INVESTIGAÇÃO OPERACIONAL

Dezembro 1993
Número 2
Volume 13

Publicação Científica da

Associação Portuguesa para o Desenvolvimento
da Investigação Operacional

INVESTIGAÇÃO OPERACIONAL

Propriedade:
APDIO — Associação Portuguesa para o Desenvolvimento da Investigação Operacional

ESTATUTO EDITORIAL

«Investigação Operacional», órgão oficial da APDIO cobre uma larga gama de assuntos reflectindo assim a grande diversidade de profissões e interesses dos sócios da Associação, bem como as muitas áreas de aplicação da I. O. O seu objectivo primordial é promover a aplicação do método e técnicas da I.O. aos problemas da Sociedade Portuguesa.

A publicação acolhe contribuições nos campos da metodologia, técnicas, e áreas de aplicação e software de I. O. sendo no entanto dada prioridade a bons casos de estudo de carácter eminentemente prático.

Distribuição gratuita aos sócios da APDIO

INVESTIGAÇÃO OPERACIONAL

Volume 13 - nº 2 - Dezembro 1993 Publicação semestral

Editor Principal: Joaquim J.Júdice
Universidade de Coimbra

Comissão Editorial

A Revista "INVESTIGAÇÃO OPERACIONAL" está registada na Secretaria de Estado da Comunicação Social sob o nº 108335.
Esta Revista é distribuida gratuitamente aos sócios da APDIO. As informações sobre inscrições na Associação, assim como a correspondência para a Revista devem ser enviadas para a sede da APDIO - Associação Portuguesa para o Desenvolvimento da Investigação Operacional - CESUR, Instituto Superior Técnico, Av. Rovisco Pais, 1000 Lisboa.

Este Volume foi subsidiado por :

Instituto Nacional de Investigação Científica (INIC)

Junta Nacional de Investigação Científica e Tecnológica (JNICT)

Fundação Calouste Gulbenkian

Para efeitos de dactilografia e composição, foram utilizados equipamentos gentilmente postos à disposição pelo CEAUL (DEIO - Faculdade de Ciências de Lisboa).

Assinatura: 5.000$00

INVESTIGAÇÃO OPERACIONAL, vol. 13, n.° 2 (Dezembro 1993) 101-117

A GERAÇÃO RÁPIDA DE COLUNAS: UMA ALTERNATIVA NA TÉCNICA DE GILMORE E GOMORY PARA PROBLEMAS DE CORTE

José Fernando da Costa Oliveira
José António Soeiro Ferreira
DEEC - Faculdade de Engenharia da Univ. Porto
INESC - Instituto de Engenharia de Sistemas e Computadores (Porto)

Abstract

The delayed (or implicit) column generation technique was for the first time applied to the cutting stock problem resolution by Gilmore and Gomory in the early sixties. Since then, this technique has been widely applied, mainly to solve one-dimensional cutting problems, i.e., cutting problems where only one of the objects dimensions is relevant for the problem. Its application to two-dimensional cutting problems has been strongly conditioned by the computational effort needed to solve the auxiliary problems (knapsack problems) generated by the application of this technique.

Gilmore and Gomory presented several algorithms for these special knapsack problems. Considering that their performance is crucial for the performance of the technique itself, other authors also developed algorithms for these auxiliary problems resolution, each one considering different constraints on the cutting patterns, or tried to improve the original Gilmore and Gomory algorithms.

In this paper a different approach is presented, the fast column generation. The idea behind this approach has been to increase the performance of the columns generation technique proposed by Gilmore and Gomory by reducing the number of items that the auxiliary problem is solved until optimality instead of trying to accelerate the resolution of each problem. Although this approach can be applied to cutting problems of any dimension, in this paper it will be presented and tested over two-dimensional rectangular cutting problems.

Resumo

A técnica da geração atrasada (ou implícita) de colunas, aplicada à resolução de problemas de cortes, foi pela primeira vez proposta por Gilmore e Gomory no início dos anos sessenta. Desde logo, e sempre que os problemas concretos se adaptavam às restrições inerentes à própria técnica, ela foi vastamente aplicada, sobretudo para resolver problemas de cortes a uma dimensão, isto é, problemas em que apenas uma das dimensões dos objectos a cortar é relevante para o problema dos cortes.

A aplicação desta técnica a problemas de cortes bidimensionais, foi fortemente condicionada pelo elevado peso computacional da resolução dos problemas auxiliares (problemas mochila) que a técnica supõe e para os quais Gilmore e Gomory apresentaram vários algoritmos. Dada a especificidade dos problemas mochila que surgem na resolução de problemas de corte, e a sua importância para a eficiência da técnica de geração atrasada de colunas, outros autores tentaram desenvolver novos algoritmos para a resolução dos problemas auxiliares, diferentes conforme as restrições consideradas para os padrões de corte, ou mesmo adaptar e melhorar os propostos por Gilmore e Gomory.

Neste artigo apresentamos uma abordagem diferente, a geração rápida de colunas. A ideia subjacente a esta abordagem consiste não em tentar acelerar a resolução dos problemas auxiliares, mas em reduzir o número de vezes que eles são resolvidos, tornando assim a técnica de Gilmore e Gomory bastante mais eficiente. Embora esta abordagem seja aplicável a problemas de cortes de qualquer dimensão, ao longo deste artigo ela será apresentada e testada sobre problemas bidimensionais (formas rectangulares).

Keywords

cutting-stock, delayed column generation, 2D cutting patterns.

1. Introdução

A resolução de um problema de cortes consiste em, dado um conjunto de objectos "grandes", ditos de stock, de que se dispõe, e um outro conjunto de objectos "pequenos", ditos pedidos, que se pretende obter a partir dos primeiros, encontrar a forma de dividir os objectos grandes nos pequenos, de acordo com um dado objectivo (normalmente a minimização dos desperdícios, isto é, das formas que se obtêm dos objectos grandes e que não pertencem ao conjunto dos objectos pedidos). Dá-se o nome de padrão de corte a uma forma de dividir um objecto grande em pequenos (Figura 1). A solução de um problema de cortes será então um ou mais padrões de corte. Para uma descrição mais pormenorizada da tipologia dos problemas de cortes, ou uma classificação das várias formas do problema e técnicas de resolução, sugerimos [2], [3], [11], [13] e [14].

No início dos anos sessenta Gilmore e Gomory apresentaram uma nova técnica para a resolucão do problema dos cortes multi-padrão, isto é, aquela forma do problema em que se pretende determinar os padrões de corte que satisfazem todos os pedidos (isto em oposição à forma uni-padrão em que, dado o conjunto de pedidos, apenas se pretende obter um padrão de corte, reduzindo assim a lista de pedidos). A resolução deste tipo de problemas assenta num modelo de Programação Linear (PL), em que a cada coluna da matriz das restrições está associado um padrão de corte, e na sua resolução pelo método simplex.

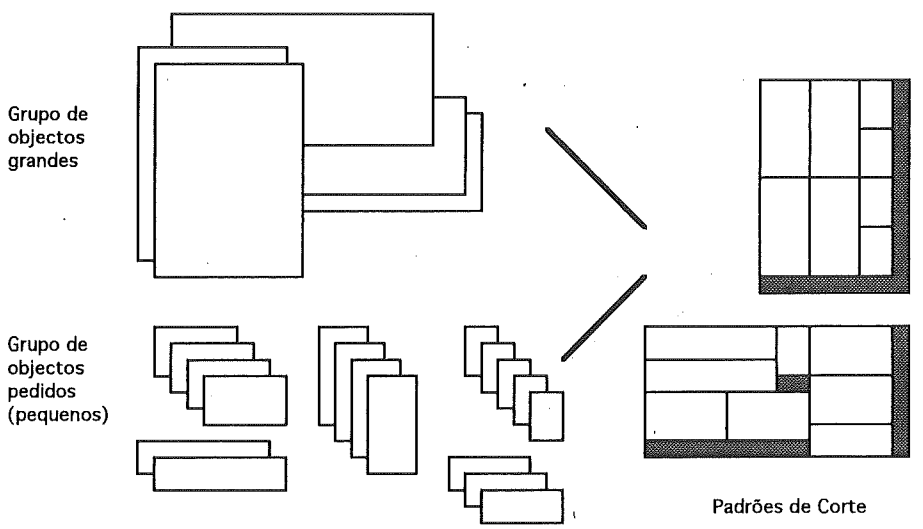


Figura 1 – Estrutura básica de um problema de corte

Para evitar a consideração explícita de todos os padrões de corte possíveis (que poderiam atingir alguns milhões), Gilmore e Gomory propuseram a técnica de geração atrasada (ou implícita) de colunas. Usando esta técnica, a resolução do problema PL arranca com algumas colunas (padrões) apenas, sendo as restantes geradas apenas à medida que forem, e se forem, necessárias. Para determinar se uma nova coluna deve ser gerada e introduzida no problema, é resolvido um problema auxiliar em cada iteração do método Simplex, problema este que toma a

forma de um problema mochila com duas restrições "normais" e mais algumas específicas da natureza geométrica do problema. Esta técnica será descrita na secção 2 deste artigo.

No intuito de melhorar a eficiência da técnica de geração atrasada de colunas, aplicada a problemas de cortes a duas dimensões (rectangulares), vários autores se debruçaram sobre a resolução do problema auxiliar, nomeadamente Herz [10], Farley [5] e Beasley [1]. Na abordagem por nós proposta neste artigo não procuramos resolver o problema auxiliar mais depressa, mas sim resolvê-lo menos vezes. Esta abordagem será vantajosa se o tempo que se gasta, para que o problema auxiliar seja resolvido menos vezes, fôr inferior ao tempo que se ganha com a sua não resolução. Na secção 3 esta nova abordagem é descrita, sendo apresentados na secção 4 os resultados dos testes computacionais efectuados, comparando a abordagem da geração rápida de colunas com a técnica clássica de Gilmore e Gomory.

2. A técnica da geração atrasada de colunas de Gilmore e Gomory

A resolução de um problema de cortes multi-padrão passa pela geração de todos os padrões de corte admissíveis, em que a admissibilidade de um padrão depende das restrições geométricas consideradas em cada caso concreto (cortes em guilhotina ou não, em duas, três ou n fases, etc. - Figuras 2 e 3), e pela determinação de quantas vezes cada um desses padrões deve ser utilizado, de forma a satisfazer os pedidos existentes e a minimizar o número de chapas de stock gastas na satisfação desses pedidos.

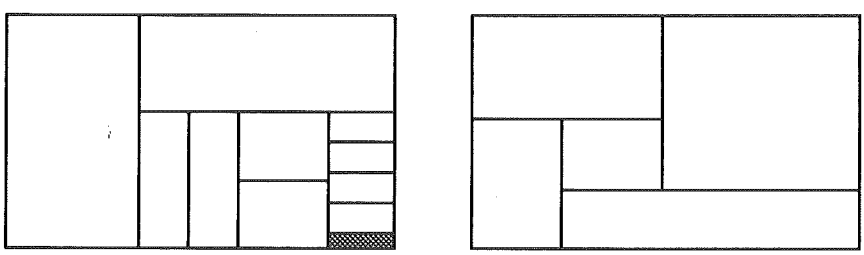


Figura 2 – Padrão de corte em guilhotina e não guilhotinável

Formulando matematicamente este problema chega-se, praticamente por transposição directa, a um modelo de Programação Linear (PL) com a seguinte estrutura:

$$\min \quad \sum_j x_j \qquad (1)$$

sujeito a:

$$\sum_j a_{ij} x_j = b_j, \qquad i = 1,\ldots, m$$

$$x_j \geq 0 \text{ e inteiro}, \quad j = 1,\ldots, n$$

em que:

x_j - número de vezes que padrão de corte j é utilizado;
b_i - número de peças do tipo i que deve ser cortado;
a_{ij} - número de peças do tipo i que o padrão de corte j gera;
$[a_{1j},\ldots, a_{mj}]^T$ - coluna j da matriz das restrições = padrão de corte j;
m - número de peças diferentes a cortar;
n - número de padrões de corte.

É de salientar que o modelo apresentado não inclui qualquer consideração do tipo geométrico, pois supõe que os padrões de corte já gerados, isto é, que as colunas do problema já estão definidas. O modelo é portanto válido para problemas de cortes de qualquer dimensão.

Este modelo apresenta contudo duas dificuldades de ordem prática. A primeira delas tem a ver com a integralidade das soluções. A exigência de integralidade obriga a recorrer a técnicas de resolução que, mesmo para problemas relativamente pequenos, tornam esta formulação completamente inviável. Por esta razão, é habitual relaxar o problema, admitindo soluções não inteiras e arredondando-se no fim [6]. Em segundo lugar, para que se obtenha a solução óptima, mesmo considerando o modelo relaxado, é necessário que estejam presentes no problema todos os padrões de corte admissíveis, isto é, todos os padrões de corte que respeitam as restrições do problema. Dado o carácter combinatório da geração de padrões, o número de padrões admissíveis é sempre muito elevado, mesmo para problemas aparentemente pequenos, originando um problema de PL com um número intratável de colunas (padrões de corte).

É neste contexto que Gilmore e Gomory, [6] e [7], sugerem a aplicação da técnica de geração atrasada de colunas à resolução dos Problemas de Cortes. Esta técnica arranca o problema de PL apenas com algumas das muitas colunas possíveis, e em cada iteração, se tal for vantajoso, cria e insere uma nova coluna no problema. Esta abordagem baseia-se na utilização do método Simplex para a resolução do problema de PL e utiliza os custos marginais das variáveis não básicas da forma que seguidamente se apresenta.

Num problema de minimização (no caso presente, o modelo (1) sem as restrições de integralidade), em cada iteração do método Simplex entra na base a variável não básica que tiver um custo marginal mais negativo. Como o custo marginal de uma variável não básica é dado pela expressão:

$$\bar{c}_j = 1 - \sum_i \pi_i a_{ij}$$

em que π_i são os multiplicadores para cada uma das restrições, isto é:

$$B^T \Pi = C_B$$

em que:

B - matriz base;
Π - vector dos multiplicadores, π_i;
C_B - coeficientes das variáveis básicas,

o problema de minimizar $\bar{c}_j$ sobre todas as colunas possíveis, quer as que já estão no problema quer aquelas que não foram ainda geradas, transforma-se no problema de maximizar:

$$\sum_i \pi_i a_{ij}$$

sob a restrição de $[a_{1j},\ldots, a_{mj}]^T$ descrever um padrão de corte admissível.

Considerando

w_p - comprimento do padrão de corte descrito por $[a_{1j},\ldots, a_{mj}]^T$;

h_p - largura do padrão de corte descrito por $[a_{1j},\ldots, a_{mj}]^T$;

w - comprimento da chapa de stock;

h - largura da chapa de stock;

a admissibilidade do padrão $[a_{1j},\ldots, a_{mj}]^T$ passará, para além de outras, pelas condições:

$w_p \leq w$

$h_p \leq h$

a_{ij} = inteiro não negativo, para todo o i

A procura da variável que em cada iteração do método Simplex deve entrar na base, resulta então na resolução do seguinte problema auxiliar:

$$\text{max} \quad V = \sum_i \pi_i \, a_{ij}$$

$$\text{sujeito a:} \quad w_p \leq w$$

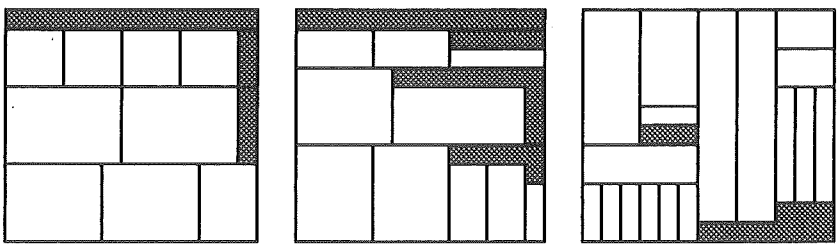
$$h_p \leq h$$

$$a_{ij} \text{ inteiros não negativos}$$

Se $V > 1$, o padrão respectivo deve ser construído e a coluna correspondente entrar na base, senão a solução corrente é óptima (todos os custos marginais são não negativos). Como se pode constatar, este problema auxiliar toma a forma de um problema mochila com duas restrições "normais" ($w_p \leq w$ e $h_p \leq h$), para além das restrições geométricas que se colocam na construção do padrão de corte.

A viabilidade da abordagem da geração e introdução das colunas no problema de PL apenas se, e quando, elas melhorarem o valor da função objectivo, está sobretudo dependente da forma como se vão gerar os padrões que dão origem a custos marginais negativos, o que no caso desta abordagem proposta por Gilmore e Gomory se traduz na existência de técnicas eficientes para a resolução do problema mochila. Em [6] Gilmore e Gomory apresentam um algoritmo de Programação Dinâmica (PD) para o problema a uma dimensão. Em [7] os mesmos autores, e ainda para o caso a uma dimensão, apresentam um algoritmo de pesquisa lexicográfica que afirmam ser cinco vezes mais rápido que a técnica de PD. Em [8] os autores abordam o problema da resolução do problema mochila para o caso dos cortes a duas dimensões. Neste artigo é considerado o caso dos padrões de corte tipo guilhotina em duas ou três fases. Esta abordagem consiste basicamente em decompor a resolução do problema em duas partes: na primeira parte são geradas bandas óptimas, no sentido de maximizarem o valor das peças que se consegue encaixar numa banda com largura y, e na segunda parte as bandas são escolhidas e

combinadas de forma a formarem um padrão 2D com o maior valor possível. A diferença entre a geração de padrões para duas ou três fases de corte encontra-se apenas na parte da geração das bandas óptimas. Enquanto que para padrões em duas fases se exige que todas as peças rectangulares tenham a mesma largura, para os padrões em três fases apenas se exige que a largura das peças não exceda a largura prevista para a banda (Figura 3 (i) e (ii)). Em ambos os casos, os dois problemas a uma dimensão que surgem (um na geração das bandas óptimas e outro na sua colocação) são resolvidos através do algoritmo de pesquisa lexicográfica já mencionado.



(i) padrão de corte em duas fases

(ii) padrão de corte em três fases

(iii) padrão de corte em n fases

Figura 3 – Padrão de corte em duas, três e n fases

Farley, em [5], analisa este algoritmo de pesquisa lexicográfica e faz algumas adaptações, nomeadamente na geração do conjunto inicial de padrões com que a PL arranca, na consideração de soluções múltiplas por problema mochila, na valorização explícita de sub e sobreposição de peças e na valorização das sobras, isto é, daquilo que habitualmente é encarado como desperdício.

Entretanto, em [9], Gilmore e Gomory apresentam uma rotina, baseada em PD, para aresolução do problema mochila a duas dimensões, sem mais restrições além da dos cortes serem do tipo guilhotina, ou seja, padrões em n fases (Figura 3 (iii)).

Na próxima secção será apresentada uma abordagem diferente à técnica de Gilmore e Gomory, por nós desenvolvida. Esta abordagem, é válida independentemente do algoritmo escolhido para a resolução dos problemas mochila que irão surgir. Na escolha do algoritmo de resolução dos problemas mochila deverá sempre pesar o facto de o algoritmo de PD a duas dimensões gerar padrões mais genéricos (com menos restrições) que a dupla pesquisa lexicográfica, embora requerendo mais tempo de cálculo. Estas diferenças serão quantificadas na secção 4, onde serão apresentados os resultados dos testes computacionais efectuados.

3. Uma abordagem diferente à técnica de geração atrasada de colunas: a geração rápida de colunas

A utilização da técnica de geração atrasada de colunas na resolução do problema dos cortes, formulado como problema de PL, se por um lado viabiliza a obtenção da solução óptima (modelo (1) relaxando as condições de integralidade), sem implicar a geração de todos os

padrões de corte admissíveis, por outro lado, ao obrigar à resolução de um problema auxiliar em cada iteração do método Simplex, torna o custo computacional da resolução de problemas de corte de dimensão média, extremamente elevado, quer em termos de tempo quer em termos de memória.

Os diversos autores que se têm debruçado sobre a técnica de programação linear com geração atrasada de colunas, têm procurado aumentar o seu desempenho através da procura de algoritmos mais eficientes para a resolução do problema auxiliar. Na nossa abordagem, a geração rápida de colunas, procura-se reduzir o peso da resolução do problema auxiliar diminuindo ao número de vezes que ele é resolvido.

Na programação linear com geração atrasada de colunas, como já foi discutido, para decidir em cada iteração qual a variável que deve entrar na base, seguindo o critério dos custos marginais mínimos, é necessário então resolver o problema auxiliar:

$$\max \quad \sum_i \pi_i \, a_{ij} \qquad (2)$$

sujeito a: $[a_{1j},\ldots, a_{mj}]$ representa um padrão de corte admissível, para todo o j.

A admissibilidade do padrão de corte dependerá das restrições que à partida sejam impostas, ou pela própria tecnologia de corte, ou pelo facto de se pretender chegar a um modelo cuja resolução seja viável. Por exemplo, se por um lado algumas ferramentas de corte só poderão executar padrões do tipo guilhotina, por outro lado não existe nenhuma técnica que resolva o problema (2) se os padrões não incluirem a restrição de serem guilhotináveis.

Posta de parte a hipótese de resolver o problema (2) para padrões de corte não guilhotináveis, a hipótese seguinte seria a utilização da rotina de PD, apresentada por Gilmore e Gomory [9], que gera padrões com cortes em guilhotina e em n fases. A viabilidade da aplicação desta técnica vai contudo depender fortemente da dimensão do problema, pois envolve um número de operações O(wh(w+h)) e uma capacidade de armazenamento de números O(wh).

Embora alguns autores tenham introduzido algumas modificações neste algoritmo (Beasley [1]), ou mesmo desenvolvido algoritmos alternativos (Herz [10]), as vantagens retiradas das alterações introduzidas não compensam o acréscimo de dificuldade na implementação desses algoritmos, nem resolvem o verdadeiro problema do algoritmo de Gilmore e Gomory, isto é, continuam a não ser praticáveis para problemas médios ou grandes [1].

Numa terceira abordagem, poder-se-ia introduzir mais uma restrição no problema auxiliar e considerar-se apenas padrões num número pré-determinado de fases, usualmente duas ou três. Computacionalmente muito menos pesada, Farley [5] pega nesta hipótese, implementa-a e introduz mesmo no algoritmo de pesquisa lexicográfica, apresentado por Gilmore e Gomory [7] para estes problemas, algumas capacidades adicionais. Ainda no campo da utilização de padrões em duas ou três fases, Beasley [1] implementa e modifica o algoritmo de PD de Gilmore e Gomory [6].

De uma forma geral, a perspectiva dos autores que têm abordado a questão da resolução do problema auxiliar na geração atrasada de colunas, tem sido a de encontrar técnicas mais eficientes para a resolução desse problema.

A abordagem por nós proposta, a geração rápida de colunas, assenta num princípio diferente: procura-se resolver menos vezes o problema auxiliar. A filosofia subjacente a esta abordagem não passa pela minimização do custo marginal. Assim, em cada iteração procura-se uma coluna que, entrando na base, melhore a função objectivo, e não aquela coluna que potencialmente mais melhoraria a função objectivo (aquela que possui um custo marginal mínimo). Uma consequência imediata desta abordagem é o aumento do número de iteraçnoes para atingir a solução óptima, o que implica um acréscimo de tempo na resolução do problema de PL. Contudo, se cada iteração for mais rápida, o produto *tempo por iteração* × *nº de iterações* poderá ser favorável. Foi sob esta premissa que a abordagem da geração rápida de colunas foi desenvolvida.

Não existindo, em cada iteração, o objectivo da minimização dos custos marginais, mas apenas o da determinação de uma variável que tenha um custo marginal negativo, convirá ainda que, sem se procurar o custo marginal óptimo, se encontrem colunas com um custo marginal bastante negativo, isto para que o número de iterações não cresça em demasia. Para construir um padrão a que corresponda um bom custo marginal, implementou-se uma heurística que gera padrões tipo guilhotina em duas, três ou n fases, conforme a opção, uma vez que a abordagem da geração rápida de colunas é perfeitamente genérica e independente do tipo de padrão em causa. Esta heurística é "glutona", pois, respeitando as restrições (geométricas) impostas aos padrões de corte, coloca as peças por ordem decrescente de valor, isto é, começando pelas mais valiosas (π_i elevado), de forma a que

$$1 - \sum_i \pi_i \, a_{ij}$$

venha bastante negativo.

Naturalmente, existirão certos momentos na resolução do problema PL em que a heurística será incapaz de gerar um padrão com custo marginal negativo. Para obter a solução óptima do problema PL é então forçoso resolver de facto o problema auxiliar da minimização dos custos marginais. Note-se, contudo, que este problema só será resolvido quando a heurística falha na tentativa de gerar um padrão de corte com um custo marginal negativo.

Resumindo, num problema de cortes, ao resolver o problema de PL, com geração rápida de colunas, para determinar em cada iteração a variável que entra na base, seguir-se-á o seguinte procedimento:

calcula π_i, i = 1,..., m;
calcula os custos marginais das variáveis presentes no problema;
seja c_j o menor desses custos marginais e x_j a respectiva variável;
se $c_j < 0$ então x_j entra na base
    senão

corre a heurística;
seja c_j o custo marginal da variável correspondente ao padrão gerado pela heurística, x_j;
se $c_j < 0$ então x_j entra na base
senão
resolve o problema auxiliar da minimização dos custos marginais;
seja c_j o custo marginal da variável correspondente ao padrão gerado, x_j;
se $c_j < 0$ então x_j entra na base
senão a base actual corresponde à solução óptima

4. Estudo comparativo do desempenho da geração atrasada de colunas e da geração rápida de colunas: resultados computacionais

Nesta secção compara-se o desempenho da técnica clássica de Gilmore e Gomory para a resolução de problemas de cortes multi-padrão, Programação Linear com geração atrasada de colunas, com a Programação Linear com geração rápida de colunas.

A utilização de um modelo de PL para o problema dos cortes, introduz à partida uma simplificação, uma vez que um problema de cortes é inerentemente um problema inteiro. Por isso, após a resolução da PL é necessária uma outra fase, a que se chamará fase de pós-optimização, onde as soluções não inteiras são transformadas em soluções inteiras. Na primeira parte desta secção várias alternativas serão analisadas, sendo apresentada a abordagem escolhida para os presentes testes computacionais.

Outra questão que a implementação destas técnicas levanta, é a da geração do conjunto inicial de colunas/padrões para a PL. De facto, quer a geração atrasada de colunas quer a geração rápida de colunas, pressupõem a existência de uma base admissível inicial para o problema PL. Este assunto será também abordado na parte inicial desta secção.

Finalmente, e antes da apresentação dos resultados computacionais propriamente ditos, faz-se uma descrição da estrutura da experiência efectuada, o que inclui os indicadores de desempenho, o conjunto de testes e os diferentes tipos de problemas usados. Em seguida, e após a apresentação dos resultados computacionais, são retiradas algumas conclusões.

Por uma questão de simplicidade e clareza, a técnica de Programação Linear com geração atrasada de colunas será designada por PLGAC, e a Programação Linear com geração rápida de colunas por PLGRC.

Integralidade da solução

Obtida a solução do problema de PL, que no caso geral será não inteira, surge a dificuldade de a transformar numa solução inteira, uma vez que não faz sentido dizer que um determinado padrão de corte deve ser utilizado, por exemplo, duas vezes e meia.

A forma mais imediata de obter a integralidade seria por arredondamento da solução não inteira. Contudo, o arredondamento simples das soluções, segundo a regra convencional de arredondar para cima quando a primeira casa decimal é maior ou igual a 5, e no caso contrário arredondar para baixo, faria com que a solução inteira assim obtida não verificasse as restrições do problema PL, isto é, daria origem a um excesso no número de peças cortadas de alguns tipos (sobreproduções), motivado pelos arredondamentos superiores, e à falta de peças de outros tipos (subproduções), motivadas pelos arredondamentos inferiores. Em processos produtivos em que sobre e subproduções sejam admitidas, esta é a forma mais prática de obter uma solução inteira.

Existem, porém muitas situações concretas onde, embora se possam admitir sobreproduções, as subproduções são completamente inadmissíveis, uma vez que tal significaria a não satisfação de clientes ou de outros sectores dentro do mesmo processo produtivo. Nesses casos, é possível arredondar os valores não inteiros todos superiormente. Deste modo serão evitadas as subproduções, embora à custa de um aumento significativo das sobreproduções.

Em [5], Farley estuda os efeitos do arredondamento das soluções, tomando como ponto de arredondamento não só o 0.5, mas também outros valores, desde o 0.0 (todos os valores arredondados superiormente) ao 0.8 (quase todos os valores arredondados inferiormente). Nesse estudo, Farley conclui que o impacto do arredondamento depende sobretudo de dois factores:

– o número de vezes que cada peça surge num padrão de corte, e
– os pedidos existentes para cada peça.

Se o número de vezes que uma dada peça surge num padrão for elevado, pequenas alterações no número de vezes que o padrão será utilizado, levarão a grandes alterações no sobre ou subprodução dessa peça. Este facto poderia inclusivé levar à consideração de um limite superior ao número de vezes que cada peça surgiria num padrão de corte. Por outro lado, se as quantidades pedidas, de uma dada peça, forem elevadas, qualquer sub ou sobreprosução será percentualmente menor e logo mais aceitável segundo todos os pontos de vista.

Quando qualquer sub ou sobreprodução é inaceitável, então estas técnicas não respondem à situação. Uma abordagem possível é o arredondamento inferior de todos os valores (truncagem) e consequente resolução de um novo problema de cortes com os pedidos não satisfeitos. Este processo seria sucessivamente repetido até que as quantidades em jogo fossem suficientemente pequenas para que o problema pudesse ser resolvido manualmente ou através de uma heurística. A desvantagem deste método reside no facto de em cada iteração serem muito provavelmente criadas soluções com cada vez maiores percentagens de desperdício, para além do tempo que tal processo iterativo tomaria. Uma alternativa a este método seria o arredondamento inferior dos valores das variáveis do problema de PL e a utilização de uma heurística, intrinsecamente inteira, para gerar os padrões que cobririam as subproduções. Um

método deste tipo teria a vantagem de ser mais rápido e de fornecer bons resultados, em termos de desperdícios.

