SOLUÇÃO EFICIENTE DE PROBLEMAS DE PROGRAMAÇÃO LINEAR DE GRANDE PORTE

GUILHERME GENTILE DE AGUIAR† - RA: 105020

Resumo. Nas últimas décadas vários métodos vêm sendo propostos para a resolução de um problema de programação linear, em especial os métodos de pontos interiores. A pesquisa sobre métodos de resolução de problemas de programação linear ainda continua, visto que com o desenvovimento da computação, e com a nescessidade real, cada vez mais é nescessário resolver problemas maiores em menor tempo.

Na década de 1990, Dantzig divulgou um algoritmo proposto por Von Neumann em 1948. Do algoritmo de Neumann foi proposto um algoritmo de ajustamento pelo par ótimo que trabalha com duas variáveis. Em cada iteração do algoritmo é resolvido um subproblema, o que gera o maior esforço computacional do algoritmo. ãA partir disso, foi proposto um algoritmo que trabalha com p variáveis, igualmente o maior esforço computacional vêm de resolver um subproblema. Ao resolvermos o subproblema por métodos de pontos interiores, o grande esforço vêm de resolver sistemas lineares.

Para resolvermos um sistema linear por métodos diretos é nescessário a fatoração de uma matriz, no caso de ordem $p+1$, o que possui complexidade $O((p+1)^3)$, e em seguida são resolvidos dois sistemas lineares mais simples, os quais estão associados a matrizes triangulares,que possuem complexidade $O((p+1)^2)$. O que é um custo muito baixo, se considerarmos que um problema de programação linear de grande porte.

1. Introdução

A otimização está presente no nosso dia a dia. Sempre que precisamos tomar uma decisão, procuramos escolher entre as várias alternativas aquela que, naquele momento, nos dê maior satisfação. Nos últimos anos, a matemática vem estudando como encontrar essa solução que dê nos essa maior satisfação.

Na matemática, a área que estuda problemas de otimização é geralmente chamada de Programação Matemática. Esta denominação identifica uma ampla classe de problemas. O nome programação foi empregado porque os militares se referem ao planejamento de atividades como "programa". Boa parte dos acontecimentos que culminaram com a criação desta importante área da matemática, se deram durante a Segunda Guerra Mundial. De fato, George B. Dantzig usou o termo Programação Linear (mais especificamente "Programming in a linear Structure", mais tarde resumido para Linear Programming e generalizado como Mathematical Programming) para analisar um problema de planejamento para a força aérea americana. Com a disseminação do uso do computador, programação passou a ser entendido como a codificação de um algoritmo em uma determinada linguagem e às vezes a Programação Matemática é confundida com programação de computadores. Linguagens de programação e computadores são muito usados no estudo de problemas de otimização, mas a Programação Matemática é muito mais do que a codificação de um algoritmo.

†Graduando de Matemática Aplicada e Computacional da Unicamp e Bolsista de Iniciação Científica do CNPq. Email: aguiar.guig@gmail.com.

No nosso caso queremos estudar os problemas de programação linear, que consiste em otimizar uma função linear sujeita à um conjunto de restrições lineares. Há muitos métodos, ou programas, que resolvem esse problema. Um desses métodos é o método Simplex, desenvolvido por George B. Dantzig. O método Simplex tornou-se uma ferramenta fundamental em programação linear durante muitos anos. No entanto, este método possui convergência exponencial no número de variáveis, apesar de nem em todos os problemas tal complexidade ocorrer. Com o desenvolvimento dos computadores há uma busca por métodos que tenham melhor convergência.

Em 1984 Karmakar apresentou um algoritmo para programação linear com complexidade polinomial. A partir da publicação desse algoritmo início o desenvolvimento de outros algoritmos, chamados de algoritmos de pontos interiores. Nesse trabalho, vamos estudar sobre um método de pontos interiores.

O método estudado é baseado no algoritmo de ajustamento pelo par ótimo. O algoritmo de ajustamento pelo par ótimo trabalha com duas varíaveis em cada iteração. O método é apresentado por Jair Silva em sua tese de doutorado [1] trabalha com p variáveis a cada iteração.

2. O problema de programação linear

O problema de programação linear (PPL) consiste em minimizar uma função linear, chamada de função objetivo, sujeita a um conjunto de restrições lineares. A seguir temos um problema de programação linear representado na forma padrão:

$$\begin{array}{ll} mimimizar & c^t x \\ s.a. & Ax = b \\ & x \geq 0 \end{array} \tag{2.1}$$

onde $A \in \mathbb{R}^{n \times m}$, $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ e $b \in \mathbb{R}^m$. O problema escrito nessa forma é chamado de problema de programação linear na forma padrão. Geralmente os problemas não aparecem nessa forma, mas todos podem ser reduzidos à esta forma.

