks. *Association for Computing Machinery Transactions on Mathematical Software* 13, No. 4, 350-367.

[2] Ahuja, R. K., and J. B. Orlin. 1993. Use of Representative Operations Counts in Computational Testing of Algorithms. Working Paper, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

[3] Ahuja, R. K., J. L. Batra, and S. K. Gupta. 1984. A Parametric Algorithm for the Convex Cost Network Flow and Related Problems. *European Journal of Operational Research* 16, No. 2, 222-235.

[4] Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Englewood Cliffs, NJ, Prentice-Hall.

[5] Ali, A. I., E. P. Allen, R. S. Barr, and J. L. Kennington. 1986. Reoptimization Procedures for Bounded Variable Primal Simplex Network Algorithms. European Journal of Operational Research 23, 256-263.

[6] Ali, A. I., R. Padman, and H. Thiagarajan. 1989. Dual Algorithms for Pure Network Problems. *Operations Research* 37, No. 1, 159-171.

[7] Amini, M. M., and M. Racer. 1994. A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem. *Management Science* 40, No. 7, 868-890.

[8] Amini, M. M., and R. S. Barr. 1989. Network Reoptimization Testing System (NTRS): Users Guide. Working Paper, Department of Management Information Systems and Decision Sciences, The Fogelman College of Business and Economics, Memphis State University, Memphis - Tennessee, USA.

[9] Amini, M. M., and R. S. Barr. 1990. Network Reoptimization: A Computational Comparison of Algorithmic Alternatives. Technical Report 90-MISDS-1, Department of Management Information

Systems and Decision Sciences, The Fogelman College of Business and Economics, Memphis State University, Memphis - Tennessee, USA.

[10] Amini, M. M., and R. S. Barr. 1993. Network Reoptimization Algorithms: A Statistically Designed Comparison. *ORSA Journal on Computing* 5, No. 4, 395-409.

[11] Anderson, V. L., and R. A. McLean. 1974. *Design of Experiments: a Realistic Approach*. NY, Marcel Dekker.

[12] Anthonisse, J. M., K. M. van Hee, and J. K. Lenstra. 1988. Resource-Constrained Project Scheduling: An International Exercise in DSS. *Decision Support Systems* 3, 249-257.

[13] Barr, R. S., F. Glover, and D. Klingman. 1977. The Alternating Path Basis Algorithm for the Assignment Problem. *Mathematical Programming* 13, No. 1, 1-13.

[14] Barras, J., S. Alec, C. Pasche, P. A. Chamorel, A. J. Germond, and D. de Werra. 1987. Network Simplex Method Applied to AC Load-Flow Calculation. *Institute of Electrical and Electronics Engineers Transactions on Power Systems* Vol PWRS-2, No. 1, 197-203.

[15] Barrodale, I., and F. D. K. Roberts. 1973. An Improved Algorithm for Discrete L_1 Linear Approximation. *SIAM Journal on Numerical Analysis* 10, 839-848.

[16] Bartels, R. H. 1980. A Penalty Linear Programming Method Using Reduced-Gradient Basis-Exchange Techniques. *Linear Algebra and Its Applications* 29, 17-32.

[17] Beale, E. M. L. 1959. An Algorithm for Solving the Transportation Problem when Shipping Cost over each Route is Convex. *Naval Research Logistics Quarterly* 6, 43-56.

[18] Bland, R. G., and D. L. Jensen. 1992. On the Computational Behaviour of a Polynomial-Time Network Flow Algorithm. *Mathematical Programming* 54, No. 1, 1-39.

[19] Charnes, A., and C. E. Lemke. 1954. Minimization of Non-Linear Separable Convex Functionals. *Naval Research Logistics Quarterly* 1, 301-312.

[20] Charnes, A., and W. W. Cooper. 1957. Nonlinear Power of Adjacent Extreme Point Methods in Linear Programming. *Econometrica* 25, 132-153.

unesp

[21] Charret, D. E. 1970. Plastic Analysis and Optimum Design of Slabs and Frames. Special Report S2/1970, Civil Engineering, Monash University, Clayton - Victoria, Australia.

[22] Collins, M., L. Cooper, R. Helgason, J. Kennington, and L. LeBlanc. 1978. Solving the Pipe Network Analysis Problem Using Optimization Techniques. *Management Science* 24, No. 7, 747-760.

[23] Conn, A. R. 1976. Linear Programming via a Nondifferentiable Penalty Function. *SIAM Journal on Numerical Analysis* 13, No. 1, 145-154.

[24] Cooper, L., and L. J. LeBlanc. 1977. Stochastic Transportation Problems and Other Networks Related Convex Problems. *Naval Research Logistics Quarterly* 24, No. 2, 327-337.

[25] Crowder, H. P., and P. B. Saunders. 1980. Results of a Survey on Mathematical Programming Performance Indicators. *COAL Newsletter*, January, 2-6.

[26] Crowder, H. P., R. S. Dembo, and J. M. Mulvey. 1978. Reporting Computational Experiments in Mathematical Programming. *Mathematical Programming* 15, No. 3, 316-329.

[27] Cunningham, W. H. 1976. A Network Simplex Method. *Mathematical Programming* 11, No. 2, 105-116.

[28] Cunningham, W. H. 1979. Theoretical Properties of the Network Simplex Method. *Mathematics of Operations Research* 4, No. 2, 196-208.

[29] Dantzig, G. 1956. Recent Advances in Linear Programming. *Management Science* 2, No. 2, 131-144.

[30] Dantzig, G. B. 1955. Linear Programming under Uncertainty. *Management Science* 1, No. 3, 197-206.

[31] Dantzig, G., S. Johnson, and W. White. 1958. A Linear Programming Approach to the Chemical Equilibrium Problem. *Management Science* 5, No. 1, 38-43.

[32] Davies, M.. 1967. Linear Approximation Using the Criterion of Least Total Deviations. *Journal of the Royal Statistical Society* B 29, 101-109.

unesp

[33] Dembo, R. S., A. Chiarri, J. G. Martin, L. Paradinas. 1990. Managing Hidroeléctrica Española's Hydroelectric Power System. *Interfaces* 20, No. 1, 115-135.

[34] De Wolf, D., O. J. de Bisthoven, and Y. Smeers. 1991. The Simplex Algorithm Extended to Piecewise Linearly Constrained Problems I: the Method and an Implementation. Discussion Paper No. 9119, . Center for Operations Research & Econometrics, Universite Catholique de Louvain, Belgium.

[35] Dongarra, J. J., and E. Grosse. 1987. Distribution of Mathematical Software via Electronic Mail. *Communications of the Association for Computing Machinery* 30, No. 5, 403-407.

[36] Edmonds, J., and R. M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the Association for Computing Machinery* 19, No. 2, 248-264.

[37] Ferguson, A. R., and G. B. Dantzig. 1957. The Allocation of Aircrafts to Routes: An Example of Linear Programming under Uncertain Demand. *Management Science* 3, No. 1, 45-73.

[38] Fernandes, J. F. R. 1979. Piecewise-Linear Programming: A Large-Scale Problem Solving by Decomposition Techniques. Ph.D. Thesis, Electrical Engineering College, State University of Campinas, Campinas - SP, Brazil.

[39] Ferreira, E. P. 1979. Piecewise-Linear Programming: A Dual-Primal Method. Master Dissertation, Electrical Engineering College, State University of Campinas, Campinas - SP, Brazil.

[40] Fourer, R. 1985. A Simplex Algorithm for Piecewise-Linear Programming I: Derivation and Proof. *Mathematical Programming* 33, No. 2, 204-233.

[41] Fourer, R. 1988. A Simplex Algorithm for Piecewise-Linear Programming II: Finiteness, Feasibility and Degeneracy. *Mathematical Programming* 41, No. 3, 281-315.

[42] Fourer, R. 1992. A Simplex Algorithm for Piecewise-Linear Programming III: Computational Analysis and Applications. *Mathematical Programming* 53, No. 2, 213-235.

[43] Fourer, R., and R. E. Marsten. 1992. Solving Piecewise-Linear Programs: Experiments with a Simplex Approach. *ORSA Journal on Computing* 4, No. 1, 15-31.

unesp

[44] França, P. M., J. F. R. Fernandes, and H. M. F. Tavares. 1987. Telephonic Network Expansion. *Controle & Automação* 1, No. 3.

[45] Garcia, A. S. 1978. A Dual-Simplex Method for Piecewise-Linear Programming. Master Dissertation, Institute of Mathematics, Statistics and Computer Sciences, State University of Campinas, Campinas - SP, Brazil.

[46] Geoffrion, A. M. 1977. Objective Function Approximations in Mathematical Programming. *Mathematical Programming* 13, No. 1, 23-37.

[47] Glover, F., D. Klingman, and D. T. Phillips. 1992. *Networks Models in Optimization and Their Applications in Practice*. NY, John Wiley & Sons.

[48] Glover, F., D. Klingman, and N. Phillips. 1990. Netform Modeling and Applications. *Interfaces* 20, No. 4, 7-27.

[49] Goldberg, A. V., and R. Tarjan. 1990. Finding Minimum-Cost Circulations by Successive Approximation. *Mathematics of Operations Research* 15, No. 3, 430-466.

[50] Golstein, E. G. 1960. A Certain Class of Nonlinear Extremum Problems. *Soviet Mathematics* 4, 863-866.

[51] Golstein, E. G., and D. Youdine. 1973. *Problèmes Particuliers de la Programmation Lineaire.* Moscow, Editions Mir.

[52] Greenberg, H. J. 1990. Computational Testing: Why, How and How Much. *ORSA Journal on Computing* 2, No. 1, 94-97.

[53] Greenberg, H. J., R. H. F. Jackson, P. E. Gill, J. L. Kreuser, F. A. Lootsma, and J. A. Tomlin. 1975. NOTICE: The Working Committee on Algorithms. *Mathematical Programming* 9, No. 1, 135-135.

[54] Güder, F., and J. G. Morris. 1994. Optimal Objective Function Approximation for Separable Convex Quadratic Programming. *Mathematical Programming* 67, No. 1, 133-142.

unesp

[55] Gunn, E. A. .1981. An Unconstrained Dual for Linear Programming and its Solution by Piecewise Linear Optimization. *Utilitas Mathematica* 20, 5-19.

[56] Ho, J. K. 1985. Relationship Among Linear Formulation of Separable Convex Piecewise-Linear Programs. In: *Mathematical Programming Study* 24, 126-140, edited by R. W. Cottle, Amsterdam - The Netherlands, North-Holland Publishing Company.

[57] Hochbaum, D. S., and J. G. Shanthikumar. 1990. Convex Separable Optimization Is Not Much Harder than Linear Optimization. *Journal of the Association for Computing Machinery* 37, No. 4, 843-862.

[58] Hoffman, K. L., and R. H. F. Jackson. 1982. In Pursuit of a Methodology for Testing Mathematical Programming Software. In: *Evaluating Mathematical Programming Techniques, Lecture Notes in Economics and Mathematical Systems* Vol. 199, 177-199, edited by J. M. Mulvey. NY, Springer-Verlag.

[59] Ignizio, J. P. 1971. On the Establishment of Standards for Comparing Algorithms Performance. *Interfaces* 2, No. 1, 8-11.