Nos testes efectuados considerou-se a situação da inadmissibilidade de sub e sobreproduções, dado ser o caso mais complexo, tendo-se optado pelo arredondamento inferior (truncagem) seguido duma heurística. A heurística escolhida foi a *First Fit Decreasing Height* [12].

Geração dos padrões iniciais

Para que a resolução do problema de PL possa arrancar, é necessário constituit uma base admissível para o problema, devendo para tal existir um conjunto inicial de colunas/padrões. A questão que se pode colocar, é se diferentes conjuntos iniciais de padrões darão origem a diferenças significativas no tempo necessário à obtenção da solução do problema PL (com geração, atrasada ou rápida, de colunas).

Se existem m peças diferentes a cortar, então a abordagem mais simples consistirá em começar com m padrões, cada um dos quais contendo apenas uma peça, e uma peça diferente de padrão para padrão. Uma extensão desta abordagem seria incluir o maior número possível de peças de um dado tipo, em cada padrão. Tal poderia ser feito gerando riscas e depois empilhando-as sucessivamente. Contudo, as experiências mostraram que os padrões constituídos por peças de um só tipo raramente são incluídos na solução óptima, dado conterem habitualmente níveis relativamente elevados de desperdício, pelo que não se justifica o acréscimo de tempo que esta abordagem provocaria na geração do conjunto dos padrões iniciais.

Farley [5] sugere uma abordagem de geração dos padrões iniciais por minimização do desperdício, isto na linha dos trabalhos realizados por Dyson e Gregory [4]. Os passos que implementariam tal abordagem seriam:

1. Defina o conjunto S como o conjunto de todas as peças pedidas.
2. Resolva um problema mochila 2D (geração de um padrão) considerando todas as peças de S e atribuindo a cada uma um valor igual à sua área.
3. Se todas as peças já foram icnluídas em pelo menos um padrão então pare, senão:
4. Redefina o conjunto S como sendo todas as peças ainda não incluídas num padrão e vá para 2.

O resultado do passo 2 será um padrão contendo o menor desperdício possível, quando se considera o conjunto das peças pertencentes a S e o tipo de padrão imposto pela técnica usada para a resolução do problema mochila, nomeadamente no que diz respeito ao número de fases do padrão (nos testes efectuados foram considerados padrões em 3 e n fases). Esta abordagem produz um conjunto de padrões iniciais em que cada um contém uma combinação de várias peças, dando por isso origem a desperdícios relativamente pequenos, o que aumenta a probabilidade de virem a pertencer ao conjunto final, óptimo, de padrões. Note-se que o facto

de cada aparecer num e num só parão, garante a admissibilidade da solução formada por estes padrões.

A partir dos resultados de Farley [5] e da nossa própria experiência, obtida nalguns testes preliminares em que todas estas abordagens foram experimentadas, optou-se por gerar os padrões iniciais por desperdício mínimo. Embora esta abordagem seja francamente mais demorada, a diminuição no número de iterações da PL compensa o tempo gasto, sendo os tempos totais (geração dos padrões iniciais mais resolução da PL) menores do que os correspondentes à geração de padrões de peça única. A diferença de tempos dependerá sobretudo do tipo de padrão em causa, sendo mais vantajoso para os padrões em n fases, e do facto de se usar PLGAC ou PLGRC, sendo bastante mais vantajosos utilizar padrões iniciais por desperdício mínimo na PLGAC do que na PLGRC, onde por vezes chega a ser desvantajoso. De facto, a vantagem da utilização de bons padrões iniciais será tanto maior quanto mais demoradas forem, em média, as iterações da PL, o que acontecerá quando se trabalhar com padrões em n fases ou quando se usar PLGAC.

Indicadores de desempenho

Foram escolhidos os seguintes indicadores de desempenho:

- o tempo de execução (em segundos de CPU);
- o número de iterações;
- o número de padrões gerados;
- o número de chapas de stock gastas na solução final.

O tempo de execução, medido em segundos de CPU, refere-se ao VAX 11/750, correndo VMS, máquina onde os testes computacionais foram realizados. O programa que implementa as técnicas PLGAC e PLGRC foi escrito em PASCAL. Sempre que possível foram utilizadas rotinas comuns para a PLGAC e PLGRC, de modo a que os tempos de execução obtidos reflectem fielmente os diferentes desempenhos das duas técnicas. Com este mesmo objectivo, a apresentação dos tempos de execução foi dividida em duas parcelas, uma respeitante à geração dos padrões iniciais, igual para ambas as técnicas, e outra respeitante à PL propriamente dita. A fase de pós-optimização (obtenção de uma solução inteira) foi incluída nesta segunda parcela, mas o tempo que lhe corresponde é desprezável face ao tempo de execução da PL.

Embora po si só, o número de iterações não seja um indicador relevante, uma vez que os tempos por iteração não vão ser iguais na PLGAC e na PLGRC, será interessante verificar até que ponto a geração rápida de colunas aumenta o número de iterações, assim como determinar tempos médios por iteração e assim verificar as premissas que fundamentaram o desenvolvimento da PLGRC.

O número de padrões gerados, a par com o tempo de execução, constitui um indicador de desempenho valioso e fiável. Considerando que a maior parte do tempo gasta na PL corresponde à geração de padrões, este indicador, ao medir o número de padrões gerados pela

heurística e pelas técnicas de Gilmore e Gomory, dará uma indicação sobre o desempenho ds PLGAC e da PLGRC, independente do computador usado.

Finalmente, o número de chapas de stock gastas na solução final será o indicador que garantirá a optimalidade das soluções encontradas pela PLGRC, por comparação com os valores obtidos pela PLGAC. Note-se, contudo, que estes números poderão diferir ligeiramente, uma vez que se referem a valores posteriores à fase de pós-optimização, podendo soluções óptimas alternativas, portanto com o mesmo valor da função objectivo, ao sofrer arredondamentos e posterior pós-optimização, dar origem a alguma variação na solução inteira final.

Estrutura da experiência

Os testes computacionais efectuados foram divididos em duas fases, de acordo com os grupos de problemas utilizados.

Na primeira fase, utilizaram-se 10 problemas considerados típicos na distribuição das várias dimensões em jogo [12]. As suas características são apresentadas na tabela 1.

Tabela 1 – Características dos problemas da primeira fase de testes

Chapa de stock		Nº de peças diferentes	Comprimento das peças (w_i)		Largura das peças (h_i)		Pedidos por peça (b_i)	
Comp. (w)	Largura (h)	(m)	min	max	min	max	min	max
70	40	10	6	66	3	22	10	40

Nesta fase dos testes, a PLGAC e a PLGRC foram comparadas considerando quer padrões em 3 fases, quer padrões em n fases.

Na segunda fase de testes, foram utilizados 3 conjuntos de problemas apresentados por Farley [5] como sendo representativos de três aplicações marcadamente diferentes, por ele encontradas. Na tabela 2 encontram-se as características destes problemas.

Os problemas do conjunto 1 caracterizam-se por um número de pedidos por peça muito elevado e um número de peças diferentes também elevado. Por sua vez, no conjunto de 2 os problemas possuem poucas peças diferentes com um número de pedidos por peça médio. Finalmente, os problemas do conjunto 3 caracterizam-se por terem um elevado número de peças diferentes, mas com um número reduzido de pedidos por peça. Este terceiro conjunto de problemas distingue-se ainda por as peças possuirem, em média, uma área superior à dos problemas dos conjuntos 1 e 2.

Para cada conjunto foram gerados e corridos 20 problemas, tendo os valores concretos de w_i e h_i sido gerados a partir da expressão:

$$\min + (\max - \min + 1) * \sqrt{R}$$

onde R é uma variável aleatória definida sobre o intervalo [0,1[. Os valores de m e bi foram amostrados de uma distribuição uniforme, sobre os limites apresentados.

Dadas as dimensões das chapas de stock, é completamente impraticável trabalhar com padrões em n fases, dado o tempo elevado necessário à resolução de cada problema mochila que viesse a surgir. Por isso, a comparação da PLGAC com a PLGRC foi feita considerando apenas padrões em 3 fases.

Tabela 2 – Características dos problemas da segunda fase de testes

Chapa de stock			Nº de peças diferentes (m)		Comprimento das peças (w_i)		Largura das peças (h_i)		Pedidos por peça (b_i)	
Conjunto	Comp. (w)	Larg. (h)	min	max	min	max	min	max	min	max
1	200	400	20	30	20	100	20	100	100	300
2	200	400	4	10	20	100	20	100	20	100
3	200	400	20	30	50	150	50	150	3	20

Resultados computacionais

Na tabela 3 apresentam-se os resultados obtidos nos 10 problemas da primeira fase de testes, quando se consideraram (e geraram) padrões em n fases. Por uma questão de clareza, apenas são aqui apresentados os valores médios dos resultados obtidos.

Tabela 3 – Resultados da primeira fase de testes - padrõe em n fases

Técnica	Nº de chapas de stock	Nº de iterações	Nº de padrões gerados		Tempo de CPU (s)	
			GG	Heurística	Padr.iniciais	PL
PLGAC	37.7	25.8	22.5	–	59.03"	236.18"
PLGRC	37.7	33.2	12.8	9.7	59.06"	130.74"

Deste primeiro conjunto de resultados ressalta uma nítida superioridade da PLGRC. Considerando o quociente entre os tempos médios de execução das duas técnicas, a PLGRC revelou-se cerca de 1.7 vezes mais rápida. É ainda interessante notar que, um pouco contrariamente ao que seria de esperar, a diferença entre o número de iterações, entre as duas técnicas, não é elevada, existindo mesmo alguns casos em que estes valores coincidem. Este facto vem de algum modo atestar a qualidade da heurística utilizada, pois esta coincidência significa que frequentemente a heurística gerou de facto o padrão óptimo (aquele que maximiza a função objectivo do problema auxiliar).

Este mesmo conjunto de problemas foi novamente corrido, mas agora considerando padrões em 3 fases. Os valores médios dos resultados encontram-se na tabela 4.

Tabela 4 – Resultados da primeira fase de testes - padrõe em 3 fases

Técnica	Nº de chapas de stock	Nº de iterações	Nº de padrões gerados		Tempo de CPU (s)	
			GG	Heurística	Padr.iniciais	PL
PLGAC	39.2	17.0	15.2	–	0.85"	3.16"
PLGRC	39.2	23.1	8.2	8.8	0.87"	5.67"

Utilizando padrões em 3 fases os resultados invertem-se completamente: a PLGAC é francamente mais rápida que a PLGRC. O quociente entre os tempos médios de execução das duas técnicas é de 0.58. Esta diferença, face aos resultados referentes aos padrões em n fases, deve-se em primeiro lugar ao facto do algoritmo de pesquisa lexicográfica para geração de padrões em 3 fases (de Gilmore e Gomory) ser muito mais eficiente que o algoritmo de Programação Dinâmica para geração de padrões em n fases (dos mesmo autores). Por outro lado, a eficiência do algoritmo de pesquisa lexicográfica depende sobretudo do número de peças diferentes, m, em jogo, uma vez que, por cada padrão gerado, são resolvidos m+1 subproblemas a uma dimensão. Ora nestes problemas da primeira fase de testes, m é relativamente pequeno. Finalmente, há que considerar que a PLGRC corre em todas as iterações a heurística, aproveite ou não o seu resultado. Quando os tempos de execução do algoritmo de pesquisa lexicográfica e da heurística são da mesma ordem de grandeza, cada iteração em que a heurística não seja bem sucedida, com consequente execução do algoritmo de pesquisa lexicográfica, degrada consideravelmente o desempenho da técnica. Para finalizar, deve-se considerar que se é verdade que em termos relativos as posições se invertem, em termos absolutos a PLGRC tem ainda uma grande vantagem, isto é, se neste bloco de problemas perdeu alguns segundos, no bloco anterior ganhou algumas centenas de segundos. Será ainda interessante verificar como se comportará o algoritmo de pesquisa lexicográfica quando m aumentar.

Os resultados da segunda fase de testes encontram-se na tabela 5. Mais uma vez apenas se apresentam os valores médios.

Tabela 5 – Resultados da segunda fase de testes - padrõe em 3 fases

Conjunto de problemas	Técnica	Nº de chapas de stock	Nº de iterações	Nº de padrões gerados		Tempo de CPU (s)	
				GG	Heurística	Padr.iniciais	PL
1	PLGAC	359.15	92.30	77.40	–	.43.83"	450.00"
	PLGRC	359.25	123.10	40.35	19.00	43.67"	238.17"
2	PLGAC	33.80	18.55	16.35	–	0.55"	3.75"
	PLGRC	33.80	21.00	11.40	4.40	0.54"	6.86"
3	PLGAC	65.50	88.25	72.80	–	29.45"	356.02"
	PLGRC	65.65	107.70	26.95	26.15	29.53"	139.80"

No primeiro e terceiro conjuntos de problemas, a PLGRC mostrou-se novamente superior à PLGAC, sendo em média 1.9 e 2.5 vezes mais rápida, respectivamente. Tal como se previa, com o aumento do número de peças diferentes (m), a eficiência do algoritmo de pesquisa lexicográfica decaiu bastante, fazendo com que a PLGRC suplantasse a PLGAC. No conjunto de problemas 2 (m pequeno) a PLGAC foi novamente a mais rápida. Contudo, tal como na

primeira fase de testes, em termos globais a PLGRC mostrou-se muito superior, pois somando os tempos de execução para todos os conjuntos de problemas, obtem-se 809.77 para a PLGAC e 384.83 para a PLGRC, isto é, globalmente a PLGRC foi 2.1 vezes mais rápida.

Para além dos tempos de execução, indicador de desempenho mais importante e em função do qual a PLGRC foi desenvolvida, será curioso analisar o número de padrões gerados pelo algoritmo de Gilmore e Gomory e pela heurística. Em primeiro lugar, o número total de padrões gerados pela PLGRC é em média inferior à PLGAC. Isto poderá mais uma vez indicar que a heurística resulta em padrões próximos da solução óptima do problema auxiliar, ou mesmo nos padrões óptimos. Poder-se-ia assim construir uma heurística para o problema global de cortes, baseada apenas num algoritmo de PL com geração de colunas e na heurística, para gerar esses mesmos padrões. Esta heurística global poderia ser particularmente útil quando se pretendesse trabalhar com padrões em n fases, ou para problemas com padrões em 3 fases extraordinariamente grandes.

Como conclusão, pode-se afirmar que a PLGRC é uma abordagem nitidamente mais eficiente que a PLGAC, e tanto mais eficiente quanto mais complexos forem os problemas, isto é, se se utilizarem padrões em n fases, se o tamanho das chapas de stock for grande, ou se o número de peças diferentes for elevado. Estes são sem dúvida casos mais importantes. Nos problemas pequenos, dada a ordem de grandeza dos tempos envolvidos, não se justifica a sua não utilização em detrimento da PLGAC. A PLGRC é pois uma abordagem alternativa à PLGAC, cerca de 2 vezes mais rápida que esta.

5. Conclusões

Neste artigo apresentamos uma nova abordagem à técnica clássica de Gilmore e Gomory para a resolução de problemas de cortes multi-padrão: a programação linear com geração ráida de colunas.

Nesta abordagem, ao procurar-se uma coluna para entrar na base, não se minimiza os custos marginais (através da resolução de um problema mochila), mas gera-se, usando uma heurística, uma coluna com custo marginal negativo. Este procedimento aposta na diminuição do tempo médio por iteração, embora aumentando o número de iterações. A eficiência da abordagem dependeria do produto *nº de iterações* × *tempo por iteração.*

Nos testes computacionais efectuados, esta abordagem revelou-se em média cerca de 2 vezes mais rápida que a técnica clássica da geração atrasada de colunas. Mais importante ainda é o facto de a geração rápida de colunas se ter revelado (comparativamente com a geração atrasada de colunas) tanto mais eficiente quanto mais complexos são os problemas (padrões em n fases, chapas de stock grandes, ou um número elevado de pedidos diferentes).

A geração rápida de colunas constitui pois uma abordagem alternativa à geração atrasada de colunas, mais eficiente que esta, e que viabiliza a aplicação da técnica clássica de Gilmore e Gomory a problemas mais complexos.

Referências

[1] Beasley, J.E., Algorithms for Unconstrained Two-Dimensional Guillotine Cutting, Journal of the Operations Research Society 36 (1985) 297-306.
[2] Dyckhoff, H., Kruse, H., Abel, D. and Gal, T., Classification of Real World Trim Loss Problems, in: G.Fandel et al (eds), "Essays on Production Theory and Planning", Springer-Verlag, 1988, 191-208.
[3] Dyckhoff, H., A Typology of Cutting and Packing Problems, EJOR 44 (1990) 256-266.
[4] Dyson, R.G. and Gregory, A.S., The Cutting Stock Problem in the Flat Glass Industry, Operational Research Quarterly 25 (1974) 41-53.
[5] Farley, A.A., Practical Adaptations os the Gilmore-Gomory Approach to Cutting Stock Problems, OR Spektrum 10 (1988) 113-123.
[6] Gilmore, P.C. and Gomory, R.E., A Linear Programming Approach to the Cutting Stock Problem, Operations Research 9 (1961) 849-859.
[7] Gilmore, P.C. and Gomory, R.E., A Linear Programming Approach to the Cutting Stock Problem- Part II, Operatioms Research 11 (1963) 863-888.
[8] Gilmore, P.C. and Gomory, R.E., Multistage Cutting Stock Problems of Two and More Dimensions, Oerations Research 13 (1965) 94-120.
[9] Gilmore, P.C. and Gomory, R.E., The Theory and Computation of Knapsack Functions, Operations Research 15 (1967) 1045-1075.
[10] Herz, J.C., Recursive Computational Procedure for Two-Dimensional Stock Cutting, IBM Journal of Research and Development 16 (1972) 462-469.
[11] Hinxman, A.I., The Trim-Loss and Assortment Problems: a Survey, European Journal of Operational Research 5 (1980) 8-18.
[12] Israni, S.S. and Sanders, J.L., Two-Dimensional Cutting Stock Problem Research: A Review and a New Rectangular Layout Algorithm, Journal of Manufacturing Systems 1 (1982) 169-182.
[13] Oliveira, J.F., Optimização em Cortes Rectangulares, Tese de Mestrado - DEEC/FEUP, 1990.
[14] Oliveira, J.F., An Improved Version of Wang's Algorithm for Two-Dimensional Cutting Problems, EJOR 44 (1990) 256-266.

INVESTIGAÇÃO OPERACIONAL, vol. 13, n.° 2 (Dezembro 1993) 119-134

ON THE SOLUTION OF A SPATIAL EQUILIBRIUM PROBLEM

L.M.Fernandes
Escola Superior de Tecnologia de Tomar
2300 Tomar, Portugal

M.A.Coelho
EDP - Electricidade de Portugal
1000 Lisboa, Portugal

J.J.Júdice, J.M.Patrício
Departamento de Matemática
Universidade de Coimbra
3000 Coimbra, Portugal

Abstract

In this paper we discuss the use of the predictor-corrector algorithm for the solution of a monotone linear complementarity problem (LCP) that arises from a spatial equilibrium model. The special structure of the matrix of the LCP is exploited to design an efficient implementation of the predictor-corrector algorithm that incorporates a preconditioned conjugate-gradient method. Computational experience is included showing the superiority of the predictor-corrector algorithm over other alternative techniques.

Resumo

Neste artigo é discutida a aplicação do método preditor-corretor para a resolução de um problema linear complementar (LCP) associado com um modelo de equilíbrio económico espacial. É desenvolvida uma implementação do algoritmo que tira partido da estrutura especial da matriz do LCP e incorpora um método de gradientes conjugados com precondicionamento. Experiência computacional mostra a superioridade do algoritmo preditor-corretor sobre outras técnicas alternativas.

Keywords

Spatial equilibrium model, linear complementarity problem, large-scale optimization, sparse matrices.

1. Introduction

The Linear Complementarity Problem (LCP) consists of finding vectors $z \in R^n$ and $w \in R^n$ such that

$$w = q + Mz,\ z \geq 0,\ w \geq 0,\ z^T w = 0$$

for a given vector $q \in R^n$ and a square matrix $M \in R^{n \times n}$. The existence of a solution to the LCP depends on the class of the matrix M. A LCP is said to be monotone if the matrix M is positive semi-definite, that is, satisfies

$$z^T M z \geq 0, \text{ for all } z \in R^n$$

It is possible to show [1] that in this case the LCP has a solution provided the feasible set K defined by its linear constraints is nonempty. The monotone LCP has been studied intensively

during the last thirty years. A large number of direct and iterative algorithms has been proposed for its solution. Furthermore this type of LCP has found many applications in different areas of science, engineering and economics. We suggest [1,11] for excellent surveys of the most important algorithms and applications of the monotone LCP.

The solution of spatial equilibrium models has been one of the most important sources of the LCP [1]. In this paper we consider a model of this type that involves a single commodity. This model has been introduced by Glassey [3], which has proposed a special purpose algorithm for its solution. Later, Pang and Lee [13] found a mistake in that algorithm and introduced an efficient implementation of Graves principal pivoting algorithm [4] for the solution of the LCP that arises in that model. It is nowadays well accepted that single principal pivoting algorithms are quite slow for large-scale LCPs [6]. This has increased the interest on the design of other techniques capable to deal with the LCP when its dimension is large. The Projected SOR algorithm [9] has been proposed in this context [5]. This method can also be implemented in a very efficient way by exploiting the special structure of the LCP. In this paper we propose a simple implementation of the algorithm. We also investigate its performance and conclude that the algorithm is not robust for the solution of this type of LCP.

Interior-Point algorithms have become quite popular for the solution of the monotone LCP [7,14]. Mehrotra's predictor-corrector method [7,8,10] is assumed to be the most efficient technique of this type. In this paper we investigate the use of this model mentioned before. It is known that the major effort of the predictor-corrector algorithm relies on the solution of two systems of linear equations that are requested to find the search direction. For the LCP under study the matrix of these systems takes the form

$$B = E + PDP^T$$

where E and D are diagonal matrices with positive diagonal elements and P is the arc-node incidence matrix. The explicit computation of B leads into a dense matrix and prevents the use of a direct method for the solution of the systems. In this paper we propose the use of the preconditioned conjugate gradient method for performing this task.

An efficient preconditioning is introduced that takes full advantage of the struture of the matrix B. Computational experience shows that this implementation performs quite well in practice and is superior over other alternative techniques.

The organization of the paper is as follows. Section 2 contains a brief description of the model. The Projected SOR and the Predictor-Corrector algorithms are discussed in sections 3 and 4. Finally computational experience is presented in the last section of this paper.

2. The model

The single commodity model described in [3,13] is stated in terms of a network, where the nodes represent the regions and the arcs stand for the exchanges among regions. In order to state formally the model, we denote by a_i and b_i the unit price and the requirement of the commodity in region i respectively. We also denote by x_{ij} the flow from two given regions i

and j. Furthermore let c_{ij} be the cost from shipping an unit of the product from region i to region j. Finally the following triangle inequality

$$c_{ij} \leq c_{ik} + c_{kj} \tag{1}$$

is assumed to hold for any regions i, j and k. Hence no transhipment takes place among regions, which means that if $x_{ij} > 0$ then $x_{jk} = 0$ for all $k \neq i$. We also assume that the price p_i in each region i is a linear function of the demand of the form

$$p_i = a_i - b_i y_i \tag{2}$$

We note that the demand y_i in region i is given by

$$y_i = \sum_{j=1}^{m} x_{ji} - \sum_{j=1}^{m} x_{ij} \tag{3}$$

where m is the number of nodes of the network. The objective of the model is to find the price p_i of the commodity in each region i such that the following equilibrium conditions hold

$$p_i + c_{ij} - p_j \geq 0, \quad x_{ij} \geq 0, \quad x_{ij}(p_i + c_{ij} - p_j) = 0 \tag{4}$$

We suggest [3] for a detailed explanation of this model. By using the expressions (2) and (3) it is possible to write (4) as the following Linear Complementarity Problem (LCP) in the variables x_{ij}:

$$a_i - b_i \Big(\sum_{j=1}^{m} x_{ji} - \sum_{j=1}^{m} x_{ij}\Big) + c_{ij} - a_j - b_j \Big(\sum_{j=1}^{m} x_{ij} - \sum_{j=1}^{m} x_{ji}\Big) \geq 0$$

$$x_{ij}\left[a_i - b_i \Big(\sum_{j=1}^{m} x_{ji} - \sum_{j=1}^{m} x_{ij}\Big) + c_{ij} - a_j - b_j \Big(\sum_{j=1}^{m} x_{ij} - \sum_{j=1}^{m} x_{ji}\Big)\right] = 0 \tag{5}$$

After solving this LCP, we can immediately find the values of the prices p_i by using the formulas (2) and (3). If $n' = m(m-1)$, $c = [c_{ij}] \in R^{n'}$, $a = [a_i] \in R^m$, $B = \text{diag}(b_i) \in R^{m \times m}$ and $P \in R^{n' \times m}$ is the incidence matrix node-arc, then we can write (5) in the form

$$\begin{aligned} &w = q + Mz \\ &z \geq 0, \quad w \geq 0 \\ &z^T w = 0 \end{aligned} \tag{6}$$

where $q = Pa + c$ and $M = PBP^T$.

Since $\text{rank}(P) = m < n'$ and B is a diagonal matrix with positive diagonal elements, then M is a singular symmetric positive semi-definite matrix. Therefore we have to solve a monotone LCP with dimension $n' = m(m-1)$, where m is the number of nodes of the network. However, due to the triangle inequality, it is possible to reduce the dimension of the LCP. In fact, the following result holds:

Theorem 1 [13] If $a_i - a_j + c_{ij} > 0$, then $x_{ij} = 0$, that is, there is no trade from region i to region j.

As a consequence of this result, it is sufficient to solve a LCP of dimension |S|, where |S| is the number of elements of the set

$$S = \{k = (i,j) : \ a_i - a_j + c_{ij} \leq 0\} \tag{7}$$

Furthermore the vector q and the matrix M of this LCP are given by

$$q = P_S a + c_S$$
$$M = P_S B P_S^T$$

where $P_S(c_S)$ is the submatrix (subvector) of P (c) containing the rows (components) of P (c) corresponding to the arcs that belong to S.

It is easy to see that the explicit computation of M leads to a dense matrix. However, M is a product of three sparse matrices. This suggests the use of iterative methods. In the next sections we briefly describe two algorithms that can exploit the special structure of the LCP.

3. Projected SOR Algorithm

This algorithm [9] is an extension for the LCP of the well known SOR algorithm for systems of linear equations, that incorporates a projective step in order to mantain the nonnegativity of the variables z_i. The steps of the algorithm are presented below.

Algorithm 1 (Projected SOR Algorithm)

Step 0 - Set $z^0 = 0$, $\alpha \in]0,2[$ and $k = 0$.

Step 1 - For $i = 1,\ldots, n$

$$w_i^k = q_i + \sum_{j<i} m_{ij} z_j^{(k+1)} + \sum_{j \geq i} m_{ij} z_j^{(k)}$$

$$z_i^{(k+1)} = \max\left\{ 0 , \frac{z_i^{(k)} - \alpha w_i^{(k+1)}}{m_{ii}} \right\}$$

$k = k + 1$

Step 2 - For $i = 1,\ldots, n$

$$w_i = q_i + \sum_{j=1}^{n} m_{ij} z_j^k$$

If $w_i < 0$ or $(z_i^k > 0$ and $w_i^k > 0)$ then go to **Step 1**

z^k is the solution of the LCP

The convergence of the SOR algorithm depends on the class of the matrix M. If the LCP is monotone and M is symmetric, then a feasibility hypothesis is sufficient to ensure convergence. In fact, we can state the following result [9]:

Theorem 2 If M is a symmetric positive semi-definite matrix, then the projected SOR algorithm converges to the solution of the LCP, provided $\alpha \in]0,2[$ and one of the following conditions hold:

1. $Mz + q > 0$ for some $z \in R^n$.
2. $Mz > 0$ for some $z \in R^n$.

Since any positive definite matrix satisfies the condition 2. [1], this algorithm can be used to solve LCPs with this type of matrices. A major drawback of the projected SOR algorithm is its dependence upon the parameter α. The theory states that $\alpha \in]0,2[$ is sufficient to ensure convergence, but the algorithm usually performs better when α is greater than one.

Next we describe an efficient implementation for the LCP (6) associated with the model described in the previous section. The matrix M of this LCP has the following form

$$M = P_S B P_S^T$$

where S is the set given by (7). Hence the matrix P_S can be stored in a vector NET of dimension 2n = 2|S| given by

$$NET(2k - 1) = i \qquad NET(2k) = -j$$

where k = (i,j) is the arc joining the nodes i and j. Furthermore the vector w can be computed by

$$w_i = q_i + P_i B(P^T z) = q_i + P_i B\gamma, \qquad i = 1,\ldots, n$$

The vector γ is not computed from scratch, but instead is updated whenever a component of z is modified in Step 2 of the projected SOR algorithm. In fact, whenever z_i is modified into $\tilde{z}_i$ this vector γ changes into

$$\gamma + P^T(\tilde{z} - z) = \gamma + P^T(\tilde{z}_i - z_i)e_i$$

where $z = (z_1,\ldots, z_i,\ldots, z_n)$, $\tilde{z} = (z_1,\ldots, \tilde{z}_i,\ldots, z_n)$ and e_i is a vector with all components equal to zero but the i-th component that is equal to one. Since P has only two nonzero elements per column, only two multiplications are required to update γ. By incorporating this type of procedure, the projected SOR algorithm takes the following form.