Para a resolução do problema há, hoje, vários métodos .Um dos mais populares é o método Simplex proposto por Dantzig na década dem 1940. Durante muitos anos o método Simplex foi a ferramenta fundamental do estudo de programação linear. No entanto, este método possui convergência exponencial no número de variáveis, apesar de nem em todos os problemas tal complexidade ocorrer.

Em 1979 Khachian apresentou o método dos elipsóides que, apesar da convergência polinomial, não se mostrou prático. Em 1984, Karkarmar apresentou um novo algoritmo para programação linear, também com complixidade polinomial. A partir disso, iniciou-se uma nova linha de pesquisa conhecida como métodos de pontos interiores.

Os métodos de pontos interiores podem ser classificados como primal, dual e primal-dual dependendo em que espaço estão sendo realizados as iterações. O problema (2.1) traz a forma primal do problema de programação linear. A forma dual associada ao problema (2.1) é dada por:

$$\begin{array}{ll} maximizar & b^t y \\ s.a. & A^t y + z = 0 \\ & z \geq 0 \end{array} \tag{2.2}$$

onde $y \in \mathbb{R}^m$, $z \in \mathbb{R}^n$ e A, b, c como definido no problema primal.

Os problemas (2.1) e (2.2) juntos são chamados de par primal-dual. As condições de otimalidade de primeira ordem(Karush - Kush - Tucker) dos problemas (2.1) e

(2.2) são dadas por:

$$\begin{array}{l} Ax - b = 0 \\ A^t y + z - c = 0 \\ XZe = 0 \\ (x, z) \geq 0 \end{array} \tag{2.3}$$

onde $X = diag(x)$, $Z = diag(z)$ e $e \in \mathbb{R}^n$ é o vetor com todas as coordenadas iguais a um.

Se (x, y, z) for uma solução de (2.3) então x é uma solução ótima de (2.1) e (y, z) é uma solução ótima de (2.2). Um ponto (x, y, z) é factível se ele satisfaz as restrições do conjunto de restrições de problema primal e dual e o ponto é dito interior se $(x, z) > 0$. O gap de um problema de programação linear é definido como a diferença entre os valores das funções objetivos do problema primal e dual, ou seja $\gamma = c^t x - b^t y$.

Alternativamente, os métodos de pontos interiores podem ser classificados em três categorias: afim-escala, redução de potencial e trajetória central. A classe de métodos que apresentou as melhores propriedades práticas e teóricas são os métodos primais-duais pertencente a categoria trajetória central. O método aqui estudado pertencente a essa classe. A seguir, vamos falar um pouco mais sobre esses métodos.

A maioria dos métodos de pontos interiores utiliza o método de Newton em suas iterações. O método de Newton é uma generalização do método de Newton para encontrar zeros de funções, que consiste em: dado um ponto x^k, devemos encontrar uma direção Δx^k. Ao encontrarmos a direção Δx^k, atualizamos o valor de x, tal que $x^{k+1} = x^k + \Delta x^k$. Os métodos primais-duais de pontos interiores podem ser vistos como aplicações do método de Newton para calcular aproximações da solução de uma sequencia de sistemas não lineares (2.3), perturbados por um parâmetro μ, usado para relaxar as restrições de não-negatividades. Assim, temos:

$$F_\mu(x, y, z) = \begin{bmatrix} Ax - b \\ A^t y + z - c = 0 \\ -XZe + \mu e = 0 \end{bmatrix} = 0, \ (x, z) \geq 0. \tag{2.4}$$

Se $\mu = 0$, então os problemas (2.3) e (2.4) são equivalentes. Assim, a solução do problema (2.4) se aproxima da solução de (2.3) conforme $\mu \to 0$.

Um método primal-dual obtém uma solução aproximada para o problema gerando uma sequência de pontos (x_k, y_k, z_k) satisfazendo (2.4) para todo $\mu > 0$ é denominado trajetória central. A cada iteração do método é aplicado um passo do método de Newton para resolve o sistema (2.4), com um dado parâmetro μ_k. A direção de Newton, $(\Delta x^k, \Delta y^k, \Delta z^k)$, é obtida da solução de um sistema de equações lineares.

A cada iteração do método é nescessário resolver um sistema de equações lineares para determinar a direção de Newton. Quando métodos diretos são utilizados para resolver o sistema linear é feita uma fatoração de uma matriz e em seguida são resolvidos dois sistemas lineares mais simples, os quais estão associados a matrizes triangulares. O processo de fatoração de uma matriz de ordem n apresenta complexidade $O(n^3)$ e a solução de sistemas triangulares pode ser realizada com complexidade $O(n^2)$.