[60] Ignizio, J. P. 1973. Validating Claims for Algorithms Proposed for Publication. *Operations Research* 21, No. 3, 852-854.

[61] Jackson, R. H. F. , P. T. Boggs, S. G. Nash, and S. Powell. 1991. Guidelines for Reporting Results of Computational Experiments. Report of the Ad Hoc Committee. *Mathematical Programming* 49, No. 3, 413-425.

[62] Jackson, R. H. F., and J. M. Mulvey. 1978. A Critical Review of Comparisons of Mathematical Programming Algorithms and Software (1953-1977). *Journal of Research of the National Bureau of Standards* 83, No. 6, 563-584.

[63] Kamesam, P. V., and R. R. Meyer. 1984. Multipoint Methods for Separable Nonlinear Networks. In: *Mathematical Programming Study* 22, 185-205.

[64] Kao, C. Y., and R. R. Meyer. 1981. Secant Approximation Methods for Convex Optimization. In: *Mathematical Programming Study* 14, 143-162, edited by H. Hönig, B. Korte, and K. Ritter, Amsterdam - The Netherlands, North-Holland Publishing Company.

[65] Kennington, J. L., and R. V. Helgason. 1980. *Algorithms for Network Programming*. NY., John Wiley & Sons.

[66] Klein, M.. 1983. A Transportation Model for Production Planning with Convex Costs. *IIE Transactions* 15, No. 3, 272-274.

[67] Klingman, D., A. Napier, and J. Stutz. 1974. NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. *Management Science* 20, No. 5, 814-821.

[68] LeBlanc, L. J. 1976. The Use of Large Scale Mathematical Programming Models in Transportation Systems. *Transportation Research* 10, No. 6, 419-421.

[69] Levner, E. V., and A. S. Nemirosvsky. 1994. A Network Flow Algorithm for Just-in-Time Project Scheduling. *European Journal of Operational Research* 79, 167-175.

[70] Marins, F. A. S. 1987. Studies on Network Piecewise-Linear Programs. Ph.D. Thesis, Electrical Engineering College, State University of Campinas, Campinas - SP, Brazil.

[71] Marins, F. A. S., and C. Perin 1988. Strong Feasibility in Network Piecewise-Linear Programs. *Proceedings of the IV Latin-Iberian-American Congress on Operations Research and System Engineering*, 364-383, Rio de Janeiro - RJ, Brazil.

[72] Marins, F. A. S., and C. Perin 1989. An Out-of-Kilter Algorithm for Network Piecewise-Linear Programs. *Proceedings of the XII National Congress on Computational and Applied Mathematics*, 126-130, São José do Rio Preto - SP, Brazil.

[73] Marins, F. A. S., and C. Perin 1990. A Dual Algorithm for Network Piecewise-Linear Programs. Presented at the V Latin-Iberian-American Congress on Operations Research and System Engineering, Buenos Aires, Argentina.

unesp

[74] Marins, F. A. S., and C. Perin. 1991. Computational Experience with a Dual Algorithm for Network Piecewise-Linear Programs. *Proceedings of the XIII World Congress on Computational and Applied Mathematics*, 197-198, Dublin, Ireland.

[75] Marins, F. A. S., C. Perin, and A. F. Machado. 1992. A Strongly Polynomial Algorithm for Network Piecewise-Linear Programs. Presented at the XXIV Brazilian Symposium on Operations Research, Salvador - BA, Brazil.

[76] Marins, F. A. S., C. Perin, and A. F. Machado. 1993. An O(n.m'.log n.min(log nC, m'logn)) Algorithm for Network Piecewise-Linear Programs. *Proceedings of the XXV Brazilian Symposium on Operations Research*, 118-122, Campinas - SP, Brazil.

[77] Marins, F. A. S., K. Darby-Dowman, E. F. Senne, A.F. Machado, and C. Perin. 1994. Algorithms for Network Piecewise-Linear Programs: A Statistically Designed Study. Technical Report TR/20/94, Department of Mathematics and Statistics, Brunel University, Uxbridge - Middlesex, UB8 3PH, England, UK.

[78] Markowitz, H. M., and A. F. Perold. 1981. Sparsity and Piecewise Linearity in Large Portfolio Optimization Problems. In: *Sparse Matrices and Their Uses*, 89-108, edited by I. S. Duff, Academic Press, NY.

[79] Maros I. 1993. Performance Evaluation of the MINET - Minimum Cost Netflow Solver. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 12, 199-217.

[80] Meyer, R. R. 1979. Two-Segment Separable Programming. 1979. *Management Science* 25, No. 4, 385-395.

[81] Meyer, R. R. 1982. Recursive Piecewise-Linear Approximation Methods for Nonlinear Networks. In: *Evaluating Mathematical Programming Techniques, Lecture Notes in Economics and Mathematical Systems* Vol. 199, 315-322, edited by J. M. Mulvey, Springer-Verlag, NY.

[82] MIPLIB - Mixed Integer Programming Library, available by an e-mail message containing "send catalog" to *softlib@rice.edu*.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

[83] Monma, C. L., and M. Segal. 1982. A Primal Algorithm for Finding Minimum-Cost Flows in Capacitated Networks with Applications. *The Bell System Technical Journal* 61, No. 6, 949-968.

[84] Montgomery, D. C. 1991. *Design and Analysis of Experiments, 3rd edition,* NY, John Wiley & Sons.

[85] Mulvey, J. M. 1989. Advances in Nonlinear Network Models and Algorithms. In: *Algorithms and Model Formulations in Mathematical Programming - NATO ASI Series* Vol F51, edited by Stein W. Wallace, Springer-Verlag, Berlin.

[86] Nakagawa, J. M. 1984. Telephonic System Planning: Assignment of Line Centers. Master Dissertation, Electrical Engineering College, State University of Campinas, Campinas - SP, Brazil.

[87] Orchard-Hays, W.. 1968. *Advanced Linear-Programming Computing Techniques*. McGraw-Hill, New York, NY.

[88] Orden, A., and V. Nalbandian. 1968. A Bidirectional Simplex Algorithm. *Journal of the Association for Computing Machinery* 15, No. 2, 221-235.

[89] Papadimitriou, C. H., and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

[90] Perold, A. F. 1984. Large-Scale Portfolio Optimization. *Management Science* 30, No. 10, 1143-1160.

[91] Premoli, A. 1986. Piecewise-Linear Programming: The Compact (CPLP) Algorithm. *Mathematical Programming* 36, No. 2, 210-227.

[92] Ravindran, A., D. T. Phillips, and J. J. Solberg. 1987. *Operations Research - Principles and Practice*. New York, NY, John Wiley & Sons.

[93] Ribeiro, R. V. 1980. Studies on Linear Programming. Ph.D. Thesis, Electrical Engineering College, State University of Campinas, Campinas - SP, Brazil.

[94] Robers, P. D., and A. Ben-Israel. 1969. An Interval Programming Algorithm for Discrete Linear L_1 Approximation Problems. *Journal of Approximation Theory* 2, 323-336.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

[95] Röck, H.. 1980. Scaling Techniques for Minimal Cost Network Flows. In: *Discrete Structures and Algorithms*, 181-191, edited by V. Page, Carl Hanser, Munich.

[96] Rockafellar, R. T. 1981. Monotropic Programming: Descent Algorithms and Duality. In: *Nonlinear Programming* 4, 327-366, edited by O. L. Mangasarian, R. R. Meyer, and S. M. Robinson. NY, Academic Press.

[97] Rockafellar, R. T. 1984. *Network Flows and Monotropic Optimization*. Wiley-Interscience, New York.

[98] Rosenthal, R. E. 1981. A Nonlinear Network Flow Algorithm for Maximization of Benefits in a Hydroelectric Power System. *Operations Research* 29, No. 4, 763-786.

[99] Rosvany, G. I. N. 1970. Concave Programming in Structural Optimization. *International Journal of Mechanical Sciences* 12, No. 2, 131-142.

[100] Rosvany, G. I. N. 1971. Concave Programming and Piecewise-Linear Programming. *International Journal for Numerical Methods in Engineering* 3, No. 1, 131-144.

[101] Sampford, M. R., and J. Taylor. 1959. Censored Observations in Randomized Block Experiments. *Journal of the Royal Statistical Society B* 21, 214-237.

[102] Sasson, A. M., and H. D. Merrill. 1974. Some Applications of Optimization Techniques to Power Systems Problems. *Proceedings of the Institute of Electrical and Electronics Engineers* 62, No. 7, 959-972.

[103] Senne, E. L. F., F. A. S. Marins, and C. Perin. 1994. NplSys - Network Piecewise-Linear Testing System: User's Guide. Technical Report TR19/94, Department of Mathematics and Statistics, Brunel University, Uxbridge - Middlesex, UB8 3PH, England, UK.

[104] Snyder, R. D. 1981. Programming with Piecewise-Linear Objective Functions. Working Paper 9/81, Department of Econometrics and Operations Research, Monash University, Clayton, Victoria, Australia.

[105] Snyder, R. D. 1984. Linear programming with Special Ordered Sets. *Journal of the Operational Research Society* 35, No. 1, 69-74.

[106] Soares Filho, S., B. P. F. Braga Jr., and J. G. L. Caneja. 1989. Mid Term Multiobjective Operation Planning of a Multireservoir System: The Piracicaba River Basin Case. Presented at International Symposium on Water Resource Systems Application, Canada.

[107] Souza, C. R. 1977. An Application of Piecewise-Linear Programming to Power Systems. Master Dissertation, Electrical Engineering College, State University of Campinas, Campinas - SP, Brazil.

[108] Spyropoulos, K., E. Kiountouzis, and A. Young. 1973. Discrete Approximation in the L_1 Norm. *The Computer Journal* 16, 180-186.

[109] Statistical Analysis Systems Institute Inc. 1990. *SAS Language: Reference, version 6, first edition*, published by SAS Institute Inc., Cary, NC - USA.

[110] Statistical Analysis Systems Institute Inc. 1990. *SAS/STAT - User's Guide, version 6, fourth edition*, published by SAS Institute Inc., Cary, NC - USA.

[111] Sun Microsystems, Inc. 1989. *Sun Pascal Reference Manual.* Sun Microsystems, Inc.

[112] Sun Microsystems, Inc. 1989. *Sun Pascal User's Guide.* Sun Microsystems, Inc.

[113] Sun, J., and K. Tsai. 1989. An Implementation of the Network Simplex Method for Piecewise-Linear Programs. Technical Report 89-07, Department of Industrial Engineering and Management Sciences, Nothwestern University, Evanston, IL, USA.

[114] Sun, J., L. Qi, and K.-H. Tsai. Solving Stochastic Transshipment Problems as Network Piecewise Linear Programs. Technical Report, Department of Industrial Engineering and Management Sciences, Nothwestern University, Evanston, IL, USA.

[115] Tardos, É. 1985. A Strongly Polynomial Minimum Cost Circulation Algorithm. *Combinatorica* 5, No. 3, 247-255.

[116] Taylor, J. 1973. The Analysis of Designed Experiments with Censored Observations. *Biometrics* 29, 35-43.

unesp

[117] Thakur, L. S. 1978. Error Analysis for Convex Separable Programs: the Piecewise-Linear Approximation and the Bounds on the Optimal Objective Value. *SIAM Journal on Applied Mathematics* 34, No. 4, 704-714.