Algorithm 1 (Projected SOR Algorithm)

Step 0 - Set $z = 0$, $\gamma = 0$, $\alpha \in]0,2[$ and $\varepsilon > 0$ a tolerance for zero.

Step 1 - For i = 1,..., n

$$w_0 = q_i + P_i B\gamma$$

$$z_0 = \max\left\{ 0 , \frac{z_i - \alpha w_0}{m_{ii}} \right\}$$

$$\gamma = \gamma + P^T(z_0 - z_i)e_i$$

$$z_i = z_0$$

Step 2 - For i = 1,..., n

$$w_0 = q_i + P_i B\gamma$$

If $(w_0 < -\varepsilon)$ or $(z_i > \varepsilon$ and $w_0 > \varepsilon)$ then go to **Step 1**

z is the solution of the LCP.

Next we discuss the computational effort of each iteration of the algorithm. Step 1 requires 8n multiplications and n comparisons, distributed as follows:

two multiplications to compute $P_i B$.

two multiplications to compute $(P_i B)\gamma$.

two multiplications to update γ.

two multiplications and a comparison to update z.

Furthermore the number of operations in Step 2 is always smaller than or equal to 4n. As far as the storage is concerned, it is required to store the vectors NET, z, γ and q and the diagonal of B.

The implementation presented above is quite simple and requires a small amount of storage. Furthermore the computational effort of each iteration is quite reduced. So the algorithm seems to be recommended for this type of LCP. However, we show later that in general this algorithm requires many iterations to find a sufficiently accurate solution of the LCP. This has increased our interest in searching another alternative iterative techniques that may also exploit the special structure of the matrix M of the LCP without sharing the drawback of the projected SOR algorithm.

4. Predictor-Corrector Algorithm

This algorithm has become quite popular for the solution of linear and quadratic programs and linear complementarity problems [7,8,14]. Next, we present a brief description of this procedure. The LCP (6) can be restated in the following form

$$w - q - Mz = 0 \tag{8}$$

$$WZe = 0 \tag{9}$$

$$w \geq 0,\ z \geq 0$$

where $W = \mathrm{diag}(w_1,\ldots, w_n)$, $Z = \mathrm{diag}(z_1,\ldots, z_n)$ and $e = (1,\ldots, 1) \in R^n$. Therefore the LCP is equivalent to a system of nonlinear equations (8) - (9) on nonnegative variables z_i and w_i. Hence we can design a modified Newton algorithm for the solution of this system in which the iterates (z^k, w^k) are maintained positive in each iteration. To describe a general iteration of this Newton method, let $z^k > 0$ and $w^k > 0$ be given vectors. The Jacobian matrix associated to the system (8) - (9) is given by

$$\begin{bmatrix} W_k & Z_k \\ -M & I_n \end{bmatrix}$$

where $W_k = \mathrm{diag}(w_1^k,\ldots, w_n^k)$, $Z_k = \mathrm{diag}(z_1^k,\ldots, z_n^k)$ and I_n is the identity matrix of order n. Hence the Newton direction (u^k, v^k) can be found by

$$\begin{bmatrix} W_k & Z_k \\ -M & I_n \end{bmatrix} \begin{bmatrix} u^k \\ v^k \end{bmatrix} = \begin{bmatrix} -W_k Z_k e \\ Mz^k + q - w^k \end{bmatrix} \tag{10}$$

This is to turn equivalent to

$$\begin{cases} (M + Z_k^{-1} W_k) u^k = -q - Mz^k \\ v^k = -W_k e - Z_k^{-1} W_k u^k \end{cases} \tag{11}$$

After finding the Newton direction, a new point (z^{k+1}, w^{k+1}) can be found by setting

$$w^{k+1} = w^k + \theta v^k, \qquad z^{k+1} = z^k + \theta u^k \tag{12}$$

where θ is a stepsize that is required to mantain the variables z_i and w_i positive. A simple calculation leads to the following expression for θ:

$$\theta = \beta \min \left\{ \min \left\{ -\frac{z_i^k}{u_i^k} \ : \ u_i^k < 0 \right\}, \min \left\{ -\frac{w_i^k}{v_i^k} \ : \ v_i^k < 0 \right\} \right\} \qquad (13)$$

where β is fixed value such that $0 < \beta < 1$ (in practice β is usually set equal to 0.99995).

This modified Newton algorithm has been studied in [7], where it has been shown that it can process a monotone LCP under reasonable conditions. The predictor-corrector algorithm has been introduced by Mehrotra [10] and is considered to be more efficient than Newton's method. The predictor-corrector algorithm contains a further step in which the Newton direction is corrected in such a way that the points (z^k, w^k) follow the so-called Central Path, which is the set of points that are solutions of the system

$$ZWe = \mu e \qquad (14)$$
$$w - q - Mz = 0$$

with μ a positive parameter. In each iteration of the Predictor-Corrector algorithm we first obtain a predictor direction (u^k, v^k) as in Newton's method. Then we compute the parameter μ by

$$\mu = \frac{(z^k + \theta u^k)^T (w^k + \theta v^k)}{n^2} \qquad (15)$$

where θ is given by (13). Finally we find the predictor-corrector direction $(\bar{u}^k, \bar{v}^k)$ by solving the following system of linear equations

$$\begin{cases} (M + Z_k^{-1} W_k)\bar{u}^k = -q - Mz^k + \mu Z_k^{-1} e - Z_k^{-1}\Delta \\ \bar{v}^k = \mu Z_k^{-1} e - W_k e - Z_k^{-1} W_k \bar{u}^k - Z_k^{-1}\Delta \end{cases}$$

where

$$\Delta = (u_1^k v_1^k, \ldots, u_n^k v_n^k)$$

Hence the new point satisfies

$$w^{k+1} = w^k + \theta \bar{v}^k, \qquad z^{k+1} = z^k + \theta \bar{u}^k$$

As before, θ is a stepsize that maintains all the variables positive, that is,

$$\theta = \beta \min \left\{ \min \left\{ -\frac{z_i^k}{\bar{u}_i^k} \ : \ \bar{u}_i^k < 0 \right\}, \min \left\{ -\frac{w_i^k}{\bar{v}_i^k} \ : \ \bar{v}_i^k < 0 \right\} \right\}$$

where $0 < \beta < 1$ (usually $\beta = 0.99995$).

It is possible to show that if the initial point is feasible, that is, satisfies the linear constraints of the LCP, then the predictor-corrector algorithm is able to find a solution of this problem in a polynomial number of iterations [7]. In practice, finding a first feasible vector is not an easy task, since it reduces to the solution of a linear program. So, the algorithm usually starts with a pair of vectors (z^0, w^0) that have simply positive components. The choice of this vector plays an important role on the number of iterations that is required to find a solution of the LCP and depends on the problem under study. In the last section we discuss the choice of the initial vectors z^0 and w^0 for the LCP that arises from the spatial equilibrium model that is studied in this paper.

The stopping criterion is another important issue of the predictor-corrector algorithm. Since the algorithm seeks a solution of the nonlinear system (8)-(9), then we should terminate the algorithm when a pair of vectors (z^k, w^k) is found satisfying

$$\| w^k - q - Mz^k \| < TOL1 \quad \text{and} \quad (w^k)^T z^k < TOL2 \tag{16}$$

The values of these two tolerances depend on the problem under study and of the algorithm that is chosen for the solution of the linear systems that are required in each iteration of the algorithm. In the last section of this paper we discuss the values of these tolerances for the solution of the LCP that arises from the model described in section 2.

It follows from the description of the predictor-corrector algorithm that each iteration requires the solution of two systems of linear equations with the matrix

$$M + E$$

where E is a diagonal matrix with positive diagonal elements. Since M is symmetric positive semi-definite then M+E is nonsingular [1] and the systems have a unique solution. As stated before, an iterative method should be used to solve the systems with the matrix M+E. The preconditioned conjugate-gradient algorithm is the most recommended algorithm to perform these taks [12]. Next, we present the steps of the algorithm for the solution of a system of the form

$$(M+E)\, x = d$$

Preconditioned Conjugate Gradient (PCG) Algorithm

Given x^0 and a tolerance $\varepsilon > 0$

$r^0 = d - (M+E)x^0$

Solve $F\tilde{r}^0 = r^0$

$p^0 = \tilde{r}^0$

$k = 0$

For $k = 0, \ldots, n$ do

$v^k = (M+E)p^k$

$\gamma_k = (\tilde{r}^k)^T r^k$

$\alpha_k = -\dfrac{\gamma_k}{(p^k)^T v^k}$

$x^{k+1} = x^k - \alpha_k p^k$

$r^{k+1} = r^k + \alpha_k v^k$

If $(r^{k+1})^T r^{k+1} \leq \varepsilon$ then stop.

Solve $F\tilde{r}^{k+1} = r^{k+1}$

$\beta_k = \dfrac{(\tilde{r}^{k+1})^T r^{k+1}}{\gamma_k}$

$p^{k+1} = p^k + \beta_k p^k$

As far as the storage is concerned, we have to store the vectors r, v, p, x and $\tilde{r}$ together with the vectors NET, q, and B mentioned before. Since E is a diagonal matrix the product $(M+E)p^k$ can be done in a way similar to the projected SOR method.

The performance of this algorithm depends on the preconditioning matrix F. Next, we discuss two possible choices of F.

(i) Diagonal preconditioning

In this case F is chosen to be the diagonal of M+E. It easily follows from the definition of M that its diagonal elements are given by

$$m_{kk} = b_i + b_j$$

where k = (i,j) is an arc belonging to the set S defined by (7). Hence

$$F = \text{diag}(m_{11} + e_{11},\ldots, m_{nn} + e_{nn}) \qquad (17)$$

where e_{ii} are the diagonal elements of E. In this case the solution of the system $F\tilde{r}^{k+1} = r^{k+1}$ is quite straightforward and only requires n operations.

(ii) Block diagonal preconditioning

If we write

$$M = \begin{bmatrix} P_1 \\ P_2 \\ \cdot \\ \cdot \\ P_m \end{bmatrix} B \begin{bmatrix} P_1^T & P_2^T & \ldots & P_m^T \end{bmatrix}$$

where P_i is the submatrix of P_S corresponding to the region i and $E = \text{diag}[E_1,\ldots, E_m]$, then

$$M+E = \begin{bmatrix} P_1BP_1^T+E_1 & P_1BP_2^T & \ldots & P_1BP_m^T \\ P_2BP_1^T & P_2BP_2^T+E_2 & \ldots & P_2BP_m^T \\ \vdots & \vdots & \ldots & \vdots \\ P_mBP_1^T & P_mBP_2^T & \ldots & P_mBP_m^T+E_m \end{bmatrix}$$

The block diagonal preconditioning is the matrix

$$F = \begin{bmatrix} P_1BP_1^T+E_1 & & & \\ & P_2BP_2^T+E_2 & & \\ & & \cdot & \\ & & & P_mBP_m^T+E_m \end{bmatrix}$$

It is important to note that the solution of the system

$$F\tilde{r} = r \qquad (18)$$

reduces to the solution of m systems

$$F^i\tilde{r}^i = r^i, \qquad i = 1,\ldots, m \qquad (19)$$

where $r = (r^1,\ldots, r^m)$ and $\tilde{r} = (\tilde{r}^1,\ldots, \tilde{r}^m)$. If F^i is a square matrix of order m_i and $e = (1,\ldots, 1) \in R^{m_i}$, then

$$F^i = P_iBP_i^T + E_i = G + b_iee^T$$

where G is a diagonal matrix with positive diagonal elements g_i. Therefore

$$\tilde{r}^i = (G + b_iee^T)^{-1}r^i$$

Using the Sherman-Morrison Formula [2] we have

$$\tilde{r}^i = G^{-1}r^i - \frac{b_i(e^TG^{-1}r^i)}{1+b_i(e^TG^{-1}e)}\, G^{-1}e$$

Hence the solution $\bar{r}^i$ of the system (19) is given by

$$\bar{r}^i_j = \frac{r^i_j}{g_j} - \frac{b_i\left(\frac{r^i_1}{g_1} + \ldots + \frac{r^i_{m_i}}{g_{m_i}}\right)}{1+b_i\left(\frac{1}{g_1} + \ldots + \frac{1}{g_{m_i}}\right)} \cdot \frac{1}{g_j} \quad j = 1,\ldots, m_i$$

Therefore each system (19) is quite simple to solve and the same occurs with the system (18).

In the next section we describe some experiments with the predictor-corrector algorithm. The preconditioned conjugate gradient algorithm has been implemented to solve the systems required by the predictor-corrector method. The performance of the two different preconditionings is also studied.

5. Computational Experience

In this section we present some computational experience with the projected SOR and predictor-corrector algorithms for solving the monotone LCP associated to the single commodity spatial economic equillibrium model. The test problems have been generated according to the guidelines suggested in [13]. Hence we have constructed the vectors a and b and the matrix $C = [c_{ij}]$ of the transportation costs in the following manner:

- a_i and b_j are random real numbers lying in the intervals]0,50[and]0,20[respectively, $i = 1,\ldots, n$.
- $c_{ij} = d_i + d_j$, where the vector $d = [d_i]$ is defined by

$$d_i = \begin{cases} 10 \text{ if } i \text{ is even} \\ \text{a random number in }]0,20[\text{ if } i \text{ is odd} \end{cases}$$

Since $d_i > 0$, $i = 1,\ldots, m$, then the costs c_{ij} satisfy the triangular inequality. We have considered a set of test problems with different number of regions. As mentioned before, the dimension of the LCP equals the number of elements of the set S given by (7). In Table 1 we present for each test problem the number of regions and the dimension of the LCP to be solved.

Reg	30	60	75	90	105	120	135	150	165	180	195	210
n	215	776	1319	1486	2686	3141	3230	4985	5450	5947	7565	8776

Table 1 – Number of regions and dimensions of the LCPs

The computational experiments have been carried out in a Hewlett-Packard 720 Workstation with 16 Mb RAM and a 50 Mhz PA-RISC processor. All the codes have been written in FORTRAN 77 and the execution times are in CPU seconds.

It is quite well established that the predictor-corrector algorithm depends on the starting point (z^0, w^0). Our experience has shown that the values of the variables z^0_i should be much smaller than those of the variables w^0_i. The choice $z^0_i = 1.0$ and $w^0_i = 50.0$ for all $i = 1,\ldots, n$ has proven to perform well for all the test problems.

As stated before, the linear systems required by the predictor-corrector algorithm are solved by a preconditioned conjugate gradient algorithm. This latter procedure requires a value for the tolerance ε. Our experience has shown that the linear systems do not have to be solved with

great accuracy in the first iterations of the predictor-corrector. As suggested by some authors for other optimization problems, the tolerance ε is not the same for all the iterations, but instead is a decreasing function of the iteration count. Hence we propose the following choice for the tolerance ε in the iteration k of the predictor-corrector algorithm:

$$\varepsilon = \begin{cases} 10^{-3} \text{ if } g^p \sqrt{\varepsilon_M} > 10^{-3} \\ \max \left\{ g^p \sqrt{\varepsilon_M} , \sqrt{\varepsilon_M} \right\}, \text{ otherwise} \end{cases}$$

where ε_M is the machine epsilon ($\varepsilon_M = 10^{-16}$) [2], $g = (z^k)^T w^k$ is the current gap in this iteration and p is a real number greater than or equal to 1.0 (p = 2.5 is normally a good choice).

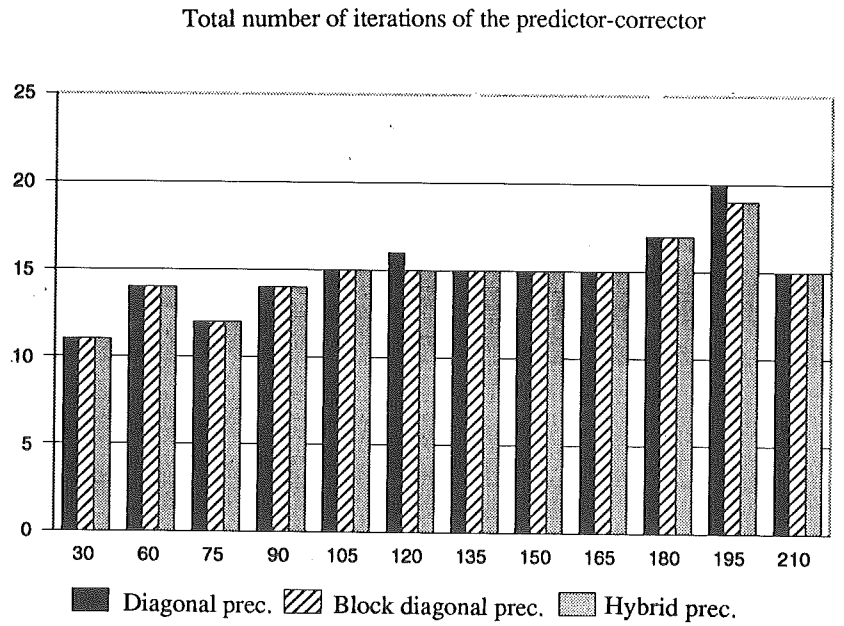
As a consequence of this stopping criterion for the PCG algorithm, the solution of the linear systems are not accurate in the first iterations but the residual tends to reduce as the algorithm progresses. The initial point x^0 for the PCG algorithm is given by $F^{-1}d$, where d is the right hand side vector of the systems and F is the preconditioning matrix.

In the first experience we have tested the efficiency of the two preconditionings described in the previous section. In Figure 1 we present the iteration count and the CPU time in seconds for the predictor-corrector method when the PCG algorithm uses the diagonal and the block diagonal preconditionings and a hybrid technique that is defined later. Figure 2 contains the information concerning the total number of iterations performed by the PCG algorithm using the three preconditionings, as well as the residual $\| \bar{w} - q - M\bar{z} \|_\infty$ of the solution $(\bar{z}, \bar{w})$, obtained by the three alternative techniques. These results show that the diagonal preconditioning is the least expensive in terms of the CPU time, but the block diagonal technique gives more accurate solutions for the predictor-corrector algorithm.

If we look deeper at the iteration counts of the PCG algorithm incorporating these preconditionings, we notice that the block diagonal technique yields more iterations than the diagonal procedure, but only in the first iterations of the predictor-corrector algorithm. The situation reverses as the iteration count increases. This can be seen in Figure 3, which contains the progress of the application of the predictor-corrector algorithm for the test problems with 195 and 210 regions respectively. This conclusion has lead us to suggest a hybrid technique, in which the diagonal preconditioning is used in the first two iterations of the predictor-corrector and the block diagonal preconditioning from then on. The results presented in Figure 4 and 5 show that this hybrid technique is usually more robust than the other two and should be recommended in general.

Figure 4 contains a comparison between the predictor-corrector algorithm and the projected SOR algorithm for all the test problems. As suggested in [5], we have used $\varepsilon = 10^{-3}$ in the stopping criterion of this latter algorithm. This means that the residual will be around 10^{-4} or 10^{-5}. In these tests we have set the relaxation parameter α equal to 1.3, since this value has shown to be the most consistent in all the tests performed by us.

Fig. 1 - Comparison among diagonal, block diagonal and hybrid preconditionings



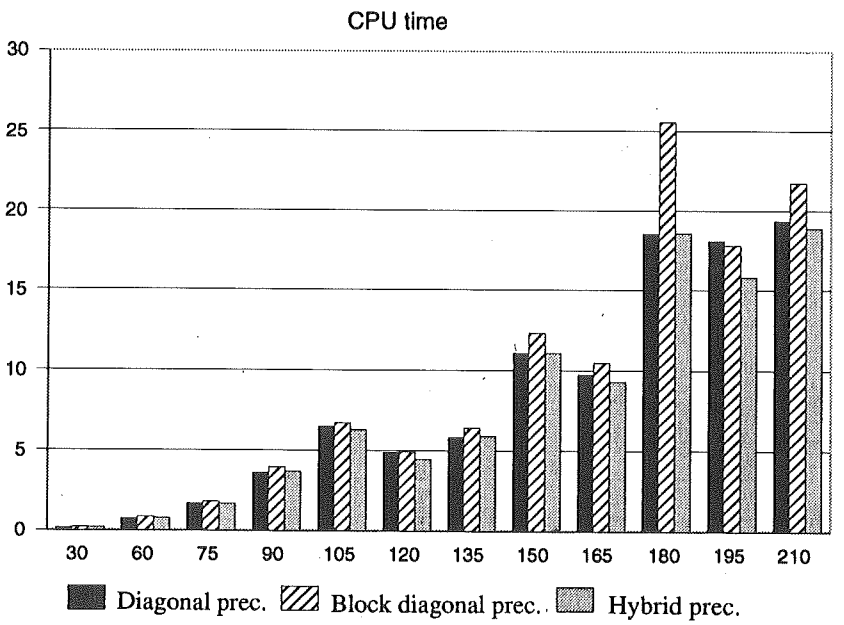
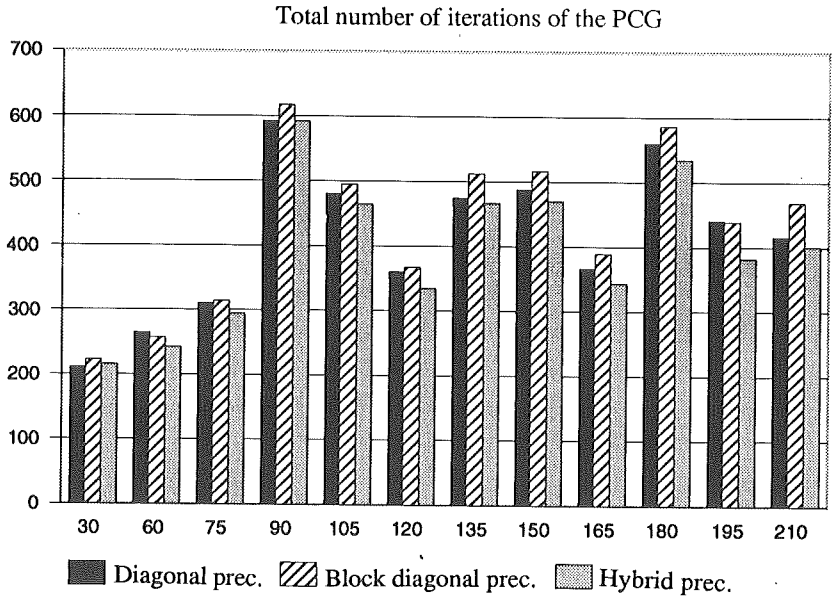


Fig. 2 - Comparison among diagonal, block diagonal and hybrid preconditionings



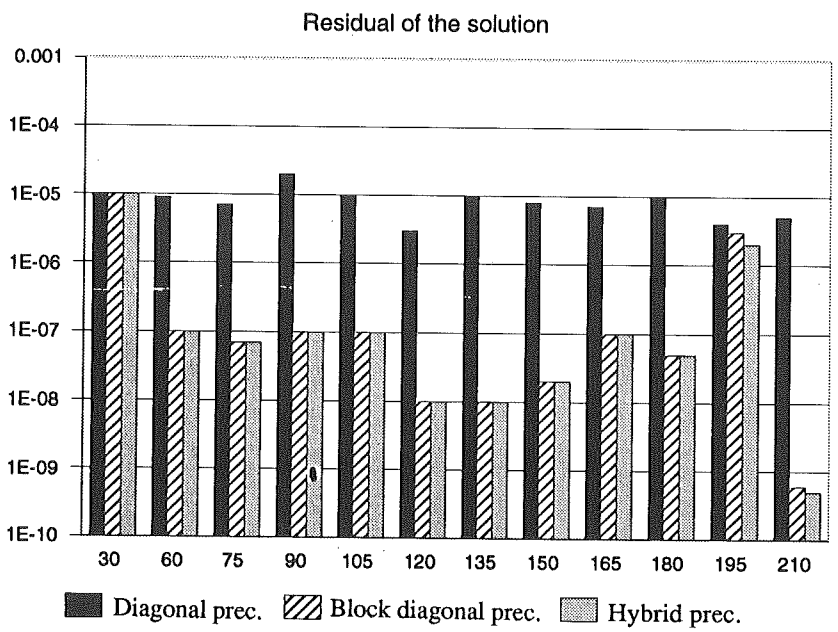


Fig. 3 - Number of PCG iterations in each iteration of the predictor-corrector algorithm

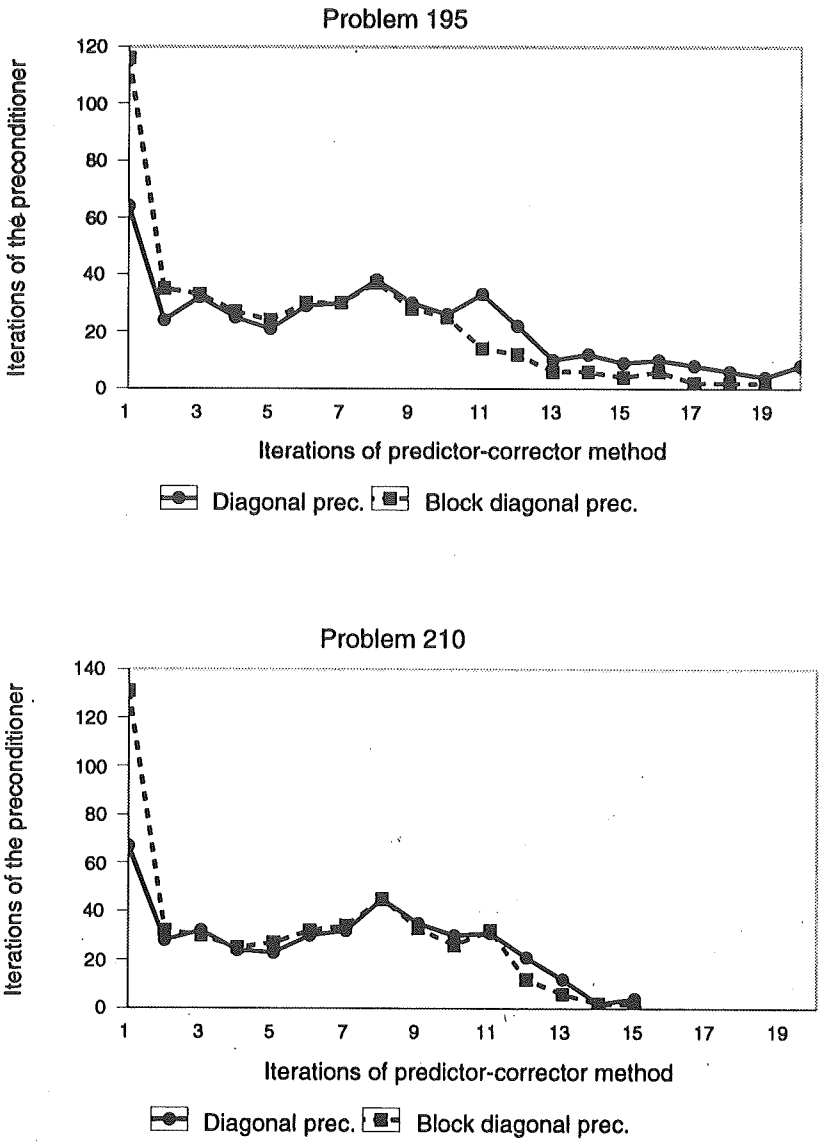
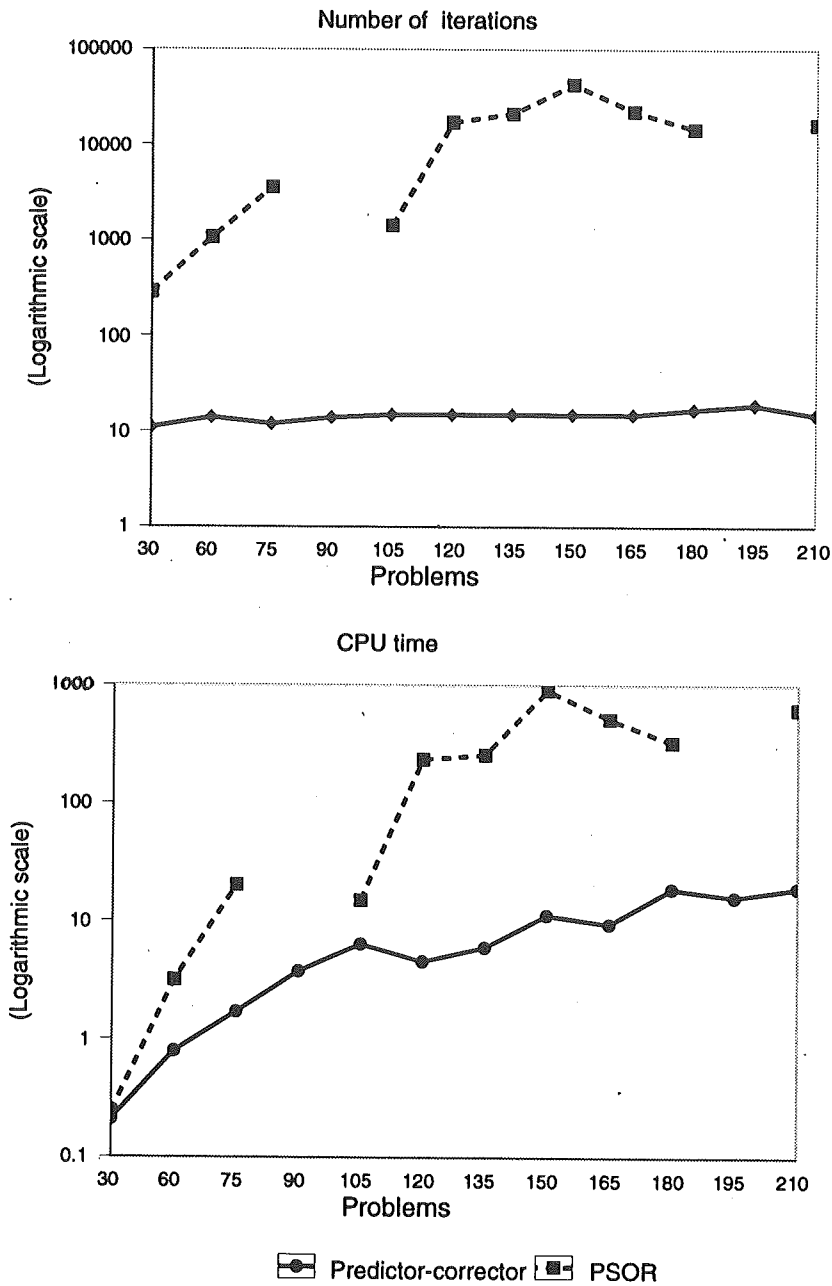


Fig. 4 - Comparison between predictor-corrector and PSOR



The results in Figure 4 show that the iteration count of the projected SOR algorithm is quite high. The method has even been unable to find a solution for two problems in less than 50000 iterations. Hence the projected SOR algorithm is not competitive with the predictor-corrector algorithm. Furthermore the results also confirm that the computational cost of each iteration of the first algorithm is quite reduced and much smaller than that of the predictor-corrector algorithm.