3. O Algoritmo de Von Neumann

O algoritmo de Von Neumann foi proposto em 1948 por Von Neumann à Dantzig, que o divulgou na década de 1990. O algoritmo possui propriedades interessantes, como simplicidade e convergência inicial rápida, porém ele não é muito prático para resolver problemas lineares. Consideremos o problema de encontrar uma solução

factível para o conjunto de restrições linear:

$$\begin{aligned} Px &= 0 \\ e^t x &= 1 \\ x &> 0 \end{aligned} \tag{3.1}$$

onde $P \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ e $e \in \mathbb{R}^n$ é o vetor com todas as coordenadas igual a um, e as colunas de P tem norma um, isto é $||P_j|| = 1$, para $j = 1, ...n$. Geometricamente as colunas P_j podem ser vistas como pontos sobre a hiperesfera m-dimensional com raio unitário e centro na origem. O problema pode ser descrito então como atribuir ponderações x_j não negativas as colunas P_j de modo que depois de reescalado seu centro de gravidade seja a origem. Problemas de programação linear podem ser colocados na forma (3.1).

O algoritmo de Von Neumann trabalha na resolução do problema (3.1). A figura 1 descreve como método trabalha em cada iteração. Primeiramente ele encontra a coluna P_s de P que forma o maior ângulo com o resíduo b^{k-1}, e então o próximo resíduo é a projeção da origem no segmento de renta ligando b^{k-1} a P_s. A seguir trazemos o algoritmo:

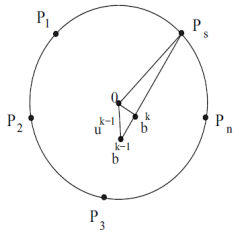

FIGURA 1. Ilustração do algoritmo de Von Neumann

Algoritmo de Von Neumann
Dado: $x^0 \geq 0$, com $e^t x^0 = 1$.
Calcule $b^0 = Px^0$.
Para $k = 1, 2, 3 \ldots$ **faça**
[1] Calcule:
$s^+ = argmin_{j=1,\ldots,n} P_j^t b^{k-1}$,
$v_{k-1} = P_{s^+}^t b^{k-1}$.
[2] **Se** $v_{k-1} > 0$, **então PARE**; o problema (3.1) é infactível.
[3] Calcule
$u^{k-1} = ||b^{k-1}||$,
$\lambda = \dfrac{1 - v_{k-1}}{(u^{k-1})^2 - 2v_{k-1} + 1}$.
[4] Atualize
$b^k = \lambda b^{k-1} + (1 - \lambda) P_{s^+}$,
$x^k = \lambda x^{k-1} + (1 - \lambda) e_{s^+}$,
onde e_{s^+} é o vetor da base canônica com 1 na s^+-ésima coordenada
$k = k + 1$
FIM.

Como dito anteriormente, esse algoritmo não é muito prático para resolver problemas lineares, então foram criados outros algoritmos à partir desse algoritmo. Um deles é o algoritmo de ajustamento pelo par ótimo. Este algoritmo indentifica os vetores P_{s^+} e P_{s^-} que tem o maior e o menor ângulo com o vetor b^{k-1}, respectivamente. Em seguida ele encontra $x_{s^+}^k$, $x_{s^-}^k$ e λ, onde $x_j^k = \lambda x_{k-1}^j$ para todo $j \neq s^+$

e $j \neq s^-$, que minimiza a distancia de b^k à origem. A seguir trazemos o algoritmo de ajustamento pelo par ótimo:

Algoritmo de ajustamento pelo par ótimo
Dado: $x^0 \geq 0$, com $e^t x^0 = 1$.
Calcule $b^0 = Px^0$.
Para $k = 1, 2, 3 \ldots$ **faça**
[1] Calcule:
$s^+ = argmin_{j=1,\ldots,n}\{P_j^t b^{k-1}\}$,
$s^- = argmax_{j=1,\ldots,n}\{P_j^t b^{k-1} | x_j > 0\}$,
$v_{k-1} = P_{s^+}^t b^{k-1}$.
[2] **Se** $v_{k-1} > 0$, **então PARE**; o problema (3.1) é infactível.
[3] Resolva o problema:

$$\begin{array}{rl} minimizar & ||\lambda_0(b^{k-1} - x_{s^+}^{k-1} P_{s^+} - x_{s^-}^{k-1} P_{s^-}) + \lambda_1 P_{s^+} + \lambda_2 P_{s^-}||^2 \\ s.a. & \lambda_0(1 - x_{s^+}^{k-1} - x_{s^-}^{k-1} + \lambda_1 + \lambda_2 = 1, \\ & \lambda_0 \geq 0, \\ & \lambda_1 \geq 0, \\ & \lambda_2 \geq 0. \end{array} \tag{3.2}$$