[118] Thakur, L. S. 1986. Successive Approximation in Separable Programming: An Improved Procedure for Convex Separable Programs. *Naval Research Logistics Quarterly* 33, No. 2, 325-358.

[119] Veldhorst, M.. 1990. A Bibliography on Network Flow Problems. *Algorithms Review* 1, No.2, 97- 117.

[120] Wang, J.-S., and S. Melhotra. 1995. Conjugate Gradient Based Implementation of Interior Point Methods for Network Flow Problems. Technical Report 95-70, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL 60208-3119, USA.

[121] Williams, A. C. 1962. A Treatment of Transportation Problems by Decomposition. *SIAM Journal* 10, No. 1, 35-48.

[122] Wolfe, P.. 1965. The Composite Simplex Algorithm. *SIAM Review* 7, 42-54.

APÊNDICE 1. MANUAL DE USUÁRIO DO NplSys

NplSys

NETWORK PIECEWISE-LINEAR TESTING SYSTEM

User's Guide

Edson Luiz França Senne[1]
Fernando Augusto Silva Marins[2]

Brunel University
Department of Mathematics and Statistics
Uxbridge, Middlesex
UB8 3PH, England - UK

Clóvis Perin Filho

Universidade Estadual de Campinas
Instituto de Matemática, Estatística e Ciência da Computação
Caixa Postal 6065
13081-970 Campinas, SP - Brazil
E-mail: clovis@ime.unicamp.br

1 Departamento de Matemática
Universidade Estadual Paulista
Caixa Postal 205
12500-000 Guaratinguetá, SP - Brazil
E-mail: elfsenne@grt000.uesp.ansp.br

2 Departamento de Produção
Universidade Estadual Paulista
Caixa Postal 205
12500-000 Guaratinguetá, SP - Brazil
E-mail: fmarins@grt000.uesp.ansp.br

Abstract

In this paper NplSys - Network Piecewise-Linear Testing System a portable testing system for computational comparisons among specialised algorithms for network piecewise-linear programmes is presented. NplSys is able to generate random samples of network piecewise-linear programmes, to solve them by four implemented algorithms (primal-simplex, dual-simplex, out-of-kilter, and cost-scaling), to collect CPU times spent by each algorithm to solve any problem sample, and to put these solution data in a suitable format in order to perform statistical tests by SAS - Statistical Analysis System procedures.

1. Introduction

An important area of Mathematical Programming is Network Piecewise-Linear Programming (NPLP) which is related to the minimisation of a convex separable piecewise-linear objective function, subject to linear equations expressing the conservation of flow over an incidence matrix of a connected network. In fact important applications of NPLP appear naturally in many situations such as, Power Systems, Stochastic Transportation, Communications Networks, Traffic Equilibrium, Electrical Systems, Pipe Networks, Multireservoir Systems, to name a few.

In order to offer useful information to researchers with interest on NPLP about four available specialised algorithms developed and implemented for this type of programme, and their relative efficiency to solve problems with different characteristics, a statistically designed experiment has been performed (see [3]). All necessary observations, i.e., data points which were analysed by this computational experiment, were created and managed by NplSys - Network Piecewise-Linear Testing System, a program written in Pascal language ([6],[7]).

In this paper we describe NplSys, which was designed to allow random generation of network piecewise-linear samples with different characteristics, and to solve them by the algorithms under consideration: primal-simplex, dual-simplex, out-of-kilter, and cost scaling. In [3] are described two experiments which include the following characteristics: (i) problem class (Transportation, Transshipment, and Circulation), problem size (small, medium, and large), and percentage (100%, 80%, 50%, 20% and 0%) of upper capacitated arcs on the underlying network; (ii) problem class (Transportation, Transshipment, and Circulation), problem size

(small, medium, and large), and number of breakpoints (3, 6 or 9) per arc on the underlying network. The NplSys sets CPU times spent for each algorithm to solve problem samples in a suitable format to allow the use of statistical testing by means of SAS - Statistical Analysis System procedures ([4],[5]).

In the next section we briefly present the network piecewise-linear programme formulation.

2. Problem Statement

Let G = (N,E) be a directed connected network with |N| = n nodes and |E| = m arcs. Assume that the vector b = (b_i) denotes node requirements satisfying $\Sigma_i b_i = 0$ for all $i \in N$. Observe that $b_i < 0$ represents supply in source node, $b_i > 0$ represents demand in sink node, and $b_i = 0$ represents a transshipment node. Moreover, let the associated node-arc incidence matrix be denoted by A = (a_{ij}), and defined by $a_{tj,j} = -1$, $a_{hj,j} = +1$ and $a_{i,j} = 0$ if $i \neq t_j, h_j$, where t_j and h_j are the tail and the head of the arc $j \in E$, respectively.

Now, consider that for each arc $j \in E$ there is a nonnegative variable flow denoted by x_j, and a convex piecewise-linear function denoted by $C_j(x_j)$, associated with a given sequence of increasing breakpoints $0 = d_j^1 < d_j^2 < \ldots < d_j^k < d_j^{k+1} < \ldots < d_j^{mj} < \infty$, and a given corresponding sequence of increasing cost coefficients $c_j^1 < c_j^2 < \ldots < c_j^k < c_j^{k+1} < \ldots < c_j^{mj}$. A typical example of this type of function is presented in Figure 1.

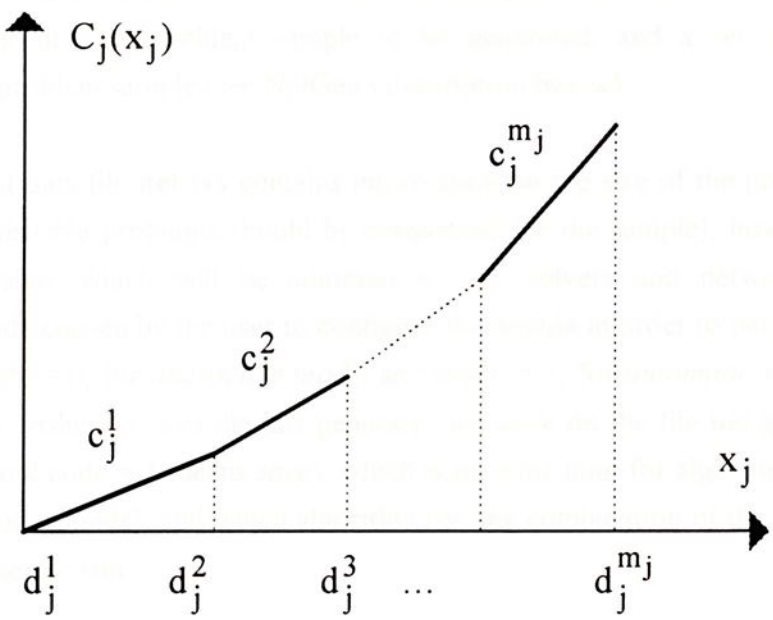


Fig. 1 - A convex piecewise-linear cost function

Observe that if the flow variable $x_j \in [d_j^{fj}, d_j^{fj+1}]$ then it is possible to show that the corresponding value of the function $C_j(x_j)$ can be computed by:

$$C_j(x_j) = \Sigma_k [c_j^k(d_j^{k+1} - d_j^k) : k = 1..f_j - 1] + c_j^{fj}(x_j - d_j^{fj}).$$

Now, define an arc flow vector in G(N,E) by $X = (x_j)$, and a convex separable piecewise-linear function associated with vector X given by $C(X) = \Sigma_j C_j(x_j)$, where for each x_j, and all $j \in E$, $C_j(x_j)$ can be computed as above.

So, a **network piecewise-linear programme** is related to find an optimal solution X^* to Min $\{C(X) : AX = b, X \geq 0\}$. More details can be found in [3].

3. Description of the Network Piecewise-Linear Testing System

The Network Piecewise-Linear Testing System (**NplSys**) is constituted by three main components: a system management programme (**SysMa**), a network piecewise-linear generator programme (**NplGen**), and a set of four solvers programmes (**Solvers**). The system also produces data in a suitable format for a statistical analysis programme (**StaTest**). The system's architecture is presented in the Appendix A.

The first component - **SysMa**, is a module with user-friendly characteristics and its task is basically to manage the computational tests. Initially, SysMa reads input data from three files: net.par, net.sys, and net.see. In the input file **net.par**, the user should specify the identification number of each problem sample to be generated, and a set of parameters associated with this problem sample (see NplGen's description below).

The input data file **net.sys** contains information on the size of the problem sample (that is, how many feasible problems should be considered for the sample), how much is the Big M (*infinity*) value, which will be admitted by the solvers and network generator programmes, the mode chosen by the user to configure the SysMa in order to perform the tests (the options are: mode = 0, for *interactive mode*, and mode = 1, for *automatic mode*), a code indicating if the user wishes to save the last generated network on the file **net.gen** (code = 0 means *do not save*, and code = 1 means *save*), which is the time limit for algorithm's execution (given in hundredth of seconds), and which algorithm (or any combination of the four available algorithms) was chosen to run.

The Appendix B presents an example of network piecewise-linear problem and how this problem is stored in the file net.gen.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

The difference between the two possible operating modes for SysMa is related to the supplying of seeds for network random generation. In the interactive mode, the user should supply specific seeds to each sample. In the automatic model, these seeds will be randomly generated by the system. If the user wishes to run SysMa in the interactive mode, then it is necessary to fill in the third data file, **net.see**, with an appropriated number of seed values (one seed per row) chosen to generate each problem sample.

After read input data, SysMa calls the network generator NplGen to generate the random networks for the problem sample corresponding to the first data parameter set in net.par (using the seeds given in net.see, when applicable), and calls the selected algorithms from the component Solvers to solve it.

For each generated network and algorithm, are computed the following results: CPU time, number of iterations, number of pivotings/phases (see Section 4 for details), the primal and dual objective function values (observe that for problems with optimal solutions these values should be the same), and a code with information about the problem status (code = 1 means *infeasible problem,* code = 2 means *unlimited solution,* code = 3 means *feasible problem*, and code = 4 means *the time limit for algorithm's execution was exceeded*).

Observe that these recorded CPU times (in fact are observed the user times) consider only the execution (optimisation process) times spent by each algorithm, i.e., the time spent to check if the time limit for execution was exceeded as well as the input and output times are not computed.

After each network has been solved by the selected algorithms, SysMa transfers the seed utilised by NplGen to generate that network and the results obtained by each algorithm to the file **net*num*.log**, where ***num*** is the identification number (with up to three digits) of the problem sample (as specified by the user in the file net.par). Then, a new random network is generated and the process continues until all the networks belonging to that sample (according the *sample size* specified in net.sys) have been solved. At the end, the mean values of CPU time, number of iterations and number of pivotings/phases for the selected algorithms are also transferred to the file net*num*.log.

The Appendix C presents an example of NplSys operation with the input files net.sys and net.par and the corresponding output files produced for these data. This appendix also presents a session with the programme **Config** which has been implemented to facilitate the input of data in the files net.sys and net.par. Note that for the experiments reported in [3] the following codes have been used:

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Code	Class	Size	Level of Capacitation	Number of Breakpoints	Method
1	Transportation	Small	100%	3	Primal Simplex
2	Transshipment	Medium	80%	6	Dual Simplex
3	Circulation	Large	50%	9	Out-of-Kilter
4			20%		Cost Scaling
5			0%		

The SysMa's operation procedure is repeated for all the parameter sets (problem samples) contained in the file net.par. In addition, SysMa writes on the file **net.sas** the obtained results in a suitable format to the program StaTest, which performs statistical analysis (see Appendix C).