As a final conclusion of this paper, we can claim that the predictor-corrector algorithm incorporating a special preconditioned conjugate gradient method seems to be quite recommendable for solving a spatial equilibrium problem with a single commodity that has been discussed in [3,13]. We believe that this type of technique may also be an interesting approach for the solution of multicommodity spatial equilibrium models [5]. This is an important topic for future research.

References

[1] Cottle, R.W., Pang, J.S. and Stone, R.E., *The Linear Complementarity Problem*, Academic Press, New York, 1992.

[2] Dennis, J. and Schnabel, R., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, New Jersey, 1983.

[3] Glassey, C., *A quadratic network optimization model for equilibrium single commodity trade flow*, Mathematical Programming 14 (1978) 98-107.

[4] Graves, R., *A principal pivoting simplex algorithm for linear and quadratic programming*, Operations Research 15 (1967) 482-494.

[5] Güder, F., Morris, J. and Yoon, S., *Parallel and serial successive overrelaxation for multicommodity spatial price equilibrium problems*, Transportation Science 26 (1992) 48-58.

[6] Júdice, J. and Pires, F., *Direct methods for convex quadratic programs subject to box constraints*, Investigação Operacional 9 (1989) 23-56.

[7] Kojima, M., Megiddo, M., Noma, T. and Yoshise, A., *A unified approach to interior point algorithms for linear complementarity problems: a summary*, Operations Research Letters 10 (1991) 247-254.

[8] Lustig, I., Marsten, R. and Shanno, D., *Computational experience with a primal-dual point method for linear programming*, Linear Algebra and its Applications 152 (1991) 191-222.

[9] Mangasarian, O., *Solution of symmetric linear complementarity problems by iterative methods*, Journal of Optimization Theory and Applications 22 (1977) 465-485.

[10] Mehrotra, S., *On the implementation of a (primal-dual) interior point method*, Tecnical Report 03, Department Civil Engineering and Management Sciences, Northwestern University, Evanson IL, 1990.

[11] Murty, K., *Linear Complementarity, Linear and Nonlinear Programming*, Heldermann Verlag, Berlin, 1988.

[12] Ortega, J., *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[13] Pang, J. and Lee, P., *A parametric linear complementarity technique for the computation of equilibrium prices in a single commodity spatial model*, Mathematical Programming 20 (1981) 81-102.

[14] Pardalos, P., Ye, Y. and Han, C.G., *An interior point algorithm for large-scale quadratic problems with box constraints*, Lecture Notes in Control and Information 144 (1990) 413-422.

INVESTIGAÇÃO OPERACIONAL, vol. 13, n.° 2 (Dezembro 1993) 135-147

A COMPARISON OF SOME ALGORITHMS FOR THE MAXIMUM NETWORK FLOW PROBLEM

Pedro M. Vilarinho
Secção Autónoma de Gestão e Engenharia Industrial
Universidade de Aveiro
3800 Aveiro

Mario S. Rosa
Departamento de Matemática
Universidade de Coimbra
3000 Coimbra

Abstract

Since the original paper of Ford and Fulkerson algorithm (1956) many efficient algorithms have been developed for the maximum flow problem. We review some of these algorithms and compare their performance on a microcomputer. All the algorithms can do the job in a reasonable time, but the Goldberg-Tarjan method seems to be the most efficient in general.

Resumo

Desde 1956, após o artigo pioneiro de Ford e Fulkerson, têm sido desenvolvidos vários algoritmos para o problema do fluxo máximo numa rede. Neste artigo os autores fazem uma breve revisão de alguns deles e apresentam resultados computacionais importantes que comparam os seus desempenhos. De todos os algoritmos apresentados o do método de Goldberg-Tarjan parece, em geral, ser o mais eficiente.

Keywords

Maximum network flow problem, graphs, data structures.

1. Introduction

It is important to have efficient algorithms for solving the maximum network flow problem. They can be applied directly to the analysis of communication and transportation networks and can also solve many combinatorial optimization problems [5, 7, 13, 14, 20]. Moreover they are often used as procedures in algorithms for solving other network problems [8, 15, 17, 21 and 25].

We begin by giving some general terminology used throughout this paper, and then describe some algorithms to solve this problem. The algorithms described are the ones we consider basic in the sense that they are representative of the different approaches to the problem. However, other algorithms can be obtained from these by incorporating some complex data structures or scaling techniques [1, 9, 10, 11, 19, 22 and 23]. Finally, we compare the practical performance of these algorithms.

2. General terminology

Let D = (N, A) be a directed graph, where N is a finite set of |N| nodes and the elements of A are |A| ordered pairs of nodes called arcs. (|S| represents the cardinal of the set S).

A flow network, R = (s, t, N, A, b), is a directed graph (N, A) containing a source node $s \in N$ (there exists no arc of the form (x, s), $x \in N$), a terminal node $t \in N$ (there is no arc of the form (t, x), $x \in N$) and such that each arc $(x, y) \in A$ has associated a positive real number b(x, y), called its capacity.

A flow f(x, y) in R is a vector of $\Re^{|A|}$ such that:

(i) $0 \leq f(x, y) \leq b(x, y)$, for all $(x, y) \in A$

(ii) $\sum_{x:(x,y)\in A} f(x, y) = \sum_{x:(y,x)\in A} f(y, x), \forall y \in N - \{s, t\}$

The value of the flow is given by $|f| = \sum_{x:(s,x)\in A} f(s, x)$

We can also state the maximum flow problem:

Given a flow network, R = (s, t, N, A, b), the maximum flow problem is defined as the problem of finding a flow $f^* \in \mathbb{R}^{|A|}$ with maximum value, $|f^*|$, between the source node s, and the terminal node t.

We suppose, without loss of generality, that the directed graph associated with the flow network is asymmetrical, i.e., there is a unique arc $(x, y) \in A$ connecting any pair of nodes (x, y), and there are no arcs of the form $(x, x) \in A$.

The maximum flow problem was first presented and solved by Ford-Fulkerson [6] and independently by Elias, Fenstein and Shannon [4]. Both the authors proved that the maximum flow-minimal cut theorem which leads to the Ford Fulkerson algorithm for solving the maximum flow problem.

3. Ford Fulkerson algorithm [6]

We define a path $p = (n_1, n_2, \ldots, n_k)$ in D = (N, A) as a sequence of nodes of N such that $(n_j, n_{j+1}) \in A$, for j = 1,..., k-1, and no node is repeated.

Given a flow network R = (s, t, N, A, b) and a flow f from s to t, a flow augmenting path c, is a path between s and t in the undirected version of R, i.e., in the undirected graph obtained from R by ignoring the direction of its arcs, with the follwoing properties:

i) $f(x, y) < b(x, y)$, for all directed arcs $(x, y) \in A$, i.e., for all the arcs crossed by c in their own direction.

ii) $f(x, y) > 0$ for all reserved arcs $(x, y) \in A$, i.e., for all the arcs crossed by c in their opposite direction.

We can then conclude that the maximum possible flow increases along a flow augmenting path c, is given by

$$\delta = \min_{\text{arcs of c}} \begin{cases} b(x,y) - f(x,y), & \text{for direct arcs} \\ f(x,y), & \text{for reverse arcs} \end{cases}$$

Ford and Fulkerson proposed an algorithm for the maximum flow problem which starts with a feasible flow, e.g. f(x, y) = 0 for all (x, y)∈ A. Then in each iteration a flow augmenting path between s and t is found and the flow is increased along that path until it is no longer possible to find a flow augmenting path. In this case the flow is maximal.

4. Edmonds and Karp algorithm [3]

To describe this algorithm we need the following concept introduced by these authors. Given a network R = (s, t, N, A, b) and a feasible flow f in R, the incremental network is R(f) = (s, t, N, A(f), cr), where A(f) has the following arcs:

i) If (x, y)∈ A and f(x, y) < b(x, y), then (x, y)∈ A(f) and cr(x, y) = b(x, y) – f(x, y).

ii) If (x, y)∈ A and f(x, y) > 0 then (y, x)∈ A(f) and cr(y, x) = f(x, y).

For each (x, y)∈ A(f), cr(x, y) is called the residual capacity of (x, y)∈ A.

We have seen that the Ford-Fulkerson algorithm searches for a flow augmenting path using the network arcs in both directions. Notice that such a flow augmenting path corresponds to a path in the incremental network R(f). Therefore the maximum flow increase in such a path is equal to the minimum value of the capacities, cr(x, y) of the arcs in that path.

Edmonds and Karp proposed an algorithm that starts with a feasible flow. In each iteration finds the shortest path in R(f), i.e., the path with the least number of arcs, and then increases the flow along the path. The algorithm stops when it is no longer possible to find a path in R(f) for augmenting the flow f in R.

5. Dinic algorithm [2]

We start by introducing the following definitions.

An auxiliar network, $R_L(f) = (s, t, N_L, A_L(f), cr_L)$, is a layered network that contains all the shortest paths, that is all the paths with the same minimum number of arcs, between the source node s and the terminal node t, of the actual incremental network, R(f).

A blocking flow in $R_L(f)$ is a feasible flow such that all the paths between the source and the terminal node contain at least one saturated arc ($(x, y)\in A_L(f) : f(x, y) = cr_L(x, y)$). For ilustration see Figs. 1-3.

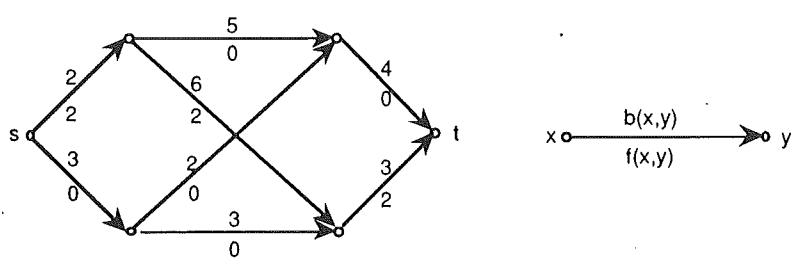


Figure 1
A network R with its arc capacities and a flow f of value 2

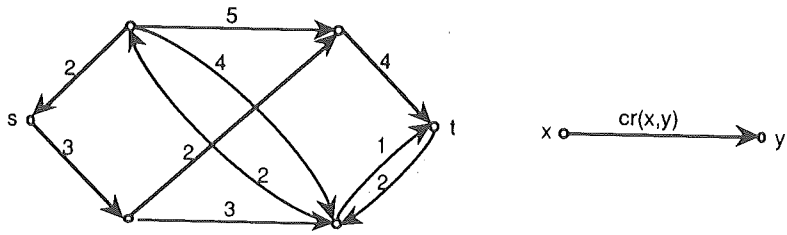


Figure 2
The incremental network R(f) corresponding
to the network of Figure 1 with flow f

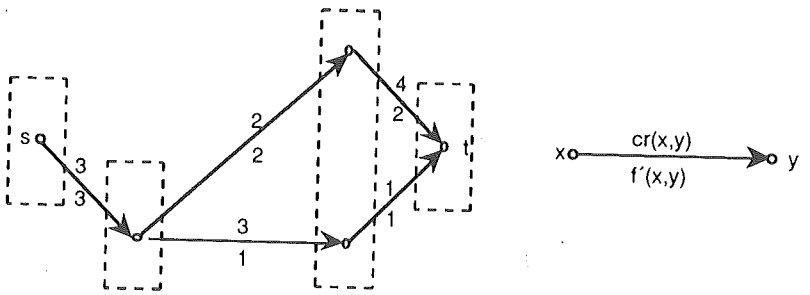


Figure 3
The layered network $R_L(f)$ extracted from
Figure 2 and showing a blocking flow f'

In each phase j of Dinic's algorithm we build the auxiliary network, $R_L(f_{j-1}) = (s, t, NL, A_L(f_{j-1}), cr_L)$, where f_{j-1} is the current flow. Then we search for a blocking flow f'_j, in $R_L(f_{j-1})$. Finally we update the flow value $f_j = f_{j-1} + f'_j$, where $f_j(x, y) = f_{j-1}(x, y) + f'_j(x, y)$ for all the direct arcs (x, y) in R(f) and $f_j(x, y) = f_{j-1}(x, y) - f'_j(x, y)$ for all the reverse arcs (x, y) in R(f).

The algorithm terminates when it is no longer possible to find a path between the source and the terminal nodes in the auxiliary network. Then the corresponding flow is a maximum flow.

Many algorithms that subsequently appeared in the literature use this idea of dividing the problem in phases and only address the question of finding a blocking flow in an auxiliary network.

Dinic proposed his own algorithm for finding a blocking flow in an auxiliary network. The algorithm starts with a null blocking flow, i.e. $f'(x, y) = 0$ for all $(x, y) \in A_L$. In each iteration we find a path c_{st}, between the source and the terminal node, by doing, for example, a depth first search of the graph associated with R_L. If such a path can be found in R_L, we set $\delta = \min\{cr_L(x, y) : (x, y) \in c_{st}\}$, i.e. δ is the minimum value of the residual capacities of the arcs in the path. The blocking flow vector is also updated, $f'(x, y) \leftarrow f'(x, y) + \delta$, as well as

the residual capacities of the arcs in c_{st}, $cr_L(x, y) \leftarrow cr_L(x, y) - \delta$, for all $(x, y) \in c_{st}$. Notice that at least one of the arcs in the path becomes saturated. If during the depth first search of the graph associated with R_L we go from a node x to a node y which has no immediate sucessors, then this path cannot lead to the terminal node. Therefore we go back to node x and delete the arc (x,y) from the auxiliary network, proceeding the search from node x.

The algorithm stops when $R_L(f)$ becomes s-t disconnected, as then it is no longer possible to find a path between the source node s, and the terminal node t, in R_L.

6. Karzanov/Tarjan algorithm [16, 24]

As we mentioned in the previous section, many of the algorithms differ from Dinic's method only in the way the blocking flow is obtained. This is the case of the algorithm proposed by Karzanov. Here we describe and test a version of this algorithm introduced by Tarjan.

We define a preflow, g(x, y), in a network R = (s, t, N, A, b) as a vector in $\Re^{|A|}$ such that

(i) $0 \leq g(x, y) \leq b(x, y)$, for all $(x, y) \in A$

(ii) $\sum_{y \in N} g(x, y) \leq \sum_{y \in N} g(y, x)$, for all $x \in N - \{s, t\}$

We denote by δx the excess of flow of node $x \in N$, that is,

$$\delta x = \sum_{y \in N} g(y, x) - \sum_{y \in N} g(x, y) \geq 0, \text{ for all } x \in N - \{s, t\}$$

A node x is said to be balanced if $\delta x = 0$ and unbalanced if $\delta x > 0$.

To obtain a blocking flow in the auxiliary network $R_L(f) = (s, t, N_L, A_L(f), cr_L)$, Tarjan algorithm finds a blocking preflow and gradually converts it into a blocking flow by balancing the nodes in successive forward and backward moves over the network. A preflow is said to be blocking if it saturates at least one arc in all the paths between the source and the terminal node in R_L.

Each node $x \in N$ may be blocked or unblocked. Initially the source node is blocked and all other nodes are unblocked. An unblocked node might become blocked but the contrary is not possible.

Let us see how we can balance a node that is in one of these states:

(i) Let $x \in N$ be an unblocked and unbalanced node, let L(x) be the set of arcs such that $L(x) = \{(x, y) : (x, y) \in A \text{ and } g(x, y) \leq cr_L(x, y)\}$ and let

$$\delta' = \min \left\{ \min_{(x,y) \in L(x)} \{cr_L(x,y) - g(x,y)\}, \delta x \right\}.$$

For each arc $(x, y) \in L(x)$, such that y is unblocked, we increase g(x, y) by δ'. If $\delta x = 0$ then x becomes balanced. If $\delta x > 0$ the balancing of x failed.

(ii) Let $y \in N$ be a blocked and unbalanced node, let M(y) be the set of arcs such that $M(y) = \{(x, y) : (x, y) \in A \text{ and } g(x, y) > 0\}$ and let

$$\delta'' = \min \left\{ \min_{(x,y) \in L(y)} \{cr_L(x,y) - g(x,y)\}, \delta y \right\}.$$

For each arc $(x, y) \in M(y)$ we decrease $g(x, y)$ by δ''. If $\delta y = 0$ then y becomes balanced.

Tarjan algorithm begins with the following preflow:

$g(s, y) = b(s, y)$, for all $y \in N$

$g(x, y) = 0$, for all $(x, y) \in A$ and $x \neq s$

In each iteration all the nodes in the auxiliary network are searched in a topological order(1) with the exception of the source and terminal nodes, balancing by (i) each unbalanced and unblocked node. If balancing of any node fails then it becomes blocked. After all nodes have been searched, we search them again, but this time in reverse topological order, balancing by (ii) each unbalanced and blocked node. The algorithm stops when all the nodes, except the source and terminal nodes, are balanced, that is when the preflow is a blocking flow.

7. Malhotra, Kumar and Maheshwari algorithm [18]

These authors have proposed another for finding a blocking flow on an auxiliary network, $R_L(f) = (s, t, N_L, A_L(f), cr_L)$.

To each node $x \in N_L$ we associate a value $p(x)$, called the node potential, defined by:

$$p(x) = \begin{cases} \min \left\{ \sum_{y:(x,y)\in A_L(f)} cr_L(x,y), \sum_{y:(y,x)\in A_L(f)} cr_L(y,x) \right\}, & \text{if } x \neq s,t \\ \sum_{y:(x,y)\in A_L(f)} cr_L(x,y), & \text{if } x = s \\ \sum_{y:(y,x)\in A_L(f)} cr_L(y,x), & \text{if } x = t \end{cases}$$

Notice that the potential of a node is the maximum value by which we can increase the flow in the arcs incident to it.

We will call a node, $r \in N_L$, the reference node and the potential associated to it the reference potential, if $p(r) = \min\{p(x) : x \in N_L\}$.

In each iteration of the algorithm we begin by finding the reference node and from it we search all the nodes in the network. We first follow a topological order until we reach the terminal node, and later we use a reverse topological order until we reach the source node.

(1) A topological order is an order such that, if $(x,y) \in A$, x is ordered before y. A reverse topological order is an order, such that if $(x, y) \in A$, y is ordered before x

Each time a node is searched according to a topological ordering, the flow that enters the node is distributed by the arcs that depart from it, saturating them one by one until all the flow entering the node has been distributed (supposing that p(r) is the flow entering the reference node). Each time a node is searched in reverse topological order, the algorithm behaves in a similar manner. Notice that if the potential of the reference node equals the flow entering (leaving) the reference node, all the arcs arriving (departing) at that node will become saturated.

After every node in the network has been searched, all the arcs that are saturated are deleted from the network. The same happens with all the arcs and nodes that are no longer on a path from the source to the terminal node, because the flow in those arcs can no longer be modified in subsequent iterations.

The algorithm stops when the source node is deleted, as it is no longer possible to find a path between the source and terminal nodes.

8. Goldberg and Tarjan algorithm [12]

Goldberg and Tarjan proposed an algorithm to find a maximum flow in a flow network which is not based on Dinic's idea of dividing the solution procedure in phases. Instead, they use the notion of a preflow introduced by Karzanov, but applied directly to the original network.

A node x is active if $x \in N - \{s, t\}$ and $\delta x > 0$, where δx is the excess flow in node x as defined in the Karzanov algorithm.

Let R = (s, t, N, A, b) be the flow network for which a maximum flow is to be found, g a preflow, and R(g) = (s, t, N, A(g), cr) the incremental network for the preflow g. Notice that although the flow is not feasible we use the same definition that has been used for incremental network.

Goldberg and Tarjan algorithm examines the active nodes in R, and pushes the excess local flow towards t along what it estimates to be the shortest path in the incremental network R(g). When in the current R(g) there are no more paths between a given active node x and t, then the algorithm takes back to the sources s the excess flow still existent in this node x again along estimated shortest paths. Eventually all the nodes, except the source and the terminal, have a null excess flow (dx = 0). Then the preflow is a maximum flow.

The only difficulty to solve here is the way the algorithm estimates the distance from a given node to the source and terminal nodes. To overtake this difficulty, Goldberg and Tarjan have introduced the concept of a valid labelling [12].

9. Computational Results

We have coded the algorithms of Dinic, Malhotra, Kumar and Maheshwari (MKM), Tarjan and Goldberg and Tarjan (Goldberg), using the Turbo Pascal 3.01 compiler, and run these programs on an IBM PS/2 Model 55SX (16 MHz clock speed, 2 Mbytes of RAM) using the MS-DOS 3.31 operating system.

The density is an important parameter when testing algorithms on networks. This is defined for a certain number of nodes as the ratio between the number of arcs that the network has and the maximum number of arcs it can have. Since the networks we use are asymmetrical, the maximum number of arcs a network of |N| nodes can have is $\frac{|N|.(|N|-1)}{2}$. Therefore the density of a network is given by $d = \frac{2.|A|}{|N|.(|N|-1)}$.

The examples used to test these programs have been generated using a Random Network Generator developed by the authors. This generator accepts as inputs the number of nodes of the network and its density. Then begining from the source node it uses the compiler random number generator to obtain node numbers that permit to build paths to the terminal node. The capacity of the arcs in these paths are also obtained using this random number generator and are integer numbers in the range [1,..., maxint]. The number of paths generated, and consequently the number of arcs, depends on the density chosen by the user.

For each value of |N| = 50, 100, 150 and 200 and d = 0.1, 0.2,..., 1.0, we have generated 100 problems and have stored them in files, so that the programs have been tested in exactly the same conditions.

In figure 4 we present the mean running times of the programs, obtained excluding input, output and preprocessing. The mean number of flow augmenting paths as displayed in figure 5.

In figures 6, 7, 8, 9, 10, 11, 12 and 13 we show in percentual terms the relative frequency of the deviations of the mean number of flow augmenting paths and of the mean running times, obtained for the same problems. Analysing these results we can conclude that the Goldberg-Tarjan algorithm is clearly superior to all the others for medium and high densities. For small densities the algorithms perform quite similarly, although the Dinic and Goldberg-Tarjan algorithms perform slightly better. As the number of nodes increases, even for sparse networks the Goldberg-Tarjan algorithm runs faster than all the others.

Notice however that the Goldberg-Tarjan algorithm is not as stable as the others, that is, the number of problems that run in a much higher or lower time than the average is greater for this algorithm than for any one of the others, especially for low densities.

We also observe that the mean number of flow augmenting paths computed by each algorithm is higher for the Goldberg-Tarjan algorithm than for the others. That number is also a quite unstable figure for this algorithm.

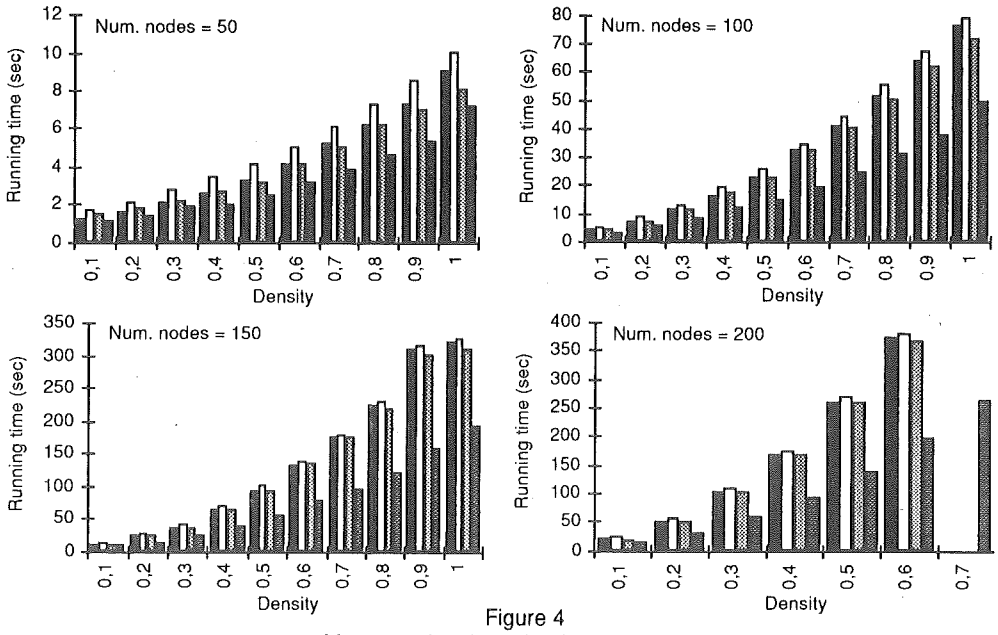


Figure 4
Mean running times for d=0.1, 0.2, ..., 1.0
Dinic MKM Tarjan Goldberg

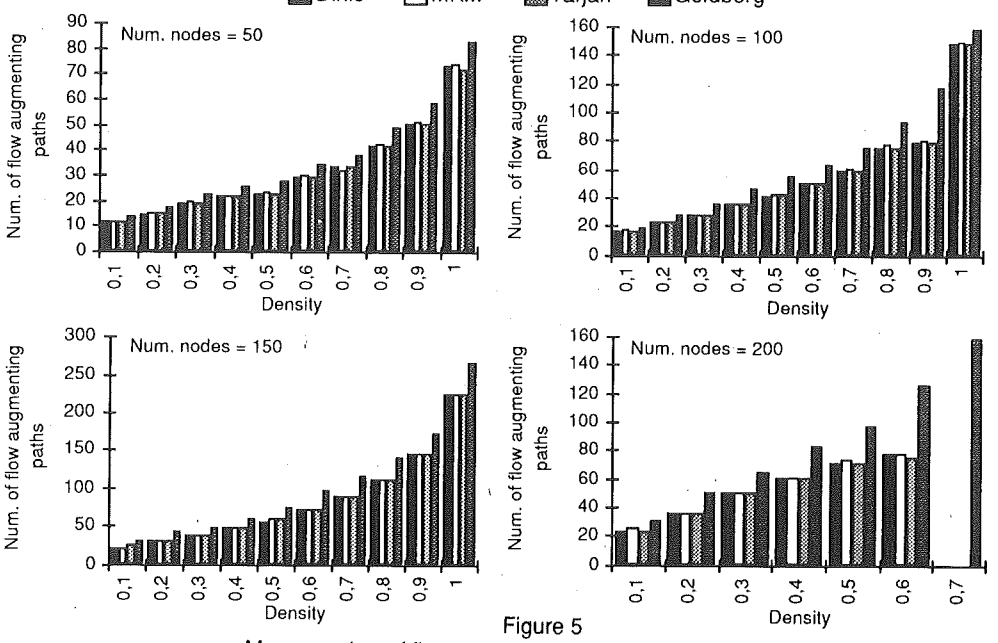


Figure 5
Mean number of flow augmenting paths for d=0.1, 0.2, ..., 1.0
Dinic MKM Tarjan Goldberg

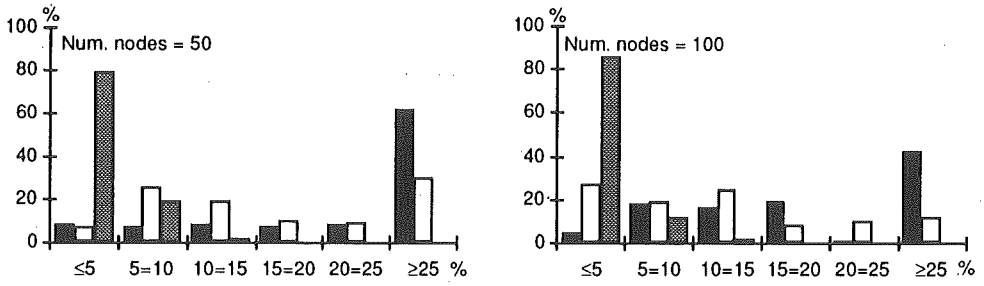


Figure 6
Relative frequency of the deviations in percent of the number of flow augmenting paths (Dinic algorithm)
■d=0,1 ▨d=0,5 □d=1,0