[4] Atualize
$b^k = \lambda_0(b^{k-1} - x_{s^+}^{k-1} P_{s^+} - x_{s^-}^{k-1} P_{s^-}) + \lambda_1 P_{s^+} + \lambda_2 P_{s^-}$,
$u^k = ||b^k||$,
$x_j^k = \begin{cases} \lambda_0 x_j^{k-1}, & j \neq s^+ \text{ e } j \neq s^-, \\ \lambda 1 & j = s^+, \\ \lambda 2 & j = s^-. \end{cases}$
$k = k + 1$.
FIM.

O maior esforço do algoritmo é a resolução do subproblema (3.2), que deve ser resolvido a cada iteração.

4. O método estudado

O algoritmo de ajustamento pelo par ótimo trabalha com duas varíaveis em cada iteração. O método proposto por Jair Silva em [1] trabalha com p variáveis. A idéia central ultilizada no algoritmo para duas variáveis é resolver o subproblema (3.2). Esse subproblema pode ser generalizado se usarmos qualquer quantidade de colunas e assim dar importância a quantas variáveis desejarmos. A escolha de p é livre, aqui não discutiremos sobre essa escolha, e apenas sobre a generalização do algoritmo para p variáveis.

O algoritmo de ajustamento ótimo para p coordenadas, começa indentificando as s_1 colunas que fazem o maior ângulo com o vetor b^{k-1}, em seguida ele encontra as s_2 colunas que fazem o menor ângulo com o vetor b^{k-1}, onde $s_1 + s_2 = p$. Depois é resolvido o subproblema de otimização, atualizando o resíduo e o ponto corrente.

Algoritmo de ajustamento ótimo para p coordenadas
Dado: $x^0 \geq 0$, com $e^t x^0 = 1$.
Calcule $b^0 = Px^0$.
Para $k = 1, 2, 3 \ldots$ **faça**
[1] Calcule:
$\{P_{\eta_1^+}, P_{\eta_2^+}, \ldots, P_{\eta_{s_1}^+}\}$ que fazem o maior ângulo com o vetor b^{k-1} .

$\{P_{\eta_1^-}, P_{\eta_2^-}, \ldots, P_{\eta_{s_1}^-}\}$ que fazem o menor ângulo com o vetor b^{k-1} e tal que $x_i^{k-1} > 0$, $i = \eta_1, \eta_2, \ldots \eta_{s_1}^-$, onde $s_1 + s_2 = p$.

$v_{k-1} = maximo_{i=1,\ldots s_1} P_{\eta_i^+}^t b^{k-1}$.

[2] **Se** $v_{k-1} > 0$, **então PARE**; o problema (3.1) é infactível.

[3] Resolva o problema:

$$\begin{aligned} minimizar\ & ||\lambda_0 \left(b^{k-1} - \textstyle\sum_{i=1}^{s_1} x_{\eta_i^+}^{k-1} P_{\eta_i^+} - \sum_{j=1}^{s_2} x_{\eta_j^-}^{k-1} P_{\eta_j^-} \right) + \sum_{i=1}^{s_1} \lambda_{\eta_i^+} P_{\eta_i^+} + \sum_{j=1}^{s_2} \lambda_{\eta_i^-} P_{\eta_j^-} ||^2 \\ s.a. \quad & \lambda_0 \left(1 - \textstyle\sum_{i=1}^{s_1} x_{\eta_i^+}^{k-1} - \sum_{j=1}^{s_2} x_{\eta_j^-}^{k-1} \right) + \sum_{i=1}^{s_1} \lambda_{\eta_i^+} + \sum_{j=1}^{s_2} \lambda_{\eta_i^-} = 1, \\ & \lambda_{\eta_i^+} \geq 0, \text{ para, } i = 1, \ldots, s_1, \\ & \lambda_{\eta_j^-} \geq 0, \text{ para, } j = 1, \ldots, s_2. \end{aligned} \tag{4.1}$$

[4] Atualize

$$b^k = \lambda_0 \left(b^{k-1} - \textstyle\sum_{i=1}^{s_1} x_{\eta_i^+}^{k-1} P_{\eta_i^+} - \sum_{j=1}^{s_2} x_{\eta_j^-}^{k-1} P_{\eta_j^-} \right) + \sum_{i=1}^{s_1} \lambda_{\eta_i^+} P_{\eta_i^+} + \sum_{j=1}^{s_2} \lambda_{\eta_i^-} P_{\eta_j^-},$$

$$u^k = ||b^k||,$$

$$x_j^k = \begin{cases} \lambda_0 x_j^{k-1}, & j \notin \{\eta_1^+, \ldots, \eta_{s_1}^+, \eta_1^+, \ldots, \eta_{s_2}^1\}, \\ \lambda 1 & j = \eta_i^+, i = 1, \ldots s_1, \\ \lambda 2 & j = \eta_i^-, i = 1, \ldots s_2. \end{cases}$$

$k = k + 1$.