Other important component of NplSys is the network piecewise-linear generator programme - **NplGen**, which was designed to include the main features of the well-know linear network generator NETGEN (see [2]). With NplGen the user is able to control the structure, problem class, problem size, and arc/node characteristics, associated with each type of problem studied. The input parameters for each type of problem are user-supplied to NlpGen in the file net.par.These input parameters are (see also Tables 1 and 2, in the Appendix D) the following:

(a) **n** = number of nodes.

(b) **m** = number of arcs.

(c) **l** = total number of different generated intervals. See Notes below.

(d) **nfub** = number of finite upper bounds (breakpoints) per arc. See Notes below.

(e) $\mathbf{p_{trans}}$ = probability of transshipment nodes occurrence. This parameter controls the class of problem which will be generated. So, if the user sets $p_{trans} = 0$, will be obtained a *Transportation Problem*. If one sets $0 < p_{trans} < 1$, then a *Transshipment Problem* is obtained. With $p_{trans} = 1$ one obtains a *Circulation Problem*. See Notes below.

(f) $\mathbf{p_{cap}}$ = probability of upper capacitated arc occurrence. If the user sets $p_{cap} = 0$, all generated arcs will be *uncapacitated* (i.e., all arc has the last upper bound as ∞). If one sets $0 < p_{cap} < 1$ then p_{cap} % of the generated arcs will be *upper capacitated* (i.e., p_{cap} % of the total of arcs have a finite last upper bound). With $p_{cap} = 1$ will be generated a *fully upper capacitated* network (i.e., all the arcs have a finite last upper bound).

Moreover, the user can control problem structure by defining on the file net.par: node requirement values range **(b_{MIN}, b_{MAX})**, arc cost values range **(c_{MIN}, c_{MAX})**, and breakpoint values range **(d_{MIN}, d_{MAX})**. For node i, arc j, and interval k, demand b_i, cost c_j^k, and breakpoint (upper bound) d_j^k values are uniformly generated on the respective ranges.

Notes:

(1) Consider a network G(N,E) generated by NplGen. For each arc j belonging to G, with generated sequence of increasing breakpoints $d_j^1 < d_j^2 < ... < d_j^{mj}$ and corresponding generated sequence of increasing cost coefficients $c_j^1 < c_j^2 < ... < c_j^{mj}$, are added two artificial intervals. The artificial intervals are created as being $(-\infty, d_j^1]$ and $[d_j^{mj}, +\infty)$ with associated cost coefficient $-\infty$, and $+\infty$, respectively. See [3] for theoretical details, and the Appendix B for an example.

(2) NplGen randomly creates different intervals for each arc until the l^{th} interval has been generated, where l is the total number of different intervals, as specified by the user. Suppose that during interval generation, this limit of intervals is achieved for the arc j. Then NplGen will allocate to the arc j+1 the same intervals of the arc 1, will allocate to the arc j+2 the intervals allocated to the arc 2, and so on. This feature reproduces practical situations where various arcs have the same flow bounds.

(3) NplGen can be used to generate a **pure linear network problem** as well. There are two possible cases:

if $X \geq 0$, then the user has to give nfub = 0, and
if $0 \leq X \leq U$, then the user has to give nfub = 1.

If the user wishes to generate **network piecewise-linear problems** then it is necessary to give nfub ≥ 1. See Appendix E for examples of input data for typical cases.

(4) Observe that for generating **Circulation Problems** with optimal solution $\mathbf{X^*} \neq \mathbf{0}$ it is necessary to use either $\mathbf{c_{MIN} < 0}$ or $\mathbf{d_{MIN} > 0}$. The first option is not recommended when the generated problem is not a fully upper capacitated network, because NplGen can sometimes generate an unbounded problem.

(5) The parameters *num*, n, m, l, nfub, b_{MIN}, b_{MAX}, c_{MIN}, c_{MAX}, d_{MIN}, and d_{MAX} should be integer numbers, but p_{TRANS}, and p_{CAP} should be real numbers belonging to the interval [0, 1]. The seed values (written on the file net.see) as well as the infinity value (usually ∞ is 99999, or a number of this magnitude, for large networks) given in net.sys should be integer numbers.

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

An important feature of NplGen is the use of a random number generator procedure. This procedure starts either with an user-supplied seed (in the SysMa's interactive mode), or with a random seed generated by the system itself (in the SysMa's automatic mode). In this way the user is always able to generate several different random network piecewise-linear with the same structure. Additionally, observe that as the seeds for network generating are saved in the file net.log, it is possible, at any time, to regenerate previously generated networks, since each input parameter is repeated.

The third component of NplSys - **Solvers**, includes the implementation of four specialised algorithms to solve network piecewise-linear problems which will have their performance tested. The implemented algorithms are: a (strongly feasible) primal-simplex, a dual-simplex, an out-of-kilter, and a (strongly polynomial) cost-scaling. These algorithms are called in NplSys as *primal, dual, kilter,* and *scale*, respectively. For details on the algorithms, see [3].

StaTest is a module with SAS programmes including statistical methods as *Variance Analysis* and *Multiple Mean Comparisons* (see [5]). The inputs for this programme are: (i) the data identification, which includes the problem's identification (class, size and percentage of upper capacitated arcs or number of breakpoints per arc) and the algorithm used to solve it, and (ii) the CPU time spent by the corresponding algorithm. These data are stored on the file net.sas (see Appendix C).

4. Comments

The reference [3] describes statistically designed experiments for a comparative study with the four algorithms referred above, which has allowed to make statistical-based inferences on their relative efficiencies to solve network piecewise-linear programmes. This study has required the implementation of NplSys to provide necessary control on generation of problem samples, algorithm selection, data collection, and statistical tests execution.

The NplSys has been implemented in two versions. The first one, **NplSys/DOS**, is suitable to run on an IBM-PC compatible microcomputer, and it is recommended for small size problems ($n \leq 300$, $m \leq 2000$, $l \leq 10000$ and any nfub). It was used the Borland Pascal version 7.0 for compilation ([1]).

The another version, **NplSys/UNIX** (adopted in [3]), is suitable to run on SUN SPARCstations and support large network piecewise-linear problems (in [3], programmes with up to $n = 2500$, $m = 25000$, $l = 50000$ and $nfub = 3$, have been studied). It was used Sun

Pascal Release 2.1/SC 1.0 with the option -O2 for compilation (see [7]). The tests were performed using a SUN SPARCstation 10 machine, in a multiprogramming environment. The utilised input data and the obtained results (for the first experiment) are shown in Appendix F.

Another characteristic of NplSys is that its execution is stopped when it can not get the number of required feasible problems (sample size) with up to *3*sample size* different seeds. In this case, the utilised seeds (besides the CPU times spent by the algorithms, the number of iterations, the number of pivotings/phases, and the associated solution code for each solved problem), as well as appropriated messages are saved on the file net*num*.log.

Due to the characteristics of the specialised algorithms for solving NPLP problems, it is necessary to observe the difference between *number of iterations* and *number of phases/pivotings* (see [3] for theoretical details) as shows the following table:

Method	Interpretation	
	Number of iterations	Number of phases/pivotings
Primal and Dual	Simplex iterations	Basis changings
Out-of-Kilter	Iterations	Primal and Dual phases
Cost Scaling	Iterations	Scales

The Appendix G presents the listing of SysMa and NplGen components of NplSys. *For free copies of NplSys programme, please contact the authors.*

5. References

[1] Borland International, Inc. 1992. *Borland Pascal with Objects: User's Guide*, version 7.0, published by Borland International, Inc., Scotts Valley, CA, USA.

[2] Klingman, D., A. Napier, and J. Stutz. 1974. NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. *Management Science* 20, No. 5, 814-821.

[3] Marins, F.A.S., K. Darby-Dowman, E.L.F. Senne, A. Machado, and C. Perin. 1994. Algorithms for Network Piecewise-Linear Programmes: A Statistically Designed Study. Technical Report, Department of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, England, UK.

[4] SAS - Statistical Analysis System Institute Inc. 1990. *SAS/STAT User's Guide*, version 6, fourth edition, published by SAS Institute Inc., Cary, NC, USA.

unesp

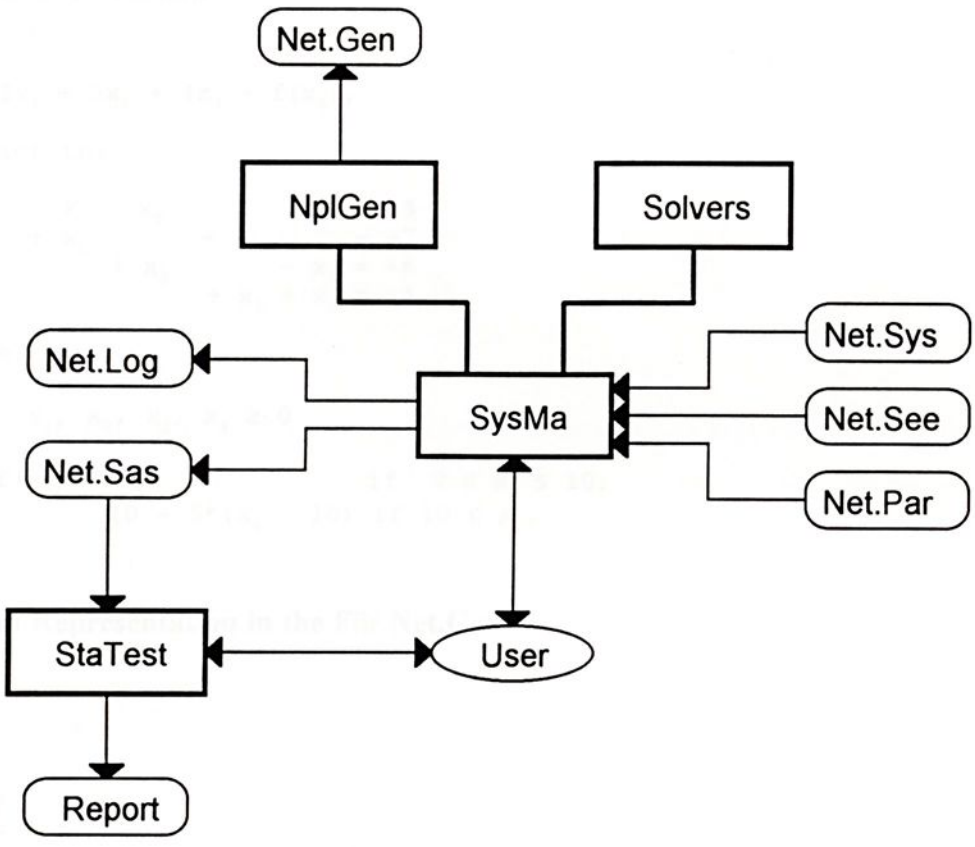
[5] SAS - Statistical Analysis System Institute Inc. 1990. *SAS* Language: Reference, version 6, first edition, published by SAS Institute, Inc., Cary, NC, USA.