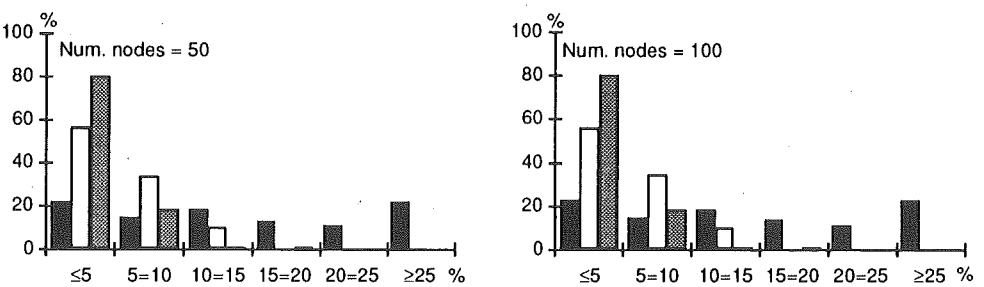


Figure 7
Relative frequency of the deviations in percent of the mean running time (Dinic algorithm)
■d=0,1 ▨d=0,5 □d=1,0

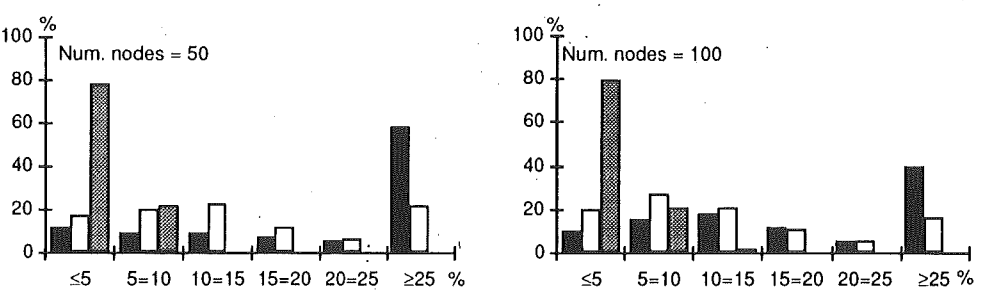


Figure 8
Relative frequency of the deviations in percent of the number of flow augmenting paths (Tarjan algorithm)
■d=0,1 ▨d=0,5 □d=1,0

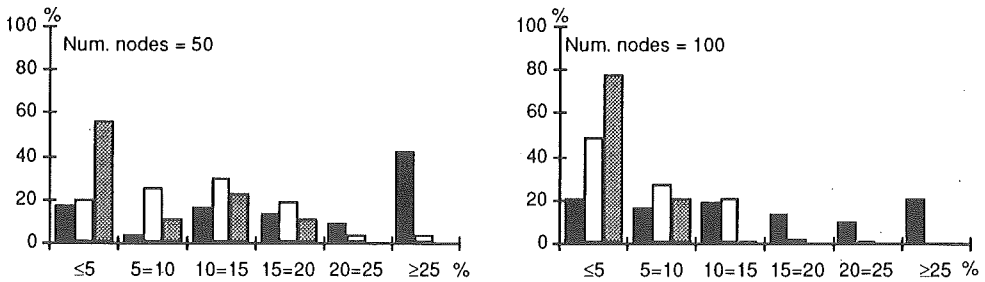


Figure 9
Relative frequency of the deviations in percent
of the mean running time (Tarjan algorithm)
■d=0,1 ▨d=0,5 □d=1,0

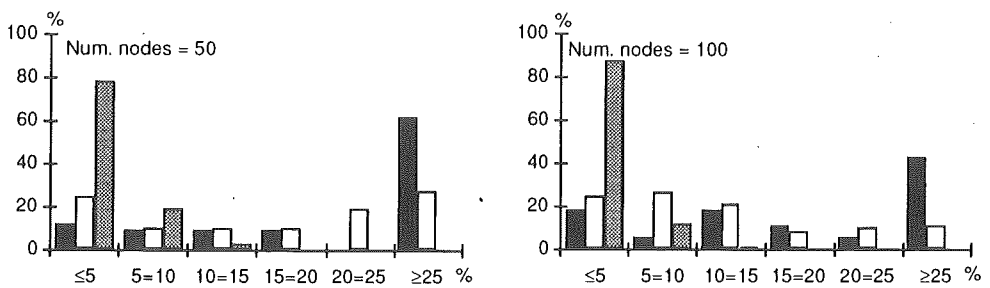


Figure 10
Relative frequency of the deviations in percent of the
number of flow augmenting paths (MKM algorithm)
■d=0,1 ▨d=0,5 □d=1,0

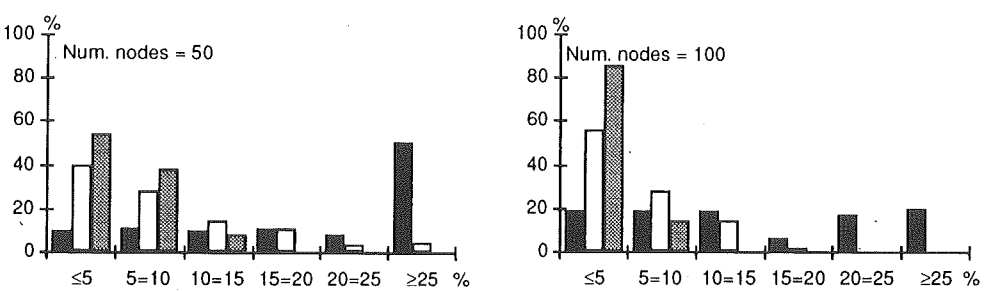


Figure 11
Relative frequency of the deviations in percent
of the mean running time (MKM algorithm)
■d=0,1 ▨d=0,5 □d=1,0

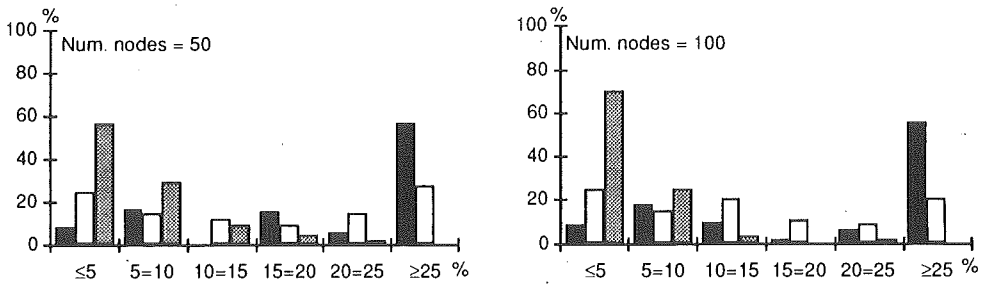


Figure 12
Relative frequency of the deviations in percent of the number of flow augmenting paths (Goldberg algorithm)
■d=0,1 ▨d=0,5 □d=1,0

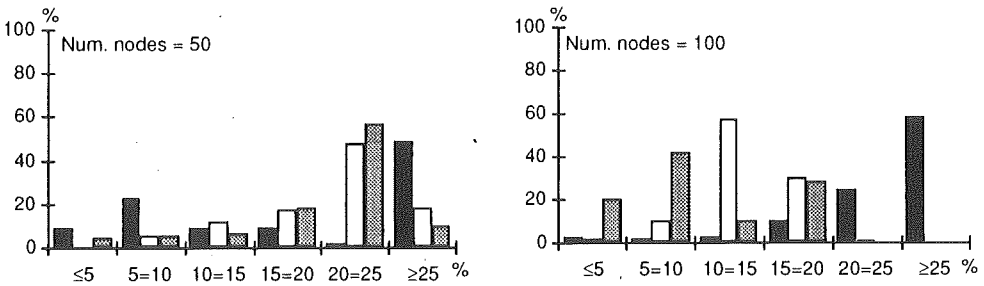


Figure 13
Relative frequency of the deviations in percent of the mean running time (Goldberg algorithm)
■d=0,1 ▨d=0,5 □d=1,0

This feature of the Goldberg-Tarjan algorithm deserves somes comments that may help explaining its behaviour. In the program code of the Goldberg-Tarjan algorithm we used a heuristic proposed by us to improve its performance. This heuristic consists in starting the algorithm with an exact valid labeling, $e(x) = \min\{d(s, x), d(x, t) + |N|\}$ for all $x \in N$, and after each $|N|$ calls of the label updating procedure it does another exact valid labeling. From a performance point of view, it shows that this algorithm is similar to the other preflow algorithms but without the aditional cost of having to rebuild the auxiliary network in each phase. However, there is still a cost of having to compute an exact labeling and we may not know exactly the right moment to do it. Sometimes we are unecessarily recalculating an exact label, while other times the performance might have been improved if it was recalculated earlier. This explains the reason why this version of the algorithm is not very stable.

Another important aspect is the much more compact and simple code of this algorithm. This is due to the fact that we need only to mantain one network (the incremental network) and it is smaller the number of data structures that are required to update than those for the other algorithms.

Note that the 640 Kbytes limit on the available memory imposed by the operating system, only permits to solve problems with a maximum of 200 nodes and a density of 0.7 for the Goldberg-Tarjan algorithms and 0.6 for all the others.

As a final conclusion we can say that all the algorithms can be used to solve the maximum flow problem in a microcomputer in useful time, but the Goldberg-Tarjan algorithm is the fastest procedure, particulary for dense networks.

References

[1] Ahuja, R.K. and Orlin, J.B., *A fast and simple algorithm for the maximum flow problem*, Tech.Rep.1905-87, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass., 1987.

[2] Dinic, E.A., *Algorithm for solution of a problem of maximum flow in networks with power estimation*, Soyietic Mathematic Doklady 11 (1970) 1277-1280.

[3] Edmonds, J. and Karp, R., *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the ACM 19 (1972) 248-264.

[4] Elias, P., Fenstein, A. and Shannon, C., *Note on maximum flow through a network*, IRE Transactions on Information Theory IT-2 (1956) 117-119.

[5] Even, S. and Tarjan R., *Network flow and testing graph connectivity*, SIAM Journal on Computing 4 (1975) 507-518.

[6] Ford, L.R. and Fulkerson, D.R., *Maximal flow trough a network*, Canadian Journal of Mathematics 8 (1956) 339-404.

[7] Ford, L.R. and Fulkerson, D.R., *Flows in networks*, Princeton University Press, Princeton, N.J., 1962.

[8] Fulkerson, D.R., *A network flow computation for project cost curves*, Management Science 7 (1961) 167-178.

[9] Gabow, H.N, *Scaling algorithms for network problems*, Journal of Computer and Systems Science 31 (1985) 148-168.

[10] Galil, Z., *An $O(V^{5/3}E^{2/3})$ algorithm for the maximal flow problem*, Acta Informática 14 (1980) 221-242.

[11] Galil, Z. and Naamad, A., *An $(EVlog^2V)$ algorithm for the maximum flow problem*, Journal on Computer and Systems Science 21 (1980) 203-217.

[12] Goldberg, A. and Tarjan, R., *A new approach to the maximum flow problem*, Journal of the ACM 35 (1988) 921-940.

[13] Gondran, M. and Minoux, M., *Graphs and algorithms*, Wiley Interscience, Chichester, 1984.

[14] Gusfield, D., Martel, C. and Fernandez-Bava, D., *Fast algorithms for bipartite network flow*, SIAM Journal on Computing 16 (1987) 237-251.

[15] Hut, T.C., *Optimum communication spanning trees*, SIAM Journal on Computing 3 (1974) 188-195.

[16] Karzanov, A.V., *Determining the maximum flow in a network by the method of preflows*, Soviet Mathematic Doklady 15 (1974) 434-437.

[17] Kelley, Jr., J., *Critical-path planning and scheduling: Mathematical basis*, Operations Research 9 (1961) 296-320.

[18] Malhotra, V., Kumar, M. and Maheshwari, S., *An $O(|V|^3)$ algorithm for finding maximum flows in networks*, Information Processing Letters 7 (1978) 277-278.

[19] Orlin, J.B. and Ahuja, R.K., New distance-directed algorithms for maximum flow and parametric maximum flow problems, Tech.Rep. 1908-87, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass., 1987.

[20] Picard, J. and Queyranne, M., Selected applications of minimum cuts in networks, INFOR 20 (1982) 394-422.

[21] Stone, H., Critical load factors in two-processor distributed systems, IEEE Transactions on Software Engineering SE-4 (1978) 254-258.

[22] Sleator, D. and Tarjan, R., A data structure for dynamic trees, Journal of Computer and Systems Science 26 (1983) 362-391.

[23] Sleator, D. and Tarjan, R., Self-adjusting binary search trees, Journal of the ACM 32 (1985) 652-686.

[24] Tarjan, R., A simple version of Karzanov's blocking flow algorithm, Operations Research Letters 2 (1984) 265-268.

[25] Tardos, E. A strongly polynomial minimum cost circulation algorithm, Combinatorica 5 (1985) 247-255.

INVESTIGAÇÃO OPERACIONAL, vol. 13, n.° 2 (Dezembro 1993) 149-168

O PROCESSAMENTO PARALELO EM PROBLEMAS DE OPTIMIZAÇÃO

João P. Costa
João N. Clímaco
Faculdade de Economia – Universidade de Coimbra

INESC - Núcleo de Coimbra
Av. Antero Quental, nº 231-cv
3000 Coimbra

Abstract

The recent development of low cost computer parallel architectures raised the possibility of overcoming some traditional difficulties of optimization through parallel processing.

This paper presents a 'ill-structured' approach to map sequential optimization algorithms into transputer network parallel environments. First two illustrative examples are presented: the parallelization of a method to find all the alternative basic solutions of a linear programming problem and the parallelization of Dijkstra's method to find the shortest path from a node to all the others of a network. Second some methodological suggestions about the 'art' of transforming sequential algorithms into parallel ones are presented. Finally some hints about the algorithm's suitability for parallel processing are pointed out.

Resumo

O recente desenvolvimento de arquitecturas paralelas de baixo custo levou a comunidade científica a encarar a possibilidade de ultrapassar algumas dificuldades tradicionais dos problemas de optimização, utilizando o processamento paralelo.

Pretende-se neste artigo apresentar uma abordagem semi-estruturada à tarefa de paralelizar algoritmos normalmente utilizados na resolução de problemas de optimização. Assim, com vista a ilustrar a parte central da comunicação, apresentam-se na sua parte introdutória dois problemas concretos, nomeadamente, a paralelização de métodos para determinar todas as soluções básicas óptimas alternativas de um problema de programação linear (PPL), a partir de uma solução básica óptima, e para determinar a árvore dos caminhos mais curtos de um nó a todos os outros de uma rede.

Na parte central da comunicação fazem-se algumas considerações sobre a paralelização de algoritmos em redes de "transputers". Isto é, apresentam-se sugestões metodológicas de abordagem à "arte" do desenvolvimento em paralelo de algoritmos de optimização já existentes e largamente utilizados em máquinas sequenciais; tentam-se identificar algumas características destes algoritmos que indiciem a obtenção de bons resultados, face às técnicas de processamento paralelo disponíveis em redes de "transputers"; e indicam-se ainda, as características fundamentais que devem possuir os algoritmos desenvolvidos especificamente para processamento nas redes actuais de "transputers".

Keywords

Multiprocessing parallel algorithms, Combinatorial optimization.

1. Introdução

Ao longo dos últimos anos, muitos têm sido os modelos de computação propostos com o objectivo de fornecer os recursos necessários à resolução dos problemas de grandes dimensões e/ou complexidade que se põem hoje em dia na maior parte dos ramos do conhecimento. Os limites físicos da tecnologia actual de semi-condutores (silício) para o modelo sequencial de Von-Neumann são conhecidos e foram praticamente atingidos pelos microprocessadores actualmente existentes no mercado. Daí o aparecimento e interesse dos modelos computacionais baseados em arquitecturas paralelas. No entanto, devido às limitações tecnológicas actuais, quase todos os computadores paralelos existentes no mercado, baseiam-se nas arquitecturas mais simples e fáceis de executar.

Nesta comunicação será apenas abordado o processamento paralelo em máquinas cujas arquitecturas se baseiam em redes de "transputers". Um mito comum e que se ouve repetir muitas vezes, é o de que um programador não necessita de perceber o "hardware" que utiliza. Como todos os mitos, contém algo de verdadeiro. No entanto, sempre que a eficiência se torna crítica, os programadores utilizam o seu conhecimento sobre paginação, tamanho da memória "cache", registos do microprocessador, etc, para afinar as suas implementações. Devido à grande variedade de arquitecturas existentes, a situação para o processamento paralelo ainda é pior do que para o sequencial: o estilo do código depende frequentemente do paralelismo conseguido através do "hardware". Assim começar-se-á por descrever, ainda que sucintamente, o tipo de processamento paralelo sobre o qual esta comunicação se baseia. Na secção 3 apresentar-se-ão dois problemas concretos: um algoritmo para determinar todas as soluções óptimas alternativas de um problema de programação linear (PPL), a partir de uma solução básica óptima, e um outro algoritmo para determinar a árvore dos caminhos mais curtos de um nó a todos os outros de uma rede. A apresentação destes algoritmos tem por intuito ilustrar a parte seguinte do artigo. Nessa parte, a partir de algumas características do ambiente computacional, tais como tempos de comunicação e processamento e estruturas lógicas utilizadas na partilha de recursos, introduzem-se sugestões metodológicas de abordagem à "arte" de desenvolvimento em paralelo de algoritmos aplicados a problemas de optimização. Identificam-se também, algumas características de problemas de optimização, que indiciem a obtenção de bons resultados a partir da aplicação das técnicas de processamento paralelo, disponíveis no ambiente computacional utilizado.

2. Caracterização do ambiente de desenvolvimento

O ambiente computacional utilizado compreende uma placa com quatro transputers IMS T800-20, instalada num IBM-PC-AT compatível, equipado com um microprocessador INTEL 286 (hospedeiro). O diagrama das ligações entre os transputers e o hospedeiro é o do fig.1.

Um transputer é um componente programável desenhado para suportar sistemas com um grau elevado de concorrência e/ou paralelismo, constituindo um microprocessador com memória montada no próprio componente e quatro elos de ligação ponto a ponto, que podem ser ligados aos elos de ligação de outros transputers [15]. O interface de controlo dos elos de ligação e o próprio microprocessador funcionam em paralelo, permitindo continuar o processamento ao mesmo tempo que há transferência de dados nos elos de ligação.

Os transputers podem ser utilizados em sistemas com apenas um processador, ou, dado possuirem os elos de ligação, podem ser facilmente utilizados para construir redes de processadores, constituindo sistemas de grande capacidade de processamento e elevado grau de paralelismo. O paralelismo nestes sistemas baseia-se no modelo de processos sequenciais comunicando através de mensagens. Um processo é encarado como uma caixa preta que entrega processada, à saída, informação de entrada. Neste modelo computacional o sistema é constituído por um conjunto de processos sequenciais que só podem comunicar entre si através de canais específicos enviando mensagens. Um canal é então um modo de ligação unidireccional entre dois processos. No ambiente computacional utilizado as mensagens têm um tamanho limite máximo. Se a mensagem a enviar ultrapassar este limite, é necessário dividi-la em vários pedaços e enviá-los um a um como mensagens independentes. Se a cada um destes pedaços de mensagens (ou à própria mensagem, se esta não exceder os limites máximos) se acrescentar informação de controlo (necessária para o protocolo de comunicação) obtemos um *pacote* de dados a transmitir de um processo para o outro. Quando um processo quer enviar uma mensagem através de um canal tem que esperar até que o processo receptor esteja preparado/pronto para ler a mensagem - comunicação sincronizada.

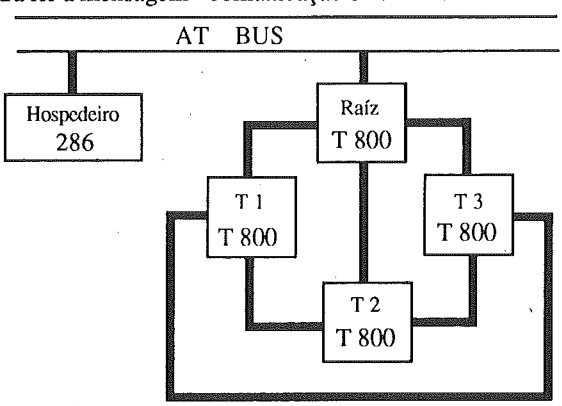


Figura 1 – Diagrama de ligações entre os transputers e o microprocessador onde a placa está instalada

Do que até aqui se disse, pode-se enquadrar a máquina utilizada dentro das que utilizam multiprocessamento como forma de obter as suas características de processamento paralelo. Num sistema multiprocessador, cada transputer pode ser visto como a materialização de um processo. No entanto, é também possível ter vários processos (o número de processos é limitado pela quantidade de memória disponível em cada transputer) num só processador, aos

quais se podem atribuir as prioridades: "urgente" e "não urgente". Um processo pode ser interrompido por ter executado uma instrução que o faça esperar (um semáforo por exemplo) ou, e só no caso dos processos não urgentes, por ter executado sem interrupções durante um certo período de tempo, pois neste caso o transputer partilha igualmente, entre todos os processos, a sua capacidade de processamento.

A execução óptima de um algoritmo numa rede de transputers depende da partição do problema correspondente em módulos de execução sequencial e do seu escalonamento pelos processadores disponíveis, com vista a obter o menor tempo de execução possível. Estes módulos denominam-se normalmente por *grãos* de processamento [10], numa tentativa de distinção do termo processo. Assim apenas será considerado com fazendo parte do grão a quantidade de código necessária para resolver o problema em questão. Se se juntar ao grão o necessário código de controlo e comunicação, obtém-se o chamado processo. O tamanho do grão é muito importante na procura do algoritmo óptimo, pois se este é demasiado grande o paralelismo é limitado e se é demasiado pequeno os atrasos resultantes das comunicações e controlo necessários reduzem a eficiência global do algoritmo. A granularidade deve ser determinada como um compromisso entre o tempo de processamento do grão, o atraso devido ao controle e o atraso resultante das comunicações. O número de processadores requeridos por um algoritmo paralelo está também intimamente ligado à sua granularidade. Os algoritmos em paralelo podem geralmente utilizar eficazmente apenas um número N de processadores. Acima deste N, os ganhos em eficiência podem diminuir significativamente ou desaparecer, quando todas as oportunidades de paralelização estiverem exploradas.

2.1 Técnicas de processamento

A linguagem de programação utilizada nesta comunicação é o Parallel C da 3L [12]. Nesta secção descrevem-se genericamente algumas características que conferem paralelismo explícito a esta linguagem. Os processos são codificados em *tarefas* ("tasks"), cada uma delas com a sua própria área de endereçamento de memória para código, dados e portos de entrada e saída.

Existe um programa configurador que tem por função distribuir as tarefas, que constituem uma aplicação, pelos processadores do sistema. Esta distribuição é feita uma única vez, no ínicio da execução da aplicação, sem que possa ser alterada dinamicamente, isto é, alterada no decorrer da execução da aplicação. As tarefas apenas podem comunicar entre si por meio de mensagens que enviam através de canais. As tarefas que se localizam no mesmo processador podem ter um qualquer número de canais a interligá-las, mas aquelas que se localizam em processadores diferentes só podem ser interligadas através dos canais para os quais existam as respectivas ligações físicas entre os processadores. Cada canal lógico entre tarefas colocadas em processadores diferentes utiliza exclusivamente um canal físico de ligação. Se forem necessários mais canais lógicos de ligação entre tarefas do que os existentes fisicamente, será necessário desenhar tarefas com a função de multiplexar os diferentes canais lógicos num único

a que esteja associada uma ligação física. O exemplo da secção 3.2, versão 2, ilustra esta situação.

O código de uma tarefa pode ser escrito em linguagem C de uma forma sequencial, excepto no que diz respeito a uma característica necessária às linguagens baseadas no modelo de processos sequenciais, que comunicam através de mensagens. Ou seja, pode ser necessário que uma tarefa espere até receber uma mensagem de um qualquer número de canais lógicos de entrada. Por exemplo, no caso em que o ciclo principal de uma tarefa tem de esperar que uma qualquer das suas tarefas "clientes" disponibilize uma mensagem, esta não pode verificar sequencialmente todos os canais de entrada de mensagens, pois a mensagem pode vir através de qualquer um deles, por qualquer ordem. Se a tarefa tenta verificar se há alguma mensagem disponível num canal, fica ligada a esse canal até que a mensagem apareça, podendo entretanto aparecer outra mensagem noutro canal. No Parallel C, esta característica é conseguida à custa da criação de novos *fios* ("threads") de execução concorrente: a tarefa cria um fio de execução para cada um dos canais de entrada, com a função de verificar se chegou alguma mensagem ao canal a que está associado. Assim as mensagens podem ser tratadas de uma forma concorrente, pela ordem a que chegam à tarefa. No exemplo secção 3.1. ilustra-se esta característica criando-se dois fios de execução na tarefa coordenadora: um para receber os resultados e outro para enviar dados.

Um fio criado por uma tarefa, executa em concorrência com qualquer outro fio dessa tarefa, no entanto, um fio não é aquilo que habitualmente se designa por processo, pois enquanto um processo pode existir autonomamente (a menos de operações de entrada e saída que eventualmente necessite) um fio necessita do enquadramento de uma tarefa. Basicamente pode definir-se um fio como sendo um módulo ("function" no Parallel C), de uma tarefa, que executa em concorrência com outros da mesma tarefa, partilhando com eles a área de memória da "heap" e do código, e apenas mantendo privada a sua zona de memória da pilha ("stack"). Comparando os fios de execução com as tarefas pode dizer-se que:

- Cada tarefa tem o seu próprio código (incluíndo as bibliotecas de "run-time") e área de dados, separados de todas as outras tarefas, mesmo que pertençam ao mesmo processador, ao passo que os fios de execução partilham o código e área de dados com todos os outros fios que pertençam à mesma tarefa.
- Os fios de execução podem comunicar utilizando canais como os das tarefas (desde que pertençam a tarefas distintas) ou através de memória partilhada (desde que pertençam à mesma tarefa - zona da "heap"), enquanto as tarefas só podem comunicar através de canais.

A linguagem de programação utilizada possui ainda características que permitem aplicar uma técnica de programação paralela que se passará a designar por *rede de cálculo indiferenciado* ("farm processing"), RCI, desde que cada processador possua memória disponível para o código adicional que suporta uma rede e trata as suas mensagens, e que a

aplicação seja adequada a este tipo de técnica. Nesta técnica uma aplicação é constituída por uma tarefa designada "coordenadora", que divide o trabalho a realizar em pacotes pequenos e independentes, e por diversas outras tarefas, designadas por "tarefas de cálculo", que processam o trabalho empacotado pela coordenadora. Os pacotes, lançados para a rede pela coordenadora, são distribuídos automaticamente pela rede de processadores, através de um programa de encaminhamento. A tarefa de cálculo que esteja livre de momento (cada uma localizada num processador diferente), aceita um dos pacotes de trabalho da rede, processa-o, e lança os resultados para a rede, que se encarrega de os encaminhar para a coordenadora, ficando a tarefa de cálculo novamente livre e pronta a aceitar mais pacotes de trabalho. Na secção 3.1. ilustra-se esta técnica e na secção 4, indica-se o seu campo de aplicação.

3. Aplicações

Nesta secção apresentam-se, como exemplos ilustrativos, os algoritmos paralelos de duas aplicações. O primeiro algoritmo permite determinar todas as soluções básicas óptimas alternativas de um PPL, a partir de uma solução básica óptima, e ilustra a técnica RCI, referida na secção anterior. O segundo permite determinar a árvore dos caminhos mais curtos de um nó a todos os outros de uma rede e ilustra a utilização de algumas características paralelas do ambiente computacional utilizado.

Embora se comentem genericamente alguns dos resultados obtidos dos testes efectuados à implementação da versão 1 do algoritmo apresentado na secção 3.2, não se apresentam resultados computacionais de testes de comparação da eficiência dos algoritmos apresentados. Tendo em atenção os objectivos desta comunicação, a apresentação desses resultados não faria sentido, pois os algoritmos pretendem ser meramente ilustrativos das técnicas de processamento paralelo, tendo-se priveligiado a clareza e simplicidade em detrimento da eficiência (caso contrário teríamos que entrar em consideração com questões tais como estruturas de dados e alocação de espaço em memória, que fogem ao âmbito desta comunicação). Por outro lado a eficiência dos algoritmos teria que ser comparada não só com a eficiência dos algoritmos sequenciais correspondentes, mas também com a eficiência de outros algoritmos baseados no mesmo modelo paralelo e mesmo noutros modelos paralelos, o que implicaria um aumento significativo das dimensões desta comunicação.

3.1 Óptimos alternativos

Nesta secção apresenta-se como exemplo ilustrativo da técnica RCI, a paralelização do algoritmos que determina todos os óptimos alternativos de um PPL. Este algoritmo, cuja descrição sequencial se encontra no livro de Steuer (Cap. 4) [14], constitui uma terceira fase do método de Simplex. Esta fase é aplicável sempre que a base óptima obtida no final da segunda fase do método não fôr única, e consiste em pivotar para todas as base óptimas alternativas, para-calcular todos os pontos básicos óptimos alternativos e/ou arestas óptimas ilimitadas. Omite-se a demonstração da validade do algoritmo, podendo esta, no entanto, encontrar-se em

[14]. Este algoritmo foi escolhido como exemplo ilustrativo da técnica RCI, por ser essencial na maior parte das características inovadoras da base de métodos multiobjectivos TOMMIX [1]. Nesta base de métodos a introdução interactiva de restrições adicionais aos valores das funções objectivo requer o cálculo, por repetidas vezes, de todos os óptimos alternativos de um PPL. Por outro lado a sua estrutura e simplicidade tornam este algoritmo adequado à ilustração da passagem de um algoritmo sequencial a paralelo, através da técnica referida.

O algoritmo utiliza as seguintes listas:

LB – Contém todas as base óptimas codificadas.

LX – Contém os pontos extremos óptimos codificados, correspondentes às entradas em LB

Le – Contém os pontos extremos óptimos codificados, dos quais emanam arestas óptimas ilimitadas.