FIM.

Em cada iteração do algoritmo de ajustamento ótimo para p coordenadas, é nescessário resolver o subproblema (4.1). Estre subproblema é resolvido encontrando uma solução no octante positivo, sujeito a factibilidade para um sistema linear de ordem no máximo $(p + 1) \times (p + 1)$. Uma forma de resolver este subproblema é verificar todos os possíveis casos de soluções factíveis. Contudo, ao resolver o subproblema desta forma o número de casos possíveis de solução aumenta exponencialmente com o valor de p, mais precisamente $2^{p+1} + 1$.

Com a finalidade de contornar o problema do crescimento exponencial do números de casos com o valor de p, podemos abordar o subproblema (4.1) de outra forma e resolvê-lo aplicando métodos de pontos interiores. A grande vantagem de usar métodos de pontos interiores é que em cada iteração do método de pontos interiores surge sistemas lineares, cujo o custo é a fatoração de uma matriz de ordem $(p + 1) \times (p + 1)$.

O custo computacional de fatorar uma matriz de ordem 10×10 ou 100×100, por exemplo, não muito diferente se considerarmos que o problema de programação linear a ser resolvido é de grande porte. Já verificar $2^{10} - 1$ ou $2^{100} - 1$ casos possíveis representam custos bem diferentes.

Em [1] é mostrada a convergência do método. O método apresenta um desempenho superior em relação ao algoritmo de Von Neumann. Também é mostrado que se $p_2 \geq p_1$ o algoritmo de ajustamento ótimo para p_2 coordenadas possui desempenho superior em relação ao algoritmo de ajustamento ótimo para p_1 coordenadas.

5. Implementação computacional

O código PCx [2] foi desenvolvido no Optimization Technology Center at Argonne National Laboratory and Northwestern University e implementa o método primal-dual preditor-corretor com múltiplas correções. Esta abordagem é considerada a mais eficiente das variantes de métodos de pontos interiores. As rotinas são implementadas em C exceto as rotinas responsáveis pela solução dos sistemas

lineares, que utiliza uma biblioteca para fatoração esparsa de Cholesky, desenvolvida em FORTRAN77. O PCx trabalha com problemas de programação linear no formato MPS. Os problemas podem conter restrições de igualdade e desigualdade, variáveis livres e limitadas. Uma rotina de preprocessamento converte os modelos em um formato padrão e após a solução ser encontrada é transformada em termos da formulação original.

A estratégia utilizada para realizar os experimentos foi fazer algumas iterações com o algoritmos simples antes da centralização dos pontos. A justificativa para esta estratégia é que se usarmos a família de algoritmos simples depois de centralizar os pontos melhoramos estes, mas podemos perder sua centralidade que é importante para os métodos de pontos interiores.

Foi trabalhado na implementação do algoritmo simples a fim de melhorá-la. Foram feitas alterações no cálculo da direção e na fatoração de Cholesky.

Utiliza-mos 86 problemas da coleção NETLIB [3]. Na tabela 1 apresentamos uma comparação do desempenho do PCx original e do PCx modificado com auxílio do algoritmo simples. As notações são as seguintes:

- Problema: nome do problema;
- Linhas: quantidade de linhas;
- Colunas: quantidade de colunas;
- Nnulos: número de elementos nulos;
- itAux: quantidade de iterações realizadas com o algoritmo de ajustamento ótimo para p coordenadas;
- PCxori: número total de iterações do PCx original;
- PCxoriM: número total de iterações do PCx original modificado auxiliado pelo algoritmo simples.