[6] Sun Microsystem, Inc. 1989. *Sun Pascal User's Guide*, published by Sun Microsystem, Inc.

[7] Sun Microsystem, Inc. 1989. *Sun Pascal Reference Manual*, published by Sun Microsystem, Inc.

Appendix A

NplSys Architecture



In this figure:

⟶	denotes the information flow direction.
———	denotes the control flow direction.
Net.Par	is an input file with the values for the parameters: *num*, n, m, l, nfub, b_{MIN}, b_{MAX}, p_{TRANS}, c_{MIN}, c_{MAX}, d_{MIN}, d_{MAX}, and p_{CAP}, related to each problem sample.
Net.Sys	is an input file with the following parameters: sample size, Big M value, mode for SysMa operating, a code indicating if the last generated network should be saved or not, time limit for algorithm's execution, and which algorithms were chosen to run.
Net.See	is an input file with the seed values for the interactive mode of SysMa operating.
Net.Gen	is an output file with the last generated network.
Net*num*.Log	is an output file which contains the performance results of the algorithms used to solve each problem sample.
Net.Sas	is an output file which contains the performance results in a suitable format to SAS procedures.

Appendix B

An Example of Network Piecewise-Linear Problem

B.1 - Problem Formulation

$$\text{Min } 2x_1 + 5x_2 + 4x_3 + f(x_4),$$

subject to:

$$\begin{aligned} -x_1 - x_2 &= -3 \\ +x_1 - x_3 &= -7 \\ +x_2 - x_4 &= +6 \\ +x_3 + x_4 &= +4 \end{aligned}$$

where:

$$x_1, x_2, x_3, x_4 \geq 0$$

$$f(x_4) = \begin{cases} x_4 & \text{if } 0 \leq x_4 \leq 10, \\ 10 + 5*(x_4 - 10) & \text{if } 10 \leq x_4. \end{cases}$$

B.2 - Problem Representation in the File Net.Gen

```
-3
-7
+6
+4
1    2    2
1    3    5
2    4    8
3    4   11
-99  -99
  0    2
 99   99
-99  -99
  0    5
 99   99
-99  -99
  0    4
 99   99
-99  -99
  0    1
 10    5
 99   99
```

In the file Net.Gen, first of all are specified the demand values b_i for each node i, that is:

```
-3
-7
+6
+4
```

Note that, the first two nodes are source nodes (since $b_i < 0$) and the last two nodes are sink nodes (since $b_i > 0$). In this case, there is no transshipment node ($b_i = 0$).

Next, the head, the tail and the number of the first non-artificial interval for each arc are specified. That is:

```
1   2   2
1   3   5
2   4   8
3   4  11
```

Note, in this case, there is an arc from node 1 to node 2 for which the first non-artificial interval is the interval number 2; there is an arc from node 1 to node 3 for which the first non-artificial interval is the interval number 5; and so on.

Finally, the (artificial and non-artificial) intervals as well as the associated costs are specified. In the case above, there are 13 intervals (note that, the value 99 is used as the infinity representation), that is:

```
 1)  -99  -99
 2)    0    2
 3)   99   99
 4)  -99  -99
 5)    0    5
 6)   99   99
 7)  -99  -99
 8)    0    4
 9)   99   99
10)  -99  -99
11)    0    1
12)   10    5
13)   99   99
```

The first interval is $(-\infty, 0]$ with cost = $-\infty$. This is an artificial interval. The interval number 2 (a non-artificial interval) corresponds to $[0, \infty)$ with cost = 2. So, the intervals 1, 3, 4, 6, 7, 9, 10 and 13 are artificial intervals and the intervals 2, 5, 8, 11 and 12 are non-artificial intervals. For arc 4, for example, the first non-artificial interval is the interval number 11 (which corresponds the interval [0, 10], with cost = 1), and the second non-artificial interval is the interval number 12 (which corresponds the interval $[10, \infty)$ with cost = 5).

unesp

Appendix C

An Example of NplSys Operation

C.1 - Input File Net.Sys

```
10    99999    1    0    360000    PDKS
```

That is:

- sample size = 10
- Big M value = 99999
- operating mode = 1 (automatic mode)
- saving code = 0 (the last generated network will not be saved)
- time limit for algorithm's execution = 360000 (1 hour = 360000 hundredth of seconds)
- selected algorithms = PDKS (P for Primal, D for Dual, K for Kilter, S for Cost Scaling)

C.2 - Input File Net.Par

```
213    50    500    500  3  -100  100  0.5  0  250  -5  20  0.5
```

That is:

- Problem Sample Identification = 213
- Number of Nodes = 50
- Number of Arcs = 500
- Total Number of Different Intervals = 500
- Number of Finite Upper Bounds (Breakpoints) per Arc = 3
- Demand Lower Bound = -100
- Demand Upper Bound = 100
- Probability of Transshipment Node Occurrence = 0.5
- Interval Lower Bound = 0
- Interval Upper Bound = 250
- Interval Cost Lower Bound = -5
- Interval Cost Upper Bound = 20
- Probability of Upper Capacitated Arc Occurrence = 0.5

unesp

C.3 - Input of Data by Config Programme

```
                    Computational Experiments with
                  Network Piecewise Linear Programmes
                  ===================================
                         *** CONFIGURATION ***

    What Do You Wish?

    1. Update System Configuration File (NET.SYS)
    2. Update Problem Parameters File (NET.PAR)
    3. Exit

 -> 1
>>> Updating NET.SYS...

    Sample Size (Number of Feasible Problems) ........ 10
    Representation for Infinity (Big M) .............. 99999
    Random Seeds for Network Generation? (Y or N) .... Y
    Save the Last Network Generated? (Y or N) ........ N
    Time Limit for Algorithm's Execution ............. 360000
    Selected Algorithms (P/D/K/S) .................... PDKS

>>> NET.SYS Updated!
Press ENTER to continue...

                    Computational Experiments with
                  Network Piecewise Linear Programmes
                  ===================================
                         *** CONFIGURATION ***

    What Do You Wish?

    1. Update System Configuration File (NET.SYS)
    2. Update Problem Parameters File (NET.PAR)
    3. Exit

 -> 2
>>> Updating NET.PAR...

    Parameter Set No. ................................ 213
    Number of Nodes .................................. 50
    Number of Arcs ................................... 500
    Total Number of Different Intervals .............. 500
    Number of Breakpoints per Arc .................... 3
    Demand Lower Bound ............................... -100
    Demand Upper Bound ............................... 100
    Probability of Transshipment Node Occurrence ..... 0.5
    Interval Lower Bound ............................. 0
    Interval Upper Bound ............................. 250
    Interval Cost Lower Bound ........................ -5
    Interval Cost Upper Bound ........................ 20
    Probability of Upper Capacitated Arc Occurrence .... 0.5

>>> More? (Y or N): N
>>> NET.PAR Updated!
```


cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

C.4 - Output File Net213.Log

```
 SYSMA/UNIX [Version 1994] Report       28 Oct 94    14:55:53       [net213.log]
-------------------------------------------------------------------------------
   N       M       L  NBRK   BMIN   BMAX  PTRANS   DMIN   DMAX   CMIN   CMAX  PCAP
  50     500     500     6   -100    100    0.50      0    250     -5     20  0.50

             CPU   NITER   NPHAS        CX        YB      CODG
NplGen ------------------------------------------------------- [seed =  34819]
Primal        15     921     932    -20844    -20844         3
  Dual       106    1272    2070    -20844    -20844         3
Kilter        78     264     278    -20844    -20844         3
 Scale       186     818     809    -20844    -20844         3
NplGen ------------------------------------------------------- [seed =  10481]
Primal        20    1090    1101    -25366    -25366         3
  Dual        38     521     875    -25366    -25366         3
Kilter        41     224     239    -25366    -25366         3
 Scale       169     799     790    -25366    -25366         3
NplGen ------------------------------------------------------- [seed =  98681]
Primal        11     929     952    -22844    -22844         3
  Dual        58     642    1232    -22844    -22844         3
Kilter        60     254     270    -22844    -22844         3
 Scale       191     841     832    -22844    -22844         3
NplGen ------------------------------------------------------- [seed =  42758]
Primal        18     961     975    -22074    -22074         3
  Dual        88    1009    1734    -22074    -22074         3
Kilter        73     271     286    -22074    -22074         3
 Scale       205     860     851    -22074    -22074         3
NplGen ------------------------------------------------------- [seed =  28047]
Primal         9    1141    1162    -20862    -20862         3
  Dual        70     861    1367    -20862    -20862         3
Kilter        70     236     249    -20862    -20862         3
 Scale       181     836     827    -20862    -20862         3
NplGen ------------------------------------------------------- [seed =  36216]
Primal        21     943     953    -20562    -20562         3
  Dual       121    1398    2301    -20562    -20562         3
Kilter        70     276     290    -20562    -20562         3
 Scale       211     861     852    -20562    -20562         3
NplGen ------------------------------------------------------- [seed =  97826]
Primal        16     992    1016    -26031    -26031         3
  Dual       100    1170    2065    -26031    -26031         3
Kilter        60     272     285    -26031    -26031         3
 Scale       230     926     916    -26031    -26031         3
NplGen ------------------------------------------------------- [seed =  82446]
Primal        16     959     973    -22322    -22322         3
  Dual        83    1010    1648    -22322    -22322         3
Kilter        56     251     263    -22322    -22322         3
 Scale       170     810     800    -22322    -22322         3
NplGen ------------------------------------------------------- [seed =  37306]
Primal        11     843     858    -18465    -18465         3
  Dual        54     670    1176    -18465    -18465         3
Kilter        51     231     245    -18465    -18465         3
 Scale       209     883     871    -18465    -18465         3
NplGen ------------------------------------------------------- [seed =  21257]
Primal        11    1008    1021    -22614    -22614         3
  Dual       103    1044    1676    -22614    -22614         3
Kilter        73     241     254    -22614    -22614         3
 Scale       183     818     808    -22614    -22614         3

   ------------------------ Mean Values ------------------------
                      CPU             NITER             NPHAS
   -------------------------------------------------------------
    Primal          14.80            978.70            994.30
      Dual          82.10            959.70           1614.40
    Kilter          63.20            252.00            265.90
     Scale         193.50            845.20            835.60
   -------------------------------------------------------------
```

C.5 - SAS Programme

```
/* -------------------Full Factorial Experiment--------------------- */
/*  Dependent Variable:                                              */
/*    CPU times spent by algorithms Primal, Dual, Out-of-Kilter,     */
/*    and Cost Scaling to solve Network Piecewise-Linear Programmes. */
/*  Independent Variables:                                           */
/*    Class of Problems (ki, i = 1, 2, 3, respectively to            */
/*      Transportation, Transshipment, and Circulation);             */
/*    Size of Problems (sj, j = 1, 2, 3, respectively to             */
/*      Small, Medium, and Large);                                   */
/*    % Upper Capacitated Arcs (ck, k = 1,..., 5, respectively to    */
/*      100%, 80%, 50%, 20%, 0%);                                    */
/*    Method Utilized (mh, h = 1,..., 4, respectively to             */
/*      Primal, Dual, Out-of-Kilter, and Cost-Scale).                */
/* ------------------------------------------------------------------ */

title 'Comparative Testing with Algorithms for NPLP using a Full Factorial
Experiment';
   data nplsys;
   infile 'net.sas';
   input k 1 s 2 c 3 m 4 cpu;
proc anova;
   class k s c m;
   model cpu=  k s c m k*s  k*c s*c k*s*c  k*m s*m k*s*m c*m k*c*m s*c*m;
   means k s c m k*s  k*c s*c k*s*c  k*m s*m k*s*m c*m k*c*m s*c*m /
   tukey cldiff lines;
run;
```