Ld – Contém as direcções codificadas das arestas óptimas ilimitadas que emanam das correspondentes entradas em Le.

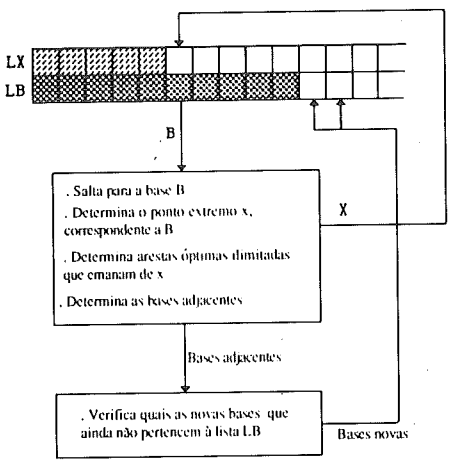


Figura 2. Representação simplificada do algoritmo sequencial da terceira fase do método de Simplex

Os elementos correspondentes em LB e LX estão associados como pares, tal como os elementos correspondentes em Le e Ld. "Saltando" de elemento em elemento de LB e analisando cada base óptima à procura de novas bases óptimas adjacentes, que se guardam em LB, obtém-se um meio de relembrar onde é que "já se esteve" e onde é que "ainda se tem que ir". "Saltar" é o processo pelo qual o algoritmo se move de uma base para outra, podendo esta ser ou não adjacente. Onde "já se esteve" é indicado pelas bases em LB para onde já se saltou e que têm o elemento correspondente em LX já preenchido. Para onde "ainda se tem de ir" é indicado pelas bases em LB, para onde já se saltou e que têm o elemento correspondente em LX já preenchido. Para onde "ainda se tem de ir" é indicado pelas bases em LB, para onde ainda não se saltou e portanto que ainda não têm a entrada correspondente em LX. De início, LB

cresce mais rapidamente do que o processamento das entradas da lista, mas, mais tarde ou mais cedo, o processo de saltar encontra o fim da lista LB preenchida. Quando a última base da lista foi processada e não se detectam mais bases óptimas adjacentes que ainda não pertençam à lista, a terceira fase do método de Simplex termina. A fig. 2 apresenta uma representação simplificada do algoritmo sequencial que implementa a terceira fase do método de Simplex. Note-se que no esquema apenas se refere o fluxo e controlo de dados.

A técnica RCI foi aplicada na paralelização deste algoritmo. O desenho do algoritmo paralelo foi feito por forma a alcançar um compromisso entre a maximização do grau de paralelismo e a minimização do tempo de processamento consumido em comunicações e controlo.

Na fig. 3 apresenta-se uma versão simplificada do algoritmo paralelo. O processamento correspondente aos blocos apresentados no mesmo nível horizontal executam em "simultâneo": em concorrência no caso da tarefa coordenadora e em paralelo, em processadores diferentes, no caso das tarefas de cálculo.

Seguidamente apresenta-se a sistematização do algoritmo de cálculo em paralelo dos óptimos alternativos de um PPL.

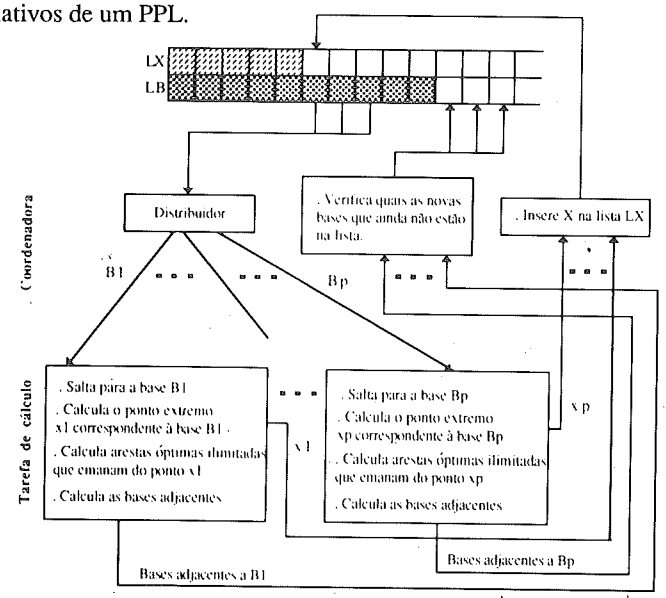


Figura 3. Representação simplificada do algoritmo paralelo da terceira fase do método de Simplex

Notação utilizada:

n - Número de variáveis do PPL.

m - Número de restrições do PPL.

B - Matriz, m × m, da base.

A - Matriz, m × n, dos coeficientes tecnológicos.

c - Vector, n × 1, constituido pelos coeficientes da função objectivo.

$Y = B^{-1} A$.

c_B - Vector, $m \times 1$, constituido pelos coeficientes da função objectivo das variáveis básicas.

c' - Transposto do vector c.

$c' - c'_B Y$ - Vector, $1 \times m$, dos custos reduzidos.

b - Vector, $m \times 1$, constituido pelos termos independentes das restrições.

t_j - Elemento j do vector t.

E_{ij} - Elemento da linha i, coluna j, da matriz E.

Algoritmo para "saltar" de base em base (que podem ser não-adjacentes):

O quadro de Simplex óptimo inicial está completo.

1. Selecciona-se, no quadro, as colunas correspondentes às variáveis da nova base para onde se quer saltar. Expandem-se estas colunas com a última da matriz identidade I_{m+1} e constroi-se a base B_{m+1}.
2. Calcula-se a inversa da matriz B_{m+1}.
3. Multiplica-se a última linha da matriz invertida pelo quadro óptimo inicial (só para as variáveis não básicas na nova base) para calcular o vector dos custos reduzidos, do novo quadro.
4. Para todos os elementos calculados (não básicos), do vector $c' - c'_B Y$, que são nulos:
   4.1. Calcular a coluna correspondente da matriz Y:
   4.2. Para todo o i:
      4.2.1. Se $Y_{ij} \neq 0$ e $b_i = 0$ então tem-se uma base óptima adjacente (e degenerada)
   4.3. Se para todos o i $Y_{ij} \leq 0$ então tem-se uma aresta óptima e ilimitada, se não para o menor valor positivo de b_i/Y_{ij}, tem-se uma base óptima adjacente.

Algoritmo da tarefa coordenadora:

1. Inicialização
2. Seleccionar do quadro óptimo inicial as bases óptimas adjacentes, e as arestas óptimas ilimitadas que emanam do ponto extremo inicial.
3. Guardar os respectivos códigos nas listas LB, LX, Le e Ld.
4. Criar dois fios de execução.

   Fio de execução (‘):

   4.1’. Seleccionar da lista LB uma base que não tenha a correspondente entrada na lista LX.

   4.2’. Enviar essa base às tarefas de cálculo através da rede.

4.3'. Se não existirem mais bases na lista LB, sem as correspondentes entradas na lista LX, o fio de execução pára, se existirem volta ao ponto 4.1'.

Nota: Mantém-se o registo do número de ordem da última base enviada, para evitar a mesma base mais do que uma vez.

Fio de execução (''):

4.1''. Esperar por uma mensagem de resultados vinda das tarefas de cálculo.

4.2''. Guardar os códigos correspondentes ao ponto extremo, às arestas óptimas ilimitadas e às novas bases, nas listas respectivas. O ponto extremo forma um par com a sua base correspondente. As novas bases serão armazenadas na lista LB, a não ser que já lá existam.

4.3''. Se não existirem bases na lista LB sem o correspondente ponto extremo em LX, o fio de execução pára, se existirem, volta ao ponto 4.1''.

Nota 1: Quando um fio de execução acede a variáveis partilhadas, protege a sua entrada com semáforos.

Nota 2: Quando o fio referenciado por (') não encontra nada para enviar, pendura-se num semáforo até o fio referenciado por ('') escrever qualquer coisa nas listas.

Algoritmo da tarefa de cálculo (existem várias idênticas):

Esta tarefa mantém o quadro óptimo inicial.

1. Inicialização
2. Espera por uma mensagem da tarefa coordenadora, com uma base para saltar.
3. Salta para a base enviada pela coordenadora.
4. Cria uma mensagem com:
   - A base enviada pela coordenadora.
   - O ponto extremo correspondente a essa base.
   - As arestas óptimas ilimitadas que emanam desse ponto extremo.
   - As novas bases óptimas adjacentes à que foi enviada pela coordenadora.
5. Envia a mensagem à coordenadora através da rede e volta ao passo 2.

3.2 Caminho mais curto

A determinação do caminho mais curto de um nó a todos os outros nós de uma rede pode ser considerado um clássico de redes em optimização. O algoritmo mais conhecido para este tipo de problemas é o devido a Dijkstra, apresentado por exemplo em [8], embora recentemente existam outros que apresentam melhores resultados em determinadas circunstâncias, por exemplo [13] ou [6,7].

Um algoritmo paralelo que determina o caminho mais curto de um nó a todos os outros de uma rede, será apresentado seguidamente. Este algoritmo paralelo tem por origem o algoritmo

de Dijkstra, aplicando-se portanto, também a ele, as restrições do algoritmo de Dijkstra, que por isso não serão aqui referidas. É também por esse motivo que não se apresenta demonstração da sua validade.

O algoritmo é apresentado em duas versões ilustrativas: a primeira é adequada ao ambiente computacional descrito na secção 2 e com a topologia da fig.1, enquanto a segunda pode ser utilizada numa rede com a topologia da fig.5, com um qualquer número de conjuntos de 4 transputers.

Notação utilizada:

n - Número máximo de nós da rede.

N - Conjunto de nós da rede.

c_{ij} - Custo do arco que liga o nó i ao nó j. Se o arco não existe $c_{ij} = +\infty$.

r - Nó origem.

d_i - Distância do nó origem ao nó i, $\forall i \in N$.

p_i - Nó predecessor do nó i.

T - Conjunto de transputers na rede, excluindo aquele que está ligado ao hospedeiro (processador da máquina na qual está instalada a placa de transputers, ver fig.1).

t - Número máximo de transputers na rede, excluindo o que está ligado ao hospedeiro. Coincide com o número das tarefas de cálculo. Inicialmente t = 3.

Versão 1: (para o ambiente computacional da fig.1)

A aplicação possui basicamente duas tarefas distintas: uma que se situa no transputer ligado ao hospedeiro, que se designa por "coordenadora" e outra da qual existe uma cópia em cada um dos outros três transputers da rede, que se designa por "tarefa de cálculo" (veja-se fig.4).

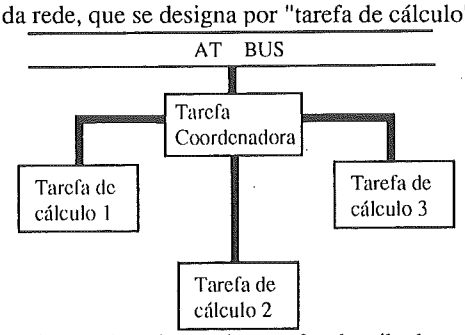


Figura 4. Distribuição da tarefa coordenadora e das tarefas de cálculo pela rede de transputers

Seguidamente apresentam-se os algoritmos de ambas as tarefas.

Algoritmo da tarefa coordenadora:

1. Enviar c_{ij}, $\forall i,j \in N$ a todas as tarefas de cálculo.
2. Dividir o número de nós da rede por cada uma das tarefas de cálculo (e transputers):

   Para T_1 os nós $\{1, t+1, 2t+1, 3t+1, \ldots\} = N_1$, $\#N_1 = n_1$

   Para T_2 os nós $\{2, t+2, 2t+2, 3t+2, \ldots\} = N_2$, $\#N_2 = n_2$

   ...

   Para T_t os nós $\{t, t+t, 2t+t, 3t+t, \ldots\} = N_t$, $\#N_t = n_t$

3. Enviar N_i à tarefa de cálculo i, situada no transputer i, i = 1, 2,..., t.
4. $d_r \leftarrow 0$; $p_r \leftarrow r$.
5. $d'_r \leftarrow 0$; $p'_r \leftarrow r$; $r' \leftarrow r$.
6. Para todos os transputers que ainda pertençam a T (isto é, para as tarefas de cálculo que ainda tenham nós por analizar): Enviar d'_r, r'.
7. Para todo o $i \in T$:
   7.1. Recebe j^*_i, p^*_{ij} e d^*_{ji}.
   7.2. Se $j^*_i = -1$ então $T \leftarrow T \setminus \{i\}$.
   7.3. Se $T = \{\}$ então pára.
8. $s \leftarrow \min \left\{ s | (d_s = \min\{d^*_{jk} \mid k \in T\}) \wedge (s \in \{j^*_i \mid i \in T\}) \right\}$.
9. $d'_r \leftarrow d_s$; $p'_r \leftarrow p_s$; $r' \leftarrow s$.
10. Volta ao passo 6.

Algoritmo da tarefa de cálculo i, $i \in T$:

1. Recebe c_{ij}, $\forall i,j \in N$.
2. Recebe N_i.
3. $d_{N_i} \leftarrow +\infty$; $p_{N_i} \leftarrow -1$.
4. Recebe r e d_r, da tarefa coordenadora.
5. Se $r \in N_i$ então
   5.1 $N_i = N_i \setminus \{r\}$
   5.2 $d_{N_i r} = d_r$ (Este passo só é necessário na primeira iteração)
6. Se $N_i = \{\}$ então
   6.1 Envia à tarefa coordenadora: $j^* = -1$, $p^*_j = -1$ e $d^*_j = -1$.
   6.2 Pára.
7. Para todo o $j \in N_i$, se $d_j > d_r + c_{rj}$:
   7.1 $d_j \leftarrow d_r + c_{rj}$.
   7.2 $p_j \leftarrow r$.
8. $j^* \leftarrow \min \left\{ j \mid (d_j = \min\{d_k \mid k \in N_i\}) \wedge (j \in N_i) \right\}$.
9. Envia à tarefa coordenadora: j^*, p^*_j e d^*_j
10. Volta ao passo 4.

Apresentam-se agora alguns comentários aos resultados obtidos nos testes efectuados à implementação deste algoritmo de cálculo de caminho mais curto de um nó a todos os outros de uma rede. Foram efectuados testes a diversas redes (geradas aleatoriamente) com as seguintes razões entre o número de arcos e o número de nós: 1, 2, 3, 4, 5 e 6. O número de nós da rede, por sua vez variou entre o número 100 e 1200, com incrementos de 100. Por outro lado testaram-se estas redes geradas aleatoriamente para as seguintes configurações da máquina paralela:

a) apenas um transputer (situando-se neste transputer a tarefa coordenadora e a tarefa de cálculo, interligadas sequencialmente - caso sequencial);

b) dois transputers (situando-se num a tarefa coordenadora e no outro a tarefa de cálculo);

c) três transputers (uma tarefa coordenadora e duas de cálculo);

d) quatro transputers (uma tarefa coordenadora e três de cálculo).

Os tempos de execução obtidos com a topologia b) foram mais elevados que os tempos obtidos com a topologia a), de uma ordem sempre inferior a 6.5%, notando-se tendência para a diminuição desta percentagem, com o aumento das dimensões das redes. Esta diferença deve-se essencialmente aos tempos de comunicação entre as duas tarefas da alínea b). Os tempos de execução obtidos com a topologia b) foram mais elevados que os tempos obtidos com as topologias c) e d), de ordem sempre superior a 197% e 296% respectivamente, notando-se a tendência para atingir os 200% e 300% respectivamente, com o aumento das dimensões das redes em teste. Daqui podemos concluir que os tempos de execução diminuem linearmente com o número de processadores envolvidos, para as dimensões e topologias indicadas.

Versão 2: (para o ambiente computacional da fig.5)

Nesta versão considera-se a existência de quatro tarefas distintas:

- tarefa coordenadora, igual à da versão 1,
- tarefa de cálculo, igual à da versão 1,
- tarefa subcoordenadora I, e
- tarefa subcoordenadora II.

A localização destas quatro tarefas na rede de transputers é a indicada na fig. 5.

Basicamente as tarefas subcoordenadas cumprem funções de encaminhamento de dados na rede. Elas podem também, ser consideradas tarefas multiplexadoras, veja-se secção 2, pois transformam um canal real em vários canais lógicos. À parte estas funções, cabe-lhes algum cálculo, como se pode ver na descrição seguinte.

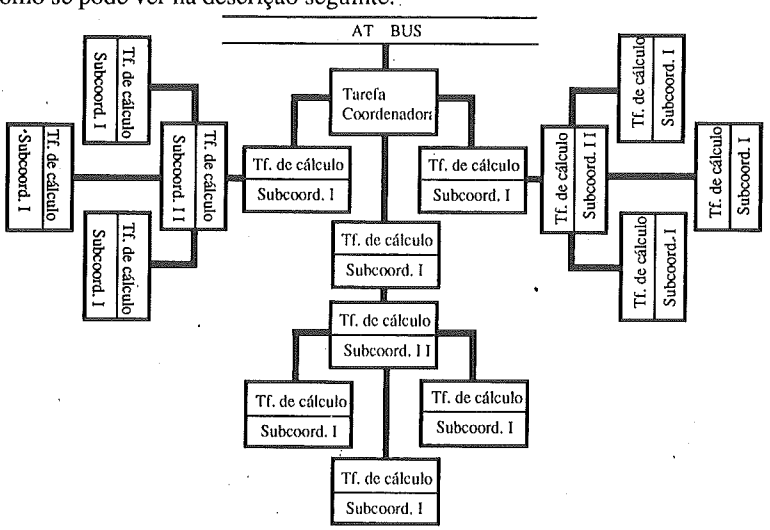


Figura 5. Distribuição das tarefas pela rede de transputers

Algoritmo da tarefa subcoordenadora I:

1. Recebe, c_{ij} $\forall i,j \in N$, da tarefa subccordenadora II, que lhe está associada a montante ou da tarefa coordenadora.
2. Envia c_{ij}, $\forall i,j \in N$, à tarefa de cálculo com que partilha o transputer e à tarefa subcoordenadora II a que está ligado a jusante.
3. Recebe b conjuntos de nós.
4. Envia o primeiro conjunto de nós à tarefa de cálculo com que partilha o transputer.
5. Envia restantes conjuntos de nós (b-1 conjunto de nós) à tarefa subcoordenadora II que lhe está ligada a jusante.
6. Recebe r' e d'_r.
7. Envia r' e d'_r à tarefa de cálculo com que partilha o transputer se esta ainda tem nós para processar, e à tarefa subcoordenadora II que lhe está ligada a jusante, se esta ainda tem tarefas de cálculo associadas, com nós por processar.
8. Recebe j_1^*, p_{j1}^* e d_{j1}^* da tarefa de cálculo com que partilha o transputer. Se $j_1^* = -1$, então a tarefa de cálculo já não tem nós para processar.
9. Recebe j_2^*, p_{j2}^* e d_{j2}^* da tarefa subcoordenadora II que lhe está ligada a jusante. Se $j_2^* = -1$, então as tarefas de cálculo associadas a esta tarefa subcoordenadora II já não têm nós para processar.
10. Se $j_1^* = j_2^* = -1$ então envia $j^* = -1$, $p_j^* = -1$ e $d_j^* = -1$ à tarefa subcoordenadora II que lhe está associada a montante ou à tarefa coordenadora, e pára.
11. $i \leftarrow \min\{i \mid (d_i = \min\{d_{jk}^* \mid (k \in \{1,2\} \wedge (j_1^* \neq -1, j_2^* \neq -1)) \vee (k = 1, j_1^* \neq -1, j_2^* = -1) \vee (k = 2, j_1^* = -1, j_2^* \neq -1))\}$
12. Envia i, p_i e d_i, à tarefa subcoordenadora II que lhe está associada a montante ou à tarefa coordenadora.
13. Volta ao passo 6.

Algoritmo da tarefa subcoordenadora II:

Esta tarefa é idêntica à tarefa subcoordenadora I, daí que a sua descrição detalhada não seja apresentada. A grande diferença entre estas duas tarefas reside no número de tarefas de cálculo com que lidam directamente, isto é, a tarefa subcoordenadora II, tem que escolher entre quatro valores de d_j^* a saber: um da tarefa de cálculo com que partilha o transputer e os outros três enviados pelas tarefas subcoordenadoras I que lhe ficam a jusante.

4. Paralelização algorítmica

4.1. Considerações genéricas

Os exemplos da secção 3 ilustram as duas técnicas diferentes de programação em paralelo disponíveis no ambiente computacional descrito na secção 2. A técnica utilizada no exemplo da secção 3.1. (RCI), sendo mais simples de utilizar, apresenta algumas desvantagens face à que

se utiliza na secção 3.2. O facto do encaminhamento dos pacotes, de e para a tarefa coordenadora ser da responsabilidade do programa de gestão da rede (não tendo portanto o programador que se preocupar com o assunto), aumenta o tempo de comunicação e diminui o espaço de memória dinâmica existente em cada processador da rede, destinado ao código e dados do algoritmo que se pretende implementar. Por outro lado, utilizando esta técnica é geralmente difícil (é possível utilizando aluns truques de programação, que podem falhar nalgumas circunstâncias) enviar a partir da tarefa coordenadora a mesma mensagem a todas as tarefas de cálculo ("broadcasting"), e ter a certeza que todas elas a receberam. A técnica RCI não permite ainda, por concepção, que as tarefas de cálculo comuniquem directamente entre si, nem permite que a coordenadora envie uma determinada mensagem a uma tarefa de cálculo em particular: quando a coordenadora põe uma mensagem na rede não sabe qual das tarefeiras a irá receber. Todas estas características limitam a aplicação desta técnica de processamento paralelo a uma pequena gama dos algoritmos existentes em optimização. (Existem outras áreas mais adequadas a este tipo de técnica, nomeadamente alguns algoritmos de processamento de imagem). O algoritmo do caminho mais curto, descrito na secção 3.2., por exemplo, não poderia ter sido implementado com a mesma generalidade, utilizando esta técnica.

A técnica de processamento paralelo que se ilustrou na secção 3.2, e que denominaremos por genérica, se não possui as limitações indicadas no parágrafo anterior, possui no entanto, a grande desvantagem de implicar a reconfiguração ou mesmo a recompilação das aplicações, quando o número de transputers ou a topologia da rede variam. Veja-se a versão 2 do exemplo da secção 3.2, onde, devido aumento do número de transputers da rede, seria necessário recodificar a aplicação.

Existem também algumas limitações na utilização dos recursos (disco duro, "écran", teclado e periféricos) do microcomputador que serve de hospedeiro à placa de transputers. Estas limitações derivam do facto da rede de transputers ter apenas um dos seus nós (designado raíz) ligado directamente ao BUS do microcomputador hospedeiro, ver fig. 1. Isto implica que todos os transputers da rede se vejam obrigados a comunicar com os recursos do hospedeiro, através da raíz. Se esta comunicação fôr intensa, pode gerar-se um engarrafamento no elo de ligação entre a raíz e o BUS hospedeiro, podendo o somatório da velocidade de processamento de todos os nós da rede, ser reduzido à velocidade daquele elo de ligação. Por exemplo a velocidade de processamento de um método de IO que utilize intensamente a técnica de visualizar gráficos interactivos ("Visual Interactive Graphics") pode reduzir-se à velocidade de actualização do écran do hospedeiro, ou a velocidade de um método baseado em perguntas ("Query") a uma base de dados (sistema pericial, por exemplo), pode reduzir-se à velocidade de transferência de dados de um disco duro.

4.2. Metodologia

A metodologia de programação em paralelo pode definir-se como a arte de encontrar o melhor compromisso entre o tempo de processamento das diversas tarefas que compõem uma aplicação e o tempo gasto por essas tarefas a comunicar entre si, por forma a obter o menor tempo possível de execução do programa considerado como um todo. Existem diversas abordagens mais ou menos estruturadas, das quais se pode destacar a devida a Kruatrachue e Lewis [10], mas nenhuma delas garante a melhor partição de um problema genérico, em grãos a executar em paralelo, nem a melhor distribuição destes grãos de código pelos processadores de uma rede.

A obtenção de sucesso na realização destas tarefas continua a estar fortemente dependente do bom senso e experiência dos programadores em paralelo. Se o grão de processamento é demasiado grande, o paralelismo é limitado, se este é demasiado pequeno, o atraso nas comunicações reduz a eficiência do sistema. Grãos pequenos obtêm-se maximizando o paralelismo e grãos grandes minimizando as comunicações.

Poderia pensar-se que o melhor programa em paralelo seria aquele que mantém todos os processadores o mais ocupados possível, iniciando-se cada tarefa logo possível. No entanto esta abordagem pode levar a maus resultados se o atraso nas comunicações fôr elevado. Por exemplo, na fig.6 se o tempo de comunicação entre a tarefa 1 e a tarefa 3 fôr maior que o tempo de execução da tarefa 2, o melhor programa é aquele em que as tarefas são todas executadas num só processador. Repare-se que na fig. 6.b) e c) a tarefa 2 só se inicia depois de ter decorrido o tempo de comunicação entre a tarefa 1 e 3 . Apesar de um transputer permitir continuar o processamento de tarefas, ao mesmo tempo que há transferência de dados nos elos de ligação (veja-se a secção 2.1), o facto de os dados a transmitir se dividirem em pacotes e as comunicações serem sincronizadas, implica que o processador 1 só fique livre das tarefas de transmissão, após a última mensagem. No entanto, a tarefa 2 pode ser iniciada em concorrência com as tarefas de comunicação, partilhando o processador. De qualquer modo, como no caso em exemplo isso não influenciaria o tempo total de execução (está dependente do fim da tarefa 3), preferiu-se simplificar, não considerando a hipótese da tarefa 2 ser lançada em concorrência.

Por outro lado, tal como a fig. 6 d) ilustra, a repetição do mesmo cálculo em diversos processadores pode trazer benefícios importantes.

Seguidamente apresenta-se uma abordagem semi-estruturada ao projecto de uma aplicação em paralelo.

1º *Sub-dividir o problema a resolver no maior número de tarefas que possam ser executadas em paralelo.* Embora muitos dos métodos mais divulgados de resolução de problemas de optimização aparente possuir poucas tarefas que possam executar em paralelo, pode encontrar-se na maioria deles características paralelas. O problema é que estes métodos foram estudados para máquinas sequenciais, encobrindo-se no processo de optimização da eficiência sequencial, algumas das suas características fundamentais.

Geralmente obtêm-se melhores resultados, nesta fase, se análise do problema for conduzida desde o início, reconstruindo-se o edificio teórico que serve de suporte ao método, pois assim é possível reencontrar algumas das suas características fundamentais. Não poucas vezes se conseguem obter melhores algoritmos paralelos a partir de algoritmos de pior eficiência sequencial quando comparados com outros também sequenciais.

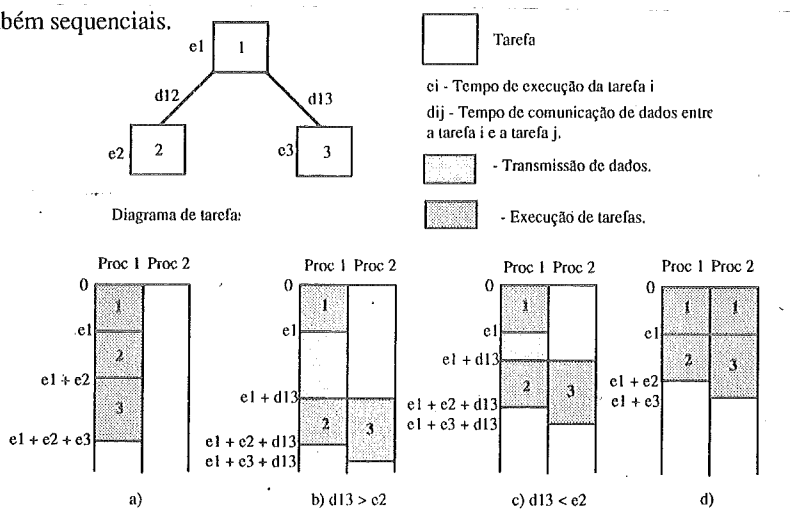


Figura 6. Exemplo de localização de tarefas em dois processadores

2º *Encontrar o diagrama de tarefas que torne evidente a ordem de execução das tarefas e quais as que podem executar em paralelo.* Este ponto não é difícil, trata-se apenas de estruturar sobre a forma de um diagrama o ponto 1.

3º *Determinar os tempos de execução de cada tarefa e os diferentes tempos de comunicação entre tarefas.* Se na maior parte dos casos o cálculo (ou estimativa) dos tempos de execução não causa problemas de maior (embora possa ser dependente dos dados a processar, que à partida poderão ser desconhecidos), o mesmo não se poderá dizer dos tempos de comunicação entre tarefas, pois eles dependem de alguns factores não facilmente quantificáveis. Os tempos de comunicação entre tarefas dependem linearmente da quantidade de dados a transferir, da localização das tarefas na rede de processamento e do ambiente computacional em termos de protocolos de comunicação e de hardware. Os dados são transmitidos em pacotes com um tamanho máximo determinado pelo protocolo e pela memória disponível na rede para os armazenar. No exemplo da secção 3.2., os dados que se transmitem em cada iteração do algoritmo formam apenas um pacote de dados, mas os que se transmitem inicialmente têm que ser divididos em diversos pacotes. O tempo de transmissão de um pacote, de um transputer para outro, não é constante, havendo que contar com repetições da transmissão devidas a possíveis erros: compensa normalmente dividir os dados a enviar, em pacotes bem mais pequenos do que o seu tamanho máximo. Repare-se que,

no exemplo da secção 3.2., o tempo de execução das tarefas de cálculo não é constante, mas depende do número de nós que a tarefa ainda tem que analizar, pelo que os ganhos de eficiência obtidos da paralelização efectuada, só são efectivos a partir de uma determinada dimensão da rede (Tal como se indica na secção 1, só há ganhos de eficiência na paralelização dum algoritmo, a partir de uma determinada dimensão dos problemas, estando esta dimensão intimamente ligada ao número de processadores da rede.) e na parte final da análise da rede não existe qualquer ganho de performance. Note-se também que a eficiência do algoritmo depende do ponto 2. do algoritmo da tarefa coordenadora, isto é, depende da distribuição dos nós pelas tarefas de cálculo (e da topologia da rede em análise), pois pode acontecer que uma das tarefas de cálculo termine (ficando um transputer sem funcionar) enquanto as outras têm ainda por processar, todos os nós que lhe foram atribuídos.