Tabela 1. PCxori × PCxM

Problema	Linhas	Colunas	Nnulos	PcxOriginal	itAux	PcxM
ADLITTLE	55	137	417	11	7	11
AFIRO	27	51	102	7	14	7
AGG	390	447	2055	18	5	19
AGG2	514	750	4558	21	3	21
AGG3	514	750	4574	19	25	25
BANDM	240	395	1894	16	17	16
BEACONFD	86	171	1316	10	3	10
BLEND	71	111	477	9	5	9
BNL1	610	1491	5256	33	13	39
BNL2	1964	4008	14037	33	*	*
BOEING1	331	697	3104	18	9	19
BOEING2	125	264	979	12	21	12
BORE3D	81	138	549	15	5	15
BRANDY	133	238	1911	17	3	16
CAPRI	241	436	1528	17	31	20
CYCLE	1420	2773	15004	23	*	*
CZPROB	671	2779	5531	26	42	34
D2Q06C	2132	5728	31965	24	*	*
D6CUBE	403	5444	34233	16	3	16
DEGEN2	442	757	4167	11	5	11
DEGEN3	1501	2604	25425	13	*	*
DFL001	5984	12143	35274	37	*	*

TABELA 2. Continuação da tabela PCxori × PCxM

Problema	Linhas	Colunas	Nnulos	PcxOriginal	itAux	PcxM
E226	198	429	2515	18	19	16
ETAMACRO	334	669	1995	26	39	23
FFFFF800	322	826	5164	26	3	27
FINNIS	438	935	2332	22	3	24
FIT1D	24	1049	13427	17	5	17
FIT1P	627	1677	9868	16	3	15
FIT2D	25	10524	129042	19	33	22
FIT2P	3000	13525	50284	20	*	*
FORPLAN	121	447	4415	21	3	21
GANGES	1113	1510	6537	16	*	*
GREENBEA	1933	4164	23765	49	*	*
GREENBEB	1932	4154	23673	43	*	*
GROW15	300	645	5620	29	44	17
GROW22	440	946	8252	21	49	18
GROW7	140	301	2612	16	5	17
ISRAEL	174	316	2443	19	5	15
KB2	43	68	413	11	26	11
LOFTI	133	346	867	13	3	14
MAROS	846	1966	6634	18	5	18
MAROS-R7	2152	7440	100486	14	*	*
MODSZKI	665	1599	3065	20	4	19
NESM	654	2922	13244	25	8	26
PEROLD	593	1389	5636	33	3	33
PILOT	1368	4543	41879	30	*	*
PILOT 4	396	1022	5001	46	10	48
PILOT 87	1971	6373	72163	24	*	*
RECIPE	64	123	443	8	4	8
SC105	104	162	339	9	4	8
SC205	203	315	663	10	4	10
SC50A	49	77	159	7	3	7
SC50B	48	76	146	6	3	6
SCAGR25	469	669	1715	17	14	17
SCAGR7	127	183	455	13	4	17
SCFXM1	305	568	2732	17	7	16
SCFXM2	610	1136	4919	19	3	18
SCFXM3	915	1704	8206	19	3	19
SCORPION	340	412	1534	11	13	11
SCRS8	412	1199	3036	21	17	21
SCSD1	78	760	2388	8	15	8
SCSD6	147	1350	4316	11	3	11
SCSD8	397	2750	8584	10	5	10
SCTAP1	284	644	1802	14	17	14
SCTAP2	1033	2413	7052	12	*	*
SCTAP3	1408	3268	9383	14	*	*
SEBA	448	901	8252	12	13	14

Tabela 3. Continuação da tabela PCxori × PCxM

Problema	Linhas	Colunas	Nnulos	PcxOriginal	itAux	PcxM
SHARE1B	112	248	1148	18	13	16
SHARE2B	96	112	777	17	5	17
SHELL	487	1451	2904	20	43	20
SHIP04L	292	1905	4290	12	12	11
SHIP04S	216	1281	2875	12	3	12
SHIP08L	470	3121	7122	14	4	14
SHIP08S	276	1604	3644	11	6	11
SHIP12L	610	4171	9254	15	5	14
SHIP12S	340	1942	4297	12	7	12
SIERRA	1212	2705	7771	18	*	*
STAIR	356	532	3813	13	14	14
STANDATA	314	796	1403	13	3	12
STANDGUB	314	796	1403	13	3	12
STANDMPS	422	1192	2831	24	3	23
STOCFOR1	102	150	421	11	*	*
STOCFOR2	1980	2868	8090	19	*	*
TUFF	257	567	4095	18	4	18
WOOD1P	171	1718	44575	23	10	22
WOODW	708	5364	19809	30	10	28

Os testes foram realizados num computador com processador Intel Core 2 Duo, 2,2 GHz de processamento e 4GB de memória RAM, utilizando Linux. Foram utilizados compiladores gcc e gfortran. Foram testados 86 problemas, desses 34 mantiveram a quantidade de iterações em relação ao PCx original, 19 diminuiram e 16 aumentaram. 17 problemas apresentaram problemas na resolução, esses problemas são os denotados por * na tabela.