C.6 - Input File (net.sas) for SAS Programme

```
1111        231
1112       3576
1113       1381
1114       2555
1121        230
1122       4560
1123       1365
1124       2875
1131        248
1132       5293
1133       1330
1134       2876
1141        248
1142       3425
1143       1451
1144       2985
1151        240
1152       4315
1153       1323
1154       2805
1211       9923
1212     720002
1213      51700
1214      61848
1221      10333
1222     720004
1223      49364
1224      79794
1231       9861
1232     720001
```

```
1233       48713
1234       79498
1241        9466
1242      720002
1243       48423
1244       79405
1251        9875
1252      720002
1253       51788
1254       79829
1311       84100
1312      720001
1313      270711
1314      335658
1321       83444
1322      720001
1323      261328
1324      383238
1331       82535
1332      720002
1333      261671
1334      396252
1341       85293
1342      720004
1343      264268
1344      435192
1351       84961
1352      720003
1353      264313
1354      382521
2111          305
2112         6485
2113         1363
2114         3038
2121          321
2122         5591
2123         1330
2124         3193
2131          320
2132         5035
2133         1273
2134         3261
2141          328
2142         6160
2143         1295
2144         3210
2151          318
2152         4043
2153         1303
2154         3083
2211        13310
2212      720003
2213        50265
2214      100900
2221        13655
2222      720001
2223        50265
2224      111544
2231        13876
2232      720002
2233        48924
2234      111988
2241        13176
2242      720002
```




2243	51008
2244	113219
2251	12654
2252	720001
2253	47821
2254	107235
2311	111525
2312	720001
2313	274129
2314	577810
2321	110186
2322	720002
2323	271984
2324	652237
2331	112342
2332	720004
2333	270209
2334	644612
2341	112942
2342	720001
2343	266800
2344	650273
2351	113281
2352	720001
2353	271750
2354	634027
3111	728
3112	14238
3113	816
3114	1275
3121	791
3122	50750
3123	921
3124	1606
3131	765
3132	47030
3133	815
3134	1435
3141	776
3142	14940
3143	800
3144	1465
3151	728
3152	198076
3153	713
3154	1266
3211	35818
3212	720002
3213	29479
3214	37843
3221	35715
3222	720001
3223	30885
3224	49180
3231	35568
3232	720002
3233	27869
3234	51011
3241	36240
3242	720001
3243	29881
3244	48179
3251	35631
3252	720002



```
3253      29375
3254      47529
3311     272657
3312     720002
3313     184516
3314     259166
3321     269278
3322     720001
3323     177668
3324     287676
3331     273143
3332     720002
3333     176986
3334     296540
3341     274078
3342     720002
3343     176815
3344     286096
3351     274823
3352     720002
3353     173145
3354     295329
```

C.7 - Some Outputs Generated by SAS Programme

```
1                                                    The SAS System
16:43 Wednesday, November 30, 1994

NOTE: Copyright(c) 1989 by SAS Institute Inc., Cary, NC USA.
NOTE: SAS (r) Proprietary Software Release 6.07.02
      Licensed to BRUNEL - THE UNIVERSITY OF WEST LONDON, Site 0020900129.

NOTE: SAS initialization used:
      real time           8.019 seconds
      cpu time            0.700 seconds

NOTE: The infile 'net.sas' is:
      FileName=/a/tycho/vol/fileservers/tycho/tycho1/masr/masrees/pascaln/net.sas,
      Owner Name=masrees,Group Name=masf,
      Access Permission=rw-r-----,
      File Size (bytes)=1978

NOTE: 180 records were read from the infile 'net.sas'.
      The minimum record length was 8.
      The maximum record length was 11.
NOTE: The data set WORK.NPLSYS has 180 observations and 5 variables.
NOTE: DATA statement used:
      real time           1.738 seconds
      cpu time            0.140 seconds

NOTE: The PROCEDURE ANOVA printed pages 1-16.
NOTE: PROCEDURE ANOVA used:
      real time           2.732 seconds
      cpu time            0.790 seconds

NOTE: The SAS System used:
      real time           13.791 seconds
      cpu time            1.800 seconds

NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC 27512-8000


Comparative Testing with Algorithms for NPLP using a Full Factorial Experiment   1
                                                    16:43 Wednesday, November 30, 1994
```

Analysis of Variance Procedure
Class Level Information

Class	Levels	Values
K	3	1 2 3
S	3	1 2 3
C	5	1 2 3 4 5
M	4	1 2 3 4

Number of observations in data set = 180

Comparative Testing with Algorithms for NPLP using a Full Factorial Experiment 2
16:43 Wednesday, November 30, 1994

Analysis of Variance Procedure

Dependent Variable: CPU

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	131	12802260774443.9000000	97727181484.3050000	522.61	0.0001
Error	48	8975936693.7060500	186998681.1188760		
Corrected Total	179	12811236711137.6000000			

R-Square	C.V.	Root MSE	CPU Mean
0.999299	6.732288	13674.74610802	203121.82777778

Source	DF	Anova SS	Mean Square	F Value	Pr > F
K	2	25163921532.34470000	12581960766.17230000	67.28	0.0001
S	2	4342048388059.51000000	2171024194029.75000000	11609.84	0.0001
C	4	1644630007.41113000	411157501.85278300	2.20	0.0832
M	3	5178210920960.99000000	1726070306986.99000000	9230.39	0.0001
K*S	4	45058990732.05560000	11264747683.01390000	60.24	0.0001
K*C	8	1605394784.65332000	200674348.08166500	1.07	0.3975
S*C	8	1904140481.49121000	238017560.18640100	1.27	0.2799
K*S*C	16	2898437387.44433000	181152336.71527100	0.97	0.5034
K*M	6	184086388960.45600000	30681064826.74260000	164.07	0.0001
S*M	6	2783334906951.15000000	463889151158.52600000	2480.71	0.0001
K*S*M	12	219504272212.34300000	18292022684.36190000	97.82	0.0001
C*M	12	4885381113.47949000	407115092.78995700	2.18	0.0285
K*C*M	24	4370065128.32714000	182086047.01363100	0.97	0.5140
S*C*M	24	7544936132.29004000	314372338.84541800	1.68	0.0626

Comparative Testing with Algorithms for NPLP using a Full Factorial Experiment 3
16:43 Wednesday, November 30, 1994

Analysis of Variance Procedure

Tukey's Studentized Range (HSD) Test for variable: CPU

NOTE: This test controls the type I experimentwise error rate.

Alpha= 0.05 Confidence= 0.95 df= 48 MSE= 1.87E8
Critical Value of Studentized Range= 3.420
Minimum Significant Difference= 6038.1

Comparisons significant at the 0.05 level are indicated by '***'.

K Comparison	Simultaneous Lower Confidence Limit	Difference Between Means	Simultaneous Upper Confidence Limit	
2 - 3	17414	23453	29491	***
2 - 1	20405	26443	32481	***
3 - 2	-29491	-23453	-17414	***
3 - 1	-3048	2991	9029	
1 - 2	-32481	-26443	-20405	***
1 - 3	-9029	-2991	3048	

Comparative Testing with Algorithms for NPLP using a Full Factorial Experiment 10
16:43 Wednesday, November 30, 1994

cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

Analysis of Variance Procedure

Tukey's Studentized Range (HSD) Test for variable: CPU

NOTE: This test controls the type I experimentwise error rate, but generally has a higher type II error rate than REGWQ.

Alpha= 0.05 df= 48 MSE= 1.87E8
Critical Value of Studentized Range= 3.764
Minimum Significant Difference= 7672.5

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	M
A	488302	45	2
B	171390	45	4
C	93990	45	3
D	58806	45	1

Level of K	Level of S	N	CPU Mean	CPU SD
1	1	20	2165.600	1584.062
1	2	20	213991.550	300732.196
1	3	20	363774.800	238716.950
2	1	20	2562.750	2059.643
2	2	20	222992.450	296488.895
2	3	20	433705.800	257477.549
3	1	20	16996.700	45123.145
3	2	20	208010.550	303353.075
3	3	20	363896.250	215311.135

Level of K	Level of C	N	CPU Mean	CPU SD
1	1	12	188473.833	271389.865
1	2	12	193044.667	273143.403
1	3	12	194023.333	274058.936
1	4	12	197513.500	276986.470
1	5	12	193497.917	273045.517
2	1	12	214927.833	288706.328
2	2	12	221692.417	297529.489
2	3	12	220987.167	296511.243
2	4	12	221534.500	297001.839
2	5	12	219626.417	295514.740
3	1	12	189711.667	267014.495
3	2	12	195372.667	265058.983
3	3	12	195930.500	265752.236
3	4	12	192439.417	267207.173
3	5	12	208051.583	261765.774

Appendix D

Classification of the Input Parameters

Table 1 - Input parameters associated with network class, network size, and arc/node characteristics.

PARAMETER	NOTATION
Number of Nodes	**n**
Number of Arcs	**m**
Number of Finite Upper Bounds per Arc	**nfub**
Maximum Number of Different Intervals	**l**
Probability of Transshipment Nodes Occurrence	$\mathbf{p_{TRANS}}$
Probability of Upper Capacitated Arcs Occurrence	$\mathbf{p_{CAP}}$

Table 2 - Input parameters associated with the network structure.

PARAMETER	NOTATION
Node Requirement Range	$\mathbf{[b_{MIN}, b_{MAX}]}$
Arc Cost Range	$\mathbf{[c_{MIN}, c_{MAX}]}$
Breakpoint Range	$\mathbf{[d_{MIN}, d_{MAX}]}$

It is important to observe that, for each node i, arc j, and interval k, the demands b_i are uniformly generated on $[b_{MIN}, b_{MAX}]$, the costs c_j^k are uniformly generated on $[c_{MIN}, c_{MAX}]$ and the breakpoints d_j^k are uniformly generated on $[d_{MIN}, d_{MAX}]$.