4º *Encontrar a melhor localização das diferentes tarefas do problema, na rede de processamento, cumprindo o diagrama de tarefas.* Deve ser considerada a hipótese de repetir a execução das mesmas tarefas em processadores diferentes e a de aglomerar diversas tarefas numa só sequencial. Este ponto está intimamente ligado ao anterior pois o tempo de transmissão de dados depende da localização das tarefas, podendo ser necessário realizar ambos ao mesmo tempo.

5. Algumas características dos algoritmos para problemas de optimização utilizando processamento paralelo em transputers

Nesta secção apresentar-se-ão algumas características que indiciam a obtenção de bons resultados na aplicação das técnicas de processamento paralelo disponíveis no ambiente computacional apresentado na secção 2. Não se pretende exaustividade, mas apenas indicar pistas para a realização do primeiro ponto do método apresentado na secção anterior. Não é necessário que um algoritmo possua todas as características que se indicam e pode até acontecer encontrarem-se alguns algoritmos potencialmente paralelizáveis, sem possuirem nenhuma dessas características, no entanto, como veremos, pode dizer-se que, regra geral, os algoritmos dedicados a problemas combinatórios são especialmente vocacionados ao uso das técnicas de processamento disponíveis no ambiente computacional utilizado.

Pode-se considerar a possibilidade de modelar o problema que se pretende abordar, segundo uma árvore de pesquisa ou de decisão como a primeira das características indicadoras da obtenção de bons resultados. Sempre que a determinação do óptimo implique a pesquisa em profundidade de diversos ramos de uma árvore, é natural que se possam explorar técnicas de processamento paralelo com vista a acelerar o processo de pesquisa. O algoritmo para determinar todas as soluções básicas óptimas alternativas de um PPL, que se apresenta na secção 3.1., é modelável segundo uma árvore de pesquisa de todas as bases óptimas alternativas. Outro exemplo, mas para árvores de decisão, pode ser encontrado em [11].

Também os algoritmos de optimização onde é necessário realizar uma qualquer operação (ou operações) sobre um conjunto de dados referenciados por um índice (dados de natureza idêntica apenas distínguiveis entre si por um índice que os associa a uma entidade distinta), podendo iniciar-se essa operação por qualquer um dos índices, sem que exista uma ordem pré-estabelecida, indiciam bons resultados da sua paralelização. Está neste caso, por exemplo, o algoritmo de Ford e Fulkerson [8], para determinar o fluxo máximo entre dois nós de uma rede orientada. Como se sabe, este algoritmo não especifica nem a ordem pela qual os nós são rotulados, nem a ordem pela qual são pesquisados os nós rotulados.

Outra característica é a ilustrada pelo exemplo da secção 3.2, no algoritmo de cálculo do caminho mais curto de um nó a todos os outros nós de uma rede. Durante o cálculo, todos os nós da rede vêm a ser considerados origem (ou raíz), assim de iteração para iteração temos apenas uma alteração (a nova raíz corrente), que implica o reajustamento das etiquetas associadas aos nós da rede que ainda não foram dados como raíz. Pode dizer-se então que se uma única alteração implicar o reajustamento directo de uma determinada estrutura de dados de grandes dimensões, podem presumir-se bons resultados na aplicação de técnicas paralelas. Semelhante a esta situação é a que encontramos quando pretendemos escolher um máximo ou um mínimo de entre uma grande quantidade de dados (podemos ter também algumas operações associadas a esta escolha).

Por último, aponta-se uma característica óbvia: se o problema que se pretende resolver possui diversos sub-problemas, então é certamente vantajoso resolvê-los em paralelo. Um exemplo deste caso é o método de Simplex generalizado, dedicado a PPLs [3]. A ideia base, consiste em garantir que as soluções admissíveis, que se vão calculando durante a aplicação do método, pertençam ao interior relativo de faces de dimensão arbitrária, sendo os deslocamentos em direcção ao óptimo realizados através da faces de dimensão imediatamente inferior àquela em que se encontra a solução corrente. O algoritmo correspondente a este método inclui a resolução de sub-problemas independentes que, portanto podem ser resolvidos em paralelo.

Os autores agradecem a colaboração prestada por Helena Maria Rodrigues, na implementação em paralelo do algoritmo do caminho mais curto de um nó a todos os outros de uma rede, que se apresenta na secção 3.2.

Bibliografia

[1] Antunes, Carlos Henggeler, Maria João Alves, Ana Luísa Silva e João N. Clímaco, An integrated MOLP method base package - A guided tour of TOMMIX, Computers and Operations Research 19 (1992) 609-625.

[2] Behning, Rochell, Ralph M. Butler, Billy E. Gillett, A parallel integer linear programming algorithm, EJOR 34, 393-398.

[3] Cardoso, Domingos, João Clímaco, A generalized Simplex method, Operations Research Letters 12 (1992).

[4] Clímaco, João N., João Paulo Costa, Carlos Henggeler Antunes, Maria J. Alves, Parallel processing in MOLP method base development - A discussion using two case studies, Submetido para publicação.

[5] Duncan, Ralph, A survey of parallel computer architectures, IEEE Computer Magazine 5-16, 1990.

[6] Glover, F., D.Klingman, N.Philips, A new polynomially bounded shortest path algorithm, Operations Research 33 (1985) 65-73.
[7] Glover, F., D.Klingman, N.Philips, R.Schneider, New polynomial shortest path algorithms and their computational attributes, Management Science 31 (1985) 1106-1128.
[8] Hu, T.C., Combinatorial algorithms, Addison-Wesley Publishing Company, 1982.
[9] Kindervater, G.A.P., H.W.J.M.Trienekens, Experiments with parallel algorithms for combinatorial problems, EJOR 33 (1988) 65-81.
[10] Kruatrachue, Boontee, Ted Lewis, Grain size determination for parallel processing, IEEE Software Magazine, 23-32, 1988.
[11] Moura-Pires, Fernando, Luís Gomes, Adolfo Steiger-Garção, Uma implementação paralela de algoritmos de indução de árvores de decisão, 3ª Jornadas Nacionais de Projecto, Planeamento e Produção assistidas por computador.
[12] Parallel C, User guide, 3L Id, 1988.
[13] Sherali, Hanif D., On the equivalence between some shortest path algorithms, Operation Research Letters 10 (1991) 61-65.
[14] Steuer, R.E., Multiple criteria optimization: Theory, computation and application, Wiley, 1986.
[15] The transputer data book, INMOS 1989.
[16] Trippi, R., Efrain Turban, Parallel processing and OR/MS, Computers Ops.Res. 18 (1991) 199-210.

PLANIFICAÇÃO VISUAL E INTERACTIVA DE REUNIÕES – PlaVIR

Ana Isabel Botto de Barros
Joaquim António dos Santos Gromicho
DEIO - Faculdade de Ciências
Universidade de Lisboa

Abstract

In schools where the classes are given by several teachers the scheduling of regular teachers' meetings without overlapping can be a difficult task. Therefore a Decision Support System was developed to help this planning.

The system is very flexible and user-friendly. Starting with an efficient and fast graph colouring heuristic solution it allows the user to make all the local adjustments that he wishes.

The system also suggests to the user good ways of changing the current solution according to his own criteria but leaving the final decision to him.

The system has been tested in several Portuguese high schools with success due mainly to the fast and absolutely error free way of finding good schedules.

Resumo

O problema tratado neste trabalho consiste na marcação de reuniões simultâneas de pessoas com sobreposição de interesses, aplicado ao caso concreto das reuniões de turma em Escolas do Ensino Preparatório e Secundário.

Foi desenvolvido um Sistema integrado de Apoio à Decisão suficientemente flexível para poder comportar o tratamento de incompatibilidades adicionais.

O sistema é bastante amigável, fornecendo ao utilizador uma proposta inicial baseada numa solução heurística e permitindo-lhe fazer todos os ajustamentos locais considerados convenientes.

Palavras-chave

Sistema de Apoio à Decisão; *Interface*; Coloração de Grafos; Clique; Heurística; Planificação.

1. Introdução

De um modo geral nas Escolas Preparatórias e Secundárias os alunos encontram-se distribuídos por turmas sendo as várias disciplinas leccionadas por diferentes professores. Com este regime de ensino, é necessário efectuar periodicamente reuniões com todos os professores de cada turma de modo a permitir o conhecimento do comportamento e aproveitamento dos alunos em cada disciplina e, assim, contribuir para a sua avaliação.

Para tal, cada Escola precisa de estabelecer um plano de marcação de reuniões por forma a que não possam ser convocadas para a mesma altura reuniões de turma que partilhem algum professor. A Escola pretende ainda que esta planificação não só ocupe o menor tempo possível

mas também que, respeitando as directivas de Escola, permita uma "boa" distribuição das reuniões ao longo dos dias.

Cada Escola delega a responsabilidade da elaboração da planificação de reuniões em um ou dois professores. Em resultado de uma pesquisa realizada em algumas Escolas podemos concluir que o processo seguido é, em geral, manual e bastante moroso.

Pareceu, pois, adequado o desenvolvimento de um "suporte" informático que, de alguma forma, tivesse em atenção estes aspectos, e simultaneamente contribuisse para a obtenção de boas soluções.

Assim, o sistema PlaVIR é o resultado da integração dum processo de planificação num Sistema de Apoio à Decisão por forma a facilitar a elaboração da planificação de reuniões recorrendo à experiência do professor responsável.

PlaVIR já foi utilizado em Escolas da área da grande Lisboa com uma grande aceitação devido à qualidade dos resultados e, em especial, à facilidade de manuseamento e à rapidez com que se obtêm "boas" planificações.

2. Descrição do problema

A planificação das reuniões para uma Escola tem que ser feita por forma a que cada professor assista a todas as reuniões das turmas que lecciona. Esta restrição embora não pareça à primeira vista muito díficil de satisfazer revela-se na prática bastante problemática devido à grande quantidade de informação a considerar.

Na prática existem ainda outros aspectos que dificultam a elaboração da planificação como o facto de o número de professores por turma ser variável (dos 3 aos 15) e existirem professores, como por exemplo os de Educação Visual, que leccionam um grande número de turmas.

Para a planificação é necessário ter ainda em conta o tipo de reunião em vista: Reuniões de Conselho de Turma que são convocadas no fim de cada período escolar para efeitos de avaliação e Reuniões Intercalares convocadas a meio do período escolar por motivos diversos.

A Escola pretende que o tempo total dispendido nas reuniões não simultâneas seja o mínimo indispensável e ainda que a marcação das reuniões ao longo dos dias tenha em atenção situações especiais como por exemplo limitações de disponibilidade horária de alguns professores.

A planificação feita manualmente subdivide-se em duas fases:

1ª obtenção de uma configuração com o menor número possível de blocos ou conjuntos de turmas compatíveis (sem professores comuns);

2ª distribuição dos blocos encontrados ao longo dos dias de reuniões.

Entre os problemas que surgem neste tipo de planificação salientam-se:

1º dificuldade do manuseamento da informação devido ao número de professores por turma;

2º muito tempo dispendido (gasta-se no mínimo um dia completo de trabalho, em média uma semana);

3º possibilidade de cometer erros (marcação de reuniões incompatíveis para a mesma altura);

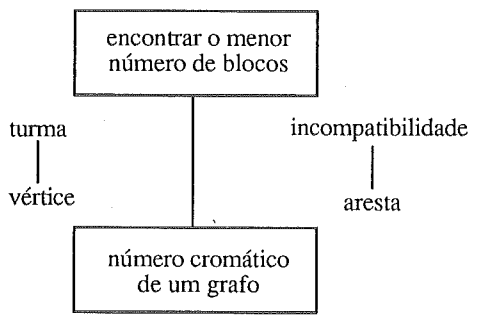
4º dificuldade em avaliar a qualidade da planificação;

5º dificuldade em atender a situações especiais (na fase de distribuição dos blocos).

3. Modelação

Facilmente se verifica que este problema tem uma componente bem estruturada. De facto, o principal objectivo é obter o menor número possível de blocos de reuniões simultâneas. As restrições impedem que sejam consideradas como simultâneas reuniões a que devam assistir o mesmo professor. É a primeira fase do problema.

Considerando cada turma como o vértice de um grafo cujas arestas ligam turmas com algum professor em comum surge um problema de Coloração de Grafos ([1], [2], [3] e [6]). Os vértices coloridos com a mesma cor são afinal as turmas que se podem reunir simultaneamente.



Este problema é NP-Completo ([1], [2], [3] e [6]) e é considerado como um dos mais difíceis entre os problemas de Optimização Discreta sendo o problema de encontrar uma coloração perto da óptima também NP-Completo ([6]).

Devido às dimensões dos problemas colocados e aos tempos de execução reportados como necessários para a sua resolução exacta ([1], [2] e [6]) consideraram-se apenas heurísticas.

Até ao ínicio deste projecto a segunda fase merecia pouca atenção, já que a planificação era feita manualmente. Assim não se encontraram critérios bem definidos no sentido de caracterizar a "melhor" distribuição dos blocos de reuniões ao longo dos dias.

Foram apontados alguns critérios, geralmente conflituosos, que pareceriam conduzir a uma abordagem multicritério. Por exemplo em alguns casos, poder-se-ia pretender que os professores tivessem as suas reuniões no mesmo dia ou em dias consecutivos ou ainda que para os directores de turma existissem intervalos entre as suas reuniões, etc. Existem Escolas em que se consideram em primeiro lugar os professores com cargos especiais (por exemplo pertencentes ao Conselho Directivo) e colocam-se as respectivas reuniões nos dias da sua permanência obrigatória na Escola.

Contudo e tendo em vista uma solução de compromisso optou-se, por sugestão dos professores envolvidos no projecto, por propôr uma solução inicial baseada numa regra simples e permitir uma total maleabilidade no posterior ajustamento dessa solução por forma a atender as situações específicas de cada planificação.

A regra adoptada consiste em ordenar os blocos de reuniões simultâneas obtidas na primeira fase por ordem decrescente do número de turmas que os constituem. Assim esta solução, ao concentrar um grande número de reuniões nos primeiros dias, procura fazer com que um grande número de professores tenha a sua última reunião o mais cedo possível.

Dadas as características deste problema integrou-se a planificação das reuniões num Sistema de Apoio à Decisão ([4], [5] e [7]) por forma a permitir aos professores responsáveis obter as planificações mais ajustadas às necessidades de cada Escola.

4. PlaVIR

PlaVIR é um sistema interactivo, desenvolvido para micro-computadores, cujo *interface* com o professor/utilizador foi particularmente cuidado, recorrendo ao *mouse* e a *écrans* coloridos e animados, por forma a lhe assegurar uma fácil e completa visualização de cada etapa da planificação em curso.

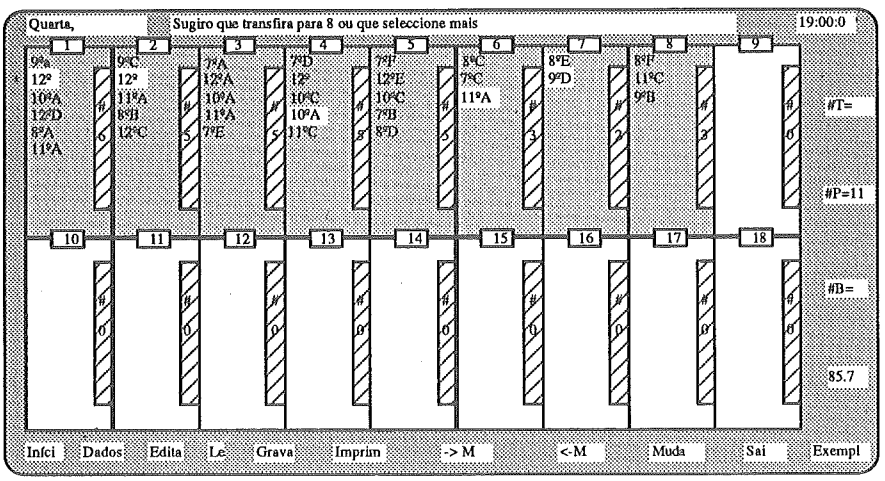
PlaVIR procura seguir o processo manual de planificação fornecendo contudo sugestões e indicações baseadas em critérios de Optimização e tirando partido da experiência prévia do professor/utilizador.

A partir de um dado jogo de dados PlaVIR começa por propor uma solução dada por uma heurística de pesquisa sequencial baseada em conjuntos independentes (conjuntos de turmas mutuamente compatíveis).

Esta heurística, descrita em [6], é referenciada como a que tem melhor pior caso. Tal característica é bastante importante já que as dimensões e tipo dos problemas (número de turmas e incompatibilidades) a resolver variam muito de Escola para Escola e se pretende um comportamento estável por parte do programa.

Tendo em conta a experiência do professor/utilizador possibilitam-se trocas de turmas entre os blocos por forma a tentar melhorar a solução corrente. Para auxiliar o processo de transferência existem diversas informações como por exemplo:

- Se ainda existem turmas compatíveis, e quais, com o conjunto de turmas seleccionadas;
- Se a turma ou turmas seleccionadas cabem em algum bloco;
- Se existir mais que um bloco capaz de receber as turmas seleccionadas o sistema sugere um deles de acordo com um procedimento heurístico de trocas locais ([1] e [6]).



PlaVIR afere a qualidade da solução corrente comparando o seu número de blocos com o maior número de turmas que partilham o mesmo professor. Esse número de turmas é uma aproximação para o maior número de turmas incompatíveis, número de clique ([3]), que é um limite inferior para o menor número de blocos, número cromático ([3]).

Verificou-se que, devido às características especiais deste problema, esta aproximação é bastante razoável sendo mesmo nalguns casos igual ao número mínimo de blocos de reuniões possível. Tal parece dever-se ao facto de existirem professores (por exemplo de Educação Visual) que leccionando um grande número de turmas, de diferentes anos, condicionam bastante a planificação.

O sistema PlaVIR permite ainda a memorização de configurações intermédias, às quais se pode sempre retornar.

Se a dimensão dos dados a processar não permite que estes sejam visiveis na sua totalidade PlaVIR permite o *scroll* lateral dos blocos no *écran* e dentro de cada bloco o *scroll* vertical da lista de turmas que o constituem, servindo o *mouse* para permitir o control total destas operações.

Logo que o professor/utilizador considere que a configuração corrente é adequada passa-se à distribuição dos blocos ao longo dos dias. É de realçar que a configuração considerada "boa" pode ser guardada por forma a ser retomada posteriormente, por exemplo quando for necessário proceder à distribuição das reuniões ao longo dos dias de outra forma.

PlaVIR propõe, a partir da configuração dos blocos já construída, uma distribuição das reuniões ao longo dos dias com base num conjunto de parâmetros estabelecidos pelo professor/utilizador (data de início das reuniões, hora de início, duração das reuniões, número máximo de reuniões por dia), e obedecendo ao critério de ordenação referido.

Pode-se proceder a todo o tipo de ajustes a esta solução inicial, por forma a atender a situações particulares, pois o sistema permite, recorrendo à utilização do *mouse*, a troca de

blocos ao longo do dia e a tranferência de blocos para outros dias. Estes ajustes são facilitados pelo facto de se conhecer, em qualquer momento, as turmas que cada professor lecciona (que permanecem assinaladas enquanto se fazem ajustamentos e transfências) e também para cada turma os respectivos professores.

Os jogos de dados de que o sistema PlaVIR se socorre são os já compilados nas folhas de serviço da Escola. O processo de edição e alteração está integrado no sistema recorrendo a um mini-editor de texto muito simples e de fácil manuseamento.

A planificação das reuniões fica completamente estabelecida com a impressão das tabelas de reuniões com os respectivos horários, cujo aspecto é o seguinte:

Sexta-feira, 14/12/1990

9:00	9ºA	12ºF	10ºA3	12ºD	8ºA	11ºA1
11:00	7ºA	12ºA	10ºA2	11Aº3	7ºE	
14:30	9ºC	12ºB	11ºA4	8ºB	12ºC	

Sábado, 15/12/1990

8:30	8ºF	11ºC	9ºB
10:30	8ºE	9ºD	

Segunda-feira, 17/12/1990

8:00	7ºD	12ºC	10ºC3	10ºA1	11ºC2
10:00	7ºF	12ºE	10ºC1	7ºB	8ºD
14:00	8ºC	7ºC	11ºA2		

5. Conclusões

Nos contactos estabelecidos com professores responsáveis pela planificação as dificuldades mais referidas eram o tempo dispendido na obtenção de uma única configuração, a possibilidade de cometer erros e a dificuldade de compreensão dos dados na sua globalidade.

PlaVIR foi construído tendo em conta não só estes factores mas também o facto de que seria utilizado por professores sem grande experiência e contacto com computadores. Assim, e devido ao aspecto colorido, amigável e ao recurso ao *mouse* aquele problema foi rapidamente ultrapassado.

PlaVIR permite que a elaboração da planificação tome menos tempo e que recorrendo à experiência do professor/utilizador origine planificações mais ajustadas às necessidades de cada Escola.

PlaVIR foi desenvolvido tendo em vista a sua aplicação a qualquer Escola, não se tendo verificado qualquer problema na sua aplicação nas Escolas contactadas.

PlaVIR é considerado pelas Escolas contactadas como uma ferramenta de trabalho adequada às necessidades actuais, no que respeita às possibilidades oferecidas e às dimensões suportadas.

Agradecimentos

Agradece-se toda a colaboração prestada pelos Conselhos Directivos das Escolas Secundárias D. Filipa de Lencastre e Rainha D. Leonor e Escola C+S de Camarate em particular pelos responsáveis pela planificação das reuniões, bem como pelo Departamento de Estatística e Investigação Operacional da Faculdade de Ciências de Lisboa.

6. Referências

[1] Brélaz, D., *New Methods to Color the Vertices of a Graph*, Management Science 22-4 (1979).
[2] Christofides, N., Mingozzi, A., Toth, P., Sandi, C., *Combinatorial Optimization*, John Wiley & Sons (1979).
[3] Gondran, M., Minoux, M., Vajda, S., *Graphs and Algorithms*, John Wiley & Sons, (1984).
[4] Speranza, M.G., Woerlee, A.P., *A Decision Support System for operational production scheduling*, European Journal of Operational Research 55-3 (1991).
[5] Sprague, R.H.Jr., Watson, H.J., *Decision Support Systems - Putting Theory into Practice*, Englewood Cliffs, NewJersey, Prentice-Hall, INC. (1986).
[6] Syslo, M., Deo., N., Kowalik, M., *Discrete Optimization Algorithms - with Pascal Programs*, Englewood Cliffs, NewJersey, Prentice-Hall, INC. (1983).
[7] Turban, E., Lucas, H.C.Jr., *Decision Support Systems and Expert Systems - Managerial Perspectives*, New York University, MacMillan Publishing Company (1988).

INVESTIGAÇÃO OPERACIONAL, vol. 13, n.° 2 (Dezembro 1993) 177-188

FORECASTING WITH FRACTIONALLY DIFFERENCED ARIMA MODELS

Nuno Crato
Department of Pure and Applied Mathematics,
Stevens Institute of Technology, Hoboken, NJ 07030, USA

Abstract

Fractional differenced ARMA models (FARMA or ARFIMA) were introduced by Granger and Jyeux (1980) and Hosking (1981). They provide a discrete-time approach to the analysis of time series that present long-range dependence. Previous applications of FARMA models to the analysis of macroeconomic time series provided some indications that these models could also be useful on long-range forecasting.

In this paper we discuss some theoretical and computational issues in FARMA forecasting. We also critically review the existent empirical work and present the results of a new application.

Resumo

Os modelos ARMA com diferenciação fraccional (FARMA ou ARFIMA) foram introduzidos por Granger and Joyeaux (1980) e por Hosking (1981). Esses modelos fornecem um instrumento em tempo discreto para a análise de sucessões cronológicas que apresentem uma memória de tipo longo. Aplicações de modelos FARMA a diversas séries macroeconómicas sugerem a sua utilidade para a previsão de longo prazo.

Neste artigo discutem-se várias questões relacionadas com a previsão de processos FARMA. A experiência relatada na literatura é revista criticamente e são apresentados os resultados de uma nova aplicação.

Keywords

ARFIMA, CPI, prediction, time-series.

1. Introduction

The time series and the econometric literature have recently shown an increased interest in the so-called long-memory and persistent processes. A particular appealing long-memory model is the fractional differenced ARMA, which has been introduced by Granger and Joyeux (1980) and by Hosking (1981) and has been applied in various types of econometric analysis[1].

Forecasting is an important problem in time series analysis, and we should investigate about how these fractional models compare with the other procedures used in practical economic forecasting.

[1] See, e.g., Crato (1991) for a review and Crato (1993) for a recent application. A generalization of the fractional ARMA model has been developed by Gonçalves (1987).

Few empirical results are yet available, and they come mainly from the researchers who pioneered the use of fractional differenced models. Moreover, as we will see, some objections can be raised to their conclusions.

In this paper we present a short review of the state of the art in forecasting with fractional ARMA models. We discuss a new perspective on the forecasting formulae, critically review existent empirical results, and present a new application.

The plan of the paper is as follows. Section 2 introduces some preliminary definitions and explains the notation. Section 3 presents new simplified proofs of the essential forecasting formulae, shows an interesting decomposition formula, and develops computational results for some cases. Section 4 critically reviews the empirical results set forward in the literature. Section 5 presents a new application. Section 6 concludes.

2. Preliminary Definitions and Notation

In order to introduce fractional ARMA models we will start setting up the notation.

A time series will be represented by (X_t). The unit backward shift operator B is defined by $BX_t := X_{t-1}$, and the first difference operator at lag 1, ∇, is defined by $\nabla X_t := X_t - X_{t-1} = (1-B)X_t$. A process (ε_t) will be called *white noise* with mean zero and variance $\sigma^2 < \infty$, written $\varepsilon_t \sim WN(0, \sigma^2)$, if (i) $E\varepsilon_t = 0\ \forall t$, and (ii) the autocovariances $\gamma_\varepsilon(k) = 0$ for all $k \neq 0$ and $\gamma_\varepsilon(0) = \sigma^2 > 0$.

An ARMA(p, q) process (X_t), defined by

$$X_t - \emptyset_1 X_{t-1} - \ldots - \emptyset_p X_{t-p} = \varepsilon_t + \theta_1\varepsilon_{t-1} + \ldots + \theta_q\varepsilon_{t-q}, \qquad (1)$$

where $\varepsilon_t \sim WN(0, \sigma_\varepsilon^2)$, will be written

$$\emptyset(B)X_t = \theta(B)\varepsilon_t, \qquad (2)$$

with the autoregressive $\emptyset(z) := 1 - \emptyset_1 z - \ldots - \emptyset_p z^p$ and the moving-average polynomials $\theta(z) := 1 + \theta_1 z + \ldots + \theta_q z^q$. It will be assumed that the roots of $\emptyset(z)$ and $\theta(z)$ are outside the unit circle, i.e., that the ARMA is causal and invertible (see, e.g., Brockwell and Davis 1991). It is well-known that, if (X_t) follows an ARMA(p,q) process, then its autocorrelation function $\rho(h)$ is geometrically bounded, i.e., there exist constants $C > 0$ and r in the open interval $]0, 1[$ such that $|\rho(h)| \leq Cr^{|h|}$. Hence, $(\rho(h))_h$ is absolutely summable.

If $Y_t := \nabla^d X_t$ is a causal ARMA(p,q) process then (X_t) is said to be an ARIMA(p,d,q) process, $p,d,q \in N_0$.

The rate of decay of the autocorrelation function of a stationary process defines its memory characteristics. A second-order stationary process (X_t) would be called of long-memory type if, for some $C > 0$, and $\alpha < 0$, its autocorrelation function has the following asymptotic behaviour[2]

[2] The symbol $\sim$ is used in the standard way: $a_n \sim cb_n$ where $c \neq 0$ iff $a_n/b_n \to c$.

$$|\rho(h)| \sim C|h|^{\alpha} \text{ as } h \to \infty. \tag{3}$$

If, in addition, $\alpha > -1$ and so $\sum |\rho(h)| = \infty$, we will say that (X_t) is persistent.

An important example of long-memory and persistent process is given by the fractionally differenced noise, a model introduced independently by Granger and Joyeux (1980) and by Hosking (1981).

If d is any real number, then the fractional difference operator ∇^d will be defined as

$$\nabla^d = (1-B)^d := \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k, \tag{4}$$

where

$$\binom{d}{k} := \frac{d}{k}\,\frac{d-1}{k-1} \cdots \frac{d-k+1}{1}. \tag{5}$$

The process (X_t) is called a fractionally differenced noise or fractional noise, with $d \in]-.5,.5[$ if (X_t) is a stationary solution of the equation

$$\nabla^d X_t = \varepsilon_t, \text{ with } \varepsilon_t \sim WN(0, \sigma^2) \tag{6}$$

and with $\nabla^d X_t$ denoting the mean-square limit of the series.