A seguir é feita uma comparação de tempo de execução comparando uma implementação anterior da modificação, a implementação atual e o PCx original. Os resultados, são mostrados na tabela 4. As notações são as seguintes:

- Problema: nome problema;
- PCxori: tempo de de execução do PCx para aquele problema.
- PCxM1: tempo de execução do PCx modificaco auxiliado com uma implementação anterior do algoritmo simples;
- S1: tempo de execução da implementação anterior do algoritmo simples;
- PCxM2: tempo de execução do PCx modificaco auxiliado com uma implementação atual do algoritmo simples;
- S2: tempo de execução da implementação atual do algoritmo simples.

Os tempos são medidos em segundos. Os problemas testes que tiveram problemas são omitidos nessa tabela.

Tabela 4. Tempos de execução em segundos

Problema	PCXOri	S2	PcxM2	S1	PcxM1
ADLITTLE	0,0030	0,0000	0,0139	0,0000	0,0139
AFIRO	0,0010	0,0000	0,0139	0,0000	0,0139
AGG	0,0289	0,0100	0,0389	0,0100	0,0309
AGG2	0,0390	0,0100	0,0719	0,0100	0,0639
AGG3	0,0559	0,0100	0,0779	0,0300	0,0719

TABELA 5. Tempos de execução em segundos

Problema	PCXOri	S2	PcxM2	S1	PcxM1
BANDM	0,0189	0,0200	0,0359	0,0100	0,0279
BEACONFD	0,0069	0,0000	0,0099	0,0100	0,0119
BLEND	0,0039	0,0000	0,0069	0,0000	0,0099
BNL1	0,0739	0,0500	0,1049	0,0600	0,1119
BOEING1	0,0309	0,0400	0,0489	0,0500	0,0589
BOEING2	0,0089	0,0100	0,0249	0,0200	0,0309
BORE3D	0,0069	0,0100	0,0309	0,0200	0,0289
BRANDY	0,0209	0,0100	0,0209	0,0100	0,0219
CAPRI	0,0229	0,0200	0,0349	0,0200	0,0419
CZPROB	0,0379	0,0600	0,0859	0,0700	0,0879
D6CUBE	0,2769	0,1400	0,4167	0,1400	0,4089
DEGEN2	0,0499	0,0300	0,0799	0,0400	0,0689
E226	0,0249	0,0200	0,0429	0,0300	0,0449
ETAMACRO	0,0489	0,0200	0,0559	0,0300	0,0599
FFFFF800	0,0579	0,0300	0,0679	0,0200	0,0599
FINNIS	0,0269	0,0100	0,0469	0,0200	0,0449
FIT1D	0,0269	0,0100	0,0469	0,0040	0,0499
FIT1P	0,0419	0,0300	0,0699	0,0400	0,0679
FIT2D	0,5759	0,6100	0,9478	0,6200	0,9568
FORPLAN	0,0209	0,0300	0,0429	0,0300	0,0409
GROW15	0,0419	0,0700	0,0860	0,0500	0,0849
GROW22	0,0559	0,0900	0,1219	0,0800	0,1199
GROW7	0,0219	0,0300	0,0519	0,0300	0,0429
ISRAEL	0,0509	0,0100	0,0359	0,0100	0,0409
KB2	0,0300	0,0100	0,0149	0,0100	0,0179
LOFTI	0,0089	0,0000	0,0150	0,0100	0,0139
MAROS	0,0539	0,0300	0,0589	0,0300	0,0589
MODSZKI	0,0489	0,0200	0,0519	0,0300	0,0539
NESM	0,1089	0,0800	0,1449	0,0800	0,1499
PEROLD	0,1089	0,0300	0,1509	0,0300	0,1519
PILOT 4	0,1399	0,0300	0,1669	0,0300	0,1669
RECIPE	0,0019	0,0000	0,0119	0,0100	0,0169
SC105	0,0030	0,0100	0,0119	0,0100	0,0149
SC205	0,0059	0,0100	0,0229	0,0200	0,0269
SC50A	0,0010	0,0000	0,0069	0,0000	0,0069
SC50B	0,0010	0,0000	0,0050	0,0000	0,0059
SCAGR25	0,0239	0,0100	0,0269	0,0200	0,0289
SCAGR7	0,0049	0,0100	0,0219	0,0100	0,0209
SCFXM1	0,0259	0,0100	0,0239	0,0100	0,0239
SCFXM2	0,0399	0,0300	0,0479	0,0300	0,0479
SCFXM3	0,0529	0,0300	0,0639	0,0400	0,0649
SCORPION	0,0099	0,0100	0,0309	0,0200	0,0249
SCRS8	0,0209	0,0700	0,0719	0,0800	0,0719
SCSD1	0,0069	0,0300	0,0359	0,0400	0,0389
SCSD6	0,0189	0,0100	0,0389	0,0200	0,0419
SCSD8	0,0199	0,0300	0,0809	0,0300	0,0809
SCTAP1	0,0159	0,0100	0,0399	0,0200	0,0329
SEBA	0,2189	0,0500	0,2629	0,0800	0,2599
SHARE1B	0,0099	0,0000	0,0159	0,0100	0,0899
SHARE2B	0,0080	0,0000	0,0219	0,0100	0,0239