Appendix E

Input Parameters for Typical Network Piecewise-Linear Problems

PROBLEM	INPUT PARAMETER	
	nfub	**p_{CAP}**
Linear Uncapacitated Network: $X \geq 0$ and $c \in [c_{MIN}, c_{MAX}]$	0	-
Linear Capacitated Network: $0 \leq X \leq U$ and $c \in [c_{MIN}, c_{MAX}]$	1	1.0
Linear Partial Capacitated Network: p % of the arcs with $0 \leq X \leq U$ and $c \in [c_{MIN}, c_{MAX}]$. Other arcs with $X \geq 0$ and $c \in [c_{MIN}, c_{MAX}]$	1	p/100
Piecewise-Linear Uncapacitated Network with One Finite Upper Bound: $0 \leq X \leq U_1$ with $c_1 \in [c_{MIN}, c_{MAX}]$, and $X \geq U_1$ with $c_2 \in [c_{MIN}, c_{MAX}]$	1	0.0
Piecewise-Linear Capacitated Network with n Finite Upper Bounds: $0 \leq X \leq U_1$ and $c_1 \in [c_{MIN}, c_{MAX}]$, $U_1 \leq X \leq U_2$ and $c_2 \in [c_{MIN}, c_{MAX}]$, ..., $U_{n-1} \leq X \leq U_n$ and $c_n \in [c_{MIN}, c_{MAX}]$	n	1.0
Piecewise-Linear Partially Capacitated Network with n Finite Upper Bounds: p % of the arcs with $0 \leq X \leq U_1$ and $c_1 \in [c_{MIN}, c_{MAX}]$, $U_1 \leq X \leq U_2$ and $c_2 \in [c_{MIN}, c_{MAX}]$, ..., $U_{n-1} \leq X \leq U_n$ and $c_n \in [c_{MIN}, c_{MAX}]$. Other arcs with the last interval given by $X \geq U_n$ and $c_n \in [c_{MIN}, c_{MAX}]$	n	p/100
Piecewise-Linear Uncapacitated Network with n Finite Upper Bounds: $0 \leq X \leq U_1$ with $c_1 \in [c_{MIN}, c_{MAX}]$, $U_1 \leq X \leq U_2$ with $c_2 \in [c_{MIN}, c_{MAX}]$, ..., and $U_n \leq X$ with $c_n \in [c_{MIN}, c_{MAX}]$	n	0.0

Appendix F

Some Results

F.1 - Size of Problems

SIZE	# NODES	# ARCS	# TOTAL INTERVALS	# FINITE UPPER BOUNDS
Small	250	2400	10000	3
Medium	1000	12000	30000	3
Large	2500	25000	50000	3

F.2 - Input Parameters Associated with the Network Structure

b_{MIN}	b_{MAX}	c_{MIN}	c_{MAX}	d_{MIN}	d_{MAX}
- 100	**+ 100**	**- 5**	**20**	**0**	**250**

F.3 - Input Parameters for Transportation Piecewise-Linear Problems

$p_{TRANS} = 0.0$			
# SAMPLE	**SIZE**	**p_{CAP}**	**SEED**
111	Small	1.0	74321
112		0.8	
113		0.5	
114		0.2	
115		0.0	
121	Medium	1.0	24689
122		0.8	
123		0.5	
124		0.2	
125		0.0	
131	Large	1.0	39763
132		0.8	
133		0.5	
134		0.2	
135		0.0	

F.4 - Input Parameters for Transshipment Piecewise-Linear Problems

p_{TRANS} = 0.2			
# SAMPLE	**SIZE**	**p_{CAP}**	**SEED**
211	Small	1.0	13459
212		0.8	
213		0.5	
214		0.2	
215		0.0	
221	Medium	1.0	51137
222		0.8	
223		0.5	
224		0.2	
225		0.0	
231	Large	1.0	44331
232		0.8	
233		0.5	
234		0.2	
235		0.0	

F.5 - Input Parameters for Circulation Piecewise-Linear Problems

p_{TRANS} = 1.0			
# SAMPLE	**SIZE**	**p_{CAP}**	**SEED**
311	Small	1.0	92567
312		0.8	
313		0.5	
314		0.2	
315		0.0	
321	Medium	1.0	64743
322		0.8	
323		0.5	
324		0.2	
325		0.0	
331	Large	1.0	70848
332		0.8	
333		0.5	
334		0.2	
335		0.0	

F.6 - Mean CPU Times for Transportation Piecewise-Linear Sample Problems

CLASS: TRANSPORTATION PIECEWISE-LINEAR PROBLEMS					
		MEAN CPU TIMES (hundredth of seconds)			
SIZE	P_{CAP}	PRIMAL	DUAL	KILTER	SCALE
Small	100	231	3576	1381	2555
	80	230	4560	1365	2875
	50	248	5293	1330	2876
	20	248	3425	1451	2985
	0	240	4315	1323	2805
Medium	100	9923	⊗	51700	61848
	80	10333	⊗	49364	79794
	50	9861	⊗	48713	79498
	20	9466	⊗	48423	79405
	0	9875	⊗	51788	79829
Large	100	84100	⊗	270711	335658
	80	83444	⊗	261328	383238
	50	82535	⊗	261671	396252
	20	85293	⊗	264268	435192
	0	84961	⊗	264313	382521

Observation: The symbol ⊗ indicates that the algorithm was unable to solve the problem within of two hours.

F.7 - Mean CPU Times for Transshipment Piecewise-Linear Sample Problems

CLASS: TRANSSHIPMENT PIECEWISE-LINEAR PROBLEMS					
		MEAN CPU TIMES (hundredth of seconds)			
SIZE	P_{CAP}	PRIMAL	DUAL	KILTER	SCALE
Small	100	305	6485	1363	3038
	80	321	5591	1330	3193
	50	320	5035	1273	3261
	20	328	6160	1295	3210
	0	318	4043	1303	3083
Medium	100	13310	⊗	50265	100900
	80	13655	⊗	50265	111544
	50	13876	⊗	48924	111988
	20	13176	⊗	51008	113219
	0	12654	⊗	47821	107235
Large	100	111525	⊗	274129	577810
	80	110186	⊗	271984	652237
	50	112342	⊗	270209	644612
	20	112942	⊗	266800	650273
	0	113281	⊗	271750	634027

Observation: The symbol ⊗ indicates that the algorithm was unable to solve the problem within of two hours.



F.8 - Mean CPU Times for Circulation Piecewise-Linear Sample Problems

CLASS: CIRCULATION PIECEWISE-LINEAR PROBLEMS					
		MEAN CPU TIMES (hundredth of seconds)			
SIZE	P_{CAP}	PRIMAL	DUAL	KILTER	SCALE
Small	100	728	14238	816	1275
	80	791	50750	921	1606
	50	765	47030	815	1435
	20	776	14940	800	1465
	0	728	198076	713	1266
Medium	100	35818	⊗	29479	37843
	80	35715	⊗	30885	49180
	50	35568	⊗	27869	51011
	20	36240	⊗	29881	48179
	0	35631	⊗	29375	47529
Large	100	272657	⊗	184516	259166
	80	269278	⊗	177668	287676
	50	273143	⊗	176986	296540
	20	274078	⊗	176815	286096
	0	274823	⊗	173145	295329

Observation: The symbol ⊗ indicates that the algorithm was unable to solve the problem within of two hours.

Appendix G

Listing of Some NplSys/UNIX Components

G.1 - The System Management Programme

```
program SysMa;
{ ---
              Network Piecewise Linear Programming (NPLP)
              ===========================================

     This program controls NPLP experiments with the following methods:

       1. Strongly Feasible Primal Simplex Method
       2. Dual Simplex Method
       3. Out-of-Kilter Method
       4. Iteractive Maximum Flow Method

     Initially, the program reads the following configuration parameters
     from the file NET.SYS:

           G_PROG - Number of feasible problems to be considered in each
                    experiment (sample size).
             INFT - The representation for infinity.
             GENE - The value of this parameter should be 1 or 0. The value
                    1 indicates the seed for network generation will be
                    generated at random. The value 0 indicates this seed
                    will be supplied by the user.
           N_SAVE - The value of this parameter should be 1 or 0, as well.
                    The value 1 indicates the generated network should be
                    saved on the file NET.LOG. If the value 0 is specified
                    then the generated network will not be saved.
           TLIMIT - Time limit for algorithm's execution.
              ALG - Selected algorithms

     For the experiments, the program reads the following parameters
     from the file NET.PAR:

             NSET - Problem sample identification
                N - Number of nodes
                M - Number of arcs
                L - Total number of breakpoints
             NFUB - Number of finite upper bounds per arc
        BMIN,BMAX - Demand bounds
           PTRANS - Probability of transshipment node occurrence
        DMIN,DMAX - Interval bounds
        CMIN,CMAX - Interval costs bounds
             PCAP - Probability of upper capacitated arc occurrence

     If necessary (see parameter GENE, above), the program reads the
     seeds for network generation from the file NET.SEE

     The program produces the following files:

          NET.LOG - The execution log file.
          NET.GEN - The last generated network (if N_SAVE = 1).
          NET.SAS - CPU times for SAS statistical analysis.

  ------------------------------------------------------------------- }
const
  n_nos  = 6000;          { maximum number of nodes }
  m_arc  = 200000;        { maximum number of arcs }
  l_int  = 200000;        { maximum number of intervals }
  infs   = 1;             { solution code: infeasible problem }
  nltd   = 2;             { solution code: unlimited solution }
  otma   = 3;             { solution code: optimal solution }
  tlim   = 4;             { solution code: time limit exceeded }
```


unesp

```
type
  Vet_n = array [0..n_nos] of integer;
  Vet_m = array [0..m_arc] of integer;
  Vet_l = array [1..l_int] of integer;
  Real4 = array [1..4] of real;
  Str4  = array [1..4] of string;
  Bool4 = array [1..4] of boolean;
  Str10 = varying[10] of char;
var
  i,nm      : integer;  { auxiliary variables }
  rstart,               { sets the random number generation seed }
  s_seed,               { save the network generator seed }
  n_seed,               { network generator seed }
  nprob     : integer;  { number of the problem }
  nfnet     : integer;  { number of the non-feasible generated network }
  method    : Str4;     { name of solution methods }
  exmeth    : Bool4;    { indicates if the method was selected to run }

  nset      : integer;  { number of the problem sample }
  n         : integer;  { number of nodes }
  m         : integer;  { number of arcs }
  l         : integer;  { total number of breakpoints }
  nfub      : integer;  { number of finite upper bounds per arc }
  nbrk      : integer;  { number of breakpoints per arc = nfub + 3 }
  bmin,bmax : integer;  { demand bounds }
  dmin,dmax : integer;  { interval bounds }
  cmin,cmax : integer;  { interval costs bounds }
  ptrans,               { probability of transshipment node occurrence }
  pcap      : real;     { probability of upper capacitated arc occurrence }

  b         : Vet_n;    { demand coefficients }
  c         : Vet_l;    { cost coefficients }
  d         : Vet_l;    { flow coefficients }
  t         : Vet_m;    { tail [Arc] }
  h         : Vet_m;    { head [Arc] }
  e         : Vet_m;    { null constant interval [Intr] }

  g_prob    : integer;  { number of feasible problems }
  inft      : integer;  { infinity }
  gene      : boolean;  { the seed is random or supplied by the user }
  n_save    : boolean;  { the last generated network is saved or not }
  tlimit    : integer;  { time limit for algorithm's execution }
  alg       : Str10;    { selected algoritms }

  sys,                  { configuration file }
  par,                  { parameter file }
  gen,                  { network file }
  sas,                  { cpu times for SAS statistical analysis }
  see,                  { seeds for network generation file }
  log       : text;     { log (accumulated results) file }

  etime,
  cpu,                  { cpu time for problem solving }
  ncpu,                 { cpu time for network generation }
  nitr,                 { number of iterations }
  nphas,                { number of phases }
  cx,                   { primal objective function value }
  yb,                   { dual objective function value }
  codg      : integer;  { solution code }
  m_cpu,                { mean value for cpu time }
  m_nitr,               { mean value for number of iterations }
  m_nphas   : Real4;    { mean value for number of phases }

procedure clear; external c;

procedure Configuration;
var
  i,g,s : integer;
begin
  open(sys,'net.sys','old');
  reset(sys);
  readln(sys,g_prob,inft,g,s,tlimit,alg);
```


cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

```
  gene := (g = 1);
  n_save := (s = 1);
  for i := 1 to length(alg) do begin
    if (alg[i] = 'p') or (alg[i] = 'P') then exmeth[1] := true
    else if (alg[i] = 'd') or (alg[i] = 'D') then exmeth[2] := true
    else if (alg[i] = 'k') or (alg[i] = 'K') then exmeth[3] := true
    else if (alg[i] = 's') or (alg[i] = 'S') then exmeth[4] := true;
    end;
  close(sys);
end;

function int2str(n,l : integer) : Str10;
var
  s : Str10;
  d : integer;
begin
  s := '';
  while n > 0 do begin
    d := n mod 10;
    s := chr(48+d) + s;
    n := n div 10;
    end;
  while length(s) < l do s := '0' + s;
  int2str := substr(s,1,l);
end;

procedure ReadPar;
var
  logname  : Str10;
  t_today,
  d_today  : alfa;
begin
  time(t_today);
  date(d_today);
  readln(par,nset,n,m,l,nfub,bmin,bmax,ptrans,dmin,dmax,cmin,cmax,pcap);
  nbrk := nfub + 3;
  if l > nbrk*m then l := nbrk*m;

  writeln(' ':5,'   ------------------------------------------------------------');
  writeln(' ':5,'    NODES      ARCS       NFUB     INTERV      PTRANS    PCAP');
  writeln(' ':5,n:9,m:10,nfub:10,l:12,ptrans:11:2,pcap:9:2);
  writeln(' ':5,'   ------------------------------------------------------------');
  writeln;
  writeln(' ':14,'CPU   NITER   NPHAS        CX        YB      CODG');

  logname := int2str(nset,3);
  logname := 'net'+logname;
  logname := logname+'.log';
  open(log,logname,'new');
  rewrite(log);

  writeln(log,' SYSMA/UNIX [Version 1994] Report ',' ':6,d_today:10,
              ' ':2,t_today:10,' ':5,'[',logname,']');
  writeln(log,'  ------------------------------------------------------------',
              '----------------------');
  writeln(log,'     N       M       L  NFUB  BMIN   BMAX  PTRANS  DMIN   DMAX',
              '  CMIN   CMAX  PCAP');
  writeln(log,n:6,m:8,l:8,nfub:5,bmin:7,bmax:7,ptrans:8:2,dmin:6,dmax:7,cmin:6,
              cmax:7,pcap:6:2);
  writeln(log);
  writeln(log,' ':16,'CPU   NITER   NPHAS        CX        YB      CODG');
end;

procedure SaveNet;
var
  i,j,k : integer;
begin
  rewrite(gen);
  for i := 1 to n do writeln(gen,b[i]:10);
  for j := 1 to m do writeln(gen,t[j]:10,h[j]:10,e[j]:10);
  for k := 1 to l do writeln(gen,d[k]:10,c[k]:10);
  close(gen);
```


cm 1 2 3 4 5 6 7 unesp 11 12 13 14 15 16 17 18

```
end;

{ ---
    Include modules for network generation and solution methods...
    ------------------------------------------------------------ }
#include "nplgen.p"
#include "primal.p"
#include "dual.p"
#include "kilter.p"
#include "scale.p"

begin
  method[1] := 'Primal ';
  method[2] := '  Dual ';
  method[3] := 'Kilter ';
  method[4] := ' Scale ';
  for i := 1 to 4 do exmeth[i] := false;

  Configuration;

  open(par,'net.par','old');
  reset(par);
  open(gen,'net.gen','new');
  if not gene then begin
    open(see,'net.see','old');
    reset(see);
    end;
  open(sas,'net.sas','new');
  rewrite(sas);

{ ---
    Set the random number generator seed...
    --------------------------------------- }
  rstart := sysclock;
  rstart := seed(rstart);

  clear;
  writeln(' ':20,'   Computational Experiments with');
  writeln(' ':20,'Network Piecewise Linear Programmes');
  writeln(' ':20,'===================================');
  writeln;
  while not eof(par) do begin
    ReadPar;
    nprob := 0;
    for i := 1 to 4 do begin
      m_cpu[i] := 0;
      m_nitr[i] := 0;
      m_nphas[i] := 0;
      end;
    nfnet := 0;
    repeat
{ ---
    Generate a network...
    -------------------- }
      if nfnet > 3*g_prob then begin
        writeln;
        writeln('>>>>>> ABORTED: Too Many Infeasible Generated Problems',
                ' (> 3*Sample Size)');
        writeln(log);
        writeln(log,'  >>>>>> ABORTED: Too Many Infeasible Generated Problems',
                    ' (> 3*Sample Size)');
        exit;
        end;
      if gene then
        n_seed := trunc(99999*random(rstart))
      else
        readln(see,n_seed);
      s_seed := n_seed;
      n_seed := seed(n_seed);
      NplGen;
      if n_save then SaveNet;
      writeln('NplGen ----------------------------------------------------------',
```

```
            ' [seed = ',s_seed:6,']');
    writeln(log,'  NplGen ------------------------------------------------',
            '------ [seed = ',s_seed:6,']');
{ ---
  Solve the network by the selected methods...
  ------------------------------------------- }
    for nm := 1 to 4 do
      if exmeth[nm] then begin
        case nm of
          1: Primal;
          2: Dual;
          3: Kilter;
          4: Scale;
          end;
        writeln(method[nm]:9,cpu:8,nitr:8,nphas:8,cx:10,yb:10,codg:9);
        writeln(log,' ':2,method[nm]:9,cpu:8,nitr:8,nphas:8,cx:10,yb:10,codg:9);
{ ---
  Collect the results either a optimal solution was obtained
  or the time limit for execution was exceeded...
  ---------------------------------------------------------- }
        if (codg = 3) or (codg = 4) then begin
          writeln(sas,nset:3,nm:1,' ',cpu);
          m_cpu[nm] := m_cpu[nm] + cpu;
          m_nitr[nm] := m_nitr[nm] + nitr;
          m_nphas[nm] := m_nphas[nm] + nphas;
          end
{ ---
  Otherwise, increase the number of infeasible problems...
  ------------------------------------------------------- }
        else begin
          nfnet := nfnet + 1;
          exit;
          end;
        end;
{ ---
  Replicate the experiment for max_prob problems...
  ------------------------------------------------ }
    if (codg = 3) or (codg = 4) then
      nprob := nprob + 1;
  until (nprob = g_prob);

{ ---
  Calculate the mean cpu time for each method...
  ---------------------------------------------- }
  if nprob > 0 then begin
    for i := 1 to 4 do begin
      m_cpu[i] := m_cpu[i] / nprob;
      m_nitr[i] := m_nitr[i] / nprob;
      m_nphas[i] := m_nphas[i] / nprob;
      end;
    writeln(log);
    writeln(log,' ':5,
            '------------------------ Mean Values ------------------------');
    writeln(log,' ':5,
            '                              CPU              NITER              NPHAS');
    writeln(log,' ':5,
            '-----------------------------------------------------------------');
    for i := 1 to 4 do
      if exmeth[i] then
        writeln(log,' ':6,method[i]:9,' ':2,m_cpu[i]:13:2,' ':5,m_nitr[i]:13:2,
                    ' ':6,m_nphas[i]:13:2);
    writeln(log,' ':5,
            '-----------------------------------------------------------------');
    writeln;
    writeln(' ':5,
            '------------------------ Mean Values ------------------------');
    writeln(' ':5,
            '                              CPU              NITER              NPHAS');
    writeln(' ':5,
            '-----------------------------------------------------------------');
    for i := 1 to 4 do
      if exmeth[i] then
```

```
          writeln(' ':6,method[i]:9,' ':2,m_cpu[i]:13:2,' ':5,m_nitr[i]:13:2,
                  ' ':6,m_nphas[i]:13:2);
      writeln(' ':5,
              '-------------------------------------------------------------------');
      end;
    close(log);
    end;
  close(par);
  if not gene then close(see);
  close(sas);
end.
```

G.2 - The Network Piecewise-Linear Generator Programme

```
procedure NplGen;
{ ---
    Network Piecewise Linear Generator
    ---------------------------------- }
var
  tj,                  { tail }
  hj,                  { head }
  lj,                  { breakpoint }
  som,sig : integer;   { balancing of demands }
  i,j,k   : integer;   { i: node, j: arc, k: interval }

function Rand : real;
var
  r : real;
begin
  r := random(n_seed);
  n_seed := trunc(inft*r);
  Rand := r;
end;

begin
  ncpu := clock;
{ ---
    Generating demands and accumulating the excess (or slack)...
    ------------------------------------------------------------ }
  som := 0;
  for i := 1 to n do
    begin
    if Rand < ptrans then
      b[i] := 0
    else
      b[i] := round(bmin + (bmax-bmin) * Rand);
    som := som - b[i];
    end;

{ ---
    Distributing units of excess (or slack) among the demands...
    ------------------------------------------------------------ }
  sig := 1;
  if som < 0 then sig := -1;
  while (som <> 0) do begin
    i := trunc(1 + n * Rand);
    if (som <> 0) then
      if (b[i] <> 0) and
         (b[i]+sig >= bmin) and (b[i]+sig <= bmax) then begin
        b[i] := b[i]+sig;
        som := som-sig;
        end;
    end;

  lj := 2;
  for j := 1 to m do begin
    repeat
      tj := trunc(1 + n * Rand);
      hj := trunc(1 + (n-1) * Rand);
      if hj >= tj then hj := hj + 1;
```

```
    until (b[tj] <= 0) and (b[hj] >= 0);
    t[j] := tj;
    h[j] := hj;
    e[j] := lj;
    lj := lj + nbrk;
    if (lj+nbrk > l) then lj := 2;
    end;

  if nbrk = 3 then begin
    k := 0;
    repeat
      d[k+1] := -inft;
      c[k+1] := -inft;
      d[k+2] := dmin;
      c[k+2] := round(cmin + (cmax-cmin) * Rand);
      d[k+3] := inft;
      c[k+3] := inft;
      k := k + nbrk;
    until k >= l;
    end
  else begin
    k := 0;
    repeat
      d[k+1] := -inft;
      c[k+1] := -inft;
      d[k+2] := dmin;
      c[k+2] := round(cmin + (cmax-cmin) * Rand / (nbrk-2));
      for i := 1 to nbrk-4 do begin
        d[k+2+i] := round(dmin + (dmax-dmin) * (Rand + i - 1) / (nbrk-3));
        c[k+2+i] := round(cmin + (cmax-cmin) * (Rand + i) / (nbrk-2));
        end;
      d[k+nbrk-1] := round(dmin + (dmax-dmin) * (Rand + nbrk-4) / (nbrk-3));
      if Rand < pcap then
        c[k+nbrk-1] := inft
      else
        c[k+nbrk-1] := round(cmin + (cmax-cmin) * (Rand + nbrk-3) / (nbrk-2));
      d[k+nbrk] := inft;
      c[k+nbrk] := inft;
      k := k + nbrk;
    until k >= l
    end;
  ncpu := (clock - ncpu) div 10;
end;
```


unesp