Equation (6) provides an autoregressive representation of the process

$$\sum_{k=0}^{\infty} \pi_k X_{t-k} = \varepsilon_t, \tag{7}$$

with $\pi_k = \binom{d}{k} (-1)^k$. Explicitly

$$X_t - dX_{t-1} + \frac{d(d-1)}{2} X_{t-2} - \frac{d(d-1)(d-2)}{6} X_{t-3} + \ldots = \varepsilon_t.$$

By the law of exponents we have, formally, $X_t = \nabla^{-d}\varepsilon_t$. Then, if we define $\psi_k := \binom{d}{k} (-1)^k$ we would have the representation

$$X_t = \sum_{k=0}^{\infty} \psi_k \varepsilon_{t-k}. \tag{8}$$

Explicitly,

$$X_t = \varepsilon_t - (-d)\varepsilon_{t-1} + \frac{-d(-d-1)}{2} \varepsilon_{t-2} - \frac{-d(-d-1)(-d-2)}{6} \varepsilon_{t-3} + \ldots .$$

A more general model is the fractionally integrated ARMA model, i.e., an ARIMA where the order of integration is the non-integer $d \in]-.5,.5[$.

More precisely, a fractional autoregressive moving-average process FARMA(p,d,q) with $d \in]-.5,.5[$, is a stationary ARMA(p,q) driven by fractional noise, i.e., (X_t) is a FARMA(p,q,d) if it is a stationary solution of the difference equation

$$\emptyset(B)X_t = \theta(B)\xi_t, \ \xi_t := \nabla^{-d}\varepsilon_t, \ \varepsilon_t \sim WN(0, \sigma^2) \tag{9}$$

where $\emptyset(B)$ and $\theta(B)$ are lag polynomials of order p and q, respectively, with roots outside the unit circle.

A generalization which allows for nonstationarity is the fractional ARIMA: a fractionally integrated autoregressive moving-average process (X_t), ARFIMA(p,d,q) with $d \in]-.5, \infty[$, is a process such that $(\nabla^d X_t)$ is a stationary ARMA(p,q).

3. Forecasting Formulae

The general formulae for FARMA models have been studied by Peiris (1987) and by Peiris and Perera (1988). Their main result is contained in the following proposition for which we outline a direct and simple proof.

Proposition 1 Let (X_t) be a FARMA process satisfying the difference equations $\emptyset(B)\nabla^d X_t = \theta(B)\varepsilon_t$, with $\varepsilon_t \sim WN(0, \sigma^2)$, and with $d \in]-.5,.5[$. Let $\nabla^d X_{t+h|t} = \hat{E}[\nabla^d X_{t+h} | \nabla^d X_s, s \le t]$ represent the $\nabla^d X_{t+h}$ projection on the closure of the linear space generated by $\{\nabla^d X_s | s \le t\}$ and let $\hat{X}_{t+h|t} = \hat{E}[X_{t+h}|X_s, s \le t]$ represent the analogous predictor for X_{t+h}. Then, the predictors satisfy the following recursive relations

$$\hat{X}_{t+1|t} = \sum_{k=0}^{\infty} (-1)^k \binom{d}{k+1} X_{t-k} + \nabla^d X_{t+1|t}$$
$$= dX_t - \frac{d(d-1)}{2} X_{t-1} + \ldots + \nabla^d X_{t+1|t} \qquad (10)$$

$$\ldots$$
$$\sum_{k=0}^{h-1} \sum_{\ell=0}^{k} (-1)^\ell \binom{d}{\ell} \hat{X}_{t+h-k|t} = \sum_{k=0}^{\infty} \sum_{\ell=k+1}^{k+h} (-1)^{\ell-1} \binom{d}{\ell} X_{t-k}$$
$$+ \sum_{k=1}^{h} \nabla^d X_{t+k|t}. \qquad (11)$$

If we substitute the minimum mean square linear predictor $\hat{E}[\cdot|\cdot]$ by the minimum mean square predictor $E[\cdot|\cdot]$ the previous equations remain valid.

Proof. See Peiris and Perera (1988) for a discussion. The direct proof is obvious if we write

$$\hat{E}[\nabla^d X_{t+1}|X_s, s \le t] = \hat{E}[\nabla^d X_{t+1}|\nabla^d X_s, s \le t]$$
$$\hat{E}[X_{t+1} + \textstyle\sum_{k=1}^{\infty} (-1)^k \binom{d}{k} X_{t+1-k}|X_s, s \le t] = \nabla^d X_{t+1|t}$$
$$\hat{E}[X_{t+1}|X_s, s \le t] + \hat{E}[\textstyle\sum_{k=0}^{\infty} (-1)^{k+1} \binom{d}{k+1} X_{t-k}|X_s, s \le t] = \nabla^d X_{t+1|t}$$
$$\hat{X}_{t+1|t} = \textstyle\sum_{k=0}^{\infty} (-1)^k \binom{d}{k+1} X_{t-k} + \nabla^d X_{t+1|t}.$$

For $\hat{X}_{t+h|t}$, $h > 1$, the proof follows by induction replacing X_{t-k} by their predictors for $k \ge 1$. For the conditional expectation the proof is identical. |

Remarks. This inductive method of proof provides an easy algorithm for the computation of h-step-ahead forecasts. First, the series is transformed by the operator ∇^d. Then, an ARMA model is fit to $(\nabla^d X_t)$ and predictors for $\nabla^d X_{t+1}$, $\nabla^d X_{t+2}, \ldots, \nabla^d X_{t+h}$ are computed with Durbin-Levinson algorithm or any other similar procedure. Finally, predictors for X_{t+1}, $X_{t+2}, \ldots, X_{t+h}$ are computed recursively applying the operator ∇^d to the original series and adding the ARMA forecasts–wherever values for X_t are not available they are replaced by their estimates previously computed. In practice we apply the operator ∇^d, $k = 1, 2, \ldots, h$, truncated for the available observations. The whole procedure is easy to visualize in (10). To see that (11) is just the recursive application of (10) note that the double sum $\sum_k \sum_\ell$ collapses to the sum ∇^d.

Now, let $\pi_k := \binom{d}{k}(-1)^k$, $k = 0, 1, \ldots$, be the coefficients of the operator ∇^d. The expression (10) can be rewritten

$$\hat{X}_{t+1|t} = \widehat{\nabla^d X}_{t+1|t} - \sum_{k=1}^{\infty} \pi_k X_{t-k+1} \,. \tag{12}$$

This decomposition formula can be applied recursively, replacing the estimates by the observations when latter are no longer available. But this formulation is also very interesting, for it decomposes the predictor in a sum of two variables: the ARMA predictor for the fractionally differenced process ($\nabla^d X_t$) and an infinite moving average of past observations of the original process (X_t). In other words, the predictor of the process can be thought of as the sum of the short-term ARMA predictor for a transformation of the observations and of a smoothed version of the whole past of the process. On the short run, the ARMA part is likely to predominate, but on the long run, it will converge rapidly to the average of the process, while the infinite moving average is likely to converge much slower. It is also interesting to note that the infinite moving average $\sum_k \pi_k X_{t-k+1}$ is independent of the existence of short-memory ARMA components.

Now, let $\emptyset_0 (\equiv 1), \emptyset_1, \ldots, \emptyset_p$ be the coefficients of the autoregressive polynomial $\emptyset(B)$. If $q = 0$, i.e., if the moving average part of the FARMA process does not exist, then the forecasting procedure can be greatly simplified since an explicit expression for $\nabla^d X_{t+k|k}$ is readily available.

Proposition 2 Let $(X_t)_{t\in Z}$ be an autoregressive fractional process satisfying the difference equations $\emptyset(B)\nabla^d X_t = \varepsilon_t$, with $\varepsilon_t \sim WN(0, \sigma^2)$, $d \in]-.5,.5[$. Then, the predictors $\hat{X}_{t+h|t}$ defined as before satisfy the following recursive relations

$$\begin{aligned}
\hat{X}_{t+1|t} \;=\; & (\emptyset_1 - \pi_1)X_t \\
& + (\emptyset_2 + \pi_1\emptyset_1 - \pi_2)X_{t-1} \\
& + \ldots \\
& + (\emptyset_p + \pi_1\emptyset_{p-1} + \ldots + \pi_{p-1}\emptyset_1 - \pi_p)X_{t-p+1} \\
& + (\pi_1\emptyset_p + \pi_2\emptyset_{p-1} + \ldots + \pi_p\emptyset_1 - \pi_{p+1})X_{t-p} \\
& + (\pi_2\emptyset_p + \pi_3\emptyset_{p-1} + \ldots + \pi_{p+1}\emptyset_1 - \pi_{p+2})X_{t-p-1} \\
& + \ldots \\
& + (\pi_{k-p+1}\emptyset_p + \pi_{k-p+2}\emptyset_{p-1} + \ldots + \pi_{k-2}\emptyset_{k-1} - \pi_p)X_{t-p-1} \\
& + \ldots
\end{aligned} \tag{13}$$

$$\hat{X}_{t+2|t} \;=\; (\emptyset_1 - \pi_1)\hat{X}_{t+1|t} + (\emptyset_2 + \pi_1\emptyset_1 - \pi_2)X_{t-1} + \ldots \tag{14}$$

Proof. $\nabla^d X_{t+1} \sim AR(p)$, then

$$\nabla^d X_{t+1|t} = \emptyset_1 \nabla^d X_t + \emptyset_2 \nabla^d X_{t-1} + \ldots + \emptyset_p \nabla^d X_{t-p+1}.$$

If we substitute in (10) and rearrange we get the result. |

Remarks. If we define $\emptyset_0^* = -1$, $\emptyset_1^* = \emptyset_1, \ldots, \emptyset_p^* = \emptyset_p$, $\emptyset_n^* = 0$ for $n > p$ and $n < 0$, and $\pi_n = 0$ for $n < 0$ we get the equivalent expression

$$\hat{X}_{t+1|t} = \sum_{k=0}^{\infty} \sum_{\ell=0}^{p} \pi_{k+\ell} \, \emptyset^{*}_{k+1+\ell} \, X_{t-k} \, . \qquad (15)$$

When $h > 1$ we just replace X_{t-k} by its previously obtained forecast.

If the true model is a FARMA(p,d,q), with q non-zero, and if it is misspecified as a FARMA(p,d,0), then the result from the application of these last forecasting recursions is a biased forecast. However, this bias is only determined by the forecasts for the first q variables[3], since the above recursions should be rigorously true for $h > q$.

Notice that the recursions follow easily, as from (13) to (14). An expression similar to (11) would be cumbersome.

Finally, we should indicate the forecasting expressions for a general ARFIMA model. As usual, the forecasts for (X_t) can be made up recursively with forecasts for the differences. Suppose that $X_t \sim$ ARIMA(p,1,q); then forecasts for ∇X_t are made with usual procedures for an ARMA(p,q) process and $\hat{X}_{t+1|t} = X_t + \widehat{\nabla X}_{t+1|t}$, $\hat{X}_{t+2|t} = \hat{X}_{t+1|t} + \widehat{\nabla X}_{t+2|t}, \ldots$, $\hat{X}_{t+h|t} = \hat{X}_{t+h-1|t} + \widehat{\nabla X}_{t+h|t}$. If the integer d is greater than one then the general formula is

$$\hat{X}_{t+h|t} = \sum_{k=1}^{d} \binom{d}{k} (-1)^{k+1} \hat{X}_{t+h-k|t} + \widehat{\nabla^{d} X}_{t+h|t} \, . \qquad (16)$$

As usual[4] we assume that $\{X_1, X_2, \ldots, X_d\}$ and $\{\nabla^d X_1, \nabla^d X_2, \ldots, \nabla^d X_t\}$ are uncorrelated (or independent). Here $\hat{X}_{t+h|t}$ designates the projection on the subspace generated by $\{X_1, X_2, \ldots, X_d, \nabla X_{d+1}, \nabla X_{d+2}, \ldots, \nabla X_t\}$, and $\nabla X_{t+h|t}$ denotes the projection on the one generated by $\{\nabla X_{d+1}, \nabla X_{d+2}, \ldots, \nabla X_t\}$.

Proposition 3 Let (X_t) be an ARFIMA process satisfying the difference equations $\emptyset(B)\nabla^d X_t = \theta(B)\varepsilon_t$, with $\varepsilon_t \sim WN(0, \sigma^2)$, $d > -.5$ and $n := [d+.5]$.

Then the predictors can be obtained recursively from the formula

$$\hat{X}_{t+1|t} = \sum_{\ell=1}^{n} \binom{n}{\ell} (-1)^{\ell+1} X_{t+1-\ell} + \sum_{k=0}^{\infty} (-1)^{k} \binom{d-n}{k+1} \nabla^{n} X_{t-k} \, \widehat{\nabla^{d} X}_{t} \, . \qquad (17)$$

When observations are no longer available $X_{t+1-\ell}$ and $\nabla^n X_{t-k}$ are replaced by their predictors.

Proof. $\nabla^d X_t = \nabla^{d-n} (\nabla^n X_t) \sim$ ARMA(p,q) and the corresponding forecasts are obtained through the usual algorithms. The forecasts for the FARMA process $\nabla^n X_t$ are computed through the formula (10). These forecasts are introduced in (16) yielding the predictors $\hat{X}_{t+k|t}$.

---

[3] See, for instance, formula (5.13.17) in Brockwell and Davis (1991) for the recursive prediction of an ARMA with Durbin-Lenvinson algorithm

[4] See, for instance, Brockwell and Davis (1991), 9.5

Thus, the following expressions are sequentially used

$$\nabla^n X_{t+1|t} = \sum_{k=0}^{\infty} (-1)^k \binom{d-n}{k+1} \nabla^n X_{t-k} + \widehat{\nabla^{d-n}(\nabla^n X_t)}, \tag{18}$$

$$\hat{X}_{t+1|t} = \sum_{\ell=1}^{n} \binom{n}{\ell} (-1)^{\ell+1} X_{t+1-\ell} + \nabla^n X_{t+1|t}, \tag{19}$$

what amounts to (17). |

4. Review of the Empirical Work

This first FARMA forecasting exercise was performed by Granger and Joyeux (1980) in the same paper where they introduced fractional differenced models.

They studied the U.S. monthly index of consumer food prices, not seasonally adjusted, for the period January 1942 to June 1978. The series is nonstationary and the autocorrelations die out very slowly, clearly indicating that differencing is necessary. For the first differenced series, Granger and Joyeux reported positive autocorrelations up to lag 72, with fifteen values greater than twice the standard error. The partial autocorrelations were small except for the first, but the sixth, ninth and tenth were greater than twice the standard error. These values point towards an ARIMA(1,1,0) that was estimated. This ARIMA yielded a 19.85 mean square error (MSE) in ten-step forecast, and a random-walk ARIMA(0,1,0) yielded MSE = 27.6. Then, Granger and Joyeux estimated a fractional model $\nabla^d X_t = \varepsilon_t$ using the first differenced series. They tried a differencing parameter in a range of values from -.9 to +.45, and reported the corresponding MSE for ten-step-ahead forecasts. On the range +.2 to +.45, corresponding then to $d \in [1.2, 1.45]$, they obtained MSE around 16-17, representing an improvement over the ARIMA forecasts. The minimum MSE of 16.03 was obtained with d = 1.35.

The procedure suggested by Granger and Joyeux is relatively easy. First, the given series is transformed by traditional methods in order to obtain stationarity: let $(X_t)_{t=1}^N$, represent the transformed series, then fractionally differenced series $(\nabla^d X_t)_{t=1}^N$ are estimated for given values of d in a grid. The new series are obtained with the truncated fractional operator $\overset{t}{\underset{1}{\nabla^d}}$:

$$\nabla^d X_t \simeq \overset{t}{\underset{1}{\nabla^d}} X_t := \sum_{k=0}^{t-1} \pi_k X_{t-k}, \quad t = 1, 2, \ldots, N,$$

with $\pi_k = (-1)^k \binom{d}{k}$, i.e.

$$\overset{t}{\underset{1}{\nabla^d}} X_t := X_t - \binom{d}{1} X_{t-1} + \binom{d}{2} X_{t-2} - \ldots (-1)^{t-1} \binom{d}{t-1} X_1 . \tag{20}$$

The series $(\overset{t}{\underset{1}{\nabla^d}} X_t)_{t=1}^N$ is then split in sample and out-of-sample series:

$(\overset{t}{\underset{1}{\nabla^d}} X_t)_{t=1}^S$, and $(\overset{t}{\underset{1}{\nabla^d}} X_t)_{t=S+1}^N$.

Next, h-step-ahead forecasts are obtained using the sample series, with the spliting value S sliding from a minimum of S_1, say, to a maximum of $S_N = N - k$. The value of d that minimizes the mean square error of the forecasts is then chosen.

To obtain the forecasts, the relations (20) are again used. Let $\hat{X}_{S+h|S}$ represent the forecast for the period S+h using the observations up to period S. Since $E\nabla^d X_t = E\varepsilon_t = 0$, i.e., $EX_t = -\sum_{k=1}^{\infty} \pi_k X_{t-k}$, the following recursive relations are used

$$\hat{X}_{S+1|S} = \binom{d}{1} X_S - \binom{d}{2} X_{S-1} + \ldots - (-1)^S \binom{d}{S} X_1$$

$$\hat{X}_{S+2|S} = \binom{d}{1} \hat{X}_{S+1|S} - \binom{d}{2} X_S + \ldots - (-1)^{S+1} \binom{d}{S+1} X_1 \qquad (21)$$

. . . .

The procedure is computationally expensive and has the obvious limitation of dealing only with fractional noise.

Geweke and Porter-Hudak (1983) tested ARFIMA models with consumer prices indexes, following Granger's and Joyeux' lead.

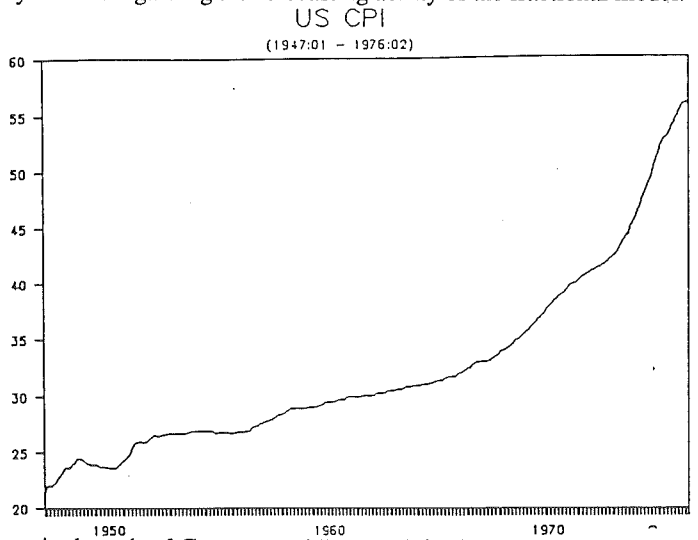
They run in parallel four forecasting procedures: (i) a fractional noise $\nabla^d X_t = \varepsilon_t$; (ii) an ARFIMA model $\emptyset(B)\nabla^d X_t = \theta(B)\varepsilon_t$ with d real; (iii) a conventional ARIMA with d integer; and (iv) a long autoregression of order fifty. The series tested were three: the consumer price index for food (Food CPI), the wholesale price index (WPI), and the consumer price index(CPI). The series covered periods from January 1947 to 1976-1978, roughly three hundred and fifty observations for each series.

The results of their experiment were mixed. In the sample period the fractional noise and the ARFIMA models provided competitive mean square errors, both for 10 and for 20 step-ahead forecasts. The lowest errors were always given by one of the two fractional models. But in the out-of-sample period the long autoregression performed better for Food CPI and for CPI, and only for WPI the ARFIMA model provided the best forecasts. Interesting enough, ARIMA model performed globally below any of the fractional models, and in most cases, ARFIMA scheme did not represent an improvement over the simple fractional noise.

On the whole, the results suggest that simple fractional noise can be reasonable conservative procedure for multi-step-ahead forecasts. The fact that the model, parameterized only by the value of d, could compete with the more flexible ARMA schemes and with the heavily parameterized autoregression of order 50, is by itself a significant asset.

We should question, however, the choice of the series. Granger and Joyeux (1980) as well as Geweke and Porter-Hudak (1983) computed the autocorrelations of the price indexes and their first differences. The original series are clearly nonstationary, and these authors argue that the autocorrelations of the differenced series, always positive and slowly decaying, are not typical of a finite ARMA model. But the point is that we cannot consider stationary the first differences of a price index! As the Figure 1 clearly suggests, the US CPI increases exponentially during the period under analysis. In fact, what could be considered stationary is

the rate of growth of the series, i.e., the inflation rates that correspond to the first difference of the logarithmized series. Facing this, we should question the possibility of inferring from these experiments any lessons regarding the forecasting ability of the fractional model.[5]



After the seminal work of Granger and Joyeux (1980) and of Geweke and Porter-Hudak (1983), few more papers discussed forecasting issues in the ARFIMA models setting.

Simulations of Chen, Jarrett, and Ray (1992), showed that ARFIMA and ARIMA models are likely to provide competitive tools in small sample fit and forecasting of ARIMA processes, and that estimated ARFIMA models do not necessarily outperform ARIMA models when fitting and forecasting small realizations of ARFIMA processes.

The theoretical work of Bonnie Ray (1993) singled out many important issues in forecasting long-memory models that are misspecified as short-memory ARIMA. She proved that this type of misspecification can yield large increases in the mean square forecasting error in medium- and long-range forecasting. In Crato (1992) a related problem was studied, the case where a stationary fractional noise is misspecified as a nonstationary ARIMA. It was there argued that this misspecification is highly plausible; in particular the case in which a trend is erroneously identified was discussed.

5. A New Look at ARFIMA CPI Forecasts

The double raised about the appropriate transformations to stationarize a price index prompted us to test FARMA forecasts for the US CPI rates. The data we used ranged from January 1947 to February 1976, monthly observations, reproduced from Business Statistics.

[5] The problem is similar to the explanation of Hurst effect with nonstationary models. The Hurst effect is not surprising in nonstationary phenomena. The superiority of long-memory models that try to fit long cycles is also not surprising for series that grow "almost monotonically", like the first differences of price indexes.

First, we computed the monthly rates of inflation as the first differences of the logarithms of the series. Next, over this new series four different forecasting methods were applied:

(i) a fractional noise model, i.e., a FARMA(0,d,0) with the parameter d estimated by the grid method as the minimizer of 1-20 mean square forecast error;

(ii) a FARMA(p,d,q) model selected by the AIC criterion[6] over 16 FARMA models (p,q = 0, 1, 2, 3) all estimated with the maximum likelihood approach of Sowell (1992), by using his program QGSTFRAC;

(iii) an ARMA(p,q) model selected by AIC criterion over 16 ARMA models (p,q = 0, 1, 2, 3,) all estimated by the maximum likelihood approach by using the package PEST from Brockwell and Davis (1991);

(iv) a simple exponential smoothing procedure with the smoothing parameter chosen as the minimizer of in-sample errors.

A long autoregressive model was not used. The log differentitation applied resulted in nonsignificant autoregressive coefficients for orders larger than the one in (iii).

For the in-sample period 331 observations were considered: 1945:01 to 1974:06. The out-of-sample range included 20 periods: 1974:07 to 1076:02. This way, the series are similar to both the data used by Granger and Joyeux (1980) and the data used by Geweke and Porter-Hudak (1983).

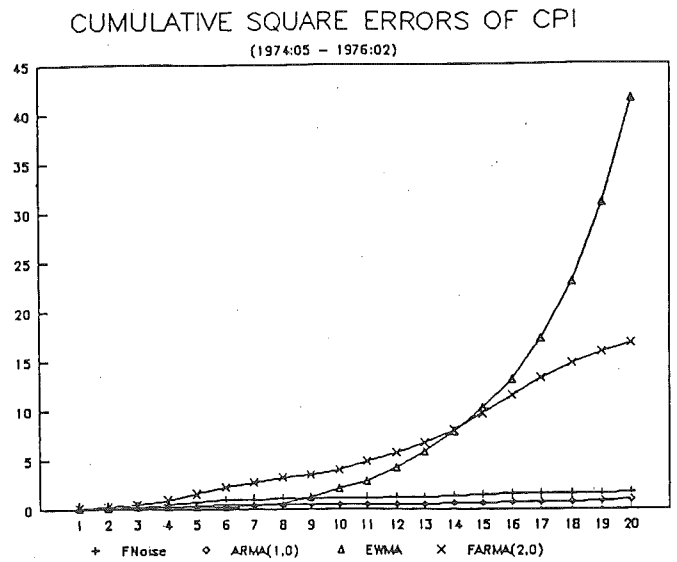
As result of the in-sample estimations, the models chosen were the following:

(i) fractional noise $\hat{d} = .20$;

(ii) a FARMA(1,d,0) with $\hat{\emptyset}_1 = .16$ and $\hat{d} = .40$;

(iii) an ARMA(3,0); with $\hat{\emptyset}_1 = .26$, $\hat{\emptyset}_2 = .22$, $\hat{\emptyset}_3 = .15$

(iv) an exponential smoothing with constant $\alpha = .20$.



[6] See, e.g., Brockwell and Davis (1991), p.304.

The performance of the four methods on the rates of inflation is graphically compared on the Figure 2, that represents the cumulative square errors of the prediction of the inflation rates. At a short range, all methods perform more or less equally well. At the forth step-ahead, the general FARMA model starts giving divergent predictions, but, after a while, it is the exponential smoothing approach (EWMA) that gives the worse forecasts. The best overall predictions are given by the ARMA model, but the fractional noise forecasts are always very competitive. This seems to confirm the conclusions of Geweke and Porter-Hudak (1983) that asserted that the fractional noise model seemed to give predictions that are pretty reasonable for such a parsimoniously parametrized model.

6. Conclusions

Various theoretical and practical problems related to ARFIMA forecasting were discussed. Both the available theoretical results and the empirical forecasting experiments were critically reviewed.

A decomposition formula of the predictors was obtained, highlighting the relative roles played by the short-term and by the long-range components of the ARFIMA models. A computationally simple formula was derived for the ARFIMA(p,d,0) case.

In reviewing the empirical experiments in forecasting with long-memory models, the validity of the main empirical example presented in the literature was questioned. When changing the approach in order to avoid the criticized problems, the success of the ARFIMA forecasts was not so apparent, but the performance of the pure fractional noise model remained very competitive.

Acknowledgment

This work is partially based on an invited paper presented by the author at the 11th International Symposium on Forecasting, New York, June 1991. This research was partly done while the author was working with a grant from the ISEG, Technical University of Lisboa; this grant and an additional funding from JNICT, Portugal, are gratefully acknowledged. The author would also like to thank Peter Brockwell and Fallaw Sowell for the gracious supply of their ARFIMA estimation programs, as well as an anonymous referee who made valuable suggestions and comments.

References

[1] Brockwell, Peter J. and Davis, Richard A., *Time Series: Theory and Methods*, second edition, Springer-Verlag, 1991.

[2] Chen, Shaw K, Jarett, Jeffrey and Ray, Bonnie K., *Forecasting financial series by ARIMA methods; a comparison of integer and fractionally differenced models*, Department of Management Science and Information Systems, University of Rhode Island, 1992.

[3] Crato, Nuno, *Recent research on fractional noise in economic time series*, Proceedings of the Third Cemapre Conference, ISEG, Lisbon, 1991.

[4] Crato, Nuno, *Long-memory models misspecified as nonstationary ARIMA*, Proceedings of the Business and Economic Statistics Section, American Statistical Association, 1992.

[5] Crato, Nuno, *Some international evidence regarding the stochastic memory of stock returns*, Applied Financial Economics 3 (1993) forthcoming.
[6] Geweke, John and Porter-Hudak, Susan, *The estimation and application of long memory time series models*, Journal of Time Series Analysis 4 (1983) 221-238.
[7] Gonçalves, Esmeralda, *Une generalisation des processus ARMA*, Annales d'Economie et de Statistique 5 (1987) 109-145.
[8] Granger, C.W.J. and Joyeux, Roselyne, *An introduction to long-memory time series models and fractional differencing*, Journal of Time Series Analysis 1 (1980) 15-29.
[9] Hosking, Jon R.M., *Fractional differencing*, Biometrika 68 (1981) 165-176.
[10] Peiris, M. Shelton, *A note on the predictors of differenced sequences*, Australian Journal of Statistics 29 (1987) 42-48.
[11] Peiris, M.S. and Perera, B.J.C., *On prediction with fractionally differences ARIMA models*, Journal of Time Series Analysis 9 (1988) 215-220.
[12] Ray, Bonnie K., *Modelling long-memory processes for optimal long-range prediction*, Journal of Time Series Analysis (1983) forthcoming.
[13] Sowell, Fallaw, *Maximum likelihood estimation of stationary univariate fractionally integrated time series models*, Journal of Econometrics 53 (1992) 165-188.

INSTRUÇÕES AOS AUTORES

Os autores que desejem submeter um artigo à Investigação Operacional devem enviar três cópias desse trabalho para:

Prof. Joaquim J. Júdice
Departamento de Matemática
Universidade de Coimbra
3000 Coimbra, Portugal

Os artigos devem ser escritos em Português ou Inglês. A primeira página deve conter a seguinte informação:

- Título do artigo
- Autor(es) e instituição(ões) a que pertence(em)
- Abstract (em inglês)
- Resumo
- Keywords (em inglês)
- Título abreviado

As figuras devem aparecer em separado de modo a poderem ser reduzidas e fotocopiadas. As referências devem ser numeradas consecutivamente e aparecer por ordem alfabética de acordo com os seguintes formatos:

Artigos: autor(es), título, título e número da revista (livro com indicação dos editores), ano, páginas.

Livros: autor(es), título, editorial, local de edição, ano.

Fotografia, Montagem
Impressão e Acabamentos
Tip.Nocamil
COIMBRA

ÍNDICE

Associação Portuguesa para o Desenvolvimento da Investigação Operacional

CESUR - Instituto Superior Técnico - Avenida Rovisco Pais 1000 Lisboa - Telef. 80 74 55