Tabela 6. Tempos de execução em segundos

Problema	PCXOri	S2	PcxM2	S1	PcxM1
SHELL	0,0259	0,0500	0,0599	0,0400	0,0099
SHIP04L	0,0139	0,0300	0,0569	0,0200	0,0519
SHIP04S	0,0139	0,0100	0,0449	0,0100	0,0369
SHIP08L	0,0249	0,0400	0,0349	0,0400	0,0349
SHIP08S	0,1599	0,0100	0,0139	0,0200	0,0159
SHIP12L	0,0359	0,0600	0,1149	0,1200	0,1129
SHIP12S	0,0129	0,0300	0,0238	0,0300	0,0239
STAIR	0,0349	0,0300	0,0639	0,0400	0,0699
STANDATA	0,0089	0,0100	0,0159	0,0100	0,0189
STANDGUB	0,0998	0,0100	0,0179	0,0100	0,0129
STANDMPS	0,0229	0,0100	0,0499	0,0100	0,0499
STOCFOR1	0,0049	0,0000	0,0129	0,0000	0,0139
TUFF	0,0329	0,0300	0,0529	0,0200	0,0399
WOOD1P	0,1989	0,2400	0,2569	0,2400	0,2729
WOODW	0,1849	0,1000	0,3009	0,1200	0,2995

De 70 problemas, 31 tiveram melhoria no tempo de execução do algoritmo simples, 31 mantiveram o tempo de execução e 8 apresentaram um tempo de de execução melhor na implementação antiga. Comparando o tempo de execução do PCx modificado com o algoritmo simples, a implementação atual conseguiu um desempenho melhor em 33 problemas, manteve o desempenho em 11, e apresentou um aumento no tempo de execução em 26 problemas.

Tanto a melhora do tempo de execução, como o problema em alguns testes vem da alteração do cálculo da direção de busca. Essa busca foi alterada para que ela fosse realizada de forma mais eficiente, contudo ela não conseguiu resolver todos os problemas.

6. Conclusão

O estudo realizado foi sobre o algoritmo de ajustamento ótimo para p coordenadas para programação linear. Para isso foi feito um estudo sobre pogramação linear, métodos de pontos interiores e do Algoritmo de Von Neumman, o qual o algoritmo se baseia.

Em cada iteração do algoritmo de ajustamento ótimo para p coordenadas é resolvido um subproblema verificando as condições de KKT. A proposta é resolver as equações de KKT do subproblema por métodos de pontos interiores, o que é vantajoso.

A grande vantagem de usar métodos de pontos interiores é que em cada iteração do método de pontos interiores surge sistemas lineares. Quando métodos diretos são utilizados para resolver o sistema linear é feita uma fatoração de uma matriz e em seguida são resolvidos dois sistemas lineares mais simples, os quais estão associados a matrizes triangulares. O processo de fatoração de uma matriz de ordem n apresenta complexidade $O(n^3)$ e a solução de sistemas triangulares pode ser realizada com complexidade $O(n^2)$. Considerando que temos um problema de grande porte, e que os sistemas lineares a serem resolvidos possuem ordem $p+1$ o custo é proporcionalmente pequeno.

A partir de uma implementação do algoritmo, foi feita uma melhoria na implementação. Ambas implementações do algoritmo apresententaram um desempenho

muito bom, a implementação melhorada apresentou um desempenho, em comparação de tempo, superior a implementação anterior. Contudo a implementação melhorada apresentou problemas em alguns testes.

Referências

[1] Silva, Jair *Uma família de algoritmos de programação linear baseada no algoritmo de Von Neumann,* Matemática Aplicada, Unicamp, Campinas, SP, 2009.

[2] Czyzyk, Joseph & Mehrotra, Sanjay & Wagner, Michael & Wright, Stephen J, *PCx: an interior-point code for linear programming*, Optimization Methods and Software, 1999, volume 11, number 1, pages 397 - 430.

[3] *NETLIB collection LP test sets. Netlib lp repository.* Online at http://www.netlib.org/lp/data.